## Contents

*Early Adopter Software* ................................................................. v

**Chapter 1 • SAS Functions and CALL Routines** ................................. 1
- Definitions of Functions and CALL Routines ................................. 2
- Syntax ....................................................................................... 3
- Using Functions and CALL Routines ............................................ 4
- Function Compatibility with SBCS, DBCS, and MBCS Character Sets .... 10
- Using Random-Number Functions and CALL Routines .................... 11
- Using SYSRANDOM and SYSRANEND Macro Variables to
  Produce Random Number Streams .............................................. 28
- Date and Time Intervals ............................................................ 31
- Pattern Matching Using Perl Regular Expressions (PRX) .................. 42
- Using Perl Regular Expressions in the DATA Step ......................... 43
- Writing Perl Debug Output to the SAS Log .................................... 52
- Perl Artistic License Compliance ............................................... 53
- SAS Functions for Web Applications .......................................... 54
- Functions in SAS and CAS ....................................................... 54

**Chapter 2 • Dictionary of Functions and Call Routines for SAS and the CAS Server** ............... 57
- SAS Functions and CALL Routines for CAS and SAS by Category .... 63
- Dictionary .................................................................................. 79

**Chapter 3 • Dictionary of Functions and CALL Routines for SAS Only** ......................... 697
- SAS Functions and CALL Routines Documented in Other SAS Publications ... 701
- SAS Functions and CALL Routines by Category ............................. 701
- Dictionary .................................................................................. 714

**Chapter 4 • References** ............................................................... 1111
- References .................................................................................. 1111

**Appendix 1 • Tables of Perl Regular Expression (PRX) Metacharacters** ....................... 1113

*Recommended Reading* ................................................................. 1123
*Index* ....................................................................................... 1125
iv Contents
Early Adopter Software

THIS DOCUMENTATION FOR AN EARLY ADOPTER PRODUCT IS A PRELIMINARY DRAFT AND IS PROVIDED BY SAS INSTITUTE INC. ("SAS") ON AN "AS IS" BASIS WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND/OR FITNESS FOR A PARTICULAR PURPOSE. SAS does not warrant that this documentation is complete, accurate, or similar to that which may be released to the general public, or that any such documentation will be released. The company shall not be liable whatsoever for any damages arising out of the use of this documentation, including any direct, indirect, or consequential damages. SAS reserves the right to alter or abandon use of this documentation at any time.
vi  Early Adopter Software
Chapter 1
SAS Functions and CALL Routines

Definitions of Functions and CALL Routines
- Definition of Functions ........................................... 2
- Definition of CALL Routines ................................. 2
Syntax
- Syntax of Functions ........................................... 3
- Syntax of CALL Routines ...................................... 4
Using Functions and CALL Routines
- Restrictions Affecting Function Arguments .............. 4
- Using the OF Operator with Temporary Arrays ........ 5
- Characteristics of Target Variables ......................... 5
- Notes about Descriptive Statistic Functions ............ 6
- Notes about Financial Functions ............................. 6
- Using DATA Step Functions within Macro Functions .. 8
- Using CALL Step Functions within Macro Functions .. 9
- Using Functions to Manipulate Files ..................... 9
Function Compatibility with SBCS, DBCS, and MBCS Character Sets
- Overview ....................................................... 10
- I18N Level 0 .................................................. 10
- I18N Level 1 .................................................. 10
- I18N Level 2 .................................................. 10
Using Random-Number Functions and CALL Routines
- Types of Random-Number Functions ..................... 11
- Seed Values .................................................. 11
- Understanding How Functions Generate a Random-Number Stream .......................... 11
- Comparison of Seed Values in Random-Number Functions and CALL Routines ........ 15
- Generating Multiple Streams from Multiple Seeds in
  Random-Number CALL Routines .......................... 15
- Generating Multiple Variables from One Seed in Random-Number Functions ........... 22
- Using the RAND Function as an Alternative ............ 25
- Effectively Using the Random-Number CALL Routines .......................... 26
- Comparison of Changing the Seed in a CALL Routine and in a Function ........... 27
Using SYSRANDOM and SYSRANEND Macro Variables to
Produce Random Number Streams
- Overview of the SYSRANDOM and SYSRANEND Macro Variables ............ 28
- The SYSRANDOM Macro Variable ......................... 28
- The SYSRANEND Macro Variable ......................... 29
- Example: Reproducing Results ......................... 29
- Example: Creating a Reproducible Random Number Stream .................. 30
Definitions of Functions and CALL Routines

Definition of Functions

A SAS function is a component of the SAS programming language that can accept arguments, perform a computation or other operation, and return a value. Functions can return either numeric or character results. The value that is returned can be used in an assignment statement or elsewhere in expressions. Many functions are included with SAS, and you can write your own functions as well.

In SAS software, you can use SAS functions in DATA step programming statements, in a WHERE expression, in macro language statements, in PROC REPORT, and in Structured Query Language (SQL).

Some statistical procedures also use SAS functions. In addition, some other SAS software products offer functions that you can use in the DATA step. For more information about these functions, see the documentation that pertains to the specific SAS software product.

Definition of CALL Routines

A CALL routine alters variable values or performs other system functions. CALL routines are similar to functions, but differ from functions in that you cannot use them in assignment statements or expressions.
All SAS CALL routines are invoked with CALL statements. That is, the name of the routine must appear after the keyword CALL in the CALL statement.

Syntax

Syntax of Functions

The syntax of a function has one of the following forms:

- `function-name (argument-1, …, argument-n)`
- `function-name (OF variable-list)`
- `function-name (<argument | OF variable-list | OF array-name[*]> <…, <argument | OF variable-list | OF array-name[*]>>)`

`function-name` names the function.

`argument` can be a variable name, constant, or any SAS expression, including another function. The number and type of arguments that SAS allows are described with individual functions. Multiple arguments are separated by a comma.

Note: If the value of an argument is invalid (for example, missing or outside the prescribed range), SAS writes a note to the log indicating that the argument is invalid, sets `_ERROR_` to 1, and sets the result to a missing value. Here are examples:

- `x=max(cash, credit);`
- `x=sqrt(1500);`
- `NewCity=left(upcase(City));`
- `x=min(YearTemperature-July, YearTemperature-Dec);`
- `s=repeat('-+.16);`
- `x=min((enroll-drop),(enroll-fail));`
- `dollars=int(cash);`
- `if sum(cash, credit)>1000 then put 'Goal reached';`

`variable-list` can be any form of a SAS variable list, including individual variable names. If more than one variable list appears, separate them with a space or with a comma and another OF.

- `a=sum(of x y z);`
- `z=sum(of y1-y10);`
- `z=msplint(x0, 5, of x1-x5, of y1-y5,-2,2);`

Example

The following two examples are equivalent.

- `a=sum(of x1-x10 y1-y10 z1-z10);`
- `a=sum(of x1-x10, of y1-y10, of z1-z10);`
array-name{*}

names a currently defined array. Specifying an array with an asterisk as a subscript causes SAS to treat each element of the array as a separate argument.

The OF operator has been extended to accept temporary arrays. You can use temporary arrays in OF lists for most SAS functions just as you can use regular variable arrays, but there are some restrictions.

See For a list of these restrictions, see “Using the OF Operator with Temporary Arrays” on page 5.

Syntax of CALL Routines

The syntax of a CALL routine has one of the following forms:

CALL routine-name (argument-1, ..., argument-n);
CALL routine-name (OF variable-list);
CALL routine-name (argument-1 | OF variable-list-1, ..., argument-n | OF variable-list-n);

routine-name

names a SAS CALL routine.

argument

can be a variable name, a constant, any SAS expression, an external module name, an array reference, or a function. Multiple arguments are separated by a comma. The number and type of arguments that are allowed are described with individual CALL routines in the dictionary section. Here are examples:

• call prxsubstr(prx,string,position);
• call prxchange('/old/new',1+k,trim(string),result,length);
• call set(dsid);
• call ranbin(Seed_1,n,p,X1);
• call label(abc{j},lab);
• call cats(result,'abc',123);

variable-list

can be any form of a SAS variable list, including variable names. If more than one variable list appears, separate them with a space or with a comma and another OF.

• call cats(inventory, of y1-y15, of z1-z15);
• call catt(of item17-item23 pack17-pack23);

Using Functions and CALL Routines

Restrictions Affecting Function Arguments

If the value of an argument is invalid, SAS writes a note or error message to the log and sets the result to a missing value. Here are some common restrictions for function arguments:
• Some functions require that their arguments be restricted within a certain range. For example, the argument of the LOG function must be greater than 0.
• When a numeric argument has a missing value, many functions write a note to the SAS log and return a missing value. Exceptions include some of the descriptive statistics functions and financial functions.
• For some functions, the allowed range of the arguments is platform-dependent, such as with the EXP function.

**Using the OF Operator with Temporary Arrays**

You can use the OF operator with temporary arrays. This capability enables the passing of temporary arrays to most functions whose arguments contain a varying number of parameters. You can use temporary arrays in OF lists in some functions, just as you can use temporary arrays in OF lists in regular variable arrays.

There are some limitations in using temporary arrays. These limitations are listed after the example.

The following example shows how you can use temporary arrays:

```sas
data _null_;  array y[10] _temporary_ (1,2,3,4,5,6,7,8,9,10);  x = sum(of y{*});  put x=;  run;

data _null_;  array y[10] $10 _temporary_ ('1','2','3','4','5','6','7','8','9','10');  x = max(of y{*});  put x=;  run;
```

**Log 1.1  Log Output for the Example of Using Temporary Arrays**

```
x=55
x=10
```

The following limitations affect temporary array OF lists:

• cannot be used as array indices
• can be used in functions where the number of parameters matches the number of elements in the OF list, as with regular variable arrays
• can be used in functions that take a varying number of parameters
• cannot be used with the DIF, LAG, SUBSTR, LENGTH, TRIM, or MISSING functions, nor with any of the variable information functions such as VLENGTH

**Characteristics of Target Variables**

Some character functions produce resulting variables, or target variables, with a default length of 200 bytes. Numeric target variables have a default length of 8 bytes. Character functions to which the default target variable lengths do not apply are shown in the
following table. These functions obtain the length of the return argument based on the length of the first argument.

**Table 1.1 Functions Whose Return Argument Is Based on the Length of the First Argument**

<table>
<thead>
<tr>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPBL</td>
</tr>
<tr>
<td>COMPRESS</td>
</tr>
<tr>
<td>DEQUOTE</td>
</tr>
<tr>
<td>INPUTC</td>
</tr>
<tr>
<td>LEFT</td>
</tr>
<tr>
<td>LOWCASE</td>
</tr>
<tr>
<td>PUTC</td>
</tr>
<tr>
<td>REVERSE</td>
</tr>
<tr>
<td>RIGHT</td>
</tr>
<tr>
<td>STRIP</td>
</tr>
<tr>
<td>SUBSTR</td>
</tr>
<tr>
<td>SUBSTRN</td>
</tr>
<tr>
<td>TRANSLATE</td>
</tr>
<tr>
<td>TRIM</td>
</tr>
<tr>
<td>TRIMN</td>
</tr>
<tr>
<td>UPCASE</td>
</tr>
</tbody>
</table>

The following list of functions shows the length of the target variable if the target variable has not been assigned a length:

BYTE
- target variable is assigned a default length of 1.

INPUT
- length of the target variable is determined by the width of the informat.

PUT
- length of the target variable is determined by the width of the format.

VTYPE
- target variable is assigned a default length of 1.

VTYPEX
- target variable is assigned a default length of 1.

**Notes about Descriptive Statistic Functions**

SAS provides functions that return descriptive statistics. Many of these functions correspond to the statistics produced by the MEANS and UNIVARIATE procedures. The computing method for each statistic is discussed in the elementary statistics procedures section of the *SAS Viya Utility Procedures Guide*. SAS calculates descriptive statistics for the nonmissing values of the arguments.

**Notes about Financial Functions**

**Types of Financial Functions**

SAS provides a group of functions that perform financial calculations. The functions are grouped into the following types:
<table>
<thead>
<tr>
<th>Function Type</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cash Flow</td>
<td>CONVX, CONVXP</td>
<td>calculates convexities for cash flows</td>
</tr>
<tr>
<td></td>
<td>DUR, DURP</td>
<td>calculates modifies duration for cash flows.</td>
</tr>
<tr>
<td></td>
<td>PVP, YIELDP</td>
<td>calculates present value and yield-to-maturity for a periodic cash flow</td>
</tr>
<tr>
<td>Parameter calculations</td>
<td>COMPOUND</td>
<td>calculates compound interest parameters</td>
</tr>
<tr>
<td></td>
<td>MORT</td>
<td>calculates amortization parameters</td>
</tr>
<tr>
<td>Internal rate of return</td>
<td>INTRRR, IRR</td>
<td>calculates the internal rate of return</td>
</tr>
<tr>
<td>Net present and future value</td>
<td>NETPV, NPV</td>
<td>calculates net present and future values</td>
</tr>
<tr>
<td></td>
<td>SAVING</td>
<td>calculates the future value of periodic saving</td>
</tr>
<tr>
<td>Depreciation</td>
<td>DACCxx</td>
<td>calculates the accumulated depreciation up to the specified period</td>
</tr>
<tr>
<td></td>
<td>DEPxxx</td>
<td>calculates depreciation for a single period</td>
</tr>
<tr>
<td>Pricing</td>
<td>BLKSHCLPRC, BLKSHPTPRC</td>
<td>calculated call prices and put prices for European options on stocks, based on the Black-Scholes model</td>
</tr>
<tr>
<td></td>
<td>BLACKPLPRC, BLACKPTPRC</td>
<td>calculates call prices and put prices for European options on futures, based on the Black model</td>
</tr>
<tr>
<td></td>
<td>GARKHCLPRC, GARKHPTPRC</td>
<td>calculates call prices and put prices for European options on stocks, based on the Garman-Kohlhagen model</td>
</tr>
<tr>
<td></td>
<td>MARGRCLPRC, MARGRPTPRC</td>
<td>calculates call options and put prices for European options on stocks, based on the Margrabe model</td>
</tr>
</tbody>
</table>
Using Pricing Functions
A pricing model is used to calculate a theoretical market value (price) for a financial instrument. This value is referred to as a mark-to-market (MtM) value. Typically, a pricing function has the following form:

\[
price = function(rf1, rf2, rf3, \ldots)
\]

In the pricing function, \(rf1\), \(rf2\), and \(rf3\) are risk factors such as interest rates or foreign exchange rates. The specific values of the risk factors that are used to calculate the MtM value are the base case values. The set of base case values is known as the base case market state.

After determining the MtM value, you can perform the following tasks with the base case values of the risk factors (\(rf1\), \(rf2\), and \(rf3\)):

- Set the base case values to specific values to perform scenario analyses.
- Set the base case values to a range of values to perform profit/loss curve analyses and profit/loss surface analyses.
- Automatically set the base case values to different values to calculate sensitivities - that is, to calculate the delta and gamma values of the risk factors.
- Perturb the base case values to create many possible market states so that many possible future prices can be calculated, and simulation analyses can be performed. For Monte Carlo simulation, the values of the risk factors are generated using mathematical models and the copula methodology.

A list of pricing functions and their descriptions are included in “Types of Financial Functions” on page 6.

Using DATA Step Functions within Macro Functions
The macro functions %SYSFUNC and %QSYSFUNC can call most DATA step functions to generate text in the macro facility. %SYSFUNC and %QSYSFUNC have one difference: %QSYSFUNC masks special characters and mnemonics and %SYSFUNC does not. For more information about these functions, see %QSYSFUNC and %SYSFUNC in SAS Viya Macro Language: Reference.

%SYSFUNC arguments are a single DATA step function and an optional format, as shown in the following examples:

\[
\begin{align*}
\%sysfunc(date(),worddate.) \\
\%sysfunc(attrn(&dsid,NOBS))
\end{align*}
\]

You cannot nest DATA step functions within %SYSFUNC. However, you can nest %SYSFUNC functions that call DATA step functions. For example:

\[
\begin{align*}
\%sysfunc(compress(%sysfunc(getoption(sasautos)), \\
\quad %str(%)%(%'))) \\
\end{align*}
\]

All arguments in DATA step functions within %SYSFUNC must be separated by commas. You cannot use argument lists that are preceded by the word OF.

Because %SYSFUNC is a macro function, you do not need to enclose character values in quotation marks as you do in DATA step functions. For example, the arguments to the OPEN function are enclosed in quotation marks when you use the function alone, but the arguments do not require quotation marks when used within %SYSFUNC.

\[
\begin{align*}
dsid=\text{open}("\text{sasuser.houses","i")}; \\
dsid=\text{open}("\text{&mydata","&mode")}; \\
\%let \text{dsid}=\%sysfunc(open(\text{sasuser.houses},i));
\end{align*}
\]
Using CALL Routines and the %SYSCALL Macro Statement

When the %SYSCALL macro statement invokes a CALL routine, the value of each macro variable argument is retrieved and passed unresolved to the CALL routine. Upon completion of the CALL routine, the value for each argument is written back to the respective macro variable. If %SYSCALL encounters an error condition, the execution of the CALL routine terminates without updating the macro variable values and an error message is written to the log.

When %SYSCALL invokes a CALL routine, the argument value is passed unresolved to the CALL routine. The unresolved argument value might have been quoted using macro quoting functions and might contain delta characters. The argument value in its quoted form can cause unpredictable results when character values are compared. Some CALL routines unquote their arguments when they are called by %SYSCALL and return the unquoted values. Other CALL routines do not need to unquote their arguments. Here is a list of CALL routines that unquote their arguments when called by %SYSCALL:

- “CALL COMPCOST Routine” on page 742
- “LEXCOMB Function” on page 977
- “LEXPERK Function” on page 981
- “CALL LEXPERM Routine” on page 770
- “CALL PRXCHANGE Routine” on page 782
- “CALL PRXNEXT Routine” on page 787
- “CALL PRXSUBSTR Routine” on page 792
- “CALL SCAN Routine” on page 128
- “CALL SORTC Routine” on page 836
- “CALL STDIZE Routine” on page 838

In comparison, %SYSCALL invokes a CALL routine and returns an unresolved value, which contains delta characters. %SYSFUNC invokes a function and returns a resolved value, which does not contain delta characters. For more information, see “Macro Quoting” in SAS Viya Macro Language: Reference, “%SYSCALL Statement” in SAS Viya Macro Language: Reference, and “%SYSFUNC and %QSYSFUNC Functions” in SAS Viya Macro Language: Reference.

Using Functions to Manipulate Files

SAS manipulates files in different ways, depending on whether you use functions or statements. If you use functions such as FOPEN, FGET, and FCLOSE, you have more opportunity to examine and manipulate your data than when you use statements such as INFILE, INPUT, and PUT.

When you use external files, the FOPEN function allocates a buffer called the File Data Buffer (FDB) and opens the external file for reading or updating. The FREAD function reads a record from the external file and copies the data into the FDB. The FGET function then moves the data to the DATA step variables. The function returns a value that you can check with statements or other functions in the DATA step to determine how to further process your data. After the records are processed, the FWRITE function writes the contents of the FDB to the external file, and the FCLOSE function closes the file.
When you use SAS data sets, the OPEN function opens the data set. The FETCH and FETCHOBS functions read observations from an open SAS data set into the Data Set Data Vector (DDV). The GETVARC and GETVARN functions then move the data to DATA step variables. The functions return a value that you can check with statements or other functions in the DATA step to determine how you want to further process your data. After the data is processed, the CLOSE function closes the data set.

For a complete listing of functions and CALL routines, see “SAS Functions and CALL Routines by Category” on page 701. For complete descriptions and examples, see the dictionary section of this book.

Function Compatibility with SBCS, DBCS, and MBCS Character Sets

Overview

SAS string functions and CALL routines can be categorized by level numbers that are used in internationalization. I18N is the abbreviation for internationalization, and indicates string functions that can be adapted to different languages and locales without program changes.

I18N recognizes the following three levels that identify the character sets that you can use:

- “I18N Level 0” on page 10
- “I18N Level 1” on page 10
- “I18N Level 2” on page 10

For more information about function compatibility, see “Internationalization Compatibility for SAS String Functions” in SAS Viya National Language Support (NLS): Reference Guide.

I18N Level 0

I18N Level 0 functions are designed for use with Single-Byte Character Sets (SBCS) only.

I18N Level 1

I18N Level 1 functions should be avoided, if possible, if you are using a non-English language. The I18N Level 1 functions might not work correctly with Double-Byte Character Set (DBCS) or Multi-Byte Character Set (MBCS) encodings under certain circumstances.

I18N Level 2

I18N Level 2 functions are designed for use with SBCS, DBCS, and MBCS (UTF8).
Types of Random-Number Functions

Two types of random-number functions are available in SAS. The newest random-number function is the RAND function. It uses the Mersenne-Twister pseudo-random number generator (RNG) that was developed by Matsumoto and Nishimura (1998). This RNG has a very long period of $2^{19937} - 1$, and has very good statistical properties. (A period is the number of occurrences before the pseudo-random number sequence repeats.)

The RAND function is started with a single seed. However, the state of the process cannot be captured by a single seed, which means that you cannot stop and restart the generator from its stopping point. Use the STREAMINIT function to produce a sequence of values that begins at the beginning of a stream. For more information, see the Details section of the “RAND Function” on page 555.

The older random-number generators include the UNIFORM, NORMAL, RANUNI, RANNOR, and other functions that begin with RAN. These functions have a period of only $2^{31} - 2$ or less. The pseudo-random number stream is started with a single seed, and the state of the process can be captured in a new seed. This means that you can stop and restart the generator from its stopping point by providing the proper seed to the corresponding CALL routines. You can use the random-number functions to produce a sequence of values that begins in the middle of a stream.

Seed Values

Random-number functions and CALL routines generate streams of pseudo-random numbers from an initial starting point, called a seed, that either the user or the computer clock supplies. A seed must be a nonnegative integer with a value less than $2^{31} - 1$ (or 2,147,483,647). If you use a positive seed, you can always replicate the stream of random numbers by using the same DATA step. If you use zero as the seed, the computer clock initializes the stream, and the stream of random numbers cannot be replicated.

Understanding How Functions Generate a Random-Number Stream

Using the DATA Step to Generate a Single Stream of Random Numbers

The DATA steps in this section illustrate several properties of the random-number functions. Each of the DATA steps that call a function generates a single stream of pseudo-random numbers based on a seed value of 7, because that is the first seed for the first call for every step. Some of the DATA steps change the seed value in various ways. Some of the steps have single function calls and others have multiple function calls. None of these DATA steps change the seed. The only seed that is relevant to the function calls is the seed that was used with the first execution of the first random-number function. There is no way to create separate streams with functions (CALL routines are used for this purpose), and the only way that you can restart the function random-number stream is to start a new DATA step.
The following example executes multiple DATA steps:

```sas
/* This DATA step produces a single stream of random numbers */
/* based on a seed value of 7. */
data a;
  a = ranuni (7); output;
  a = ranuni (7); output;
  a = ranuni (7); output;
  a = ranuni (7); output;
  a = ranuni (7); output;
  a = ranuni (7); output;
  a = ranuni (7); output;
  a = ranuni (7); output;
  a = ranuni (7); output;
  a = ranuni (7); output;
  a = ranuni (7); output;
  a = ranuni (7); output;
run;

/* This DATA step uses a DO statement to produce a single */
/* stream of random numbers based on a seed value of 7. */
data b (drop = i);
  do i = 7 to 18;
    b = ranuni (i);
    output;
  end;
run;

/* This DATA step uses a DO statement to produce a single */
/* stream of random numbers based on a seed value of 7. */
data c (drop = i);
  do i = 1 to 12;
    c = ranuni (7);
    output;
  end;
run;

/* This DATA step calls the RANUNI and the RANROR functions */
/* and produces a single stream of random numbers based on */
/* a seed value of 7. */
data d;
  d = ranuni (7); f = ' '; output;
  d = ranuni (8); f = ' '; output;
  d = rannor (9); f = 'n'; output;
  d = .;       f = ' '; output;
  d = ranuni (0); f = ' '; output;
  d = ranuni (1); f = ' '; output;
  d = rannor (2); f = 'n'; output;
  d = .;       f = ' '; output;
  d = ranuni (3); f = ' '; output;
  d = ranuni (4); f = ' '; output;
  d = rannor (5); f = 'n'; output;
  d = .;       f = ' '; output;
run;
```
/* This DATA step calls the RANNOR function and produces a single stream of random numbers based on a seed value of 7. */
data e (drop = i);
do i = 1 to 6;
   e = rannor (7); output; 
   e = .;          output; 
end;
run;

/* This DATA step merges the output data sets that were created from the previous five DATA steps. */
data five;
   merge a b c d e;
run;

/* This procedure writes the output from the merged data sets. */
proc print label data=five;
   options missing = ' '; 
   label f = '00'x; 
   title 'Single Random Number Streams';
run;

The following output shows the program results.

**Figure 1.1  Results from Generating a Single Random-Number Stream**

<table>
<thead>
<tr>
<th>Obs</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.29474</td>
<td>0.29474</td>
<td>0.29474</td>
<td>0.29474</td>
<td>0.39464</td>
</tr>
<tr>
<td>2</td>
<td>0.79062</td>
<td>0.79062</td>
<td>0.79062</td>
<td>0.79062</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.79877</td>
<td>0.79877</td>
<td>0.79877</td>
<td>0.26928</td>
<td>0.26928</td>
</tr>
<tr>
<td>4</td>
<td>0.81579</td>
<td>0.81579</td>
<td>0.81579</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.45122</td>
<td>0.45122</td>
<td>0.45122</td>
<td>0.45122</td>
<td>0.27475</td>
</tr>
<tr>
<td>6</td>
<td>0.78494</td>
<td>0.78494</td>
<td>0.78494</td>
<td>0.78494</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0.80085</td>
<td>0.80085</td>
<td>0.80085</td>
<td>-0.11729</td>
<td>-0.11729</td>
</tr>
<tr>
<td>8</td>
<td>0.72184</td>
<td>0.72184</td>
<td>0.72184</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0.34856</td>
<td>0.34856</td>
<td>0.34856</td>
<td>0.34856</td>
<td>-1.41879</td>
</tr>
<tr>
<td>10</td>
<td>0.46597</td>
<td>0.46597</td>
<td>0.46597</td>
<td>0.46597</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>0.73523</td>
<td>0.73523</td>
<td>0.73523</td>
<td>-0.39033</td>
<td>-0.39033</td>
</tr>
<tr>
<td>12</td>
<td>0.66709</td>
<td>0.66709</td>
<td>0.66709</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The pseudo-random number streams in output data sets A, B, and C are identical. The stream in output data set D mixes calls to the RANUNI and RANNOR functions. In
observations 1, 2, 5, 6, 9, and 10, the values that are returned by RANUNI exactly match the values in the previous streams. Observations 3, 7, and 11, which are flagged by “n”, contain the values that are returned by the RANNOR function. The mix of the function calls does not affect the generation of the pseudo-random number stream. All of the results are based on a single stream of uniformly distributed values. Some are transformed and returned from other functions such as RANNOR. The results of the RANNOR function are produced from two internal calls to RANUNI. The DATA step that creates output data set D executes the following steps three times to create 12 observations:

- call to RANUNI
- call to RANUNI
- call to RANNOR (which internally calls RANUNI twice)
- skipped line to compensate for the second internal call to RANUNI

In the DATA step that creates data set E, RANNOR is called six times, each time skipping a line to compensate for the fact that two internal calls to RANUNI are made for each call to RANNOR. Notice that the three values that are returned from RANNOR in the DATA step that creates data set D match the corresponding values in data set E.

### Using the %SYSFUNC Macro to Generate a Single Stream of Random Numbers

When the RANUNI function is called through the macro language by using %SYSFUNC, one pseudo-random number stream is created. You cannot change the seed value unless you close SAS and start a new SAS session. The %SYSFUNC macro produces the same pseudo-random number stream as the DATA steps that generated the data sets A, B, and C for the first macro invocation only. Any subsequent macro calls produce a continuation of the single stream.

```sas
%macro ran;
%do i = 1 %to 12;
%let x = %sysfunc (ranuni (7));
%put &x;
%end;
%mend;

%ran;
```

SAS writes the following output to the log:
Comparison of Seed Values in Random-Number Functions and CALL Routines

Each random-number function and CALL routine generates pseudo-random numbers from a specific statistical distribution. Each random-number function requires a seed value expressed as an integer constant or a variable that contains the integer constant. Each CALL routine calls a variable that contains the seed value. Also, every CALL routine requires a variable that contains the generated pseudo-random numbers.

The seed variable must be initialized before the first execution of the function or CALL routine. After each execution of a function, the current seed is updated internally, but the value of the seed argument remains unchanged. However, after each iteration of the CALL routine the seed variable contains the current seed in the stream that generates the next pseudo-random number. With a function, it is not possible to control the seed values, and, therefore, the pseudo-random numbers after the initialization.

SAS provides a CALL routine that has the same name as each random-number function. The exception is the NORMAL and UNIFORM functions, which are equivalent to the RANNOR and RANUNI functions. Using CALL routines gives you greater control over the seed values.

Generating Multiple Streams from Multiple Seeds in Random-Number CALL Routines

Overview of Random-Number CALL Routines and Streams
You can use the random-number CALL routines to generate multiple streams of pseudo-random numbers within a single DATA step. If you supply a different seed value to initialize each of the seed variables, the streams of the generated pseudo-random numbers are computationally independent, but they might not be statistically independent unless you select the seed values carefully.
Note: Although you can create multiple streams with multiple seeds, this practice is not recommended. It is always safer to create a single stream. With multiple streams, as the streams become longer, the chances of the stream overlapping increase.

The following two examples deliberately select seeds to illustrate worst-case scenarios. The examples show how to produce multiple streams by using multiple seeds. Although this practice is not recommended, you can use the random-number CALL routines with multiple seeds.

Example 1: Using Multiple Seeds to Generate Multiple Streams
This example shows that you can use multiple seeds to generate multiple streams of pseudo-randomly distributed values by using the random-number CALL routines. The first DATA step creates a data set with three variables that are normally distributed. The second DATA step creates variables that are uniformly distributed. The SGSCATTER procedure (see the *SAS ODS Graphics: Procedures Guide*) is used to show the relationship between each pair of variables for each of the two distributions.

```sas
data normal;
  seed1 = 11111;
  seed2 = 22222;
  seed3 = 33333;
  do i = 1 to 10000;
    call rannor(seed1, x1);
    call rannor(seed2, x2);
    call rannor(seed3, x3);
    output;
  end;
run;

data uniform;
  seed1 = 11111;
  seed2 = 22222;
  seed3 = 33333;
  do i = 1 to 10000;
    call ranuni(seed1, x1);
    call ranuni(seed2, x2);
    call ranuni(seed3, x3);
    output;
  end;
run;

proc sgscatter data = normal;
  title 'Nonindependent Random Normal Variables';
  plot x1*x2 x1*x3 x3*x2 / markerattrs = (size = 1);
run;

proc sgscatter data = uniform;
  title 'Nonindependent Random Uniform Variables';
  plot x1*x2 x1*x3 x3*x2 / markerattrs = (size = 1);
run;
```
Figure 1.2  Multiple Streams from Multiple Seeds: Nonindependent Random Normal Variables
The first plot (Figure 1.2 on page 17) shows that normal variables appear to be linearly uncorrelated, but they are obviously not independent. The second plot (Figure 1.3 on page 18) shows that uniform variables are clearly related. With this class of random-number generators, there is never any guarantee that the streams are independent.

**Example 2: Using Different Seeds with the CALL RANUNI Routine**

The following example uses three different seeds and the CALL RANUNI routine to produce multiple streams.

```sas
data uniform(drop=i);
  seed1 = 255793849;
  seed2 = 1408147117;
  seed3 = 961782675;
  do i=1 to 10000;
    x1 = ranuni(seed1);
    x2 = ranuni(seed2);
    x3 = ranuni(seed3);
  end;
run;
```

call ranuni(seed1, x1);
call ranuni(seed2, x2);
call ranuni(seed3, x3);
i2 = lag(x2);
i3 = lag2(x3);
output;
end;
label i2='Lag(x2)' i3='Lag2(x3)';
run;

title 'Random Uniform Variables with Overlapping Streams';
proc sgscatter data=uniform;
   plot x1*x2 x1*x3 x3*x2 / markerattrs = (size = 1);
run;

proc sgscatter data=uniform;
   plot i2*x1 i3*x1 / markerattrs = (size = 1);
run;

proc print noobs data=uniform(obs=10);
run;
Figure 1.4 Using Different Seeds with CALL RANUNI: Random Uniform Variables with Overlapping Streams, Plot 1
Figure 1.5  Using Different Seeds with CALL RANUNI: Random Uniform Variables with Overlapping Streams, Plot 2
Random Uniform Variables with Overlapping Streams

<table>
<thead>
<tr>
<th>seed1</th>
<th>seed2</th>
<th>seed3</th>
<th>x1</th>
<th>x2</th>
<th>x3</th>
<th>i2</th>
<th>i3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1408147117</td>
<td>961782675</td>
<td>383001085</td>
<td>0.65572</td>
<td>0.44786</td>
<td>0.17835</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>961782675</td>
<td>383001085</td>
<td>1989090982</td>
<td>0.44786</td>
<td>0.17835</td>
<td>0.92624</td>
<td>0.44786</td>
<td>.</td>
</tr>
<tr>
<td>383001085</td>
<td>1989090982</td>
<td>1375749095</td>
<td>0.17835</td>
<td>0.92624</td>
<td>0.64063</td>
<td>0.17835</td>
<td>0.17835</td>
</tr>
<tr>
<td>1989090982</td>
<td>1375749095</td>
<td>89319994</td>
<td>0.92624</td>
<td>0.64063</td>
<td>0.04159</td>
<td>0.92624</td>
<td>0.92624</td>
</tr>
<tr>
<td>1375749095</td>
<td>89319994</td>
<td>1345897251</td>
<td>0.64063</td>
<td>0.04159</td>
<td>0.62673</td>
<td>0.64063</td>
<td>0.64063</td>
</tr>
<tr>
<td>89319994</td>
<td>1345897251</td>
<td>561406336</td>
<td>0.04159</td>
<td>0.62673</td>
<td>0.26143</td>
<td>0.04159</td>
<td>0.04159</td>
</tr>
<tr>
<td>1345897251</td>
<td>561406336</td>
<td>1333490358</td>
<td>0.62673</td>
<td>0.26143</td>
<td>0.62095</td>
<td>0.62673</td>
<td>0.62673</td>
</tr>
<tr>
<td>561406336</td>
<td>1333490358</td>
<td>963442111</td>
<td>0.26143</td>
<td>0.62095</td>
<td>0.44864</td>
<td>0.26143</td>
<td>0.26143</td>
</tr>
<tr>
<td>1333490358</td>
<td>963442111</td>
<td>1557707418</td>
<td>0.62095</td>
<td>0.44864</td>
<td>0.72536</td>
<td>0.62095</td>
<td>0.62095</td>
</tr>
<tr>
<td>963442111</td>
<td>1557707418</td>
<td>137842443</td>
<td>0.44864</td>
<td>0.72536</td>
<td>0.06419</td>
<td>0.44864</td>
<td>0.44864</td>
</tr>
</tbody>
</table>

The first plot (Figure 1.4 on page 20) shows expected results: the variables appear to be statistically independent. However, the second plot (Figure 1.5 on page 21) and the listing of the first 10 observations show that there is almost complete overlap between the two streams. The last 9999 values in x1 match the first 9999 values in x2, and the last 9998 values in x1 match the first 9998 values in x3. In other words, there is perfect agreement between the nonmissing parts of x1 and lag(x2) and also x1 and lag2(x3). Even if the streams appear to be independent at first glance as in the first plot, there might be overlap, which might be undesirable depending on how the streams are used.

In practice, if you make multiple small streams with separate and randomly selected seeds, you probably do not encounter the problems that are shown in the first two examples. Figure 1.5 on page 21 deliberately selects seeds to illustrate worst-case scenarios.

It is always safer to create a single stream. With multiple streams, as the streams get longer, the chances of the streams overlapping increase.

Generating Multiple Variables from One Seed in Random-Number Functions

Overview of Functions and Streams

If you use functions in your program, you cannot generate more than one stream of pseudo-random numbers by supplying multiple seeds within a DATA step.

The following example uses the RANUNI function to show the safest way to create multiple variables from the same stream with a single seed.
Example: Generating Random Uniform Variables with Overlapping Streams

In the following example, the RANUNI function is used to create random uniform variables with overlapping streams. The example shows the safest way to create multiple variables by using the RANUNI function. All variables are created from the same stream with a single seed.

```sas
data uniform(drop=i);
  do i = 1 to 10000;
    x1 = ranuni(11111);
    x2 = ranuni(11111);
    x3 = ranuni(11111);
    i2 = lag(x2);
    i3 = lag2(x3);
    output;
  end;
label i2 = 'Lag(x2)' i3 = 'Lag2(x3)';
run;

title 'Random Uniform Variables with Overlapping Streams';
proc sgscatter data = uniform;
  plot x1*x2 x1*x3 x3*x2 / markerattrs = (size = 1);
run;

proc sgscatter data = uniform;
  plot i2*x1 i3*x1 / markerattrs = (size = 1);
run;
```
Figure 1.7 Random Uniform Variables with Overlapping Streams: Plot 1
In “Example: Generating Random Uniform Variables with Overlapping Streams” on page 23, it appears that the variables are independent. However, even this programming approach might not work well in general. The random-number functions and CALL routines have a period of only $2^{31} - 2$ or less (approximately 2.1 billion). When this limit is reached, the stream repeats. Modern computers performing complicated simulations can easily exhaust the entire stream in minutes.

**Using the RAND Function as an Alternative**

A better approach to generating random uniform variables is to use the RAND function, where multiple streams are not permitted. The RAND function has a period of $2^{19937} - 1$. This limit is never reached, at least with computers of the early 21st century. The number $2^{19937} - 1$ is approximately $10^{6000}$ (1 followed by 6000 zeros). In comparison, the largest value that can be represented in eight bytes on most computers that run SAS is approximately $10^{307}$.

The RAND function, which is the latest random-number function that was designed, does not allow multiple streams. The RAND function uses a different algorithm from the random-number CALL routines, which enable you to create multiple streams with multiple seeds. Because the state of the RAND process cannot be captured by a single seed, you cannot stop and restart the generator from its stopping point. Therefore, the RAND function allows only a single stream of numbers, but it can be used to make multiple streams, just as the RANUNI function can.
Effectively Using the Random-Number CALL Routines

Starting, Stopping, and Restarting a Stream
A reasonable use of the random-number CALL routines is starting and stopping a single stream, provided the stream never exhausts the RANUNI stream. For example, you might want SAS to perform iterations, stop, evaluate the results, and then restart the stream at the point that it stopped. The following example illustrates this principle.

Example: Starting, Stopping, and Restarting a Stream
This example generates a stream of five numbers, stops, restarts, generates five more numbers from the same stream, combines the results, and generates the full stream for comparison. In the first DATA step, the state of the random-number seed is stored in a macro variable seed for use as the starting seed in the next step. The separate streams in the example output match the full stream.

```
data u1(keep=x);
  seed = 104;
  do i = 1 to 5;
    call ranuni(seed, x);
    output;
  end;
  call symputx('seed', seed);
run;

data u2(keep=x);
  seed = &seed;
  do i = 1 to 5;
    call ranuni(seed, x);
    output;
  end;
run;

data all;
  set u1 u2;
  z = ranuni(104);
run;
proc print label;
  title 'Random Uniform Variables with Overlapping Streams';
  label x = 'Separate Streams' z = 'Single Stream';
run;
```
Random Uniform Variables with Overlapping Streams

<table>
<thead>
<tr>
<th>Obs</th>
<th>Separate Streams</th>
<th>Single Stream</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.23611</td>
<td>0.23611</td>
</tr>
<tr>
<td>2</td>
<td>0.88923</td>
<td>0.88923</td>
</tr>
<tr>
<td>3</td>
<td>0.58173</td>
<td>0.58173</td>
</tr>
<tr>
<td>4</td>
<td>0.97746</td>
<td>0.97746</td>
</tr>
<tr>
<td>5</td>
<td>0.84667</td>
<td>0.84667</td>
</tr>
<tr>
<td>6</td>
<td>0.80484</td>
<td>0.80484</td>
</tr>
<tr>
<td>7</td>
<td>0.46983</td>
<td>0.46983</td>
</tr>
<tr>
<td>8</td>
<td>0.29594</td>
<td>0.29594</td>
</tr>
<tr>
<td>9</td>
<td>0.17858</td>
<td>0.17858</td>
</tr>
<tr>
<td>10</td>
<td>0.92292</td>
<td>0.92292</td>
</tr>
</tbody>
</table>

Comparison of Changing the Seed in a CALL Routine and in a Function

Example: Changing Seeds in a CALL Routine and in a Function

If you use a CALL routine to change the seed, the results are different from using a function to change the seed. The following example shows the difference.

data seeds;
  retain Seed1 Seed2 Seed3 104;
  do i = 1 to 10;
    call ranuni(Seed1,X1);
    call ranuni(Seed2,X2);
    X3 = ranuni(Seed3);
    if i = 5 then do;
      Seed2 = 17;
      Seed3 = 17;
    end;
    output;
  end;
run;

proc print data = seeds;
  title 'Random Uniform Variables with Overlapping Streams';
  id i;
Figure 1.10 Changing Seeds in a CALL Routine and in a Function

Random Uniform Variables with Overlapping Streams

<table>
<thead>
<tr>
<th>i</th>
<th>Seed1</th>
<th>Seed2</th>
<th>Seed3</th>
<th>X1</th>
<th>X2</th>
<th>X3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>507036483</td>
<td>507036483</td>
<td>104</td>
<td>0.23611</td>
<td>0.23611</td>
<td>0.23611</td>
</tr>
<tr>
<td>2</td>
<td>190959212</td>
<td>190959212</td>
<td>104</td>
<td>0.88923</td>
<td>0.88923</td>
<td>0.88923</td>
</tr>
<tr>
<td>3</td>
<td>1249251009</td>
<td>1249251009</td>
<td>104</td>
<td>0.58173</td>
<td>0.58173</td>
<td>0.58173</td>
</tr>
<tr>
<td>4</td>
<td>2099077474</td>
<td>2099077474</td>
<td>104</td>
<td>0.97746</td>
<td>0.97746</td>
<td>0.97746</td>
</tr>
<tr>
<td>5</td>
<td>1818205895</td>
<td>17</td>
<td>17</td>
<td>0.84667</td>
<td>0.84667</td>
<td>0.84667</td>
</tr>
<tr>
<td>6</td>
<td>1728390132</td>
<td>310018657</td>
<td>17</td>
<td>0.80484</td>
<td>0.14436</td>
<td>0.80484</td>
</tr>
<tr>
<td>7</td>
<td>1003960848</td>
<td>1055505749</td>
<td>17</td>
<td>0.46983</td>
<td>0.49151</td>
<td>0.46983</td>
</tr>
<tr>
<td>8</td>
<td>635524535</td>
<td>1711572821</td>
<td>17</td>
<td>0.29594</td>
<td>0.79701</td>
<td>0.29594</td>
</tr>
<tr>
<td>9</td>
<td>383494893</td>
<td>879989345</td>
<td>17</td>
<td>0.17858</td>
<td>0.40978</td>
<td>0.17858</td>
</tr>
<tr>
<td>10</td>
<td>1981958542</td>
<td>1432895200</td>
<td>17</td>
<td>0.92292</td>
<td>0.66724</td>
<td>0.92292</td>
</tr>
</tbody>
</table>

Changing Seed2 in the CALL RANUNI statement when i=5, forces the stream for X2 to deviate from the stream for X1. However, changing Seed3 in the RANUNI function has no effect. The X3 stream continues on as if nothing has changed, and the X1 and X3 streams are the same.

Using SYSRANDOM and SYSRANEND Macro Variables to Produce Random Number Streams

Overview of the SYSRANDOM and SYSRANEND Macro Variables

Many SAS procedures (for example, FREQ, GLM, MCMC, OPTEX, and PLAN) use random number streams. These procedures use the same random number functions and CALL routines that you use in SAS DATA steps. SAS procedures use a SEED= option to provide the seed that initializes the random number stream.

SAS procedures with random number seeds create two macro variables, SYSRANDOM and SYSRANEND. You can use these macro variables to produce reproducible random number streams across procedures.

The SYSRANDOM Macro Variable

The SYSRANDOM macro variable stores the random number seed from the most recent procedure. The macro variable corresponds to the integer that is specified in the SEED= option. Many procedures, such as FREQ, GLM, MI, MCMC, OPTEX, PHREG, and
PLAN, have a SEED= option. The SEED= option specifies the integer that is used to
start the random number stream. Positive seed specifications are used as specified. If a
seed is not specified in the SEED= option, or if the seed is less than or equal to zero, the
procedure generates a seed from the clock time. You can use the SYSRANDOM macro
variable to recover both directly specified and internally generated seeds.

The SYSRANEND Macro Variable

The SYSRANEND macro variable stores a seed that can be used to start the next step in
your program. In some cases, this seed captures the state of the random number process
when a procedure is completed. Your program can contain multiple steps and can control
the random number sequence without specifying an explicit seed for each procedure.
You can start the simulation with one seed, and use the SYSRANEND macro variable to
provide the seed in all subsequent procedures. In some cases, you can also use the
SYSRANEND macro variable to stop and restart the random number generators (RNGs)
that are continuing the same stream.

There are two types of RNGs in SAS procedures. The older RNG, which is used by the
RANUNI function, starts the pseudo-random number stream with a single seed, and the
state of the process can be captured in a new seed. The GLM, GLIMMIX, MI, OPTEX,
PLAN, and other procedures use this older RNG. When the procedure exits, the value
that is stored in SYSRANEND is the new seed. You can stop and restart the generator
from its stopping point by using the SYSRANEND macro variable.

Other procedures, such as MCMC, GENMOD, LIFEREG, and PHREG, use the newer
Mersenne-Twister RNG. This RNG is also used in the RAND function and does not
propagate the state of the stream through a single seed. Some procedures use one RNG
for some computations and the other RNG for other computations. You can use the
SYSRANEND macro variable from these procedures to make a sequence of procedure
runs reproducible, but the random streams is not equal to a single, long, procedure run.

Example: Reproducing Results

The following example shows how to recover the seed and use it to reproduce a set of
results. The MCMC procedure generates samples from a posterior distribution. The
following statements produce posterior samples from a linear regression model:

```sas
PROC MCMC DATA=SASHHELP.CLASS SEED=0 OUTPOST=OUT1;
   PARMS BETA0 0 BETA1 0;
   PRIOR BETA0 BETA1 ~ NORMAL(MEAN=0, VAR=1E6);
   MU=BETA0 + BETA1*HEIGHT;
   MODEL WEIGHT ~ N(MU, VAR=137);
RUN;
```

Because SEED=0 was specified, a random number seed is automatically generated from
the clock time. This seed is stored in the SYSRANDOM macro variable. You can use a
%PUT statement to display its value:

```sas
%PUT SYSRANDOM=&SYSRANDOM;
```

The following step creates the same results as the previous step by using the same seed:

```sas
PROC MCMC DATA=SASHHELP.CLASS SEED=&SYSRANDOM OUTPOST=OUT2;
   PARMS BETA0 0 BETA1 0;
   PRIOR BETA0 BETA1 ~ NORMAL(MEAN=0, VAR=1E6);
   MU=BETA0 + BETA1*HEIGHT;
```
Example: Creating a Reproducible Random Number Stream

The PLAN procedure constructs and randomizes full factorial experimental designs. The OPTEX procedure searches a set of candidate design points for an optimal experimental design. Both procedures have a SEED= option. You can use the SYSRANEND macro variable to make a sequence of steps reproducible, as shown in the following example:

```sas
proc plan seed=17;
    factors x1=4 x2=4 x3=2 x4=2 x5=3 x6=3 x7=2 x8=2 / noprint;
    output out=cand;
run;
quit;
%put sysranend=&sysranend;
proc optex data=cand seed=&sysranend;
    class x1-x8;
    model x1-x8;
    generate n=26 iter=10 method=m_federov;
    output out=des;
run;
quit;
```

You can call PROC OPTEX multiple times and stop when a design with an efficiency (the measure of how good the design is) greater than 98% is found. You can use the SYSRANDOM and SYSRANEND macro variables to do this. The following statements call PROC OPTEX to create 100 designs and output the best \(D\)-efficiency:

```sas
proc plan seed=17;
    factors x1=4 x2=4 x3=2 x4=2 x5=3 x6=3 x7=2 x8=2 / noprint;
    output out=cand;
run;
quit;
%macro design;
    ods listing close;
    %do %until(%sysevalf(&eff > 98));
        proc optex data=Cand seed=&sysranend;
            class x1-x8;
            model x1-x8;
            generate n=26 iter=100 keep=1 method=m_federov;
            ods output efficiencies=e1;
        run;
        quit;
    data _null_;
        set e1;
        call symputx('eff', dcrterion, 'L');
    run;
    %end;
%mend design;
```
The $D$-efficiency is stored in a macro variable, and iteration stops when $D$-efficiency is greater than 98. The seed from the last step is used to reproduce and display the final results.

---

## Date and Time Intervals

### Definition of a Date and Time Interval

An interval is a unit of measurement that SAS counts within an elapsed period of time, such as days, months or hours. SAS determines date and time intervals based on fixed points on the calendar or clock. The starting point of an interval calculation defaults to the beginning of the period in which the beginning value falls, which might not be the actual beginning value that is specified. For example, if you are using the INTCK function to count the months between two dates, regardless of the actual day of the month that is specified by the date in the beginning value, SAS treats the beginning value as the first day of that month.

### Interval Names and SAS Dates

Specific interval names are used with SAS date values, and other interval names are used with SAS time and datetime values. The interval names that are used with SAS date values are YEAR, SEMIYEAR, QTR, MONTH, SEMIMONTH, TENDAY, WEEK, WEEKDAY, and DAY. The interval names that are used with SAS time and datetime values are HOUR, MINUTE, and SECOND.

Interval names that are used with SAS date values can be prefixed with 'DT' to construct interval names for use with SAS datetime values. The interval names DTYEAR, DTSEMIYEAR, DTQTR, DTMONTH, DTSEMIMONTH, DTTENDAY, DTWEEK, DTWEEKDAY, and DTDAY are used with SAS time or datetime values.

### Incrementing Dates and Times By Using Multipliers and By Shifting Intervals

SAS provides date, time, and datetime intervals for counting different periods of elapsed time. By using multipliers and shift indexes, you can create multiples of intervals and shift their starting point to construct more complex interval specifications.

The general form of an interval name is

```
namenmultiplier<shift-index>```

Both the argument *multiplier* and the *shift–index* argument are optional and default to 1. For example, YEAR, YEAR1, YEAR.1, and YEAR1.1 are all equivalent ways of specifying ordinary calendar years that begin in January. If you specify other values for *multiplier* and for *shift–index*, you can create multiple intervals that begin in different parts of the year. For example, the interval WEEK6.11 specifies six-week intervals starting on second Wednesdays.

**Commonly Used Time Intervals**

Time intervals that do not nest within years or days are aligned relative to the SAS date or datetime value 0. SAS uses the arbitrary reference time of midnight on January 1, 1960, as the origin for non-shifted intervals. Shifted intervals are defined relative to January 1, 1960.

For example, MONTH13 defines the intervals January 1, 1960, February 1, 1961, March 1, 1962, and so on, and the intervals December 1, 1958, November 1, 1957, and so on, before the base date January 1, 1960.

As another example, the interval specification WEEK6.13 defines six-week periods starting on second Fridays. The convention of alignment relative to the period that contains January 1, 1960, determines where to start counting to determine which dates correspond to the second Fridays of six-week intervals.

The following table lists time intervals that are commonly used.

<table>
<thead>
<tr>
<th>Interval</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAY3</td>
<td>Three-day intervals</td>
</tr>
<tr>
<td>WEEK</td>
<td>Weekly intervals starting on Sundays</td>
</tr>
<tr>
<td>WEEK.7</td>
<td>Weekly intervals starting on Saturdays</td>
</tr>
<tr>
<td>WEEK6.13</td>
<td>Six-week intervals starting on second Fridays</td>
</tr>
<tr>
<td>WEEK2</td>
<td>Biweekly intervals starting on first Sundays</td>
</tr>
<tr>
<td>WEEK1.1</td>
<td>Same as WEEK</td>
</tr>
<tr>
<td>WEEK.2</td>
<td>Weekly intervals starting on Mondays</td>
</tr>
<tr>
<td>WEEK6.3</td>
<td>Six-week intervals starting on first Tuesdays</td>
</tr>
<tr>
<td>WEEK6.11</td>
<td>Six-week intervals starting on second Wednesdays</td>
</tr>
<tr>
<td>WEEK4</td>
<td>Four-week intervals starting on first Sundays</td>
</tr>
<tr>
<td>WEEKDAY</td>
<td>Five-day work week with a Saturday-Sunday weekend</td>
</tr>
<tr>
<td>WEEKDAY1W</td>
<td>Six-day week with Sunday as a weekend day</td>
</tr>
<tr>
<td>WEEKDAY35W</td>
<td>Five-day week with Tuesday and Thursday as weekend days (W indicates that day 3 and day 5 are weekend days)</td>
</tr>
<tr>
<td>Interval</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>WEEKDAY17W</td>
<td>Same as WEEKDAY</td>
</tr>
<tr>
<td>WEEKDAY67W</td>
<td>Five-day week with Friday and Saturday as weekend days</td>
</tr>
<tr>
<td>WEEKDAY3.2</td>
<td>Three-weekday intervals with Saturday and Sunday as weekend days (The intervals are aligned with respect to Jan. 1, 1960. For intervals that nest within a year, it is not necessary to go back to Jan. 1, 1960 to determine the alignment.)</td>
</tr>
<tr>
<td>TENDAY4.2</td>
<td>Four ten-day periods starting at the second TENDAY period</td>
</tr>
<tr>
<td>SEMIMONTH2.2</td>
<td>Intervals from the sixteenth of one month through the fifteenth of the next month</td>
</tr>
<tr>
<td>MONTH2.2</td>
<td>February–March, April–May, June–July, August–September, October–November, and December–January of the following year</td>
</tr>
<tr>
<td>MONTH2</td>
<td>January–February, March–April, May–June, July–August, September–October, November–December</td>
</tr>
<tr>
<td>QTR3.2</td>
<td>Nine-month intervals starting on February 1, 1960, November 1, 1960, August 1, 1961, May 1, 1962, and so on.</td>
</tr>
<tr>
<td>SEMIYEAR.3</td>
<td>Six-month intervals, March–August, and September–February</td>
</tr>
<tr>
<td>YEAR.10</td>
<td>Fiscal years starting in October</td>
</tr>
<tr>
<td>YEAR2.7</td>
<td>Biennial intervals starting in July of even years</td>
</tr>
<tr>
<td>YEAR2.19</td>
<td>Biennial intervals starting in July of odd years</td>
</tr>
<tr>
<td>YEAR4.11</td>
<td>Four-year intervals starting in November of leap years (frequency of U.S. presidential elections)</td>
</tr>
<tr>
<td>YEAR4.35</td>
<td>Four-year intervals starting in November of even years between leap years (frequency of U.S. midterm elections)</td>
</tr>
<tr>
<td>DTMONTH13</td>
<td>Thirteen-month intervals starting at midnight of January 1, 1960, such as November 1, 1957, December 1, 1958, January 1, 1960, February 1, 1961, and March 1, 1962</td>
</tr>
<tr>
<td>HOUR8.7</td>
<td>Eight-hour intervals starting at 6 a.m., 2 p.m., and 10 p.m. (might be used for work shifts)</td>
</tr>
</tbody>
</table>
Retail Calendar Intervals: ISO 8601 Compliant

The retail industry often accounts for its data by dividing the yearly calendar into four 13-week periods, based on one of the following formats: 4-4-5, 4-5-4, or 5-4-4. The first, second, and third numbers specify the number of weeks in the first, second, and third months of each period, respectively.

The intervals that are created from the formats can be used in any of the following functions: INTCINDEX, INTCK, INTCYCLE, INTFIT, INTFMT, INTGET, INTINDEX, INTNX, INTSEAS, INTSHIFT, and INTTEST.

Custom Time Intervals

Reasons for Using Custom Time Intervals

Standard time intervals (for example, QTR, MONTH, WEEK, and so on) do not always fit the data. Also, some time series are measured at standard intervals where there are gaps in the data. For example, you might want to use fiscal months that begin on the 10th day of each month. In this case, using the MONTH interval is not appropriate because the MONTH interval begins on the first day of each month. You can use a custom interval to model data at a frequency that is familiar to the business and to eliminate gaps in the data by compressing the data. The intervals must be listed in ascending order. There cannot be gaps between intervals, and intervals cannot overlap.

As another example, you might want to collect data hourly for a business that is closed at night. In this case, using the DTHOUR interval results in gaps in the data that can cause problems in standard time series analysis. You might also want to calculate the number of business days between dates, excluding holidays and weekends, but holidays are counted when you use the INTCK function with the WEEKDAY interval. These are cases in which custom intervals can be used effectively.

Using Custom Time Intervals in a SAS Program

You can define custom intervals in a data set within a SAS program. Using a custom interval requires that you follow two steps for each interval:

1. Associate a data set name with a custom interval name by using the INTERVALDS= system option in an OPTIONS statement.

   Here is an example of the arguments in an INTERVALDS= system option. The example associates the data set StoreHoursDS with the custom interval StoreHours:

   ```sas
   options intervalds=(StoreHours, StoreHoursDS);
   ```

   For more information, see “INTERVALDS= System Option” in SAS Viya System Options: Reference.

2. Create a data set that describes the custom interval.

   The data set must contain the begin variable; it can also contain end and season variables. In your SAS program, include a FORMAT statement that is associated with the begin variable that specifies a SAS date, datet ime, or numeric format that matches the begin variable data. If an end variable is present, include it in the FORMAT statement. A numeric format that is not a SAS date or SAS datet ime format indicates that the values are observation numbers. If the end variable is not present, the implied value of end at each observation is one less than the value of begin at the next observation.
Include in the span of the custom interval data set any dates or times that are necessary for performing calculations on the time series, including backcasting, forecasting, and other operations that might extend beyond the series.

After you define custom intervals by using the preceding steps, the custom interval can be specified in SAS procedures and functions in places where a standard time interval can be specified.

The following examples show how to use some of the date and time functions. You can also find additional examples in the SAS Press book excerpt, *SAS Functions by Example*, by Ron Cody. This SAS Press book contains information for approximately 200 functions. Also, Ron includes many examples and details concerning optional arguments.

**Example 1: Creating Store Hours for a Business Using the INTNX Function**

The following DATA step creates the StoreHoursDS data set for a business that is open from 9:00 AM to 6:00 PM Monday through Friday, and Saturday from 9:00 AM to 1:00 PM. The example uses the “INTNX Function” on page 370, which increments a date, time, or datetime value by a given time interval, and returns a date, time, or datetime value. In this example, StoreHours is the interval, and StoreHoursDS is the data set that contains user-supplied holidays:

```sas
options intervalds=(StoreHours=StoreHoursDS);
data StoreHoursDS(keep=begin end);
  start = '01JAN2009'd;
  stop  = '31DEC2009'd;
  do date = start to stop;
    dow = weekday(date);
    datetime=dhms(date,0,0,0);
    if dow not in (1,7) then
      do hour = 9 to 17;
        begin=intnx('hours',datetime,hour,'b');
        end=intnx('hours',datetime,hour,'e');
        output;
      end;
    else if dow = 7 then
      do hour = 9 to 12;
        begin=intnx('hours',datetime,hour,'b');
        end=intnx('hours',datetime,hour,'e');
        output;
      end;
  end;
  format begin end datetime.;
run;
title 'Store Hours Custom Interval';

proc print data=StoreHoursDS (obs=18);
run;
```
The output shows the first 18 observations of the custom interval data set.

**Figure 1.11  A Custom Interval for Store Hours**

<table>
<thead>
<tr>
<th>Obs</th>
<th>begin</th>
<th>end</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>01JAN09:09:00:00</td>
<td>01JAN09:09:59:59</td>
</tr>
<tr>
<td>2</td>
<td>01JAN09:10:00:00</td>
<td>01JAN09:10:59:59</td>
</tr>
<tr>
<td>3</td>
<td>01JAN09:11:00:00</td>
<td>01JAN09:11:59:59</td>
</tr>
<tr>
<td>4</td>
<td>01JAN09:12:00:00</td>
<td>01JAN09:12:59:59</td>
</tr>
<tr>
<td>5</td>
<td>01JAN09:13:00:00</td>
<td>01JAN09:13:59:59</td>
</tr>
<tr>
<td>6</td>
<td>01JAN09:14:00:00</td>
<td>01JAN09:14:59:59</td>
</tr>
<tr>
<td>7</td>
<td>01JAN09:15:00:00</td>
<td>01JAN09:15:59:59</td>
</tr>
<tr>
<td>8</td>
<td>01JAN09:16:00:00</td>
<td>01JAN09:16:59:59</td>
</tr>
<tr>
<td>9</td>
<td>01JAN09:17:00:00</td>
<td>01JAN09:17:59:59</td>
</tr>
<tr>
<td>10</td>
<td>02JAN09:09:00:00</td>
<td>02JAN09:09:59:59</td>
</tr>
<tr>
<td>11</td>
<td>02JAN09:10:00:00</td>
<td>02JAN09:10:59:59</td>
</tr>
<tr>
<td>12</td>
<td>02JAN09:11:00:00</td>
<td>02JAN09:11:59:59</td>
</tr>
<tr>
<td>13</td>
<td>02JAN09:12:00:00</td>
<td>02JAN09:12:59:59</td>
</tr>
<tr>
<td>14</td>
<td>02JAN09:13:00:00</td>
<td>02JAN09:13:59:59</td>
</tr>
<tr>
<td>15</td>
<td>02JAN09:14:00:00</td>
<td>02JAN09:14:59:59</td>
</tr>
<tr>
<td>16</td>
<td>02JAN09:15:00:00</td>
<td>02JAN09:15:59:59</td>
</tr>
<tr>
<td>17</td>
<td>02JAN09:16:00:00</td>
<td>02JAN09:16:59:59</td>
</tr>
<tr>
<td>18</td>
<td>02JAN09:17:00:00</td>
<td>02JAN09:17:59:59</td>
</tr>
</tbody>
</table>

**Example 2: Creating the Fiscal Month Custom Interval Using the INTNX Function**

The following DATA step creates the FMDS data set to define a custom interval, FiscalMonth. It is appropriate for a business that uses fiscal months that start on the 10th day of each month. The SAME alignment option of the INTNX function specifies that the dates that are generated by the INTNX function be the same day of the month as the date in the start variable. The MONTH function assigns the month of the begin variable to the season variable, which specifies monthly seasonality:

```sas
options intervalds=(FiscalMonth=FMDS);

data FMDS(keep=begin season);
  start = '10JAN1999'd;
  stop = '10JAN2001'd;
  nmonths = intck('month',start,stop);
  do i=0 to nmonths;
    begin = intnx('month',start,i,'S');
    season = month(begin);
    output;
  end;
format begin date9.;
```
The difference between the custom FiscalMonth interval and a standard interval is seen in the following example. The output from the program compares how the data is accumulated. For the FiscalMonth interval, values in the first nine days of the month are accumulated with the interval that begins in the previous month. For the standard MONTH interval, values in the first nine days of the month are accumulated with the calendar month.
data sales(keep=date sales);
   do date = '01JAN2000'd to '31DEC2000'd;
      month = MONTH(date);
      dayofmonth = DAY(date);
      sales = 0;
      if (dayofmonth lt 10) then sales= month/9;
      output;
   end;
   format date monyy.;
run;

proc timeseries data=sales out=dataInFiscalMonths;
   id date interval=FiscalMonth accumulate=total;
   var sales;
run;

proc timeseries data=sales out=dataInStdMonths;
   id date interval=Month accumulate=total;
   var sales;
run;

data compare;
   merge dataInFiscalMonths(rename=(sales=FM_sales))
      dataInStdMonths(rename=(sales=SM_sales));
      by date;
run;

title 'Standard Monthly Data and Fiscal Month Data';

proc print data=compare;
run;
Example 3: Using Custom Intervals with the INTCK Function

The following example uses custom intervals in the INTCK function to omit holidays when counting business days:

```plaintext
options intervalds=(BankingDays=BankDayDS);
data BankDayDS(keep=begin);
  start = '15DEC1998'd;
  stop  = '15JAN2002'd;
  nwkdays = intck('weekday',start,stop);
  do i = 0 to nwkdays;
    begin = intnx('weekday',start,i);
    year = year(begin);
    if begin ne holiday('NEWYEAR',year) and
...
begin ne holiday('MLK',year) and
begin ne holiday('USPRESIDENTS',year) and
begin ne holiday('MEMORIAL',year) and
begin ne holiday('USINDEPENDENCE',year) and
begin ne holiday('LABOR',year) and
begin ne holiday('COLUMBUS',year) and
begin ne holiday('VETERANS',year) and
begin ne holiday('THANKSGIVING',year) and
begin ne holiday('CHRISTMAS',year) then
output;
end;
format begin date9.;
run;
data CountDays;
start = '01JAN1999'd;
stop  = '31DEC2001'd;
ActualDays = intck('DAYS',start,stop);
Weekdays   = intck('WEEKDAYS',start,stop);
BankDays   = intck('BankingDays',start,stop);
format start stop date9.;
run;
title 'Methods of Counting Days';
proc print data=CountDays;
run;

Figure 1.14  Bank Days Custom Interval

<table>
<thead>
<tr>
<th>Obs</th>
<th>start</th>
<th>stop</th>
<th>ActualDays</th>
<th>Weekdays</th>
<th>BankDays</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>01JAN1999</td>
<td>31DEC2001</td>
<td>1095</td>
<td>781</td>
<td>757</td>
</tr>
</tbody>
</table>

**Best Practices for Custom Interval Names**

The following items list best practices to use when you are creating custom interval names:

- Custom interval names should not conflict with existing SAS interval names. For example, if BASE is a SAS interval name, do not use the following formats for the name of a custom interval:
  - BASE
  - BASEm
  - BASEm.n
  - DTBASE
  - DTBASEm
  - DTBASEm.n
The following paragraphs describe the variables:

\( m \)

specifies an optional multiplier that sets the interval equal to a multiple of the period of the basic interval type. For example, the interval YEAR2 consists of two-year, or biennial, periods.

\( n \)

specifies an optional shift index that shifts the interval to start at a specified subperiod starting point. For example, YEAR.3 specifies yearly periods that are shifted to start on the first day of March of each calendar year and end in February of the following year.

If you define a custom interval such as CUSTBASE, then you can use CUSTBASE\( m.n \).

Because of these rules, do not begin a custom interval name with DT, and do not end the custom interval name with a number.

• To ensure that custom intervals work reliably, always include one of the following formats:

  date-format with beginning and ending values
  specifies intervals that are used with SAS date values.

  datetime-format with beginning and ending values
  specifies intervals that are used with SAS datetime values.

  number-format with beginning and ending values
  specifies intervals that are used with SAS observation numbers.

• Beginning and ending values should be of the same type. Both values should be date values, datetime values, or observation numbers.

• Calculations for custom intervals cannot be performed before the first begin value or after the last end value. If you use the begin variable only, then the last end value that you can calculate is the last begin value \(-1\). If you forecast or backcast the time series, be sure to include time definitions for the forecast and backcast values.

• CUSTBASE\( m.2 \) is never able to calculate a beginning period for the first date value in a data set because, by definition, the beginning of the first interval starts before the data set begins (at the \(-(m-2)\) th observation). For example, you might have an interval called CUSTBASE4.2 with the first interval beginning before the first observation:

<table>
<thead>
<tr>
<th>OBS</th>
<th>Start/End of CUSTBASE4.2 Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2</td>
<td>Start of partial CUSTBASE4.2 interval observation: (-(4-2) = -2.)</td>
</tr>
<tr>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>End of partial CUSTBASE4.2 interval observation: This is the first observation in the data set.</td>
</tr>
<tr>
<td>1</td>
<td>Start of first complete CUSTBASE4.2 interval.</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>End of first complete CUSTBASE4.2 interval.</td>
</tr>
<tr>
<td>6</td>
<td>Start of 2nd CUSTBASE4.2 interval.</td>
</tr>
</tbody>
</table>

If you execute the INTNX function, the result must return the date that is associated with OBS \(-2\), which does not exist:

\[
\text{INTNX('CUSTBASE4.2', date-at-obs1, 0, 'B');}
\]
Include a variable named `season` in the custom interval data set to define the seasonal index. This result is similar to the result of `INTINDEX('interval', date);`

In the following example, the data set is associated with the custom interval `CUSTWEEK`:

<table>
<thead>
<tr>
<th>Obs</th>
<th>begin</th>
<th>season</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>27DEC59</td>
<td>52</td>
</tr>
<tr>
<td>2</td>
<td>03JAN60</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>10JAN60</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>17JAN60</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>24JAN60</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>31JAN60</td>
<td>5</td>
</tr>
</tbody>
</table>

The following examples show the results of using custom interval functions:

- `INTINDEX('CUSTWEEK', '03JAN60'D);` returns a value of 1.
- `INTSEAS('CUSTWEEK');` returns a value of 52, which is the largest value of the season.
- `INTCYCLE('CUSTWEEK');` returns `CUSTWEEK52`, which is `CUSTBASEmax(season)`.
- `INTCINDEX('CUSTWEEK', '27DEC59'D);` returns a value of 1.
- `INTCINDEX('CUSTWEEK', '03JAN60'D)` returns a value of 2.

A new cycle begins when the season is less than the previous value of `season`.

- Seasonality occurs when seasons are identified, such as `season1`, `season2`, `season3`, and so on. If all seasons are identified as `season1`, then there is no seasonality. No seasonality is also called trivial seasonality.

Only trivial seasonality is available for intervals of the form `CUSTBASEm`. If `season` is not included in the data set, then trivial seasonality is valid.

- If a format for the `begin` variable is included in a data set, then a message generated by `INTFMT('CUSTBASE', 'l')` or `INTFMT('CUSTBASE', 's')` appears. The message recommends a format based on the format that is specified in the data set.
- Executing `INTSHIFT('CUSTBASE');` or `INTSHIFT('CUSTBASEm.s');` returns the value of `CUSTBASE`.
- With `INTNX`, `INTCK`, and `INTTEST`, the intervals `CUSTBASE`, `CUSTBASEm`, and `CUSTBASEm.s` work as expected.

---

Pattern Matching Using Perl Regular Expressions (PRX)

**Definition of Pattern Matching**

Pattern matching enables you to search for and extract multiple matching patterns from a character string in one step. Pattern matching also enables you to make several substitutions in a string in one step. You do this by using the PRX functions and CALL routines in the DATA step.
For example, you can search for multiple occurrences of a string and replace those strings with another string. You can search for a string in your source file and return the position of the match. You can find words in your file that are doubled.

**Definition of Perl Regular Expression (PRX) Functions and CALL Routines**

Perl regular expression (PRX) functions and CALL routines refers to a group of functions and CALL routines that use a modified version of Perl as a pattern-matching language to parse character strings. You can perform the following tasks:

- search for a pattern of characters within a string
- extract a substring from a string
- search and replace text with other text
- parse large amounts of text, such as web logs or other text data

Perl regular expressions comprise the character string matching category for functions and CALL routines. For a short description of these functions and CALL routines, see Functions and CALL Routines by Category in the Dictionary section of this document.

**Benefits of Using Perl Regular Expressions in the DATA Step**

Using Perl regular expressions in the DATA step enhances search-and-replace options in text. You can use Perl regular expressions to perform the following tasks:

- validate data
- replace text
- extract a substring from a string

You can write SAS programs that do not use regular expressions to produce the same results as you do when you use Perl regular expressions. However, the code without the regular expressions requires more function calls to handle character positions in a string and to manipulate parts of the string.

Perl regular expressions combine most, if not all, of these steps into one expression. The resulting code is less prone to error, easier to maintain, and clearer to read.

---

**Using Perl Regular Expressions in the DATA Step**

**Syntax of Perl Regular Expressions**

**The Components of a Perl Regular Expression**

Perl regular expressions consist of characters and special characters that are called metacharacters. When performing a match, SAS searches a source string for a substring that matches the Perl regular expression that you specify. Using metacharacters enables SAS to perform special actions. These actions include forcing the match to begin in a particular location, and matching a particular set of characters. Paired forward slashes are the default delimiters. The following two examples show metacharacters and the values that they match:

- If you use the metacharacter \\d, SAS matches a digit between 0–9.
If you use `/dt/`, SAS finds the digits in the string “Raleigh, NC 27506”.

You can see lists of PRX metacharacters in Appendix 1, “Tables of Perl Regular Expression (PRX) Metacharacters,” on page 1113. For a complete list of metacharacters, see the Perl documentation.

**Basic Syntax for Finding a Match in a String**

You use the PRXMATCH function to find the position of a matched value in a source string. PRXMATCH has the following general form:

```
/search-string/source-string/
```

The following example uses the PRXMATCH function to find the position of `search-string` in `source-string`:

```
prxmatch('world', 'Hello world!');
```

The result of PRXMATCH is the value 7, because `world` occurs in the seventh position of the string `Hello world!`.

**Basic Syntax for Searching and Replacing Text**

The basic syntax for searching and replacing text has the following form:

```
s/regular-expression/replacement-string/
```

The following example uses the PRXCHANGE function to show how substitution is performed:

```
prxchange('s/world/planet/', 1, 'Hello world!');
```

**Arguments**

- `s` specifies the metacharacter for substitution.
- `world` specifies the regular expression.
- `planet` specifies the replacement value for `world`.
- `1` specifies that the search ends when one match is found.
- `Hello world!` specifies the source string to be searched.

The result of the substitution is `Hello planet`.

**Another Example of Using Basic Syntax for Searching and Replacing Text**

Another example of using the PRXCHANGE function changes the value `Jones, Fred` to `Fred Jones`:

```
prxchange('s/\w+,(\w+)/$2 $1/,-1, 'Jones, Fred');
```

In this example, the Perl regular expression is `s/\w+,(\w+)/$2 $1/,-1, 'Jones, Fred'`. The number of times to search for a match is –1. The source string is 'Jones, Fred'. The value –1 specifies that matching patterns continue to be replaced until the end of the source is reached.

The Perl regular expression can be divided into its elements:
s specifies a substitution regular expression.

\(\w+\)
matches one or more word characters (alphanumeric and underscore). The parentheses indicate that the value is stored in capture buffer 1.

,<space>
matches a comma and a space.

\(\w+\)
matches one or more word characters (alphanumeric and underscore). The parentheses indicate that the value is stored in capture buffer 2.

/ separator between the regular expression and the replacement string.

\$2 part of the replacement string that substitutes the value in capture buffer 2, which in this case is the word after the comma, puts the substitution in the results.

<space> puts a space in the result.

\$1 puts capture buffer 1 into the result. In this case, it is the word before the comma.

**Replacing Text**
The following example uses the `\u` and `\L` metacharacters to replace the second character in MCLAUREN with a lowercase letter:

```sas
data _null_;  
x = 'MCLAUREN';  
x = prxchange("s/(MC)/\u\L$1/i", -1, x);  
put x=;  
run;
```

SAS writes the following output to the log:

```
x=McLAUREN
```

**Example 1: Validating Data**
You can test for a pattern of characters within a string. For example, you can examine a string to determine whether it contains a correctly formatted telephone number. This type of test is called data validation.

The following example validates a list of phone numbers. To be valid, a phone number must have one of the following forms: (XXX) XXX-XXXX or XXX-XXX-XXXX.

```sas
data _null_;  
  if _N_ = 1 then  
    do;  
      paren = "\([2-9]\d\d\) ?\[2-9\]\d\d-\d\d\d\d\d\d";  
      dash = "\[2-9\]\d\d-\[2-9\]\d\d-\d\d\d\d\d";  
      expression = "/(^| paren \| | \d\d\d\) |(\* \| dash \| \* )/\";  
      retain re;  
      re = prxparse(expression);  
      if missing(re) then  
        do;
```

```sas
```
putlog "ERROR: Invalid expression " expression;  
    stop;
end;
end;

length first last home business $ 16;
input first last home business;

if "prxmatch(re, home) then  
    putlog "NOTE: Invalid home phone number for " first last home;

if "prxmatch(re, business) then  
    putlog "NOTE: Invalid business phone number for " first last business;

datafiles;
Jerome Johnson (919)319-1677 (919)846-2198
Romeo Montague 800-899-2164 360-973-6201
Imani Rashid (508)852-2146 (508)366-9821
Palinor Kent . 919-782-3199
Ruby Archuleta .
Takei Ito 7042982145 .
Tom Joad 209/963/2764 2099-66-8474
;
run;

The following items correspond to the lines that are numbered in the DATA step that is shown above.

1 Create a DATA step.

2 Build a Perl regular expression to identify a phone number that matches (XXX)XXX-XXXX, and assign the variable PAREN to hold the result. Use the following syntax elements to build the Perl regular expression:

\( \) matches the open parenthesis in the area code.
[2-9] matches the digits 2–9, which is the first number in the area code.
\d matches a digit, which is the second number in the area code.
\d matches a digit, which is the third number in the area code.
\) matches the closed parenthesis in the area code.
<space>? matches the space (which is the preceding subexpression) zero or one time. Spaces are significant in Perl regular expressions. They match a space in the text that you are searching. If a space precedes the question mark metacharacter (as it does in this case), the pattern matches either zero spaces or one space in this position in the phone number.

3 Build a Perl regular expression to identify a phone number that matches XXX-XXX-XXXX, and assign the variable DASH to hold the result.

4 Build a Perl regular expression that concatenates the regular expressions for (XXX)XXX-XXXX and XXX—XXX—XXXX. The concatenation enables you to search for both phone number formats from one regular expression.

The PAREN and DASH regular expressions are placed within parentheses. The bar metacharacter (|) that is located between PAREN and DASH instructs the compiler to match either pattern. The slashes around the entire pattern tell the compiler where the start and end of the regular expression is located.
Pass the Perl regular expression to PRXPARSE and compile the expression. PRXPARSE returns a value to the compiled pattern. Using the value with other Perl regular expression functions and CALL routines enables SAS to perform operations with the compiled Perl regular expression.

Use the MISSING function to check whether the regular expression was successfully compiled.

Use the PUTLOG statement to write an error message to the SAS log if the regular expression did not compile.

Search for a valid home phone number. PRXMATCH uses the value from PRXPARSE along with the search text and returns the position where the regular expression was found in the search text. If there is no match for the home phone number, the PUTLOG statement writes a note to the SAS log.

Search for a valid business phone number. PRXMATCH uses the value from PRXPARSE along with the search text and returns the position where the regular expression was found in the search text. If there is no match for the business phone number, the PUTLOG statement writes a note to the SAS log.

Log 1.3 Output from Validating Data

NOTE: Invalid home phone number for Palinor Kent
NOTE: Invalid home phone number for Ruby Archuleta
NOTE: Invalid business phone number for Ruby Archuleta
NOTE: Invalid home phone number for Takei Ito 7042982145
NOTE: Invalid business phone number for Takei Ito
NOTE: Invalid home phone number for Tom Joad 209/963/2764
NOTE: Invalid business phone number for Tom Joad 2099-66-8474

Example 2: Matching and Replacing Text

This example uses a Perl regular expression to find a match and replace the matching characters with other characters. PRXPARSE compiles the regular expression and uses PRXCHANGE to find the match and perform the replacement. The example replaces all occurrences of a less than sign with &lt;, a common substitution when converting text to HTML.

```
data _null_; 1
  input; 2
  _infile_ = prxchange('s/<&lt;/', -1, _infile_); 3
  put _infile_; 4
  datalines; 5
  x + y < 15
  x < 10 < y
  y < 11
  ;
  run;
```

The following items correspond to the numbered lines in the DATA step that is shown above.

1 Create a DATA step.
2 Bring an input data record into the input buffer without creating any SAS variables.
3 Call the PRXCHANGE routine to perform the pattern exchange. The format for the regular expression is s/regular-expression/replacement-text/. The s
before the regular expression signifies that this is a substitution regular expression. The –1 is a special value that is passed to PRXCHANGE and indicates that all possible replacements should be made.

4 Write the current output line to the log by using the _INFILE_ option with the PUT statement.

5 Identify the input file.

**Log 1.4 Output from Replacing Text**

```
x + y &lt; 15
x &lt; 10 &lt; y
y &lt; 11
```

The ability to pass a regular expression to PRXCHANGE and return a result enables calling PRXCHANGE from a PROC SQL query. The following query produces a column with the same character substitution as in the preceding example. From the input table the query reads text_lines, changes the text for the column line, and places the results in a column named html_line:

```
proc sql;
  select prxchange('s/&lt;/&lt;/', -1, line)
    as html_line
  from text_lines;
quit;
```

**Example 3: Extracting a Substring from a String**

You can use Perl regular expressions to find and easily extract text from a string. In this example, the DATA step creates a subset of North Carolina business phone numbers. The program extracts the area code and checks it against a list of area codes for North Carolina.

```
data _null_; 1
  if _N_ = 1 then do;
    paren = "\((\[[2-9]d\d\d\) ?[2-9]\d\d-\d\d\d\d\d\); 2
    dash = "\([2-9]\d\d\-\[2-9]\d\d\-\d\d\d\d\d\d\); 3
    regexp = "/(" || paren || ")|(" || dash || ")*/; 4
    retain re;
    re = prxparse(regexp);
    if missing(re) then 6
      do;
        putlog "ERROR: Invalid regexp * regexp; 7"
        stop;
      end;
    end;
  end;
  retain areacode_re;
  areacode_re = prxparse("/828|336|704|910|919|252/*"); 8
  if missing(areacode_re) then
    do;
      putlog "ERROR: Invalid area code regexp";
      stop;
    end;
  end;
```
length first last home business $ 25;
length areacode $ 3;
input first last home business;

if "prxmatch(re, home) then
   putlog "NOTE: Invalid home phone number for " first last home;

if prxmatch(re, business) then 3
do;
   which_format = prxparen(re); 10
   call prxposn(re, which_format, pos, len);
   areacode = substr(business, pos, len);
   if prxmatch(areacode_re, areacode) then 12
      put "In North Carolina: " first last business;
   end;
else
   putlog "NOTE: Invalid business phone number for " first last business;
end;
datalines;
Jerome Johnson (919)319-1677 (919)846-2198
Romeo Montague 800-899-2164 360-973-6201
Imani Rashid (508)852-2146 (508)366-9821
Palinor Kent 704-782-4673 704-782-3199
Ruby Archuleta 905-384-2839 905-328-3892
Takei Ito 704-298-2145 704-298-4738
Tom Joad 515-372-4829 515-389-2838

1 Create a DATA step.
2 Build a Perl regular expression to identify a phone number that matches (XXX)XXX-XXXX, and assign the variable PAREN to hold the result. Use the following syntax elements to build the Perl regular expression:

\( matches the open parenthesis in the area code. The open parenthesis marks the start of the submatch.
\d matches a digit, which is the second number in the area code.
\d matches a digit, which is the third number in the area code.
\) matches the closed parenthesis in the area code. The closed parenthesis marks the end of the submatch.
? matches the space (which is the preceding subexpression) zero or one time. Spaces are significant in Perl regular expressions. They match a space in the text that you are searching. If a space precedes the question mark metacharacter (as it does in this case), the pattern matches either zero spaces or one space in this position in the phone number.

3 Build a Perl regular expression to identify a phone number that matches XXX-XXX-XXXX, and assign the variable DASH to hold the result.
4 Build a Perl regular expression that concatenates the regular expressions for (XXX)XXX-XXXX and XXX—XXX—XXXX. The concatenation enables you to search for both phone number formats from one regular expression.

The PAREN and DASH regular expressions are placed within parentheses. The bar metacharacter (|) that is located between PAREN and DASH instructs the compiler to
match either pattern. The slashes around the entire pattern tell the compiler where the start and end of the regular expression is located.

5 Pass the Perl regular expression to PRXPARSE and compile the expression. PRXPARSE returns a value to the compiled pattern. Using the value with other Perl regular expression functions and CALL routines enables SAS to perform operations with the compiled Perl regular expression.

6 Use the MISSING function to check whether the Perl regular expression compiled without error.

7 Use the PUTLOG statement to write an error message to the SAS log if the regular expression did not compile.

8 Compile a Perl regular expression that searches a string for a valid North Carolina area code.

9 Search for a valid business phone number.

10 Use the PRXPAREN function to determine which submatch to use. PRXPAREN returns the last submatch that was matched. If an area code matches the form (XXX), PRXPAREN returns the value 2. If an area code matches the form XXX, PRXPAREN returns the value 4.

11 Call the PRXPOSN routine to retrieve the position and length of the submatch.

12 Use the PRXMATCH function to determine whether the area code is a valid North Carolina area code, and write the observation to the log.

Log 1.5  Output from Extracting a Substring from a String

| In North Carolina: Jerome Johnson (919)846-2198 |
| In North Carolina: Palinor Kent 704-782-3199 |
| In North Carolina: Takei Ito 704-298-4738 |

Example 4: Another Example of Extracting a Substring from a String

In this example, the PRXPOSN function is passed to the original search text instead of to the position and length variables. PRXPOSN returns the text that is matched.

```sas
data _null_; 3
length first last phone $ 16;
retain re;
if _N_ = 1 then do; 4
  re=prxparse("/\((\[[2-9]\d\d)\) ?\[2-9]\d\d-\d\d\d\d/\)\); 5
end;

input first last phone & 16.;
if prxmatch(re, phone) then do; 6
  area_code = prxposn(re, 1, phone); 7
  if area_code ^in ("828"
    "336"
    "704"
    "910"
    "919"
    "252") then
    putlog "NOTE: Not in North Carolina: "
    first last phone; 8
```
The following items correspond to the numbered lines in the DATA step that is shown above.

1. Create a DATA step.

2. If this is the first record, find the value of `re`.

3. Build a Perl regular expression for pattern matching. Use the following syntax elements to build the Perl regular expression:

   /  is the beginning delimiter for a regular expression.

   \(  marks the next character entry as a character or a literal.

   (  marks the start of the submatch.

   [2–9] matches the digits 2–9 and identifies the first number in the area code.

   \d matches a digit, which is the second number in the area code.

   \d matches a digit, which is the third number in the area code.

   ) matches the close parenthesis in the area code. The close parenthesis marks the end of the submatch.

   ? matches the space (which is the preceding subexpression) zero or one time. Spaces are significant in Perl regular expressions. The spaces match a space in the text that you are searching. If a space precedes the question mark metacharacter (as it does in this case), the pattern matches either zero spaces or one space in this position in the phone number.

   || is the concatenation operator.

   [2–9] matches the digits 2–9 and identifies the first number in the seven-digit phone number.

   \d matches a digit, which is the second number in the seven-digit phone number.

   \d matches a digit, which is the third number in the seven-digit phone number.

   – is the hyphen between the first three and last four digits of the phone number after the area code.

   \d matches a digit, which is the fourth number in the seven-digit phone number.

   \d matches a digit, which is the fifth number in the seven-digit phone number.

   \d matches a digit, which is the sixth number in the seven-digit phone number.

   \d matches a digit, which is the seventh number in the seven-digit phone number.

   /  is the ending delimiter for a regular expression.

4. Return the position at which the string begins.
5 Identify the position at which the area code begins.

6 Search for an area code from the list. If the area code is not valid for North Carolina, use the PUTLOG statement to write a note to the SAS log.

7 Identify the input file.

**Log 1.6** Output from Extracting a Substring from a String

```
NOTE: Not in North Carolina: Lucy Mallory (800)899-2164
NOTE: Not in North Carolina: Tom Joad (508)852-2146
```

---

**Writing Perl Debug Output to the SAS Log**

The DATA step provides debugging support with the CALL PRXDEBUG routine. CALL PRXDEBUG enables you to turn on and off Perl debug output messages that are sent to the SAS log.

The following example writes Perl debug output to the SAS log.

```sas
data _null_
;
/* CALL PRXDEBUG(1) turns on Perl debug output. */
call prxdebug(1);
putlog 'PRXPARSE: ';  
re = prxparse('/[bc]d[ef*g]+h[ij]k$/');
putlog 'PRXMATCH: ';  
pos = prxmatch(re, 'abcdefg_gh_');
/* CALL PRXDEBUG(0) turns off Perl debug output. */
call prxdebug(0);
run;
```

SAS writes the following output to the log.
Log 1.7  SAS Debugging Output

PRXPARSE:
Compiling REx '^[bc]d\{ef\}g+h\{ij\}k$'
size 41 first at 1
rarest char g at 0
rarest char d at 0
 1: ANYOF^[bc](10)
 10: EXACT <d>(12)
 12: CURLYX[0] {1,32767}(26)
 14: OPEN1(16)
 16: EXACT <e>(18)
 18: STAR(21)
 19: EXACT <f>(23)
 21: EXACT <g>(23)
 23: CLOSE1(25)
 25: WHILEM[1/1](0)
 26: NOTHING(27)
 27: EXACT <h>(29)
 29: ANYOF^[ij](38)
 38: EXACT <k>(40)
 40: EOL(41)
 41: END(0)
anchored `de' at 1 floating `gh' at 3..2147483647 (checking floating) stclass
`ANYOF^[bc]` minlen 7

PRXMATCH:
Guessing start of match, REx '^[bc]d\{ef\}g+h\{ij\}k$' against `abcdefg_gh_`...
Did not find floating substr `gh'...
Match rejected by optimizer

For a detailed explanation of Perl debug output, see “CALL PRXDEBUG Routine” on page 785.

Perl Artistic License Compliance

Perl regular expressions are supported beginning with SAS® 9.

The PRX functions use a modified version of Perl 5.6.1 to perform regular expression compilation and matching. Perl is compiled into a library for use with SAS. This library is shipped with SAS® 9. The modified and original Perl 5.6.1 files are freely available in a ZIP file from the Technical Support Web site. The ZIP file is provided to comply with the Perl Artistic License and is not required in order to use the PRX functions. Each of the modified files has a comment block at the top of the file describing how and when the file was changed. The executables were given nonstandard Perl names. The standard version of Perl can be obtained from the Perl website.

Only Perl regular expressions are accessible from the PRX functions. Other parts of the Perl language are not accessible. The modified version of Perl regular expressions does not support the following items:

- Perl variables (except the capture buffer variables $1 - $n, which are supported).
- The regular expression options /c and /g, and the /e option with substitutions.
- The regular expression option /o in SAS 9.0. (It is supported in SAS 9.1 and later.)
- Named characters, which use the \N{name} syntax.
• The metacharacters \pP, \PP, and \X.
• Executing Perl code within a regular expression, which includes the syntax (\(?{code}\)), (\??{code}), and (\?{code}).
• Unicode pattern matching.
• Using ?PATTERN?. ? is treated like an ordinary regular expression start and end delimiter.
• The metacharacter \G.
• Perl comments between a pattern and replacement text. For example: s\{regexp\} # perl comment \{replacement\} is not supported.
• Matching backslashes with m/\\/. Instead, use m/\V to match a backslash.

SAS Functions for Web Applications

Four functions that manipulate web-related content are available in SAS software. HTML ENCODE and URLENCODE return encoded strings. HTML DECODE and URL DECODE return decoded strings.

Functions in SAS and CAS

SAS Viya enables you to process functions in SAS and in SAS Cloud Analytic Services (CAS). Some functions and other language elements are not supported in SAS. An example is the DATALINES statement. This section explains how to run code using DATALINES.

Creating a Data Set Using the DATALINES Statement

You can use the DATALINES statement to add data to a data set. The DATALINES statement is supported in SAS but not in CAS. This example shows how to use the DATALINES statement in SAS to create a data set and then access the data set in CAS and use the function’s functionality in CAS.

Examples of Creating a Data Set Using the DATALINES Statement in SAS and CAS

This example shows the results of using the STRIP function to delete leading and trailing blanks. The DATALINES statement is used to create a data set.

```sas
data lengthn;
  input string $char8.;
  original='*' || string || '*';
  stripped='*' || strip(string) || '*';
datalines;
abcd
abcd
abcd
eabcdefgh
xyz
```

Chapter 1 • SAS Functions and CALL Routines
```sas
proc print data=lengthn;
run;
```

**The SAS System**

<table>
<thead>
<tr>
<th>Obs</th>
<th>string</th>
<th>original</th>
<th>stripped</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>abcd</td>
<td><em>abcd</em></td>
<td><em>abcd</em></td>
</tr>
<tr>
<td>2</td>
<td>abcd</td>
<td>* abcd*</td>
<td><em>abcd</em></td>
</tr>
<tr>
<td>3</td>
<td>abcd</td>
<td>* abcd*</td>
<td><em>abcd</em></td>
</tr>
<tr>
<td>4</td>
<td>abcdefgh</td>
<td><em>abcdefgh</em></td>
<td><em>abcdefgh</em></td>
</tr>
<tr>
<td>5</td>
<td>xyz</td>
<td><em>xyz</em></td>
<td><em>xyz</em></td>
</tr>
</tbody>
</table>

This example uses the DATA step to delete leading and trailing blanks. The first section of code in this example runs in SAS and creates the data set, lengthn, with the CAS engine. The second section of code runs in CAS, accesses the data in lengthn that is in CAS, and then creates the lengthnn data set.

```sas
data casuser.lengthn;
input string $char8.;
datalines;
  abcd
  abcd
  abcdefgh
  xyz
;

data casuser.lengthnn;
set casuser.lengthn;
original='*' || string || '*';
stripped='*' || strip(string) || '*';
run;
```

```sas
proc print data=casuser.lengthnn;
run;
```

**The SAS System**

<table>
<thead>
<tr>
<th>Obs</th>
<th>string</th>
<th>original</th>
<th>stripped</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>abcd</td>
<td><em>abcd</em></td>
<td><em>abcd</em></td>
</tr>
<tr>
<td>2</td>
<td>abcd</td>
<td>* abcd*</td>
<td><em>abcd</em></td>
</tr>
<tr>
<td>3</td>
<td>abcd</td>
<td>* abcd*</td>
<td><em>abcd</em></td>
</tr>
<tr>
<td>4</td>
<td>abcdefgh</td>
<td><em>abcdefgh</em></td>
<td><em>abcdefgh</em></td>
</tr>
<tr>
<td>5</td>
<td>xyz</td>
<td><em>xyz</em></td>
<td><em>xyz</em></td>
</tr>
</tbody>
</table>
Chapter 2
Dictionary of Functions and Call Routines for SAS and the CAS Server

SAS Functions and CALL Routines for CAS and SAS by Category ................. 63

Dictionary ......................................................... 79
ABS Function .................................................... 79
AIRY Function .................................................. 79
ANYALNUM Function ........................................... 80
ANYALPHA Function ............................................ 82
ANYCNTRNL Function ......................................... 84
ANYDIGIT Function ............................................. 86
ANYFIRST Function ............................................. 87
ANYGRAPH Function ........................................... 89
ANYLOWER Function ........................................... 91
ANYNAME Function ............................................. 93
ANYPRT Function .............................................. 95
ANYPUNCT Function ........................................... 97
ANYSAMPLE Function .......................................... 99
ANYUPPER Function ........................................... 101
ANYXDIGIT Function .......................................... 102
ARCOS Function ............................................... 104
ARCOSH Function ............................................... 105
ARSIN Function ............................................... 106
ARSINH Function ............................................... 106
ARTANH Function ............................................... 107
ATAN Function .................................................. 108
ATAN2 Function ................................................ 109
BAND Function .................................................. 110
BETA Function ................................................... 111
BETAINV Function ............................................. 112
BLACKCLPRC Function ........................................ 113
BLACKPTPRC Function ........................................ 115
BLKSHCLPRC Function ......................................... 116
BLKSHPTPRC Function ......................................... 118
BLSHIFT Function .............................................. 120
BNOT Function .................................................. 121
BOR Function .................................................... 121
BRSHIFT Function .............................................. 122
BXOR Function .................................................. 123
BYTE Function ................................................... 123
CALL LABEL Routine .......................................... 124
CALL MISSING Routine ........................................ 126
CALL PRXFREE Routine ....................................... 127
CALL SCAN Routine ........................................... 128
CALL SLEEP Routine ............................................................. 138
CALL STREAMINIT Routine .................................................. 139
CALL VNAME Routine .......................................................... 141
CAT Function .................................................................. 143
CATS Function ................................................................ 145
CATT Function ................................................................ 147
CATX Function ................................................................ 149
CDF Function ................................................................... 153
CEIL Function .................................................................. 171
CEILZ Function ................................................................ 172
CHOOS Function ................................................................ 173
CHOOSN Function ................................................................ 175
CNCT Function .................................................................. 176
COALESCE Function ............................................................ 177
COALESCEC Function ............................................................ 178
COLLATE Function ............................................................... 179
COMB Function .................................................................. 183
COMPARE Function ............................................................... 184
COMPBL Function .............................................................. 187
COMPFUZZ Function ........................................................... 188
COMPOUND Function .......................................................... 190
COMPRESS Function ............................................................ 191
CONSTANT Function ............................................................. 195
CONVX Function ................................................................. 199
CONVXP Function ............................................................... 200
COS Function .................................................................... 201
COSH Function ................................................................... 202
COT Function ..................................................................... 203
COUNT Function .................................................................. 204
COUNTC Function ............................................................... 206
COUNTW Function ................................................................ 209
CSC Function ..................................................................... 212
CSS Function ..................................................................... 213
CUMIPMT Function ............................................................. 214
CUMPRINC Function ............................................................ 215
CV Function ....................................................................... 216
DAIRY Function .................................................................. 216
DATDIF Function ............................................................... 217
DATE Function .................................................................... 220
DATEJUL Function ............................................................... 221
DATEPART Function ............................................................. 221
DATETIME Function ............................................................. 222
DAY Function .................................................................... 223
DEQUOTE Function .............................................................. 224
DEVIANCE Function ............................................................ 226
DHMS Function .................................................................. 230
DIF Function ...................................................................... 231
DIGAMMA Function ............................................................. 233
DIM Function ..................................................................... 234
DIVIDE Function ............................................................... 235
DUR Function ..................................................................... 237
DURP Function ................................................................... 238
EFFRAT Function ............................................................... 239
ERF Function ...................................................................... 240
ERFC Function .................................................................... 241
EXP Function ...................................................................... 242
FACT Function ................................................................. 242
FINANCE Function ....................................................... 243
FIND Function ............................................................... 285
FINDC Function ............................................................. 288
FINDW Function ............................................................. 295
FLOOR Function .............................................................. 301
FLOORZ Function ........................................................... 302
FNONCT Function ........................................................... 303
FUZZ Function ............................................................... 305
GAMINV Function ........................................................... 306
GAMMA Function ........................................................... 306
GARKHCLPRC Function .................................................... 307
GARKHPTPRC Function ..................................................... 309
GCD Function ................................................................. 311
GEODIST Function .......................................................... 312
GEOMEAN Function ........................................................ 315
GEOMEANZ Function ....................................................... 316
HARMEAN Function ........................................................ 318
HARMEANZ Function ....................................................... 319
HBOUND Function .......................................................... 320
HMS Function ................................................................. 322
HOLIDAY Function .......................................................... 323
HOUR Function ............................................................... 325
IBESSEL Function ........................................................... 326
INDEX Function .............................................................. 327
INDEXC Function ........................................................... 329
INDEXW Function ........................................................... 330
INPUT Function .............................................................. 334
INPUTC Function ............................................................ 337
INPUTN Function ............................................................ 339
INT Function ................................................................. 340
INTCINDEX Function ........................................................ 341
INTCK Function ............................................................. 344
INTCYCLE Function ......................................................... 350
INTFIT Function ............................................................ 353
INTFMT Function ........................................................... 356
INTGET Function ............................................................ 359
INTINDEX Function ........................................................ 361
INTTEST Function .......................................................... 367
INTNX Function .............................................................. 370
INTRR Function ............................................................. 377
INTSEAS Function .......................................................... 378
INTSHIFT Function ........................................................ 382
INTTEST Function .......................................................... 384
INTZ Function ............................................................... 385
IPMT Function ............................................................... 387
IQR Function ................................................................. 388
IRR Function ................................................................. 389
JBESSEL Function ........................................................... 390
JULDATE Function .......................................................... 390
JULDATE7 Function .......................................................... 391
KURTOSIS Function ......................................................... 392
LAG Function ............................................................... 393
LARGEST Function .......................................................... 400
LBOUND Function .......................................................... 402
LCM Function ............................................................... 403
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCOMB Function</td>
<td>404</td>
</tr>
<tr>
<td>LEFT Function</td>
<td>405</td>
</tr>
<tr>
<td>LENGTH Function</td>
<td>406</td>
</tr>
<tr>
<td>LENGTHC Function</td>
<td>407</td>
</tr>
<tr>
<td>LENGTHM Function</td>
<td>408</td>
</tr>
<tr>
<td>LENGTHN Function</td>
<td>410</td>
</tr>
<tr>
<td>LFACT Function</td>
<td>411</td>
</tr>
<tr>
<td>LGamma Function</td>
<td>412</td>
</tr>
<tr>
<td>LOG Function</td>
<td>413</td>
</tr>
<tr>
<td>LOG1PX Function</td>
<td>413</td>
</tr>
<tr>
<td>LOG10 Function</td>
<td>414</td>
</tr>
<tr>
<td>LOG2 Function</td>
<td>415</td>
</tr>
<tr>
<td>LOGBETA Function</td>
<td>415</td>
</tr>
<tr>
<td>LOGCDF Function</td>
<td>416</td>
</tr>
<tr>
<td>LOGPDF Function</td>
<td>418</td>
</tr>
<tr>
<td>LOGSDSF Function</td>
<td>420</td>
</tr>
<tr>
<td>LOWCASE Function</td>
<td>423</td>
</tr>
<tr>
<td>LPERM Function</td>
<td>423</td>
</tr>
<tr>
<td>MAD Function</td>
<td>424</td>
</tr>
<tr>
<td>MARGRCLPRC Function</td>
<td>425</td>
</tr>
<tr>
<td>MARGRPPTPRC Function</td>
<td>427</td>
</tr>
<tr>
<td>MAX Function</td>
<td>429</td>
</tr>
<tr>
<td>MD5 Function</td>
<td>430</td>
</tr>
<tr>
<td>MDY Function</td>
<td>432</td>
</tr>
<tr>
<td>MEAN Function</td>
<td>433</td>
</tr>
<tr>
<td>MEDIAN Function</td>
<td>434</td>
</tr>
<tr>
<td>MIN Function</td>
<td>434</td>
</tr>
<tr>
<td>MINUTE Function</td>
<td>435</td>
</tr>
<tr>
<td>MISSING Function</td>
<td>436</td>
</tr>
<tr>
<td>MOD Function</td>
<td>438</td>
</tr>
<tr>
<td>MODZ Function</td>
<td>440</td>
</tr>
<tr>
<td>MONTH Function</td>
<td>442</td>
</tr>
<tr>
<td>MORT Function</td>
<td>443</td>
</tr>
<tr>
<td>N Function</td>
<td>444</td>
</tr>
<tr>
<td>NETPV Function</td>
<td>445</td>
</tr>
<tr>
<td>NMISS Function</td>
<td>446</td>
</tr>
<tr>
<td>NOMRATE Function</td>
<td>447</td>
</tr>
<tr>
<td>NOTALNUM Function</td>
<td>448</td>
</tr>
<tr>
<td>NOTALPHA Function</td>
<td>450</td>
</tr>
<tr>
<td>NOTCNTRL Function</td>
<td>452</td>
</tr>
<tr>
<td>NOTDIGIT Function</td>
<td>453</td>
</tr>
<tr>
<td>NOTFIRST Function</td>
<td>455</td>
</tr>
<tr>
<td>NOTGRAPH Function</td>
<td>457</td>
</tr>
<tr>
<td>NOTLOWER Function</td>
<td>459</td>
</tr>
<tr>
<td>NOTNAME Function</td>
<td>461</td>
</tr>
<tr>
<td>NOTPRINT Function</td>
<td>462</td>
</tr>
<tr>
<td>NOTPUNCT Function</td>
<td>464</td>
</tr>
<tr>
<td>NOTSPACE Function</td>
<td>466</td>
</tr>
<tr>
<td>NOTUPPER Function</td>
<td>468</td>
</tr>
<tr>
<td>NOTXDIGIT Function</td>
<td>470</td>
</tr>
<tr>
<td>NPV Function</td>
<td>472</td>
</tr>
<tr>
<td>NWKDOM Function</td>
<td>473</td>
</tr>
<tr>
<td>ORDINAL Function</td>
<td>475</td>
</tr>
<tr>
<td>PCTL Function</td>
<td>476</td>
</tr>
<tr>
<td>PDF Function</td>
<td>477</td>
</tr>
<tr>
<td>PERM Function</td>
<td>497</td>
</tr>
</tbody>
</table>
TAN Function .................................................. 624
TIME Function .............................................. 625
TIMEPART Function ......................................... 626
TIMEVALUE Function ....................................... 626
TINV Function ............................................... 628
 TNONCT Function ........................................... 629
TODAY Function ............................................. 630
TRANSLATE Function ....................................... 632
TRANSTRN Function ....................................... 633
TRANWRD Function ......................................... 635
TRIGAMMA Function ....................................... 639
TRIM Function .............................................. 640
TRIMN Function ............................................. 642
TRUNC Function ............................................ 643
UNIFORM Function ......................................... 644
UPCASE Function ........................................... 644
USS Function ............................................... 645
UUIDGEN Function ......................................... 646
VAR Function ............................................... 646
VARRAY Function ........................................... 647
VARRAYX Function ......................................... 648
VERIFY Function ............................................. 649
VFORMAT Function ......................................... 651
VFORMATD Function ....................................... 652
VFORMATDX Function ...................................... 653
VFORMATN Function ....................................... 654
VFORMATNX Function ...................................... 655
VFORMATW Function ....................................... 656
VFORMATWX Function ...................................... 657
VFORMATX Function ....................................... 658
VARRAY Function .......................................... 659
VARRAYX Function ......................................... 660
VINFORMAT Function ....................................... 662
VINFORMATD Function ..................................... 663
VINFORMATDX Function ................................... 664
VINFORMATN Function ..................................... 665
VINFORMATNX Function ................................... 666
VINFORMATW Function ................................... 667
VINFORMATWX Function ................................... 668
VINFORMATX Function ..................................... 669
VLABEL Function .......................................... 670
VLABELX Function .......................................... 671
VLENGTH Function ......................................... 672
VLENGTHX Function ......................................... 674
VNAME Function ........................................... 675
VNAMEX Function ........................................... 676
VTYPE Function ............................................. 677
VTYPEX Function ........................................... 678
VVALUE Function .......................................... 679
VVALUEX Function .......................................... 681
WEEK Function ............................................. 682
WEEKDAY Function ......................................... 687
WHICHCH Function .......................................... 688
WHICHTH Option ............................................ 689
YEAR Function .............................................. 690
YIELD Function ............................................. 691
# SAS Functions and CALL Routines for CAS and SAS by Category

<table>
<thead>
<tr>
<th>Category</th>
<th>Language Elements</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arithmetic</strong></td>
<td>DIVIDE Function (p. 235)</td>
<td>Returns the result of a division that handles special missing values for ODS output.</td>
</tr>
<tr>
<td><strong>Array</strong></td>
<td>DIM Function (p. 234)</td>
<td>Returns the number of elements in an array.</td>
</tr>
<tr>
<td></td>
<td>HBOUND Function (p. 320)</td>
<td>Returns the upper bound of an array.</td>
</tr>
<tr>
<td></td>
<td>LBOUND Function (p. 402)</td>
<td>Returns the lower bound of an array.</td>
</tr>
<tr>
<td><strong>Bitwise Logical Operations</strong></td>
<td>BAND Function (p. 110)</td>
<td>Returns the bitwise logical AND of two arguments.</td>
</tr>
<tr>
<td></td>
<td>BLSHIFT Function (p. 120)</td>
<td>Returns the bitwise logical left shift of two arguments.</td>
</tr>
<tr>
<td></td>
<td>BNOT Function (p. 121)</td>
<td>Returns the bitwise logical NOT of an argument.</td>
</tr>
<tr>
<td></td>
<td>BOR Function (p. 121)</td>
<td>Returns the bitwise logical OR of two arguments.</td>
</tr>
<tr>
<td></td>
<td>BRSHIFT Function (p. 122)</td>
<td>Returns the bitwise logical right shift of two arguments.</td>
</tr>
<tr>
<td></td>
<td>BXOR Function (p. 123)</td>
<td>Returns the bitwise logical EXCLUSIVE OR of two arguments.</td>
</tr>
<tr>
<td><strong>Character</strong></td>
<td>ANYALNUM Function (p. 80)</td>
<td>Searches a character string for an alphanumeric character, and returns the first position at which the character is found.</td>
</tr>
<tr>
<td></td>
<td>ANYALPHA Function (p. 82)</td>
<td>Searches a character string for an alphabetic character, and returns the first position at which the character is found.</td>
</tr>
<tr>
<td></td>
<td>ANYCNTRL Function (p. 84)</td>
<td>Searches a character string for a control character, and returns the first position at which that character is found.</td>
</tr>
<tr>
<td></td>
<td>ANYDIGIT Function (p. 86)</td>
<td>Searches a character string for a digit, and returns the first position at which the digit is found.</td>
</tr>
<tr>
<td></td>
<td>ANYFIRST Function (p. 87)</td>
<td>Searches a character string for a character that is valid as the first character in a SAS variable name under VALIDVARNAME=V7, and returns the first position at which that character is found.</td>
</tr>
<tr>
<td></td>
<td>ANYGRAPH Function (p. 89)</td>
<td>Searches a character string for a graphical character, and returns the first position at which that character is found.</td>
</tr>
<tr>
<td></td>
<td>ANYLOWER Function (p. 91)</td>
<td>Searches a character string for a lowercase letter, and returns the first position at which the letter is found.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ANYNAME Function (p. 93)</td>
<td>Searches a character string for a character that is valid in a SAS variable name under VALIDVARNAME=V7, and returns the first position at which that character is found.</td>
<td></td>
</tr>
<tr>
<td>ANYPRINT Function (p. 95)</td>
<td>Searches a character string for a printable character, and returns the first position at which that character is found.</td>
<td></td>
</tr>
<tr>
<td>ANYPUNCT Function (p. 97)</td>
<td>Searches a character string for a punctuation character, and returns the first position at which that character is found.</td>
<td></td>
</tr>
<tr>
<td>ANYSPACE Function (p. 99)</td>
<td>Searches a character string for a Whitespace character (blank, horizontal and vertical tab, carriage return, line feed, and form feed), and returns the first position at which that character is found.</td>
<td></td>
</tr>
<tr>
<td>ANYUPPER Function (p. 101)</td>
<td>Searches a character string for an uppercase letter, and returns the first position at which the letter is found.</td>
<td></td>
</tr>
<tr>
<td>ANYXDIGIT Function (p. 102)</td>
<td>Searches a character string for a hexadecimal character that represents a digit, and returns the first position at which that character is found.</td>
<td></td>
</tr>
<tr>
<td>BYTE Function (p. 123)</td>
<td>Returns one character in the ASCII collating sequence.</td>
<td></td>
</tr>
<tr>
<td>CALL MISSING Routine (p. 126)</td>
<td>Assigns missing values to the specified character or numeric variables.</td>
<td></td>
</tr>
<tr>
<td>CALL SCAN Routine (p. 128)</td>
<td>Returns the position and length of the nth word from a character string.</td>
<td></td>
</tr>
<tr>
<td>CAT Function (p. 143)</td>
<td>Does not remove leading or trailing blanks, and returns a concatenated character string.</td>
<td></td>
</tr>
<tr>
<td>CATS Function (p. 145)</td>
<td>Removes leading and trailing blanks, and returns a concatenated character string.</td>
<td></td>
</tr>
<tr>
<td>CATT Function (p. 147)</td>
<td>Removes trailing blanks, and returns a concatenated character string.</td>
<td></td>
</tr>
<tr>
<td>CATX Function (p. 149)</td>
<td>Removes leading and trailing blanks, inserts delimiters, and returns a concatenated character string.</td>
<td></td>
</tr>
<tr>
<td>CHOOSEC Function (p. 173)</td>
<td>Returns a character value that represents the results of choosing from a list of arguments.</td>
<td></td>
</tr>
<tr>
<td>CHOOSEN Function (p. 175)</td>
<td>Returns a numeric value that represents the results of choosing from a list of arguments.</td>
<td></td>
</tr>
<tr>
<td>COALESCEC Function (p. 178)</td>
<td>Returns the first nonmissing value from a list of character arguments.</td>
<td></td>
</tr>
<tr>
<td>COLLATE Function (p. 179)</td>
<td>Returns a character string in the ASCII collating sequence.</td>
<td></td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>COMPARE Function</td>
<td>(p. 184)</td>
<td>Returns the position of the leftmost character by which two strings differ, or returns 0 if there is no difference.</td>
</tr>
<tr>
<td>COMPBL Function</td>
<td>(p. 187)</td>
<td>Removes multiple blanks from a character string.</td>
</tr>
<tr>
<td>COMPRESS Function</td>
<td>(p. 191)</td>
<td>Returns a character string with specified characters removed from the original string.</td>
</tr>
<tr>
<td>COUNT Function</td>
<td>(p. 204)</td>
<td>Counts the number of times that a specified substring appears within a character string.</td>
</tr>
<tr>
<td>COUNTC Function</td>
<td>(p. 206)</td>
<td>Counts the number of characters in a string that appear or do not appear in a list of characters.</td>
</tr>
<tr>
<td>COUNTW Function</td>
<td>(p. 209)</td>
<td>Counts the number of words in a character string.</td>
</tr>
<tr>
<td>DEQUOTE Function</td>
<td>(p. 224)</td>
<td>Removes matching quotation marks from a character string that begins with a quotation mark, and deletes all characters to the right of the closing quotation mark.</td>
</tr>
<tr>
<td>FIND Function</td>
<td>(p. 285)</td>
<td>Searches for a specific substring of characters within a character string.</td>
</tr>
<tr>
<td>FINDC Function</td>
<td>(p. 288)</td>
<td>Searches a string for any character in a list of characters.</td>
</tr>
<tr>
<td>FINDW Function</td>
<td>(p. 295)</td>
<td>Returns the character position of a word in a string, or returns the number of the word in a string.</td>
</tr>
<tr>
<td>INDEX Function</td>
<td>(p. 327)</td>
<td>Searches a character expression for a string of characters, and returns the position of the string's first character for the first occurrence of the string.</td>
</tr>
<tr>
<td>INDEXC Function</td>
<td>(p. 329)</td>
<td>Searches a character expression for any of the specified characters, and returns the position of that character.</td>
</tr>
<tr>
<td>INDEXW Function</td>
<td>(p. 330)</td>
<td>Searches a character expression for a string that is specified as a word, and returns the position of the first character in the word.</td>
</tr>
<tr>
<td>LEFT Function</td>
<td>(p. 405)</td>
<td>Left-aligns a character string.</td>
</tr>
<tr>
<td>LENGTH Function</td>
<td>(p. 406)</td>
<td>Returns the length of a non-blank character string, excluding trailing blanks, and returns 1 for a blank character string.</td>
</tr>
<tr>
<td>LENGTHC Function</td>
<td>(p. 407)</td>
<td>Returns the length of a character string, including trailing blanks.</td>
</tr>
<tr>
<td>LENGTHM Function</td>
<td>(p. 408)</td>
<td>Returns the amount of memory (in bytes) that is allocated for a character string.</td>
</tr>
<tr>
<td>LENGTHN Function</td>
<td>(p. 410)</td>
<td>Returns the length of a character string, excluding trailing blanks.</td>
</tr>
<tr>
<td>LOWCASE Function</td>
<td>(p. 423)</td>
<td>Converts all letters in an argument to lowercase.</td>
</tr>
<tr>
<td>MD5 Function</td>
<td>(p. 430)</td>
<td>Returns the result of the message digest of a specified string.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td><strong>MISSING Function (p. 436)</strong></td>
<td>Returns a numeric result that indicates whether the argument contains a missing value.</td>
</tr>
<tr>
<td></td>
<td><strong>NOTALNUM Function (p. 448)</strong></td>
<td>Searches a character string for a non-alphanumeric character, and returns the first position at which the character is found.</td>
</tr>
<tr>
<td></td>
<td><strong>NOTALPHA Function (p. 450)</strong></td>
<td>Searches a character string for a nonalphabetic character, and returns the first position at which the character is found.</td>
</tr>
<tr>
<td></td>
<td><strong>NOTCNTRL Function (p. 452)</strong></td>
<td>Searches a character string for a character that is not a control character, and returns the first position at which that character is found.</td>
</tr>
<tr>
<td></td>
<td><strong>NOTDIGIT Function (p. 453)</strong></td>
<td>Searches a character string for any character that is not a digit, and returns the first position at which that character is found.</td>
</tr>
<tr>
<td></td>
<td><strong>NOTFIRST Function (p. 455)</strong></td>
<td>Searches a character string for an invalid first character in a SAS variable name under VALIDVARNAME=V7, and returns the first position at which that character is found.</td>
</tr>
<tr>
<td></td>
<td><strong>NOTGRAPH Function (p. 457)</strong></td>
<td>Searches a character string for a non-graphical character, and returns the first position at which that character is found.</td>
</tr>
<tr>
<td></td>
<td><strong>NOTLOWER Function (p. 459)</strong></td>
<td>Searches a character string for a character that is not a lowercase letter, and returns the first position at which that character is found.</td>
</tr>
<tr>
<td></td>
<td><strong>NOTNAME Function (p. 461)</strong></td>
<td>Searches a character string for an invalid character in a SAS variable name under VALIDVARNAME=V7, and returns the first position at which that character is found.</td>
</tr>
<tr>
<td></td>
<td><strong>NOTPRINT Function (p. 462)</strong></td>
<td>Searches a character string for a nonprintable character, and returns the first position at which that character is found.</td>
</tr>
<tr>
<td></td>
<td><strong>NOTPUNCT Function (p. 464)</strong></td>
<td>Searches a character string for a character that is not a punctuation character, and returns the first position at which that character is found.</td>
</tr>
<tr>
<td></td>
<td><strong>NOTSPACE Function (p. 466)</strong></td>
<td>Searches a character string for a character that is not a whitespace character (blank, horizontal and vertical tab, carriage return, line feed, and form feed), and returns the first position at which that character is found.</td>
</tr>
<tr>
<td></td>
<td><strong>NOTUPPER Function (p. 468)</strong></td>
<td>Searches a character string for a character that is not an uppercase letter, and returns the first position at which that character is found.</td>
</tr>
<tr>
<td></td>
<td><strong>NOTXDIGIT Function (p. 470)</strong></td>
<td>Searches a character string for a character that is not a hexadecimal character, and returns the first position at which that character is found.</td>
</tr>
<tr>
<td></td>
<td><strong>QUOTE Function (p. 553)</strong></td>
<td>Adds double quotation marks to a character value.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>RANK Function (p. 568)</td>
<td></td>
<td>Returns the position of a character in the ASCII collating sequence.</td>
</tr>
<tr>
<td>REPEAT Function (p. 569)</td>
<td></td>
<td>Returns a character value that consists of the first argument repeated n+1 times.</td>
</tr>
<tr>
<td>REVERSE Function (p. 570)</td>
<td></td>
<td>Reverses a character string.</td>
</tr>
<tr>
<td>RIGHT Function (p. 571)</td>
<td></td>
<td>Right aligns a character expression.</td>
</tr>
<tr>
<td>SCAN Function (p. 586)</td>
<td></td>
<td>Returns the nth word from a character string.</td>
</tr>
<tr>
<td>SHA256HEX Function (p. 602)</td>
<td></td>
<td>Returns the result of the message digest of a specified string and converts the string to hexadecimal representation.</td>
</tr>
<tr>
<td>SHA256HMACHEX Function (p. 604)</td>
<td></td>
<td>Returns the result of the message digest of a specified string by using the Hash-based Message Authentication (HMAC) algorithm.</td>
</tr>
<tr>
<td>STRIP Function (p. 613)</td>
<td></td>
<td>Returns a character string with all leading and trailing blanks removed.</td>
</tr>
<tr>
<td>SUBSTR (left of =) Function (p. 615)</td>
<td></td>
<td>Replaces character value contents.</td>
</tr>
<tr>
<td>SUBSTR (right of =) Function (p. 617)</td>
<td></td>
<td>Extracts a substring from an argument.</td>
</tr>
<tr>
<td>SUBSTRN Function (p. 618)</td>
<td></td>
<td>Returns a substring, allowing a result with a length of zero.</td>
</tr>
<tr>
<td>TRANSLATE Function (p. 632)</td>
<td></td>
<td>Replaces specific characters in a character expression.</td>
</tr>
<tr>
<td>TRANSTRN Function (p. 633)</td>
<td></td>
<td>Replaces or removes all occurrences of a substring in a character string.</td>
</tr>
<tr>
<td>TRANWRD Function (p. 635)</td>
<td></td>
<td>Replaces all occurrences of a substring in a character string.</td>
</tr>
<tr>
<td>TRIM Function (p. 640)</td>
<td></td>
<td>Removes trailing blanks from a character string, and returns one blank if the string is missing.</td>
</tr>
<tr>
<td>TRIMN Function (p. 642)</td>
<td></td>
<td>Removes trailing blanks from character expressions, and returns a string with a length of zero if the expression is missing.</td>
</tr>
<tr>
<td>UPCASE Function (p. 644)</td>
<td></td>
<td>Converts all letters in an argument to uppercase.</td>
</tr>
<tr>
<td>VERIFY Function (p. 649)</td>
<td></td>
<td>Returns the position of the first character in a string that is not in any of several other strings.</td>
</tr>
<tr>
<td>Character String Matching</td>
<td>CALL PRXFREX Routine (p. 127)</td>
<td>Frees memory that was allocated for a Perl regular expression.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Category</td>
<td>PRXCHANGE Function (p.</td>
<td>Performs a pattern-matching replacement.</td>
</tr>
<tr>
<td></td>
<td>527)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PRXMATCH Function (p.</td>
<td>Searches for a pattern match and returns the position at which the</td>
</tr>
<tr>
<td></td>
<td>533)</td>
<td>pattern is found.</td>
</tr>
<tr>
<td></td>
<td>PRXPARSE Function (p.</td>
<td>Compiles a Perl regular expression (PRX) that can be used for</td>
</tr>
<tr>
<td></td>
<td>538)</td>
<td>pattern matching of a character value.</td>
</tr>
<tr>
<td></td>
<td>PRXPOSN Function (p.</td>
<td>Returns a character string that contains the value for a capture</td>
</tr>
<tr>
<td></td>
<td>541)</td>
<td>buffer.</td>
</tr>
<tr>
<td>Combinatorial</td>
<td>COMB Function (p. 183)</td>
<td>Computes the number of combinations of n elements taken r at a time.</td>
</tr>
<tr>
<td></td>
<td>LCOMB Function (p. 404)</td>
<td>Computes the logarithm of the COMB function, which is the logarithm of the</td>
</tr>
<tr>
<td></td>
<td>LFACT Function (p. 411)</td>
<td>logarithm of the number of combinations of n objects taken r at a time.</td>
</tr>
<tr>
<td></td>
<td>LPERM Function (p. 423)</td>
<td>Computes the logarithm of the PERM function, which is the logarithm of the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>number of permutations of n objects, with the option of including r number</td>
</tr>
<tr>
<td></td>
<td></td>
<td>of elements.</td>
</tr>
<tr>
<td></td>
<td>PERM Function (p. 497)</td>
<td>Computes the number of permutations of n items that are taken r at a time.</td>
</tr>
<tr>
<td>Date and Time</td>
<td>DATDIF Function (p. 217)</td>
<td>Returns the number of days between two dates after computing the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>difference between the dates according to specified day count conventions.</td>
</tr>
<tr>
<td></td>
<td>DATE Function (p. 220)</td>
<td>Returns the current date as a SAS date value.</td>
</tr>
<tr>
<td></td>
<td>DATEJUL Function (p. 221)</td>
<td>Converts a Julian date to a SAS date value.</td>
</tr>
<tr>
<td></td>
<td>DATEPART Function (p. 221)</td>
<td>Extracts the date from a SAS datetime value.</td>
</tr>
<tr>
<td></td>
<td>DATETIME Function (p. 222)</td>
<td>Returns the current date and time of day as a SAS datetime value.</td>
</tr>
<tr>
<td></td>
<td>DAY Function (p. 223)</td>
<td>Returns the day of the month from a SAS date value.</td>
</tr>
<tr>
<td></td>
<td>DHMS Function (p. 230)</td>
<td>Returns a SAS datetime value from date, hour, minute, and second values.</td>
</tr>
<tr>
<td></td>
<td>HMS Function (p. 322)</td>
<td>Returns a SAS time value from hour, minute, and second values.</td>
</tr>
<tr>
<td></td>
<td>HOLIDAY Function (p. 323)</td>
<td>Returns a SAS date value of a specified holiday for a specified year.</td>
</tr>
<tr>
<td></td>
<td>HOUR Function (p. 325)</td>
<td>Returns the hour from a SAS time or datetime value.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>INTCINDEX Function (p. 341)</td>
<td>Returns the cycle index when a date, time, or datetime interval and value are specified.</td>
<td></td>
</tr>
<tr>
<td>INTCK Function (p. 344)</td>
<td>Returns the number of interval boundaries of a given kind that lie between two dates, times, or datetime values.</td>
<td></td>
</tr>
<tr>
<td>INTCYCLE Function (p. 350)</td>
<td>Returns the date, time, or datetime interval at the next higher seasonal cycle when a date, time, or datetime interval is specified.</td>
<td></td>
</tr>
<tr>
<td>INTFIT Function (p. 353)</td>
<td>Returns a time interval that is aligned between two dates.</td>
<td></td>
</tr>
<tr>
<td>INTFMT Function (p. 356)</td>
<td>Returns a recommended SAS format when a date, time, or datetime interval is specified.</td>
<td></td>
</tr>
<tr>
<td>INTGET Function (p. 359)</td>
<td>Returns a time interval based on three date or datetime values.</td>
<td></td>
</tr>
<tr>
<td>INTINDEX Function (p. 361)</td>
<td>Returns the seasonal index when a date, time, or datetime interval and value are specified.</td>
<td></td>
</tr>
<tr>
<td>INTNEST Function (p. 367)</td>
<td>Calculates the number of whole periods of the smaller interval that will fit into the period of the larger interval.</td>
<td></td>
</tr>
<tr>
<td>INTNX Function (p. 370)</td>
<td>Increments a date, time, or datetime value by a given time interval, and returns a date, time, or datetime value.</td>
<td></td>
</tr>
<tr>
<td>INTSEAS Function (p. 378)</td>
<td>Returns the length of the seasonal cycle when a date, time, or datetime interval is specified.</td>
<td></td>
</tr>
<tr>
<td>INTSHIFT Function (p. 382)</td>
<td>Returns the shift interval that corresponds to the base interval.</td>
<td></td>
</tr>
<tr>
<td>INTTEST Function (p. 384)</td>
<td>Returns 1 if a time interval is valid, and returns 0 if a time interval is invalid.</td>
<td></td>
</tr>
<tr>
<td>JULDATE Function (p. 390)</td>
<td>Returns the Julian date from a SAS date value.</td>
<td></td>
</tr>
<tr>
<td>JULDATE7 Function (p. 391)</td>
<td>Returns a seven-digit Julian date from a SAS date value.</td>
<td></td>
</tr>
<tr>
<td>MDY Function (p. 432)</td>
<td>Returns a SAS date value from month, day, and year values.</td>
<td></td>
</tr>
<tr>
<td>MINUTE Function (p. 435)</td>
<td>Returns the minute from a SAS time or datetime value.</td>
<td></td>
</tr>
<tr>
<td>MONTH Function (p. 442)</td>
<td>Returns the month from a SAS date value.</td>
<td></td>
</tr>
<tr>
<td>NWKDOM Function (p. 473)</td>
<td>Returns the date for the nth occurrence of a weekday for the specified month and year.</td>
<td></td>
</tr>
<tr>
<td>QTR Function (p. 548)</td>
<td>Returns the quarter of the year from a SAS date value.</td>
<td></td>
</tr>
<tr>
<td>SECOND Function (p. 600)</td>
<td>Returns the second from a SAS time or datetime value.</td>
<td></td>
</tr>
<tr>
<td>TIME Function (p. 625)</td>
<td>Returns the current time of day as a numeric SAS time value.</td>
<td></td>
</tr>
<tr>
<td>TIMEPART Function (p. 626)</td>
<td>Extracts a time value from a SAS datetime value.</td>
<td></td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Language Elements</td>
<td>TODAY Function (p. 630)</td>
<td>Returns the current date as a numeric SAS date value.</td>
</tr>
<tr>
<td></td>
<td>WEEK Function (p. 682)</td>
<td>Returns the week-number value.</td>
</tr>
<tr>
<td></td>
<td>WEEKDAY Function (p. 687)</td>
<td>From a SAS date value, returns an integer that corresponds to the day of the week.</td>
</tr>
<tr>
<td></td>
<td>YEAR Function (p. 690)</td>
<td>Returns the year from a SAS date value.</td>
</tr>
<tr>
<td></td>
<td>YRDIF Function (p. 692)</td>
<td>Returns the difference in years between two dates according to specified day count conventions; returns a person’s age.</td>
</tr>
<tr>
<td></td>
<td>YYQ Function (p. 695)</td>
<td>Returns a SAS date value from year and quarter year values.</td>
</tr>
<tr>
<td>Descriptive Statistics</td>
<td>CSS Function (p. 213)</td>
<td>Returns the corrected sum of squares.</td>
</tr>
<tr>
<td></td>
<td>CV Function (p. 216)</td>
<td>Returns the coefficient of variation.</td>
</tr>
<tr>
<td></td>
<td>GEOMEAN Function (p. 315)</td>
<td>Returns the geometric mean.</td>
</tr>
<tr>
<td></td>
<td>GEOMEANZ Function (p. 316)</td>
<td>Returns the geometric mean, using zero fuzzing.</td>
</tr>
<tr>
<td></td>
<td>HARMEAN Function (p. 318)</td>
<td>Returns the harmonic mean.</td>
</tr>
<tr>
<td></td>
<td>HARMEANZ Function (p. 319)</td>
<td>Returns the harmonic mean, using zero fuzzing.</td>
</tr>
<tr>
<td></td>
<td>IQR Function (p. 388)</td>
<td>Returns the interquartile range.</td>
</tr>
<tr>
<td></td>
<td>KURTOSIS Function (p. 392)</td>
<td>Returns the kurtosis.</td>
</tr>
<tr>
<td></td>
<td>LARGEST Function (p. 400)</td>
<td>Returns the kth largest nonmissing value.</td>
</tr>
<tr>
<td></td>
<td>MAD Function (p. 424)</td>
<td>Returns the median absolute deviation from the median.</td>
</tr>
<tr>
<td></td>
<td>MAX Function (p. 429)</td>
<td>Returns the largest value.</td>
</tr>
<tr>
<td></td>
<td>MEAN Function (p. 433)</td>
<td>Returns the arithmetic mean (average).</td>
</tr>
<tr>
<td></td>
<td>MEDIAN Function (p. 434)</td>
<td>Returns the median value.</td>
</tr>
<tr>
<td></td>
<td>MIN Function (p. 434)</td>
<td>Returns the smallest value.</td>
</tr>
<tr>
<td></td>
<td>MISSING Function (p. 436)</td>
<td>Returns a numeric result that indicates whether the argument contains a missing value.</td>
</tr>
<tr>
<td></td>
<td>N Function (p. 444)</td>
<td>Returns the number of nonmissing numeric values.</td>
</tr>
<tr>
<td></td>
<td>NMISS Function (p. 446)</td>
<td>Returns the number of missing numeric values.</td>
</tr>
<tr>
<td></td>
<td>ORDINAL Function (p. 475)</td>
<td>Returns the kth smallest of the missing and nonmissing values.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
</tbody>
</table>
## Category | Language Elements | Description
--- | --- | ---
DUR Function (p. 237) | Returns the modified duration for an enumerated cash flow. | 
DURP Function (p. 238) | Returns the modified duration for a periodic cash flow stream, such as a bond. | 
EFFRATE Function (p. 239) | Returns the effective annual interest rate. | 
FINANCE Function (p. 243) | Computes financial calculations such as depreciation, maturation, accrued interest, net present value, periodic savings, and internal rates of return. | 
GARKHCLPRC Function (p. 307) | Calculates call prices for European options on stocks, based on the Garman-Kohlhagen model. | 
GARKHPTPRC Function (p. 309) | Calculates put prices for European options on stocks, based on the Garman-Kohlhagen model. | 
INTRR Function (p. 377) | Returns the internal rate of return as a fraction. | 
IPMT Function (p. 387) | Returns the interest payment for a given period for a constant payment loan or the periodic savings for a future balance. | 
IRR Function (p. 389) | Returns the internal rate of return as a percentage. | 
MARGRCLPRC Function (p. 425) | Calculates call prices for European options on stocks, based on the Margrabe model. | 
MARGRPTPRC Function (p. 427) | Calculates put prices for European options on stocks, based on the Margrabe model. | 
MORT Function (p. 443) | Returns amortization parameters. | 
NETPV Function (p. 445) | Returns the net present value as a percent. | 
NOMRATE Function (p. 447) | Returns the nominal annual interest rate. | 
NPV Function (p. 472) | Returns the net present value with the rate expressed as a percentage. | 
PMT Function (p. 498) | Returns the periodic payment for a constant payment loan or the periodic savings for a future balance. | 
PPMT Function (p. 500) | Returns the principal payment for a given period for a constant payment loan or the periodic savings for a future balance. | 
PVP Function (p. 547) | Returns the present value for a periodic cash flow stream (such as a bond), with repayment of principal at maturity. | 
SAVINGS Function (p. 585) | Returns the balance of a periodic savings by using variable interest rates. | 
TIMEVALUE Function (p. 626) | Returns the equivalent of a reference amount at a base date by using variable interest rates. |
<table>
<thead>
<tr>
<th>Category</th>
<th>Language Elements</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hyperbolic</td>
<td>YIELDP Function</td>
<td>Returns the yield-to-maturity for a periodic cash flow stream, such as a bond.</td>
</tr>
<tr>
<td></td>
<td>ARCOSH Function</td>
<td>Returns the inverse hyperbolic cosine.</td>
</tr>
<tr>
<td></td>
<td>ARSINH Function</td>
<td>Returns the inverse hyperbolic sine.</td>
</tr>
<tr>
<td></td>
<td>ARTANH Function</td>
<td>Returns the inverse hyperbolic tangent.</td>
</tr>
<tr>
<td></td>
<td>COSH Function</td>
<td>Returns the hyperbolic cosine.</td>
</tr>
<tr>
<td></td>
<td>SINH Function</td>
<td>Returns the hyperbolic sine.</td>
</tr>
<tr>
<td>Mathematical</td>
<td>ABS Function</td>
<td>Returns the absolute value.</td>
</tr>
<tr>
<td></td>
<td>AIRY Function</td>
<td>Returns the value of the Airy function.</td>
</tr>
<tr>
<td></td>
<td>BETA Function</td>
<td>Returns the value of the beta function.</td>
</tr>
<tr>
<td></td>
<td>CNONCT Function</td>
<td>Returns the noncentrality parameter from a chi-square distribution.</td>
</tr>
<tr>
<td></td>
<td>COALESCE Function</td>
<td>Returns the first nonmissing value from a list of numeric arguments.</td>
</tr>
<tr>
<td></td>
<td>COMPFUZZ Function</td>
<td>Performs a fuzzy comparison of two numeric values.</td>
</tr>
<tr>
<td></td>
<td>CONSTANT Function</td>
<td>Computes machine and mathematical constants.</td>
</tr>
<tr>
<td></td>
<td>DAIRY Function</td>
<td>Returns the derivative of the AIRY function.</td>
</tr>
<tr>
<td></td>
<td>DEVIANCE Function</td>
<td>Returns the deviance based on a probability distribution.</td>
</tr>
<tr>
<td></td>
<td>DIGAMMA Function</td>
<td>Returns the value of the digamma function.</td>
</tr>
<tr>
<td></td>
<td>ERF Function</td>
<td>Returns the value of the (normal) error function.</td>
</tr>
<tr>
<td></td>
<td>ERFC Function</td>
<td>Returns the value of the complementary (normal) error function.</td>
</tr>
<tr>
<td></td>
<td>EXP Function</td>
<td>Returns the value of the exponential function.</td>
</tr>
<tr>
<td></td>
<td>FACT Function</td>
<td>Computes a factorial.</td>
</tr>
<tr>
<td></td>
<td>FNONCT Function</td>
<td>Returns the value of the noncentrality parameter of an F distribution.</td>
</tr>
<tr>
<td></td>
<td>GAMMA Function</td>
<td>Returns the value of the gamma function.</td>
</tr>
<tr>
<td></td>
<td>GCD Function</td>
<td>Returns the greatest common divisor for one or more integers.</td>
</tr>
<tr>
<td></td>
<td>IBESSEL Function</td>
<td>Returns the value of the modified Bessel function.</td>
</tr>
<tr>
<td></td>
<td>JBESSEL Function</td>
<td>Returns the value of the Bessel function.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Category</td>
<td>LCM Function (p. 403)</td>
<td>Returns the least common multiple.</td>
</tr>
<tr>
<td>Category</td>
<td>LGAMMA Function (p. 412)</td>
<td>Returns the natural logarithm of the Gamma function.</td>
</tr>
<tr>
<td>Category</td>
<td>LOG Function (p. 413)</td>
<td>Returns the natural (base e) logarithm.</td>
</tr>
<tr>
<td>Category</td>
<td>LOG1PX Function (p. 413)</td>
<td>Returns the log of 1 plus the argument.</td>
</tr>
<tr>
<td>Category</td>
<td>LOG10 Function (p. 414)</td>
<td>Returns the logarithm to the base 10.</td>
</tr>
<tr>
<td>Category</td>
<td>LOG2 Function (p. 415)</td>
<td>Returns the logarithm to the base 2.</td>
</tr>
<tr>
<td>Category</td>
<td>LOGBETA Function (p. 415)</td>
<td>Returns the logarithm of the beta function.</td>
</tr>
<tr>
<td>Category</td>
<td>MOD Function (p. 438)</td>
<td>Returns the remainder from the division of the first argument by the second argument, fuzzed to avoid most unexpected floating-point results.</td>
</tr>
<tr>
<td>Category</td>
<td>MODZ Function (p. 440)</td>
<td>Returns the remainder from the division of the first argument by the second argument, using zero fuzzing.</td>
</tr>
<tr>
<td>Category</td>
<td>SIGN Function (p. 601)</td>
<td>Returns the sign of a value.</td>
</tr>
<tr>
<td>Category</td>
<td>SQRT Function (p. 611)</td>
<td>Returns the square root of a value.</td>
</tr>
<tr>
<td>Category</td>
<td>TNONCT Function (p. 629)</td>
<td>Returns the value of the noncentrality parameter from the Student's t distribution.</td>
</tr>
<tr>
<td>Category</td>
<td>TRIGAMMA Function (p. 639)</td>
<td>Returns the value of the trigamma function.</td>
</tr>
<tr>
<td>Probability</td>
<td>CDF Function (p. 153)</td>
<td>Returns a value from a cumulative probability distribution.</td>
</tr>
<tr>
<td>Probability</td>
<td>LOGCDF Function (p. 416)</td>
<td>Returns the logarithm of a left cumulative distribution function.</td>
</tr>
<tr>
<td>Probability</td>
<td>LOGPDF Function (p. 418)</td>
<td>Returns the logarithm of a probability density (mass) function.</td>
</tr>
<tr>
<td>Probability</td>
<td>LOGSDF Function (p. 420)</td>
<td>Returns the logarithm of a survival function.</td>
</tr>
<tr>
<td>Probability</td>
<td>PDF Function (p. 477)</td>
<td>Returns a value from a probability density (mass) distribution.</td>
</tr>
<tr>
<td>Probability</td>
<td>POISSON Function (p. 499)</td>
<td>Returns the probability from a Poisson distribution.</td>
</tr>
<tr>
<td>Probability</td>
<td>PROBBETA Function (p. 501)</td>
<td>Returns the probability from a beta distribution.</td>
</tr>
<tr>
<td>Probability</td>
<td>PROBBNML Function (p. 502)</td>
<td>Returns the probability from a binomial distribution.</td>
</tr>
<tr>
<td>Probability</td>
<td>PROBBNRM Function (p. 504)</td>
<td>Returns a probability from a bivariate normal distribution.</td>
</tr>
<tr>
<td>Probability</td>
<td>PROBCHI Function (p. 505)</td>
<td>Returns the probability from a chi-square distribution.</td>
</tr>
<tr>
<td>Probability</td>
<td>PROBF Function (p. 506)</td>
<td>Returns the probability from an F distribution.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PROB</td>
<td>PROBGAM Function (p. 507)</td>
<td>Returns the probability from a gamma distribution.</td>
</tr>
<tr>
<td></td>
<td>PROBHYPR Function (p. 508)</td>
<td>Returns the probability from a hypergeometric distribution.</td>
</tr>
<tr>
<td></td>
<td>PROBMC Function (p. 511)</td>
<td>Returns a probability or a quantile from various distributions for multiple comparisons of means.</td>
</tr>
<tr>
<td></td>
<td>PROBNEGB Function (p. 524)</td>
<td>Returns the probability from a negative binomial distribution.</td>
</tr>
<tr>
<td></td>
<td>PROBNORM Function (p. 525)</td>
<td>Returns the probability from the standard normal distribution.</td>
</tr>
<tr>
<td></td>
<td>PROBT Function (p. 526)</td>
<td>Returns the probability from a t distribution.</td>
</tr>
<tr>
<td></td>
<td>SDF Function (p. 596)</td>
<td>Returns a survival function.</td>
</tr>
<tr>
<td>Quantile</td>
<td>BETAINV Function (p. 112)</td>
<td>Returns a quantile from the beta distribution.</td>
</tr>
<tr>
<td></td>
<td>GAMINV Function (p. 306)</td>
<td>Returns a quantile from the gamma distribution.</td>
</tr>
<tr>
<td></td>
<td>PROBIT Function (p. 510)</td>
<td>Returns a quantile from the standard normal distribution.</td>
</tr>
<tr>
<td></td>
<td>QUANTILE Function (p. 549)</td>
<td>Returns the quantile from a distribution when you specify the left probability (CDF).</td>
</tr>
<tr>
<td></td>
<td>TINV Function (p. 628)</td>
<td>Returns a quantile from the t distribution.</td>
</tr>
<tr>
<td>Random Number</td>
<td>CALL STREAMINIT Routine (p. 139)</td>
<td>Specifies a seed value to use for subsequent random number generation by the RAND function.</td>
</tr>
<tr>
<td></td>
<td>RAND Function (p. 555)</td>
<td>Generates random numbers from a distribution that you specify.</td>
</tr>
<tr>
<td></td>
<td>UNIFORM Function (p. 644)</td>
<td>Returns a random variate from a uniform distribution.</td>
</tr>
<tr>
<td>Search</td>
<td>WHICHC Function (p. 688)</td>
<td>Searches for a character value that is equal to the first argument, and returns the index of the first matching value.</td>
</tr>
<tr>
<td></td>
<td>WHICHN Function (p. 689)</td>
<td>Searches for a numeric value that is equal to the first argument, and returns the index of the first matching value.</td>
</tr>
<tr>
<td>Special</td>
<td>CALL SLEEP Routine (p. 138)</td>
<td>For a specified period of time, suspends the execution of a program that invokes this CALL routine.</td>
</tr>
<tr>
<td></td>
<td>DIF Function (p. 231)</td>
<td>Returns differences between an argument and its nth lag.</td>
</tr>
<tr>
<td></td>
<td>INPUT Function (p. 334)</td>
<td>Returns the value that is produced when SAS converts an expression by using the specified informat.</td>
</tr>
<tr>
<td></td>
<td>INPUTC Function (p. 337)</td>
<td>Enables you to specify a character informat at run time.</td>
</tr>
<tr>
<td></td>
<td>INPUTN Function (p. 339)</td>
<td>Enables you to specify a numeric informat at run time.</td>
</tr>
<tr>
<td></td>
<td>LAG Function (p. 393)</td>
<td>Returns values from a queue.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>PUT Function (p. 545)</td>
<td>Returns a value using a specified format.</td>
</tr>
<tr>
<td></td>
<td>SLEEP Function (p. 608)</td>
<td>Suspends the execution of a program that invokes this function for a period of time.</td>
</tr>
<tr>
<td></td>
<td>UUIDGEN Function (p. 646)</td>
<td>Returns the short or binary form of a Universally Unique Identifier (UUID).</td>
</tr>
<tr>
<td>Trigonometric</td>
<td>ARCOS Function (p. 104)</td>
<td>Returns the arccosine.</td>
</tr>
<tr>
<td></td>
<td>ARSIN Function (p. 106)</td>
<td>Returns the arcsine.</td>
</tr>
<tr>
<td></td>
<td>ATAN Function (p. 108)</td>
<td>Returns the arc tangent.</td>
</tr>
<tr>
<td></td>
<td>ATAN2 Function (p. 109)</td>
<td>Returns the arc tangent of the ratio of two numeric variables.</td>
</tr>
<tr>
<td></td>
<td>COS Function (p. 201)</td>
<td>Returns the cosine.</td>
</tr>
<tr>
<td></td>
<td>COT Function (p. 203)</td>
<td>Returns the cotangent.</td>
</tr>
<tr>
<td></td>
<td>CSC Function (p. 212)</td>
<td>Returns the cosecant.</td>
</tr>
<tr>
<td></td>
<td>SEC Function (p. 599)</td>
<td>Returns the secant.</td>
</tr>
<tr>
<td></td>
<td>SIN Function (p. 605)</td>
<td>Returns the sine.</td>
</tr>
<tr>
<td></td>
<td>TAN Function (p. 624)</td>
<td>Returns the tangent.</td>
</tr>
<tr>
<td>Truncation</td>
<td>CEIL Function (p. 171)</td>
<td>Returns the smallest integer that is greater than or equal to the argument, fuzzed to avoid unexpected floating-point results.</td>
</tr>
<tr>
<td></td>
<td>CEILZ Function (p. 172)</td>
<td>Returns the smallest integer that is greater than or equal to the argument, using zero fuzzing.</td>
</tr>
<tr>
<td></td>
<td>FLOOR Function (p. 301)</td>
<td>Returns the largest integer that is less than or equal to the argument, fuzzed to avoid unexpected floating-point results.</td>
</tr>
<tr>
<td></td>
<td>FLOORZ Function (p. 302)</td>
<td>Returns the largest integer that is less than or equal to the argument, using zero fuzzing.</td>
</tr>
<tr>
<td></td>
<td>FUZZ Function (p. 305)</td>
<td>Returns the nearest integer if the argument is within 1E−12 of that integer.</td>
</tr>
<tr>
<td></td>
<td>INT Function (p. 340)</td>
<td>Returns the integer value, fuzzed to avoid unexpected floating-point results.</td>
</tr>
<tr>
<td></td>
<td>INTZ Function (p. 385)</td>
<td>Returns the integer portion of the argument, using zero fuzzing.</td>
</tr>
<tr>
<td></td>
<td>ROUND Function (p. 573)</td>
<td>Rounds the first argument to the nearest multiple of the second argument, or to the nearest integer when the second argument is omitted.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>ROUNDDE Function (p. 580)</td>
<td>Rounds the first argument to the nearest multiple of the second argument, and returns an even multiple when the first argument is halfway between the two nearest multiples.</td>
</tr>
<tr>
<td></td>
<td>ROUNDZ Function (p. 582)</td>
<td>Rounds the first argument to the nearest multiple of the second argument, using zero fuzzing.</td>
</tr>
<tr>
<td></td>
<td>TRUNC Function (p. 643)</td>
<td>Truncates a numeric value to a specified number of bytes.</td>
</tr>
<tr>
<td>Variable Control</td>
<td>CALL LABEL Routine (p. 124)</td>
<td>Assigns a variable label to a specified character variable.</td>
</tr>
<tr>
<td></td>
<td>CALL VNAME Routine (p. 141)</td>
<td>Assigns a variable name as the value of a specified variable.</td>
</tr>
<tr>
<td>Variable Information</td>
<td>VARRAY Function (p. 647)</td>
<td>Returns a value that indicates whether the specified name is an array.</td>
</tr>
<tr>
<td></td>
<td>VARRAYX Function (p. 648)</td>
<td>Returns a value that indicates whether the value of the specified argument is an array.</td>
</tr>
<tr>
<td></td>
<td>VFORMAT Function (p. 651)</td>
<td>Returns the format that is associated with the specified variable.</td>
</tr>
<tr>
<td></td>
<td>VFORMATD Function (p. 652)</td>
<td>Returns the decimal value of the format that is associated with the specified variable.</td>
</tr>
<tr>
<td></td>
<td>VFORMATDX Function (p. 653)</td>
<td>Returns the decimal value of the format that is associated with the value of the specified argument.</td>
</tr>
<tr>
<td></td>
<td>VFORMATN Function (p. 654)</td>
<td>Returns the format name that is associated with the specified variable.</td>
</tr>
<tr>
<td></td>
<td>VFORMATNX Function (p. 655)</td>
<td>Returns the format name that is associated with the value of the specified argument.</td>
</tr>
<tr>
<td></td>
<td>VFORMATW Function (p. 656)</td>
<td>Returns the format width that is associated with the specified variable.</td>
</tr>
<tr>
<td></td>
<td>VFORMATWX Function (p. 657)</td>
<td>Returns the format width that is associated with the value of the specified argument.</td>
</tr>
<tr>
<td></td>
<td>VFORMATX Function (p. 658)</td>
<td>Returns the format that is associated with the value of the specified argument.</td>
</tr>
<tr>
<td></td>
<td>VINARRAY Function (p. 659)</td>
<td>Returns a value that indicates whether the specified variable is a member of an array.</td>
</tr>
<tr>
<td></td>
<td>VINARRAYX Function (p. 660)</td>
<td>Returns a value that indicates whether the value of the specified argument is a member of an array.</td>
</tr>
<tr>
<td></td>
<td>VININFORMAT Function (p. 662)</td>
<td>Returns the informat that is associated with the specified variable.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td>VINFORMATD Function (p. 663)</td>
<td>Returns the decimal value of the informat that is associated with the specified variable.</td>
</tr>
<tr>
<td></td>
<td>VINFORMATDX Function (p. 664)</td>
<td>Returns the decimal value of the informat that is associated with the value of the specified variable.</td>
</tr>
<tr>
<td></td>
<td>VINFORMATN Function (p. 665)</td>
<td>Returns the informat name that is associated with the specified variable.</td>
</tr>
<tr>
<td></td>
<td>VINFORMATNX Function (p. 666)</td>
<td>Returns the informat name that is associated with the value of the specified argument.</td>
</tr>
<tr>
<td></td>
<td>VINFORMATW Function (p. 667)</td>
<td>Returns the informat width that is associated with the specified variable.</td>
</tr>
<tr>
<td></td>
<td>VINFORMATWX Function (p. 668)</td>
<td>Returns the informat width that is associated with the value of the specified argument.</td>
</tr>
<tr>
<td></td>
<td>VINFORMATX Function (p. 669)</td>
<td>Returns the informat that is associated with the value of the specified argument.</td>
</tr>
<tr>
<td></td>
<td>VLABEL Function (p. 670)</td>
<td>Returns the label that is associated with the specified variable.</td>
</tr>
<tr>
<td></td>
<td>VLABELX Function (p. 671)</td>
<td>Returns the label that is associated with the value of the specified argument.</td>
</tr>
<tr>
<td></td>
<td>VLENGTH Function (p. 672)</td>
<td>Returns the compile-time (allocated) size of the specified variable.</td>
</tr>
<tr>
<td></td>
<td>VLENGTHX Function (p. 674)</td>
<td>Returns the compile-time (allocated) size for the variable that has a name that is the same as the value of the argument.</td>
</tr>
<tr>
<td></td>
<td>VNAME Function (p. 675)</td>
<td>Returns the name of the specified variable.</td>
</tr>
<tr>
<td></td>
<td>VNAMEX Function (p. 676)</td>
<td>Validates the value of the specified argument as a variable name.</td>
</tr>
<tr>
<td></td>
<td>VTYPE Function (p. 677)</td>
<td>Returns the type (character or numeric) of the specified variable.</td>
</tr>
<tr>
<td></td>
<td>VTYPEX Function (p. 678)</td>
<td>Returns the type (character or numeric) for the value of the specified argument.</td>
</tr>
<tr>
<td></td>
<td>VVALUE Function (p. 679)</td>
<td>Returns the formatted value that is associated with the variable that you specify.</td>
</tr>
<tr>
<td></td>
<td>VVALUEX Function (p. 681)</td>
<td>Returns the formatted value that is associated with the argument that you specify.</td>
</tr>
</tbody>
</table>
Dictionary

ABS Function

Returns the absolute value.

Category: Mathematical

Syntax

\texttt{ABS(\textit{argument})}

Required Argument

\textit{argument} specifies a numeric constant, variable, or expression.

Details

The ABS function returns a nonnegative number that is equal in magnitude to the magnitude of the argument.

Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=abs(2.4);</td>
<td>2.4</td>
</tr>
<tr>
<td>x=abs(-3);</td>
<td>3</td>
</tr>
</tbody>
</table>

AIRY Function

Returns the value of the Airy function.

Category: Mathematical

Syntax

\texttt{AIRY(x)}

Required Argument

\textit{x} specifies a numeric constant, variable, or expression.
Details

The AIRY function returns the value of the Airy function. (See a list of References on page 111.) It is the solution of the differential equation

\[ w'' - xw = 0 \]

with the conditions

\[ w(0) = \frac{1}{\frac{2}{3} \Gamma\left(\frac{2}{3}\right)} \]

and

\[ w'(0) = -\frac{1}{\frac{1}{3} \Gamma\left(\frac{1}{3}\right)} \]

Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=airy(2.0);</td>
<td>0.0349241304</td>
</tr>
<tr>
<td>x=airy(-2.0);</td>
<td>0.2274074282</td>
</tr>
</tbody>
</table>

ANYALNUM Function

Searches a character string for an alphanumeric character, and returns the first position at which the character is found.

Category: Character

Restriction: This function is assigned an I18N Level 1 status. If possible, avoid I18N Level 1 functions if you are using a non-English language. Under certain circumstances, the I18N Level 1 functions might not work correctly with Double-Byte Character Set (DBCS) or Multi-Byte Character Set (MBCS) encodings. For more information, see Internationalization Compatibility.

Syntax

\[ \text{ANYALNUM}(\text{string}<,\text{start}>) \]

Required Argument

\text{string}

specifies a character constant, variable, or expression to search.
Optional Argument

*start*

is an optional integer that specifies the position at which the search should start and the direction in which to search.

Details

The ANYALNUM function searches a string for the first occurrence of any character that is a digit or an uppercase or lowercase letter. If such a character is found, ANYALNUM returns the position in the string of that character. If no such character is found, ANYALNUM returns a value of 0.

If you use only one argument, ANYALNUM begins the search at the beginning of the string. If you use two arguments, the absolute value of the second argument, *start*, specifies the position at which to begin the search. The direction in which to search is determined in the following way:

- If the value of *start* is positive, the search proceeds to the right.
- If the value of *start* is negative, the search proceeds to the left.
- If the value of *start* is less than the negative length of the string, the search begins at the end of the string.

ANYALNUM returns a value of zero when one of the following is true:

- The character that you are searching for is not found.
- The value of *start* is greater than the length of the string.
- The value of *start* = 0.

Comparisons

The ANYALNUM function searches a character string for an alphanumeric character. The NOTALNUM function searches a character string for a non-alphanumeric character.

Examples

Example 1: Scanning a String from Left to Right

The following example uses the ANYALNUM function to search a string from left to right for alphanumeric characters.

```plaintext
data _null_;
  string='Next = Last + 1;';
  j=0;
  do until(j=0);
    j=anyalnum(string, j+1);
    if j=0 then put +3 "That's all";
    else do;
      c=substr(string, j, 1);
      put +3 j= c=;
    end;
  end;
run;
```
SAS writes the following output to the log:

```
j=1 c=N
j=2 c=e
j=3 c=x
j=4 c=t
j=8 c=L
j=9 c=a
j=10 c=s
j=11 c=t
j=15 c=1
That's all
```

**Example 2: Scanning a String from Right to Left**
The following example uses the ANYALNUM function to search a string from right to left for alphanumeric characters.

```
data _null_; string='Next = Last + 1;'; j=999999; do until(j=0);   j=anyalnum(string, 1-j);   if j=0 then put +3 "That's all"; else do;      c=substr(string, j, 1);      put +3 j= c=;   end; end; run;
```

SAS writes the following output to the log:

```
j=15 c=1
j=11 c=t
j=10 c=s
j=9 c=a
j=8 c=L
j=4 c=t
j=3 c=x
j=2 c=e
j=1 c=N
That's all
```

**See Also**

**Functions:**
- “NOTALNUM Function” on page 448

**ANYALPHA Function**

Searches a character string for an alphabetic character, and returns the first position at which the character is found.

**Category:** Character
Restriction: This function is assigned an I18N Level 1 status. If possible, avoid I18N Level 1 functions if you are using a non-English language. Under certain circumstances, the I18N Level 1 functions might not work correctly with Double-Byte Character Set (DBCS) or Multi-Byte Character Set (MBCS) encodings. For more information, see Internationalization Compatibility.

Syntax

ANYALPHA(string <,start>)

Required Argument

string

is the character constant, variable, or expression to search.

Optional Argument

start

is an optional integer that specifies the position at which the search should start and the direction in which to search.

Details

The ANYALPHA function searches a string for the first occurrence of any character that is an uppercase or lowercase letter. If such a character is found, ANYALPHA returns the position in the string of that character. If no such character is found, ANYALPHA returns a value of 0.

If you use only one argument, ANYALPHA begins the search at the beginning of the string. If you use two arguments, the absolute value of the second argument, start, specifies the position at which to begin the search. The direction in which to search is determined in the following way:

• If the value of start is positive, the search proceeds to the right.
• If the value of start is negative, the search proceeds to the left.
• If the value of start is less than the negative length of the string, the search begins at the end of the string.

ANYALPHA returns a value of zero when one of the following is true:

• The character that you are searching for is not found.
• The value of start is greater than the length of the string.
• The value of start = 0.

Comparisons

The ANYALPHA function searches a character string for an alphabetic character. The NOTALPHA function searches a character string for a non-alphabetic character.

Examples

Example 1: Searching a String for Alphabetic Characters

The following example uses the ANYALPHA function to search a string from left to right for alphabetic characters.
Example 2: Identifying Control Characters By Using the ANYALPHA Function

You can execute the following program to show the control characters that are identified by the ANYALPHA function.

```sas
data test;
do dec=0 to 255;
  byte=byte(dec);
  hex=put(dec, hex2.);
  anyalpha=anyalpha(byte);
  output;
end;

proc print data=test;
run;
```

See Also

Functions:

- “NOTALPHA Function” on page 450

ANYCNTRL Function

Searches a character string for a control character, and returns the first position at which that character is found.

Category: Character

Restriction: This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility.
Syntax

ANYCNTRL(*string*, *start*)

**Required Argument**

*string*

is the character constant, variable, or expression to search.

**Optional Argument**

*start*

is an optional integer that specifies the position at which the search should start and the direction in which to search.

**Details**

The ANYCNTRL function searches a string for the first occurrence of a control character. If such a character is found, ANYCNTRL returns the position in the string of that character. If no such character is found, ANYCNTRL returns a value of 0.

If you use only one argument, ANYCNTRL begins the search at the beginning of the string. If you use two arguments, the absolute value of the second argument, *start*, specifies the position at which to begin the search. The direction in which to search is determined in the following way:

- If the value of *start* is positive, the search proceeds to the right.
- If the value of *start* is negative, the search proceeds to the left.
- If the value of *start* is less than the negative length of the string, the search begins at the end of the string.

ANYCNTRL returns a value of zero when one of the following is true:

- The character that you are searching for is not found.
- The value of *start* is greater than the length of the string.
- The value of *start* = 0.

**Comparisons**

The ANYCNTRL function searches a character string for a control character. The NOTCNTRL function searches a character string for a character that is not a control character.

**Example**

You can execute the following program to show the control characters that are identified by the ANYCNTRL function.

```plaintext
data test;
do dec=0 to 255;
drop byte;
byte=byte(dec);
hex=put(dec, hex2.);
anycntrl=anycntrl(byte);
if anycntrl then output;
end;
```
proc print data=test;
run;

See Also

Functions:
- “NOTCNTRL Function” on page 452

ANYDIGIT Function
Searches a character string for a digit, and returns the first position at which the digit is found.

Category: Character
Restriction: This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility.

Syntax
ANYDIGIT(string <,start>)

Required Argument
string
is the character constant, variable, or expression to search.

Optional Argument
start
is an optional integer that specifies the position at which the search should start and the direction in which to search.

Details
The ANYDIGIT function does not depend on the TRANTAB, ENCODING, or LOCALE system options.

The ANYDIGIT function searches a string for the first occurrence of any character that is a digit. If such a character is found, ANYDIGIT returns the position in the string of that character. If no such character is found, ANYDIGIT returns a value of 0.

If you use only one argument, ANYDIGIT begins the search at the beginning of the string. If you use two arguments, the absolute value of the second argument, start, specifies the position at which to begin the search. The direction in which to search is determined in the following way:

- If the value of start is positive, the search proceeds to the right.
- If the value of start is negative, the search proceeds to the left.
- If the value of start is less than the negative length of the string, the search begins at the end of the string.

ANYDIGIT returns a value of zero when one of the following is true:
• The character that you are searching for is not found.
• The value of \texttt{start} is greater than the length of the string.
• The value of \texttt{start} = 0.

\textbf{Comparisons}

The \texttt{ANYDIGIT} function searches a character string for a digit. The \texttt{NOTDIGIT} function searches a character string for any character that is not a digit.

\textbf{Example}

The following example uses the \texttt{ANYDIGIT} function to search for a character that is a digit.

\begin{verbatim}
data _null_;  
  string='Next = _n_ + 12E3;';  
  j=0;  
  do until(j=0);  
    j=anydigit(string, j+1);  
    if j=0 then put +3 "That's all";  
    else do;  
      c=substr(string, j, 1);  
      put +3 j= c=;  
    end;  
  end;  
run;
\end{verbatim}

SAS writes the following output to the log:

\begin{verbatim}
  j=14 c=1
  j=15 c=2
  j=17 c=3
  That's all
\end{verbatim}

\textbf{See Also}

\textbf{Functions:}

• “\texttt{NOTDIGIT Function}” on page 453

\textbf{ANYFIRST Function}

Searches a character string for a character that is valid as the first character in a SAS variable name under \texttt{VALIDVARNAME=V7}, and returns the first position at which that character is found.

\begin{itemize}
  \item \textbf{Category:} Character
  \item \textbf{Restriction:} This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see \texttt{Internationalization Compatibility}.
\end{itemize}

\textbf{Syntax}

\texttt{ANYFIRST(string <,start>)}
**Required Argument**

`string`

is the character constant, variable, or expression to search.

**Optional Argument**

`start`

is an optional integer that specifies the position at which the search should start and the direction in which to search.

**Details**

The ANYFIRST function does not depend on the TRANTAB, ENCODING, or LOCALE system options.

The ANYFIRST function searches a string for the first occurrence of any character that is valid as the first character in a SAS variable name under VALIDVARNAMES=V7. These characters are the underscore (_) and uppercase or lowercase English letters. If such a character is found, ANYFIRST returns the position in the string of that character. If no such character is found, ANYFIRST returns a value of 0.

If you use only one argument, ANYFIRST begins the search at the beginning of the string. If you use two arguments, the absolute value of the second argument, `start`, specifies the position at which to begin the search. The direction in which to search is determined in the following way:

- If the value of `start` is positive, the search proceeds to the right.
- If the value of `start` is negative, the search proceeds to the left.
- If the value of `start` is less than the negative length of the string, the search begins at the end of the string.

ANYFIRST returns a value of zero when one of the following is true:

- The character that you are searching for is not found.
- The value of `start` is greater than the length of the string.
- The value of `start` = 0.

**Comparisons**

The ANYFIRST function searches a string for the first occurrence of any character that is valid as the first character in a SAS variable name under VALIDVARNAMES=V7. The NOTFIRST function searches a string for the first occurrence of any character that is not valid as the first character in a SAS variable name under VALIDVARNAMES=V7.

**Example**

The following example uses the ANYFIRST function to search a string for any character that is valid as the first character in a SAS variable name under VALIDVARNAMES=V7.

```sas
data _null_;
  string='Next = _n_ + 12E3;';
  j=0;
  do until(j=0);
    j=anyfirst(string, j+1);
    if j=0 then put +3 "That's all";
    else do;
```

c=substr(string, j, 1);
put +3 j= cs;
end;
end;
run;

SAS writes the following output to the log:

<table>
<thead>
<tr>
<th>j</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>N</td>
</tr>
<tr>
<td>2</td>
<td>e</td>
</tr>
<tr>
<td>3</td>
<td>x</td>
</tr>
<tr>
<td>4</td>
<td>t</td>
</tr>
<tr>
<td>8</td>
<td>_</td>
</tr>
<tr>
<td>9</td>
<td>n</td>
</tr>
<tr>
<td>10</td>
<td>_</td>
</tr>
<tr>
<td>16</td>
<td>E</td>
</tr>
</tbody>
</table>

That's all

See Also

Functions:
- “NOTFIRST Function” on page 455

ANYGRAPH Function

Searches a character string for a graphical character, and returns the first position at which that character is found.

Category: Character

Restriction: This function is assigned an I18N Level 1 status. If possible, avoid I18N Level 1 functions if you are using a non-English language. Under certain circumstances, the I18N Level 1 functions might not work correctly with Double-Byte Character Set (DBCS) or Multi-Byte Character Set (MBCS) encodings. For more information, see Internationalization Compatibility.

Syntax

ANYGRAPH(string <,start>)

Required Argument

string

is the character constant, variable, or expression to search.

Optional Argument

start

is an optional integer that specifies the position at which the search should start and the direction in which to search.

Details

The ANYGRAPH function searches a string for the first occurrence of a graphical character. A graphical character is defined as any printable character other than white
space. If such a character is found, ANYGRAPH returns the position in the string of that character. If no such character is found, ANYGRAPH returns a value of 0.

If you use only one argument, ANYGRAPH begins the search at the beginning of the string. If you use two arguments, the absolute value of the second argument, start, specifies the position at which to begin the search. The direction in which to search is determined in the following way:

- If the value of start is positive, the search proceeds to the right.
- If the value of start is negative, the search proceeds to the left.
- If the value of start is less than the negative length of the string, the search begins at the end of the string.

ANYGRAPH returns a value of zero when one of the following is true:

- The character that you are searching for is not found.
- The value of start is greater than the length of the string.
- The value of start = 0.

**Comparisons**

The ANYGRAPH function searches a character string for a graphical character. The NOTGRAPH function searches a character string for a non-graphical character.

**Examples**

**Example 1: Searching a String for Graphical Characters**

The following example uses the ANYGRAPH function to search a string for graphical characters.

```sas
data _null_
  string='Next = _n_ + 12E3;';
  j=0;
  do until(j=0);
    j=anygraph(string, j+1);
    if j=0 then put +3 "That's all";
    else do;
      c=substr(string, j, 1);
      put +3 j= c=;
    end;
  end;
run;
```
SAS writes the following output to the log:

```
j=1 c=N
j=2 c=e
j=3 c=x
j=4 c=t
j=6 c==
j=8 c=_
j=9 c=n
j=10 c=_
j=12 c+=
j=14 c=1
j=15 c=2
j=16 c=E
j=17 c=3
j=18 c=
That's all
```

Example 2: Identifying Control Characters By Using the ANYGRAPH Function
You can execute the following program to show the control characters that are identified by the ANYGRAPH function.

```sas
data test;
  do dec=0 to 255;
    byte=byte(dec);
    hex=put(dec, hex2.);
    anygraph=anygraph(byte);
    output;
  end;

  proc print data=test;
  run;
```

See Also

Functions:

- “NOTGRAPH Function” on page 457

---

ANYLOWER Function

Searches a character string for a lowercase letter, and returns the first position at which the letter is found.

**Category:** Character

**Restriction:** This function is assigned an I18N Level 1 status. If possible, avoid I18N Level 1 functions if you are using a non-English language. Under certain circumstances, the I18N Level 1 functions might not work correctly with Double-Byte Character Set (DBCS) or Multi-Byte Character Set (MBCS) encodings. For more information, see Internationalization Compatibility.

**Syntax**

```
ANYLOWER(string <,start>)
```
Required Argument

string

is the character constant, variable, or expression to search.

Optional Argument

start

is an optional integer that specifies the position at which the search should start and the direction in which to search.

Details

The ANYLOWER function searches a string for the first occurrence of a lowercase letter. If such a character is found, ANYLOWER returns the position in the string of that character. If no such character is found, ANYLOWER returns a value of 0.

If you use only one argument, ANYLOWER begins the search at the beginning of the string. If you use two arguments, the absolute value of the second argument, start, specifies the position at which to begin the search. The direction in which to search is determined in the following way:

• If the value of start is positive, the search proceeds to the right.
• If the value of start is negative, the search proceeds to the left.
• If the value of start is less than the negative length of the string, the search begins at the end of the string.

ANYLOWER returns a value of zero when one of the following is true:

• The character that you are searching for is not found.
• The value of start is greater than the length of the string.
• The value of start = 0.

Comparisons

The ANYLOWER function searches a character string for a lowercase letter. The NOTLOWER function searches a character string for a character that is not a lowercase letter.

Example

The following example uses the ANYLOWER function to search a string for any character that is a lowercase letter.

data _null_;  
  string='Next = _n_ + 12E3;';  
j=0;  
do until(j=0);  
j=anylower(string, j+1);  
  if j=0 then put +3 "That's all";  
  else do;  
    c=substr(string, j, 1);  
    put +3 j= c=;  
  end;  
end;  
run;
SAS writes the following output to the log:

```plaintext
j=2  c=e 
j=3  c=x 
j=4  c=t 
j=9  c=n 
That's all
```

See Also

Functions:

- “NOTLOWER Function” on page 459

**ANYNAME Function**

Searches a character string for a character that is valid in a SAS variable name under VALIDVARNAME=V7, and returns the first position at which that character is found.

**Category:** Character

**Restriction:** This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility.

**Syntax**

```plaintext
ANYNAME(string <,start>)
```

**Required Argument**

`string`

is the character constant, variable, or expression to search.

**Optional Argument**

`start`

is an optional integer that specifies the position at which the search should start and the direction in which to search.

**Details**

The ANYNAME function does not depend on the TRANTAB, ENCODING, or LOCALE system options.

The ANYNAME function searches a string for the first occurrence of any character that is valid in a SAS variable name under VALIDVARNAME=V7. These characters are the underscore (_), digits, and uppercase or lowercase English letters. If such a character is found, ANYNAME returns the position in the string of that character. If no such character is found, ANYNAME returns a value of 0.

If you use only one argument, ANYNAME begins the search at the beginning of the string. If you use two arguments, the absolute value of the second argument, `start`, specifies the position at which to begin the search. The direction in which to search is determined in the following way:
• If the value of `start` is positive, the search proceeds to the right.
• If the value of `start` is negative, the search proceeds to the left.
• If the value of `start` is less than the negative length of the string, the search begins at the end of the string.

ANYNAME returns a value of zero when one of the following is true:
• The character that you are searching for is not found.
• The value of `start` is greater than the length of the string.
• The value of `start` = 0.

Comparisons

The ANYNAME function searches a string for the first occurrence of any character that is valid in a SAS variable name under VALIDVARNAME=V7. The NOTNAME function searches a string for the first occurrence of any character that is not valid in a SAS variable name under VALIDVARNAME=V7.

Example

The following example uses the ANYNAME function to search a string for any character that is valid in a SAS variable name under VALIDVARNAME=V7.

```
data _null_;    
    string='Next = _n_ + 12E3';
    j=0;
    do until(j=0);
        j=anyname(string, j+1);
        if j=0 then put +3 "That's all";
        else do;
        c=substr(string, j, 1);
        put +3 j= c=;
        end;
    end;
    run;
```

SAS writes the following output to the log:

```
j=1 c=N
j=2 c=e
j=3 c=x
j=4 c=t
j=8 c=_
j=9 c=n
j=10 c=_
j=14 c=I
j=15 c=z
j=16 c=E
j=17 c=3
That's all
```

See Also

Functions:
• “NOTNAME Function” on page 461
ANYPRINT Function

Searches a character string for a printable character, and returns the first position at which that character is found.

**Category:** Character

**Restriction:** This function is assigned an I18N Level 1 status. If possible, avoid I18N Level 1 functions if you are using a non-English language. Under certain circumstances, the I18N Level 1 functions might not work correctly with Double-Byte Character Set (DBCS) or Multi-Byte Character Set (MBCS) encodings. For more information, see Internationalization Compatibility.

**Syntax**

\[
\text{ANYPRINT}(\text{string}, \text{start})
\]

**Required Argument**

*string*

is the character constant, variable, or expression to search.

**Optional Argument**

*start*

is an optional integer that specifies the position at which the search should start and the direction in which to search.

**Details**

The ANYPRINT function searches a string for the first occurrence of a printable character. If such a character is found, ANYPRINT returns the position in the string of that character. If no such character is found, ANYPRINT returns a value of 0.

If you use only one argument, ANYPRINT begins the search at the beginning of the string. If you use two arguments, the absolute value of the second argument, *start*, specifies the position at which to begin the search. The direction in which to search is determined in the following way:

- If the value of *start* is positive, the search proceeds to the right.
- If the value of *start* is negative, the search proceeds to the left.
- If the value of *start* is less than the negative length of the string, the search begins at the end of the string.

ANYPRINT returns a value of zero when one of the following is true:

- The character that you are searching for is not found.
- The value of *start* is greater than the length of the string.
- The value of *start* = 0.

**Comparisons**

The ANYPRINT function searches a character string for a printable character. The NOTPRINT function searches a character string for a non-printable character.
Examples

**Example 1: Searching a String for a Printable Character**
The following example uses the ANYPRINT function to search a string for printable characters.

```sas
data _null_
  string='Next = _n_ + 12E3;';
  j=0;
  do until(j=0);
    j=anyprint(string, j+1);
    if j=0 then put +3 "That's all";
    else do;
      c=substr(string, j, 1);
      put +3 j= c=;
    end;
  end;
run;

SAS writes the following output to the log:

```
| j=1 | c= N |
| j=2 | c= e |
| j=3 | c= x |
| j=4 | c= t |
| j=5 | c= |
| j=6 | c= |
| j=7 | c= |
| j=8 | c= _ |
| j=9 | c= n |
| j=10| c= _ |
| j=11| c= |
| j=12| c= + |
| j=13| c= |
| j=14| c=1 |
| j=15| c=2 |
| j=16| c=E |
| j=17| c=3 |
| j=18| c= |
```

That's all

**Example 2: Identifying Control Characters By Using the ANYPRINT Function**
You can execute the following program to show the control characters that are identified by the ANYPRINT function.

```sas
data dec test;
  do dec=0 to 255;
    byte=byte(dec);
    hex=put(dec,hex2.);
    anyprint=anyprint(byte);
    output;
  end;
  proc print data=test;
run;
```
ANYPUNCT Function

Searches a character string for a punctuation character, and returns the first position at which that character is found.

**Category:** Character

**Restriction:** This function is assigned an I18N Level 1 status. If possible, avoid I18N Level 1 functions if you are using a non-English language. Under certain circumstances, the I18N Level 1 functions might not work correctly with Double-Byte Character Set (DBCS) or Multi-Byte Character Set (MBCS) encodings. For more information, see Internationalization Compatibility.

### Syntax

\[ \text{ANYPUNCT}(string <start>) \]

**Required Argument**

- **string**
  - is the character constant, variable, or expression to search.

**Optional Argument**

- **start**
  - is an optional integer that specifies the position at which the search should start and the direction in which to search.

### Details

The ANYPUNCT function searches a string for the first occurrence of a punctuation character. If such a character is found, ANYPUNCT returns the position in the string of that character. If no such character is found, ANYPUNCT returns a value of 0.

If you use only one argument, ANYPUNCT begins the search at the beginning of the string. If you use two arguments, the absolute value of the second argument, start, specifies the position at which to begin the search. The direction in which to search is determined in the following way:

- If the value of start is positive, the search proceeds to the right.
- If the value of start is negative, the search proceeds to the left.
- If the value of start is less than the negative length of the string, the search begins at the end of the string.

ANYPUNCT returns a value of zero when one of the following is true:

- The character that you are searching for is not found.
- The value of start is greater than the length of the string.
• The value of $start = 0$ or if it is missing, then
  \[ x = \text{anypunct}(x, .); \]

**Comparisons**

The ANYPUNCT function searches a character string for a punctuation character. The NOTPUNCT function searches a character string for a character that is not a punctuation character.

**Examples**

**Example 1: Searching a String for Punctuation Characters**

The following example uses the ANYPUNCT function to search a string for punctuation characters.

```sas
data _null_;  
  string='Next = _n_ + 12E3;';  
  j=0;  
  do until(j=0);  
    j=anypunct(string, j+1);  
    if j=0 then put +3 "That's all";  
    else do;  
      c=substr(string, j, 1);  
      put +3 j= c=;  
    end;  
  end;  
run;
```

SAS writes the following output to the log:

```
j=6 c==
j=8 c=_
j=10 c=_
j=12 c=+
j=18 c=;
That's all
```

**Example 2: Identifying Punctuation Characters By Using the ANYPUNCT Function**

You can execute the following program to show the control characters that are identified by the ANYPUNCT function.

```sas
data test;  
  do dec=0 to 255;  
    byte=byte(dec);  
    hex=put(dec, hex2.);  
    anypunct=anypunct(byte);  
    output;  
  end;  
  proc print data=test;  
run;
```
ANYSPACE Function

Searches a character string for a Whitespace character (blank, horizontal and vertical tab, carriage return, line feed, and form feed), and returns the first position at which that character is found.

**Category:** Character

**Restriction:** This function is assigned an I18N Level 1 status. If possible, avoid I18N Level 1 functions if you are using a non-English language. Under certain circumstances, the I18N Level 1 functions might not work correctly with Double-Byte Character Set (DBCS) or Multi-Byte Character Set (MBCS) encodings. For more information, see [Internationalization Compatibility](#).

**Syntax**

`ANYSPACE(string <start>)`

**Required Argument**

`string` is the character constant, variable, or expression to search.

**Optional Argument**

`start` is an optional integer that specifies the position at which the search should start and the direction in which to search.

**Details**

The ANYSPACE function searches a string for the first occurrence of any character that is a blank, horizontal tab, vertical tab, carriage return, line feed, or form feed. If such a character is found, ANYSPACE returns the position in the string of that character. If no such character is found, ANYSPACE returns a value of 0.

If you use only one argument, ANYSPACE begins the search at the beginning of the string. If you use two arguments, the absolute value of the second argument, `start`, specifies the position at which to begin the search. The direction in which to search is determined in the following way:

- If the value of `start` is positive, the search proceeds to the right.
- If the value of `start` is negative, the search proceeds to the left.
- If the value of `start` is less than the negative length of the string, the search begins at the end of the string.

ANYSPACE returns a value of zero when one of the following is true:

- The character that you are searching for is not found.
- The value of `start` is greater than the length of the string.
• The value of \( start = 0 \).

**Comparisons**

The ANYSPACE function searches a character string for the first occurrence of a character that is a blank, horizontal tab, vertical tab, carriage return, line feed, or form feed. The NOTSPACE function searches a character string for the first occurrence of a character that is not a blank, horizontal tab, vertical tab, carriage return, line feed, or form feed.

**Examples**

**Example 1: Searching a String for a Whitespace Character**
The following example uses the ANYSPACE function to search a string for a character that is a Whitespace character.

```sas
data _null_;
  string='Next = _n_ + 12E3;';
  j=0;
  do until(j=0);
    j=anyspace(string, j+1);
    if j=0 then put +3 "That's all";
    else do;
      c=substr(string, j, 1);
      put +3 j= c=;
    end;
  end;
run;
```

SAS writes the following output to the log:

```
j=5 c=
j=7 c=
j=11 c=
j=13 c=
That's all
```

**Example 2: Identifying Control Characters By Using the ANYSPACE Function**
You can execute the following program to show the control characters that are identified by the ANYSPACE function.

```sas
data test;
  do dec=0 to 255;
    byte=byte(dec);
    hex=put(dec, hex2.);
    anyspace=anyspace(byte);
    output;
  end;
proc print data=test;
run;
```
See Also

Functions:
- “NOTSPACE Function” on page 466

ANYUPPER Function

Searches a character string for an uppercase letter, and returns the first position at which the letter is found.

**Category:** Character

**Restriction:** This function is assigned an I18N Level 1 status. If possible, avoid I18N Level 1 functions if you are using a non-English language. Under certain circumstances, the I18N Level 1 functions might not work correctly with Double-Byte Character Set (DBCS) or Multi-Byte Character Set (MBCS) encodings. For more information, see Internationalization Compatibility.

**Syntax**

\[
\text{ANYUPPER}(\text{string} <,\text{start}>)
\]

**Required Argument**

\textit{string}

is the character constant, variable, or expression to search.

**Optional Argument**

\textit{start}

is an optional integer that specifies the position at which the search should start and the direction in which to search.

**Details**

The ANYUPPER function searches a string for the first occurrence of an uppercase letter. If such a character is found, ANYUPPER returns the position in the string of that character. If no such character is found, ANYUPPER returns a value of 0.

If you use only one argument, ANYUPPER begins the search at the beginning of the string. If you use two arguments, the absolute value of the second argument, \textit{start}, specifies the position at which to begin the search. The direction in which to search is determined in the following way:

- If the value of \textit{start} is positive, the search proceeds to the right.
- If the value of \textit{start} is negative, the search proceeds to the left.
- If the value of \textit{start} is less than the negative length of the string, the search begins at the end of the string.

ANYUPPER returns a value of zero when one of the following is true:

- The character that you are searching for is not found.
- The value of \textit{start} is greater than the length of the string.
• The value of \texttt{start} = 0.

**Comparisons**

The \texttt{ANYUPPER} function searches a character string for an uppercase letter. The \texttt{NOTUPPER} function searches a character string for a character that is not an uppercase letter.

**Example**

The following example uses the \texttt{ANYUPPER} function to search a string for an uppercase letter.

```sas
data _null_;
  string='Next = _n_ + 12E3;';
  j=0;
  do until(j=0);
    j=anyupper(string, j+1);
    if j=0 then put +3 "That's all";
    else do;
      c=substr(string, j, 1);
      put +3 j= c=;
    end;
  end;
run;
```

SAS writes the following output to the log:

```
j=1 c=N
j=16 c=E
That's all
```

**See Also**

**Functions:**

• “\texttt{NOTUPPER Function}” on page 468

**ANYXDIGIT Function**

Searches a character string for a hexadecimal character that represents a digit, and returns the first position at which that character is found.

- **Category:** Character
- **Restriction:** This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility.

**Syntax**

\begin{verbatim}
ANYXDIGIT\((string <,start>)\)
\end{verbatim}
**Required Argument**

*string*

is the character constant, variable, or expression to search.

**Optional Argument**

*start*

is an optional integer that specifies the position at which the search should start and the direction in which to search.

**Details**

The ANYXDIGIT function does not depend on the TRANTAB, ENCODING, or LOCALE system options.

The ANYXDIGIT function searches a string for the first occurrence of any character that is a digit or an uppercase or lowercase A, B, C, D, E, or F. If such a character is found, ANYXDIGIT returns the position in the string of that character. If no such character is found, ANYXDIGIT returns a value of 0.

If you use only one argument, ANYXDIGIT begins the search at the beginning of the string. If you use two arguments, the absolute value of the second argument, *start*, specifies the position at which to begin the search. The direction in which to search is determined in the following way:

- If the value of *start* is positive, the search proceeds to the right.
- If the value of *start* is negative, the search proceeds to the left.
- If the value of *start* is less than the negative length of the string, the search begins at the end of the string.

ANYXDIGIT returns a value of zero when one of the following is true:

- The character that you are searching for is not found.
- The value of *start* is greater than the length of the string.
- The value of *start* = 0.

**Comparisons**

The ANYXDIGIT function searches a character string for a character that is a hexadecimal character. The NOTXDIGIT function searches a character string for a character that is not a hexadecimal character.

**Example**

The following example uses the ANYXDIGIT function to search a string for a hexadecimal character that represents a digit.

```plaintext
data _null_
  string='Next = _n_ + 12E3;'
  j=0;
do until(j=0);
j=anyxdigit(string, j+1);
if j=0 then put +3 "That's all"
else do;
c=substr(string, j, 1);
put +3 j= c=;
```

SAS writes the following output to the log:

```
j=2 c=e
j=14 c=1
j=15 c=2
j=16 c=E
j=17 c=3
That's all
```

See Also

Functions:
- “NOTXDIGIT Function” on page 470

ARCOS Function

Returns the arccosine.

**Category:** Trigonometric

**Syntax**

```
ARCOS (argument)
```

**Required Argument**

Argumnet

specifies a numeric constant, variable, or expression.

**Range**

between −1 and 1

**Details**

The ARCOS function returns the arccosine (inverse cosine) of the argument. The value that is returned is specified in radians.

**Example**

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=arccos(1);</td>
<td>0</td>
</tr>
<tr>
<td>x=arccos(0);</td>
<td>1.5707963268</td>
</tr>
<tr>
<td>x=arccos(-0.5);</td>
<td>2.0943951024</td>
</tr>
</tbody>
</table>
ARCOSH Function

Returns the inverse hyperbolic cosine.

**Category:** Hyperbolic

**Syntax**

\[
\text{ARCOSH}(x)
\]

**Required Argument**

- \(x\)
  - specifies a numeric constant, variable, or expression.

**Range** \(x \geq 1\)

**Details**

The ARCOSH function computes the inverse hyperbolic cosine. The ARCOSH function is mathematically defined by the following equation, where \(x \geq 1\):

\[
\text{ARCOSH}(x) = \log\left(x + \sqrt{x^2 - 1}\right)
\]

**Example**

The following example computes the inverse hyperbolic cosine.

```sas
data _null_;  
x=arcosh(5);  
x1=arcosh(13);  
put x=;  
put x1=;  
run;  
```

SAS writes the following output to the log:

```
x=2.2924316696  
x1=3.2566139548
```

**See Also**

**Functions:**

- “TANH Function” on page 1079
- “ARSINH Function” on page 106
- “COSH Function” on page 202
- “SINH Function” on page 606
- “ARTANH Function” on page 107
ARSIN Function
Returns the arcsine.

Category: Trigonometric

Syntax
ARSIN(argument)

Required Argument
argument
  specifies a numeric constant, variable, or expression.

Range: between -1 and 1

Details
The ARSIN function returns the arcsine (inverse sine) of the argument. The value that is
returned is specified in radians.

Example

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=arsin(0);</td>
<td>0</td>
</tr>
<tr>
<td>x=arsin(1);</td>
<td>1.5707963268</td>
</tr>
<tr>
<td>x=arsin(-0.5);</td>
<td>-0.523598776</td>
</tr>
</tbody>
</table>

ARSINH Function
Returns the inverse hyperbolic sine.

Category: Hyperbolic

Syntax
ARSINH(x)

Required Argument
x
  specifies a numeric constant, variable, or expression.
Details

The ARSINH function computes the inverse hyperbolic sine. The ARSINH function is mathematically defined by the following equation, where \(-\infty < x < \infty\):

\[
\text{ARSINH}(x) = \log \left( x + \sqrt{x^2 + 1} \right)
\]

Replace the infinity symbol with the largest double precision number that is available on your machine.

Example

The following example computes the inverse hyperbolic sine.

```
data _null_;  
x=arsinh(5);  
x1=arsinh(-5);  
put x=;  
put x1=;  
run;  
```

SAS writes the following output to the log:

```
x=2.3124383413  
x1=-2.312438341
```

See Also

Functions:

- “ARCOSH Function” on page 105
- “ARTANH Function” on page 107
- “COSH Function” on page 202
- “SINH Function” on page 606
- “TANH Function” on page 1079

**ARTANH Function**

Returns the inverse hyperbolic tangent.

**Category:** Hyperbolic

**Syntax**

```
ARTANH(x)
```

**Required Argument**

\(x\)

specifies a numeric constant, variable, or expression.
Range  \(-1 < x < 1\)

**Details**

The ARTANH function computes the inverse hyperbolic tangent. The ARTANH function is mathematically defined by the following equation, where \(-1 < x < 1\):

\[
\text{ARTANH}(x) = \frac{1}{2} \log \left( \frac{1 + x}{1 - x} \right)
\]

**Example**

The following example computes the inverse hyperbolic tangent.

```sas
data _null_;  
x=artanh(0.5);  
put x=;  
run;
```

SAS writes the following output to the log:

```
x=0.5493061443
```

**See Also**

**Functions:**
- “ARCOSH Function” on page 105
- “ARSINH Function” on page 106
- “COSH Function” on page 202
- “SINH Function” on page 606
- “TANH Function” on page 1079

---

**ATAN Function**

Returns the arc tangent.

**Category:** Trigonometric

**Syntax**

\[
\text{ATAN}(\text{argument})
\]

**Required Argument**

\[
\text{argument}
\]

specifies a numeric constant, variable, or expression.

**Details**

The ATAN function returns the 2-quadrant arc tangent (inverse tangent) of the argument. The value that is returned is the angle (in radians) whose tangent is \(x\) and whose value
ranges from \(-\pi/2\) to \(\pi/2\). If the argument is missing, then ATAN returns a missing value.

**Comparisons**

The ATAN function is similar to the ATAN2 function except that ATAN2 calculates the arc tangent of the angle from the ratio of two arguments rather than from one argument.

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>data_null_;</td>
<td></td>
</tr>
<tr>
<td>x=atan(0);</td>
<td>x=0</td>
</tr>
<tr>
<td>put x=;</td>
<td></td>
</tr>
<tr>
<td>run;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>data_null;</td>
<td></td>
</tr>
<tr>
<td>x=atan(1);</td>
<td>x=0.7853981634</td>
</tr>
<tr>
<td>put x=;</td>
<td></td>
</tr>
<tr>
<td>run;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>data_null_;</td>
<td></td>
</tr>
<tr>
<td>x=atan(-9.0);</td>
<td>x=-1.460139106</td>
</tr>
<tr>
<td>put x=;</td>
<td></td>
</tr>
<tr>
<td>run;</td>
<td></td>
</tr>
</tbody>
</table>

**See Also**

Functions:

- “ATAN2 Function” on page 109

**ATAN2 Function**

Returns the arc tangent of the ratio of two numeric variables.

**Category:** Trigonometric

**Syntax**

\[
\text{ATAN2}(\text{argument-1}, \text{argument-2})
\]

**Required Arguments**

- \textit{argument-1} specifies a numeric constant, variable, or expression.
- \textit{argument-2} specifies a numeric constant, variable, or expression.
Details

The ATAN2 function returns the arc tangent (inverse tangent) of two numeric variables. The result of this function is similar to the result of calculating the arc tangent of argument-1 / argument-2, except that the signs of both arguments are used to determine the quadrant of the result. ATAN2 returns the result in radians, which is a value between \(-\pi\) and \(\pi\). If either of the arguments in ATAN2 is missing, then ATAN2 returns a missing value.

Comparisons

The ATAN2 function is similar to the ATAN function except that ATAN calculates the arc tangent of the angle from the value of one argument rather than from two arguments.

Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>a=atan2(-1, 0.5);</td>
<td>-1.107148718</td>
</tr>
<tr>
<td>b=atan2(6, 8);</td>
<td>0.6435011088</td>
</tr>
<tr>
<td>c=atan2(5, -3);</td>
<td>2.1112158271</td>
</tr>
</tbody>
</table>

See Also

Functions:
- “ATAN Function” on page 108

BAND Function

Returns the bitwise logical AND of two arguments.

Category: Bitwise Logical Operations

Syntax

\[
\text{band}(\text{argument-1, argument-2})
\]

Required Argument

\(\text{argument-1, argument-2}\)

specifies a numeric constant, variable, or expression.

Range: between 0 and \((2^{32})-1\) inclusive
Details

If either argument contains a missing value, then the function returns a missing value and sets _ERROR_ equal to 1.

Example

The following SAS statements produce this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>data <em>null</em>;</td>
<td>x=00000005</td>
</tr>
<tr>
<td>x=band(0Fx, 05x);</td>
<td></td>
</tr>
<tr>
<td>put x=hex,;</td>
<td></td>
</tr>
<tr>
<td>run;</td>
<td></td>
</tr>
</tbody>
</table>

BETA Function

Returns the value of the beta function.

Category: Mathematical

Syntax

BETA(a, b)

Required Arguments

a

is the first shape parameter, where \(a>0\).

b

is the second shape parameter, where \(b>0\).

Details

The BETA function is mathematically given by the equation

\[
\beta(a, b) = \int_0^1 x^{a-1} (1-x)^{b-1} dx
\]

with \(a>0\), \(b>0\). It should be noted that

\[
\beta(a, b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}
\]

where \(\Gamma(\cdot)\) is the gamma function.

If the expression cannot be computed, BETA returns a missing value.

Example

The following SAS statement produces this result:
SAS Statement | Result
---|---
\texttt{x=beta(5, 3);} | 0.9523809524e-2

See Also

Functions:

- “LOGBETA Function” on page 415

BETAINV Function

Returns a quantile from the beta distribution.

Category: Quantile

Syntax

\texttt{BETAINV}(p, a, b)

Required Arguments

\( p \)

is a numeric probability.

Range \( 0 \leq p \leq 1 \)

\( a \)

is a numeric shape parameter.

Range \( a > 0 \)

\( b \)

is a numeric shape parameter.

Range \( b > 0 \)

Details

The BETAINV function returns the \( p \)th quantile from the beta distribution with shape parameters \( a \) and \( b \). The probability that an observation from a beta distribution is less than or equal to the returned quantile is \( p \).

Note: BETAINV is the inverse of the PROBBETA function.

Example

The following SAS statements produce this result:
### See Also

**Functions:**
- “QUANTILE Function” on page 549

---

**BLACKCLPRC Function**

Calculates call prices for European options on futures, based on the Black model.

**Category:** Financial

**Syntax**

\[
\text{BLACKCLPRC}(E, t, F, r, \sigma)
\]

**Required Arguments**

- **E**
  - is a nonmissing, positive value that specifies exercise price.
  - **Requirement** Specify \(E\) and \(F\) in the same units.

- **t**
  - is a nonmissing value that specifies time to maturity, in years.

- **F**
  - is a nonmissing, positive value that specifies future price.
  - **Requirement** Specify \(F\) and \(E\) in the same units.

- **r**
  - is a nonmissing, positive value that specifies the annualized risk-free interest rate, continuously compounded.

- **sigma**
  - is a nonmissing, positive fraction that specifies the volatility (the square root of the variance of \(r\)).

**Details**

The BLACKCLPRC function calculates call prices for European options on futures, based on the Black model. The function is based on the following relationship:

\[
\text{CALL} = e^{-rt} \left( \text{FN}(d_1) - \text{EN}(d_2) \right)
\]

**Arguments**
\( F \)
- specifies future price.

\( N \)
- specifies the cumulative normal density function.

\( E \)
- specifies the exercise price of the option.

\( r \)
- specifies the risk-free interest rate, which is an annual rate that is expressed in terms of continuous compounding.

\( t \)
- specifies the time to expiration, in years.

\[ d_1 = \frac{\ln \left( \frac{F}{E} \right) + \left( \frac{\sigma^2}{2} \right) t}{\sigma \sqrt{t}} \]

\[ d_2 = d_1 - \sigma \sqrt{t} \]

The following arguments apply to the preceding equation:

\( \sigma \)
- specifies the volatility of the underlying asset.

\( \sigma^2 \)
- specifies the variance of the rate of return.

For the special case of \( t=0 \), the following equation is true:

\[ \text{CALL} = \max(F - E, 0) \]

For information about the basics of pricing, see Using Pricing Functions on page 8.

**Comparisons**

The BLACKCLPRC function calculates call prices for European options on futures, based on the Black model. The BLACKPTPRC function calculates put prices for European options on futures, based on the Black model. These functions return a scalar value.

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>a=blackclprc(50, .25, 48, .05, .25); put a;</td>
<td>1.5513014272</td>
</tr>
<tr>
<td>b=blackclprc(9, 1/12, 10, .05, .2); put b;</td>
<td>1.0031514194</td>
</tr>
</tbody>
</table>

**See Also**

**Functions:**
BLACKPTPRC Function

Calculates put prices for European options on futures, based on the Black model.

**Category:** Financial

**Syntax**

BLACKPTPRC\( (E, t, F, r, \sigma) \)

**Required Arguments**

\( E \)

is a nonmissing, positive value that specifies exercise price.

**Requirement** Specify \( E \) and \( F \) in the same units.

\( t \)

is a nonmissing value that specifies time to maturity, in years.

\( F \)

is a nonmissing, positive value that specifies future price.

**Requirement** Specify \( F \) and \( E \) in the same units.

\( r \)

is a nonmissing, positive value that specifies the annualized risk-free interest rate, continuously compounded.

\( \sigma \)

is a nonmissing, positive fraction that specifies the volatility (the square root of the variance of \( r \)).

**Details**

The BLACKPTPRC function calculates put prices for European options on futures, based on the Black model. The function is based on the following relationship:

\[
PUT = \text{CALL} + e^{-rt}(E - F)
\]

**Arguments**

\( E \)

specifies the exercise price of the option.

\( r \)

specifies the risk-free interest rate, which is an annual rate that is expressed in terms of continuous compounding.

\( t \)

specifies the time to expiration, in years.

\( F \)

specifies future price.
\[ d_1 = \frac{\ln(E) + \left(\frac{\sigma^2 t}{2}\right)}{\sigma \sqrt{t}} \]

\[ d_2 = d_1 - \sigma \sqrt{t} \]

The following arguments apply to the preceding equation:

\( \sigma \)

specifies the volatility of the underlying asset.

\( \sigma^2 \)

specifies the variance of the rate of return.

For the special case of \( t=0 \), the following equation is true:

\[ \text{PUT} = \max((E - F), 0) \]

For information about the basics of pricing, see Using Pricing Functions on page 8.

**Comparisons**

The BLACKPTPRC function calculates put prices for European options on futures, based on the Black model. The BLACKCLPRC function calculates call prices for European options on futures, based on the Black model. These functions return a scalar value.

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>a=blackptprc(298, .25, 350, .06, .25); put a;</td>
<td>1.8598056393</td>
</tr>
<tr>
<td>b=blackptprc(145, .5, 170, .05, .2); put b;</td>
<td>1.4123497991</td>
</tr>
</tbody>
</table>

**See Also**

Functions:

- “BLACKCLPRC Function” on page 113

**BLKSHCLPRC Function**

Calculates call prices for European options on stocks, based on the Black-Scholes model.

**Category:** Financial

**Syntax**

\[
\text{BLKSHCLPRC}(E, t, S, r, \sigma) \]
**Required Arguments**

\( E \)

is a nonmissing, positive value that specifies the exercise price.

**Requirement**  Specify \( E \) and \( S \) in the same units.

\( t \)

is a nonmissing value that specifies the time to maturity, in years.

\( S \)

is a nonmissing, positive value that specifies the share price.

**Requirement**  Specify \( S \) and \( E \) in the same units.

\( r \)

is a nonmissing, positive value that specifies the annualized risk-free interest rate, continuously compounded.

\( \sigma \)

is a nonmissing, positive fraction that specifies the volatility of the underlying asset.

**Details**

The BLKSHCLPRC function calculates the call prices for European options on stocks, based on the Black-Scholes model. The function is based on the following relationship:

\[
\text{CALL} = SN(d_1) - EN(d_2)e^{-rt}
\]

**Arguments**

\( S \)

is a nonmissing, positive value that specifies the share price.

\( N \)

specifies the cumulative normal density function.

\( E \)

is a nonmissing, positive normal density function.

\( d_1 = \left( \ln \left( \frac{S}{E} \right) + \left( r + \frac{\sigma^2}{2} \right) t \right) / \sigma \sqrt{t} \)

\( d_2 = d_1 - \sigma \sqrt{t} \)

The following arguments apply to the preceding equation:

\( t \)

specifies the time to expiration, in years.

\( r \)

specifies the risk-free interest rate, which is an annual rate that is expressed in terms of continuous compounding.

\( \sigma \)

specifies the volatility (the square root of the variance).

\( \sigma^2 \)

specifies the variance of the rate of return.

For the special case of \( t=0 \), the following equation is true:
CALL = max((S – E), 0)

For information about the basics of pricing, see Using Pricing Functions on page 8.

Comparisons

The BLKSHCLPRC function calculates the call prices for European options on stocks, based on the Black-Scholes model. The BLKSHPTPRC function calculates the put prices for European options on stocks, based on the Black-Scholes model. These functions return a scalar value.

Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>a=blkshclprc(50, .25, 48, .05, .25); put a;</td>
<td>1.7989420195</td>
</tr>
<tr>
<td>b=blkshclprc(9, 1/12, 10, .05, .2); put b;</td>
<td>1.0435083341</td>
</tr>
</tbody>
</table>

See Also

Functions:

- “BLKSHPTPRC Function” on page 118

BLKSHPTPRC Function

Calculates put prices for European options on stocks, based on the Black-Scholes model.

**Category:** Financial

**Syntax**

\[ \text{BLKSHPTPRC}(E, t, S, r, \sigma) \]

**Required Arguments**

\( E \)

is a nonmissing, positive value that specifies the exercise price.

**Requirement** Specify \( E \) and \( S \) in the same units.

\( t \)

is a nonmissing value that specifies the time to maturity, in years.

\( S \)

is a nonmissing, positive value that specifies the share price.
 Requirement Specify $S$ and $E$ in the same units.

$r$

is a nonmissing, positive value that specifies the annualized risk-free interest rate, continuously compounded.

$\sigma$

is a nonmissing, positive fraction that specifies the volatility of the underlying asset.

**Details**

The BLKSHPTPRC function calculates the put prices for European options on stocks, based on the Black-Scholes model. The function is based on the following relationship:

$$\text{PUT} = \text{CALL} - S + E e^{-rt}$$

**Arguments**

$S$

is a nonmissing, positive value that specifies the share price.

$E$

is a nonmissing, positive value that specifies the exercise price of the option.

$$d_1 = \frac{\ln\left(\frac{S}{E}\right) + \left(r + \frac{\sigma^2}{2}\right)t}{\sigma\sqrt{t}}$$

$$d_2 = d_1 - \sigma\sqrt{t}$$

The following arguments apply to the preceding equation:

$t$

specifies the time to expiration, in years.

$r$

specifies the risk-free interest rate, which is an annual rate that is expressed in terms of continuous compounding.

$\sigma$

specifies the volatility (the square root of the variance).

$\sigma^2$

specifies the variance of the rate of return.

For the special case of $t=0$, the following equation is true:

$$\text{PUT} = \max(E - S, 0)$$

For information about the basics of pricing, see Using Pricing Functions on page 8.

**Comparisons**

The BLKSHPTPRC function calculates the put prices for European options on stocks, based on the Black-Scholes model. The BLKSHCLPRC function calculates the call prices for European options on stocks, based on the Black-Scholes model. These functions return a scalar value.

**Example**

The following SAS statements produce these results:
SAS Statement | Result
---|---
a=blkhptprc(230, .5, 290, .04, .25); | 1.5659744295
   put a; | 
b=blkhptprc(350, .3, 400, .05, .2); | 1.6409194307
   put b; | 

See Also

Functions:
- “BLKSHCLPRC Function” on page 116

BLSHIFT Function

Returns the bitwise logical left shift of two arguments.

Category: Bitwise Logical Operations

Syntax

BLSHIFT(argument-1, argument-2)

Required Arguments

argument-1
specifies a numeric constant, variable, or expression.
Range between 0 and \(2^{32}-1\) inclusive

argument-2
specifies a numeric constant, variable, or expression.
Range 0 to 31, inclusive

Details

If either argument contains a missing value, then the function returns a missing value and sets _ERROR_ equal to 1.

Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
</table>
x=blshift('07x', 2); | x=0000001C
   put x=hex.; | 

**BNOT Function**

Returns the bitwise logical NOT of an argument.

**Category:** Bitwise Logical Operations

---

**Syntax**

*BNOT*(argument)

**Required Argument**

*argument*

- specifies a numeric constant, variable, or expression.
- **Range** between 0 and $(2^{32})$-1 inclusive

---

**Details**

If the argument contains a missing value, then the function returns a missing value and sets _ERROR_ equal to 1.

---

**Example**

The following SAS statements produce this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=bnot(0F000000Fx); put x=hex.;</td>
<td>x=0FFFFFF0</td>
</tr>
</tbody>
</table>

---

**BOR Function**

Returns the bitwise logical OR of two arguments.

**Category:** Bitwise Logical Operations

---

**Syntax**

*BOR*(argument-1, argument-2)

**Required Argument**

*argument-1, argument-2*

- specifies a numeric constant, variable, or expression.
- **Range** between 0 and $(2^{32})$-1 inclusive
Details
If either argument contains a missing value, then the function returns a missing value and sets _ERROR_ equal to 1.

Example
The following SAS statements produce this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=bor(01x, 0F4x);</td>
<td></td>
</tr>
<tr>
<td>put x=hex.;</td>
<td></td>
</tr>
<tr>
<td>x=000000F5</td>
<td></td>
</tr>
</tbody>
</table>

BRSHIFT Function
Returns the bitwise logical right shift of two arguments.

Category:  Bitwise Logical Operations

Syntax
BRSHIFT(argument-1, argument-2)

Required Arguments

argument-1
specifies a numeric constant, variable, or expression.

Range  between 0 and (2^32)-1 inclusive

argument-2
specifies a numeric constant, variable, or expression.

Range  0 to 31, inclusive

Details
If either argument contains a missing value, then the function returns a missing value and sets _ERROR_ equal to 1.

Example
The following SAS statements produce this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=brshift(01Cx, 2);</td>
<td></td>
</tr>
<tr>
<td>put x=hex.;</td>
<td></td>
</tr>
<tr>
<td>x=00000007</td>
<td></td>
</tr>
</tbody>
</table>
BXOR Function

Returns the bitwise logical EXCLUSIVE OR of two arguments.

**Category:** Bitwise Logical Operations

**Syntax**

`BXOR(argument-1, argument-2)`

**Required Argument**

*argument-1, argument-2*

specifies a numeric constant, variable, or expression.

**Range**

between 0 and \(2^{32}-1\) inclusive

**Details**

If either argument contains a missing value, then the function returns a missing value and sets _ERROR_ equal to 1.

**Example**

The following SAS statements produce this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x=bxor(03x, 01x);</code></td>
<td><code>x=00000002</code></td>
</tr>
<tr>
<td><code>put x=hex.;</code></td>
<td><code>x=00000002</code></td>
</tr>
</tbody>
</table>

BYTE Function

Returns one character in the ASCII collating sequence.

**Category:** Character

**Restriction:**

This function is assigned an I18N Level 0 status, and is designed for SBCS data. Do not use this function to process DBCS or MBCS data. For more information, see [Internationalization Compatibility](#).

**Syntax**

`BYTE (n)`

**Required Argument**

*n*

specifies an integer that represents a specific ASCII character.
Details

Length of Returned Variable
In a DATA step, if the BYTE function returns a value to a variable that has not previously been assigned a length, then that variable is assigned a length of 1.

ASCII Collating Sequences
For ASCII collating sequences, the characters that correspond to values between 0 and 127 represent the standard character set. Other ASCII characters that correspond to values between 128 and 255 are available on certain ASCII operating environments, but the information those characters represent varies with the operating environment.

Example
The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII</td>
<td>P</td>
</tr>
<tr>
<td>x=byte(80);</td>
<td>P</td>
</tr>
<tr>
<td>put x;</td>
<td>P</td>
</tr>
</tbody>
</table>

See Also

Functions:
- “COLLATE Function” on page 179
- “RANK Function” on page 568

CALL LABEL Routine
Assigns a variable label to a specified character variable.

Category: Variable Control

Note: Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

Syntax

CALL LABEL(variable-1, variable-2);
Required Arguments

**variable-1**
specifies any SAS variable. If `variable-1` does not have a label, the variable name is assigned as the value of `variable-2`.

**variable-2**
specifies any SAS character variable. Variable labels can be up to 256 characters long. Therefore, the length of `variable-2` should be at least 256 characters to avoid truncating variable labels.

**Note** To conserve space, you should set the length of `variable-2` to the length of the label for `variable-1`, if it is known.

Details
The CALL LABEL routine assigns the label of the `variable-1` variable to the character variable `variable-2`.

Example
This example uses the CALL LABEL routine with array references to assign the labels of all variables in the data set OLD as values of the variable LAB in data set NEW:

```sas
data new;
  set old;
  length lab $256;
  array abc{*} _character_
  array def{*} _numeric_
  do i=1 to dim(abc);
    call label(abc{i},lab);
    output;
  end;
  do j=1 to dim(def);
    call label(def{j},lab);
    output;
  end;
  stop;
  keep lab;
run;
```

See Also

Functions:
- “VLABEL Function” on page 670
CALL MISSING Routine
Assigns missing values to the specified character or numeric variables.

**Category:** Character

**Note:** Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

**Syntax**

```
CALL MISSING(variable-name-1 <, variable-name-2, ...>);
```

**Required Argument**

variable-name

specifies the name of SAS character or numeric variables.

**Details**

The CALL MISSING routine assigns an ordinary numeric missing value (.) to each numeric variable in the argument list.

The CALL MISSING routine assigns a character missing value (a blank) to each character variable in the argument list. If the current length of the character variable equals the maximum length, the current length is not changed. Otherwise, the current length is set to 1.

You can mix character and numeric variables in the argument list.

**Comparisons**

The MISSING function checks whether the argument has a missing value but does not change the value of the argument.

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>prod='shoes';</td>
<td>prod=shoes invty=7498 sales=.</td>
</tr>
<tr>
<td>invty=7498;</td>
<td>prod=shoes invty=7498 sales=.</td>
</tr>
<tr>
<td>sales=23759;</td>
<td>prod=shoes invty=7498 sales=.</td>
</tr>
<tr>
<td>call missing(sales);</td>
<td>prod=shoes invty=7498 sales=.</td>
</tr>
<tr>
<td>put prod= invty= sales=;</td>
<td>prod=shoes invty=7498 sales=.</td>
</tr>
<tr>
<td>prod='shoes';</td>
<td>prod=shoes invty=7498 sales=.</td>
</tr>
<tr>
<td>invty=7498;</td>
<td>prod=shoes invty=7498 sales=.</td>
</tr>
<tr>
<td>sales=23759;</td>
<td>prod=shoes invty=7498 sales=.</td>
</tr>
<tr>
<td>call missing(prod, invty);</td>
<td>prod=shoes invty=7498 sales=.</td>
</tr>
<tr>
<td>put prod= invty= sales=;</td>
<td>prod invty=. sales=23759</td>
</tr>
</tbody>
</table>
CALL PRXFREE Routine

Frees memory that was allocated for a Perl regular expression.

**Category:** Character String Matching

**Restriction:** Use with the PRXPARSE function.

**Note:** Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

### Syntax

CALL PRXFREE(regular-expression-id);

**Required Argument**

- **regular-expression-id**
  
  Specifies a numeric variable with a value that is the identification number that is returned by the PRXPARSE function. `regular-expression-id` is set to missing if the call to the PRXFREE routine occurs without error.

### Details

The CALL PRXFREE routine frees unneeded resources that were allocated for a Perl regular expression.

For more information about pattern matching, see Pattern Matching Using Perl Regular Expressions (PRX) on page 42.

### Comparisons

The Perl regular expression (PRX) functions and CALL routines work together to manipulate strings that match patterns. To see a list and short description of these functions and CALL routines, see the Character String Matching category in “SAS Functions and CALL Routines by Category” on page 701.
CALL SCAN Routine

Returns the position and length of the \( n \)th word from a character string.

**Category:** Character

**Interaction:** When invoked by the \%SYSCALL macro statement, CALL SCAN removes the quotation marks from its arguments. For more information, see “Using CALL Routines and the \%SYSCALL Macro Statement” on page 9.

**Note:** Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

**Syntax**

```call
CALL SCAN(<string>, count, position, length <, <character-list> <, <modifier(s)>>>);
```

**Required Arguments**

- **count**
  - is a nonzero numeric constant, variable, or expression that has an integer value that specifies the number of the word in the character string that you want the CALL SCAN routine to select. For example, a value of 1 indicates the first word, a value of 2 indicates the second word, and so on. The following rules apply:
  - If \( \text{count} \) is positive, then CALL SCAN counts words from left to right in the character string.
  - If \( \text{count} \) is negative, then CALL SCAN counts words from right to left in the character string.

- **position**
  - specifies a numeric variable in which the position of the word is returned. If \( \text{count} \) exceeds the number of words in the string, then the value that is returned in \( \text{position} \)
is zero. If count is zero or missing, then the value that is returned in position is missing.

length
specifies a numeric variable in which the length of the word is returned. If count exceeds the number of words in the string, then the value that is returned in length is zero. If count is zero or missing, then the value that is returned in length is missing.

Optional Arguments

string
specifies a character constant, variable, or expression.

caller-list
specifies an optional character constant, variable, or expression that initializes a list of characters. This list determines which characters are used as the delimiters that separate words. The following rules apply:

- By default, all characters in character-list are used as delimiters.
- If you specify the K modifier in the modifier argument, then all characters that are not in character-list are used as delimiters.

Tip  You can add more characters to character-list by using other modifiers.

modifier
specifies a character constant, variable, or expression in which each non-blank character modifies the action of the CALL SCAN routine. Blanks are ignored. You can use the following characters as modifiers:

- a or A adds alphabetic characters to the list of characters.
- b or B scans backward, from right to left instead of from left to right, regardless of the sign of the count argument.
- c or C adds control characters to the list of characters.
- d or D adds digits to the list of characters.
- f or F adds an underscore and English letters (that is, valid first characters in a SAS variable name using VALIDVARNAME=V7) to the list of characters.
- g or G adds graphic characters to the list of characters. Graphic characters are those that, when printed, produce an image on paper.
- h or H adds a horizontal tab to the list of characters.
- i or I ignores the case of the characters.
- k or K causes all characters that are not in the list of characters to be treated as delimiters. That is, if K is specified, then characters that are in the list of characters are kept in the returned value rather than being omitted because they are delimiters. If K is not specified, then all characters that are in the list of characters are treated as delimiters.
- l or L adds lowercase letters to the list of characters.
- m or M specifies that multiple consecutive delimiters, and delimiters at the beginning or end of the string argument, refer to words that have a length of zero. If the M modifier is not specified, then multiple consecutive delimiters are treated as one delimiter, and delimiters at the beginning or end of the string argument are ignored.
n or N   adds digits, an underscore, and English letters (that is, the characters that can appear in a SAS variable name using VALIDVARNAME=V7) to the list of characters.

o or O   processes the character-list and modifier arguments only once, rather than every time the CALL SCAN routine is called. Using the O modifier in the DATA step can make CALL SCAN run faster when you call it in a loop where the character-list and modifier arguments do not change. The O modifier applies separately to each instance of the CALL SCAN routine in your SAS code, and does not cause all instances of the CALL SCAN routine to use the same delimiters and modifiers.

p or P   adds punctuation marks to the list of characters.

q or Q   ignores delimiters that are inside substrings that are enclosed in quotation marks. If the value of the string argument contains unmatched quotation marks, then scanning from left to right produces different words than scanning from right to left.

s or S   adds space characters to the list of characters (blank, horizontal tab, vertical tab, carriage return, line feed, and form feed).

t or T   trims trailing blanks from the string and character-list arguments. If you want to remove trailing blanks from just one character argument instead of both character arguments, then use the TRIM function instead of the CALL SCAN routine with the T modifier.

u or U   adds uppercase letters to the list of characters.

w or W   adds printable (writable) characters to the list of characters.

x or X   adds hexadecimal characters to the list of characters.

Tip    If the modifier argument is a character constant, then enclose it in quotation marks. Specify multiple modifiers in a single set of quotation marks. A modifier argument can also be expressed as a character variable or expression.

Details

Some language elements are supported in SAS but not in CAS. An example is the DATALINES statement. Some examples in this document use DATALINES in SAS to create a data set using the CAS engine. For more information, see “Functions in SAS and CAS” on page 54.

Definition of "Delimiter" and "Word"

A delimiter is any of several characters that are used to separate words. You can specify the delimiters in the character-list and modifier arguments.

If you specify the Q modifier, then delimiters inside substrings that are enclosed in quotation marks are ignored.

In the CALL SCAN routine, "word" refers to a substring that has all of the following characteristics:

• is bounded on the left by a delimiter or the beginning of the string
• is bounded on the right by a delimiter or the end of the string
• contains no delimiters
A word can have a length of zero if there are delimiters at the beginning or end of the string, or if the string contains two or more consecutive delimiters. However, the CALL SCAN routine ignores words that have a length of zero unless you specify the M modifier.

**Using Default Delimiters in ASCII and EBCDIC Environments**
If you use the CALL SCAN routine with only four arguments, then the default delimiters depend on whether your computer uses ASCII or EBCDIC characters.

- If your computer uses ASCII characters, then the default delimiters are as follows:
  \[ \text{blank} \quad ! \quad $ \quad \% \quad \& \quad ( \quad * \quad + \quad , \quad . \quad / \quad ; \quad < \quad ^ \quad \_ \quad \} \quad \] \]
  In ASCII environments that do not contain the \(^\) character, the CALL SCAN routine uses the \(~\) character instead.

- If your computer uses EBCDIC characters, then the default delimiters are as follows:
  \[ \text{blank} \quad ! \quad $ \quad \% \quad \& \quad ( \quad * \quad + \quad , \quad . \quad / \quad ; \quad < \quad \_ \quad ¥ \quad \} \quad \] \]
  If you use the Modifier argument without specifying any characters as delimiters, then the only delimiters used are those that are defined by the Modifier argument. In this case, the lists of default delimiters for ASCII and EBCDIC environments are not used. In other words, modifiers add to the list of delimiters that are explicitly specified by the character-list argument. Modifiers do not add to the list of default modifiers.

**Using the CALL SCAN Routine with the M Modifier**
If you specify the M modifier, then the number of words in a string is defined as one plus the number of delimiters in the string. However, if you specify the Q modifier, delimiters that are inside quotation marks are ignored.

If you specify the M modifier, the CALL SCAN routine returns a positive position and a length of zero if one of the following conditions is true:

- The string begins with a delimiter and you request the first word.
- The string ends with a delimiter and you request the last word.
- The string contains two consecutive delimiters and you request the word that is between the two delimiters.

In you specify a count that is greater in absolute value than the number of words in the string, then the CALL SCAN routine returns a position and length of zero.

**Using the CALL SCAN Routine without the M Modifier**
If you do not specify the M modifier, then the number of words in a string is defined as the number of maximal substrings of consecutive non-delimiters. However, if you specify the Q modifier, delimiters that are inside quotation marks are ignored.

If you do not specify the M modifier, then the CALL SCAN routine does the following:

- ignores delimiters at the beginning or end of the string
- treats two or more consecutive delimiters as if they were a single delimiter

If the string contains no characters other than delimiters, or if you specify a count that is greater in absolute value than the number of words in the string, then the CALL SCAN routine returns a position and length of zero.
Finding the Word as a Character String
To find the designated word as a character string after calling the CALL SCAN routine, use the SUBSTRN function with the string, position, and length arguments:

\[
\text{substrn}(\text{string}, \text{position}, \text{length});
\]

Because CALL SCAN can return a length of zero, using the SUBSTR function can cause an error.

Using Null Arguments
The CALL SCAN routine allows character arguments to be null. Null arguments are treated as character strings with a length of zero. Numeric arguments cannot be null.

Examples

Example 1: Scanning for a Word in a String
The following example shows how you can use the CALL SCAN routine to find the position and length of a word in a string.

```sas
data artists;
  input string $60.;
  drop string;
  do i=1 to 99;
    call scan(string, i, position, length);
    if not position then leave;
    Name=substrn(string, position, length);
    output;
  end;
  datalines;
Picasso Toulouse-Lautrec Turner "Van Gogh" Velazquez
;
proc print data=artists;
run;
```

Figure 2.1 Output from Scanning for a Word in a String

<table>
<thead>
<tr>
<th>Obs</th>
<th>i</th>
<th>position</th>
<th>length</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>7</td>
<td>Picasso</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>9</td>
<td>8</td>
<td>Toulouse</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>18</td>
<td>7</td>
<td>Lautrec</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>26</td>
<td>6</td>
<td>Turner</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>33</td>
<td>4</td>
<td>&quot;Van&quot;</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>38</td>
<td>5</td>
<td>&quot;Van Gogh&quot;</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>44</td>
<td>9</td>
<td>Velazquez</td>
</tr>
</tbody>
</table>
You can transform the data set, artists, to a CAS table.

libname mycas cas;
data mycas.artists;
    set artists;
run;
proc print data=mycas.artists;
run;

Figure 2.2  Output from CALL SCAN Function in a CAS Table

Example 2: Finding the First and Last Words in a String
The following example scans a string for the first and last words. Note the following:

- A negative count instructs the CALL SCAN routine to scan from right to left.
- Leading and trailing delimiters are ignored because the M modifier is not used.
- In the last observation, all characters in the string are delimiters, so no words are found.

data firstlast;
    input String $60.;
    call scan(string, 1, First_Pos, First_Length);
    First_Word=substrn(string, First_Pos, First_Length);
    call scan(string, -1, Last_Pos, Last_Length);
    Last_Word=substrn(string, Last_Pos, Last_Length);
    datalines4;
    Jack and Jill
    & Bob & Carol & Ted & Alice & Leonardo
    ! $ % & ( ) * + - . / ;
    ;;;
    proc print data=firstlast;
      var First: Last:;
    run;
Example 3: Finding All Words in a String without Using the M Modifier

The following example scans a string from left to right until no more words are found. Because the M modifier is not used, the CALL SCAN routine does not return any words that have a length of zero. Because blanks are included among the default delimiters, the CALL SCAN routine returns a position or length of zero only when the count exceeds the number of words in the string. The loop can be stopped when the returned position is less than or equal to zero. It is safer to use an inequality comparison to end the loop, rather than to use a strict equality comparison with zero, in case an error causes the position to be missing. (In SAS, a missing value is considered to have a lesser value than any nonmissing value.)

```sas
data all;
  length word $20;
  drop string;
  string=' The quick brown fox jumps over the lazy dog.   ';  
  do until(position <= 0);
    count+1;
    call scan(string, count, position, length);
    word=substrn(string, position, length);
    output;
  end;
run;
proc print data=all noobs;
  var count position length word;
run;
```
Example 4: Finding All Words in a String By Using the M and O Modifiers

The following example shows the results of using the M modifier with a comma as a delimiter. With the M modifier, leading, trailing, and multiple consecutive delimiters cause the CALL SCAN routine to return words that have a length of zero.

The O modifier is used for efficiency because the delimiters and modifiers are the same in every call to the CALL SCAN routine.

data comma;
  length word $30;
  string=',leading, trailing,and multiple,,delimiters, ,';
  do until(position <= 0);
    count + 1;
    call scan(string, count, position, length, ', ', 'mo');
    word=substrn(string, position, length);
    output;
  end;
run;
proc print data=comma noobs;
  var count position length word;
run;
Example 5: Using Comma-Separated Values, Substrings in Quotation Marks, and the O Modifier

The following example uses the CALL SCAN routine with the O modifier and a comma as a delimiter.

The O modifier is used for efficiency because in each call of the CALL SCAN routine, the delimiters, and modifiers do not change.

```sas
data test;
  length word word_r $30;
  string='He said, "She said, "No!"", not "Yes!"';
  do until(position <= 0);
    count + 1;
    call scan(string, count, position, length, ',', 'oq');
    word=substrn(string, position, length);
    output;
  end;
run;
```

```
proc print data=test noobs;
  var count position length word;
run;
```
Example 6: Finding Substrings of Digits By Using the D and K Modifiers

The following example finds substrings of digits. The character-list argument is null, and consequently the list of characters is initially empty. The D modifier adds digits to the list of characters. The K modifier treats all characters that are not in the list as delimiters. Therefore, all characters except digits are delimiters.

data digits;
  length digits $20;
  string='Call (800) 555-1234 now!';
  do until(position <= 0);
    count+1;
    call scan(string, count, position, length, , 'dko');
    digits=substrn(string, position, length);
    output;
  end;
run;
proc print data=digits noobs;
  var count position length digits;
run;

CALL SLEEP Routine

For a specified period of time, suspends the execution of a program that invokes this CALL routine.

**Category:** Special

**Note:** Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

**Syntax**

CALL SLEEP(n <, unit>);

**Required Argument**

\( n \)

is a numeric constant, variable, or expression that specifies the number of seconds for which you want to suspend execution of a program.

**Range** \( n \geq 0 \)

**Tips**

If you do not use the \( unit \) argument, the value of \( n \) is the next integer value that is less than \( n \).

If you use a fraction for the \( n \) argument, the \( unit \) argument is required if you want to suspend execution for a fraction of a second. For example, CALL SLEEP (.25); does not suspend execution. CALL SLEEP (1.25); suspends execution for 1 second. CALL SLEEP (1.25, 1); suspends execution for 1.25 seconds.

**Optional Argument**

\( unit \)

specifies the unit of time in seconds, which is applied to \( n \). For example, 1 corresponds to 1 second, .001 corresponds to 1 millisecond, and 5 corresponds to 5 seconds.

Default .001

**Details**

The CALL SLEEP routine suspends the execution of a program that invokes this CALL routine for a period of time that you specify. The program can be a DATA step, macro, IML, SCL, or anything that can invoke a CALL routine.
Examples

**Example 1: Suspending Execution for a Specified Period of Time**
The following example tells SAS to suspend the execution of the DATA step PAYROLL for 1 minute and 10 seconds:

```sas
data payroll;
   call sleep(7000, .01);
   ...more SAS statements...
run;
```

**Example 2: Suspending Execution Based on a Calculation of Sleep Time**
The following example tells SAS to suspend the execution of the DATA step BUDGET until March 1, 2013, at 3:00 AM. SAS calculates the length of the suspension based on the target date and the date and time that the DATA step begins to execute.

```sas
data budget;
   sleeptime='01mar2013:03:00'dt-datetime();
   call sleep(sleeptime, 1);
   ...more SAS statements...
run;
```

**Example 3: Using an Expression to Specify a Specific Period of Time**
The following examples show how to use an expression in the CALL SLEEP routine.

```sas
data _null_
   call sleep(.5+.8,1);
run;
```

The following log shows the results.

**Log 2.1 Using an Expression in the CALL SLEEP Routine**

```
NOTE: DATA statement used (Total process time):
  real time           1.30 seconds
  cpu time            0.00 seconds
```

See Also

Functions:
- “SLEEP Function” on page 608

---

**CALL STREAMINIT Routine**

Specifies a seed value to use for subsequent random number generation by the RAND function.

- **Category:** Random Number
- **Note:** Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.
Syntax

CALL STREAMINIT<(seed), 'TRNG'>;

Optional Arguments

seed
is an integer seed value.

Range  $seed < 2^{31} – 1$

Tip  If you specify a nonpositive seed, then CALL STREAMINIT is ignored. Any subsequent random number generation seeds itself from the system clock.

TRNG
selects the True Random Number Generator.

Restriction  The TRNG feature is available only on processors that support the RDRAND instructions: Ivy Bridge and later processors.

Note  When you specify the TRNG argument, all calls to the RAND function use the True Random Number Generator built into the Intel processor. If you specify no argument or a numeric argument, then the deterministic pseudo-random Mersenne Twister generator is used.

Details

General Information
If you want to create reproducible streams of random numbers, then specify CALL STREAMINIT before any calls to the RAND random number function. If you call the RAND function before you specify a seed with the CALL STREAMINIT routine (or if you specify a nonpositive seed value in the CALL STREAMINIT routine), then the RAND function uses a call to the system clock to seed itself. Each DATA step honors one CALL STREAMINIT seed. The prevailing seed value is the one that is specified immediately prior to the first RAND function call. For more information about seed values see “Seed Values” on page 11.

Windows Specific Information
You can access the True Random Number Generator by specifying the TRNG argument. When you specify TRNG, the following note is displayed if your processor provides RDRAND support:

NOTE: True Random Number Generator (TRNG) has been selected.
Nondeterministic behavior is expected.

If your processor does not provide RDRAND support, the following note is displayed:

ERROR: There is no hardware support on this system for the option "TRNG" in streaminit. The default setting will be used with a seed from the system clock.
Example: Creating a Reproducible Stream of Random Numbers

The following example shows how to specify a seed value with CALL STREAMINIT to create a reproducible stream of random numbers with the RAND function.

```sas
data random;
  call streaminit(123);
  do i=1 to 10;
    x1=rand('cauchy');
    output;
  end;
proc print data=random;
  id i;
run;
```

*Figure 2.8  Number String Seeded with CALL STREAMINIT*

**The SAS System**

<table>
<thead>
<tr>
<th>i</th>
<th>x1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-0.17593</td>
</tr>
<tr>
<td>2</td>
<td>3.76106</td>
</tr>
<tr>
<td>3</td>
<td>1.23427</td>
</tr>
<tr>
<td>4</td>
<td>0.49095</td>
</tr>
<tr>
<td>5</td>
<td>-0.05094</td>
</tr>
<tr>
<td>6</td>
<td>0.72496</td>
</tr>
<tr>
<td>7</td>
<td>-0.51646</td>
</tr>
<tr>
<td>8</td>
<td>7.61304</td>
</tr>
<tr>
<td>9</td>
<td>0.89784</td>
</tr>
<tr>
<td>10</td>
<td>1.69348</td>
</tr>
</tbody>
</table>

See Also

Functions:
- “RAND Function” on page 555

**CALL VNAME Routine**

Assigns a variable name as the value of a specified variable.
Category: Variable Control

Note: Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

Syntax

CALL VNAME(\textit{variable-1}, \textit{variable-2});

\textbf{Required Arguments}

\textit{variable-1}

specifies any SAS variable.

\textit{variable-2}

specifies any SAS character variable. Because SAS variable names can contain up to 32 characters, the length of \textit{variable-2} should be at least 32.

Details

The CALL VNAME routine assigns the name of the \textit{variable-1} variable as the value of the \textit{variable-2} variable.

Example: Using the CALL VNAME Routine

This example uses the CALL VNAME routine with array references to return the names of all variables in the data set OLD:

```sas
data new(keep=name);
  set old;
  /* all character variables in old */
  array abc{*} _character_
  /* all numeric variables in old */
  array def{*} _numeric_
  /* name is not in either array */
  length name $32;
  do i=1 to dim(abc);
    /* get name of character variable */
    call vname(abc{i}, name);
    /* write name to an observation */
    output;
  end;
  do j=1 to dim(def);
    /* get name of numeric variable */
    call vname(def{j}, name);
    /* write name to an observation */
    output;
  end;
  stop;
run;
```

See Also

Functions:
CAT Function

Does not remove leading or trailing blanks, and returns a concatenated character string.

**Category:** Character

**Restriction:** This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility.

**Tip:** DBCS equivalent function is KSTRCAT in SAS Viya National Language Support (NLS): Reference Guide.

### Syntax

```
CAT(item-1, ..., item-n)
```

### Required Argument

**item**

specifies a constant, variable, or expression, either character or numeric. If *item* is numeric, then its value is converted to a character string by using the BESTw. format. In this case, leading blanks are removed and SAS does not write a note to the log.

### Details

**Length of Returned Variable**

In a DATA step, if the CAT function returns a value to a variable that has not previously been assigned a length, then that variable is given a length of 200 bytes. If the concatenation operator (||) returns a value to a variable that has not previously been assigned a length, then that variable is given a length that is the sum of the lengths of the values that are being concatenated.

**Length of Returned Variable: Special Cases**

The CAT function returns a value to a variable, or returns a value in a temporary buffer. The value that is returned from the CAT function has the following length:

- up to 200 characters in WHERE clauses and in PROC SQL
- up to 32767 characters in the DATA step except in WHERE clauses
- up to 65534 characters when CAT is called from the macro processor

If CAT returns a value in a temporary buffer, the length of the buffer depends on the calling environment, and the value in the buffer can be truncated after CAT finishes processing. In this case, SAS does not write a message about the truncation to the log.

If the length of the variable or the buffer is not large enough to contain the result of the concatenation, SAS does the following:

- changes the result to a blank value in the DATA step, and in PROC SQL
writes a warning message to the log stating that the result was either truncated or set to a blank value, depending on the calling environment

- writes a note to the log that shows the location of the function call and lists the argument that caused the truncation

- sets _ERROR_ to 1 in the DATA step

The CAT function removes leading and trailing blanks from numeric arguments after it formats the numeric value with the BESTw. format.

**Comparisons**

The results of the CAT, CATS, CATT, and CATX functions are *usually* equivalent to results that are produced by certain combinations of the concatenation operator (||) and the TRIM and LEFT functions. However, the default length for the CAT, CATS, CATT, and CATX functions is different from the length that is obtained when you use the concatenation operator. For more information, see “Length of Returned Variable” on page 143.

Using the CAT, CATS, CATT, and CATX functions is faster than using TRIM and LEFT, and you can use them with the OF syntax for variable lists in calling environments that support variable lists.

The following table shows equivalents of the CAT, CATS, CATT, and CATX functions. The variables X1 through X4 specify character variables, and SP specifies a delimiter, such as a blank or comma.

<table>
<thead>
<tr>
<th>Function</th>
<th>Equivalent Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAT(OF X1-X4)</td>
<td>X1</td>
</tr>
<tr>
<td>CATS(OF X1-X4)</td>
<td>TRIM(LEFT(X1))</td>
</tr>
<tr>
<td>CATT(OF X1-X4)</td>
<td>TRIM(X1)</td>
</tr>
<tr>
<td>CATX(SP, OF X1-X4)</td>
<td>TRIM(LEFT(X1))</td>
</tr>
</tbody>
</table>

**Example**

The following example shows how the CAT function concatenates strings.

```sas
data _null_;
x=' The 2012 Olympic Arts Festival';
y='Val included works by Dale Chihuly.';
z='val included works by Dale Chihuly.';
a='ale Chihuly.';
result=cat(x, y, z, a);
p=put result $char.;
run;
```

SAS writes the following output to the log:

```
The 2012 Olympic Arts Festival included works by Dale Chihuly.
```
CATS Function

Removes leading and trailing blanks, and returns a concatenated character string.

**Category:** Character

**Restriction:** This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see [Internationalization Compatibility](#).

**Syntax**

```sas
CATS(item-1 <, ..., item-n>)
```

**Required Argument**

`item`

speifies a constant, variable, or expression, either character or numeric. If `item` is numeric, then its value is converted to a character string by using the `BESTw` format. In this case, SAS does not write a note to the log.

**Details**

**Length of Returned Variable**

In a DATA step, if the CATS function returns a value to a variable that has not previously been assigned a length, then that variable is given a length of 200 bytes. If the concatenation operator (||) returns a value to a variable that has not previously been assigned a length, then that variable is given a length that is the sum of the lengths of the values that are being concatenated.

**Length of Returned Variable: Special Cases**

The CATS function returns a value to a variable, or returns a value in a temporary buffer. The value that is returned from the CATS function has the following length:

- up to 200 characters in WHERE clauses and in PROC SQL
up to 32767 characters in the DATA step except in WHERE clauses
up to 65534 characters when CATS is called from the macro processor

If CATS returns a value in a temporary buffer, the length of the buffer depends on the calling environment, and the value in the buffer can be truncated after CATS finishes processing. In this case, SAS does not write a message about the truncation to the log.

If the length of the variable or the buffer is not large enough to contain the result of the concatenation, SAS does the following:

- changes the result to a blank value in the DATA step, and in PROC SQL
- writes a warning message to the log stating that the result was either truncated or set to a blank value, depending on the calling environment
- writes a note to the log that shows the location of the function call and lists the argument that caused the truncation
- sets _ERROR_ to 1 in the DATA step

The CATS function removes leading and trailing blanks from numeric arguments after it formats the numeric value with the BESTw. format.

Comparisons

The results of the CAT, CATS, CATT, and CATX functions are usually equivalent to results that are produced by certain combinations of the concatenation operator (||) and the TRIM and LEFT functions. However, the default length for the CAT, CATS, CATT, and CATX functions is different from the length that is obtained when you use the concatenation operator. For more information, see “Length of Returned Variable” on page 145.

Using the CAT, CATS, CATT, and CATX functions is faster than using TRIM and LEFT, and you can use them with the OF syntax for variable lists in calling environments that support variable lists.

The following table shows equivalents of the CAT, CATS, CATT, and CATX functions.

The variables X1 through X4 specify character variables, and SP specifies a delimiter, such as a blank or comma.

<table>
<thead>
<tr>
<th>Function</th>
<th>Equivalent Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAT(OF X1-X4)</td>
<td>X1</td>
</tr>
<tr>
<td>CATS(OF X1-X4)</td>
<td>TRIM(LEFT(X1))</td>
</tr>
<tr>
<td>CATT(OF X1-X4)</td>
<td>TRIM(X1)</td>
</tr>
<tr>
<td>CATX(SP, OF X1-X4)</td>
<td>TRIM(LEFT(X1))</td>
</tr>
</tbody>
</table>

Example

The following example shows how the CATS function concatenates strings.

data _null_
  x=' The Olym'
  y='pic Arts Festi'

z='val includes works by D';
a='ale Chihuly.;'
result=cats(x, y, z, a);
put result $char.;
run;

SAS writes the following output to the log:

The Olympic Arts Festival includes works by Dale Chihuly.

See Also

Functions:
- “CAT Function” on page 143
- “CATQ Function” on page 846
- “CATT Function” on page 147
- “CATX Function” on page 149

CALL Routines:
- “CALL CATS Routine” on page 736
- “CALL CATT Routine” on page 738
- “CALL CATX Routine” on page 740

CATT Function

Removes trailing blanks, and returns a concatenated character string.

Category: Character
Restriction: This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility.

Syntax
CATT(item-1 <, ... item-n>)

Required Argument
item
specifies a constant, variable, or expression, either character or numeric. If item is numeric, then its value is converted to a character string by using the BESTw. format. In this case, leading blanks are removed and SAS does not write a note to the log.
Details

Length of Returned Variable
In a DATA step, if the CATT function returns a value to a variable that has not previously been assigned a length, then that variable is given a length of 200 bytes. If the concatenation operator (||) returns a value to a variable that has not previously been assigned a length, then that variable is given a length that is the sum of the lengths of the values that are being concatenated.

Length of Returned Variable: Special Cases
The CATT function returns a value to a variable, or returns a value in a temporary buffer. The value that is returned from the CATT function has the following length:

- up to 200 characters in WHERE clauses and in PROC SQL
- up to 32767 characters in the DATA step except in WHERE clauses
- up to 65534 characters when CATT is called from the macro processor

If CATT returns a value in a temporary buffer, the length of the buffer depends on the calling environment, and the value in the buffer can be truncated after CATT finishes processing. In this case, SAS does not write a message about the truncation to the log. If the length of the variable or the buffer is not large enough to contain the result of the concatenation, SAS does the following:

- changes the result to a blank value in the DATA step, and in PROC SQL
- writes a warning message to the log stating that the result was either truncated or set to a blank value, depending on the calling environment
- writes a note to the log that shows the location of the function call and lists the argument that caused the truncation
- sets _ERROR_ to 1 in the DATA step

The CATT function removes leading and trailing blanks from numeric arguments after it formats the numeric value with the BESTw. format.

Comparisons
The results of the CAT, CATS, CATT, and CATX functions are usually equivalent to results that are produced by certain combinations of the concatenation operator (||) and the TRIM and LEFT functions. However, the default length for the CAT, CATS, CATT, and CATX functions is different from the length that is obtained when you use the concatenation operator. For more information, see “Length of Returned Variable” on page 148.

Using the CAT, CATS, CATT, and CATX functions is faster than using TRIM and LEFT, and you can use them with the OF syntax for variable lists in calling environments that support variable lists.

The following table shows equivalents of the CAT, CATS, CATT, and CATX functions. The variables X1 through X4 specify character variables, and SP specifies a delimiter, such as a blank or comma.

<table>
<thead>
<tr>
<th>Function</th>
<th>Equivalent Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAT(OF X1–X4)</td>
<td>X1</td>
</tr>
</tbody>
</table>

Chapter 2 • Dictionary of Functions and Call Routines for SAS and the CAS Server
### Function

<table>
<thead>
<tr>
<th>Function</th>
<th>Equivalent Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>CATS(OF X1-X4)</td>
<td>TRIM(LEFT(X1))</td>
</tr>
<tr>
<td>CATT(OF X1-X4)</td>
<td>TRIM(X1)</td>
</tr>
<tr>
<td>CATX(SP, OF X1-X4)</td>
<td>TRIM(LEFT(X1))</td>
</tr>
</tbody>
</table>

### Example

The following example shows how the CATT function concatenates strings.

```sas
data _null_;  
x=' The Olym';  
y='pic Arts Festi';  
z=' val includes works by D ';  
a='ale Chihuly.';  
result=catt(x, y, z, a);  
put result $char.;  
run;
```

SAS writes the following output to the log:

```
The Olympic Arts Festival includes works by Dale Chihuly.
```

### See Also

**Functions:**
- “CAT Function” on page 143
- “CATQ Function” on page 846
- “CATS Function” on page 145
- “CATX Function” on page 149

**CALL Routines:**
- “CALL CATS Routine” on page 736
- “CALL CATT Routine” on page 738
- “CALL CATX Routine” on page 740

---

**CATX Function**

Removes leading and trailing blanks, inserts delimiters, and returns a concatenated character string.

**Category:** Character

**Restriction:** This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see [Internationalization Compatibility](#).
Syntax

\texttt{CATX(delimiter, item-1, \ldots, item-n)}

\textbf{Required Arguments}

\textit{delimiter}

specifies a character string that is used as a delimiter between concatenated items.

\textit{item}

specifies a constant, variable, or expression, either character or numeric. If \textit{item} is numeric, then its value is converted to a character string by using the \texttt{BESTw}. format. In this case, SAS does not write a note to the log. For more information, see “The Basics” on page 150.

\textbf{Details}

\textbf{The Basics}

The CATX function first copies \textit{item-1} to the result, omitting leading and trailing blanks. Then for each subsequent argument \textit{item-i}, \(i=2, \ldots, n\), if \textit{item-i} contains at least one non-blank character, then CATX appends \texttt{delimiter} and \textit{item-i} to the result, omitting leading and trailing blanks from \textit{item-i}. CATX does not insert the delimiter at the beginning or end of the result. Blank items do not produce delimiters at the beginning or end of the result, nor do blank items produce multiple consecutive delimiters.

Some language elements are supported in SAS but not in CAS. An example is the DATALINES statement. Some examples in this document use DATALINES in SAS to create a data set using the CAS engine. For more information, see “Functions in SAS and CAS” on page 54.

\textbf{Length of Returned Variable}

In a DATA step, if the CATX function returns a value to a variable that has not previously been assigned a length, then that variable is given a length of 200 bytes. If the concatenation operator (||) returns a value to a variable that has not previously been assigned a length, then that variable is given a length that is the sum of the lengths of the values that are being concatenated.

\textbf{Length of Returned Variable: Special Cases}

The CATX function returns a value to a variable, or returns a value in a temporary buffer. The value that is returned from the CATX function has the following length:

- up to 200 characters in WHERE clauses and in PROC SQL
- up to 32767 characters in the DATA step except in WHERE clauses
- up to 65534 characters when CATX is called from the macro processor

If CATX returns a value in a temporary buffer, the length of the buffer depends on the calling environment, and the value in the buffer can be truncated after CATX finishes processing. In this case, SAS does not write a message about the truncation to the log.

If the length of the variable or the buffer is not large enough to contain the result of the concatenation, SAS does the following:

- changes the result to a blank value in the DATA step, and in PROC SQL
• writes a warning message to the log stating that the result was either truncated or set to a blank value, depending on the calling environment

• writes a note to the log that shows the location of the function call and lists the argument that caused the truncation

• sets _ERROR_ to 1 in the DATA step

Comparisons

The results of the CAT, CATS, CATT, and CATX functions are usually equivalent to results that are produced by certain combinations of the concatenation operator (||) and the TRIM and LEFT functions. However, the default length for the CAT, CATS, CATT, and CATX functions is different from the length that is obtained when you use the concatenation operator. For more information, see “Length of Returned Variable” on page 150.

Using the CAT, CATS, CATT, and CATX functions is faster than using TRIM and LEFT, and you can use them with the OF syntax for variable lists in calling environments that support variable lists.

Note: In the case of variables that have missing values, the concatenation produces different results. See “Example 2: Concatenating Strings That Have Missing Values” on page 152.

The following table shows equivalents of the CAT, CATS, CATT, and CATX functions. The variables X1 through X4 specify character variables, and SP specifies a delimiter, such as a blank or comma.

<table>
<thead>
<tr>
<th>Function</th>
<th>Equivalent Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAT(OF X1-X4)</td>
<td>X1</td>
</tr>
<tr>
<td>CATS(OF X1-X4)</td>
<td>TRIM(LEFT(X1))</td>
</tr>
<tr>
<td>CATT(OF X1-X4)</td>
<td>TRIM(X1)</td>
</tr>
<tr>
<td>CATX(SP, OF X1-X4)</td>
<td>TRIM(LEFT(X1))</td>
</tr>
</tbody>
</table>

Examples

Example 1: Concatenating Strings That Have No Missing Values

The following example shows how the CATX function concatenates strings the have no missing values.

data _null_;  
  separator='%%$%%';  
  x='The Olympic ';  
  y=' Arts Festival ';  
  z=' includes works by ';  
  a='Dale Chihuly.';  
  result=catx(separator, x, y, z, a);  
  put result $char.;  
run;
SAS writes the following output to the log:

```
The Olympic Arts Festival includes works by Dale Chihuly.
```

**Example 2: Concatenating Strings That Have Missing Values**

The following example shows how the CATX function concatenates strings that contain missing values.

```sas
data one;
    length x1-x4 $1;
    input x1-x4;
    datalines;
    A B C D
    E . F G
    H . . J
    ;
run;
data two;
    set one;
    SP='^';
    test1=catx(sp, of x1-x4);
    test2=trim(left(x1)) || sp || trim(left(x2)) || sp || trim(left(x3)) || sp || trim(left(x4));
run;
proc print data=two;
run;
```

**Figure 2.9 Using CATX with Missing Values**

<table>
<thead>
<tr>
<th>Obs</th>
<th>x1</th>
<th>x2</th>
<th>x3</th>
<th>x4</th>
<th>SP</th>
<th>test1</th>
<th>test2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>^</td>
<td>A^B^C^D</td>
<td>A^B^C^D</td>
</tr>
<tr>
<td>2</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>^</td>
<td>sp</td>
<td>E^F^G</td>
<td>E^F^G</td>
</tr>
<tr>
<td>3</td>
<td>H</td>
<td>J</td>
<td>^</td>
<td></td>
<td>sp</td>
<td>H^J</td>
<td>H^J</td>
</tr>
</tbody>
</table>

**See Also**

**Functions:**
- “CAT Function” on page 143
- “CATQ Function” on page 846
- “CATS Function” on page 145
- “CATT Function” on page 147

**CALL Routines:**
- “CALL CATS Routine” on page 736
CDF Function

Returns a value from a cumulative probability distribution.

**Category:** Probability

**Note:** The QUANTILE function returns the quantile from a distribution that you specify. The QUANTILE function is the inverse of the CDF function. For more information, see “QUANTILE Function” on page 549.

**Syntax**

CDF(*distribution*, *quantile* <, *parameter-1*, ..., *parameter-k*>)

**Required Arguments**

distribution

is a character constant, variable, or expression that identifies the distribution. Valid distributions are as follows:

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bernoulli</td>
<td>BERNOULLI</td>
</tr>
<tr>
<td>Beta</td>
<td>BETA</td>
</tr>
<tr>
<td>Binomial</td>
<td>BINOMIAL</td>
</tr>
<tr>
<td>Cauchy</td>
<td>CAUCHY</td>
</tr>
<tr>
<td>Chi-Square</td>
<td>CHISQUARE</td>
</tr>
<tr>
<td>Conway-Maxwell-Poisson</td>
<td>CONMAXPOI</td>
</tr>
<tr>
<td>Exponential</td>
<td>EXPONENTIAL</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>Gamma</td>
<td>GAMMA</td>
</tr>
<tr>
<td>Generalized Poisson</td>
<td>GENPOISSON</td>
</tr>
<tr>
<td>Geometric</td>
<td>GEOMETRIC</td>
</tr>
<tr>
<td>Hypergeometric</td>
<td>HYPERGEOMETRIC</td>
</tr>
<tr>
<td>Laplace</td>
<td>LAPLACE</td>
</tr>
</tbody>
</table>
### Distribution

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logistic</td>
<td>LOGISTIC</td>
</tr>
<tr>
<td>Lognormal</td>
<td>LOGNORMAL</td>
</tr>
<tr>
<td>Negative binomial</td>
<td>NEGBINOMIAL</td>
</tr>
<tr>
<td>Normal</td>
<td>NORMAL</td>
</tr>
<tr>
<td>Normal mixture</td>
<td>NORMALMIX</td>
</tr>
<tr>
<td>Pareto</td>
<td>PARETO</td>
</tr>
<tr>
<td>Poisson</td>
<td>POISSON</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>Tweedie</td>
<td>TWEEDIE</td>
</tr>
<tr>
<td>Uniform</td>
<td>UNIFORM</td>
</tr>
<tr>
<td>Wald (inverse Gaussian)</td>
<td>WALD</td>
</tr>
<tr>
<td>Weibull</td>
<td>WEIBULL</td>
</tr>
</tbody>
</table>

**Note**  
Except for T, F, and NORMALMIX, you can minimally identify any distribution by its first four characters.

**quantile**

is a numeric constant, variable, or expression that specifies the value of the random variable.

### Optional Argument

*parameter-1, …, parameter-k*

are optional constants, variables, or expressions that specify the values of *shape*, *location*, or *scale* parameters that are appropriate for the specific distribution.

See “Details” on page 154 for complete information about these parameters.

### Details

The CDF function computes the left cumulative distribution function from various continuous and discrete probability distributions.

**Bernoulli Distribution**

CDF('BERNOULLI', x, p)

### Arguments
is a numeric constant, variable, or expression that specifies a random variable.

\( p \)
is a numeric constant, variable, or expression that specifies a probability of success.

Range \( 0 \leq p \leq 1 \)

Details

The CDF function for the Bernoulli distribution returns the probability that an observation from a Bernoulli distribution, with probability of success equal to \( p \), is less than or equal to \( x \). The equation follows:

\[
CDF(BERN, x, p) = \begin{cases} 
0 & x < 0 \\
1 - p & 0 \leq x < 1 \\
1 & x \geq 1 
\end{cases}
\]

Note: There are no location or scale parameters for this distribution.

Beta Distribution

CDF('BETA', \( x, a, b <, l, r> \))

Arguments

\( x \)
is a numeric constant, variable, or expression that specifies a random variable.

\( a \)
is a numeric constant, variable, or expression that specifies a shape parameter.

Range \( a > 0 \)

\( b \)
is a numeric constant, variable, or expression that specifies a shape parameter.

Range \( b > 0 \)

\( l \)
is a numeric constant, variable, or expression that specifies the left location parameter.

Default 0

\( r \)
is a numeric constant, variable, or expression that specifies the right location parameter.

Default 1

Range \( r > l \)

Details

The CDF function for the beta distribution returns the probability that an observation from a beta distribution, with shape parameters \( a \) and \( b \), is less than or equal to \( v \). The following equation describes the CDF function of the beta distribution:
CDF('BETA', x, a, b, l, r) = \begin{cases} 
0 & x \leq l \\
\frac{1}{\beta(a, b)} \int_{l}^{x} \frac{(y-b)^{a-1}(r-y)^{b-1}}{(r-y)^{a+b-1}} \, dy & l < x \leq r \\
1 & x > r 
\end{cases}

The following relationship applies to the preceding equation:

\beta(a, b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}

The following relationship applies to the preceding equation:

\Gamma(a) = \int_{0}^{\infty} x^{a-1}e^{-x} \, dx

**Binomial Distribution**

CDF('BINOMIAL', m, p, n)

**Arguments**

- **m**
  - is an integer random variable that counts the number of successes.
  - Range: \( m = 0, 1, ... \)

- **p**
  - is a numeric constant, variable, or expression that specifies a probability of success parameter.
  - Range: \( 0 \leq p \leq 1 \)

- **n**
  - is a numeric constant, variable, or expression that specifies an integer parameter that counts the number of independent Bernoulli trials.
  - Range: \( n = 0, 1, ... \)

**Details**

The CDF function for the binomial distribution returns the probability that an observation from a binomial distribution, with parameters \( p \) and \( n \), is less than or equal to \( m \). The equation follows:

\[
CDF('BINOM', m, p, n) = \begin{cases} 
0 & m < 0 \\
\sum_{j=0}^{m} \binom{n}{j} p^{j} (1 - p)^{n-j} & 0 \leq m \leq n \\
1 & m > n 
\end{cases}
\]

**Note:** There are no location or scale parameters for the binomial distribution.

**Cauchy Distribution**

CDF('CAUCHY', x <,\beta>, <,\lambda>)

**Arguments**

- **x**
  - is a numeric constant, variable, or expression that specifies a random variable.
θ is a numeric constant, variable, or expression that specifies a location parameter.

Default 0

λ is a numeric constant, variable, or expression that specifies a scale parameter.

Default 1

Range λ > 0

Details

The CDF function for the Cauchy distribution returns the probability that an observation from a Cauchy distribution, with the location parameter θ and the scale parameter λ, is less than or equal to x. The equation follows:

$$CDF(\text{CAUCHY}', x, \theta, \lambda) = \frac{1}{2} + \frac{1}{\pi} \tan^{-1}\left(\frac{x - \theta}{\lambda}\right)$$

Chi-Square Distribution

CDF(‘CHISQUARE’, x, df <, nc>)

Arguments

x is a numeric constant, variable, or expression that specifies a random variable.

df is a numeric constant, variable, or expression that specifies a degrees of freedom parameter.

Range df > 0

nc is a numeric constant, variable, or expression that specifies an optional noncentrality parameter.

Range nc ≥ 0

Details

The CDF function for the chi-square distribution returns the probability that an observation from a chi-square distribution, with df degrees of freedom and the noncentrality parameter nc, is less than or equal to x. This function accepts non-integer degrees of freedom. If nc is omitted or equal to zero, the value returned is from the central chi-square distribution. In the following equation, let ν = df and let λ = nc. The following equation describes the CDF function of the chi-square distribution:

$$CDF(\text{CHISQ}', x, \nu, \lambda) = \begin{cases} 0 & x < 0 \\ \sum_{j=0}^{\infty} e^{-\frac{\lambda}{2}} \left(\frac{\lambda}{2}\right)^j j! P_{c}(x, \nu + 2j) & x \geq 0 \end{cases}$$

In the equation, $P_{c}(..)$ denotes the probability from the central chi-square distribution:

$$P_{c}(x, \alpha) = P_{\alpha}\left(\frac{x}{\frac{\alpha}{2}}, \frac{\alpha}{2}\right)$$
In the equation, \( P_g(y, b) \) is the probability from the gamma distribution given by the equation:

\[
P_g(y, b) = \frac{y^b}{\Gamma(b)} \int_0^y e^{-v} v^{b-1} dv
\]

**Conway-Maxwell-Poisson Distribution**

\( \text{CDF}(\text{CONMAXPOI}, y, \lambda, \nu) \)

**Arguments**

- \( y \) is a numeric constant, variable, or expression that specifies a nonnegative integer that represents counts data.
- \( \lambda \) is similar to the mean, as in the Poisson distribution.
- \( \nu \) is a numeric constant, variable, or expression that specifies a dispersion parameter.

**Details**

The CDF function returns cumulative probability from 0 to \( y \). For more information, see “Conway-Maxwell-Poisson” distribution in the PDF function on page 482.

**Exponential Distribution**

\( \text{CDF}(\text{EXPONENTIAL}, x, \lambda) \)

**Arguments**

- \( x \) is a numeric constant, variable, or expression that specifies a random variable.
- \( \lambda \) is a numeric constant, variable, or expression that specifies a scale parameter.

**Default** 1

**Range** \( \lambda > 0 \)

**Details**

The CDF function for the exponential distribution returns the probability that an observation from an exponential distribution, with the scale parameter \( \lambda \), is less than or equal to \( x \). The equation follows:

\[
\text{CDF}(\text{EXPONENTIAL}, x, \lambda) = \begin{cases} 
0 & x < 0 \\
\frac{x}{\lambda} & x \geq 0
\end{cases}
\]

**F Distribution**

\( \text{CDF}(\text{F}, x, ndf, ddf, nc) \)

**Arguments**

- \( x \) is a numeric constant, variable, or expression that specifies a random variable.
\( ndf \)

is a numeric constant, variable, or expression that specifies a numerator degrees of freedom parameter.

Range \( ndf > 0 \)

\( ddf \)

is a numeric constant, variable, or expression that specifies a denominator degrees of freedom parameter.

Range \( ddf > 0 \)

\( nc \)

is a numeric constant, variable, or expression that specifies a noncentrality parameter.

Range \( nc \geq 0 \)

**Details**

The CDF function for the \( F \) distribution returns the probability that an observation from an \( F \) distribution, with \( ndf \) numerator degrees of freedom, \( ddf \) denominator degrees of freedom, and the noncentrality parameter \( nc \), is less than or equal to \( x \). This function accepts noninteger degrees of freedom for \( ndf \) and \( ddf \). If \( nc \) is omitted or equal to zero, the value returned is from a central \( F \) distribution. In the following equation, let \( v_1 = ndf \), let \( v_2 = ddf \), and let \( \lambda = nc \). The following equation describes the CDF function of the \( F \) distribution:

\[
CDF(F, x, v_1, v_2, \lambda) = \begin{cases} 
0 & \text{if } x < 0 \\
\sum_{j=0}^{\infty} e^{-\frac{\lambda}{2}} \frac{(\frac{\lambda}{2})^j}{j!} P_F(x, v_1 + 2j, v_2) & \text{if } x \geq 0 
\end{cases}
\]

In the equation, \( P_F(x, u_1, u_2) \) is the probability from the central \( F \) distribution with

\[
P_F(x, u_1, u_2) = P_B\left(\frac{u_1 x}{u_1 x + u_2}, \frac{u_1}{2}, \frac{u_2}{2}\right)
\]

and \( P_B(x,a,b) \) is the probability from the standard beta distribution.

*Note:* There are no location or scale parameters for the \( F \) distribution.

**Gamma Distribution**

CDF('GAMMA', \( x, a <, \lambda > \))

**Arguments**

\( x \)

is a numeric constant, variable, or expression that specifies a random variable.

\( a \)

is a numeric constant, variable, or expression that specifies a shape parameter.

Range \( a > 0 \)

\( \lambda \)

is a numeric constant, variable, or expression that specifies a scale parameter.
Default 1

Range $\lambda > 0$

Details
The CDF function for the gamma distribution returns the probability that an observation from a gamma distribution, with the shape parameter $a$ and the scale parameter $\lambda$, is less than or equal to $x$. The equation follows:

$$CDF('GAMMA', x, a, \lambda) = \begin{cases} 0 & x < 0 \\ \frac{1}{x^a F(a)} \int_0^x v^{a-1} e^{-\frac{v}{\lambda}} dv & x \geq 0 \end{cases}$$

**Generalized Poisson Distribution**

$CDF('GENPOISSON', x, \theta, \eta)$

Arguments

$x$

is a numeric constant, variable, or expression that specifies an integer random variable.

$\theta$

is a numeric constant, variable, or expression that specifies a shape parameter.

Range $\leq 5$ and $>0$

$\eta$

is a numeric constant, variable, or expression that specifies a shape parameter.

Range $\geq 0$ and $<0.95$

Tip When $\eta =0$, the distribution is the Poisson distribution with a mean and variance of $\theta$. When $\eta>0$, the mean is $\theta \div (1 - \eta)$ and the variance is $\theta \div (1 - \eta)^2$.

Details
The probability mass function for the generalized Poisson distribution follows:

$$f(x; \theta, \eta) = \theta(\theta + \eta x)^{x-1} e^{-\theta - \eta x} / x! \, , \, x = 0, 1, 2, ... , \, \theta > 0, \, 0 \leq \eta < 1$$

If $\eta =0$, then the generalized Poisson distribution becomes the standard Poisson distribution with the shape parameter $\theta$.

**Geometric Distribution**

$CDF('GEOMETRIC', m, p)$

Arguments

$m$

is a numeric random variable that specifies the number of failures.

Range $m = 0, 1, ...$
\( p \)

is a numeric constant, variable, or expression that specifies a probability of success.

Range \( 0 \leq p \leq 1 \)

Details

The CDF function for the geometric distribution returns the probability that an observation from a geometric distribution, with the parameter \( p \), is less than or equal to \( m \). The equation follows:

\[
CDF('GEOM', m, p) = \begin{cases} 
0 & m < 0 \\
1 - (1 - p)^{(m + 1)} & m \geq 0
\end{cases}
\]

Note: There are no location or scale parameters for this distribution.

**Hypergeometric Distribution**

CDF('HYPER', \( x \), \( N \), \( R \), \( n < \), \( o > \))

Arguments

\( x \)

is a numeric constant, variable, or expression that specifies an integer random variable.

\( N \)

is a numeric constant, variable, or expression that specifies an integer population size parameter.

Range \( N = 1, 2, \ldots \)

\( R \)

is a numeric constant, variable, or expression that specifies an integer number of items in the category of interest.

Range \( R = 0, 1, \ldots, N \)

\( n \)

is a numeric constant, variable, or expression that specifies an integer sample size parameter.

Range \( n = 1, 2, \ldots, N \)

\( o \)

is a numeric constant, variable, or expression that specifies an optional numeric odds ratio parameter.

Range \( o > 0 \)

Details

The CDF function for the hypergeometric distribution returns the probability that an observation from an extended hypergeometric distribution, with population size \( N \), number of items \( R \), sample size \( n \), and odds ratio \( o \), is less than or equal to \( x \). If \( o \) is omitted or equal to 1, the value returned is from the usual hypergeometric distribution. The equation follows:
CDF ('HYPER', x, N, R, n, o) =
\[\begin{align*}
0 & \quad x < \text{max}(0, R + n - N) \\
\sum_{i=0}^{N} \frac{(N - R)}{n - i} x^i & \quad \text{max}(0, R + n - N) \leq x \leq \min(R, n) \\
1 & \quad x > \min(R, n)
\end{align*}\]

**Laplace Distribution**

CDF('LAPLACE', x < \theta, \lambda>)

**Arguments**

\( x \)

is a numeric constant, variable, or expression that specifies a random variable.

\( \theta \)

is a numeric constant, variable, or expression that specifies a location parameter.

Default 0

\( \lambda \)

is a numeric constant, variable, or expression that specifies a scale parameter.

Default 1

Range \( \lambda > 0 \)

**Details**

The CDF function for the Laplace distribution returns the probability that an observation from the Laplace distribution, with the location parameter \( \theta \) and the scale parameter \( \lambda \), is less than or equal to \( x \). The equation follows:

\[
CDF('LAPLACE', x, \theta, \lambda) = \begin{cases} 
\frac{1}{2} e^{\frac{(x - \theta)}{\lambda}} & x < \theta \\
1 - \frac{1}{2} e^{\frac{- (x - \theta)}{\lambda}} & x \geq \theta 
\end{cases}
\]

**Logistic Distribution**

CDF('LOGISTIC', x < \theta, \lambda>)

**Arguments**

\( x \)

is a numeric constant, variable, or expression that specifies a random variable.

\( \theta \)

is a numeric constant, variable, or expression that specifies a location parameter.

Default 0

\( \lambda \)

is a numeric constant, variable, or expression that specifies a scale parameter.

Default 1
Range $\lambda > 0$

Details

The CDF function for the Logistic distribution returns the probability that an observation from a Logistic distribution, with the location parameter $\theta$ and the scale parameter $\lambda$, is less than or equal to $x$. The equation follows:

$$CDF('LOGISTIC', x, \theta, \lambda) = \frac{1}{1 + e^{\left(-\frac{x-\theta}{\lambda}\right)}}$$

Lognormal Distribution

CDF('LOGNORMAL', $x < \theta, \lambda)$

Arguments

$x$

is a numeric constant, variable, or expression that specifies a random variable.

$\theta$

is a numeric constant, variable, or expression that specifies a log scale parameter.

e($\theta$) is a scale parameter.

Default 0

$\lambda$

is a numeric constant, variable, or expression that specifies a shape parameter.

Default 1

Range $\lambda > 0$

Details

The CDF function for the lognormal distribution returns the probability that an observation from a lognormal distribution, with the log scale parameter $\theta$ and the shape parameter $\lambda$, is less than or equal to $x$. The equation follows:

$$CDF('LOGN', x, \theta, \lambda) = \begin{cases} 0 & x \leq 0 \\ \frac{1}{\lambda \sqrt{2\pi}} \int_{-\infty}^{x} e^{\left(-\frac{(\log(x) - \theta)^2}{2\lambda^2}\right)} \, dv & x > 0 \end{cases}$$

Negative Binomial Distribution

CDF('NEGBINOMIAL', $m, p, n$)

Arguments

$m$

is a numeric constant, variable, or expression that specifies a positive integer random variable that counts the number of failures.

Range $m = 0, 1, ...$

$p$

is a numeric constant, variable, or expression that specifies a probability of success.
Range \( 0 \leq p \leq 1 \)

\( n \)
is a numeric constant, variable, or expression that specifies a value that counts the number of successes.

Range \( n > 0 \)

Details

The CDF function for the negative binomial distribution returns the probability that an observation from a negative binomial distribution, with the probability of success \( p \) and the number of successes \( n \), is less than or equal to \( m \). The equation follows:

\[
CDF('NEGB', m, p, n) = \begin{cases} 
0 & m < 0 \\
p^n \sum_{j=0}^{m} \frac{(n + j - 1)!}{n! (j)!} (1 - p)^j & m \geq 0 
\end{cases}
\]

Note: There are no location or scale parameters for the negative binomial distribution.

**Normal Distribution**

**CDF('NORMAL', x, \theta, \lambda)\)**

**Arguments**

\( x \)
is a numeric constant, variable, or expression that specifies a random variable.

\( \theta \)
is a numeric constant, variable, or expression that specifies a location parameter.

Default \( 0 \)

\( \lambda \)
is a numeric constant, variable, or expression that specifies a scale parameter.

Default \( 1 \)

Range \( \lambda > 0 \)

Details

The CDF function for the Normal distribution returns the probability that an observation from the Normal distribution, with the location parameter \( \theta \) and the scale parameter \( \lambda \), is less than or equal to \( x \). The equation follows:

\[
CDF('NORMAL', x, \theta, \lambda) = \frac{1}{\lambda \sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{(v-\theta)^2}{2\lambda^2}} \, dv
\]

**Normal Mixture Distribution**

**CDF('NORMALMIX', x, n, p, m, s)\)**

**Arguments**

\( x \)
is a numeric constant, variable, or expression that specifies a random variable.
\( n \)
is a numeric constant, variable, or expression that specifies the number of mixtures.
Range \( n = 1, 2, \ldots \)

\( p \)
is a numeric constant, variable, or expression that specifies the \( n \) proportions,
\( p_1, p_2, \ldots, p_n \), where \( \sum_{i=1}^{i=n} p_i = 1 \).
Range \( p = 0, 1, \ldots \)

\( m \)
is a numeric constant, variable, or expression that specifies the \( n \) means
\( m_1, m_2, \ldots, m_n \).

\( s \)
is a numeric constant, variable, or expression that specifies the \( n \) standard deviations
\( s_1, s_2, \ldots, s_n \).
Range \( s > 0 \)

Details
The CDF function for the Normal Mixture distribution returns the probability that an observation from a mixture of normal distribution is less than or equal to \( x \). The equation follows:
\[
CDF('NORMALMIX', x, n, p, m, s) = \sum_{i=1}^{i=n} p_i CDF('NORMAL', x, m_i, s_i)
\]

Weights for the Normal Mixture distribution must be nonnegative. If the sum of the weights does not equal 1, then the weights are treated as relative weights and adjusted so that the sum equals 1.

Note: There are no location or scale parameters for the Normal Mixture distribution.

\textbf{Pareto Distribution}
\texttt{CDF('PARETO', x, a < k>)}

Arguments
\( x \)
is a numeric constant, variable, or expression that specifies a random variable.

\( a \)
is a numeric constant, variable, or expression that specifies a shape parameter.
Range \( a > 0 \)

\( k \)
is a numeric constant, variable, or expression that specifies a scale parameter.
Default \( 1 \)
Range \( k > 0 \)

Details
The CDF function for the Pareto distribution returns the probability that an observation from a Pareto distribution, with the shape parameter \( a \) and the scale parameter \( k \), is less than or equal to \( x \). The equation follows:

\[
CDF('PARETO', x, a, k) = \begin{cases} 
0 & x < k \\
1 - \left(\frac{k}{x}\right)^a & x \geq k 
\end{cases}
\]

**Poisson Distribution**

\[
CDF('POISSON', n, m)
\]

**Arguments**

\( n \)

is a numeric constant, variable, or expression that specifies an integer random variable.

Range \( n = 0, 1, ... \)

\( m \)

is a numeric constant, variable, or expression that specifies a mean parameter.

Range \( m > 0 \)

**Details**

The CDF function for the Poisson distribution returns the probability that an observation from a Poisson distribution, with mean \( m \), is less than or equal to \( n \). The equation follows:

\[
CDF('POISSON', n, m) = \begin{cases} 
0 & n < 0 \\
\sum_{i=0}^{n} e^{-m} \frac{m^i}{i!} & n \geq 0 
\end{cases}
\]

**Note:** There are no location or scale parameters for the Poisson distribution.

**T Distribution**

\[
CDF('T', t, df <, nc>)
\]

**Arguments**

\( t \)

is a numeric constant, variable, or expression that specifies a random variable.

\( df \)

is a numeric constant, variable, or expression that specifies the degrees of freedom.

Range \( df > 0 \)

\( nc \)

is a numeric constant, variable, or expression that specifies an optional noncentrality parameter.

**Details**

The CDF function for the \( T \) distribution returns the probability that an observation from a \( T \) distribution, with degrees of freedom \( df \) and the noncentrality parameter \( nc \), is less
than or equal to \( x \). This function accepts noninteger degrees of freedom. If \( nc \) is omitted or equal to zero, the value returned is from the central \( T \) distribution. In the following equation, let \( \nu = df \) and let \( \delta = nc \). The equation follows:

\[
CDF(T', t, \nu, \delta) = \frac{1}{2^{(\nu/2)-1}T(\frac{\nu}{2})} \int_0^\infty x^{\nu-1} e^{-\frac{x^2}{2}} \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\nu} e^{-\frac{1}{2}(u-\delta)^2} du dx
\]

Note: There are no location or scale parameters for the \( T \) distribution.

**Tweedie Distribution**

CDF (‘TWEEDIE’, \( y, p <, \mu, \phi > \))

Arguments

\( y \)

is a numeric constant, variable, or expression that specifies a random variable.

Range \( y \geq 0 \)

Notes

This argument is required.

When \( p > 1 \), \( y \) is numeric. When \( p = 1 \), \( y \) is an integer.

\( p \)

is a numeric constant, variable, or expression that specifies the power parameter.

Range \( p \geq 1 \)

Note

This argument is required.

\( \mu \)

is a numeric constant, variable, or expression that specifies the mean parameter.

Default 1

Range \( \mu > 0 \)

\( \phi \)

is a numeric constant, variable, or expression that specifies the dispersion parameter.

Default 1

Range \( \phi > 0 \)

Details

The CDF function for the Tweedie distribution returns an exponential dispersion model with variance and mean related by the equation \( \text{variance} = \phi \times \mu^p \).

The equation follows:

\[
\int_0^\infty \left( \sum_{j=1}^y \frac{y^{-ja}(p-1)^ja}{\phi^j j! (1-a)^2j^2 \Gamma(ja)} \right) e^{\left( \frac{1}{\phi} \left( \frac{\mu^{1-p}-1}{1-p} - \frac{\mu^{2-p}-1}{2-p} \right) \right)} dy
\]

The following relationship applies to the preceding equation:
\[ \alpha = \frac{2 - p}{1 - p} \]

Note: The accuracy of computed Tweedie probabilities is highly dependent on the location in parameter space. Ten digits of accuracy are usually available except when \( p \) is near 2 or phi is near 0. In that case, the accuracy might be as low as six digits.

**Uniform Distribution**

CDF('UNIFORM', \( x < l, r > \))

**Arguments**

\( x \)

is a numeric constant, variable, or expression that specifies a random variable.

\( l \)

is a numeric constant, variable, or expression that specifies the left location parameter.

Default 0

\( r \)

is a numeric constant, variable, or expression that specifies the right location parameter.

Default 1

Range \( r > l \)

**Details**

The CDF function for the uniform distribution returns the probability that an observation from a uniform distribution, with the left location parameter \( l \) and the right location parameter \( r \), is less than or equal to \( x \). The equation follows:

\[
CDF('UNIFORM', x, l, r) = \begin{cases} 
0 & x < l \\
\frac{x-l}{r-l} & l \leq x < r \\
1 & x \geq r 
\end{cases}
\]

Note: The default values for \( l \) and \( r \) are 0 and 1, respectively.

**Wald (Inverse Gaussian) Distribution**

CDF('WALD', \( x, \lambda <, \mu > \))

CDF('IGAUSS', \( x, \lambda <, \mu > \))

**Arguments**

\( x \)

is a numeric constant, variable, or expression that specifies a random variable.

\( \lambda \)

is a numeric constant, variable, or expression that specifies a shape parameter.

Range \( \lambda > 0 \)

\( \mu \)

is a numeric constant, variable, or expression that specifies the mean parameter.
Details
The CDF function for the Wald distribution returns the probability that an observation from a Wald distribution, with the shape parameter $\lambda$, is less than or equal to $x$. The equation follows:

$$F_x(x) = \Phi \left( \frac{\lambda}{\mu} \left( \frac{x}{\mu} - 1 \right) \right) + e^{2^{\lambda/\mu}} \Phi \left( -\frac{\lambda}{\mu} \left( \frac{x}{\mu} + 1 \right) \right)$$

In the equation, $\Phi(.)$ is the standard normal cumulative distribution function. When $x \leq 0$, CDF is 0.

**Weibull Distribution**

CDF(’WEIBULL’, $x$, $a$, $\lambda$)

Arguments

$x$

is a numeric constant, variable, or expression that specifies a random variable.

$a$

is a numeric constant, variable, or expression that specifies a shape parameter.

Range $a > 0$

$\lambda$

is a numeric constant, variable, or expression that specifies a scale parameter.

Default 1

Range $\lambda > 0$

Details
The CDF function for the Weibull distribution returns the probability that an observation from a Weibull distribution, with the shape parameter $a$ and the scale parameter $\lambda$, is less than or equal to $x$. The equation follows:

$$CDF(’WEIBULL’, x, a, \lambda) = \begin{cases} 0 & x < 0 \\ 1 - e^{-\left(\frac{x}{\lambda}\right)^a} & x \geq 0 \end{cases}$$

Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>y=cdf(’BERN’, 0, .25);</td>
<td>0.75</td>
</tr>
<tr>
<td>y=cdf(’BETA’, 0.2, 3, 4);</td>
<td>0.09888</td>
</tr>
<tr>
<td>SAS Statement</td>
<td>Result</td>
</tr>
<tr>
<td>---------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>y=cdf('BINOM', 4, .5, 10);</td>
<td>0.37695</td>
</tr>
<tr>
<td>y=cdf('CAUCHY', 2);</td>
<td>0.85242</td>
</tr>
<tr>
<td>y=cdf('CHISQ', 11.264, 11);</td>
<td>0.57858</td>
</tr>
<tr>
<td>y=cdf('CONMAXPOI', 5, 2.3, .4);</td>
<td>0.244511535</td>
</tr>
<tr>
<td>y=cdf('EXPO', 1);</td>
<td>0.63212</td>
</tr>
<tr>
<td>y=cdf('F', 3.32, 2, 3);</td>
<td>0.82639</td>
</tr>
<tr>
<td>y=cdf('GAMMA', 1, 3);</td>
<td>0.080301</td>
</tr>
<tr>
<td>y=cdf('GENPOISSON', 9, 1, .7);</td>
<td>0.906162963</td>
</tr>
<tr>
<td>y=cdf('HYPER', 2, 200, 50, 10);</td>
<td>0.52367</td>
</tr>
<tr>
<td>y=cdf('LAPLACE', 1);</td>
<td>0.81606</td>
</tr>
<tr>
<td>y=cdf('LOGISTIC', 1);</td>
<td>0.73106</td>
</tr>
<tr>
<td>y=cdf('LOGNORMAL', 1);</td>
<td>0.5</td>
</tr>
<tr>
<td>y=cdf('NBIB', 1, .5, 2);</td>
<td>0.5</td>
</tr>
<tr>
<td>y=cdf('NORMAL', 1.96);</td>
<td>0.97500</td>
</tr>
<tr>
<td>y=cdf('NORMALMIX', 2.3, .33, .33, .34, .5, 1.5, 2.5, .79, 1.6, 4.3);</td>
<td>0.7181</td>
</tr>
<tr>
<td>y=cdf('PARETO', 1, 1);</td>
<td>0</td>
</tr>
<tr>
<td>y=cdf('POISSON', 2, 1);</td>
<td>0.91970</td>
</tr>
<tr>
<td>y=cdf('T', .9, 5);</td>
<td>0.79531</td>
</tr>
<tr>
<td>y=cdf('TWEEDIE', .8, 5);</td>
<td>0.5917629164</td>
</tr>
<tr>
<td>y=cdf('UNIFORM', 0.25);</td>
<td>0.25</td>
</tr>
<tr>
<td>y=cdf('WALD', 1, 2);</td>
<td>0.62770</td>
</tr>
<tr>
<td>y=cdf('WEIBULL', 1, 2);</td>
<td>0.63212</td>
</tr>
</tbody>
</table>

See Also

Functions:
CEIL Function

Returns the smallest integer that is greater than or equal to the argument, fuzzed to avoid unexpected floating-point results.

Category: Truncation

Syntax

CEIL(argument)

Required Argument

argument

specifies a numeric constant, variable, or expression.

Details

If the argument is within 1E-12 of an integer, the function returns that integer.

Comparisons

Unlike the CEILZ function, the CEIL function fuzzes the result. If the argument is within 1E-12 of an integer, the CEIL function fuzzes the result to be equal to that integer. The CEILZ function does not fuzz the result. Therefore, with the CEILZ function you might get unexpected results.

Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>var1=2.1;</td>
<td></td>
</tr>
<tr>
<td>a=ceil(var1);</td>
<td>3</td>
</tr>
<tr>
<td>put a;</td>
<td></td>
</tr>
<tr>
<td>b=ceil(-2.4);</td>
<td>-2</td>
</tr>
<tr>
<td>put b;</td>
<td></td>
</tr>
<tr>
<td>c=ceil(1+1.e-11);</td>
<td>2</td>
</tr>
<tr>
<td>put c;</td>
<td></td>
</tr>
</tbody>
</table>
### SAS Statement

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>d=ceil(-1+1.e-11); put d;</td>
<td>0</td>
</tr>
<tr>
<td>e=ceil(1+1.e-13); put e;</td>
<td>1</td>
</tr>
<tr>
<td>f=ceil(223.456); put f;</td>
<td>224</td>
</tr>
<tr>
<td>g=ceil(763); put g;</td>
<td>763</td>
</tr>
<tr>
<td>h=ceil(-223.456); put h;</td>
<td>-223</td>
</tr>
</tbody>
</table>

### See Also

**Functions:**
- “CEILZ Function” on page 172

### CEILZ Function

Returns the smallest integer that is greater than or equal to the argument, using zero fuzzing.

**Category:** Truncation

**Syntax**

\[
\text{CEILZ}(\text{argument})
\]

**Required Argument**

- \text{argument}
  - is a numeric constant, variable, or expression.

**Details**

Unlike the CEIL function, the CEILZ function uses zero fuzzing. If the argument is within 1E-12 of an integer, the CEIL function fuzzes the result to be equal to that integer. The CEILZ function does not fuzz the result. Therefore, with the CEILZ function you might get unexpected results.

**Example**

The following SAS statements produce these results:
SAS Statement | Result
--- | ---
a=ceilz(2.1);
put a; | 3

b=ceilz(-2.4);
put b; | -2

c=ceilz(1+1.e-11);
put c; | 2
d=ceilz(-1+1.e-11);
put d; | 0
e=ceilz(1+1.e-13);
put e; | 2

f=ceilz(223.456);
put f; | 224
g=ceilz(763);
put g; | 763

h=ceilz(-223.456);
put h; | -223

See Also

Functions:
- “CEIL Function” on page 171
- “FLOOR Function” on page 301
- “FLOORZ Function” on page 302
- “INT Function” on page 340
- “INTZ Function” on page 385
- “ROUND Function” on page 573
- “ROUNDE Function” on page 580
- “ROUNDZ Function” on page 582

CHOOSEC Function

Returns a character value that represents the results of choosing from a list of arguments.

Category: Character

Restriction: This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility.
Syntax

CHOOSERC(index-expression, selection-1 <, …selection-n>)

Required Arguments

index-expression
  specifies a numeric constant, variable, or expression.

selection
  specifies a character constant, variable, or expression. The value of this argument is returned by the CHOOSEC function.

Details

Length of Returned Variable
In a DATA step, if the CHOOSEC function returns a value to a variable that has not previously been assigned a length, then that variable is given a length of 200 bytes.

The Basics
The CHOOSEC function uses the value of index-expression to select from the arguments that follow. For example, if index-expression is three, CHOOSEC returns the value of selection-3. If the first argument is negative, the function counts backward from the list of arguments, and returns that value.

Comparisons
The CHOOSEC function is similar to the CHOOSEN function except that CHOOSEC returns a character value while CHOOSEN returns a numeric value.

Example
The following example shows how CHOOSEC chooses from a series of values:

```sas
data _null_;  
  Fruit=choosec(1, 'apple', 'orange', 'pear', 'fig');  
  Color=choosec(3, 'red', 'blue', 'green', 'yellow');  
  Planet=choosec(2, 'Mars', 'Mercury', 'Uranus');  
  Sport=choosec(-3, 'soccer', 'baseball', 'gymnastics', 'skiing');  
  put Fruit= Color= Planet= Sport=;  
run;
```

SAS writes the following output to the log:

```
Fruit=apple Color=green Planet=Mercury Sport=baseball
```

See Also

Functions:
- “CHOOSEN Function” on page 175
CHOSEN Function

Returns a numeric value that represents the results of choosing from a list of arguments.

Category: Character

Restriction: This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility.

Syntax

```
CHOOSEN(index-expression, selection-1 <, ..., selection-n>)
```

Required Arguments

- **index-expression** specifies a numeric constant, variable, or expression.
- **selection** specifies a numeric constant, variable, or expression. The value of this argument is returned by the CHOOSEN function.

Details

The CHOOSEN function uses the value of *index-expression* to select from the arguments that follow. For example, if *index-expression* is 3, CHOOSEN returns the value of *selection-3*. If the first argument is negative, the function counts backward from the list of arguments, and returns that value.

Comparisons

The CHOOSEN function is similar to the CHOOSEC function except that CHOOSEN returns a numeric value while CHOOSEC returns a character value.

Example

The following example shows how CHOOSEN chooses from a series of values:

```sas
data _null_
  ItemNumber=choosen(5, 100, 50, 3784, 498, 679);
  Rank=choosen(-2, 1, 2, 3, 4, 5);
  Score=choosen(3, 193, 627, 33, 290, 5);
  Value=choosen(-5, -37, 82985, -991, 3, 1014, -325, 3, 54, -618);
  put ItemNumber= Rank= Score= Value=;
run;
```

SAS writes the following output to the log:

```
ItemNumber=679  Rank=4  Score=33  Value=1014
```

See Also

Functions:
CNONCT Function

Returns the noncentrality parameter from a chi-square distribution.

**Category:** Mathematical

**Syntax**

CNONCT(x, df, probability)

**Required Arguments**

- **x**
  - is a numeric random variable.
  - Range: \( x \geq 0 \)

- **df**
  - is a numeric degrees of freedom parameter.
  - Range: \( df > 0 \)

- **probability**
  - is a probability.
  - Range: \( 0 < probability < 1 \)

**Details**

The CNONCT function returns the nonnegative noncentrality parameter from a noncentral chi-square distribution whose parameters are \( x, \) df, and nc. If probability is greater than the probability from the central chi-square distribution with the parameters \( x \) and df, a root to this problem does not exist. In this case a missing value is returned. A Newton-type algorithm is used to find a nonnegative root nc of the equation

\[
P_x(x | df, nc) - prob = 0
\]

The following relationship applies to the preceding equation:

\[
P_x(x | df, nc) = e^{-\frac{nc}{2}} \sum_{j=0}^{\infty} \left(\frac{nc}{2}\right)^j \frac{1}{j!} P_g\left(\frac{x}{2} \left| \frac{df}{2} + j\right.\right)
\]

The following relationship applies to the preceding equation:

\[
P_g(x | a)
\]

is the probability from the gamma distribution given by
If the algorithm fails to converge to a fixed point, a missing value is returned.

Example

data work;
  x=2;
  df=4;
  do nc=1 to 3 by .5;
    probability=probchi(x, df, nc);
    ncc=cnonct(x, df, probability);
    output;
  end;
run;
proc print;
run;

Figure 2.10  Computations of the Noncentrality Parameters from the Chi-squared Distribution

<table>
<thead>
<tr>
<th>Obs</th>
<th>x</th>
<th>df</th>
<th>nc</th>
<th>prob</th>
<th>ncc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
<td>1.0</td>
<td>0.18611</td>
<td>1.0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>1.5</td>
<td>0.15592</td>
<td>1.5</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>4</td>
<td>2.0</td>
<td>0.13048</td>
<td>2.0</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>4</td>
<td>2.5</td>
<td>0.10907</td>
<td>2.5</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>4</td>
<td>3.0</td>
<td>0.09109</td>
<td>3.0</td>
</tr>
</tbody>
</table>

COALESCE Function

Returns the first nonmissing value from a list of numeric arguments.

**Category:** Mathematical

**Syntax**

```
COALESCE(argument-1<..., argument-n>)
```
**Required Argument**

*argument*

specifies a numeric constant, variable, or expression.

**Details**

**The Basics**

COALESCE accepts one or more numeric arguments. The COALESCE function checks the value of each argument in the order in which they are listed and returns the first nonmissing value. If only one value is listed, then the COALESCE function returns the value of that argument. If all the values of all arguments are missing, then the COALESCE function returns a missing value.

**Comparisons**

The COALESCE function searches numeric arguments, whereas the COALESCEC function searches character arguments.

**Example**

The following statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = COALESCE(42, .);</td>
<td>42</td>
</tr>
<tr>
<td>y = COALESCE(.A, .B, .C);</td>
<td>.</td>
</tr>
<tr>
<td>z = COALESCE(., 7, ., ., 42);</td>
<td>7</td>
</tr>
</tbody>
</table>

**See Also**

**Functions:**

- “COALESCEC Function” on page 178

---

### COALESCEC Function

Returns the first nonmissing value from a list of character arguments.

**Category:** Character

**Restriction:** This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see [Internationalization Compatibility](#).

#### Syntax

COALESCEC(*argument-1 < ..., argument-n>*

---

178 Chapter 2 • Dictionary of Functions and Call Routines for SAS and the CAS Server
**Required Argument**

*argument*

specifies a character constant, variable, or expression.

**Details**

**Length of Returned Variable**

In a DATA step, if the COALESCEC function returns a value to a variable that has not previously been assigned a length, then that variable is given a length of 200 bytes.

**The Basics**

COALESCEC accepts one or more character arguments. The COALESCEC function checks the value of each argument in the order in which they are listed and returns the first nonmissing value. If only one value is listed, then the COALESCEC function returns the value of that argument. A character value is considered missing if it has a length of zero or if all the characters are blank. If all the values of all arguments are missing, then the COALESCEC function returns a string with a length of zero.

**Comparisons**

The COALESCEC function searches character arguments, whereas the COALESCE function searches numeric arguments.

**Example**

The following statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>COALESCEC('', 'Hello')</td>
<td>Hello</td>
</tr>
<tr>
<td>COALESCEC ('', 'Goodbye', 'Hello')</td>
<td>Goodbye</td>
</tr>
</tbody>
</table>

**See Also**

**Functions:**

- “COALESCE Function” on page 177

---

**COLLATE Function**

Returns a character string in the ASCII collating sequence.

**Category:** Character

**Restriction:** This function is assigned an I18N Level 0 status, and is designed for SBCS data. Do not use this function to process DBCS or MBCS data. For more information, see *Internationalization Compatibility*.
Syntax
COLLATE\((\text{start-position} <, \text{end-position}> ) \mid (\text{start-position} <, , \text{length}> )\)

Required Argument
\textit{start-position}

specifies the numeric position in the collating sequence of the first character to be returned.

\textbf{Interaction} If you specify only \textit{start-position}, COLLATE returns consecutive characters from that position to the end of the collating sequence or up to 255 characters, whichever comes first.

Optional Arguments
\textit{end-position}

specifies the numeric position in the collating sequence of the last character to be returned.

For ASCII collating sequences, the characters that correspond to \textit{end-position} values between 0 and 127 represent the standard character set. Other ASCII characters that correspond to \textit{end-position} values between 128 and 255 are available on certain ASCII operating environments, but the information that those characters represent varies with the operating environment.

\textbf{Tips} \textit{end-position} must be larger than \textit{start-position}.

If you omit \textit{end-position} and use \textit{length}, mark the \textit{end-position} place with a comma.

\textit{length}

specifies the number of characters in the collating sequence.

If you specify both \textit{end-position} and \textit{length}, COLLATE ignores \textit{length}.

\textbf{Default} 200

\textbf{Tip} If you omit \textit{end-position}, use \textit{length} to specify the length of the result explicitly.

Details

A \textit{collating sequence} is the order in which a set of characters are sorted. For example, when the \texttt{SORT} procedure is executed, the collating sequence determines the sort order (higher, lower, or equal to) of a particular character in relation to other characters in the set. The collating sequence is based on the session encoding.

The COLLATE function returns a string of ASCII characters. The ASCII collating sequence contains 256 positions, referenced with the position numbers 0 through 255. Characters above position 127 correspond to characters that are used in European languages that are defined in the ISO 8859 character set.

The string that the COLLATE function returns begins with the ASCII character that is specified by the \textit{start-position} argument. If you specify the \textit{end-position} argument, the COLLATE function returns a string that contains all the ASCII characters between the \textit{start-position} and \textit{end-position} arguments, inclusively. If you specify the \textit{length}
argument instead of the end-position argument, the COLLATE function returns a string
that contains the specified number of characters, beginning at the start-position.

In a DATA step, if the COLLATE function returns a value to a variable that has not
previously been assigned a length, that variable is given a length of 200 bytes. If you
request a string longer than the remainder of the sequence, COLLATE returns a string
through the end of the sequence and pads the end of the string with spaces. Because the
full ASCII collating sequence is longer than 200 characters, the default return string is
truncated to 200 characters. To return a string with a length of 201 to 256 ASCII
characters, use one of the following methods in the DATA step:

• Use a format such as $256. for the return string variable.
• Define the length of the return string variable by using a LENGTH statement.

Examples

Example 1

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII</td>
<td>-------1-------2--</td>
</tr>
<tr>
<td>x=collate(48, , 10);</td>
<td></td>
</tr>
<tr>
<td>y=collate(48, 57);</td>
<td>0123456789 0123456789</td>
</tr>
<tr>
<td>put @1 x @14 y;</td>
<td></td>
</tr>
</tbody>
</table>

Example 2: Returning an ASCII String By Using the Return Variable Default String Length

In this example, the return code variable x uses the default return string length of 200.
Therefore, the COLLATE function returns 200 characters of the collating sequence to x.
The length is set for y to 256 characters. You can see in the output that the length of y
contains more characters.

data order;
  length y $256;
  x = collate(0,255);
  put x;
  y = collate(0,255);
  put y;
run;

proc contents data=order;
run;
Output 2.1  Output to the Log from the COLLATE Function

```sas
data order;
  length y 6256;
  x = collate(0, .255);
  put x;
  y = collate(0, .255);
  put y;
run;
```

Output 2.2  Partial Output from the CONTENTS Procedure

![Alphabetic List of Variables and Attributes](image)

**Example 3: Returning an ASCII String of a Specific Length**

In this example, the return code variable `y` uses a return string length of 56, and the COLLATE function returns the first 56 characters of the ASCII collating sequence. The remainder of the 200-character string is padded with spaces.

```sas
data _null_;  
y = collate(,,56);  
put y;  
run;
```

Output 2.3  Output of a Specific Length from the COLLATE Function

```sas
NOTE: The data set WORK.ORDER has 1 observations and 2 variables.
NOTE: DATA statement used (Total process time):
  real time 0.00 seconds
  cpu time 0.00 seconds
```

See Also

Functions:
- “BYTE Function” on page 123
- “RANK Function” on page 568
COMB Function
Computes the number of combinations of $n$ elements taken $r$ at a time.

**Category:** Combinatorial

**Syntax**
COMB($n$, $r$)

**Required Arguments**

$n$
- is a nonnegative integer that represents the total number of elements from which the sample is chosen.

$r$
- is a nonnegative integer that represents the number of chosen elements.

**Restriction**
\[ r \leq n \]

**Details**
The mathematical representation of the COMB function is given by the following equation:

\[
\text{COMB} (n, r) = \binom{n}{r} = \frac{n!}{r! (n-r)!}
\]

In the preceding equation, $n \geq 0$, $r \geq 0$, and $n \geq r$.
If the expression cannot be computed, a missing value is returned. For moderately large values, it is sometimes not possible to compute the COMB function.

**Example**
The following statement produces this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=comb(5, 1);</td>
<td>5</td>
</tr>
</tbody>
</table>

**See Also**

Functions:
- “FACT Function” on page 242
- “LCOMB Function” on page 404
COMPARE Function

Returns the position of the leftmost character by which two strings differ, or returns 0 if there is no difference.

Category: Character

Restriction: This function is assigned an I18N Level 0 status, and is designed for SBCS data. Do not use this function to process DBCS or MBCS data. For more information, see Internationalization Compatibility.


Syntax

COMPARE(string-1, string-2 <, modifier(s)>)

Required Arguments

string-1
specifies a character constant, variable, or expression.

string-2
specifies a character constant, variable, or expression.

Optional Argument

modifier
specifies a character string that can modify the action of the COMPARE function. You can use one or more of the following characters as a valid modifier:

i or I ignores the case in string-1 and string-2.

l or L removes leading blanks in string-1 and string-2 before comparing the values.

n or N removes quotation marks from any argument that is a name literal and ignores the case of string-1 and string-2. A name literal is a name token that is expressed as a string within quotation marks, followed by the uppercase or lowercase letter n. Name literals enable you to use special characters (including blanks) that are not otherwise allowed in SAS data set or variable names. For COMPARE to recognize a string as a name literal, the first character must be a quotation mark.

: (colon) truncates the longer of string-1 or string-2 to the length of the shorter string, or to one, whichever is greater. If you do not specify this modifier, the shorter string is padded with blanks to the same length as the longer string.

TIP COMPARE ignores blanks that are used as modifiers.
Details

**The Basics**
The order in which the modifiers appear in the COMPARE function is relevant.

- "LN" first removes leading blanks from each string, and then removes quotation marks from name literals.
- "NL" first removes quotation marks from name literals, and then removes leading blanks from each string.

In the COMPARE function, if \( \text{string-1} \) and \( \text{string-2} \) do not differ, COMPARE returns a value of zero. If the arguments differ, then the following apply:

- The sign of the result is negative if \( \text{string-1} \) precedes \( \text{string-2} \) in a sort sequence, and positive if \( \text{string-1} \) follows \( \text{string-2} \) in a sort sequence.
- The magnitude of the result is equal to the position of the leftmost character at which the strings differ.

Some language elements are supported in SAS but not in CAS. An example is the DATALINES statement. Some examples in this document use DATALINES in SAS to create a data set using the CAS engine. For more information, see “Functions in SAS and CAS” on page 54.

**DBCS Compatibility**
The DBCS equivalent function is “KCOMPARE Function” in *SAS Viya National Language Support (NLS): Reference Guide*. There are minor differences between the COMPARE and KCOMPARE functions. Both functions accept varying numbers of arguments. Usage of the third argument is not compatible. The following example shows the differences in the syntax:

```plaintext
COMPARE(string-1, string-2, modifier(s))
KCOMPARE(string-1, position, count, string-2)
```

**Examples**

**Example 1: Understanding the Order of Comparisons When Comparing Two Strings**
The following example compares two strings by using the COMPARE function.

```plaintext
data test;
  infile datalines missover;
  input string1 $char8. string2 $char8. modifiers $char8.;
  result=compare(string1, string2, modifiers);
datalines;
1234567812345678
123     abc
abc     abx
xyz     abcdef
aBc     abc
aBc     AbC     i
     abc
     abc     l
abc     abx
abc     abx     l
```

**COMPARE Function** 185
Example 2: Truncating Strings Using the COMPARE Function

The following example uses the : (colon) modifier to truncate strings.

data test2;
  pad1=compare('abc', 'abc            ');
  pad2=compare('abc', 'abcdef         ');
  truncate1=compare('abc', 'abcdef',':');
  truncate2=compare('abcdef', 'abc',':');
run;
COMPBL Function

Removes multiple blanks from a character string.

**Category:** Character

**Restriction:** This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility.

**Syntax**

`COMPBL(source)`

**Required Argument**

`source`

specifies a character constant, variable, or expression to compress.

**Details**

**Length of Returned Variable**

In a DATA step, if the COMPBL function returns a value to a variable that has not previously been assigned a length, then the length of that variable defaults to the length of the first argument.

### See Also

- “COMPGED Function” on page 856
- “COMPLEV Function” on page 862

### CALL Routines:

- “CALL COMPCOST Routine” on page 742

### COMPBL Function

```
   blank=compare('', 'abc', ':');
run;

   proc print data=test2 noobs;
run;
```

Output 2.5  Results of Using the Truncation Modifier

<table>
<thead>
<tr>
<th>pad1</th>
<th>pad2</th>
<th>truncate1</th>
<th>truncate2</th>
<th>blank</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>.4</td>
<td>0</td>
<td>0</td>
<td>.7</td>
</tr>
</tbody>
</table>
The Basics
The COMPBL function removes multiple blanks in a character string by translating each occurrence of two or more consecutive blanks into a single blank.

Comparisons
The COMPRESS function removes every occurrence of the specific character from a string. If you specify a blank as the character to remove from the source string, the COMPRESS function removes all blanks from the source string. The COMPBL function compresses multiple blanks to a single blank and has no effect on a single blank.

Example
The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>string='Hey Diddle Diddle';</td>
<td>Hey Diddle Diddle</td>
</tr>
<tr>
<td>string=compbl(string);</td>
<td></td>
</tr>
<tr>
<td>put string;</td>
<td></td>
</tr>
<tr>
<td>string='125 E Main St';</td>
<td>125 E Main</td>
</tr>
<tr>
<td>length address $10;</td>
<td></td>
</tr>
<tr>
<td>address=compbl(string);</td>
<td></td>
</tr>
<tr>
<td>put address;</td>
<td></td>
</tr>
</tbody>
</table>

See Also
Functions:
- “COMPRESS Function” on page 191

COMPFUZZ Function
Performs a fuzzy comparison of two numeric values.

Category: Mathematical

Syntax
COMPFUZZ(value–1, value–2 <, fuzz <, scale>>)

Required Arguments
value–1 specifies the first of two numeric values to be compared.

value–2 specifies the second numeric value to be compared.
Optional Arguments

\textit{fuzz}\n
is a nonnegative numeric value that specifies the relative threshold for comparisons. Values greater than or equal to one are treated as multiples of the machine precision.

Default 1024

\textit{scale}\n
specifies the scale factor.

Default \(\text{MAX (ABS (value–1), ABS (value–2))}\)

Details

The COMPFUZZ function returns the following values if you specify all four arguments:

- \(-1\) if \(\text{value–1} < \text{value–2} – \text{threshold}\)
- \(0\) if \(\text{ABS(\text{value–1} - \text{value–2})} \leq \text{threshold}\)
- \(1\) if \(\text{value–1} > \text{value–2} + \text{threshold}\)

The following relationships exist:

- \(\text{threshold} = \text{fuzz} * \text{ABS(scale)}\) if \(0 \leq \text{fuzz} < 1\)
- \(\text{threshold} = \text{fuzz} * \text{ABS(scale)} * \text{CONSTANT('MACEPS')}\) if \(1 \leq \text{fuzz} < 1 / \text{CONSTANT('MACEPS')}\)

COMPFUZZ avoids floating point underflow or overflow.

Comparisons

The COMPFUZZ function compares two floating point numbers and returns a value based on the comparison. The ROUND function rounds an argument to a value that is very close to a multiple of a second argument. The result might not be an exact multiple of the second argument.

Example

In floating-point arithmetic, the value of a sum sometimes depends on the order in which the numbers are added. One approximate bound for the floating point error in the computation of a sum of \(n\) numbers, \(x_1\) through \(x_n\) is expressed by the following formula:

\[ n \times \text{machine precision} \times \text{sum (abs}(x_1) + \ldots + \text{abs}(x_n)) \]

To compare sums of \(n\) floating point numbers with the COMPFUZZ function, you can therefore use \(n\) as the fuzz value and the sum of the absolute values as the scale factor, as shown in the following DATA step:

```plaintext
data _null_
  x1 = -1/3
  x2 = 22/7
  x3 = -1234567891
  x4 = 1234567890
  /* Add the numbers in two different orders. */
  sum1 = x1 + x2 + x3 + x4;
  sum2 = x4 + x3 + x2 + x1;
```
diff = abs(sum1 - sum2);
put sum1= / sum2= / diff=;
/* Using only a fuzz value gives the wrong result. The fuzz value */
/* is 8 because there are four numbers in each sum, for a total of */
/* eight numbers. */
compfuzz = compfuzz(sum1, sum2, 8);
put "fuzz only (wrong): " compfuzz=;
/* Using a fuzz factor and a scale value gives the correct result. */
scale = abs(x1) + abs(x2) + abs(x3) + abs(x4);
compfuzz = compfuzz(sum1, sum2, 8, scale);
put "fuzz and scale (correct): " compfuzz=;
run;

SAS writes the following output to the log:

Log 2.2  Partial SAS Output for the COMPFUZZ Function

<table>
<thead>
<tr>
<th>sum1=1.8095238209</th>
<th>sum2=1.8095238095</th>
</tr>
</thead>
<tbody>
<tr>
<td>diff=1.1353266E-8</td>
<td>fuzz only (wrong): compfuzz=1</td>
</tr>
<tr>
<td>fuzz and scale (correct): compfuzz=0</td>
<td></td>
</tr>
</tbody>
</table>

See Also

Functions:

- “FUZZ Function” on page 305
- “ROUND Function” on page 573

COMPOUND Function

Returns compound interest parameters.

Category: Financial

Syntax

COMPOUND(a, f, r, n)

Required Arguments

a
  is numeric, and specifies the initial amount.

Range  \( a \geq 0 \)

f
  is numeric, and specifies the future amount (at the end of \( n \) periods).

Range  \( f \geq 0 \)

r
  is numeric, and specifies the periodic interest rate expressed as a fraction.
Range $r \geq 0$

\( n \) is an integer, and specifies the number of compounding periods.

Range \( n \geq 0 \)

Details
The COMPOUND function returns the missing argument in the list of four arguments from a compound interest calculation. The arguments are related by the following equation:

\[
f = a(1 + r)^n
\]

One missing argument must be provided. A compound interest parameter is then calculated from the remaining three values. No adjustment is made to convert the results to round numbers.

If \( n = 0 \), then

\[
f = a
\]

and

\[(1 + r)^n\]

are equal to 1.

Note: If you choose \( r \) as your missing value, then COMPOUND returns an error.

Example
The accumulated value of an investment of $2000 at a nominal annual interest rate of 9%, compounded monthly after 30 months, can be expressed as the following:

\[
f = \text{compound}(2000, \., 0.09/12, 30);
\]

The value returned is 2502.54. The second argument has been set to missing, indicating that the future amount is to be calculated. The 9% nominal annual rate has been converted to a monthly rate of 0.09/12. The rate argument is the fractional (not the percentage) interest rate per compounding period.

COMPRESS Function
Returns a character string with specified characters removed from the original string.

Category: Character

Restriction: This function is assigned an I18N Level 0 status, and is designed for SBCS data. Do not use this function to process DBCS or MBCS data. For more information, see Internationalization Compatibility.

Tip: DBCS equivalent function is KCOMPRESS.
Syntax

COMPRESS(source <, characters> <, modifier(s)>)

Required Argument

source
specifies a character constant, variable, or expression from which specified characters are removed.

Optional Arguments

characters
specifies a character constant, variable, or expression that initializes a list of characters.

By default, the characters in this list are removed from the source argument. If you specify the K modifier in the third argument, then only the characters in this list are kept in the result.

TIP You can add more characters to this list by using other modifiers in the third argument.

TIP Enclose a literal string of characters in quotation marks.

modifier
specifies a character constant, variable, or expression in which each non-blank character modifies the action of the COMPRESS function. Blanks are ignored. The following characters can be used as modifiers:

a or A adds alphabetic characters to the list of characters.
c or C adds control characters to the list of characters.
d or D adds digits to the list of characters.
f or F adds the underscore character and English letters to the list of characters.
g or G adds graphic characters to the list of characters.
h or H adds a horizontal tab to the list of characters.
i or I ignores the case of the characters to be kept or removed.
k or K keeps the characters in the list instead of removing them.
l or L adds lowercase letters to the list of characters.
n or N adds digits, the underscore character, and English letters to the list of characters.
o or O processes the second and third arguments once rather than every time the COMPRESS function is called. Using the O modifier in the DATA step (excluding WHERE clauses), or in the SQL procedure, can make COMPRESS run much faster when you call it in a loop where the second and third arguments do not change.
p or P adds punctuation marks to the list of characters.
s or S adds space characters (blank, horizontal tab, vertical tab, carriage return, line feed, form feed, and NBSP (‘A0’x, or 160 decimal ASCII) to the list of characters.
t or T trims trailing blanks from the first and second arguments.
u or U adds uppercase letters to the list of characters.

w or W adds printable characters to the list of characters.

x or X adds hexadecimal characters to the list of characters.

**TIP** If the modifier is a constant, enclose it in quotation marks. Specify multiple constants in a single set of quotation marks. Modifier can also be expressed as a variable or an expression.

### Details

#### Length of Returned Variable

In a DATA step, if the COMPRESS function returns a value to a variable that has not previously been assigned a length, then that variable is given the length of the first argument.

#### The Basics

The COMPRESS function allows null arguments. A null argument is treated as a string that has a length of zero.

Based on the number of arguments, the COMPRESS functions works as follows:

<table>
<thead>
<tr>
<th>Number of Arguments</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>only the first argument, <em>source</em></td>
<td>All blanks have been removed from the argument. If the argument is completely blank, then the result is a string with a length of zero. If you assign the result to a character variable with a fixed length, then the value of that variable will be padded with blanks to fill its defined length.</td>
</tr>
<tr>
<td>the first two arguments, <em>source</em> and <em>chars</em></td>
<td>All characters that appear in the second argument are removed from the result.</td>
</tr>
<tr>
<td>three arguments, <em>source</em>, <em>chars</em>, and <em>modifier(s)</em></td>
<td>The K modifier (specified in the third argument) determines whether the characters in the second argument are kept or removed from the result.</td>
</tr>
</tbody>
</table>

The COMPRESS function compiles a list of characters to keep or remove, comprising the characters in the second argument plus any types of characters that are specified by the modifiers. For example, the D modifier specifies digits. Both of the following function calls remove digits from the result:

```plaintext
compress(source, "1234567890");
compress(source, , "d");
```

To remove digits and plus or minus signs, you can use either of the following function calls:

```plaintext
compress(source, "1234567890+-");
compress(source, "+-", "d");
```
Examples

Example 1: Compressing Blanks

The following SAS statements produce this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>a='AB C D ';</td>
<td></td>
</tr>
<tr>
<td>b=compress(a);</td>
<td></td>
</tr>
<tr>
<td>put b;</td>
<td>ABCD</td>
</tr>
</tbody>
</table>

Example 2: Compressing Lowercase Letters

The following SAS statements produce this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x='123-4567-8901 B 234-5678-9012 c';</td>
<td></td>
</tr>
<tr>
<td>y=compress(x, 'ABCD', 'l');</td>
<td></td>
</tr>
<tr>
<td>put y;</td>
<td>123-4567-8901 234-5678-9012</td>
</tr>
</tbody>
</table>

Example 3: Compressing Space Characters

The following SAS statements produce this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x='1 2 3 4 5';</td>
<td></td>
</tr>
<tr>
<td>y=compress(x, 's');</td>
<td></td>
</tr>
<tr>
<td>put y;</td>
<td>12345</td>
</tr>
</tbody>
</table>

Example 4: Keeping Characters in the List

The following SAS statements produce this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x='Math A English B Physics A';</td>
<td></td>
</tr>
<tr>
<td>y=compress(x, 'ABCD', 'k');</td>
<td></td>
</tr>
<tr>
<td>put y;</td>
<td>ABA</td>
</tr>
</tbody>
</table>

Example 5: Compressing a String and Returning a Length of 0
The following SAS statements produce this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=' ';</td>
<td></td>
</tr>
<tr>
<td>l=lengthn(compress(x));</td>
<td>0</td>
</tr>
<tr>
<td>put l;</td>
<td></td>
</tr>
</tbody>
</table>

See Also

Functions:
- “COMPBL Function” on page 187
- “LEFT Function” on page 405
- “TRIM Function” on page 640

CONSTANT Function

Computes machine and mathematical constants.

| Category: | Mathematical |

Syntax

CONSTANT(constant <, parameter>)

Required Argument

constant

is a character constant, variable, or expression that identifies the constant to be returned. Valid constants are as follows:

<table>
<thead>
<tr>
<th>Description</th>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>The natural base</td>
<td>'E'</td>
</tr>
<tr>
<td>Euler constant</td>
<td>'EULER'</td>
</tr>
<tr>
<td>Pi</td>
<td>'PI'</td>
</tr>
<tr>
<td>Exact integer</td>
<td>'EXACTINT' &lt;, nbytes&gt;</td>
</tr>
<tr>
<td>The largest double-precision number</td>
<td>'BIG'</td>
</tr>
<tr>
<td>The log with respect to base of BIG</td>
<td>'LOGBIG' &lt;, base&gt;</td>
</tr>
<tr>
<td>The square root of BIG</td>
<td>'SQRTBIG'</td>
</tr>
<tr>
<td>Description</td>
<td>Constant</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>The smallest double-precision number</td>
<td>'SMALL'</td>
</tr>
<tr>
<td>The log with respect to <code>base</code> of SMALL</td>
<td>'LOGSMALL',&lt;base&gt;</td>
</tr>
<tr>
<td>The square root of SMALL</td>
<td>'SQRTSMALL'</td>
</tr>
<tr>
<td>Machine precision constant</td>
<td>'MACEPS'</td>
</tr>
<tr>
<td>The log with respect to <code>base</code> of MACEPS</td>
<td>'LOGMACEPS',&lt;base&gt;</td>
</tr>
<tr>
<td>The square root of MACEPS</td>
<td>'SQRTMACEPS'</td>
</tr>
</tbody>
</table>

**Optional Argument**

*parameter* is an optional numeric parameter. Some of the constants specified in `constant` have an optional argument that alters the functionality of the `CONSTANT` function.

**Details**

**Overview**

*CAUTION:* In some operating environments, the run-time library might have limitations that prevent the use of the full range of floating-point numbers that the hardware provides. In such cases, the `CONSTANT` function attempts to return values that are compatible with the limitations of the run-time library. For example, if the run-time library cannot compute `EXP(LOG(CONSTANT('BIG')))`, then `CONSTANT('LOGBIG')` will not return the same value as `LOG(CONSTANT('BIG'))`, but will return a value such that `EXP(CONSTANT('LOGBIG'))` can be computed.

**The Natural Base**

`CONSTANT('E')`

The natural base is described by the following equation:

\[
\lim_{x \to 0} (1 + x)^\frac{1}{x} \approx 2.718281828459045
\]

**Euler Constant**

`CONSTANT('EULER')`

Euler's constant is described by the following equation:

\[
\lim_{n \to \infty} \left\{ \sum_{j=1}^{n} \frac{1}{j} - \log(n) \right\} \approx 0.577215664901532860
\]
Pi
CONSTANT('PI')

Pi is the ratio between the circumference and the diameter of a circle. Many expressions exist for computing this constant. One such expression for the series is described by the following equation:

\[
4 \sum_{j=0}^{\infty} \frac{(-1)^j}{2j+1} \approx 3.14159265358979323846
\]

Exact Integer
CONSTANT('EXACTINT', nbytes)

Arguments
nbytes
is a numeric value that is the number of bytes.

Range \( 2 \leq \text{nbytes} \leq 8 \)
Default 8

The exact integer is the largest integer \( k \) such that all integers less than or equal to \( k \) in absolute value have an exact representation in a SAS numeric variable of length n nbytes. This information can be useful to know before you trim a SAS numeric variable from the default 8 bytes of storage to a lower number of bytes to save storage.

The Largest Double-Precision Number
CONSTANT('BIG')

This case returns the largest double-precision floating-point number (8-bytes) that is representable on your computer.

The Logarithm of BIG
CONSTANT('LOGBIG', base)

Arguments
base
is a numeric value that is the base of the logarithm.

Restriction The base that you specify must be greater than the value of 1+SQRTMACEPS.
Default the natural base, E.

This case returns the logarithm with respect to the base of the largest double-precision floating-point number (8-bytes) that is representable on your computer.

It is safe to exponentiate the given base raised to a power less than or equal to CONSTANT('LOGBIG', base) by using the power operation (**) without causing any overflows.

It is safe to exponentiate any floating-point number less than or equal to CONSTANT('LOGBIG') by using the exponential function, EXP, without causing any overflows.

The Square Root of BIG
CONSTANT('SQRBTBIG')
This case returns the square root of the largest double-precision floating-point number (8-bytes) that is representable on your computer.

It is safe to square any floating-point number less than or equal to CONSTANT('SQRTBIG') without causing any overflows.

**The Smallest Double-Precision Number**

CONSTANT('SMALL')

This case returns the smallest double-precision floating-point number (8-bytes) that is representable on your computer.

**The Logarithm of SMALL**

CONSTANT('LOGSMALL', base)

**Arguments**

*base*

is a numeric value that is the base of the logarithm.

**Restriction**

The base that you specify must be greater than the value of 1+SQRTMACEPS.

**Default**

the natural base, E.

This case returns the logarithm with respect to the base of the smallest double-precision floating-point number (8-bytes) that is representable on your computer.

It is safe to exponentiate the given base raised to a power greater than or equal to CONSTANT('LOGSMALL', base) by using the power operation (**) without causing any underflows or 0.

It is safe to exponentiate any floating-point number greater than or equal to CONSTANT('LOGSMALL') by using the exponential function, EXP, without causing any underflows or 0.

**The Square Root of SMALL**

CONSTANT('SQRTSMALL')

This case returns the square root of the smallest double-precision floating-point number (8-bytes) that is representable on the computer.

It is safe to square any floating-point number greater than or equal to CONSTANT('SQRTBIG') without causing any underflows or 0.

**Machine Precision**

CONSTANT('MACEPS')

This case returns the smallest double-precision floating-point number (8-bytes) $\varepsilon = 2^{-j}$ for some integer $j$, such that $1 + \varepsilon > 1$.

This constant is important in finite precision computations.

**The Logarithm of MACEPS**

CONSTANT('LOGMACEPS', base)

**Arguments**

*base*

is a numeric value that is the base of the logarithm.
Restriction  The base that you specify must be greater than the value of 1+SQRTMACEPS.

Default  the natural base, E.

This case returns the logarithm with respect to the base of CONSTANT('MACEPS').

**The Square Root of MACEPS**  
**CONSTANT('SQRTMACEPS')**

This case returns the square root of CONSTANT('MACEPS').

---

**CONVX Function**

Returns the convexity for an enumerated cash flow.

**Category:**  Financial

**Syntax**

\[ \text{CONVX}(y, f, c(1), ..., c(k)) \]

**Required Arguments**

\( y \)

specifies the effective per-period yield-to-maturity, expressed as a fraction.

Range  \( 0 < y < 1 \)

\( f \)

specifies the frequency of cash flows per period.

Range  \( f > 0 \)

\( c(1), ..., c(k) \)

specifies a list of cash flows.

**Details**

The CONVX function returns the value

\[
C = \sum_{k=1}^{K} \frac{k(k + f)}{P(1 + y)^{2}f^{2}} \cdot \frac{c(k)}{P(1 + y)^{2}f^{2}}
\]

The following relationship applies to the preceding equation:

\[
P = \sum_{k=1}^{K} \frac{c(k)}{(1 + y)^{2}f^{2}}
\]
Example

data _null_;  
c=convx(1/20, 1, .33, .44, .55, .49, .50, .22, .4, .8, .01, .36, .2, .4);  
    put c;  
run;  

SAS writes the following output to the log:

42.3778

CONVXP Function

Returns the convexity for a periodic cash flow stream, such as a bond.

Category: Financial

Syntax

CONVXP(A, c, n, K, k0, y)

Required Arguments

\( A \)

specifies the par value.

Range \( A > 0 \)

\( c \)

specifies the nominal per-period coupon rate, expressed as a fraction.

Range \( 0 \leq c < 1 \)

\( n \)

specifies the number of coupons per period.

Range \( n > 0 \) and is an integer

\( K \)

specifies the number of remaining coupons.

Range \( K > 0 \) and is an integer

\( k_0 \)

specifies the time from the present date to the first coupon date, expressed in terms of the number of periods.

Range \( 0 < k_0 \leq \frac{1}{n} \)

\( y \)

specifies the nominal per-period yield-to-maturity, expressed as a fraction.

Range \( y > 0 \)
Details

The CONVXP function returns the value

\[ C = \frac{1}{n^2} \sum_{k=1}^{K} \frac{t_k(t_k + 1) - c(k)}{P(1 + \frac{y}{n})^k} \]

The following relationships apply to the preceding equation:

\[ t_k = nk_0 + k - 1 \]

\[ c(k) = \frac{e}{n} A \text{ for } k = 1, \ldots, K - 1 \]

\[ c(K) = (1 + \frac{e}{n})A \]

The following relationship applies to the preceding equation:

\[ P = \sum_{k=1}^{K} \frac{c(k)}{(1 + \frac{y}{n})^k} \]

Example

In the following example, the CONVXP function returns the convexity of a bond that has a face value of 1000, an annual coupon rate of 0.01, 4 coupons per year, and 14 remaining coupons. The time from settlement date to next coupon date is 0.165, and the annual yield-to-maturity is 0.08.

```
data _null_;  
y=convxp(1000, .01, 4, 14, .33/2, .08);  
put y;  
run;  
```

SAS writes the following output to the log:

```
11.729001987
```

COS Function

Returns the cosine.

Category: Trigonometric

Syntax

\texttt{COS(argument)}
**Required Argument**

*argument*

specifies a numeric constant, variable, or expression and is expressed in radians. If the magnitude of *argument* is so great that \( \text{mod}(\text{argument}, \pi) \) is accurate to less than about three decimal places, COS returns a missing value.

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=cos(0.5);</td>
<td>0.8775825619</td>
</tr>
<tr>
<td>x=cos(0);</td>
<td>1</td>
</tr>
<tr>
<td>x=cos(3.14159/3);</td>
<td>0.500000766</td>
</tr>
</tbody>
</table>

**COSH Function**

Returns the hyperbolic cosine.

**Category:** Hyperbolic

**Syntax**

COSH(*argument*)

**Required Argument**

*argument*

specifies a numeric constant, variable, or expression.

**Details**

The COSH function returns the hyperbolic cosine of the argument, given by

\[
\frac{e^{\text{argument}} + e^{-\text{argument}}}{2}
\]

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=cosh(0);</td>
<td>1</td>
</tr>
<tr>
<td>x=cosh(-5.0);</td>
<td>74.209948525</td>
</tr>
</tbody>
</table>
COT Function

Returns the cotangent.

Category: Trigonometric

Syntax

COT(argument)

Required Argument

argument specifies a numeric constant, variable, or expression and is expressed in radians.

Restriction argument cannot be 0 or a multiple of PI.

Comparisons

The COT function is related to the TAN function in this way:

cot(x) = 1/tan(x)

Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=cot(0.5);</td>
<td>1.8304877217</td>
</tr>
<tr>
<td>x=cot(1);</td>
<td>0.6420926159</td>
</tr>
<tr>
<td>x=cot(3.14159/3);</td>
<td>0.5773514486</td>
</tr>
</tbody>
</table>

Note: If you use x=cot(0);, then the COT function returns a missing value, and a note is written to the log that indicates you entered an invalid argument to the function. This is the correct behavior.

See Also

Functions:

- “COS Function” on page 201
- “CSC Function” on page 212
COUNT Function

Counts the number of times that a specified substring appears within a character string.

**Category:** Character

**Restriction:** This function is assigned an I18N Level 0 status, and is designed for SBCS data. Do not use this function to process DBCS or MBCS data. For more information, see Internationalization Compatibility.

**Tip:** You can use the KCOUNT function for DBCS processing, but the functionality is different. See DBCS Compatibility on page 205.

**Syntax**

COUNT(string, substring <, modifier(s)>)

**Required Arguments**

**string**

specifies a character constant, variable, or expression in which substrings are to be counted.

**Tip** Enclose a literal string of characters in quotation marks.

**substring**

is a character constant, variable, or expression that specifies the substring of characters to count in string.

**Tip** Enclose a literal string of characters in quotation marks.

**Optional Argument**

**modifier**

is a character constant, variable, or expression that specifies one or more modifiers. The following modifiers can be in uppercase or lowercase:

i ignores character case during the count. If this modifier is not specified, COUNT counts only character substrings with the same case as the characters in substring.

T Tip If the modifier is a constant, enclose it in quotation marks. Specify multiple constants in a single set of quotation marks. Modifier can also be expressed as a variable or an expression.
Details

The Basics
The COUNT function searches string, from left to right, for the number of occurrences of the specified substring, and returns that number of occurrences. If the substring is not found in string, COUNT returns a value of 0.

CAUTION:
If two occurrences of the specified substring overlap in the string, the result is undefined. For example, `count('boobooboo', 'booboo')` might return either a 1 or a 2.

DBCS Compatibility
For DBCS processing, you can use the “KCOUNT Function” in SAS Viya National Language Support (NLS): Reference Guide, but the functionality is different.

If the value of substring in the COUNT function is longer than two bytes, then the COUNT function can handle DBCS strings. The following examples show the differences in syntax:

```
COUNT(string, substring <, modifier(s)>)
KCOUNT(string)
```

Comparisons
The COUNT function counts substrings of characters in a character string, whereas the COUNTC function counts individual characters in a character string.

Example
The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>xyz='This is a thistle? Yes, this is a thistle.';</code> <code>howmanythis=count(xyz, 'this')</code>; <code>put howmanythis;</code></td>
<td>3</td>
</tr>
<tr>
<td><code>xyz='This is a thistle? Yes, this is a thistle.';</code> <code>howmanyis=count(xyz, 'is')</code>; <code>put howmanyis;</code></td>
<td>6</td>
</tr>
<tr>
<td><code>howmanythis_i=count('This is a thistle? Yes, this is a thistle.' , 'this', 'i');</code> <code>put howmanythis_i;</code></td>
<td>4</td>
</tr>
<tr>
<td><code>variable1='This is a thistle? Yes, this is a thistle.';</code> <code>variable2='is ';</code> <code>variable3='i';</code> <code>howmanyis_i=count(variable1, variable2, variable3);</code> <code>put howmanyis_i;</code></td>
<td>4</td>
</tr>
</tbody>
</table>
expression1='This is a thistle? '||'Yes, this is a thistle. ';
expression2=kscan('This is',2)||'     ';
expression3=compress('i   '||'     t');
howmanyis_it=count(expression1, expression2, expression3);
put howmanyis_it;

See Also

Functions:
- “COUNTC Function” on page 206
- “COUNTW Function” on page 209
- “FIND Function” on page 285
- “INDEX Function” on page 327

COUNTC Function

Counts the number of characters in a string that appear or do not appear in a list of characters.

- **Category:** Character
- **Restriction:** This function is assigned an I18N Level 0 status, and is designed for SBCS data. Do not use this function to process DBCS or MBCS data. For more information, see Internationalization Compatibility.

**Syntax**

\[
\text{COUNTC}(\text{string, character-list } <, \text{ modifier(s)}>)
\]

**Required Arguments**

- **string**
  - Specifies a character constant, variable, or expression in which characters are counted.
  - **Tip**: Enclose a literal string of characters in quotation marks.

- **character-list**
  - Specifies a character constant, variable, or expression that initializes a list of characters. COUNTC counts characters in this list, provided that you do not specify the V modifier in the modifier argument. If you specify the V modifier, then all characters that are not in this list are counted. You can add more characters to the list by using other modifiers.
  - **Tips**: Enclose a literal string of characters in quotation marks.

If there are no characters in the list after processing the modifiers, COUNTC returns 0.
Optional Argument

**modifier**

specifies a character constant, variable, or expression in which each non-blank character modifies the action of the COUNTC function. Blanks are ignored. The following characters, in uppercase or lowercase, can be used as modifiers:

- blank is ignored.
- a or A adds alphabetic characters to the list of characters.
- b or B scans string from right to left, instead of from left to right.
- c or C adds control characters to the list of characters.
- d or D adds digits to the list of characters.
- f or F adds an underscore and English letters (that is, the characters that can begin a SAS variable name using VALIDVARNAME=V7) to the list of characters.
- g or G adds graphic characters to the list of characters.
- h or H adds a horizontal tab to the list of characters.
- i or I ignores case.
- l or L adds lowercase letters to the list of characters.
- n or N adds digits, an underscore, and English letters (that is, the characters that can appear in a SAS variable name using VALIDVARNAME=V7) to the list of characters.
- o or O processes the character-list and modifier arguments only once, at the first call to this instance of COUNTC. If you change the value of character-list or modifier in subsequent calls, the change might be ignored by COUNTC.
- p or P adds punctuation marks to the list of characters.
- s or S adds space characters to the list of characters (blank, horizontal tab, carriage return, line feed, and form feed).
- t or T trims trailing blanks from string and character-list. If you want to remove trailing blanks from only one character argument instead of both (or all) character arguments, use the TRIM function instead of the COUNTC function with the T modifier.
- u or U adds uppercase letters to the list of characters.
- v or V counts characters that do not appear in the list of characters. If you do not specify this modifier, then COUNTC counts characters that do appear in the list of characters.
- w or W adds printable characters to the list of characters.
- x or X adds hexadecimal characters to the list of characters.

**Tip** If modifier is a constant, enclose it in quotation marks. Specify multiple constants in a single set of quotation marks.
Details

The COUNTC function allows character arguments to be null. Null arguments are treated as character strings with a length of zero. If there are no characters in the list of characters to be counted, COUNTC returns zero.

Comparisons

The COUNTC function counts individual characters in a character string, whereas the COUNT function counts substrings of characters in a character string.

Example

The following example uses the COUNTC function with and without modifiers to count the number of characters in a string.

```sas
data test;
  string = 'Baboons Eat Bananas     ';
  a = countc(string, 'a');
  b = countc(string, 'b');
  b_i = countc(string, 'b', 'i');
  abc_i = countc(string, 'abc', 'i');
  /* Scan string for characters that are not "a", "b", */
  /* and "c", ignore case, (and include blanks). */
  abc_iv = countc(string, 'abc', 'iv');
  /* Scan string for characters that are not "a", "b", */
  /* and "c", ignore case, and trim trailing blanks. */
  abc_ivt = countc(string, 'abc', 'ivt');
run;

proc print data=test noobs;
run;
```

Figure 2.11  Output from Using the COUNTC Functions with and without Modifiers

<table>
<thead>
<tr>
<th>string</th>
<th>a</th>
<th>b</th>
<th>b_i</th>
<th>abc_i</th>
<th>abc_iv</th>
<th>abc_ivt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baboons Eat Bananas</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>8</td>
<td>16</td>
<td>11</td>
</tr>
</tbody>
</table>

See Also

Functions:

- “ANYALNUM Function” on page 80
- “ANYALPHA Function” on page 82
- “ANYCNTRL Function” on page 84
- “ANYDIGIT Function” on page 86
- “ANYGRAPH Function” on page 89
- “ANYLOWER Function” on page 91
COUNTW Function

Counts the number of words in a character string.

Category: Character
Restriction: This function is assigned an I18N Level 0 status, and is designed for SBCS data. Do not use this function to process DBCS or MBCS data. For more information, see Internationalization Compatibility.

Syntax

\[ \text{COUNTW}(<\text{string}> <, <\text{character(s)}> <, <\text{modifier(s)}>>( )) \]

Optional Arguments

\textit{string}

specifies a character constant, variable, or expression in which words are counted.

\textit{character}

specifies an optional character constant, variable, or expression that initializes a list of characters. The characters in this list are the delimiters that separate words, provided that you do not use the K modifier in the \textit{modifier} argument. If you specify the K modifier, then all characters that are not in this list are delimiters. You can add more characters to the list by using other modifiers.
Tip  Character arguments can be null. Null arguments are treated as character strings with a length of zero. Numeric arguments cannot be null.

**modifier**

specifies a character constant, variable, or expression in which each non-blank character modifies the action of the COUNTW function. The following characters, in uppercase or lowercase, can be used as modifiers:

- **blank** is ignored.
- **a or A** adds alphabetic characters to the list of characters.
- **b or B** counts from right to left instead of from left to right. Right-to-left counting makes a difference only when you use the **Q** modifier and the string contains unbalanced quotation marks.
- **c or C** adds control characters to the list of characters.
- **d or D** adds digits to the list of characters.
- **f or F** adds an underscore and English letters (that is, the characters that can begin a SAS variable name using VALIDVARNAME=V7) to the list of characters.
- **g or G** adds graphic characters to the list of characters.
- **h or H** adds a horizontal tab to the list of characters.
- **i or I** ignores the case of the characters.
- **k or K** causes all characters that are not in the list of characters to be treated as delimiters. If **K** is not specified, then all characters that are in the list of characters are treated as delimiters.
- **l or L** adds lowercase letters to the list of characters.
- **m or M** specifies that multiple consecutive delimiters, and delimiters at the beginning or end of the **string** argument, refer to words that have a length of zero. If the **M** modifier is not specified, then multiple consecutive delimiters are treated as one delimiter, and delimiters at the beginning or end of the **string** argument are ignored.
- **n or N** adds digits, an underscore, and English letters (that is, the characters that can appear after the first character in a SAS variable name using VALIDVARNAME=V7) to the list of characters.
- **o or O** processes the **characters** and **modifier** arguments only once, rather than every time the COUNTW function is called. Using the **O** modifier in the DATA step (excluding WHERE clauses), or in the SQL procedure, can make COUNTW run faster when you call it in a loop where **chars** and **modifier** arguments do not change.
- **p or P** adds punctuation marks to the list of characters.
- **q or Q** ignores delimiters that are inside substrings that are enclosed in quotation marks. If the value of **string** contains unmatched quotation marks, then scanning from left to right produces different words than scanning from right to left.
- **s or S** adds space characters (blank, horizontal tab, vertical tab, carriage return, line feed, and form feed) to the list of characters.
- **t or T** trims trailing blanks from the **string** and **chars** arguments.
- **u or U** adds uppercase letters to the list of characters.
w or W adds printable characters to the list of characters.
x or X adds hexadecimal characters to the list of characters.

Details

Some language elements are supported in SAS but not in CAS. An example is the DATALINES statement. Some examples in this document use DATALINES in SAS to create a data set using the CAS engine. For more information, see “Functions in SAS and CAS” on page 54.

Definition of "Word"

In the COUNTW function, "word" refers to a substring that has one of the following characteristics:

• is bounded on the left by a delimiter or the beginning of the string
• is bounded on the right by a delimiter or the end of the string
• contains no delimiters, except if you use the Q modifier and the delimiters are within substrings that have quotation marks

Note: The definition of "word" is the same in both the SCAN function and the COUNTW function.

Delimiter refers to any of several characters that you can specify to separate words.

Using the COUNTW Function in ASCII and EBCDIC Environments

If you use the COUNTW function with only two arguments, the default delimiters depend on whether your computer uses ASCII or EBCDIC characters.

• If your computer uses ASCII characters, then the default delimiters are as follows:
  blank ! $ % & ( ) * + , . / ; < ^ |
  In ASCII environments that do not contain the ^ character, the SCAN function uses the ~ character instead.
• If your computer uses EBCDIC characters, then the default delimiters are as follows:
  blank ! $ % & ( ) * + , . / ; < ¬ | ¢

Using the M Modifier

If you do not use the M modifier, then a word must contain at least one character. If you use the M modifier, then a word can have a length of zero. In this case, the number of words is one plus the number of delimiters in the string, not counting delimiters inside strings that are enclosed in quotation marks when you use the Q modifier.

Example

The following example shows how to use the COUNTW function with the M and P modifiers.

data test;
  length default blanks mp 8;
  input string $char60.;
  default=countw(string);
  blanks=countw(string, ' ');
  mp=countw(string, ', 'mp');
CSC Function

Returns the cosecant.

Category: Trigonometric

Syntax

\[
\text{CSC}(\text{argument})
\]

Required Argument

\textit{argument}

specifies a numeric constant, variable, or expression and is expressed in radians.
Restriction argument cannot be 0 or a multiple of PI.

Comparisons

The CSC function is related to the SIN function in this way:
\[ \csc(x) = \frac{1}{\sin(x)} \]

Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=csc(0.5);</td>
<td>2.0858296429</td>
</tr>
<tr>
<td>x=csc(1);</td>
<td>1.1883951058</td>
</tr>
<tr>
<td>x=csc(3.14159/3);</td>
<td>1.1547011281</td>
</tr>
</tbody>
</table>

Note: If you use \( x = \text{csc}(0) \);, then the CSC function returns a missing value, and a note is written to the log that indicates you entered an invalid argument to the function. This is the correct behavior.

See Also

Functions:
- “COS Function” on page 201
- “COT Function” on page 203
- “SEC Function” on page 599
- “SIN Function” on page 605
- “TAN Function” on page 624

CSS Function

Returns the corrected sum of squares.

Syntax

\[
\text{CSS}(\text{argument-1}, ..., \text{argument-n})
\]

Required Argument

*argument*

specifies a numeric constant, variable, or expression. At least one nonmissing argument is required. Otherwise, the function returns a missing value. If you have
more than one argument, the argument list can consist of a variable list, which is preceded by OF.

Example
The following SAS statements produce these results.

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1=css(5,9,3,6);</td>
<td>18.75</td>
</tr>
<tr>
<td>x2=css(5,8,9,6,.);</td>
<td>10</td>
</tr>
<tr>
<td>x3=css(8,9,6,.);</td>
<td>4.6666666667</td>
</tr>
<tr>
<td>x4=css(of x1-x3);</td>
<td>101.11574074</td>
</tr>
</tbody>
</table>

CUMIPMT Function
Returns the cumulative interest paid on a loan between the start and end period.

Category: Financial

Syntax

\[
\text{CUMIPMT}(\text{rate}, \text{number-of-periods}, \text{principal-amount}, <\text{start-period}, <\text{end-period}, <\text{type}>)
\]

Required Arguments

*rate*
  - specifies the interest rate per payment period.

*number-of-periods*
  - specifies the number of payment periods. *Number-of-periods* must be a positive integer value.

*principal-amount*
  - specifies the principal amount of the loan. Zero is assumed if a missing value is specified.

Optional Arguments

*start-period*
  - specifies the start period for the calculation.

*end-period*
  - specifies the end period for the calculation.

*type*
  - specifies whether the payments occur at the beginning or end of a period. 0 represents the end-of-period payments, and 1 represents the beginning-of-period payments. 0 is assumed if *type* is omitted or if a missing value is specified.
Example

- The cumulative interest that is paid during the second year of a $125,000, 30-year loan with end-of-period monthly payments and a nominal annual interest rate of 9%, is computed as follows:

\[
\text{TotalInterest} = \text{CUMIPMT}(0.09/12, 360, 125000, 13, 24, 0);
\]

This computation returns a value of 11,135.23.

- The interest that is paid on the first period of the same loan is computed in the following way:

\[
\text{first_period_interest} = \text{CUMIPMT}(0.09/12, 360, 125000, 1, 1, 0);
\]

This computation returns a value of 937.50.

CUMPRINC Function

Returns the cumulative principal paid on a loan between the start and end period.

Category: Financial

Syntax

\[
\text{CUMPRINC}(\text{rate}, \text{number-of-periods}, \text{principal-amount}, <\text{start-period}>, <\text{end-period}>, <\text{type}>)
\]

Required Arguments

- \text{rate}
  - specifies the interest rate per payment period.

- \text{number-of-periods}
  - specifies the number of payment periods. \text{Number-of-periods} must be a positive integer value.

- \text{principal-amount}
  - specifies the principal amount of the loan. Zero is assumed if a missing value is specified.

Optional Arguments

- \text{start-period}
  - specifies the start period for the calculation.

- \text{end-period}
  - specifies the end period for the calculation.

- \text{type}
  - specifies whether the payments occur at the beginning or end of a period. 0 represents the end-of-period payments, and 1 represents the beginning-of-period payments. 0 is assumed if \text{type} is omitted or if a missing value is specified.
Example

- The cumulative principal that is paid during the second year of a $125,000, 30-year loan with end-of-period monthly payments and a nominal annual interest rate of 9%, is computed as follows:
  \[
  \text{PrincipalYear2} = \text{CUMPRINC}(0.09/12, 360, 125000, 12, 24, 0);
  \]
  This computation returns a value of 934.107.

- The principal that is paid on the second year of the same loan with beginning-of-period payments is computed as follows:
  \[
  \text{PrincipalYear2b} = \text{CUMPRINC}(0.09/12, 360, 125000, 12, 24, 1);
  \]
  This computation returns a value of 927.153.

---

**CV Function**

Returns the coefficient of variation.

**Category:** Descriptive Statistics

**Syntax**

\[
\text{CV}(\text{argument-1, argument-2 <, ...argument-n>})
\]

**Required Argument**

- **argument** specifies a numeric constant, variable, or expression. At least two arguments are required. The argument list can consist of a variable list, which is preceded by OF.

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1=CV(5, 9, 3, 6);</td>
<td>43.47826087</td>
</tr>
<tr>
<td>x2=CV(5, 8, 9, 6, .);</td>
<td>26.082026548</td>
</tr>
<tr>
<td>x3=CV(8, 9, 6, .);</td>
<td>19.924242152</td>
</tr>
<tr>
<td>x4=CV(of x1-x3);</td>
<td>40.953539216</td>
</tr>
</tbody>
</table>

---

**DAIRY Function**

Returns the derivative of the AIRY function.

**Category:** Mathematical
Syntax

DAIRY(x)

Required Argument

x
specifies a numeric constant, variable, or expression.

Details

The DAIRY function returns the value of the derivative of the AIRY function. (See a list of References on page 1111.)

Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=dairy(2.0);</td>
<td>-0.053090384</td>
</tr>
<tr>
<td>x=dairy(-2.0);</td>
<td>0.6182590207</td>
</tr>
</tbody>
</table>

DATDIF Function

Returns the number of days between two dates after computing the difference between the dates according to specified day count conventions.

Category: Date and Time

Syntax

DATDIF(start-date, end-date, basis)

Required Arguments

start-date
specifies a SAS date value that identifies the starting date.

Tip If start-date falls at the end of a month, then SAS treats the date as if it were the last day of a 30-day month.

end-date
specifies a SAS date value that identifies the ending date.

Tip If end-date falls at the end of a month, then SAS treats the date as if it were the last day of a 30-day month.
basis

specifies a character string that represents the day count basis. The following values for **basis** are valid:

'30/360'
specifies a 30-day month and a 360-day year, regardless of the actual number of calendar days in a month or year.

A security that pays interest on the last day of a month will either always make its interest payments on the last day of the month, or it will always make its payments on the numerically same day of a month, unless that day is not a valid day of the month, such as February 30. For more information, see “Method of Calculation for Day Count Basis (30/360)” on page 218.

**Alias**  '360'

'ACT/ACT'
uses the actual number of days between dates. Each month is considered to have the actual number of calendar days in that month, and each year is considered to have the actual number of calendar days in that year.

**Alias**  'Actual'

'ACT/360'
uses the actual number of calendar days in a particular month, and 360 days as the number of days in a year, regardless of the actual number of days in a year.

**Tip**  *ACT/360* is used for short-term securities.

'ACT/365'
uses the actual number of calendar days in a particular month, and 365 days as the number of days in a year, regardless of the actual number of days in a year.

**Tip**  *ACT/365* is used for short-term securities.

Details

The Basics

The DATDIF function has a specific meaning in the securities industry, and the method of calculation is not the same as the actual day count method. Calculations can use months and years that contain the actual number of days. Calculations can also be based on a 30-day month or a 360-day year. For more information about standard securities calculation methods, see the References section at the bottom of this function.

**Note:** When counting the number of days in a month, DATDIF *always* includes the starting date and excludes the ending date.

Method of Calculation for Day Count Basis (30/360)

To calculate the number of days between two dates, use the following formula:

\[
\text{Number of days} = \left[ (Y2 - Y1) \times 360 \right] + \left[ (M2 - M1) \times 30 \right] + (D2 - D1)
\]

**Arguments**

Y2
specifies the year of the later date.
Y1  
  specifies the year of the earlier date.

M2  
  specifies the month of the later date.

M1  
  specifies the month of the earlier date.

D2  
  specifies the day of the later date.

D1  
  specifies the day of the earlier date.

Because all months can contain only 30 days, you must adjust for the months that do not contain 30 days. Do this before you calculate the number of days between the two dates.

The following rules apply:

- If the security follows the End-of-Month rule, and D2 is the last day of February (28 days in a non-leap year, 29 days in a leap year), and D1 is the last day of February, then change D2 to 30.
- If the security follows the End-of-Month rule, and D1 is the last day of February, then change D1 to 30.
- If the value of D2 is 31 and the value of D1 is 30 or 31, then change D2 to 30.
- If the value of D1 is 31, then change D1 to 30.

**Example**

In the following example, DATDIF returns the actual number of days between two dates, as well as the number of days based on a 30-day month and a 360-day year.

```sas
data _null;
  sdate='16oct78'd;
  edate='16feb96'd;
  actual=datdif(sdate, edate, 'act/act');
  days360=datdif(sdate, edate, '30/360');
  put actual=;
  run;
```

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put actual=;</td>
<td>6332</td>
</tr>
<tr>
<td>put days360=;</td>
<td>6240</td>
</tr>
</tbody>
</table>

**See Also**

**Functions:**

- “YRDIF Function” on page 692
DATE Function

Returns the current date as a SAS date value.

**Category:** Date and Time

**Alias:** TODAY

**Interaction:** If the value of the TIMEZONE= system option is set to a time zone name or time zone ID, the date and time values that are returned for this function are determined by the time zone.

**See:** “TODAY Function” on page 630

### Syntax

DATE()

### Details

The DATE function produces the current date in the form of a SAS date value, which is the number of days since January 1, 1960.

### Examples

**Example 1: Determining a Date Value for an America/Los_Angeles Time Zone**

This example shows how the DATE function returns a date value based on the value of the TIMEZONE= system option.

```sas
option timezone='America/Los_Angeles';
data _null_;  
d1=date();  
put d1=nldate.;  
run;
```

SAS writes the following results to the log:

```
d1=November 07, 2012
```

**Example 2: Determining a Date Value for an Asia/Shanghai Time Zone**

This example shows how the DATE function returns a date value based on the value of the TIMEZONE= system option.

```sas
option timezone='Asia/Shanghai';
data _null_;  
```
d2=date();
   put d2=nldate.;
run;

SAS writes the following results to the log:

| d2=November 08, 2012 |

---

**DATEJUL Function**

Converts a Julian date to a SAS date value.

**Category:** Date and Time

**Syntax**

```sas
DATEJUL(julian-date)
```

**Required Argument**

`julian-date`

specifies a SAS numeric expression that represents a Julian date. A Julian date in SAS is a date in the form `yyddd` or `yyyyddd`, where `yy` or `yyyy` is a two-digit or four-digit integer that represents the year and `ddd` is the number of the day of the year. The value of `ddd` must be between 1 and 365 (or 366 for a leap year).

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xstart=datejul(94365); put Xstart / Xstart date9.;</td>
<td>12783 31DEC1994</td>
</tr>
<tr>
<td>Xend=datejul(2001001); put Xend / Xend date9.;</td>
<td>14976 01JAN2001</td>
</tr>
</tbody>
</table>

**See Also**

**Functions:**

- “JULDATE Function” on page 390

---

**DATEPART Function**

Extracts the date from a SAS datetime value.
Category: Date and Time

Syntax

```
DATEPART(datetime)
```

Required Argument

datetime
specifies a SAS expression that represents a SAS datetime value.

Example

The following SAS statements produce this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>conn='01feb94:8:45'dt; servdate=datepart(conn); put servdate worddate.;</td>
<td>February 1, 1994</td>
</tr>
</tbody>
</table>

See Also

Functions:

- “DATETIME Function” on page 222
- “TIMEPART Function” on page 626

DATETIME Function

Returns the current date and time of day as a SAS datetime value.

Category: Date and Time

Interaction: If the value of the TIMEZONE= system option is set to a time zone name or time zone ID, the date and time values that are returned for this function are determined by the time zone.

Syntax

```
DATETIME()
```

Examples

**Example 1: Returning the Number of Seconds with the DATETIME Function**

This example returns a SAS value that represents the number of seconds between January 1, 1960, and the current time.

```
when=datetime();
```
Example 2: Determining a Datetime Value for an America/Los_Angeles Time Zone
This example shows how the DATETIME function returns datetime values based on the value of the TIMEZONE= system option.

```sas
option timezone='America/Los_Angeles';
data _null_
  dt1=datetime();
  put dt1=nldatm.;
r
SAS writes the following results to the log:

```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
**Required Argument**

date

specifies a SAS expression that represents a SAS date value.

**Details**

The DAY function produces an integer from 1 to 31 that represents the day of the month.

**Example**

The following SAS statements produce this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>now='05may97'd;</td>
<td></td>
</tr>
<tr>
<td>d=day(now) ;</td>
<td>5</td>
</tr>
<tr>
<td>put d;</td>
<td></td>
</tr>
</tbody>
</table>

**See Also**

Functions:

- “MONTH Function” on page 442
- “YEAR Function” on page 690

---

**DEQUOTE Function**

Removes matching quotation marks from a character string that begins with a quotation mark, and deletes all characters to the right of the closing quotation mark.

**Category:** Character

**Restriction:** This function is assigned an I18N Level 1 status. If possible, avoid I18N Level 1 functions if you are using a non-English language. Under certain circumstances, the I18N Level 1 functions might not work correctly with Double-Byte Character Set (DBCS) or Multi-Byte Character Set (MBCS) encodings. For more information, see Internationalization Compatibility.

**Syntax**

DEQUOTE(string)

**Required Argument**

string

specifies a character constant, variable, or expression.
Details

**Length of Returned Variable**
In a DATA step, if the DEQUOTE function returns a value to a variable that has not been previously assigned a length, then that variable is given the length of the argument.

**The Basics**
The value that is returned by the DEQUOTE function is determined as follows:

- If the first character of `string` is not a single or double quotation mark, DEQUOTE returns `string` unchanged.
- If the first two characters of `string` are both single quotation marks or both double quotation marks, and the third character is not the same type of quotation mark, then DEQUOTE returns a result with a length of zero.
- If the first character of `string` is a single quotation mark, the DEQUOTE function removes that single quotation mark from the result. DEQUOTE then scans `string` from left to right, looking for more single quotation marks. Each pair of consecutive, single quotation marks is reduced to one single quotation mark. The first single quotation mark that does not have an ending quotation mark in `string` is removed and all characters to the right of that quotation mark are also removed.
- If the first character of `string` is a double quotation mark, the DEQUOTE function removes that double quotation mark from the result. DEQUOTE then scans `string` from left to right, looking for more double quotation marks. Each pair of consecutive, double quotation marks is reduced to one double quotation mark. The first double quotation mark that does not have an ending quotation mark in `string` is removed and all characters to the right of that quotation mark are also removed.

*Note:* If `string` is a constant enclosed in quotation marks, those quotation marks are not part of the value of `string`. Therefore, you do not need to use DEQUOTE to remove the quotation marks that denote a constant.

Some language elements are supported in SAS but not in CAS. An example is the DATALINES statement. Some examples in this document use DATALINES in SAS to create a data set using the CAS engine. For more information, see “Functions in SAS and CAS” on page 54.

**Example**
This example demonstrates how quotation marks in the DEQUOTE function are handled within a DATA step.

```sas
data test;
  input string $80.;
  result=dequote(string);
  datalines;
No quotation marks, no change
No "leading" quotation marks, no change
"Matching double quotation marks are removed"
'Matching single quotation marks are removed'
"Paired ""quotation marks"" are reduced"
'Paired '' quotation marks '' are reduced'
"Single 'quotation marks' inside ""double"" quotation marks are unchanged"
'Double "quotation marks" inside ""single"" quotation marks are unchanged'
"No matching quotation mark, no problem
Don't remove this apostrophe
```
**DEVIANCE Function**

Returns the deviance based on a probability distribution.

- **Category:** Mathematical

---

**Syntax**

```
DEVIANCE(distribution, variable, shape-parameters <, ε >)
```

---

**Required Arguments**

- **distribution**
  is a character constant, variable, or expression that identifies the distribution. Valid distributions are listed in the following table:

---

### Table: Input Strings and Output Results from DEQUOTE

<table>
<thead>
<tr>
<th>string</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>No quotation marks, no change</td>
<td>No quotation marks, no change</td>
</tr>
<tr>
<td>No &quot;leading&quot; quotation marks, no change</td>
<td>No &quot;leading&quot; quotation marks, no change</td>
</tr>
<tr>
<td>&quot;Matching double quotation marks are removed&quot;</td>
<td>Matching double quotation marks are removed</td>
</tr>
<tr>
<td>Matching single quotation marks are removed</td>
<td>Matching single quotation marks are removed</td>
</tr>
<tr>
<td>&quot;Paired &quot; quotation marks&quot; are reduced</td>
<td>Paired &quot; quotation marks&quot; are reduced</td>
</tr>
<tr>
<td>Paired &quot; quotation marks &quot; are reduced</td>
<td>Paired &quot; quotation marks &quot; are reduced</td>
</tr>
<tr>
<td>&quot;Single ' quotation marks' inside &quot; double&quot; quotation marks</td>
<td>Single ' quotation marks' inside &quot; double&quot; quotation marks</td>
</tr>
<tr>
<td>Double &quot; quotation marks&quot; inside &quot;&quot; single &quot; quotation marks</td>
<td>Double &quot; quotation marks&quot; inside &quot;&quot; single &quot; quotation marks</td>
</tr>
<tr>
<td>&quot;No matching quotation mark, no problem</td>
<td>No matching quotation mark, no problem</td>
</tr>
<tr>
<td>Don't remove this apostrophe</td>
<td>Don't remove this apostrophe</td>
</tr>
<tr>
<td>&quot;Text after the matching quotation mark&quot; is &quot;deleted&quot;</td>
<td>Text after the matching quotation mark</td>
</tr>
</tbody>
</table>
### Distribution

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bernoulli</td>
<td>'BERNOULLI'</td>
</tr>
<tr>
<td>Binomial</td>
<td>'BINOMIAL'</td>
</tr>
<tr>
<td>Gamma</td>
<td>'GAMMA'</td>
</tr>
<tr>
<td>Inverse Gauss (Wald)</td>
<td>'IGAUSS'</td>
</tr>
<tr>
<td>Normal</td>
<td>'NORMAL'</td>
</tr>
<tr>
<td>Poisson</td>
<td>'POISSON'</td>
</tr>
</tbody>
</table>

**variable**
- is a numeric constant, variable, or expression.

**shape-parameter**
- are one or more distribution-specific numeric parameters that characterize the shape of the distribution.

### Optional Argument

**ε**
- is an optional numeric small value used for all of the distributions, except for the normal distribution.

### Details

#### The Bernoulli Distribution

**DEVIANCE('BERNOULLI', variable, p, ε)**

**Arguments**

**variable**
- is a binary numeric random variable that has the value of 1 for success and 0 for failure.

**p**
- is a numeric probability of success with $\epsilon \leq p \leq 1 - \epsilon$.

**ε**
- is an optional positive numeric value that is used to bound $p$. Any value of $p$ in the interval $0 \leq p \leq \epsilon$ is replaced by $\epsilon$. Any value of $p$ in the interval $1 - \epsilon \leq p \leq 1$ is replaced by $1 - \epsilon$.

The DEVIANCE function returns the deviance from a Bernoulli distribution with a probability of success $p$, where success is defined as a random variable value of 1. The equation follows:

$$
\text{DEVIANCE('BERN', variable, p, ε)} = \begin{cases} 
-2\log(1 - p) & x = 0 \\
-2\log(p) & x = 1 \\
\cdot & \text{otherwise}
\end{cases}
$$
The Binomial Distribution
DEVIANCE("BINO", variable, μ, n <, ε>)

Arguments
variable
is a numeric random variable that contains the number of successes.
Range 0 ≤ variable ≤ 1

μ
is a numeric mean parameter.
Range ne ≤ μ ≤ n(1−ε)

n
is an integer number of Bernoulli trials parameter
Range n ≥ 0

ε
is an optional positive numeric value that is used to bound μ. Any value of μ in the interval 0 ≤ μ ≤ ne is replaced by ne. Any value of μ in the interval n(1−ε) ≤ μ ≤ n is replaced by n(1−ε).

The DEVIANCE function returns the deviance from a binomial distribution, with a probability of success p, and a number of independent Bernoulli trials n. The following equation describes the DEVIANCE function for the Binomial distribution, where x is the random variable.

DEVIANCE(′ BINO′, x, μ, n) = \[
\begin{align*}
\frac{2}{n} \log \left( \frac{x}{μ} \right) + \left( n - x \right) \log \left( \frac{n - x}{n - μ} \right) & \quad 0 ≤ x ≤ n \\
\frac{2}{n} \log \left( \frac{x}{μ} \right) & \quad x < 0 \\
\frac{2}{n} \log \left( \frac{x}{μ} \right) & \quad x > n
\end{align*}
\]

The Gamma Distribution
DEVIANCE(′GAMMA′, variable, μ <, ε>)

Arguments
variable
is a numeric random variable.
Range variable ≥ ε

μ
is a numeric mean parameter.
Range μ ≥ ε

ε
is an optional positive numeric value that is used to bound variable and μ. Any value of variable in the interval 0 ≤ variable ≤ ε is replaced by ε. Any value of μ in the interval 0 ≤ μ ≤ ε is replaced by ε.

The DEVIANCE function returns the deviance from a gamma distribution with a mean parameter μ. The following equation describes the DEVIANCE function for the gamma distribution, where x is the random variable:

DEVIANCE(′ GAMMA′, x, μ, n) = \[
\begin{align*}
\frac{2}{n} x \log \left( \frac{x}{μ} \right) + \left( n - x \right) \log \left( \frac{n - x}{n - μ} \right) & \quad 0 ≤ x ≤ n \\
\frac{2}{n} x \log \left( \frac{x}{μ} \right) & \quad x < 0 \\
\frac{2}{n} x \log \left( \frac{x}{μ} \right) & \quad x > n
\end{align*}
\]
The Inverse Gauss (Wald) Distribution

\[
\text{DEVIANCE}(\text{'IGAUSS' | 'WALD', variable, } \mu, \varepsilon) = \begin{cases} 
2 \left( - \log \left( \frac{x}{\mu} \right) + \frac{x - \mu}{\mu} \right) & x \geq \varepsilon, \mu \geq \varepsilon \\
\end{cases}
\]

Arguments

\textit{variable}

is a numeric random variable.

Range \hspace{1cm} variable \geq \varepsilon

\textit{\( \mu \)}

is a numeric mean parameter.

Range \hspace{1cm} \mu \geq \varepsilon

\textit{\( \varepsilon \)}

is an optional positive numeric value that is used to bound \( \text{variable} \) and \( \mu \). Any value of \( \text{variable} \) in the interval \( 0 \leq \text{variable} \leq \varepsilon \) is replaced by \( \varepsilon \). Any value of \( \mu \) in the interval \( 0 \leq \mu \leq \varepsilon \) is replaced by \( \varepsilon \).

The DEVIANCE function returns the deviance from an inverse Gaussian distribution with a mean parameter \( \mu \). The following equation describes the DEVIANCE function for the inverse Gaussian distribution, where \( x \) is the random variable:

\[
\text{DEVIANCE}(\text{'IGAUSS'}, x, \mu) = \begin{cases} 
\cdot & x < 0 \\
\frac{(x - \mu)^2}{\mu^2 x} & x \geq \varepsilon, \mu \geq \varepsilon \\
\end{cases}
\]

The Normal Distribution

\[
\text{DEVIANCE}(\text{'NORMAL' | 'GAUSSIAN', variable, } \mu) = (x - \mu)^2
\]

Arguments

\textit{variable}

is a numeric random variable.

\textit{\( \mu \)}

is a numeric mean parameter.

The DEVIANCE function returns the deviance from a normal distribution with a mean parameter \( \mu \). The following equation describes the DEVIANCE function for the normal distribution, where \( x \) is the random variable:

\[
\text{DEVIANCE}(\text{'NORMAL'}, x, \mu) = (x - \mu)^2
\]

The Poisson Distribution

\[
\text{DEVIANCE}(\text{'POISSON', variable, } \mu, \varepsilon) = (x - \mu)^2
\]

Arguments

\textit{variable}

is a numeric random variable.
Range variable $\geq 0$

$\mu$

is a numeric mean parameter.

Range $\mu \geq \varepsilon$

$\varepsilon$

is an optional positive numeric value that is used to bound $\mu$. Any value of $\mu$ in the interval $0 \leq \mu \leq \varepsilon$ is replaced by $\varepsilon$.

The DEVIANCE function returns the deviance from a Poisson distribution with a mean parameter $\mu$. The following equation describes the DEVIANCE function for the Poisson distribution, where $x$ is the random variable:

$$\text{DEVIANCE}(\text{POISSON}', x, \mu) = \begin{cases} 
 x \log \left( \frac{x}{\mu} \right) - (x - \mu) & x \geq 0, \mu \geq \varepsilon \\
 0 & \text{otherwise}
\end{cases}$$

**DHMS Function**

Returns a SAS datetime value from date, hour, minute, and second values.

**Category:** Date and Time

**Syntax**

\[ \text{DHMS}(\text{date}, \text{hour}, \text{minute}, \text{second}) \]

**Required Arguments**

- **date**: specifies a SAS expression that represents a SAS date value.
- **hour**: is numeric.
- **minute**: is numeric.
- **second**: is numeric.

**Details**

The DHMS function returns a numeric value that represents a SAS datetime value. This numeric value can be either positive or negative.

You can also use the DHMS function to combine a SAS date value with a SAS time value to produce a SAS time value. Because a SAS time value is stored in seconds, you can specify 0 for the hour variable and 0 for the minute variable to return the correct value. Here is the syntax:

\[ \text{DHMS}(\text{SAS date}, 0, 0, \text{SAS time}) \]
The fourth example shows this functionality.

### Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>dtid=dhms('01jan03'd, 15, 30, 15); put dtid; put dtid datetime.;</td>
<td>1357054215 01JAN03:15:30:15</td>
</tr>
<tr>
<td>dtid2=dhms('01jan03'd, 15, 30, 61); put dtid2; put dtid2 datetime.;</td>
<td>1357054261 01JAN03:15:31:01</td>
</tr>
<tr>
<td>dtid3=dhms('01jan03'd, 15, .5, 15); put dtid3; put dtid3 datetime.;</td>
<td>1357052445 01JAN03:15:00:45</td>
</tr>
</tbody>
</table>

The following SAS statements show how to combine a SAS date value with a SAS time value into a SAS datetime value. If you execute these statements on April 2, 2003, at the time of 15:05:02, the DHMS function produces these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>day=date(); time=time(); sasdt=dhms(day, 0, 0, time); put sasdt datetime.;</td>
<td>02APR03:15:05:02</td>
</tr>
</tbody>
</table>

### See Also

#### Functions:
- “HMS Function” on page 322

---

### DIF Function

Returns differences between an argument and its \( n \)th lag.

#### Category:
- Special

#### Syntax

\[
\text{DIF}^{<n>} \ (\text{argument})
\]
**Required Argument**

*argument*

specifies a numeric constant, variable, or expression.

**Optional Argument**

*n*

specifies the number of lags.

**Details**

The DIF functions, DIF1, DIF2, ..., DIF100, return the first differences between the argument and its *n*th lag. DIF1 can also be written as DIF. DIF*n* is defined as $DIF_n(x) = x - \text{LAG}_n(x)$.

For details about storing and returning values from the LAG*n* queue, see the LAG function.

Some language elements are supported in SAS but not in CAS. An example is the DATALINES statement. Some examples in this document use DATALINES in SAS to create a data set using the CAS engine. For more information, see “Functions in SAS and CAS” on page 54.

**Comparisons**

The function DIF2(X) is not equivalent to the second difference DIF(DIF(X)).

**Example**

This example demonstrates the difference between the LAG and DIF functions.

```sas
data two;
  input X @@;
  Z=lag(x);
  D=dif(x);
  datalines;
1 2 6 4 7
;
proc print data=two;
run;
```
See Also

Functions:
- “LAG Function” on page 393

**DIGAMMA Function**

Returns the value of the digamma function.

**Category:** Mathematical

**Syntax**

```
DIGAMMA(argument)
```

**Required Argument**

*argument*

specifies a numeric constant, variable, or expression.

**Restriction**

Nonpositive integers are invalid.

**Details**

The DIGAMMA function returns the ratio that is given by

\[ \Psi(x) = \frac{\Gamma'(x)}{\Gamma(x)} \]

where \( \Gamma(.) \) and \( \Gamma'(.) \) denote the Gamma function and its derivative, respectively. For \( argument \geq 0 \), the DIGAMMA function is the derivative of the LGAMMA function.
Example

The following SAS statement produces this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=digamma(1.0);</td>
<td>-0.577215665</td>
</tr>
</tbody>
</table>

DIM Function

Returns the number of elements in an array.

Category: Array

Syntax

- \texttt{DIM} (<n> \texttt{array-name})
- \texttt{DIM}(\texttt{array-name}, \texttt{bound-n})

Required Arguments

- \texttt{array-name}
  - specifies the name of an array that was previously defined in the same DATA step. This argument cannot be a constant, variable, or expression.
- \texttt{bound-n}
  - is a numeric constant, variable, or expression that specifies the dimension, in a multidimensional array, for which you want to know the number of elements. Use \texttt{bound-n} only when \texttt{n} is not specified.

Optional Argument

- \texttt{n}
  - specifies the dimension, in a multidimensional array, for which you want to know the number of elements. If no \texttt{n} value is specified, the DIM function returns the number of elements in the first dimension of the array.

Details

The DIM function returns the number of elements in a one-dimensional array or the number of elements in a specified dimension of a multidimensional array when the lower bound of the dimension is 1. Use DIM in array processing to avoid changing the upper bound of an iterative DO group each time you change the number of array elements.

Comparisons

- DIM always returns a total count of the number of elements in an array dimension.
- HBORDER returns the literal value of the upper bound of an array dimension.

Note: This distinction is important when the lower bound of an array dimension has a value other than 1 and the upper bound has a value other than the total number of elements in the array dimension.
Examples

Example 1: One-dimensional Array
In this example, DIM returns a value of 5. Therefore, SAS repeats the statements in the DO loop five times.

```sas
array big{5} weight sex height state city;
do i=1 to dim(big);
    ... more SAS statements ...;
end;
```

Example 2: Multidimensional Array
This example shows two ways of specifying the DIM function for multidimensional arrays. Both methods return the same value for DIM, as shown in the table that follows the SAS code example.

```sas
array mult{5, 10, 2} mult1-mult100;
```

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Alternative Syntax</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIM(MULT)</td>
<td>DIM(MULT,1)</td>
<td>5</td>
</tr>
<tr>
<td>DIM2(MULT)</td>
<td>DIM(MULT,2)</td>
<td>10</td>
</tr>
<tr>
<td>DIM3(MULT)</td>
<td>DIM(MULT,3)</td>
<td>2</td>
</tr>
</tbody>
</table>

See Also

Functions:
- “HBOUND Function” on page 320
- “LBOUND Function” on page 402

Statements:
- “ARRAY Statement” in SAS Viya Statements: Reference
- “Array Reference Statement” in SAS Viya Statements: Reference

DIVIDE Function

Returns the result of a division that handles special missing values for ODS output.

**Category:** Arithmetic

**Syntax**

```sas
DIVIDE(x, y)
```
**Required Arguments**

\(x\) is a numeric constant, variable, or expression.

\(y\) is a numeric constant, variable, or expression.

**Details**

The **DIVIDE** function divides two numbers and returns a result that is compatible with ODS conventions. The function handles special missing values for ODS output. The following list shows how certain special missing values are interpreted in ODS:

- .I as infinity
- .M as minus infinity
- _ as a blank

The following table shows the values that are returned by the **DIVIDE** function, based on the values of \(x\) and \(y\).

**Figure 2.15 Values That Are Returned by the DIVIDE Function**

<table>
<thead>
<tr>
<th>(y)</th>
<th>positive</th>
<th>zero</th>
<th>negative</th>
<th>.I</th>
<th>.M</th>
<th>_</th>
<th>other</th>
</tr>
</thead>
<tbody>
<tr>
<td>positive</td>
<td>(x/y) or .I</td>
<td>0</td>
<td>(x/y) or .M</td>
<td>.I</td>
<td>.M</td>
<td>_</td>
<td>(x)</td>
</tr>
<tr>
<td>zero</td>
<td>.I</td>
<td>.</td>
<td>.M</td>
<td>.I</td>
<td>.M</td>
<td>_</td>
<td>(x)</td>
</tr>
<tr>
<td>negative</td>
<td>(x/y) or .M</td>
<td>0</td>
<td>(x/y) or .I</td>
<td>.M</td>
<td>.I</td>
<td>_</td>
<td>(x)</td>
</tr>
<tr>
<td>.I</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>.</td>
<td>.</td>
<td>_</td>
<td>(x)</td>
</tr>
<tr>
<td>.M</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>.</td>
<td>.</td>
<td>_</td>
<td>(x)</td>
</tr>
<tr>
<td>_</td>
<td>_</td>
<td>_</td>
<td>_</td>
<td>_</td>
<td>_</td>
<td>_</td>
<td>_</td>
</tr>
</tbody>
</table>

| other | \(y\) | \(y\) | \(y\) | \(y\) | _ | \(x\) |

**Note:** The **DIVIDE** function never writes a note to the SAS log regarding missing values, division by zero, or overflow.

**Example**

The following example shows the results of using the **DIVIDE** function.

```sas
data _null_
  a=divide(1, 0)
  b=divide(2, .I)
  c=divide(.I, -1)
  d=divide(constant('big'), constant('small'))
run;
```

---

**Figure 2.15 Values That Are Returned by the DIVIDE Function**

<table>
<thead>
<tr>
<th>(y)</th>
<th>positive</th>
<th>zero</th>
<th>negative</th>
<th>.I</th>
<th>.M</th>
<th>_</th>
<th>other</th>
</tr>
</thead>
<tbody>
<tr>
<td>positive</td>
<td>(x/y) or .I</td>
<td>0</td>
<td>(x/y) or .M</td>
<td>.I</td>
<td>.M</td>
<td>_</td>
<td>(x)</td>
</tr>
<tr>
<td>zero</td>
<td>.I</td>
<td>.</td>
<td>.M</td>
<td>.I</td>
<td>.M</td>
<td>_</td>
<td>(x)</td>
</tr>
<tr>
<td>negative</td>
<td>(x/y) or .M</td>
<td>0</td>
<td>(x/y) or .I</td>
<td>.M</td>
<td>.I</td>
<td>_</td>
<td>(x)</td>
</tr>
<tr>
<td>.I</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>.</td>
<td>.</td>
<td>_</td>
<td>(x)</td>
</tr>
<tr>
<td>.M</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>.</td>
<td>.</td>
<td>_</td>
<td>(x)</td>
</tr>
<tr>
<td>_</td>
<td>_</td>
<td>_</td>
<td>_</td>
<td>_</td>
<td>_</td>
<td>_</td>
<td>_</td>
</tr>
<tr>
<td>other</td>
<td>(y)</td>
<td>(y)</td>
<td>(y)</td>
<td>(y)</td>
<td>_</td>
<td>(x)</td>
<td></td>
</tr>
</tbody>
</table>
SAS writes the following output to the log:

```
a=I (infinity)
b=0
c=M (minus infinity)
d=I (infinity because of overflow)
```

---

**DUR Function**

Returns the modified duration for an enumerated cash flow.

**Category:** Financial

**Syntax**

```
DUR(y, f, c(1), ..., c(k))
```

**Required Arguments**

- `y`
  - specifies the effective per-period yield-to-maturity, expressed as a fraction.
  - Range: `y > 0`

- `f`
  - specifies the frequency of cash flows per period.
  - Range: `f > 0`

- `c(1), ..., c(k)`
  - specifies a list of cash flows.

**Details**

The DUR function returns the value

```
C = \sum_{k=1}^{K} \frac{c(k)}{(1+y)^{f}}
```

The following relationship applies to the preceding equation:

```
P = \sum_{k=1}^{K} \frac{c(k)}{(1+y)^{f}}
```

**Example**

```plaintext
data _null_
   d=dur(1/20, 1, .33, .44, .55, .49, .50, .22, .4, .8, .01, .36, .2, .4);
   put d;
```
DURP Function

Returns the modified duration for a periodic cash flow stream, such as a bond.

Category: Financial

Syntax

\[
\text{DURP}(A, c, n, K, k_0, y)
\]

Required Arguments

\(A\)

specifies the par value.

Range \(A > 0\)

\(c\)

specifies the nominal per-period coupon rate, expressed as a fraction.

Range \(0 \leq c < 1\)

\(n\)

specifies the number of coupons per period.

Range \(n > 0\) and is an integer

\(K\)

specifies the number of remaining coupons.

Range \(K > 0\) and is an integer

\(k_0\)

specifies the time from the present date to the first coupon date, expressed in terms of the number of periods.

Range \(0 < k_0 \leq 1/n\)

\(y\)

specifies the nominal per-period yield-to-maturity, expressed as a fraction.

Range \(y > 0\)

Details

The DURP function returns the value
The following relationships apply to the preceding equation:

- \( t_k = n k_0 + k - 1 \)
- \( c(k) = \frac{c}{\frac{t}{n}} A \) for \( k = 1, \ldots, K - 1 \)
- \( c(K) = \left(1 + \frac{c}{\frac{t}{n}}\right) A \)

The following relationship applies to the preceding equation:

\[
P = \frac{\sum_{k=1}^{K} c(k)}{P \left(1 + \frac{\frac{t}{n}}{k}\right)}
\]

Example

```sas
data _null_;
d=durp(1000, 1/100, 4, 14, .33/2, .10);
put d;
run;
```

SAS writes the following output to the log:

```
3.2649588109
```

**EFFRATE Function**

Returns the effective annual interest rate.

**Category:** Financial

**Syntax**

`EFFRATE(compounding-interval, rate)`

**Required Arguments**

- `compounding-interval`
  - is a SAS interval. This value represents how often `rate` compounds.

- `rate`
  - is numeric. `Rate` is a nominal annual interest rate (expressed as a percentage) that is compounded at each compounding interval.
Details

The EFFRATE function returns the effective annual interest rate. The function computes the effective annual interest rate that corresponds to a nominal annual interest rate.

The following details apply to the EFFRATE function:

- The values for rates must be at least –99.
- In considering a nominal interest rate and a compounding interval, if `compounding-interval` is 'CONTINUOUS', then the value that is returned by EFFRATE equals $e^{\frac{\text{rate}}{100}} - 1$.

  If `compounding-interval` is not 'CONTINUOUS', and $m$ compounding intervals occur in a year, the value that is returned by EFFRATE equals $(1 + \frac{\text{rate}}{100 \cdot m})^m - 1$.

- The following values are valid for `compounding-interval`:
  - 'CONTINUOUS'
  - 'DAY'
  - 'SEMIMONTH'
  - 'MONTH'
  - 'QUARTER'
  - 'SEMIYEAR'
  - 'YEAR'
  - If the interval is 'DAY', then $m = 365$.

Example

The following examples show how the effective rate is calculated:

- If a nominal rate is 10%, then the corresponding effective rate when interest is compounded monthly can be expressed as

  \[
  \text{effective_rate1} = \text{EFFRATE('MONTH', 10)};
  \]

- If a nominal rate is 10%, then the corresponding effective rate when interest is compounded quarterly can be expressed as

  \[
  \text{effective-rate2} = \text{EFFRATE('QUARTER', 10)};
  \]

---

ERF Function

Returns the value of the (normal) error function.

**Category:** Mathematical

**Syntax**

\[
\text{ERF}(\text{argument})
\]

**Required Argument**

$\text{argument}$

specifies a numeric constant, variable, or expression.
Details

The ERF function returns the integral, given by

\[ \text{ERF}(x) = \frac{2}{\sqrt{\pi}} \int_{0}^{x} e^{-z^2} dz \]

Example

You can use the ERF function to find the probability (p) that a normally distributed random variable with mean 0 and standard deviation will take on a value less than X. For example, the quantity that is given by the following statement is equivalent to PROBNORM(X):

\[ p = 0.5 + 0.5 \times \text{erf}(x/\sqrt{2}); \]

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>y=erf(1.0);</td>
<td>0.8427007929</td>
</tr>
<tr>
<td>y=erf(-1.0);</td>
<td>-0.842700793</td>
</tr>
</tbody>
</table>

ERFC Function

Returns the value of the complementary (normal) error function.

Category:   Mathematical

Syntax

ERFC(argument)

Required Argument

\[ \text{argument} \]

specifies a numeric constant, variable, or expression.

Details

The ERFC function returns the complement to the ERF function (that is, \( 1 - \text{ERF}(argument) \)).

Example

The following SAS statements produce these results:
### EXP Function

Returns the value of the exponential function.

**Category:** Mathematical

**Syntax**

\[
\text{EXP}(\text{argument})
\]

**Required Argument**

- **argument** specifies a numeric constant, variable, or expression.

**Details**

The EXP function raises the constant \( e \), which is approximately 2.71828, to the power that is supplied by the argument. The result is limited by the maximum value of a floating-point value on the computer.

**Example**

The following SAS statements produce these results.

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x=\text{exp}(1.0); )</td>
<td>2.7182818285</td>
</tr>
<tr>
<td>( x=\text{exp}(0); )</td>
<td>1</td>
</tr>
</tbody>
</table>

### FACT Function

Computes a factorial.

**Category:** Mathematical

**Syntax**

\[
\text{FACT}(n)
\]
Required Argument

\( n \)

is a numeric constant, variable, or expression.

Details

The mathematical representation of the FACT function is given by the following equation:

\[
FACT(n) = n!
\]

In this equation, \( n \geq 0 \).

If the expression cannot be computed, a missing value is returned. For moderately large values, it is sometimes not possible to compute the FACT function.

Example

The following SAS statement produces this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=fact(5);</td>
<td>120</td>
</tr>
</tbody>
</table>

See Also

Functions:
- “COMB Function” on page 183
- “LFACT Function” on page 411
- “PERM Function” on page 497

FINANCE Function

Computes financial calculations such as depreciation, maturation, accrued interest, net present value, periodic savings, and internal rates of return.

Category: Financial

Syntax

\[
\text{FINANCE}(\text{string-identifier, parameter}_1, \text{parameter}_2, \ldots)
\]

Required Arguments

\( \text{string-identifier} \)

specifies a character constant, variable, or expression. Valid values for \( \text{string-identifier} \) are listed in the following table.
<table>
<thead>
<tr>
<th>string-identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>“ACCRINT” on page 247</td>
<td>computes the accrued interest for a security that pays periodic interest.</td>
</tr>
<tr>
<td>“ACCRINTM” on page 248</td>
<td>computes the accrued interest for a security that pays interest at maturity.</td>
</tr>
<tr>
<td>“AMORDEGRC” on page 248</td>
<td>computes the depreciation for each accounting period by using a depreciation coefficient.</td>
</tr>
<tr>
<td>“AMORLINC” on page 249</td>
<td>computes the depreciation for each accounting period.</td>
</tr>
<tr>
<td>“COUPDAYBS” on page 249</td>
<td>computes the number of days from the beginning of the coupon period to the settlement date.</td>
</tr>
<tr>
<td>“COUPDAYS” on page 250</td>
<td>computes the number of days in the coupon period that contains the settlement date.</td>
</tr>
<tr>
<td>“COUPDAYSNC” on page 250</td>
<td>computes the number of days from the settlement date to the next coupon date.</td>
</tr>
<tr>
<td>“COUPNCD” on page 251</td>
<td>computes the next coupon date after the settlement date.</td>
</tr>
<tr>
<td>“COUPNUM” on page 251</td>
<td>computes the number of coupons that are payable between the settlement date and the maturity date.</td>
</tr>
<tr>
<td>“COUPPCD” on page 251</td>
<td>computes the previous coupon date before the settlement date.</td>
</tr>
<tr>
<td>“CUMIPMT” on page 252</td>
<td>computes the cumulative interest that is paid between two periods.</td>
</tr>
<tr>
<td>“CUMPRINC” on page 252</td>
<td>computes the cumulative principal that is paid on a loan between two periods.</td>
</tr>
<tr>
<td>“DB” on page 253</td>
<td>computes the depreciation of an asset for a specified period by using the fixed-declining balance method.</td>
</tr>
<tr>
<td>“DDB” on page 253</td>
<td>computes the depreciation of an asset for a specified period by using the double-declining balance method or some other method that you specify.</td>
</tr>
<tr>
<td>“DISC” on page 254</td>
<td>computes the discount rate for a security.</td>
</tr>
<tr>
<td>“DOLLARDE” on page 254</td>
<td>converts a dollar price, expressed as a fraction, to a dollar price, expressed as a decimal number.</td>
</tr>
<tr>
<td>string-identifier</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>“DOLLARFR” on page 255</td>
<td>converts a dollar price, expressed as a decimal number, to a dollar price, expressed as a fraction.</td>
</tr>
<tr>
<td>“DURATION” on page 255</td>
<td>computes the annual duration of a security with periodic interest payments.</td>
</tr>
<tr>
<td>“EFFECT” on page 255</td>
<td>computes the effective annual interest rate.</td>
</tr>
<tr>
<td>“FV” on page 256</td>
<td>computes the future value of an investment.</td>
</tr>
<tr>
<td>“FVSCHEDULE” on page 256</td>
<td>computes the future value of an initial principal after applying a series of compound interest rates.</td>
</tr>
<tr>
<td>“INTRATE” on page 256</td>
<td>computes the interest rate for a fully invested security.</td>
</tr>
<tr>
<td>“IPMT” on page 257</td>
<td>computes the interest payment for an investment for a given period.</td>
</tr>
<tr>
<td>“IRR” on page 257</td>
<td>computes the internal rate of return for a series of cash flows.</td>
</tr>
<tr>
<td>“ISPMT” on page 258</td>
<td>calculates the interest paid during a specific period of an investment.</td>
</tr>
<tr>
<td>“MDURATION” on page 258</td>
<td>computes the Macaulay modified duration for a security with an assumed face value of $100.</td>
</tr>
<tr>
<td>“MIRR” on page 258</td>
<td>computes the internal rate of return where positive and negative cash flows are financed at different rates.</td>
</tr>
<tr>
<td>“NOMINAL” on page 259</td>
<td>computes the annual nominal interest rate.</td>
</tr>
<tr>
<td>“NPER” on page 259</td>
<td>computes the number of periods for an investment.</td>
</tr>
<tr>
<td>“NPV” on page 260</td>
<td>computes the net present value of an investment based on a series of periodic cash flows and a discount rate.</td>
</tr>
<tr>
<td>“ODDFPRICE” on page 260</td>
<td>computes the price per $100 face value of a security with an odd first period.</td>
</tr>
<tr>
<td>“ODDFYIELD” on page 260</td>
<td>computes the yield of a security with an odd first period.</td>
</tr>
<tr>
<td>“ODDLPRICE” on page 261</td>
<td>computes the price per $100 face value of a security with an odd last period.</td>
</tr>
<tr>
<td>string-identifier</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>“ODDLYIELD” on page 262</td>
<td>computes the yield of a security with an odd last period.</td>
</tr>
<tr>
<td>“PMT” on page 262</td>
<td>computes the periodic payment for an annuity.</td>
</tr>
<tr>
<td>“PPMT” on page 263</td>
<td>computes the payment on the principal for an investment for a given period.</td>
</tr>
<tr>
<td>“PRICE” on page 263</td>
<td>computes the price per $100 face value of a security that pays periodic interest.</td>
</tr>
<tr>
<td>“PRICEDISC” on page 264</td>
<td>computes the price per $100 face value of a discounted security.</td>
</tr>
<tr>
<td>“PRICEMAT” on page 264</td>
<td>computes the price per $100 face value of a security that pays interest at maturity.</td>
</tr>
<tr>
<td>“PV” on page 265</td>
<td>computes the present value of an investment.</td>
</tr>
<tr>
<td>“RATE” on page 265</td>
<td>computes the interest rate per period of an annuity.</td>
</tr>
<tr>
<td>“RECEIVED” on page 266</td>
<td>computes the amount received at maturity for a fully invested security.</td>
</tr>
<tr>
<td>“SLN” on page 266</td>
<td>computes the straight-line depreciation of an asset for one period.</td>
</tr>
<tr>
<td>“SYD” on page 267</td>
<td>computes the sum-of-years digits depreciation of an asset for a specified period.</td>
</tr>
<tr>
<td>“TBILLEQ” on page 267</td>
<td>computes the bond-equivalent yield for a treasury bill.</td>
</tr>
<tr>
<td>“TBILLPRICE” on page 267</td>
<td>computes the price per $100 face value for a treasury bill.</td>
</tr>
<tr>
<td>“TBILLYIELD” on page 268</td>
<td>computes the yield for a treasury bill.</td>
</tr>
<tr>
<td>“VDB” on page 268</td>
<td>computes the depreciation of an asset for a specified or partial period by using a declining balance method.</td>
</tr>
<tr>
<td>“XIRR” on page 268</td>
<td>computes the internal rate of return for a schedule of cash flows that is not necessarily periodic.</td>
</tr>
<tr>
<td>“XNPV” on page 269</td>
<td>computes the net present value for a schedule of cash flows that is not necessarily periodic.</td>
</tr>
<tr>
<td>“YIELD” on page 269</td>
<td>computes the yield on a security that pays periodic interest.</td>
</tr>
</tbody>
</table>
**string-identifier**

<table>
<thead>
<tr>
<th>string-identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>“YIELDDISC” on page 270</td>
<td>computes the annual yield for a discounted security (for example, a treasury bill).</td>
</tr>
<tr>
<td>“YIELDMAT” on page 270</td>
<td>computes the annual yield of a security that pays interest at maturity.</td>
</tr>
</tbody>
</table>

**parameter**

specifies a parameter that is associated with each string-identifier. The following parameters are available:

**basis**

is an optional parameter that specifies a character or numeric value that indicates the type of day count basis to use.

<table>
<thead>
<tr>
<th>Numeric Value</th>
<th>String Value</th>
<th>Day Count Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>&quot;30/360&quot;</td>
<td>US (NASD) 30/360</td>
</tr>
<tr>
<td>1</td>
<td>&quot;ACTUAL&quot;</td>
<td>Actual/actual</td>
</tr>
<tr>
<td>2</td>
<td>&quot;ACT/360&quot;</td>
<td>Actual/360</td>
</tr>
<tr>
<td>3</td>
<td>&quot;ACT/365&quot;</td>
<td>Actual/365</td>
</tr>
<tr>
<td>4</td>
<td>&quot;EU30/360&quot;</td>
<td>European 30/360</td>
</tr>
</tbody>
</table>

**interest-rates**

specifies rates that are provided as numeric values and not as percentages.

**dates**

specifies that all dates in the financial functions are SAS dates.

**sign-of-cash-values**

for all the arguments, specifies that the cash that you pay out, such as deposits to savings or other withdrawals, is represented by negative numbers. It also specifies that the cash that you receive, such as dividend checks and other deposits, is represented by positive numbers.

**Details**

**ACCRINT**

Computes the accrued interest for a security that pays periodic interest.

\[
\text{FINANCE('ACCRINT', issue, first-interest, settlement, rate, par-value, frequency, <basis>)};
\]

**Arguments**

**issue**

specifies the issue date of the security.

**first-interest**

specifies the first interest date of the security.
settlement
  specifies the settlement date.

rate
  specifies the interest rate.

par-value
  specifies the par value of the security. If you omit par-value, SAS uses the value $1000.

frequency
  specifies the number of coupon payments per year. For annual payments, frequency=1; for semiannual payments, frequency=2; for quarterly payments, frequency=4.

basis
  specifies the optional day count value.

Featured in:
  “Example 1: Computing Accrued Interest: ACCRINT” on page 271

ACCRINTM
Computes the accrued interest for a security that pays interest at maturity.

FINANCE('ACCRINTM', issue, settlement, rate, par-value, <basis>);

Arguments
issue
  specifies the issue date of the security.

settlement
  specifies the settlement date.

rate
  specifies the interest rate.

par-value
  specifies the par value of the security. If you omit par-value, SAS uses the value $1000.

basis
  specifies the optional day count value.

Featured in:
  “Example 2: Computing Accrued Interest: ACCRINTM” on page 271

AMORDEGRC
Computes the depreciation for each accounting period by using a depreciation coefficient.

FINANCE('AMORDEGRC', cost, date-purchased, first-period, salvage, period, rate, <basis>);

Arguments
cost
  specifies the initial cost of the asset.

date-purchased
  specifies the date of the purchase of the asset.

first-period
  specifies the date of the end of the first period.
**AMORLINC**
Computes the depreciation for each accounting period.

```
FINANCE('AMORLINC', cost, date-purchased, first-period, salvage, period, rate, <basis>);
```

**Arguments**
- `cost`: specifies the initial cost of the asset.
- `date-purchased`: specifies the date of the purchase of the asset.
- `first-period`: specifies the date of the end of the first period.
- `salvage`: specifies the value at the end of the depreciation (also called the salvage value of the asset).
- `period`: specifies the depreciation period.
- `rate`: specifies the rate of depreciation.
- `basis`: specifies the optional day count value.

**TIP** When the first argument of the FINANCE function is AMORLINC and the value of `basis` is 2, the function returns a missing value.

Featured in:
“Example 4: Computing Description: AMORLINC” on page 272

**COUPDAYBS**
Computes the number of days from the beginning of the coupon period to the settlement date.

```
FINANCE('COUPDAYBS', settlement, maturity, frequency, <basis>);
```

**Arguments**
settlement
specifies the settlement date of the security. The security settlement date is the date after the issue date when the security is traded to the buyer.

maturity
specifies the maturity date of the security. The maturity date is the date on which the security expires.

classify specifies the number of coupon payments per year. For annual payments, 
ofequenty=1; for semiannual payments, frequency=2; for quarterly payments, 
ofequenty=4.

basis
specifies the type of day count basis to use.

Featured in:
“Example 5: Computing Description: COUPDAYBS” on page 272

Note: Dates should be entered using the DATE function, or as results of other formulas or functions. For example, use
DATE(2011,5,23) for the 23rd day of May 2011. Problems can occur if dates are entered as text.

COUPDAYS
Computes the number of days in the coupon period that contains the settlement date.

FINANCE(’COUPDAYS’, settlement, maturity, frequency, <basis>);

Arguments

settlement
specifies the settlement date.

maturity
specifies the maturity date.

classify specifies the number of coupon payments per year. For annual payments, 
ofequenty=1; for semiannual payments, frequency=2; for quarterly payments, 
ofequenty=4.

basis
specifies the optional day count value.

Featured in:
“Example 6: Computing Description: COUPDAYS” on page 272

COUPDAYSNC
Computes the number of days from the settlement date to the next coupon date.

FINANCE(’COUPDAYSNC’, settlement, maturity, frequency, <basis>);

Arguments

settlement
specifies the settlement date.

maturity
specifies the maturity date.
frequency
specifies the number of coupon payments per year. For annual payments,
frequency=1; for semiannual payments, frequency=2; for quarterly payments,
frequency=4.

basis
specifies the optional day count value.

Featured in:
“Example 7: Computing Description: COUPDAYSNC” on page 272

**COUPNCD**
Computes the next coupon date after the settlement date.

FINANCE('COUPNCD', settlement, maturity, frequency, <basis>);

**Arguments**

*settlement*
specifies the settlement date.

*maturity*
specifies the maturity date.

*frequency*
specifies the number of coupon payments per year. For annual payments,
frequency=1; for semiannual payments, frequency=2; for quarterly payments,
frequency=4.

*basis*
specifies the optional day count value.

Featured in:
“Example 8: Computing Description: COUPNCD” on page 273

**COUPNUM**
Computes the number of coupons that are payable between the settlement date and the
maturity date.

FINANCE('COUPNUM', settlement, maturity, frequency, <basis>);

**Arguments**

*settlement*
specifies the settlement date.

*maturity*
specifies the maturity date.

*frequency*
specifies the number of coupon payments per year. For annual payments,
frequency=1; for semiannual payments, frequency=2; for quarterly payments,
frequency=4.

*basis*
specifies the optional day count value.

Featured in:
“Example 9: Computing Description: COUPNUM” on page 273

**COUPPCD**
Computes the previous coupon date before the settlement date.
FINANCE('COUPPCD', settlement, maturity, frequency, <basis>);

Arguments
settlement
  specifies the settlement date.
maturity
  specifies the maturity date.
frequency
  specifies the number of coupon payments per year. For annual payments, frequency=1; for semiannual payments, frequency=2; for quarterly payments, frequency=4.
basis
  specifies the optional day count value.

Featured in:
  “Example 10: Computing Description: COUPPCD” on page 273

CUMIPMT
Computes the cumulative interest paid between two periods.
FINANCE('CUMIPMT', rate, nper, pv, start-period, end-period, <type>);

Arguments
rate
  specifies the interest rate.
nper
  specifies the total number of payment periods.
pv
  specifies the present value or the lump-sum amount that a series of future payments is worth currently.
start-period
  specifies the first period in the calculation. Payment periods are numbered beginning with 1.
end-period
  specifies the last period in the calculation.
type
  specifies the number 0 or 1 and indicates when payments are due. If type is omitted, it is assumed to be 0.

If payments are due at the end of the period, then either omit the type argument or set it to 0. If payments are due at the beginning of the period, then set type to 1.

Featured in:
  “Example 11: Computing Description: CUMIPMT” on page 274

CUMPRINC
Computes the cumulative principal that is paid on a loan between two periods.
FINANCE('CUMPRINC', rate, nper, pv, start-period, end-period, <type>);

Arguments
rate
  specifies the interest rate.
\texttt{\textit{nper}}
\begin{itemize}
  \item specifies the total number of payment periods.
\end{itemize}

\texttt{\textit{pv}}
\begin{itemize}
  \item specifies the present value or the lump-sum amount that a series of future payments is worth currently.
\end{itemize}

\texttt{\textit{start-period}}
\begin{itemize}
  \item specifies the first period in the calculation. Payment periods are numbered beginning with 1.
\end{itemize}

\texttt{\textit{end-period}}
\begin{itemize}
  \item specifies the last period in the calculation.
\end{itemize}

\texttt{\textit{type}}
\begin{itemize}
  \item specifies the number 0 or 1 and indicates when payments are due. If \texttt{type} is omitted, it is assumed to be 0.
  \item If payments are due at the end of the period, then either omit the \texttt{type} argument or set it to 0. If payments are due at the beginning of the period, then set \texttt{type} to 1.
\end{itemize}

Featured in:
“Example 12: Computing Description: CUMPRINC” on page 274

\textbf{DB}
Computes the depreciation of an asset for a specified period by using the fixed-declining balance method.

\texttt{\textit{FINANCE('DB', cost, salvage, life, period, <month>)}};

\textbf{Arguments}
\begin{itemize}
  \item \texttt{\textit{cost}}
  \begin{itemize}
    \item specifies the initial cost of the asset.
  \end{itemize}
  \item \texttt{\textit{salvage}}
  \begin{itemize}
    \item specifies the value at the end of the depreciation (also called the salvage value of the asset).
  \end{itemize}
  \item \texttt{\textit{life}}
  \begin{itemize}
    \item specifies the number of periods over which the asset is depreciated (also called the useful life of the asset).
  \end{itemize}
  \item \texttt{\textit{period}}
  \begin{itemize}
    \item specifies the period for which you want to calculate the depreciation. \textit{Period} must use the same time units as \textit{life}.
  \end{itemize}
  \item \texttt{\textit{month}}
  \begin{itemize}
    \item specifies the number of months (month is an optional numeric argument). If month is omitted, it defaults to a value of 12.
  \end{itemize}
\end{itemize}

Featured in:
“Example 13: Computing Description: DB” on page 274

\textbf{DDB}
Computes the depreciation of an asset for a specified period by using the double-declining balance method or some other method that you specify.

\texttt{\textit{FINANCE('DDB', cost, salvage, life, period, <factor>)}};

\textbf{Arguments}
\begin{itemize}
  \item 
\end{itemize}
cost
  specifies the initial cost of the asset.

salvage
  specifies the value at the end of the depreciation (also called the salvage value of the asset).

life
  specifies the number of periods over which the asset is depreciated (also called the useful life of the asset).

period
  specifies the period for which you want to calculate the depreciation. Period must use the same time units as life.

factor
  specifies the rate at which the balance declines. If factor is omitted, it is assumed to be 2 (the double-declining balance method).

Featured in:
  “Example 14: Computing Description: DDB” on page 274

DISC
Computes the discount rate for a security.

FINANCE('DISC', settlement, maturity, price, redemption, <basis>);

Arguments

settlement
  specifies the settlement date.

maturity
  specifies the maturity date.

price
  specifies the price of security per $100 face value.

redemption
  specifies the amount to be received at maturity.

basis
  specifies the optional day count value.

Featured in:
  “Example 15: Computing Description: DISC” on page 275

DOLLARDE
Converts a dollar price, expressed as a fraction, to a dollar price, expressed as a decimal number.

FINANCE('DOLLARDE', fractional-dollar, fraction);

Arguments

fractional-dollar
  specifies the number expressed as a fraction.

fraction
  specifies the integer to use in the denominator of a fraction.

Featured in:
  “Example 16: Computing Description: DOLLARDE” on page 275
**DOLLARFR**
Converts a dollar price, expressed as a decimal number, to a dollar price, expressed as a fraction.

FINANCE('DOLLARFR', decimal-dollar, fraction);

Arguments

*decimal-dollar*
- specifies a decimal number.

*fraction*
- specifies the integer to use in the denominator of a fraction.

Featured in:
- “Example 17: Computing Description: DOLLARFR” on page 275

**DURATION**
Computes the annual duration of a security with periodic interest payments.

FINANCE('DURATION', settlement, maturity, coupon, yield, frequency, <basis>);

Arguments

*settlement*
- specifies the settlement date.

*maturity*
- specifies the maturity date.

*coupon*
- specifies the annual coupon rate of the security.

*yield*
- specifies the annual yield of the security.

*frequency*
- specifies the number of coupon payments per year. For annual payments, *frequency*=1; for semiannual payments, *frequency*=2; for quarterly payments, *frequency*=4.

*<basis>*
- specifies the optional day count value.

Featured in:
- “Example 18: Computing Description: DURATION” on page 275

**EFFECT**
Computes the effective annual interest rate.

FINANCE('EFFECT', nominal-rate, npery);

Arguments

*nominal-rate*
- specifies the nominal interest rate.

*npery*
- specifies the number of compounding periods per year.

Featured in:
- “Example 19: Computing Description: EFFECT” on page 276
**FV**
Computes the future value of an investment.

\[
\text{FINANCE}('FV', \text{rate}, \text{nper}, <\text{payment}>, <\text{present-value}>, <\text{type}>);
\]

**Arguments**

rate
specifies the interest rate.

nper
specifies the total number of payment periods.

payment
specifies the payment that is made each period; the payment cannot change over the life of the annuity. Typically, payment contains principal and interest but no fees and taxes. If payment is omitted, you must include the present-value argument.

present-value
specifies the present value or the lump-sum amount that a series of future payments is worth currently. If present-value is omitted, it is assumed to be 0 (zero), and you must include the payment argument.

type
specifies the number 0 or 1 and indicates when payments are due. If type is omitted, it is assumed to be 0.

If payments are due at the end of the period, then either omit the type argument or set it to 0. If payments are due at the beginning of the period, then set type to 1.

Featured in:
“Example 20: Computing Description: FV” on page 276

**FVSCHEDULE**
Computes the future value of the initial principal after applying a series of compound interest rates.

\[
\text{FINANCE}('FVSCHEDULE', \text{principal}, \text{schedule-1}, \text{schedule-2} ...);
\]

**Arguments**

principal
specifies the present value.

schedule
specifies the sequence of interest rates to apply.

Featured in:
“Example 21: Computing Description: FVSCHEDULE” on page 276

**INTRATE**
Computes the interest rate for a fully invested security.

\[
\text{FINANCE}('INTRATE', \text{settlement}, \text{maturity}, \text{investment}, \text{redemption}, <\text{basis}>);
\]

**Arguments**

settlement
specifies the settlement date.

maturity
specifies the maturity date.
investment
   specifies the amount that is invested in the security.

redemption
   specifies the amount to be received at maturity.

basis
   specifies the optional day count value.

Featured in:
   “Example 22: Computing Description: INTRATE” on page 276

**IPMT**
Computes the interest payment for an investment for a specified period.

**FINANCE**(IPMT, rate, period, nper, pv, <fv>, <type>);

**Arguments**

*rate*
   specifies the interest rate.

*period*
   specifies the period for which you want to calculate the depreciation. Period must use the same units as life.

*nper*
   specifies the total number of payment periods.

*pv*
   specifies the present value or the lump-sum amount that a series of future payments is worth currently. If pv is omitted, it is assumed to be 0 (zero), and you must include the fv argument.

*fv*
   specifies the future value or a cash balance that you want to attain after the last payment is made. If fv is omitted, it is assumed to be 0 (for example, the future value of a loan is 0).

*type*
   specifies the number 0 or 1 and indicates when payments are due. If type is omitted, it is assumed to be 0.

   If payments are due at the end of the period, then either omit the type argument or set it to 0. If payments are due at the beginning of the period, then set type to 1.

Featured in:
   “Example 23: Computing Description: IPMT” on page 277

**IRR**
Computes the internal rate of return for a series of cash flows.

**FINANCE**(IRR, value-1, value-2, ..., value-n);

**Arguments**

*value*
   specifies a list of numeric arguments that contain numbers for which you want to calculate the internal rate of return.

Featured in:
   “Example 24: Computing Description: IRR” on page 277
**ISPMT**
Calculates the interest paid during a specific period of an investment.

**FINANCE** (‘ISPMT’, interest-rate, period, number-payments, pv);

**Arguments**

- **interest-rate**
  - is the interest rate for the investment.

- **period**
  - is the period to calculate the interest rate. *Period* must be a value between 1 and *number-payments*.

- **number-payments**
  - is the number of payments for the annuity.

- **pv**
  - is the loan amount or present value of the payments.

Featured in:
  “Example 25: Computing Description: ISPMT” on page 277

**MDURATION**
Computes the Macaulay modified duration for a security with an assumed face value of $100.

**FINANCE**('MDURATION', settlement, maturity, coupon, yield, frequency, <basis>);

**Arguments**

- **settlement**
  - specifies the settlement date.

- **maturity**
  - specifies the maturity date.

- **coupon**
  - specifies the annual coupon rate of the security.

- **yield**
  - specifies the annual yield of the security.

- **frequency**
  - specifies the number of coupon payments per year. For annual payments, *frequency*=1; for semiannual payments, *frequency*=2; for quarterly payments, *frequency*=4.

- **basis**
  - specifies the optional day count value.

Featured in:
  “Example 26: Computing Description: MDURATION” on page 277

**MIRR**
Computes the internal rate of return where positive and negative cash flows are financed at different rates.

**FINANCE**('MIRR', value-1, ..., value-n, finance-rate, reinvest-rate);

**Arguments**
value
specifies a list of numeric arguments that contain numbers. These numbers represent a series of payments (negative values) and income (positive values) that occur at regular periods. Value must contain at least one positive value and one negative value to calculate the modified internal rate of return.

finance-rate
specifies the interest rate that you pay on the money that is used in the cash flows.

reinvest-rate
specifies the interest rate that you receive on the cash flows as you reinvest them.

Featured in:
“Example 27: Computing Description: MIRR” on page 278

NOMINAL
Computes the annual nominal interest rates.

```
FINANCE('NOMINAL', effective-rate, npery);
```

Arguments

effective-rate
specifies the effective interest rate.

npery
specifies the number of compounding periods per year.

Featured in:
“Example 28: Computing Description: NOMINAL” on page 278

NPER
Computes the number of periods for an investment.

```
FINANCE('NPER', rate, payment, pv, <fv> , <type>);
```

Arguments

rate
specifies the interest rate.

payment
specifies the payment that is made each period; the payment cannot change over the life of the annuity. Typically, payment contains principal and interest but no other fees or taxes. If payment is omitted, you must include the pv argument.

pv
specifies the present value or the lump-sum amount that a series of future payments is worth currently. If pv is omitted, it is assumed to be 0 (zero), and you must include the payment argument.

fv
specifies the future value or a cash balance that you want to attain after the last payment is made. If fv is omitted, it is assumed to be 0 (for example, the future value of a loan is 0).

type
specifies the number 0 or 1 and indicates when payments are due. If type is omitted, it is assumed to be 0.

If payments are due at the end of the period, then either omit the type argument or set it to 0. If payments are due at the beginning of the period, then set type to 1.
NPV
Computes the net present value of an investment based on a series of periodic cash flows and a discount rate.

FINANCE('NPV', rate, value-1 <, ..., value-n>);

Arguments
rate
specifies the interest rate.

value
represents the sequence of the cash flows.

ODDFPRICE
Computes the price of a security per $100 face value with an odd first period.

FINANCE('ODDFPRICE', settlement, maturity, issue, first-coupon, rate, yield, redemption, frequency, <basis>);

Arguments
settlement
specifies the settlement date.

maturity
specifies the maturity date.

issue
specifies the issue date of the security.

first-coupon
specifies the first coupon date of the security.

rate
specifies the interest rate.

yield
specifies the annual yield of the security.

redemption
specifies the amount to be received at maturity.

frequency
specifies the number of coupon payments per year. For annual payments, frequency=1; for semiannual payments, frequency=2; for quarterly payments, frequency=4.

basis
specifies the optional day count value.

ODDFYIELD
Computes the yield of a security with an odd first period.
FINANCE('ODDFYIELD', settlement, maturity, issue, first-coupon, rate, price, redemption, frequency, <basis>);

Arguments
settlement
    specifies the settlement date.
maturity
    specifies the maturity date.
issue
    specifies the issue date of the security.
first-coupon
    specifies the first coupon date of the security.
rate
    specifies the interest rate.
price
    specifies the price of the security per $100 face value.
redemption
    specifies the amount to be received at maturity.
frequency
    specifies the number of coupon payments per year. For annual payments, frequency=1; for semiannual payments, frequency=2; for quarterly payments, frequency=4.
basis
    specifies the optional day count value.

Featured in:
    “Example 32: Computing Description: ODDFYIELD” on page 279

ODDLPRICE
Computes the price of a security per $100 face value with an odd last period.
FINANCE('ODDLPRICE', settlement, maturity, last-interest, rate, yield, redemption, frequency, <basis>);

Arguments
settlement
    specifies the settlement date.
maturity
    specifies the maturity date.
last-interest
    specifies the last coupon date of the security.
rate
    specifies the interest rate.
yield
    specifies the annual yield of the security.
redemption
    specifies the amount to be received at maturity.
**frequency**

specifies the number of coupon payments per year. For annual payments, 
\( frequency=1 \); for semiannual payments, \( frequency=2 \); for quarterly payments,  
\( frequency=4 \).

**basis**

specifies the optional day count value.

Featured in:

“Example 33: Computing Description: ODDLPRICE” on page 280

**ODDLYIELD**

Computes the yield of a security with an odd last period.

\[ \text{FINANCE('ODDLYIELD', settlement, maturity, last-interest, rate, price, redemption, frequency, <basis>}); } \]

**Arguments**

**settlement**

specifies the settlement date.

**maturity**

specifies the maturity date.

**last-interest**

specifies the last coupon date of the security.

**rate**

specifies the interest rate.

**price**

specifies the price of the security per $100 face value.

**redemption**

specifies the amount to be received at maturity.

**frequency**

specifies the number of coupon payments per year. For annual payments,  
\( frequency=1 \); for semiannual payments, \( frequency=2 \); for quarterly payments,  
\( frequency=4 \).

**basis**

specifies the optional day count value.

Featured in:

“Example 34: Computing Description: ODDLYLEID” on page 280

**PMT**

Computes the periodic payment of an annuity.

\[ \text{FINANCE('PMT', rate, nper, pv, <fv>, <type>}); } \]

**Arguments**

**rate**

specifies the interest rate.

**nper**

specifies the number of payment periods.
\( pv \)
specifies the present value or the lump-sum amount that a series of future payments is worth currently. If \( pv \) is omitted, it is assumed to be 0 (zero), and you must include the \( fv \) argument.

\( fv \)
specifies the future value or a cash balance that you want to attain after the last payment is made. If \( fv \) is omitted, it is assumed to be 0 (for example, the future value of a loan is 0).

\( type \)
specifies the number 0 or 1 and indicates when payments are due. If \( type \) is omitted, it is assumed to be 0.

If payments are due at the end of the period, then either omit the \( type \) argument or set it to 0. If payments are due at the beginning of the period, then set \( type \) to 1.

Featured in:
“Example 35: Computing Description: PMT” on page 280

**PPMT**
Computes the payment on the principal for an investment for a specified period.

\[
\text{FINANCE('PPMT', rate, period, nper, pv, <fv>, <type>)};
\]

**Arguments**

\( rate \)
specifies the interest rate.

\( period \)
specifies the period.

Range: 1–\( nper \)

\( nper \)
specifies the number of payment periods.

\( pv \)
specifies the present value or the lump-sum amount that a series of future payments is worth currently. If \( pv \) is omitted, it is assumed to be 0 (zero), and you must include the \( fv \) argument.

\( fv \)
specifies the future value or a cash balance that you want to attain after the last payment is made. If \( fv \) is omitted, it is assumed to be 0 (for example, the future value of a loan is 0).

\( type \)
specifies the number 0 or 1 and indicates when payments are due. If \( type \) is omitted, it is assumed to be 0.

If payments are due at the end of the period, then either omit the \( type \) argument or set it to 0. If payments are due at the beginning of the period, then set \( type \) to 1.

Featured in:
“Example 36: Computing Description: PPMT” on page 280

**PRICE**
Computes the price of a security per $100 face value that pays periodic interest.

\[
\text{FINANCE('PRICE', settlement, maturity, rate, yield, redemption, frequency, <basis>)};
\]
Arguments

settlement
 specifies the settlement date.

maturity
 specifies the maturity date.

rate
 specifies the interest rate.

yield
 specifies the annual yield of the security.

redemption
 specifies the amount to be received at maturity.

currency
 specifies the number of coupon payments per year. For annual payments, currency=1; for semiannual payments, currency=2; for quarterly payments, currency=4.

basis
 specifies the optional day count value.

Featured in:

“Example 37: Computing Description: PRICE” on page 281

PRICEDISC
Computes the price of a discounted security per $100 face value.

FINANCE('PRICEDISC', settlement, maturity, discount, redemption, <basis>);

Arguments

settlement
 specifies the settlement date.

maturity
 specifies the maturity date.

discount
 specifies the discount rate of the security.

redemption
 specifies the amount to be received at maturity.

basis
 specifies the optional day count value.

Featured in:

“Example 38: Computing Description: PRICEDISC” on page 281

PRICEMAT
Computes the price of a security per $100 face value that pays interest at maturity.

FINANCE('PRICEMAT', settlement, maturity, issue, rate, yield, <basis>);

Arguments

settlement
 specifies the settlement date.

maturity
 specifies the maturity date.
issue
  specifies the issue date of the security.

rate
  specifies the interest rate.

yield
  specifies the annual yield of the security.

basis
  specifies the optional day count value.

Featured in:
  “Example 39: Computing Description: PRICEMAT” on page 281

PV
Computes the present value of an investment.

\[
\text{FINANCE('PV', rate, nper, payment, <fv>, <type>);}
\]

Arguments
rate
  specifies the interest rate.

nper
  specifies the total number of payment periods.

payment
  specifies the payment that is made each period; the payment cannot change over the life of the annuity. Typically, payment contains principal and interest but no other fees or taxes.

fv
  specifies the future value or a cash balance that you want to attain after the last payment is made. If fv is omitted, it is assumed to be 0 (for example, the future value of a loan is 0).

type
  specifies the number 0 or 1 and indicates when payments are due. If type is omitted, it is assumed to be 0.

If payments are due at the end of the period, then either omit the type argument or set it to 0. If payments are due at the beginning of the period, then set type to 1.

Featured in:
  “Example 40: Computing Description: PV” on page 282

RATE
Computes the interest rate per period of an annuity.

\[
\text{FINANCE('RATE', nper, payment, pv, <fv>, <type>);}
\]

Arguments
nper
  specifies the total number of payment periods.

payment
  specifies the payment that is made each period; the payment cannot change over the life of the annuity. Typically, payment contains principal and interest but no other fees or taxes. If payment is omitted, you must include the pv argument.
pv specifies the present value or the lump-sum amount that a series of future payments is worth currently. If pv is omitted, it is assumed to be 0 (zero), and you must include the fv argument.

fv specifies the future value or a cash balance that you want to attain after the last payment is made. If fv is omitted, it is assumed to be 0 (for example, the future value of a loan is 0).

type specifies the number 0 or 1 and indicates when payments are due. If type is omitted, it is assumed to be 0.

If payments are due at the end of the period, then either omit the type argument or set it to 0. If payments are due at the beginning of the period, then set type to 1.

Featured in:
“Example 41: Computing Description: RATE” on page 282

RECEIVED
Computes the amount that is received at maturity for a fully invested security.

FINANCE('RECEIVED', settlement, maturity, investment, discount, <basis>);

Arguments
settlement specifies the settlement date.
maturity specifies the maturity date.
investment specifies the amount that is invested in the security.
discount specifies the discount rate of the security.
basis specifies the optional day count value.

Featured in:
“Example 42: Computing Description: RECEIVED” on page 282

SLN
Computes the straight-line depreciation of an asset for one period.

FINANCE('SLN', cost, salvage, life);

Arguments
cost specifies the initial cost of the asset.
salvage specifies the value at the end of the depreciation (also called the salvage value of an asset).
life specifies the number of periods over which the asset is depreciated (also called the useful life of the asset).
SYD
Computes the sum-of-years digits depreciation of an asset for a specified period.

\[ \text{FINANCE('SYD', cost, salvage, life, period);} \]

**Arguments**
- **cost** specifies the initial cost of the asset.
- **salvage** specifies the value at the end of the depreciation (also called the salvage value of the asset).
- **life** specifies the number of periods over which the asset is depreciated (also called the useful life of the asset).
- **period** specifies a period in the same time units that are used for the argument **life**.

TBILLEQ
Computes the bond-equivalent yield for a treasury bill.

\[ \text{FINANCE('TBILLEQ', settlement, maturity, discount);} \]

**Arguments**
- **settlement** specifies the settlement date.
- **maturity** specifies the maturity date.
- **discount** specifies the discount rate of the security.

TBILLPRICE
Computes the price of a treasury bill per $100 face value.

\[ \text{FINANCE('TBILLPRICE', settlement, maturity, discount);} \]

**Arguments**
- **settlement** specifies the settlement date.
- **maturity** specifies the maturity date.
- **discount** specifies the discount rate of the security.
TBILLYIELD
Computes the yield for a treasury bill.

FINANCE('TBILLYIELD', settlement, maturity, price);

Arguments
settlement
specifies the settlement date.
maturity
specifies the maturity date.
price
specifies the price of the security per $100 face value.

Featured in:
“Example 47: Computing Description: TBILLYIELD” on page 283

VDB
Computes the depreciation of an asset for a specified or partial period by using a
declining balance method.

FINANCE('VDB', cost, salvage, life, start-period, end-period, <factor>, <noswitch>);

Arguments
cost
specifies the initial cost of the asset.
salvage
specifies the value at the end of the depreciation (also called the salvage value of the
asset).
life
specifies the number of periods over which the asset is depreciated (also called the
useful life of the asset).
start-period
specifies the first period in the calculation. Payment periods are numbered beginning
with 1.
end-period
specifies the last period in the calculation.
factor
specifies the rate at which the balance declines. If factor is omitted, it is assumed to
be 2 (the double-declining balance method).
noswitch
specifies a logical value that determines whether to switch to straight-line
depreciation when the depreciation is greater than the declining balance calculation.
If noswitch is omitted, it is assumed to be 1.

Featured in:
“Example 48: Computing Description: VDB” on page 284

XIRR
Computes the internal rate of return for a schedule of cash flows that is not necessarily
periodic.

FINANCE('XIRR', values, dates, <guess>);

Arguments
values
  specifies a series of cash flows that corresponds to a schedule of payments in dates. The first payment is optional and corresponds to a cost or payment that occurs at the beginning of the investment. If the first value is a cost or payment, it must be a negative value. All succeeding payments are discounted based on a 365-day year. The series of values must contain at least one positive value and one negative value.

dates
  specifies a schedule of payment dates that corresponds to the cash flow payments. The first payment date indicates the beginning of the schedule of payments. All other dates must be later than this date, but they can occur in any order.

guess
  specifies an optional number that you guess is close to the result of XIRR.

Featured in:
  “Example 49: Computing Description: XIRR” on page 284

**XNPV**
Computes the net present value for a schedule of cash flows that is not necessarily periodic.

\[ \text{FINANCE('XNPV', rate, values, dates);} \]

**Arguments**

*rate*
  specifies the interest rate.

*values*
  specifies a series of cash flows that corresponds to a schedule of payments in dates. The first payment is optional and corresponds to a cost or payment that occurs at the beginning of the investment. If the first value is a cost or payment, it must be a negative value. All succeeding payments are discounted based on a 365-day year. The series of values must contain at least one positive value and one negative value.

*dates*
  specifies a schedule of payment dates that corresponds to the cash flow payments. The first payment date indicates the beginning of the schedule of payments. All other dates must be later than this date, but they can occur in any order.

Featured in:
  “Example 50: Computing Description: XNPV” on page 284

**YIELD**
Computes the yield on a security that pays periodic interest.

\[ \text{FINANCE('YIELD', settlement, maturity, rate, price, redemption, frequency, <basis>);} \]

**Arguments**

*settlement*
  specifies the settlement date.

*maturity*
  specifies the maturity date.

*rate*
  specifies the interest rate.

*price*
  specifies the price of the security per $100 face value.
redemption
    specifies the amount to be received at maturity.

frequency
    specifies the number of coupon payments per year. For annual payments,
    frequency=1; for semiannual payments, frequency=2; for quarterly payments,
    frequency=4.

basis
    specifies the optional day count value.

Featured in:
    “Example 51: Computing Description: YIELD” on page 284

**YIELDDISC**
Computes the annual yield for a discounted security (for example, a treasury bill).

```
FINANCE('YIELDDISC', settlement, maturity, rate, price, redemption, <basis>);
```

**Arguments**

- **settlement**
  specifies the settlement date.

- **maturity**
  specifies the maturity date.

- **rate**
  specifies the interest rate.

- **price**
  specifies the price of the security per $100 face value.

- **redemption**
  specifies the amount to be received at maturity.

- **basis**
  specifies the optional day count value.

Featured in:
    “Example 52: Computing Description: YIELDDISC” on page 285

**YIELDMAT**
Computes the annual yield of a security that pays interest at maturity.

```
FINANCE('YIELDMAT', settlement, maturity, issue, rate, price, <basis>);
```

**Arguments**

- **settlement**
  specifies the settlement date.

- **maturity**
  specifies the maturity date.

- **issue**
  specifies the issue date of the security.

- **rate**
  specifies the interest rate.

- **price**
  specifies the price of the security per $100 face value.
basis

specifies the optional day count value.

Featured in:
“Example 53: Computing Description: YIELDMAT” on page 285

Examples

Example 1: Computing Accrued Interest: ACCRINT
The following example computes the accrued interest for a security that pays periodic interest.

```plaintext
data _null_;  
   issue=mdy(2, 27, 1996);  
   firstinterest=mdy(8, 31, 1998);  
   settlement=mdy(5, 1, 1998);  
   rate=0.1;  
   par=1000;  
   frequency=2;  
   basis=1;  
   r=finance('accrint', issue, firstinterest, 
               settlement, rate, par, frequency, basis);  
   put r=;  
run;
```

The value of \( r \) that is returned is 217.39728.

Example 2: Computing Accrued Interest: ACCRINTM
The following example computes the accrued interest for a security that pays interest at maturity.

```plaintext
data _null_;  
   issue=mdy(2, 28, 1998);  
   maturity=mdy(8, 31, 1998);  
   rate=0.1;  
   par=1000;  
   basis=0;  
   r=finance('accrintm', issue, maturity, rate, par, basis);  
   put r=;  
run;
```

The value of \( r \) that is returned is 50.55555556.

Example 3: Computing Depreciation: AMORDEGRC
The following example computes the depreciation for each accounting period by using a depreciation coefficient.

```plaintext
data _null_;  
   cost=2400;  
   datepurchased=mdy(8, 19, 2008);  
   firstperiod=mdy(12, 31, 2008);  
   salvage=300;  
   period=1;  
   rate=0.15;  
   basis=1;
```
r=finance('amordegrc', cost, datepurchased, 
    firstperiod, salvage, period, rate, basis);
put r=;
run;
The value of r that is returned is 776.

**Example 4: Computing Description: AMORLINC**
The following example computes the depreciation for each accounting period.

data _null_; 
cost=2400;
    dp=mdy(9, 30, 1998);
    fp=mdy(12, 31, 1998);
    salvage=245;
    period=0;
    rate=0.115;
    basis=0;
    r=finance('amorlinc', cost, dp, fp, salvage, period, rate, basis);
    put r=;
run;
The value of r that is returned is 69.

**Example 5: Computing Description: COUPDAYBS**
The following example computes the number of days from the beginning of the coupon period to the settlement date.

data _null_; 
    settlement=mdy(12,30,1994);
    maturity=mdy(11,29,1997);
    frequency=4;
    basis=2;
    r=finance('coupdaybs', settlement, maturity, frequency, basis);
    put r=;
run;
The value of r that is returned is 31.

**Example 6: Computing Description: COUPDAYS**
The following example computes the number of days in the coupon period that contains the settlement date.

data _null_; 
    settlement=mdy(1,25,2007);
    maturity=mdy(11,15,2008);
    frequency=2;
    basis=1;
    r=finance('coupdays', settlement, maturity, frequency, basis);
    put r=;
run;
The value of r that is returned is 181.

**Example 7: Computing Description: COUPDAYSNC**
The following example computes the number of days from the settlement date to the next coupon date.
data _null_;  
settlement=mdy(1,25,2007);  
maturity=mdy(11,15,2008);  
frequency=2;  
basis=1;  
r=finance('coupdaysnc', settlement, maturity, frequency, basis);  
put r=;  
run;

The value of r that is returned is 110.

**Example 8: Computing Description: COUPNCD**

The following example computes the next coupon date after the settlement date.

data _null_;  
settlement=mdy(1, 25, 2007);  
maturity=mdy(11, 15, 2008);  
frequency=2;  
basis=1;  
r=finance('coupncd', settlement, maturity, frequency, basis);  
put r=date7.;  
run;

The value of r that is returned is 15MAY07.

*Note: r is a numeric SAS value and can be printed using the DATE7 format.*

**Example 9: Computing Description: COUPNUM**

The following example computes the number of coupons that are payable between the settlement date and the maturity date.

data _null_;  
settlement=mdy(1,25,2007);  
maturity=mdy(11,15,2008);  
frequency=2;  
basis=1;  
r=finance('coupnum', settlement, maturity, frequency, basis);  
put r=;  
run;

The value of r that is returned is 4.

**Example 10: Computing Description: COUPPCD**

The following example computes the previous coupon date before the settlement date.

data _null_;  
settlement=mdy(1, 25, 2007);  
maturity=mdy(11, 15, 2008);  
frequency=2;  
basis=1;  
r=finance('couppcd', settlement, maturity, frequency, basis);  
put settlement;  
put maturity;  
put r date7.;  
run;

The value of r that is returned is 11/15/2006.
**Example 11: Computing Description: CUMIPMT**
The following example computes the cumulative interest that is paid between two periods.

```sas
data _null_;  
  rate=0.09;  
  nper=30;  
  pv=125000;  
  startperiod=13;  
  endperiod=24;  
  type=0;  
  r=finance('cumipmt', rate, nper, pv, startperiod, endperiod, type);  
  put r=;  
run;
```

The value of `r` that is returned is −94054.82033.

**Example 12: Computing Description: CUMPRINC**
The following example computes the cumulative principal that is paid on a loan between two periods.

```sas
data _null_;  
  rate=0.09;  
  nper=30;  
  pv=125000;  
  startperiod=13;  
  endperiod=24;  
  type=0;  
  r=finance('cumprinc', rate, nper, pv, startperiod, endperiod, type);  
  put r=;  
run;
```

The value of `r` that is returned is −51949.70676.

**Example 13: Computing Description: DB**
The following example computes the depreciation of an asset for a specified period by using the fixed-declining balance method.

```sas
data _null_;  
  cost=1000000;  
  salvage=100000;  
  life=6;  
  period=2;  
  month=7;  
  r=finance('db', cost, salvage, life, period, month);  
  put r=;  
run;
```

The value of `r` that is returned is 259639.41667.

**Example 14: Computing Description: DDB**
The following example computes the depreciation of an asset for a specified period by using the double-declining balance method or some other method that you specify.

```sas
data _null_;  
  cost=2400;  
  salvage=300;  
  life=10*365;
```
period=1;
factors=.;
r=finance('ddb', cost, salvage, life, period, factor);
put r=;
run;

The value of \( r \) that is returned is 1.3150684932.

**Example 15: Computing Description: DISC**
The following example computes the discount rate for a security.

```sas
data _null_;    
settlement=mdy(1, 25, 2007);    
maturity=mdy(6, 15, 2007);    
price=97.975;    
redemption=100;    
basis=1;    
r=finance('disc', settlement, maturity, price, redemption, basis);    
put r=;    
run;
```

The value of \( r \) that is returned is 0.052420213.

**Example 16: Computing Description: DOLLARDE**
The following example converts a dollar price, expressed as a fraction, to a dollar price, expressed as a decimal number.

```sas
data _null_;    
fractionaldollar=1.125;    
fraction=16;    
r=finance('dollarde', fractionaldollar, fraction);    
put r=;    
run;
```

The value of \( r \) that is returned is 1.78125.

**Example 17: Computing Description: DOLLARFR**
The following example converts a dollar price, expressed as a decimal number, to a dollar price, expressed as a fraction.

```sas
data _null_;    
decimaldollar=1.125;    
fraction=16;    
r=finance('dollarfr', decimaldollar, fraction);    
put r=;    
run;
```

The value of \( r \) that is returned is 1.02. In fraction form, the value of \( r \) is read as \( 1 \frac{2}{16} \).

**Example 18: Computing Description: DURATION**
The following example computes the annual duration of a security with periodic interest payments.

```sas
data _null_;    
settlement=mdy(1, 1, 2008);    
maturity=mdy(1, 1, 2016);    
couponrate=0.08;
```
yield=0.09;
frequency=2;
basis=1;
r=finance('duration', settlement, maturity, couponrate, yield, frequency, basis);
   put r=;
run;

The value of $r$ that is returned is 5.993775.

**Example 19: Computing Description: EFFECT**
The following example computes the effective annual interest rate.

data _null_;  
nominalrate=0.0525;
npery=4;
r=finance('effect', nominalrate, npery);
   put r=;
run;

The value of $r$ that is returned is 0.053543.

**Example 20: Computing Description: FV**
The following example computes the future value of an investment.

data _null_;  
rate=0.06/12;
nper=10;
payment=−200;
present-value=−500;
type=1;
r=finance('fv', rate, nper, payment, present-value, type);
   put r=;
run;

The value of $r$ that is returned is 2581.4033741.

**Example 21: Computing Description: FVSCHEDULE**
The following example computes the future value of the initial principal after applying a series of compound interest rates.

data _null_;  
principal=1;
r1=0.09;
r2=0.11;
r3=0.1;
r=finance('fvschedule', principal, r1, r2, r3);
   put r=;
run;

The value of $r$ that is returned is 1.33089.

**Example 22: Computing Description: INTRATE**
The following example computes the interest rate for a fully invested security.

data _null_;  
settlement=mdy(2, 15, 2008);
maturity=mdy(5, 15, 2008);
investment=1000000;
redemption=1014420;
basis=2;
r=finance('intrate', settlement, maturity, investment, redemption, basis);
put r=;
run;
The value of r that is returned is 0.05768

Example 23: Computing Description: IPMT
The following example computes the interest payment for an investment for a specified period.

data _null_;  
  rate=0.1/12;  
  per=2;  
  nper=3;  
  pv=100;  
  fv=.;  
  type=.;  
  r=finance('ipmt', rate, per, nper, pv, fv, type);
  put r=;
run;
The value of r that is returned is −0.557857564.

Example 24: Computing Description: IRR
The following example computes the internal rate of return for a series of cash flows.

data _null_;  
  v1=-70000;  
  v2=12000;  
  v3=15000;  
  v4=18000;  
  v5=21000;  
  v6=26000;  
  r=finance('irr', v1, v2, v3, v4, v5, v6);
  put r=;
run;
The value of r that is returned is 0.086630948.

Example 25: Computing Description: ISPMT
The following example computes the interest payment for a $5,000 investment that earns 7.5% annually for two years. The interest payment is calculated for the 8th month.

data ispmt;
  interest=finance('ispmt', 0.075/12, 8, 2*12, 5000);
  put interest=;
run;
The value that is returned is −20.83333333.

Example 26: Computing Description: MDURATION
The following example computes the Macaulay modified duration for a security with an assumed face value of $100.

data _null_;
settlement=mdy(1, 1, 2008);
maturity=mdy(1, 1, 2016);
couponrate=0.08;
yield=0.09;
frequency=2;
basis=1;
r=finance('mduration', settlement, maturity, couponrate, yield, frequency, basis);
put r=;
run;

The value of r that is returned is 5.7356698139.

**Example 27: Computing Description: MIRR**
The following example computes the internal rate of return where positive and negative cash flows are financed at different rates.

data _null_;  
v1=-1000;  
v2=3000;  
v3=4000;  
v4=5000;  
financerate=0.08;  
reinvestrate=0.10;  
r=finance('mirr', v1, v2, v3, v4, financerate, reinvestrate);
put r=;
run;

The value of r that is returned is 1.3531420172.

**Example 28: Computing Description: NOMINAL**
The following example computes the annual nominal interest rate.

data _null_;  
effectrate=0.08;  
npery=4;  
r= finance('nominal', effectrate, npery);
put r=;
run;

The value of r that is returned is 0.0777061876.

**Example 29: Computing Description: NPER**
The following example computes the number of periods for an investment.

data _null_;  
rate=0.08;  
payment=200;  
fv=0;  
type=0;  
r=finance('nper', rate, payment, pv, fv, type);
put r=;
run;

The value of r that is returned is −4.371981351.
**Example 30: Computing Description: NPV**
The following example computes the net present value of an investment based on a series of periodic cash flows and a discount rate.

```plaintext
data _null_;  
  rate=0.08;  
  v1=200;  
  v2=1000;  
  v3=0.;  
  r=finance('npv', rate, v1, v2, v3);  
  put r=;  
run;
```

The value of `r` that is returned is 1042.5240055.

**Example 31: Computing Description: ODDFPRICE**
The following example computes the price of a security per $100 face value with an odd first period.

```plaintext
data _null_;  
  settlement=mdy(1, 15, 93);  
  maturity=mdy(1, 1, 98);  
  issue=mdy(1, 1, 93);  
  firstcoupon=mdy(7, 1, 94);  
  rate=0.07;  
  yield=0.06;  
  redemption=100;  
  frequency=2;  
  basis=0;  
  r=finance('oddfprice', settlement, maturity, issue, firstcoupon,  
    rate, yield, redemption, frequency, basis);  
  put r=;  
run;
```

The value of `r` that is returned is 103.94103984.

**Example 32: Computing Description: ODDFYIELD**
The following example computes the interest of a yield with an odd first period.

```plaintext
data _null_;  
  settlement=mdy(1, 15, 93);  
  maturity=mdy(1, 1, 98);  
  issue=mdy(1, 1, 93);  
  firstcoupon=mdy(7, 1, 94);  
  rate=0.07;  
  price=103.94103984;  
  redemption=100;  
  frequency=2;  
  basis=0;  
  r=finance('oddfyield', settlement, maturity, issue, firstcoupon,  
    rate, price, redemption, frequency, basis);  
  put r=;  
run;
```

The value of `r` that is returned is 0.06.
Example 33: Computing Description: ODDLPRICE
The following example computes the price of a security per $100 face value with an odd last period.

```sas
data _null_
  settlement=mdy(2, 7, 2008);
  maturity=mdy(6, 15, 2008);
  lastinterest=mdy(10, 15, 2007);
  rate=0.0375;
  yield=0.0405;
  redemption=100;
  frequency=2;
  basis=0;
  r=finance('oddlprice', settlement, maturity, lastinterest,
             rate, yield, redemption, frequency, basis);
  put r=;
run;
```

The value of `r` that is returned is 99.878286015.

Example 34: Computing Description: ODDLYIELD
The following example computes the yield of a security with an odd last period.

```sas
data _null_
  settlement=mdy(2, 7, 2008);
  maturity=mdy(6, 15, 2008);
  lastinterest=mdy(10, 15, 2007);
  rate=0.0375;
  price=99.878286015;
  redemption=100;
  frequency=2;
  basis=0;
  r=finance('oddlprice', settlement, maturity, lastinterest,
             rate, yield, redemption, frequency, basis);
  put r=;
run;
```

The value of `r` that is returned is 0.0405.

Example 35: Computing Description: PMT
The following example computes the periodic payment for an annuity.

```sas
data _null_
  rate=0.08;
  nper=5;
  pv=91;
  fv=3;
  type=0;
  r=finance('pmt', rate, nper, pv, fv, type);
  put r=;
run;
```

The value of `r` that is returned is -23.30290673.

Example 36: Computing Description: PPMT
The following example computes the payment on the principal for an investment for a specified period.

```sas
```
data _null_; 
  rate=0.08; 
  period=10; 
  nper=10; 
  pv=200000; 
  fv=0; 
  type=0; 
  r=finance('ppmt', rate, period, nper, pv, fv, type); 
  put r=; 
run;

The value of r that is returned is −27598.05346.

**Example 37: Computing Description: PRICE**
The following example computes the price of a security per $100 face value that pays periodic interest.

data _null_; 
  settlement=mdy(2,15,2008); 
  maturity=mdy(11,15,2017); 
  rate=0.0575; 
  yield=0.065; 
  redemption=100; 
  frequency=2; 
  basis=0; 
  r=finance('price', settlement, maturity, rate, yield, redemption, 
               frequency, basis); 
  put r=; 
run;

The value of r that is returned is 94.634361621.

**Example 38: Computing Description: PRICEDISC**
The following example computes the price of a discounted security per $100 face value.

data _null_; 
  settlement=mdy(2,15,2008); 
  maturity=mdy(11,15,2017); 
  discount=0.0525; 
  redemption=100; 
  basis=0; 
  r=finance('pricedisc', settlement, maturity, discount, redemption, basis); 
  put r=; 
run;

The value of r that is returned is 48.8125.

**Example 39: Computing Description: PRICEMAT**
The following example computes the price of a security per $100 face value that pays interest at maturity.

data _null_; 
  settlement=mdy(2,15,2008); 
  maturity=mdy(4,13,2008); 
  issue=mdy(11,11,2007); 
  rate=0.061; 
  yield=0.061;
basis=0;
    r=finance('pricemat', settlement, maturity, issue, rate, yield, basis);
    put r=;
    run;

The value of $r$ that is returned is 99.98449888.

**Example 40: Computing Description: PV**

The following example computes the present value of an investment.

data _null_;  
        rate=0.05;  
        nper=10;  
        payment=1000;  
        fv=200;  
        type=0;  
        r=finance('pv', rate, nper, payment, fv, type);  
        put r=;  
    run;

The value of $r$ that is returned is $7844.51758$.

**Example 41: Computing Description: RATE**

The following example computes the interest rate per period of an annuity.

data _null_;  
        nper=4;  
        payment=-2481;  
        pv=8000;  
        r=finance('rate', nper, payment, pv);  
        put r=;  
    run;

The value of $r$ that is returned is $0.0921476841$.

**Example 42: Computing Description: RECEIVED**

The following example computes the amount that is received at maturity for a fully invested security.

data _null_;  
        settlement=mdy(2, 15, 2008);  
        maturity=mdy(5, 15, 2008);  
        investment=1000000;  
        discount=0.0575;  
        basis=2;  
        r=finance('received', settlement, maturity, investment, discount, basis);  
        put r=;  
    run;

The value of $r$ that is returned is $1014584.6544$.

**Example 43: Computing Description: SLN**

The following example computes the straight-line depreciation of an asset for one period.

data _null_;  
        cost=2000;  
        salvage=200;
life=11;
r=finance('sln', cost, salvage, life);
put r=;
run;
The value of $r$ that is returned is 163.63636364.

**Example 44: Computing Description: SYD**
The following example computes the sum-of-years digits depreciation of an asset for a specified period.

data _null_;  
cost=2000;  
salvage=200;  
life=11;  
period=1;
r=finance('syd', cost, salvage, life, period);
put r=;
run;
The value of $r$ that is returned is 300.

**Example 45: Computing Description: TBILLEQ**
The following example computes the bond-equivalent yield for a treasury bill.

data _null_;  
settlement=mdy(3, 31, 2008);  
maturity=mdy(6, 1, 2008);  
discount=0.0914;
r=finance('tbilleq', settlement, maturity, discount);
put r=;
run;
The value of $r$ that is returned is 0.0941514936.

**Example 46: Computing Description: TBILLPRICE**
The following example computes the price of a treasury bill per $100 face value.

data _null_;  
settlement=mdy(3, 31, 2008);  
maturity=mdy(6, 1, 2008);  
discount=0.09;
r=finance('tbillprice', settlement, maturity, discount);
put r=;
run;
The value of $r$ that is returned is 98.45.

**Example 47: Computing Description: TBILLYIELD**
The following example computes the yield for a treasury bill.

data _null_;  
settlement=mdy(3, 31, 2008);  
maturity=mdy(6, 1, 2008);  
price=98;
r=finance('tbillyield', settlement, maturity, price);
put r=;
run;
The value of $r$ that is returned is 0.1184990125.

**Example 48: Computing Description: VDB**
The following example computes the depreciation of an asset for a specified or partial period by using a declining balance method.

```sas
data _null_;  
cost=2400;  
salvage=300;  
life=10;  
startperiod=0;  
endperiod=1;  
factor=1.5;  
r=finance('vdb', cost, salvage, life, startperiod, endperiod, factor);  
put r=;  
run;
```

The value of $r$ that is returned is 360.

**Example 49: Computing Description: XIRR**
The following example computes the internal rate of return for a schedule of cash flows that is not necessarily periodic.

```sas
data _null_;  
v1=-10000; d1=mdy(1, 1, 2008);  
v2=2750; d2=mdy(3, 1, 2008);  
v3=4250; d3=mdy(10, 30, 2008);  
v4=3250; d4=mdy(2, 15, 2009);  
v5=2750; d5=mdy(4, 1, 2009);  
r=finance('xirr', v1, v2, v3, v4, v5, d1, d2, d3, d4, d5, 0.1);  
put r=;  
run;
```

The value of $r$ that is returned is 0.3733625335.

**Example 50: Computing Description: XNPV**
The following example computes the net present value for a schedule of cash flows that is not necessarily periodic.

```sas
data _null_;  
r=.09;  
v1=-10000; d1=mdy(1, 1, 2008);  
v2=2750; d2=mdy(3, 1, 2008);  
v3=4250; d3=mdy(10, 30, 2008);  
v4=3250; d4=mdy(2, 15, 2009);  
v5=2750; d5=mdy(4, 1, 2009);  
r=finance('xnpv', r, v1, v2, v3, v4, v5, d1, d2, d3, d4, d5);  
put r=;  
run;
```

The value of $r$ that is returned is 2086.647602.

**Example 51: Computing Description: YIELD**
The following example computes the yield on a security that pays periodic interest.

```sas
data _null_;  
settlement=mdy(2, 15, 2008);  
run;
```
maturity=mdy(11, 15, 2016);
rate=0.0575;
pr=95.04287;
redemption=100;
frequency=2;
basis=0;
r=finance('yield', settlement, maturity, rate, pr, redemption, frequency, basis);
put r=;
run;

The value of \( r \) that is returned is 0.0650000069.

**Example 52: Computing Description: YIELDDISC**
The following example computes the annual yield for a discounted security (for example, a treasury bill).

data _null_
settlement=mdy(2, 15, 2008);
maturity=mdy(11, 15, 2016);
pr=95.04287;
redemption=100;
basis=0;
r=finance('yielddisc', settlement, maturity, pr, redemption, basis);
put r=;
run;

The value of \( r \) that is returned is 0.0059607748.

**Example 53: Computing Description: YIELDMAT**
The following example computes the annual yield of a security that pays interest at maturity.

data _null_
settlement=mdy(3, 15, 2008);
maturity=mdy(11, 3, 2008);
issue=mdy(11, 8, 2007);
rate=0.0625;
pr=100.0123;
basis=0;
r=finance('yieldmat', settlement, maturity, issue, rate, pr, basis);
put r=;
run;

The value of \( r \) that is returned is 0.0609543337.

---

### FIND Function

Finds a specific substring of characters within a character string.

**Category:** Character

**Restriction:** This function is assigned an 18N Level 0 status, and is designed for SBCS data. Do not use this function to process DBCS or MBCS data. For more information, see Internationalization Compatibility.

**Tip:** Use the “KINDEX Function” in SAS Viya National Language Support (NLS): Reference Guide instead to write encoding independent code.
Syntax

FIND(string, substring <, modifier(s)> <, start-position>)
FIND(string, substring <, start-position> <, modifier(s)>)

Required Arguments

string
specifies a character constant, variable, or expression that is searched for substrings.

Tip Enclose a literal string of characters in quotation marks.

substring
is a character constant, variable, or expression that specifies the substring of characters to search for in string.

Tip Enclose a literal string of characters in quotation marks.

Optional Arguments

modifier
is a character constant, variable, or expression that specifies one or more modifiers.
The following modifiers are valid:

i or I
ignores character case during the search. If this modifier is not specified, FIND searches only for character substrings with the same case as the characters in substring.

t or T
trims trailing blanks from string and substring.

Note: If you want to remove trailing blanks from only one character argument instead of both (or all) character arguments, use the TRIM function instead of the FIND function with the T modifier.

Tip If modifier is a constant, enclose it in quotation marks. Specify multiple constants in a single set of quotation marks. Modifier can also be expressed as a variable or an expression.

start-position
is a numeric constant, variable, or expression with an integer value that specifies the position at which the search should start and the direction of the search.

Details

The FIND function searches string for the first occurrence of the specified substring, and returns the position of that substring. If the substring is not found in string, FIND returns a value of 0.

If start-position is not specified, FIND starts the search at the beginning of the string and searches the string from left to right. If start-position is specified, the absolute value of start-position determines the position at which to start the search. The sign of start-position determines the direction of the search.
Value of startpos | Action
---|---
greater than 0 | starts the search at position start-position and the direction of the search is to the right. If start-position is greater than the length of string, FIND returns a value of 0.

less than 0 | starts the search at position –start-position and the direction of the search is to the left. If –start-position is greater than the length of string, the search starts at the end of string.
equal to 0 | returns a value of 0.

Comparisons

• The FIND function searches for substrings of characters in a character string, whereas the FINDC function searches for individual characters in a character string.

• The FIND function and the INDEX function both search for substrings of characters in a character string. However, the INDEX function does not have the modifier nor the start-position arguments.

Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>whereisshe=find('She sells seashells? Yes, she does.','she '); put whereisshe;</td>
<td>27</td>
</tr>
<tr>
<td>variable1='She sells seashells? Yes, she does.'; variable2='she '; variable3='i'; whereisshe_i=find(variable1,variable2,variable3); put whereisshe_i;</td>
<td>1</td>
</tr>
<tr>
<td>expression1='She sells seashells? Yes, she does.'; expression2=kscan('he or she',3)</td>
<td></td>
</tr>
<tr>
<td>xyz='She sells seashells? Yes, she does.'; startposvar=22; whereisshe_22=find(xyz,'she',startposvar); put whereisshe_22;</td>
<td>27</td>
</tr>
<tr>
<td>xyz='She sells seashells? Yes, she does.'; startposexp=1-23; whereisShe_ineg22=find(xyz,'She',startposexp); put whereisShe_ineg22;</td>
<td>14</td>
</tr>
</tbody>
</table>
See Also

Functions:
- “COUNT Function” on page 204
- “FINDC Function” on page 288
- “FINDW Function” on page 295
- “INDEX Function” on page 327

FINDC Function
Searches a string for any character in a list of characters.

Category: Character
Restriction: This function is assigned an I18N Level 0 status, and is designed for SBCS data. Do not use this function to process DBCS or MBCS data. For more information, see Internationalization Compatibility.

Syntax

\[
\text{FINDC}(\text{string}, \text{character-list}) \\
\text{FINDC}(\text{string}, \text{character-list}, \text{modifier(s)}) \\
\text{FINDC}(\text{string}, \text{character-list}, \text{modifier(s}), \text{start-position}) \\
\text{FINDC}(\text{string}, \text{character-list}, \text{start-position}, \text{modifier(s)})
\]

Required Arguments

\text{string}

is a character constant, variable, or expression that specifies the character string to be searched.

Tip Enclose a literal string of characters in quotation marks.

\text{character-list}

is a constant, variable, or character expression that initializes a list of characters. FINDC searches for the characters in this list provided that you do not specify the K modifier in the \text{modifier} argument. If you specify the K modifier, FINDC searches for all characters that are not in this list of characters. You can add more characters to the list by using other modifiers.

\text{modifier}

is a character constant, variable, or expression in which each character modifies the action of the FINDC function. The following characters, in uppercase or lowercase, can be used as modifiers:

- \text{blank}
  is ignored.
a or A
adds alphabetic characters to the list of characters.

b or B
searches from right to left, instead of from left to right, regardless of the sign of the start-position argument.

c or C
adds control characters to the list of characters.

d or D
adds digits to the list of characters.

f or F
adds an underscore and English letters (that is, the characters that can begin a SAS variable name using VALIDVARNAME=V7) to the list of characters.

g or G
adds graphic characters to the list of characters.

h or H
adds a horizontal tab to the list of characters.

i or I
ignores character case during the search.

k or K
searches for any character that does not appear in the list of characters. If you do not specify this modifier, then FINDC searches for any character that appears in the list of characters. This modifier has the same functionality as the v or V modifier.

l or L
adds lowercase letters to the list of characters.

n or N
adds digits, an underscore, and English letters (that is, the characters that can appear in a SAS variable name using VALIDVARNAME=V7) to the list of characters.

o or O
processes the charlist and the modifier arguments only once, rather than every time the FINDC function is called. Using the O modifier in the DATA step (excluding WHERE clauses), or in the SQL procedure can make FINDC run faster when you call it in a loop where the character-list and the modifier arguments do not change.

p or P
adds punctuation marks to the list of characters.

s or S
adds space characters to the list of characters (blank, horizontal tab, vertical tab, carriage return, line feed, and form feed).

t or T
trims trailing blanks from the string and character-list arguments.

Note: If you want to remove trailing blanks from just one character argument instead of both (or all) character arguments, use the TRIM function instead of the FINDC function with the T modifier.

u or U
adds uppercase letters to the list of characters.
v or V
searches for any character that does not appear in the list of characters. If you do not specify this modifier, then FINDC searches for any character that appears in the list of characters. This modifier has the same functionality as the k or K modifier.

w or W
adds printable characters to the list of characters.

x or X
adds hexadecimal characters to the list of characters.

Tip If modifier is a constant, then enclose it in quotation marks. Specify multiple constants in a single set of quotation marks. Modifier can also be expressed as a variable or an expression.

Optional Argument

start-position
is an optional numeric constant, variable, or expression having an integer value that specifies the position at which the search should start and the direction in which to search.

Details

The FINDC function searches string for the first occurrence of the specified characters, and returns the position of the first character found. If no characters are found in string, then FINDC returns a value of 0.

The FINDC function allows character arguments to be null. Null arguments are treated as character strings that have a length of zero. Numeric arguments cannot be null.

If start-position is not specified, FINDC begins the search at the end of the string if you use the B modifier, or at the beginning of the string if you do not use the B modifier.

If start-position is specified, the absolute value of start-position specifies the position at which to begin the search. If you use the B modifier, the search always proceeds from right to left. If you do not use the B modifier, the sign of start-position specifies the direction in which to search. The following table summarizes the search directions:

<table>
<thead>
<tr>
<th>Value of startpos</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>greater than 0</td>
<td>search begins at position start-position and proceeds to the right. If start-position is greater than the length of the string, FINDC returns a value of 0.</td>
</tr>
<tr>
<td>less than 0</td>
<td>search begins at position –start-position and proceeds to the left. If start-position is less than the negative of the length of the string, the search begins at the end of the string.</td>
</tr>
<tr>
<td>equal to 0</td>
<td>returns a value of 0.</td>
</tr>
</tbody>
</table>

Comparisons

• The FINDC function searches for individual characters in a character string, whereas the FIND function searches for substrings of characters in a character string.
The FINDC function and the INDEXC function both search for individual characters in a character string. However, the INDEXC function does not have the modifier nor the start-position arguments.

The FINDC function searches for individual characters in a character string, whereas the VERIFY function searches for the first character that is unique to an expression. The VERIFY function does not have the modifier nor the start-position arguments.

Examples

Example 1: Searching for Characters in a String

This example searches a character string and returns the characters that are found.

```
 data _null_;  
  string='Hi, ho!';  
  charlist='hi';  
  j=0;  
  do until (j=0);  
    j=findc(string, charlist, j+1);  
    if j=0 then put +3 "That's all";  
    else do;  
      c=substr(string, j, 1);  
      put +3 j= c=;  
    end;  
  end;  
 run;  
```

SAS writes the following output to the log:

```
j=2 c=i  
j=5 c=h  
That's all
```

Example 2: Searching for Characters in a String and Ignoring Case

This example searches a character string and returns the characters that are found. The I modifier is used to ignore the case of the characters.

```
 data _null_;  
  string='Hi, ho!';  
  charlist='ho';  
  j=0;  
  do until (j=0);  
    j=findc(string, charlist, j+1, "i");  
    if j=0 then put +3 "That's all";  
    else do;  
      c=substr(string, j, 1);  
      put +3 j= c=;  
    end;  
  end;  
 run;  
```
Example 3: Searching for Characters and Using the K Modifier
This example searches a character string and returns the characters that do not appear in the character list.

```sas
data _null_
  string='Hi, ho!';
  charlist='hi';
  j=0;
  do until (j=0);
    j=findc(string, charlist, "k", j+1);
    if j=0 then put +3 "That's all";
    else do;
      c=substr(string, j, 1);
      put +3 j= c=;
    end;
  end;
run;
```

SAS writes the following output to the log:

```
j=1 c=H
j=3 c=,
j=4 c=
j=6 c=o
j=7 c=! 
That's all
```

Example 4: Searching for the Characters h, i, and Blank
This example searches for the three characters h, i, and blank. The characters h and i are in lowercase. The uppercase characters H and I are ignored in this search.

```sas
data _null_
  whereishi=0;
  do until(whereishi=0);
    whereishi=findc('Hi there, Ian!', 'hi ', whereishi+1);
    if whereishi=0 then put "The End";
    else do;
      whatfound=substr('Hi there, Ian!', whereishi,1);
      put whereishi= whatfound=;
    end;
  end;
run;
```

SAS writes the following output to the log:

```
whereishi=2 whatfound=i
whereishi=3 whatfound=
whereishi=5 whatfound=h
whereishi=10 whatfound=
The End
```
Example 5: Searching for the Characters h and i While Ignoring Case
This example searches for the four characters h, i, H, and I. FINDC with the i modifier ignores character case during the search.

```sas
data _null_; whereishi_i=0; do until(whereishi_i=0); variable1='Hi there, Ian!'; variable2='hi'; variable3='i'; whereishi_i=findc(variable1, variable2, variable3, whereishi_i+1); if whereishi_i=0 then put "The End"; else do; whatfound=substr(variable1, whereishi_i, 1); put whereishi_i= whatfound=; end; end; run;
```

SAS writes the following output to the log:

```
whereishi_i=1 whatfound=H
whereishi_i=2 whatfound=i
whereishi_i=5 whatfound=h
whereishi_i=11 whatfound=I
The End
```

Example 6: Searching for the Characters h and i with Trailing Blanks Trimmed
This example searches for the two characters h and i. FINDC with the t modifier trims trailing blanks from the string argument and the characters argument.

```sas
data _null_; whereishi_t=0; do until(whereishi_t=0); expression1='Hi there, '||'Ian!'; expression2=kscan('bye or hi', 3)||'  '; expression3=trim('t   '); whereishi_t=findc(expression1, expression2, expression3, whereishi_t+1); if whereishi_t=0 then put "The End"; else do; whatfound=substr(expression1, whereishi_t, 1); put whereishi_t= whatfound=; end; end; run;
```

SAS writes the following output to the log:

```
whereishi_t=2 whatfound=i
whereishi_t=5 whatfound=h
The End
```

Example 7: Searching for All Characters, Excluding h, i, H, and I
This example searches for all of the characters in the string, excluding the characters h, i, H, and I. FINDC with the v modifier counts only the characters that do not appear in the
characters argument. This example also includes the i modifier and therefore ignores character case during the search.

```sas
data _null_;
  whereishi_iv=0;
  do until(whereishi_iv=0);
    xyz='Hi there, Ian!';
    whereishi_iv=findc(xyz, 'hi', whereishi_iv+1, 'iv');
    if whereishi_iv=0 then put "The End";
    else do;
      whatfound=substr(xyz, whereishi_iv,1);
      put whereishi_iv= whatfound=;
    end;
  end;
run;
```

SAS writes the following output to the log:

```
whereishi_iv=3 whatfound= 
whereishi_iv=4 whatfound=t  
whereishi_iv=6 whatfound=e  
whereishi_iv=7 whatfound=r  
whereishi_iv=8 whatfound=e  
whereishi_iv=9 whatfound=,  
whereishi_iv=10 whatfound=  
whereishi_iv=12 whatfound=a  
whereishi_iv=13 whatfound=n  
whereishi_iv=14 whatfound=!  
The End
```

### See Also

#### Functions:

- “ANYALNUM Function” on page 80
- “ANYALPHA Function” on page 82
- “ANYCNTRL Function” on page 84
- “ANYDIGIT Function” on page 86
- “ANYGRAPH Function” on page 89
- “ANYPUNCT Function” on page 91
- “ANYPRINT Function” on page 95
- “ANYSPACE Function” on page 99
- “ANYUPPER Function” on page 101
- “ANYXDIGIT Function” on page 102
- “COUNTC Function” on page 206
- “INDEXC Function” on page 329
- “NOTALNUM Function” on page 448
- “NOTALPHA Function” on page 450
- “NOTCNTRL Function” on page 452
• “NOTDIGIT Function” on page 453
• “NOTGRAPH Function” on page 457
• “NOTLOWER Function” on page 459
• “NOTPRINT Function” on page 462
• “NOTPUNCT Function” on page 464
• “NOTSPACE Function” on page 466
• “NOTUPPER Function” on page 468
• “NOTXDIGIT Function” on page 470
• “VERIFY Function” on page 649

FINDW Function

Returns the character position of a word in a string, or returns the number of the word in a string.

**Category:** Character

**Restriction:** This function is assigned an I18N Level 1 status. If possible, avoid I18N Level 1 functions if you are using a non-English language. Under certain circumstances, the I18N Level 1 functions might not work correctly with Double-Byte Character Set (DBCS) or Multi-Byte Character Set (MBCS) encodings. For more information, see Internationalization Compatibility.

**Syntax**

FINDW(string, word <, character(s)>)

FINDW(string, word, character(s), modifier(s) <, start-position>)

FINDW(string, word, character(s), start-position <, modifier(s)>)

FINDW(string, word, start-position <, character(s) <, modifier(s)>>)  

**Required Arguments**

*string*

is a character constant, variable, or expression that specifies the character string to be searched.

*word*

is a character constant, variable, or expression that specifies the word to search for in *string*.

*character*

is an optional character constant, variable, or expression that initializes a list of characters.

The characters in this list are the delimiters that separate words, provided that you do not specify the K modifier in the *modifier* argument. If you specify the K modifier, then all characters that are not in this list are delimiters. You can add more characters to this list by using other modifiers.
start-position

is an optional numeric constant, variable, or expression with an integer value that specifies the position at which the search should begin and the direction in which to search.

modifier

specifies a character constant, variable, or expression in which each non-blank character modifies the action of the FINDW function.

If you use the modifier argument, then it must be positioned after the character argument.

You can use the following characters as modifiers:

blank

is ignored.

a or A

adds alphabetic characters to the list of characters.

b or B

scans from right to left instead of from left to right, regardless of the sign of the start-position argument.

c or C

adds control characters to the list of characters.

d or D

adds digits to the list of characters.

e or E

counts the words that are scanned until the specified word is found, instead of determining the character position of the specified word in the string. Fragments of a word are not counted.

f or F

adds an underscore and English letters (that is, the characters that can begin a SAS variable name using VALIDVARNANE=V7) to the list of characters.

g or G

adds graphic characters to the list of characters.

h or H

adds a horizontal tab to the list of characters.

i or I

ignores the case of the characters.

k or K

causes all characters that are not in the list of characters to be treated as delimiters. If K is not specified, then all characters that are in the list of characters are treated as delimiters.

l or L

adds lowercase letters to the list of characters.

m or M

specifies that multiple consecutive delimiters, and delimiters at the beginning or end of the string argument, refer to words that have a length of zero.

n or N

adds digits, an underscore, and English letters (that is, the characters that can appear after the first character in a SAS variable name using VALIDVARNANE=V7) to the list of characters.
o or O
processes the character and modifier arguments only once, rather than every time the FINDW function is called. Using the O modifier in the DATA step (excluding WHERE clauses), or in the SQL procedure, can make FINDW run faster when you call it in a loop where the character and modifier arguments do not change.

p or P
adds punctuation marks to the list of characters.

q or Q
ignores delimiters that are inside substrings that are enclosed in quotation marks. If the value of the string argument contains unmatched quotation marks, then scanning from left to right produces different words than scanning from right to left.

r or R
removes leading and trailing delimiters from the word argument.

s or S
adds space characters (blank, horizontal tab, vertical tab, carriage return, line feed, and form feed) to the list of characters.

t or T
trims trailing blanks from the string, word, and character arguments.

u or U
adds uppercase letters to the list of characters.

w or W
adds printable characters to the list of characters.

x or X
adds hexadecimal characters to the list of characters.

Details

Definition of “Delimiter”
“Delimiter” refers to any of several characters that are used to separate words. You can specify the delimiters by using the character argument, the modifier argument, or both. If you specify the Q modifier, then the characters inside substrings that are enclosed in quotation marks are not treated as delimiters.

Definition of “Word”
“Word” refers to a substring that has both of the following characteristics:
- bounded on the left by a delimiter or the beginning of the string
- bounded on the right by a delimiter or the end of the string

Note: A word can contain delimiters. In this case, the FINDW function differs from the SCAN function, in which words are defined as not containing delimiters.

Searching for a String
If the FINDW function fails to find a substring that both matches the specified word and satisfies the definition of a word, then FINDW returns a value of 0.

If the FINDW function finds a substring that both matches the specified word and satisfies the definition of a word, the value that is returned by FINDW depends on whether the E modifier is specified:
If you specify the E modifier, then FINDW returns the number of complete words that were scanned while searching for the specified word. If \textit{start-position} specifies a position in the middle of a word, then that word is not counted.

If you do not specify the E modifier, then FINDW returns the character position of the substring that is found.

If you specify the \textit{start-position} argument, then the absolute value of \textit{start-position} specifies the position at which to begin the search. The sign of \textit{start-position} specifies the direction in which to search:

<table>
<thead>
<tr>
<th>Value of \textit{startpos}</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>greater than 0</td>
<td>search begins at position \textit{start-position} and proceeds to the right. If \textit{start-position} is greater than the length of the string, then FINDW returns a value of 0.</td>
</tr>
<tr>
<td>less than 0</td>
<td>search begins at position \textit{start-position} and proceeds to the left. If \textit{start-position} is less than the negative of the length of the string, then the search begins at the end of the string.</td>
</tr>
<tr>
<td>equal to 0</td>
<td>FINDW returns a value of 0.</td>
</tr>
</tbody>
</table>

If you do not specify the \textit{start-position} argument or the B modifier, then FINDW searches from left to right starting at the beginning of the string. If you specify the B modifier, but do not use the \textit{start-position} argument, then FINDW searches from right to left starting at the end of the string.

\textbf{Using the FINDW Function in ASCII and EBCDIC Environments}

If you use the FINDW function with only two arguments, the default delimiters depend on whether your computer uses ASCII or EBCDIC characters.

- If your computer uses ASCII characters, then the default delimiters are as follows:
  
  \texttt{blank ! \$ \% & ( ) * + , - . / ; < ^ |}

  In ASCII environments that do not contain the \texttt{^} character, the FINDW function uses the \texttt{~} character instead.

- If your computer uses EBCDIC characters, then the default delimiters are as follows:
  
  \texttt{blank ! \$ \% & ( ) * + , - . / ; < ^ |}

\textbf{Using Null Arguments}

The FINDW function allows character arguments to be null. Null arguments are treated as character strings with a length of zero. Numeric arguments cannot be null.

\textbf{Examples}

\textit{Example 1: Searching a Character String for a Word}

The following example searches a character string for the word "she", and returns the position of the beginning of the word.

```sas
data _null_
whereisshe=findw('She sells sea shells? Yes, she does.', 'she');
put whereisshe=;
```
Example 2: Searching a Character String and Using the Character and Start-position Arguments
The following example contains two occurrences of the word "rain." Only the second occurrence is found by FINDW because the search begins in position 25. The character argument specifies a space as the delimiter.

```
data _null_;  
  result=findw('At least 2.5 meters of rain falls in a rain forest.', 'rain', ' ', 25);  
  put result=;  
run;  
```

SAS writes the following output to the log:

```
result=40
```

Example 3: Searching a Character String and Using the I Modifier and the Start-position Argument
The following example uses the I modifier and returns the position of the beginning of the word. The I modifier disregards case, and the start-position argument identifies the starting position from which to search.

```
data _null_;  
  string='Artists from around the country display their art at an art festival.';  
  result=findw(string, 'Art', ' ', 'i', 10);  
  put result=;  
run;  
```

SAS writes the following output to the log:

```
result=47
```

Example 4: Searching a Character String and Using the E Modifier
The following example uses the E modifier and returns the number of complete words that are scanned while searching for the word "art."

```
data _null_;  
  string='Artists from around the country display their art at an art festival.';  
  result=findw(string, 'art', ' ', 'E');  
  put result=;  
run;  
```

SAS writes the following output to the log:

```
result=8
```
Example 5: Searching a Character String and Using the E Modifier and the Start-position Argument
The following example uses the E modifier to count words in a character string. The word count begins at position 50 in the string. The result is 3 because "art" is the third word after the 50th character position.

```sas
data _null_
  string='Artists from around the country display their art at an art festival.';
  result=findw(string, 'art', ' ', 'E', 50);
  put result=;
run;
```

SAS writes the following output to the log:

```
result=3
```

Example 6: Searching a Character String and Using Two Modifiers
The following example uses the I and the E modifiers to find a word in a string.

```sas
data _null_
  string='The Great Himalayan National Park was created in 1984. Because of its terrain and altitude, the park supports a diversity of wildlife and vegetation.';
  result=findw(string, 'park', ' ', 'I E');
  put result=;
run;
```

SAS writes the following output to the log:

```
result=5
```

Example 7: Searching a Character String and Using the R Modifier
The following example uses the R modifier to remove leading and trailing delimiters from a word.

```sas
data _null_
  string='Artists from around the country display their art at an art festival.';
  word='  art  ';
  result=findw(string, word, ' ', 'R');
  put result=;
run;
```

SAS writes the following output to the log:

```
result=47
```

See Also

Functions:
- “COUNTW Function” on page 209
- “FIND Function” on page 285
- “FINDC Function” on page 288
FLOOR Function

Returns the largest integer that is less than or equal to the argument, fuzzed to avoid unexpected floating-point results.

**Category:** Truncation

**Syntax**

\[
\text{FLOOR(} \text{argument} \text{)}
\]

**Required Argument**

*argument*

specifies a numeric constant, variable, or expression.

**Details**

If the argument is within 1E-12 of an integer, the function returns that integer.

**Comparisons**

Unlike the FLOORZ function, the FLOOR function fuzzes the result. If the argument is within 1E-12 of an integer, the FLOOR function fuzzes the result to be equal to that integer. The FLOORZ function does not fuzz the result. Therefore, with the FLOORZ function you might get unexpected results.

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>var1=2.1;</td>
<td></td>
</tr>
<tr>
<td>a=floor(var1);</td>
<td></td>
</tr>
<tr>
<td>put a;</td>
<td>2</td>
</tr>
<tr>
<td>var2=-2.4;</td>
<td></td>
</tr>
<tr>
<td>b=floor(var2);</td>
<td></td>
</tr>
<tr>
<td>put b;</td>
<td>-3</td>
</tr>
<tr>
<td>c=floor(-1.6);</td>
<td></td>
</tr>
<tr>
<td>put c;</td>
<td>-2</td>
</tr>
</tbody>
</table>
FLOORZ Function

Returns the largest integer that is less than or equal to the argument, using zero fuzzing.

**Category:** Truncation

**Syntax**

`FLOORZ(argument)`

**Required Argument**

`argument`

is a numeric constant, variable, or expression.

**Comparisons**

Unlike the FLOOR function, the FLOORZ function uses zero fuzzing. If the argument is within 1E-12 of an integer, the FLOOR function fuzzes the result to be equal to that integer. The FLOORZ function does not fuzz the result. Therefore, with the FLOORZ function you might get unexpected results.

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>var1=2.1;</code></td>
<td></td>
</tr>
<tr>
<td><code>a=floorz(var1);</code></td>
<td>2</td>
</tr>
<tr>
<td><code>put a;</code></td>
<td></td>
</tr>
</tbody>
</table>

**See Also**

Functions:

• “FLOORZ Function” on page 302
See Also

Functions:
- “FLOOR Function” on page 301

**FNONCT Function**

Returns the value of the noncentrality parameter of an F distribution.

**Category:** Mathematical

**Syntax**

\[ \text{FNONCT}(x, \text{ndf}, \text{ddf}, \text{probability}) \]

**Required Arguments**

- **x**
  - is a numeric random variable.
  - Range: \( x \geq 0 \)

- **ndf**
  - is a numeric numerator degree of freedom parameter.
  - Range: \( ndf > 0 \)

- **ddf**
  - is a numeric denominator degree of freedom parameter.
  - Range: \( ddf > 0 \)
**probability**

is a probability.

**Range**  \( 0 < \text{probability} < 1 \)

**Details**

The FNONCT function returns the nonnegative noncentrality parameter from a noncentral \( F \) distribution whose parameters are \( x, \text{ndf}, \text{ddf}, \) and \( nc \). If \( \text{probability} \) is greater than the probability from the central \( F \) distribution whose parameters are \( x, \text{ndf}, \) and \( \text{ddf} \), a root to this problem does not exist. In this case a missing value is returned. A Newton-type algorithm is used to find a nonnegative root \( nc \) of the equation

\[
P_f(x \mid \text{ndf}, \text{ddf}, nc) - \text{prob} = 0
\]

The following relationship applies to the preceding equation:

\[
P_f(x \mid \text{ndf}, \text{ddf}, nc) = e^{-\frac{nc}{2}} \sum_{j=0}^{\infty} \left( \frac{nc}{j} \right)^j \frac{(\text{ndf})^x}{j!} \frac{(\text{ddf})^x}{(\text{ndf}+\text{ddf})^x} \left( \frac{\text{ddf}}{2} + j, \frac{\text{ddf}}{2} \right)
\]

In the equation, \( I(\ldots) \) is the probability from the beta distribution that is given by the following equation:

\[
I_x(a, b) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} \int_0^x t^{a-1}(1-t)^{b-1} dt
\]

If the algorithm fails to converge to a fixed point, a missing value is returned.

**Example**

```sas
data work;
x=2;
df=4;
ddf=5;
do nc=1 to 3 by .5;
   prob=probf(x, df, ddf, nc);
ncc=fnonct(x, df, ddf, prob);
output;
end;
run;
proc print;
run;
```
**FUZZ Function**

Returns the nearest integer if the argument is within 1E−12 of that integer.

**Category:** Truncation

**Syntax**

`FUZZ(argument)`

**Required Argument**

`argument` specifies a numeric constant, variable, or expression.

**Details**

The FUZZ function returns the nearest integer value if the argument is within 1E−12 of the integer (that is, if the absolute difference between the integer and argument is less than 1E−12). Otherwise, the argument is returned.

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>var1=5.99999999999999; x=fuzz(var1); put x 16.14</td>
<td>6.000000000000000</td>
</tr>
<tr>
<td>x=fuzz(5.99999999); put x 16.14;</td>
<td>5.999999999000000</td>
</tr>
</tbody>
</table>
GAMINV Function

Returns a quantile from the gamma distribution.

**Category:** Quantile

**Syntax**

GAMINV\((p, a)\)

**Required Arguments**

\(p\)

is a numeric probability.

Range \(0 \leq p < 1\)

\(a\)

is a numeric shape parameter.

Range \(a > 0\)

**Details**

The GAMINV function returns the \(p\)th quantile from the gamma distribution, with shape parameter \(a\). The probability that an observation from a gamma distribution is less than or equal to the returned quantile is \(p\).

*Note:* GAMINV is the inverse of the PROBGAM function.

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>q1=gaminv(0.5, 9);</td>
<td>8.6689511844</td>
</tr>
<tr>
<td>q2=gaminv(0.1, 2.1);</td>
<td>0.5841932369</td>
</tr>
</tbody>
</table>

**See Also**

- “QUANTILE Function” on page 549

GAMMA Function

Returns the value of the gamma function.
Syntax
GAMMA(argument)

Required Argument
argument
specifies a numeric constant, variable, or expression.

Restriction
Nonpositive integers are invalid.

Details
The GAMMA function returns the integral given by
\[ \Gamma(x) = \int_0^\infty t^{x-1}e^{-t} \, dt. \]

For positive integers, \( \Gamma(x) \) is \((x - 1)!\). This function is commonly denoted by \( \Gamma(x) \).

Example
This SAS statement produces this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=gamma(6);</td>
<td>120</td>
</tr>
</tbody>
</table>

GARKHCLPRC Function
Calculates call prices for European options on stocks, based on the Garman-Kohlhagen model.

Syntax
GARKHCLPRC(E, t, S, R_d, R_f, sigma)

Required Arguments
E
is a nonmissing, positive value that specifies the exercise price.

Requirement
Specify E and S in the same units.
\( t \)

is a nonmissing value that specifies the time to maturity, in years.

\( S \)

is a nonmissing, positive value that specifies the spot currency price.

**Requirement** Specify \( S \) and \( E \) in the same units.

\( R_d \)

is a nonmissing, positive fraction that specifies the risk-free domestic interest rate for period \( t \).

**Requirement** Specify a value for \( R_d \) for the same time period as the unit of \( t \).

\( R_f \)

is a nonmissing, positive fraction that specifies the risk-free foreign interest rate for period \( t \).

**Requirement** Specify a value for \( R_f \) for the same time period as the unit of \( t \).

\( \sigma \)

is a nonmissing, positive fraction that specifies the volatility of the currency rate.

**Details**

The GARKHCLPRC function calculates the call prices for European options on stocks, based on the Garman-Kohlhagen model. The function is based on the following relationship:

\[
\text{CALL} = SN(d_1) \left( e^{-R_f^t} \right) - EN(d_2) \left( e^{-R_d^t} \right)
\]

**Arguments**

\( S \)

specifies the spot currency price.

\( N \)

specifies the cumulative normal density function.

\( E \)

specifies the exercise price of the option.

\( t \)

specifies the time to expiration, in years.

\( R_d \)

specifies the risk-free domestic interest rate for period \( t \).

\( R_f \)

specifies the risk-free foreign interest rate for period \( t \).

\[
d_1 = \ln \left( \frac{S}{E} \right) + \left( R_d - R_f + \frac{\sigma^2}{2} \right) t \quad \sigma \sqrt{t}
\]

\[
d_2 = d_1 - \sigma \sqrt{t}
\]

The following arguments apply to the preceding equation:
\( \sigma \) specifies the volatility of the underlying asset.

\( \sigma^2 \) specifies the variance of the rate of return.

For the special case of \( t=0 \), the following equation is true:

\[
\text{CALL} = \max((S - E), 0)
\]

For information about the basics of pricing, see “Using Pricing Functions” on page 8.

**Comparisons**

The GARKHCLPRC function calculates the call prices for European options on stocks, based on the Garman-Kohlhagen model. The GARKHPTPRC function calculates the put prices for European options on stocks, based on the Garman-Kohlhagen model. These functions return a scalar value.

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>a=garkhclprc(40, .5, 38, .06, .04, .2); put a;</td>
<td>1.449425106</td>
</tr>
<tr>
<td>b=garkhclprc(19, .25, 20, .05, .03, .09); put b;</td>
<td>1.1304209448</td>
</tr>
</tbody>
</table>

**See Also**

**Functions:**
- “GARKHPTPRC Function” on page 309

---

**GARKHPTPRC Function**

Calculates put prices for European options on stocks, based on the Garman-Kohlhagen model.

**Category:** Financial

**Syntax**

\[
\text{GARKHPTPRC}(E, t, S, R_d, R_f, \text{sigma})
\]

**Required Arguments**

\( E \)

is a nonmissing, positive value that specifies the exercise price.
Requirement Specify \( E \) and \( S \) in the same units.

\( t \)

is a nonmissing value that specifies the time to maturity, in years.

\( S \)

is a nonmissing, positive value that specifies the spot currency price.

Requirement Specify \( S \) and \( E \) in the same units.

\( R_d \)

is a nonmissing, positive fraction that specifies the risk-free domestic interest rate for period \( t \).

Requirement Specify a value for \( R_d \) for the same time period as the unit of \( t \).

\( R_f \)

is a nonmissing, positive fraction that specifies the risk-free foreign interest rate for period \( t \).

Requirement Specify a value for \( R_f \) for the same time period as the unit of \( t \).

\( \sigma \)

is a nonmissing, positive fraction that specifies the volatility of the currency rate.

Details

The GARKHPTPRC function calculates the put prices for European options on stocks, based on the Garman-Kohlhagen model. The function is based on the following relationship:

\[
PUT = CALL - S \left( e^{-R_f t} \right) + E \left( e^{-R_d t} \right)
\]

Arguments

\( S \)

specifies the spot currency price.

\( E \)

specifies the exercise price of the option.

\( t \)

specifies the time to expiration, in years.

\( R_d \)

specifies the risk-free domestic interest rate for period \( t \).

\( R_f \)

specifies the risk-free foreign interest rate for period \( t \).

\[
d_1 = \frac{\ln \left( \frac{S}{E} \right) + \left( R_d - R_f + \frac{\sigma^2}{2} \right) t}{\sigma \sqrt{t}}
\]

\[
d_2 = d_1 - \sigma \sqrt{t}
\]

The following arguments apply to the preceding equation:
\( \sigma \) specifies the volatility of the underlying asset.

\( \sigma^2 \) specifies the variance of the rate of return.

For the special case of \( t=0 \), the following equation is true:

\[
\text{PUT} = \max((E - S), 0)
\]

For information about the basics of pricing, see “Using Pricing Functions” on page 8.

**Comparisons**

The GARKHPTPRC function calculates the put prices for European options on stocks, based on the Garman-Kohlhagen model. The GARKHCLPRC function calculates the call prices for European options on stocks, based on the Garman-Kohlhagen model. These functions return a scalar value.

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>a=garkhptprc(50, .7, 55, .05, .04, .2);</td>
<td>1.4050880945</td>
</tr>
<tr>
<td>b=garkhptprc(32, .3, 33, .05, .03, .3);</td>
<td>1.5647320514</td>
</tr>
</tbody>
</table>

**See Also**

Functions:
- “GARKHCLPRC Function” on page 307

---

**GCD Function**

Returns the greatest common divisor for one or more integers.

**Category:** Mathematical

**Syntax**

\[
\text{GCD}(x_1, x_2, x_3, \ldots, x_n)
\]

**Required Argument**

\( x \)

specifies a numeric constant, variable, or expression that has an integer value.
Details

The GCD (greatest common divisor) function returns the greatest common divisor of one or more integers. For example, the greatest common divisor for 30 and 42 is 6. The greatest common divisor is also called the highest common factor.

If any of the arguments are missing, then the returned value is a missing value.

Example

The following example returns the greatest common divisor of the integers 10 and 15.

```sas
data _null_;
  x=gcd(10, 15);
  put x=;
run;
```

SAS writes the following output to the log:

```
x=5
```

See Also

Functions:
- “LCM Function” on page 403

GEODIST Function

Returns the geodetic distance between two latitude and longitude coordinates.

**Category:** Distance

**Syntax**

```
GEODIST(latitude-1, longitude-1, latitude-2, longitude-2 <, options>)
```

**Required Arguments**

- `latitude`
  - is a numeric constant, variable, or expression that specifies the coordinate of a given position north or south of the equator. Coordinates that are located north of the equator have positive values; coordinates that are located south of the equator have negative values.

  **Restriction**  If the value is expressed in degrees, it must be between 90 and –90. If the value is expressed in radians, it must be between pi/2 and –pi/2.

- `longitude`
  - is a numeric constant, variable, or expression that specifies the coordinate of a given position east or west of the prime meridian, which runs through Greenwich, England. Coordinates that are located east of the prime meridian have positive values; coordinates that are located west of the prime meridian have negative values.
Restriction  If the value is expressed in degrees, it must be between 180 and –180.
If the value is expressed in radians, it must be between pi and –pi.

Optional Argument

option

specifies a character constant, variable, or expression that contains any of the following characters:

M specifies distance in miles.
K specifies distance in kilometers. K is the default value for distance.
D specifies that input values are expressed in degrees. D is the default for input values.
R specifies that input values are expressed in radians.

Details

The GEODIST function computes the geodetic distance between any two arbitrary latitude and longitude coordinates. Input values can be expressed in degrees or in radians.

Example, “Example 3: Calculating the Geodetic Distance in Miles Using the GEODIST Function with CAS” on page 314 shows how to run the GEODIST function in CAS.

Some language elements are supported in SAS but not in CAS. An example is the DATALINES statement. Some examples in this document use DATALINES in SAS to create a data set using the CAS engine. For more information, see “Functions in SAS and CAS” on page 54.

Examples

Example 1: Calculating the Geodetic Distance in Kilometers

The following example shows the geodetic distance in kilometers between Mobile, AL (latitude 30.68 N, longitude 88.25 W), and Asheville, NC (latitude 35.43 N, longitude 82.55 W). The program uses the default K option.

```sas
data _null_;
  distance=geodist(30.68, -88.25, 35.43, -82.55);
  put 'Distance= ' distance ' kilometers';
run;
```

SAS writes the following output to the log:

```
Distance= 748.6529147 kilometers
```

Example 2: Calculating the Geodetic Distance in Miles Using the GEODIST Function with SAS

The following example uses the M option to compute the geodetic distance in miles between Mobile, AL (latitude 30.68 N, longitude 88.25 W), and Asheville, NC (latitude 35.43 N, longitude 82.55 W).

```sas
data _null_;
  distance=geodist(30.68, -88.25, 35.43, -82.55, 'M');
  put 'Distance = ' distance ' miles';
```
Example 3: Calculating the Geodetic Distance in Miles Using the GEODIST Function with CAS

The following example uses the M option to compute the geodetic distance in miles between Mobile, AL (latitude 30.68 N, longitude 88.25 W), and Asheville, NC (latitude 35.43 N, longitude 82.55 W). The first section of code in this example runs in SAS and creates the data set, dist, with the CAS engine. The second section of code runs in CAS, accesses the data in dist that is in CAS, and then creates the dist2 data set.

```plaintext
data casuser.dist;
  input lat1 long1 lat2 long2;
datalines;
35.2 -78.1 37.6 -79.8;
data casuser.dist2;
  set casuser.dist;
  Distance = geodist(lat1, long1, lat2, long2, 'DM');
  put 'Distance = ' Distance ' miles';
run;
```

Example 4: Calculating the Geodetic Distance with Input Measured in Degrees

The following example uses latitude and longitude values that are expressed in degrees to compute the geodetic distance between two locations. Both the D and the M options are specified in the program.

```plaintext
data _null_;  
  input lat1 long1 lat2 long2;
  Distance = geodist(lat1, long1, lat2, long2, 'DM');
  put 'Distance = ' Distance ' miles';
datalines;
35.2 -78.1 37.6 -79.8;
run;
```

SAS writes the following output to the log:

```
Distance = 190.72474282 miles
```

Example 5: Calculating the Geodetic Distance with Input Measured in Radians

The following example uses latitude and longitude values that are expressed in radians to compute the geodetic distance between two locations. The program converts degrees to radians before executing the GEODIST function. Both the R and the M options are specified in this program.

```plaintext
data _null_;  
  pi = constant('pi');
  lat1 = (pi*lat1)/180;
data casuser.dist;
  input lat1 long1 lat2 long2;
datalines;
35.2 -78.1 37.6 -79.8;
data casuser.dist2;
  set casuser.dist;
  Distance = geodist(lat1, long1, lat2, long2, 'DM');
  put 'Distance = ' Distance ' miles';
run;
```

SAS writes the following output to the log:

```
Distance = 465.29081088 miles
```
long1 = (pi*long1)/180;
lat2 = (pi*lat2)/180;
long2 = (pi*long2)/180;
Distance = geodist(lat1, long1, lat2, long2, 'RM');
put 'Distance=' Distance 'miles';
datalines;
35.2  -78.1  37.6  -79.8
; run;

SAS writes the following output to the log:

Distance= 190.72474282 miles

References


---

**GEOMEAN Function**

Returns the geometric mean.

**Category:** Descriptive Statistics

**Syntax**

GEOMEAN(argument <<, argument, ...>)

**Required Argument**

argument is a nonnegative numeric constant, variable, or expression.

**Tip** The argument list can consist of a variable list, which is preceded by OF.

**Details**

If any argument is negative, then the result is a missing value. A message appears in the log that the negative argument is invalid, and _ERROR_ is set to 1. If any argument is zero, then the geometric mean is zero. If all the arguments are missing values, then the result is a missing value. Otherwise, the result is the geometric mean of the nonmissing values.

Let \( n \) be the number of arguments with nonmissing values, and let \( x_1, x_2, \ldots, x_n \) be the values of those arguments. The geometric mean is the \( n^{th} \) root of the product of the values:

\[
\sqrt[n]{x_1 \times x_2 \times \ldots \times x_n}
\]

Equivalently, the geometric mean is
Floating-point arithmetic often produces tiny numerical errors. Some computations that result in zero when exact arithmetic is used might result in a tiny nonzero value when floating-point arithmetic is used. Therefore, GEOMEAN fuzzes the values of arguments that are approximately zero. When the value of one argument is extremely small relative to the largest argument, the former argument is treated as zero. If you do not want SAS to fuzz the extremely small values, then use the GEOMEANZ function.

**Comparisons**

The MEAN function returns the arithmetic mean (average), and the HARMEAN function returns the harmonic mean, whereas the GEOMEAN function returns the geometric mean of the nonmissing values. Unlike GEOMEANZ, GEOMEAN fuzzes the values of the arguments that are approximately zero.

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1=geomean(1, 2, 2, 4);</td>
<td>2</td>
</tr>
<tr>
<td>x2=geomean(., 2, 4, 8);</td>
<td>4</td>
</tr>
<tr>
<td>x3=geomean(of x1-x2);</td>
<td>2.8284271247</td>
</tr>
</tbody>
</table>

**See Also**

**Functions:**

- “GEOMEANZ Function” on page 316
- “HARMEAN Function” on page 318
- “HARMEANZ Function” on page 319
- “MEAN Function” on page 433

**GEOMEANZ Function**

Returns the geometric mean, using zero fuzzing.

**Category:** Descriptive Statistics

**Syntax**

```
GEOMEANZ(argument <, argument, ...>)
```
**Required Argument**

**argument**
- is a nonnegative numeric constant, variable, or expression.

**Tip**  The argument list can consist of a variable list, which is preceded by OF.

**Details**

If any argument is negative, then the result is a missing value. A message appears in the log that the negative argument is invalid, and _ERROR_ is set to 1. If any argument is zero, then the geometric mean is zero. If all the arguments are missing values, then the result is a missing value. Otherwise, the result is the geometric mean of the nonmissing values.

Let \( n \) be the number of arguments with nonmissing values, and let \( x_1, x_2, \ldots, x_n \) be the values of those arguments. The geometric mean is the \( n^{th} \) root of the product of the values:

\[
\sqrt[n]{(x_1 \times x_2 \times \ldots \times x_n)}
\]

Equivalently, the geometric mean is

\[
\exp\left(\frac{\log(x_1) + \log(x_2) + \ldots + \log(x_n)}{n}\right)
\]

**Comparisons**

The MEAN function returns the arithmetic mean (average), and the HARMEAN function returns the harmonic mean, whereas the GEOMEANZ function returns the geometric mean of the nonmissing values. Unlike GEOMEAN, GEOMEANZ does not fuzz the values of the arguments that are approximately zero.

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1=geomeanz(1, 2, 2, 4);</td>
<td>2</td>
</tr>
<tr>
<td>x2=geomeanz(., 2, 4, 8);</td>
<td>4</td>
</tr>
<tr>
<td>x3=geomeanz(of x1-x2);</td>
<td>2.8284271247</td>
</tr>
</tbody>
</table>

**See Also**

**Functions:**
- “GEOMEAN Function” on page 315
- “HARMEAN Function” on page 318
- “HARMEANZ Function” on page 319
HARMEAN Function

Returns the harmonic mean.

**Category:** Descriptive Statistics

**Syntax**

```
HARMEAN(argument <, argument, …>)
```

**Required Argument**

*argument*

is a nonnegative numeric constant, variable, or expression.

**Tip** The argument list can consist of a variable list, which is preceded by OF.

**Details**

If any argument is negative, then the result is a missing value. A message appears in the log that the negative argument is invalid, and _ERROR_ is set to 1. If all the arguments are missing values, then the result is a missing value. Otherwise, the result is the harmonic mean of the nonmissing values.

If any argument is zero, then the harmonic mean is zero. Otherwise, the harmonic mean is the reciprocal of the arithmetic mean of the reciprocals of the values.

Let $n$ be the number of arguments with nonmissing values, and let $x_1, x_2, \ldots, x_n$ be the values of those arguments. The harmonic mean is

$$\frac{1}{x_1} + \frac{1}{x_2} + \ldots + \frac{1}{x_n}$$

Floating-point arithmetic often produces tiny numerical errors. Some computations that result in zero when exact arithmetic is used might result in a tiny nonzero value when floating-point arithmetic is used. Therefore, HARMEAN fuzzes the values of arguments that are approximately zero. When the value of one argument is extremely small relative to the largest argument, the former argument is treated as zero. If you do not want SAS to fuzz the extremely small values, then use the HARMEANZ function.

**Comparisons**

The MEAN function returns the arithmetic mean (average), and the GEOMEAN function returns the geometric mean, whereas the HARMEAN function returns the harmonic mean of the nonmissing values. Unlike HARMEANZ, HARMEAN fuzzes the values of the arguments that are approximately zero.

**Example**

The following SAS statements produce these results:
### SAS Statement Result

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x1=harmean(1, 2, 4, 4);</code></td>
<td>2</td>
</tr>
<tr>
<td><code>x2=harmean(., 4, 12, 24);</code></td>
<td>8</td>
</tr>
<tr>
<td><code>x3=harmean(of x1-x2);</code></td>
<td>3.2</td>
</tr>
</tbody>
</table>

### See Also

Functions:
- “GEOMEAN Function” on page 315
- “GEOMEANZ Function” on page 316
- “HARMEANZ Function” on page 319
- “MEAN Function” on page 433

### HARMEANZ Function

Returns the harmonic mean, using zero fuzzing.

**Category:** Descriptive Statistics

**Syntax**

`HARMEANZ(argument <, argument, …>)`

**Required Argument**

`argument`

is a nonnegative numeric constant, variable, or expression.

**Tip** The argument list can consist of a variable list, which is preceded by OF.

**Details**

If any argument is negative, then the result is a missing value. A message appears in the log that the negative argument is invalid, and _ERROR_ is set to 1. If all the arguments are missing values, then the result is a missing value. Otherwise, the result is the harmonic mean of the nonmissing values.

If any argument is zero, then the harmonic mean is zero. Otherwise, the harmonic mean is the reciprocal of the arithmetic mean of the reciprocals of the values.

Let \( n \) be the number of arguments with nonmissing values, and let \( x_1, x_2, \ldots, x_n \) be the values of those arguments. The harmonic mean is

\[
\frac{1}{x_1} + \frac{1}{x_2} + \ldots + \frac{1}{x_n}
\]
Comparisons

The MEAN function returns the arithmetic mean (average), and the GEOMEAN function returns the geometric mean, whereas the HARMEANZ function returns the harmonic mean of the nonmissing values. Unlike HARMEAN, HARMEANZ does not fuzz the values of the arguments that are approximately zero.

Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1=harmeanz(1, 2, 4, 4);</td>
<td>2</td>
</tr>
<tr>
<td>x2=harmeanz(. , 4, 12, 24);</td>
<td>8</td>
</tr>
<tr>
<td>x3=harmeanz(of x1-x2);</td>
<td>3.2</td>
</tr>
</tbody>
</table>

See Also

Functions:
- “GEOMEAN Function” on page 315
- “GEOMEANZ Function” on page 316
- “HARMEAN Function” on page 318
- “MEAN Function” on page 433

HBOUND Function

Returns the upper bound of an array.

Category: Array

Syntax

HBOUND<n> (array-name)
HBOUND(array-name, bound-n)

Required Arguments

array-name
is the name of an array that was defined previously in the same DATA step.

bound-n
is a numeric constant, variable, or expression that specifies the dimension for which you want to know the upper bound. Use bound-n only if n is not specified.
Optional Argument

\( n \)

is an integer constant that specifies the dimension for which you want to know the upper bound. If no \( n \) value is specified, the HBOUND function returns the upper bound of the first dimension of the array.

Details

The HBOUND function returns the upper bound of a one-dimensional array or the upper bound of a specified dimension of a multidimensional array. Use HBOUND in array processing to avoid changing the upper bound of an iterative DO group each time you change the bounds of the array. HBOUND and LBOUND can be used together to return the values of the upper and lower bounds of an array dimension.

Comparisons

- HBOUND returns the literal value of the upper bound of an array dimension.
- DIM always returns a total count of the number of elements in an array dimension.

Note: This distinction is important when the lower bound of an array dimension has a value other than 1 and the upper bound has a value other than the total number of elements in the array dimension.

Examples

Example 1: One-Dimensional Array

In this example, HBOUND returns the upper bound of the dimension, a value of 5. Therefore, SAS repeats the statements in the DO loop five times.

```sas
array big{5} weight sex height state city;
do i=1 to hbound(big5);
     ... more SAS statements ...;
end;
```

Example 2: Multidimensional Array

This example shows two ways of specifying the HBOUND function for multidimensional arrays. Both methods return the same value for HBOUND, as shown in the table that follows the SAS code example.

```sas
array mult[2:6,4:13,2] mult1-mult100;
```

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Alternative Syntax</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>HBOUND(MULT)</td>
<td>HBOUND(MULT,1)</td>
<td>6</td>
</tr>
<tr>
<td>HBOUND2(MULT)</td>
<td>HBOUND(MULT,2)</td>
<td>13</td>
</tr>
<tr>
<td>HBOUND3(MULT)</td>
<td>HBOUND(MULT,3)</td>
<td>2</td>
</tr>
</tbody>
</table>

See Also

Functions:
HMS Function

Returns a SAS time value from hour, minute, and second values.

Category: Date and Time

Syntax

HMS(hour, minute, second)

Required Arguments

*hour*

is numeric.

*minute*

is numeric.

*second*

is numeric.

Details

The HMS function returns a positive numeric value that represents a SAS time value.

Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>hrid=hms(12, 45, 10); put hrid / hrid time.;</td>
<td>45910 12:45:10</td>
</tr>
</tbody>
</table>

See Also

Functions:

* “DHMS Function” on page 230
* “HOUR Function” on page 325
* “MINUTE Function” on page 435
* “SECOND Function” on page 600
HOLIDAY Function

Returns a SAS date value of a specified holiday for a specified year.

**Category:** Date and Time

**Syntax**

\[
\text{HOLIDAY('holiday', year)}
\]

**Required Arguments**

`'holiday'`

is a character constant, variable, or expression that specifies one of the values listed in the following table.

Values for `holiday` can be in uppercase or lowercase.

**Table 2.1  Holiday Values and Their Descriptions**

<table>
<thead>
<tr>
<th>Holiday Value</th>
<th>Description</th>
<th>Date Celebrated</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOXING</td>
<td>Boxing Day</td>
<td>December 26</td>
</tr>
<tr>
<td>CANADA</td>
<td>Canada Day</td>
<td>July 1</td>
</tr>
<tr>
<td>CANADAOBSERVED</td>
<td>Canada Day observed</td>
<td>July 1, or July 2 if July 1 is a Sunday</td>
</tr>
<tr>
<td>CHRISTMAS</td>
<td>Christmas</td>
<td>December 25</td>
</tr>
<tr>
<td>COLUMBUS</td>
<td>Columbus Day</td>
<td>2nd Monday in October</td>
</tr>
<tr>
<td>EASTER</td>
<td>Easter Sunday</td>
<td>date varies</td>
</tr>
<tr>
<td>FATHERS</td>
<td>Father's Day</td>
<td>3rd Sunday in June</td>
</tr>
<tr>
<td>HALLOWEEN</td>
<td>Halloween</td>
<td>October 31</td>
</tr>
<tr>
<td>LABOR</td>
<td>Labor Day</td>
<td>1st Monday in September</td>
</tr>
<tr>
<td>MLK</td>
<td>Martin Luther King, Jr.’s birthday</td>
<td>3rd Monday in January beginning in 1986</td>
</tr>
<tr>
<td>MEMORIAL</td>
<td>Memorial Day</td>
<td>last Monday in May (since 1971)</td>
</tr>
<tr>
<td>MOTHERS</td>
<td>Mother's Day</td>
<td>2nd Sunday in May</td>
</tr>
<tr>
<td>NEWYEAR</td>
<td>New Year's Day</td>
<td>January 1</td>
</tr>
</tbody>
</table>
year

is a numeric constant, variable, or expression that specifies a four-digit year. If you use a two-digit year, then you must specify the YEARCUTOFF= system option.

Details

The HOLIDAY function computes the date on which a specific holiday occurs in a specified year. Only certain common U.S. and Canadian holidays are defined for use with this function. (See Table 2.1 on page 323 for a list of valid holidays.)

The definition of many holidays has changed over the years. In the U.S., Executive Order 11582, issued on February 11, 1971, fixed the observance of many U.S. federal holidays.

The current holiday definition is extended indefinitely into the past and future, although many holidays have a fixed date at which they were established. Some holidays have not had a consistent definition in the past.

The HOLIDAY function returns a SAS date value. To convert the SAS date value to a calendar date, use any valid SAS date format, such as the DATE9. format.

Comparisons

In some cases, the HOLIDAY function and the NWKDOM function return the same result. For example, the statement HOLIDAY('THANKSGIVING', 2012); returns the same value as NWKDOM(4, 5, 11, 2012);.
In other cases, the HOLIDAY function and the MDY function return the same result. For example, the statement `HOLIDAY('CHRISTMAS', 2012);` returns the same value as `MDY(12, 25, 2012);`.

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>thanks = holiday('thanksgiving', 2013); format thanks date9.; put thanks;</td>
<td>28NOV2013</td>
</tr>
<tr>
<td>boxing = holiday('boxing', 2013); format boxing date9.; put boxing;</td>
<td>26DEC2013</td>
</tr>
<tr>
<td>easter = holiday('easter', 2013); format easter date9.; put easter;</td>
<td>31MAR2013</td>
</tr>
<tr>
<td>canada = holiday('canada', 2013); format canada date9.; put canada;</td>
<td>01JUL2013</td>
</tr>
<tr>
<td>fathers = holiday('fathers', 2013); format fathers date9.; put fathers;</td>
<td>16JUN2013</td>
</tr>
<tr>
<td>valentines = holiday('valentines', 2013); format valentines date9.; put valentines;</td>
<td>14FEB2013</td>
</tr>
<tr>
<td>victoria = holiday('victoria', 2013); format victoria date9.; put victoria;</td>
<td>20MAY2013</td>
</tr>
</tbody>
</table>

**See Also**

**Functions:**

- “MDY Function” on page 432
- “NWKDOM Function” on page 473

---

**HOUR Function**

Returns the hour from a SAS time or datetime value.

**Category:** Date and Time
Syntax
HOUR(time | datetime)

Required Arguments
time
is a numeric constant, variable, or expression that specifies a SAS time value.
datetime
is a numeric constant, variable, or expression that specifies a SAS datetime value.

Details
The HOUR function returns a numeric value that represents the hour from a SAS time or datetime value. Numeric values can range from 0 through 23. HOUR always returns a positive number.

Example
The following SAS statements produce this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>now='1:30't;</td>
<td></td>
</tr>
<tr>
<td>h=hour(now);</td>
<td>1</td>
</tr>
<tr>
<td>put h;</td>
<td></td>
</tr>
</tbody>
</table>

See Also

Functions:
• “SECOND Function” on page 600

IBESSEL Function

Returns the value of the modified Bessel function.

Category: Mathematical

Syntax
IBESSEL(nu, x, kode)

Required Arguments
nu
specifies a numeric constant, variable, or expression.

Range    nu ≥ 0

x
specifies a numeric constant, variable, or expression.
Range \( x \geq 0 \)

\( kode \)
is a numeric constant, variable, or expression that specifies a nonnegative integer.

**Details**

The IBESSEL function returns the value of the modified Bessel function of order \( nu \) evaluated at \( x \) (Abramowitz, Stegun 1964; Amos, Daniel, Weston 1977). When \( kode \) equals 0, the Bessel function is returned. Otherwise, the value of the following function is returned:

\[ e^{-x} I_{nu}(x) \]

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=ibessel(2, 2, 0);</td>
<td>0.6889484477</td>
</tr>
<tr>
<td>x=ibessel(2, 2, 1);</td>
<td>0.0932390333</td>
</tr>
</tbody>
</table>

**INDEX Function**

Searches a character expression for a string of characters, and returns the position of the string's first character for the first occurrence of the string.

- **Category:** Character
- **Restriction:** This function is assigned an I18N Level 0 status, and is designed for SBCS data. Do not use this function to process DBCS or MBCS data. For more information, see Internationalization Compatibility.
- **Tip:** DBCS equivalent function is KINDEX in SAS Viya National Language Support (NLS): Reference Guide. See “DBCS Compatibility” on page 328.

**Syntax**

\[ INDEX(source, excerpt) \]

**Required Arguments**

- **source**
  - specifies a character constant, variable, or expression to search.

- **excerpt**
  - is a character constant, variable, or expression that specifies the string of characters to search for in source.

**Tips**

Enclose a literal string of characters in quotation marks.
Both leading and trailing spaces are considered part of the *excerpt* argument. To remove trailing spaces, include the TRIM function with the *excerpt* variable inside the INDEX function.

**Details**

**The Basics**
The INDEX function searches *source*, from left to right, for the first occurrence of the string specified in *excerpt*, and returns the position in *source* of the string's first character. If the string is not found in *source*, INDEX returns a value of 0. If there are multiple occurrences of the string, INDEX returns only the position of the first occurrence.

**DBCS Compatibility**
The DBCS equivalent function is KINDEX. For more information, see “KINDEX Function” in SAS Viya National Language Support (NLS): Reference Guide.

**Examples**

**Example 1: Finding the Position of a Variable in the Source String**
The following example finds the first position of the *excerpt* argument in *source*.

```sas
data _null_;  
a = 'ABC.DEF (X=Y)';  
b = 'X=Y';  
x = index(a, b);  
put x=;  
run;
```

SAS writes the following output to the log:

```
x=9
```

**Example 2: Removing Trailing Spaces When You Use the INDEX Function with the TRIM Function**
The following example shows the results when you use the INDEX function with and without the TRIM function. If you use INDEX without the TRIM function, leading and trailing spaces are considered part of the *excerpt* argument. If you use INDEX with the TRIM function, TRIM removes trailing spaces from the *excerpt* argument as you can see in this example. Note that the TRIM function is used inside the INDEX function.

```sas
options nodate nostimer ls=78 ps=60;  
data _null_;  
  length a b $14;  
a='ABC.DEF (X=Y)';  
b='X=Y';  
q=index(a, b);  
w=index(a, trim(b));  
put q= w=;  
run;
```

SAS writes the following output to the log:

```
q=0  w=10
```
INDEXC Function

Searches a character expression for any of the specified characters, and returns the position of that character.

**Category:** Character

**Restriction:** This function is assigned an I18N Level 0 status, and is designed for SBCS data. Do not use this function to process DBCS or MBCS data. For more information, see Internationalization Compatibility.

**Tip:** DBCS equivalent function is KINDEXC in SAS Viya National Language Support (NLS): Reference Guide.

**Syntax**

INDEXC(source, excerpt-1 <, …, excerpt-n>)

**Required Arguments**

*source*

specifies a character constant, variable, or expression to search.

*excerpt*

specifies the character constant, variable, or expression to search for in *source*.

**Tip**  
If you specify more than one excerpt, separate them with a comma.

**Details**

The INDEXC function searches *source*, from left to right, for the first occurrence of any character present in the excerpts and returns the position in *source* of that character. If none of the characters in *excerpt-1* through *excerpt-n* in *source* are found, INDEXC returns a value of 0.

**Comparisons**

The INDEXC function searches for the first occurrence of any individual character that is present within the character string, whereas the INDEX function searches for the first occurrence of the character string as a substring. The FINDC function provides more options.

**Example**

The following SAS statements produce these results:
SAS Statement | Result
--- | ---
a='ABC.DEF (X2=Y1)';
x=indexc(a, '0123', ';()=.');
put x;
4

b='have a good day';
x=indexc(b, 'pleasant', 'very');
put x;
2

See Also

Functions:
- “FINDC Function” on page 288
- “INDEX Function” on page 327
- “INDEXW Function” on page 330

INDEXW Function

Searches a character expression for a string that is specified as a word, and returns the position of the first character in the word.

**Category:** Character

**Restriction:** This function is assigned an I18N Level 0 status, and is designed for SBCS data. Do not use this function to process DBCS or MBCS data. For more information, see Internationalization Compatibility.

**Syntax**

INDEXW(source, excerpt <, delimiters>)

**Required Arguments**

- **source**
  - specifies a character constant, variable, or expression to search.

- **excerpt**
  - specifies a character constant, variable, or expression to search for in source. SAS removes leading and trailing delimiters from excerpt.

**Optional Argument**

- **delimiter**
  - specifies a character constant, variable, or expression containing the characters that you want INDEXW to use as delimiters in the character string. The default delimiter is the blank character.
Details

The INDEXW function searches source, from left to right, for the first occurrence of excerpt and returns the position in source of the substring's first character. If the substring is not found in source, then INDEXW returns a value of 0. If there are multiple occurrences of the string, then INDEXW returns only the position of the first occurrence.

The substring pattern must begin and end on a word boundary. For INDEXW, word boundaries are delimiters, the beginning of source, and the end of source. If you use an alternate delimiter, then INDEXW does not recognize the end of the text as the end data.

INDEXW has the following behavior when the second argument contains blank spaces or has a length of 0:

- If both source and excerpt contain only blank spaces or have a length of 0, then INDEXW returns a value of 1.
- If excerpt contains only blank spaces or has a length of 0, and source contains character or numeric data, then INDEXW returns a value of 0.

The example, “Example 3: Using a Semicolon (;) as the Delimiter with the INDEXW Function with CAS” on page 333 shows how to use the INDEXW function in CAS.

Some language elements are supported in SAS but not in CAS. An example is the DATALINES statement. Some examples in this document use DATALINES in SAS to create a data set using the CAS engine. For more information, see “Functions in SAS and CAS” on page 54.

Comparisons

The INDEXW function searches for strings that are words, whereas the INDEX function searches for patterns as separate words or as parts of other words. INDEXC searches for any characters that are present in the excerpts. The FINDW function provides more options.

Examples

Example 1: Table of SAS Examples

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>s='asdf adog dog'; p='dog '; x=indexw(s, p); put x;</td>
<td>11</td>
</tr>
<tr>
<td>s='abcdef x=y'; p='def'; x=indexw(s, p); put x;</td>
<td>0</td>
</tr>
<tr>
<td>x=&quot;abc,def@ xyz&quot;; abc=indexw(x, &quot; abc &quot;, &quot;;@&quot;); put abc;</td>
<td>0</td>
</tr>
</tbody>
</table>
Example 2: Using a Semicolon (;) as the Delimiter with the INDEXW Function with SAS

The following example shows how to use the semicolon delimiter in a SAS program that also calls the CATX function. A semicolon delimiter must be in place after each call to CATX, and the second argument in the INDEXW function must be trimmed or searches will not be successful.

```sas
data temp;
  infile datalines;
  input name $12.;
  datalines;
  abcdef
  abcdef;
  run;
data temp2;
  set temp;
  format name_list $1024.;
  retain name_list ' ';
  exists=indexw(name_list, trim(name), ';');
  if exists=0 then do
    name_list=catx(';', name_list, name)||' '; 
    name_count +1;
    put '-------------------------------';
    put exists= ;
    put name_list= ;
    put name_count= ;
  end;
  run;
```
SAS writes the following output to the log:

```
-------------------------------
e.exists=0
  n.name_list=abcdef;
  n.name_count=1
```

In this example, the first time CATX is called `name_list` is blank and the value of `name` is 'abcdef'. CATX returns 'abcdef' with no semicolon appended. However, when INDEXW is called the second time, the value of `name_list` is 'abcdef' followed by 1018 (1024–6) blanks, and the value of `name` is 'abcdef' followed by six blanks. Because the third argument in INDEXW is a semicolon (;), the blanks are significant and do not denote a word boundary. Therefore, the second argument cannot be found in the first argument.

If the example has no blanks, the behavior of INDEXW is easier to understand.

**Example 3: Using a Semicolon (;) as the Delimiter with the INDEXW Function with CAS**

This example shows how to use the semicolon delimiter in a SAS program that also calls the CATX function. A semicolon delimiter must be in place after each call to CATX, and the second argument in the INDEXW function must be trimmed or searches will not be successful. The first section of code in this example runs in SAS and creates the data set, temp, with the CAS engine. The second section of code runs in CAS, accesses the data in temp that is in CAS, and then creates the temp2 data set.

```
data casuser.temp;
  input name $12.;
datalines;
abcdef
abcdef
;
run;

data casuser.temp2;
  set casuser.temp;
  format name_list $1024.;
  retain name_list ' ';
  exists=indexw(name_list, trim(name), ';');
  if exists=0 then do
    name_list=catx(';', name_list, name)||';'
    name_count +1;
  put '-------------------------------'
  end;
  run;

SAS writes the following output to the log:

```
-------------------------------
e.exists=0
  n.name_list=abcdef;
  n.name_count=1
```
Example 4: Using a Space as the Delimiter

The following example uses a space as a delimiter. We expect the value of $x$ to be 0 because the complete word ABCDE was not found in the first argument:

$$x = \text{indexw('ABCDEF;XYZ', 'ABCDE', ';');}$$

The only values for the second argument that would return a nonzero result are ABCDEF and XYZ.

```sas
data temp;
  infile datalines;
  input name $12.;
  datalines;
  abcdef
  abcdef;
run;

data temp2;
  set temp;
  format name_list $1024.;
  retain name_list ' ';
  exists = indexw(name_list, name, ' ');
  if exists=0 then do
    name_list = catx(' ', name_list, name);
    name_count +1;
    put '-------------------------------';
    put exists=;
    put name_list=;
    put name_count=;
  end;
run;
```

SAS writes the following output to the log:

```
-------------------------------
exists=0
name_list=abcdef
name_count=1
```

See Also

Functions:

- “FINDW Function” on page 295
- “INDEX Function” on page 327
- “INDEXC Function” on page 329

INPUT Function

Returns the value that is produced when SAS converts an expression by using the specified informat.

Category: Special
Syntax

INPUT(source, <? | ??> informat.)

Required Arguments

\textit{source}

specifies a character constant, variable, or expression to which you want to apply a specific informat.

\textit{? or ??}

specifies the optional question mark (?) and double question mark (??) modifiers that suppress the printing of error messages and input lines when invalid data values are read. The ? modifier suppresses the invalid data message. The ?? modifier suppresses the invalid data message and prevents the automatic variable _ERROR_ from being set to 1 when invalid data is read.

\textit{informat.}

is the SAS informat that you want to apply to the source. This argument must be the name of an informat followed by a period. The argument cannot be a character constant, variable, or expression.

Details

If the INPUT function returns a character value to a variable that has not yet been assigned a length, the variable length is determined by the width of the informat.

The INPUT function enables you to convert the value of \textit{source} by using a specified informat. The informat determines whether the result is numeric or character. Use INPUT to convert character values to numeric values or other character values.

The example, “Example 2: Converting Character Values to Numeric Values Using the INPUT Function with CAS” on page 336 shows how to run the INPUT function in CAS.

Some language elements are supported in SAS but not in CAS. An example is the DATALINES statement. Some examples in this document use DATALINES in SAS to create a data set using the CAS engine. For more information, see “Functions in SAS and CAS” on page 54.

Comparisons

The INPUT function returns the value produced when a SAS expression is converted using a specified informat. You must use an assignment statement to store that value in a variable. The INPUT statement uses an informat to read a data value. Storing that value in a variable is optional.

The INPUT function requires the informat to be specified as a name followed by a period and optional decimal specification. The INPUTC and INPUTN functions allow the informat to be specified as a character constant, variable, or expression.

Examples

Example 1: Converting Character Values to Numeric Values Using the INPUT Function with SAS

This example uses the INPUT function to convert a character value to a numeric value and store the value in another variable. The COMMA9. informat reads the value of the
SALE variable, stripping the commas. The resulting value, 2115353, is stored in FMTSALE.

```sas
data testin;
  input sale $9.;
  fmtsale=input(sale, comma9.);
  datalines;
  2,115,353
;```

**Example 2: Converting Character Values to Numeric Values Using the INPUT Function with CAS**

This example uses the INPUT function to convert a character value to a numeric value and store the value in another variable. The COMMA9. informat reads the value of the SALE variable, stripping the commas. The resulting value, 2115353, is stored in FMTSALE. The first section of code in this example runs in SAS and creates the data set, testin, with the CAS engine. The second section of code runs in CAS, accesses the data in testin that is in CAS, and then creates the testinn data set.

```sas
data casuser.testin;
  input sale $9.;
  datalines;
  2,115,353
;
  data casuser.testinn;
    set casuser.testin;
    fmtsale=input(sale, comma9.);
    put fmtsale=;
  run;
```

**Example 3: Using PUT and INPUT Functions**

In this example, PUT returns a numeric value as a character string. The value 122591 is assigned to the CHARDATE variable. INPUT returns the value of the character string as a SAS date value by using a SAS date informat. The value 11681 is stored in the SASDATE variable.

```sas
numdate=122591;
chardate=put(numdate, z6.);
sasdate=input(chardate, mmddyy6.);
```

**Example 4: Suppressing Error Messages**

In this example, the question mark (?) modifier tells SAS not to print the invalid data error message if it finds data errors. The automatic variable _ERROR_ is set to 1, and input data lines are written to the SAS log.

```sas
y=input(x, ? 3.1);
```

The double question mark (??) modifier suppresses the printing of error messages and input lines and prevents the automatic variable _ERROR_ from being set to 1 when invalid data is read. Therefore, these two examples produce the same result:

- `y=input(x, ?? 2.);`
- `y=input(x, ? 2.); _error_=0;`
INPUTC Function

Enables you to specify a character informat at run time.

**Category:** Special

**Syntax**

```plaintext
INPUTC(source, informat <, w>)
```

**Required Arguments**

*source*

specifies a character constant, variable, or expression to which you want to apply the informat.

*informat*

is a character constant, variable, or expression that contains the character informat that you want to apply to *source*.

**Optional Argument**

*w*

is a numeric constant, variable, or expression that specifies a width to apply to the informat.

**Interaction**

If you specify a width here, it overrides any width specification in the informat.

**Details**

If the INPUTC function returns a value to a variable that has not yet been assigned a length, by default the variable length is determined by the length of the first argument.

**Comparisons**

The INPUTN function enables you to specify a numeric informat at run time. Using the INPUT function is faster because you specify the informat at compile time.
Example

This example shows how to specify character informats. The PROC FORMAT step in this example creates a format, TYPEFMT., that formats the variable values 1, 2, and 3 with the name of one of the three informats that this step also creates. The informats store responses of "positive," "negative," and "neutral" as different words, depending on the type of question. After PROC FORMAT creates the format and informats, the DATA step creates a SAS data set from raw data consisting of a number identifying the type of question and a response. After reading a record, the DATA step uses the value of TYPE to create a variable, RESPINF, that contains the value of the appropriate informat for the current type of question. The DATA step also creates another variable, WORD, whose value is the appropriate word for a response. The INPUTC function assigns the value of WORD based on the type of question and the appropriate informat.

```sas
proc format;
   value typefmt 1='$groupx'
                 2='$groupy'
                 3='$groupz';
   invalue $groupx 'positive'='agree'
                  'negative'='disagree'
                  'neutral'='notsure';
   invalue $groupy 'positive'='accept'
                  'negative'='reject'
                  'neutral'='possible';
   invalue $groupz 'positive'='pass'
                  'negative'='fail'
                  'neutral'='retest';
run;

data answers;
   input type response $;
   respinformat=put(type, typefmt.);
   word=inputc(response, respinformat);
datalines;
1 positive
1 negative
1 neutral
2 positive
2 negative
2 neutral
3 positive
3 negative
3 neutral
;
```

The value of WORD for the first observation is agree. The value of WORD for the last observation is retest.

See Also

Functions:
- “INPUT Function” on page 334
- “INPUTN Function” on page 339
- “PUT Function” on page 545
- “PUTC Function” on page 1025
INPUTN Function

Enables you to specify a numeric informat at run time.

**Category:** Special

**Syntax**

\[
\text{INPUTN}(\text{source, informat } <, w<, d>>)\]

**Required Arguments**

- **source** specifies a character constant, variable, or expression to which you want to apply the informat.
- **informat** is a character constant, variable, or expression that contains the numeric informat that you want to apply to `source`.

**Optional Arguments**

- **w** is a numeric constant, variable, or expression that specifies a width to apply to the informat.
  - **Interaction** If you specify a width here, it overrides any width specification in the informat.

- **d** is a numeric constant, variable, or expression that specifies the number of decimal places to use.
  - **Interaction** If you specify a number here, it overrides any decimal-place specification in the informat.

**Details**

Some language elements are supported in SAS but not in CAS. An example is the DATALINES statement. Some examples in this document use DATALINES in SAS to create a data set using the CAS engine. For more information, see “Functions in SAS and CAS” on page 54.

**Comparisons**

The INPUTC function enables you to specify a character informat at run time. Using the INPUT function is faster because you specify the informat at compile time.

**Example: Specifying Numeric Informats**

This example shows how to specify numeric informats. The PROC FORMAT step creates a format, READDATE., that formats the variable values 1 and 2 with the name of
a SAS date informat. The DATA step creates a SAS data set from raw data originally from two different sources (indicated by the value of the variable SOURCE). Each source specified dates differently. After reading a record, the DATA step uses the value of SOURCE to create a variable, DATEINFORMAT, that contains the value of the appropriate informat for reading the date. The DATA step also creates a new variable, NEWDATE, whose value is a SAS date. The INPUTN function assigns the value of NEWDATE based on the source of the observation and the appropriate informat.

```sas
proc format;
  value readdate 1='date7.'
    2='mmddyy8.';
run;
options yearcutoff=1926;
data fixdates(drop=start dateinformat);
  length jobdesc $12;
  input source id lname $ jobdesc $ start $;
  dateinformat=put(source, readdate.);
  newdate=inputn(start, dateinformat);
datalines;
1 1604 Ziminski writer 09aug99
1 2010 Clavell editor 26jan95
2 1833 Rivera writer 10/25/98
2 2222 Barnes proofreader 3/26/12
;
```

See Also

Functions:
- “INPUT Function” on page 334
- “INPUTC Function” on page 337
- “PUT Function” on page 545
- “PUTC Function” on page 1025
- “PUTN Function” on page 1028

**INT Function**

Returns the integer value, fuzzed to avoid unexpected floating-point results.

**Category:** Truncation

**Syntax**

```
INT(argument)
```

**Required Argument**

`argument`

specifies a numeric constant, variable, or expression.
Details

The INT function returns the integer portion of the argument (truncates the decimal portion). If the argument's value is within 1E-12 of an integer, the function results in that integer. If the value of argument is positive, the INT function has the same result as the FLOOR function. If the value of argument is negative, the INT function has the same result as the CEIL function.

Comparisons

Unlike the INTZ function, the INT function fuzzes the result. If the argument is within 1E-12 of an integer, the INT function fuzzes the result to be equal to that integer. The INTZ function does not fuzz the result. Therefore, with the INTZ function you might get unexpected results.

Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>var1=2.1;</td>
<td></td>
</tr>
<tr>
<td>x=int(var1);</td>
<td></td>
</tr>
<tr>
<td>put x;</td>
<td>2</td>
</tr>
<tr>
<td>var2=-2.4;</td>
<td></td>
</tr>
<tr>
<td>y=int(var2);</td>
<td></td>
</tr>
<tr>
<td>put y;</td>
<td>-2</td>
</tr>
<tr>
<td>a=int(1+1.e-11);</td>
<td></td>
</tr>
<tr>
<td>put a;</td>
<td>1</td>
</tr>
<tr>
<td>b=int(-1.6);</td>
<td></td>
</tr>
<tr>
<td>put b;</td>
<td>-1</td>
</tr>
</tbody>
</table>

See Also

Functions:

- “CEIL Function” on page 171
- “FLOOR Function” on page 301
- “INTZ Function” on page 385

INTCINDEX Function

Returns the cycle index when a date, time, or datetime interval and value are specified.

Category: Date and Time
Syntax

\[
\text{INTCINDEX}(\text{interval} \prec\text{multiple.}<\text{shift-index}>>), \text{date-time-value})
\]

Required Arguments

\textit{interval}

specifies a character constant, a variable, or an expression that contains an interval name such as WEEK, MONTH, or QTR. \textit{Interval} can appear in uppercase or lowercase.

\textbf{TIP} If \textit{interval} is a character constant, then enclose the value in quotation marks.

\textbf{TIP} Valid values for \textit{interval} depend on whether \textit{date-time-value} is a date, time, or datetime value.

Multipliers and shift indexes can be used with the basic interval names to construct more complex interval specifications. The general form of an interval name is as follows:

\textit{interval}<\text{multiple.}\text{shift-index}>

The three parts of the interval name are as follows:

\textit{interval}

specifies the name of the basic interval type. For example, YEAR specifies yearly intervals.

\textit{multiple}

specifies an optional multiplier that sets the interval equal to a multiple of the period of the basic interval type. For example, the interval \textit{YEAR}2 consists of two-year, or biennial, periods.

\textbf{See} “Incrementing Dates and Times By Using Multipliers and By Shifting Intervals” on page 31 for more information.

\textit{shift-index}

specifies an optional shift index that shifts the interval to start at a specified subperiod starting point. For example, \textit{YEAR.3} specifies yearly periods shifted to start on the first of March of each calendar year and to end in February of the following year.

Restrictions

The shift index cannot be greater than the number of subperiods in the whole interval. For example, you could use \textit{YEAR.24}, but \textit{YEAR.25} would be an error because there is no 25th month in a two-year interval.

If the default shift period is the same as the interval, then only multiperiod intervals can be shifted with the optional shift index. For example, because MONTH intervals shift by MONTH periods by default, monthly intervals cannot be shifted with the shift index. However, bimonthly intervals can be shifted with the shift index, because there are two MONTH intervals in each MONTH2 interval. For example, the interval name MONTH2.2 specifies bimonthly periods starting on the first day of even-numbered months.

\textbf{See} “Incrementing Dates and Times By Using Multipliers and By Shifting Intervals” on page 31 for more information.
*date-time-value*

specifies a date, time, or datetime value that represents a time period of a specified interval.

**Details**

The INTCINDEX function returns the index of the seasonal cycle when you specify an interval and a SAS date, time, or datetime value. For example, if the interval is MONTH, each observation in the data corresponds to a particular month. Monthly data is considered to be periodic for a one-year period. A year contains 12 months, so the number of intervals (months) in a seasonal cycle (year) is 12. WEEK is the seasonal cycle for an interval that is equal to DAY. This example returns a value of 36 because September 1, 2013, is the sixth day of the 35th week of the year.

```sas
cycle_index=intcindex('day', '01SEP2013'd );
```

For more information about working with date and time intervals, see “Date and Time Intervals” on page 31.

**Comparisons**

The INTCINDEX function returns the cycle index, whereas the INTINDEX function returns the seasonal index.

In this example, the INTCINDEX function returns the week of the year.

```sas
cycle_index=intcindex('day', '04APR2013'd );
```

In this example, the INTINDEX function returns the day of the week. In the example

```sas
index=intindex('day', '04APR2013'd );
```

In this example, the INTCINDEX function returns the hour of the day.

```sas
cycle_index=intcindex('minute', '01Sep2013:00:00:00'dt);
```

In this example, the INTINDEX function returns the minute of the hour.

```sas
index=intindex('minute', '01Sep2013:00:00:00'dt);
```

In the example `intseas(intcycle('interval'))`, the INTSEAS function returns the maximum number that could be returned by `intcindex('interval', date)`.

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cycle_index1=intcindex('day', '01SEP2013'd );</code></td>
<td>36</td>
</tr>
<tr>
<td><code>put cycle_index1;</code></td>
<td></td>
</tr>
<tr>
<td><code>cycle_index2=intcindex('dtqtr', '23MAY2013:05:03:01'dt);</code></td>
<td>1</td>
</tr>
<tr>
<td><code>put cycle_index2;</code></td>
<td></td>
</tr>
<tr>
<td><code>cycle_index3=intcindex('tenday', '13DEC2013' d);</code></td>
<td>1</td>
</tr>
<tr>
<td><code>put cycle_index3;</code></td>
<td></td>
</tr>
</tbody>
</table>
See Also

Functions:

- “INTCYCLE Function” on page 350
- “INTINDEX Function” on page 361
- “INTSEAS Function” on page 378

INTCK Function

Returns the number of interval boundaries of a given kind that lie between two dates, times, or datetime values.

Category: Date and Time

Syntax

INTCK(interval <multiple> <shift-index>, start-date, end-date, <'method'>)
INTCK(custom-interval, start-date, end-date, <'method'>)

Required Arguments

interval

specifies a character constant, a variable, or an expression that contains an interval name. Interval can appear in uppercase or lowercase.

The type of interval (date, datetime, or time) must match the type of value in start-date.

Multipliers and shift indexes can be used with the basic interval names to construct more complex interval specifications. The general form of an interval name is as follows:

interval<multiple.shift-index>

The three parts of the interval name are listed below:

interval

specifies the name of the basic interval type. For example, YEAR specifies yearly intervals.

multiple

specifies an optional multiplier that sets the interval equal to a multiple of the period of the basic interval type. For example, the interval YEAR2 consists of two-year, or biennial, periods.
custom-interval
specifies a user-defined interval that is defined by a SAS data set. Each observation contains two variables, begin and end.

Requirement
You must use the INTERVALDS system option if you use the custom-interval variable.

See
“Details” on page 346

shift-index
specifies an optional shift index that shifts the interval to start at a specified subperiod starting point. For example, YEAR.3 specifies yearly periods shifted to start on the first of March of each calendar year and to end in February of the following year.

Restrictions
The shift index cannot be greater than the number of subperiods in the entire interval. For example, you could use YEAR2.24, but YEAR2.25 would be an error because there is no 25th month in a two-year interval.

If the default shift period is the same as the interval type, then only multiperiod intervals can be shifted with the optional shift index. For example, MONTH type intervals shift by MONTH subperiods by default. Thus, monthly intervals cannot be shifted with the shift index. However, bimonthly intervals can be shifted with the shift index, because there are two MONTH intervals in each MONTH2 interval. For example, the interval name MONTH2.2 specifies bimonthly periods starting on the first day of even-numbered months.

See
“Incrementing Dates and Times By Using Multipliers and By Shifting Intervals” on page 31

start-date
specifies a SAS expression that represents the starting SAS date, time, or datetime value.

end-date
specifies a SAS expression that represents the ending SAS date, time, or datetime value.

Optional Argument
'method'
specifies that intervals are counted using either a discrete or a continuous method.

You must enclose method in quotation marks. Method can be one of these values:

CONTINUOUS
specifies that continuous time is measured. The interval is shifted based on the starting date.

The continuous method is useful for calculating anniversaries. For example, you can calculate the number of years a couple is married by executing the following program:
data b;
   WeddingDay='14feb2000'd;
   Today=today();
   YearsMarried=intck('YEAR', WeddingDay, today(), 'C');
   format WeddingDay Today date9.;
run;

proc print data=b;
run;

The results are WeddingDay=14FEB2000, Today=01OCT2012, and
YearsMarried=12.

For the CONTINUOUS method, the distance in months between January 15,
2000, and February 15, 2000, is one month.

Alias C or CONT

DISCRETE
specifies that discrete time is measured. The discrete method counts interval
boundaries (for example, end of month).

The default discrete method is useful to sort time series observations into bins for
processing. For example, daily data can be accumulated to monthly data for
processing as a monthly series.

For the DISCRETE method, the distance in months between January 31, 2013,
and February 1, 2013, is one month.

Alias D or DISC

Default DISCRETE

Details

Calendar Interval Calculations
All values within a discrete time interval are interpreted as being equivalent. This means
that the dates of January 1, 2013 and January 15, 2013 are equivalent when you specify a
monthly interval. Both of these dates represent the interval that begins on January 1,
2013 and ends on January 31, 2013. You can use the date for the beginning of the
interval (January 1, 2013) or the date for the end of the interval (January 31, 2013) to
identify the interval. These dates represent all of the dates within the monthly interval.

In the following example, the start-date ('14JAN2013'd), is equivalent to the first quarter
of 2013.

intck('qtr', '14JAN2013'd, '02SEP2013'd);

The end-date ('02SEP2013'd) is equivalent to the third quarter of 2013. The interval
count, that is, the number of times the beginning of an interval is reached in moving
from the start-date to the end-date is 2.

The INTCK function using the default discrete method counts the number of times the
beginning of an interval is reached in moving from the first date to the second. It does
not count the number of complete intervals between two dates:

- The function INTCK('MONTH', '1jan2013'd, '31jan2013'd) returns 0,
because the two dates are within the same month.
The function `INTCK('MONTH', '31jan2013'd, '1feb2013'd)` returns 1, because the two dates lie in different months that are one month apart.

The function `INTCK('MONTH', '1feb2013'd, '31jan2013'd)` returns -1 because the first date is in a later discrete interval than the second date. (INTCK returns a negative value whenever the first date is later than the second date and the two dates are not in the same discrete interval.)

Using the discrete method, WEEK intervals are determined by the number of Sundays, the default first day of the week, that occur between the start-date and the end-date, and not by how many seven-day periods fall between those dates. To count the number of seven-day periods between start-date and end-date, use the continuous method.

Both the multiple and the shift-index arguments are optional and default to 1. For example, YEAR, YEAR1, YEAR.1, and YEAR1.1 are all equivalent ways of specifying ordinary calendar years.

For more information about working with date and time intervals, see “Date and Time Intervals” on page 31.

### Date and Datetime Intervals

The intervals that you need to use with SAS datetime values are SAS datetime intervals. Datetime intervals are formed by adding the prefix "DT" to any date interval. For example, MONTH is a SAS date interval, and DTMONTH is a SAS datetime interval. Similarly, YEAR is a SAS date interval, and DTYEAR is a SAS datetime interval.

To ensure correct results with interval functions, use date intervals with date values and datetime intervals with datetime values. SAS does not return an error message if you use a date value with a datetime interval, but the results are incorrect.

The following example uses the DTDAY datetime interval and returns the number of days between August 1, 2011, and February 1, 2012:

```sas
options intervalds=(interval=libref.dataset-name);

data _null_
  ;
  days=intck('dtday', '01aug2011:00:10:48'dt, '01feb2012:00:10:48'dt);
  put days=;
run;
```

SAS writes the following output to the log:

```
days=184
```

### Custom Time Intervals

A custom time interval is defined by a SAS data set. The data set must contain the begin variable; it can also contain the end and season variables. Each observation represents one interval with the begin variable containing the start of the interval, and the end variable, if present, containing the end of the interval. The intervals must be listed in ascending order. There cannot be gaps between intervals, and intervals cannot overlap.

The SAS system option INTERVALDS= is used to define custom intervals and associate interval data sets with new interval names. The following example shows how to specify the INTERVALDS= system option:

```sas
options intervalds=(interval=libref.dataset-name);
```

**Arguments**

- `interval` specifies the name of an interval. The value of `interval` is the data set that is named in `libref.dataset-name`. 

libref.dataset-name

specifies the libref and data set name of the file that contains user-supplied holidays.

For more information, see “Custom Time Intervals” on page 34.

**Retail Calendar Intervals**

The retail industry often accounts for its data by dividing the yearly calendar into four 13-week periods, based on one of the following formats: 4-4-5, 4-5-4, or 5-4-4. The first, second, and third numbers specify the number of weeks in the first, second, and third month of each period, respectively.

**Examples**

**Example 1: Interval Examples Using INTCK**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>qtr=intck('qtr', '10jan2013'd,</td>
<td>2</td>
</tr>
<tr>
<td>'01jul2013'd);</td>
<td></td>
</tr>
<tr>
<td>put qtr;</td>
<td></td>
</tr>
<tr>
<td>year=intck('year', '31dec2012'd,</td>
<td>1</td>
</tr>
<tr>
<td>'01jan2013'd);</td>
<td></td>
</tr>
<tr>
<td>put year;</td>
<td></td>
</tr>
<tr>
<td>year=intck('year', '01jan2013'd,</td>
<td>0</td>
</tr>
<tr>
<td>'31dec2013'd);</td>
<td></td>
</tr>
<tr>
<td>put year;</td>
<td></td>
</tr>
<tr>
<td>semi=intck('semiyear', '01jan2010'd,</td>
<td>6</td>
</tr>
<tr>
<td>'01jan2013'd);</td>
<td></td>
</tr>
<tr>
<td>put semi;</td>
<td></td>
</tr>
<tr>
<td>weekvar=intck('week2.2', '07jan2013'd,</td>
<td>7</td>
</tr>
<tr>
<td>'01apr2013'd);</td>
<td></td>
</tr>
<tr>
<td>put weekvar;</td>
<td></td>
</tr>
<tr>
<td>wdvar=intck('weekday7w', '01jan2013'd,</td>
<td>27</td>
</tr>
<tr>
<td>'01feb2013'd);</td>
<td></td>
</tr>
<tr>
<td>put wdvar;</td>
<td></td>
</tr>
<tr>
<td>y='year';</td>
<td>10</td>
</tr>
<tr>
<td>date1='01sep2003'd;</td>
<td></td>
</tr>
<tr>
<td>date2='01sep2013'd;</td>
<td></td>
</tr>
<tr>
<td>newyears=intck(y, date1, date2);</td>
<td></td>
</tr>
<tr>
<td>put newyears;</td>
<td></td>
</tr>
<tr>
<td>y=trim('year       ');</td>
<td>6</td>
</tr>
<tr>
<td>date1='1sep2005'd + 300;</td>
<td></td>
</tr>
<tr>
<td>date2='1sep2013'd - 300;</td>
<td></td>
</tr>
<tr>
<td>newyears=intck(y, date1, date2);</td>
<td></td>
</tr>
<tr>
<td>put newyears;</td>
<td></td>
</tr>
</tbody>
</table>
In the second example, INTCK returns a value of 1 even though only one day has elapsed. This result is returned because the interval from December 31, 2012, to January 1, 2013, contains the starting point for the YEAR interval. However, in the third example, a value of 0 is returned even though 364 days have elapsed. This result is returned because the period between January 1, 2013, and December 31, 2013, does not contain the starting point for the interval.

In the fourth example, SAS returns a value of 6 because January 1, 2010, through January 1, 2013, contains six semiyearly intervals. (Note that if the ending date were December 31, 2012, SAS would count five intervals.) In the fifth example, SAS returns a value of 6 because there are six two-week intervals beginning on a first Monday during the period of January 7, 2013, through April 1, 2013. In the sixth example, SAS returns the value 27. That indicates that beginning with January 1, 2013, and counting only Saturdays as weekend days through February 1, 2013, the period contains 27 weekdays.

In the seventh example, the use of variables for the arguments is illustrated.

### Example 2: An Example That Compares Methods

The following example shows different values for `method`:

```sas
data a;
  interval='month';
  start='14FEB2013'd;
  end='13MAR2013'd;
  months_default=intck(interval, start, end);
  months_discrete=intck(interval, start, end, 'd');
  months_continuous=intck(interval, start, end, 'c');
  output;

  end='14MAR2013'd;
  months_default=intck(interval, start, end);
  months_discrete=intck(interval, start, end, 'd');
  months_continuous=intck(interval, start, end, 'c');
  output;

  start='31JAN2013'd;
  end='01FEB2013'd;
  months_default=intck(interval, start, end);
  months_discrete=intck(interval, start, end, 'd');
  months_continuous=intck(interval, start, end, 'c');
  output;
  format start end date9.;
run;

proc print data=a;
run;
```
INTCYCLE Function

Returns the date, time, or datetime interval at the next higher seasonal cycle when a date, time, or datetime interval is specified.

**Category:** Date and Time

**Syntax**

\[
\text{INTCYCLE}(\text{interval} \times <\text{multiple},<\text{shift-index}>>, <\text{seasonality}>)
\]

**Required Arguments**

**interval**

specifies a character constant, a variable, or an expression that contains an interval name such as WEEK, MONTH, or QTR. Interval can appear in uppercase or lowercase.

Multipliers and shift indexes can be used with the basic interval names to construct more complex interval specifications. The general form of an interval name is as follows:

\[
\text{interval}<\text{multiple.shift-index}>
\]

The three parts of the interval name are listed below:

**interval**

specifies the name of the basic interval type. For example, YEAR specifies yearly intervals.
multiple
specifies an optional multiplier that sets the interval equal to a multiple of the period of the basic interval type. For example, the interval YEAR2 consists of two-year, or biennial, periods.

See  “Incrementing Dates and Times By Using Multipliers and By Shifting Intervals” on page 31 for more information.

shift-index
specifies an optional shift index that shifts the interval to start at a specified subperiod starting point. For example, YEAR.3 specifies yearly periods shifted to start on the first of March of each calendar year and to end in February of the following year.

Restrictions The shift index cannot be greater than the number of subperiods in the whole interval. For example, you could use YEAR2.24, but YEAR2.25 would be an error because there is no 25th month in a two-year interval.

If the default shift period is the same as the interval type, then only multiperiod intervals can be shifted with the optional shift index. For example, because MONTH type intervals shift by MONTH subperiods by default, monthly intervals cannot be shifted with the shift index. However, bimonthly intervals can be shifted with the shift index, because there are two MONTH intervals in each MONTH2 interval. For example, the interval name MONTH2.2 specifies bimonthly periods starting on the first day of even-numbered months.

See  “Incrementing Dates and Times By Using Multipliers and By Shifting Intervals” on page 31 for more information.

Optional Argument

seasonality
specifies a numeric value.

This argument enables you to have more flexibility in working with dates and time cycles. You can specify whether you want a 52-week or a 53-week seasonality in a year.

Example In the following example, the function

\[
\text{INTCYCLE('MONTH', 3)}
\]

has a \textit{seasonality} argument and returns the value QTR. The function

\[
\text{INTCYCLE('MONTH')}
\]

does not have a \textit{seasonality} argument and returns the value YEAR.

Details

The Basics
The INTCYCLE function returns the interval of the seasonal cycle, depending on a date, time, or datetime interval. For example, \text{INTCYCLE('MONTH')} returns the value YEAR because the months from January through December constitute a yearly cycle. \text{INTCYCLE('DAY')} returns the value WEEK because the days from Sunday through Saturday constitute a weekly cycle.
See “Incrementing Dates and Times By Using Multipliers and By Shifting Intervals” on page 31 for information about multipliers and shift indexes. See “Commonly Used Time Intervals” on page 32 for information about how intervals are calculated.

For more information about working with date and time intervals, see “Date and Time Intervals” on page 31.

The INTCYCLE function can also be used with calendar intervals from the retail industry. These intervals are ISO 8601 compliant. For more information, see “Retail Calendar Intervals: ISO 8601 Compliant” on page 34.

**Seasonality**

Seasonality is a time series concept that measures cyclical variations at different intervals during the year. In specifying seasonality, the time of year is the most common source of the variations. For example, sales of home heating oil are regularly greater in winter than during other times of the year. Often, certain days of the week cause regular fluctuations in daily time series, such as increased spending on leisure activities during weekends. The INTCYCLE function uses the concept of seasonality and returns the date, time, or datetime interval at the next higher seasonal cycle when a date, time, or datetime interval is specified.

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>cycle_year=intcycle('year'); put cycle_year;</td>
<td>YEAR</td>
</tr>
<tr>
<td>cycle_quarter=intcycle('qtr'); put cycle_quarter;</td>
<td>YEAR</td>
</tr>
<tr>
<td>cycle_3=intcycle('month', 3); put cycle_3;</td>
<td>QTR</td>
</tr>
<tr>
<td>cycle_month=intcycle('month'); put cycle_month;</td>
<td>YEAR</td>
</tr>
<tr>
<td>cycle_weekday=intcycle('weekday'); put cycle_weekday;</td>
<td>WEEK</td>
</tr>
<tr>
<td>cycle_weekday2=intcycle('weekday', 5); put cycle_weekday2;</td>
<td>WEEK</td>
</tr>
<tr>
<td>cycle_day=intcycle('day'); put cycle_day;</td>
<td>WEEK</td>
</tr>
<tr>
<td>cycle_day2=intcycle('day', 10); put cycle_day2;</td>
<td>TENDAY</td>
</tr>
<tr>
<td>var1='second'; cycle_second=intcycle(var1); put cycle_second;</td>
<td>DTMINUTE</td>
</tr>
</tbody>
</table>
INTFIT Function

Returns a time interval that is aligned between two dates.

**Syntax**

`INTFIT(argument-1, argument-2,'type')`

**Required Arguments**

`argument`

specifies a SAS expression that represents a SAS date or datetime value, or an observation.

Tip: Observation numbers are more likely to be used as arguments if date or datetime values are not available.

`'type'`

specifies whether the arguments are SAS date values, datetime values, or observations.

The following values for `type` are valid:

- `d` specifies that `argument-1` and `argument-2` are date values.
- `dt` specifies that `argument-1` and `argument-2` are datetime values.
- `obs` specifies that `argument-1` and `argument-2` are observations.

**Details**

The INTFIT function returns the most likely time interval based on two dates, datetime values, or observations that have been aligned within an interval. INTFIT assumes that the alignment value is SAME, which specifies that the date is aligned to the same calendar date with the corresponding interval increment. For more information about the `alignment` argument, see “INTNX Function” on page 370.

If the arguments that are used with INTFIT are observations, you can determine the cycle of an occurrence by using observation numbers. In the following example, the first two arguments of INTFIT are observation numbers, and the `type` argument is `obs`. If Jason used the gym the first time and the 25th time that a researcher recorded data, you...
could determine the interval by using the following statement: \texttt{interval=intfit(1, 25, 'obs');}. In this case, the value of interval is OBS24.2.

The INTFIT function can also be used with calendar intervals from the retail industry. These intervals are ISO 8601 compliant.

**Examples**

*Example 1: Finding Intervals That Are Aligned between Two Dates*

The following example shows the intervals that are aligned between two dates. The \texttt{type} argument in this example identifies the input as date values.

```sas
data a;
  length interval $20;
  date1='01jan11'd;
  do i=1 to 25;
    date2=intnx('day', date1, i);
    interval=intfit(date1, date2, 'd');
    output;
  end;
  format date1 date2 date.;
run;
proc print data=a;
run;
```
The output shows that if the increment value is one day, then the result of the INTFIT function is DAY. If the increment value is two days, then the result of the INTFIT function is DAY2. If the increment value is three days, then the result is DAY3.2, with a shift index of 3. (If the two input dates are a Friday and a Monday, then the result is WEEKDAY.) If the increment value is seven days, then the result is WEEK.

**Example 2: Finding Intervals That Are Aligned between Two Dates When the Dates Are Identified as Observations**

The following example shows the intervals that are aligned between two dates. The `type` argument in this example identifies the input as observations.

```sas
data a;
  length interval $20;
  date1='01jan11'd;
  do i=1 to 25;
    date2=intnx('day', date1, i);
    interval=intfit(date1, date2, 'obs');
    output;
  end;
  format date1 date2 date.;
run;
```
proc print data=a;
run;

Figure 2.18  Interval Output from the INTFIT Function When Dates Are Identified as Observations

<table>
<thead>
<tr>
<th>Obs</th>
<th>interval</th>
<th>date1</th>
<th>i</th>
<th>date2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>OBS</td>
<td>01JAN11</td>
<td>1</td>
<td>02JAN11</td>
</tr>
<tr>
<td>2</td>
<td>OBS2</td>
<td>01JAN11</td>
<td>2</td>
<td>03JAN11</td>
</tr>
<tr>
<td>3</td>
<td>OBS3</td>
<td>01JAN11</td>
<td>3</td>
<td>04JAN11</td>
</tr>
<tr>
<td>4</td>
<td>OBS4</td>
<td>01JAN11</td>
<td>4</td>
<td>05JAN11</td>
</tr>
<tr>
<td>5</td>
<td>OBS5</td>
<td>01JAN11</td>
<td>5</td>
<td>06JAN11</td>
</tr>
<tr>
<td>6</td>
<td>OBS6</td>
<td>01JAN11</td>
<td>6</td>
<td>07JAN11</td>
</tr>
<tr>
<td>7</td>
<td>OBS7</td>
<td>01JAN11</td>
<td>7</td>
<td>08JAN11</td>
</tr>
<tr>
<td>8</td>
<td>OBS8</td>
<td>01JAN11</td>
<td>8</td>
<td>09JAN11</td>
</tr>
<tr>
<td>9</td>
<td>OBS9</td>
<td>01JAN11</td>
<td>9</td>
<td>10JAN11</td>
</tr>
<tr>
<td>10</td>
<td>OBS10</td>
<td>01JAN11</td>
<td>10</td>
<td>11JAN11</td>
</tr>
<tr>
<td>11</td>
<td>OBS11</td>
<td>01JAN11</td>
<td>11</td>
<td>12JAN11</td>
</tr>
<tr>
<td>12</td>
<td>OBS12</td>
<td>01JAN11</td>
<td>12</td>
<td>13JAN11</td>
</tr>
<tr>
<td>13</td>
<td>OBS13</td>
<td>01JAN11</td>
<td>13</td>
<td>14JAN11</td>
</tr>
<tr>
<td>14</td>
<td>OBS14</td>
<td>01JAN11</td>
<td>14</td>
<td>15JAN11</td>
</tr>
<tr>
<td>15</td>
<td>OBS15</td>
<td>01JAN11</td>
<td>15</td>
<td>16JAN11</td>
</tr>
<tr>
<td>16</td>
<td>OBS16</td>
<td>01JAN11</td>
<td>16</td>
<td>17JAN11</td>
</tr>
<tr>
<td>17</td>
<td>OBS17</td>
<td>01JAN11</td>
<td>17</td>
<td>18JAN11</td>
</tr>
<tr>
<td>18</td>
<td>OBS18</td>
<td>01JAN11</td>
<td>18</td>
<td>19JAN11</td>
</tr>
<tr>
<td>19</td>
<td>OBS19</td>
<td>01JAN11</td>
<td>19</td>
<td>20JAN11</td>
</tr>
<tr>
<td>20</td>
<td>OBS20</td>
<td>01JAN11</td>
<td>20</td>
<td>21JAN11</td>
</tr>
<tr>
<td>21</td>
<td>OBS21</td>
<td>01JAN11</td>
<td>21</td>
<td>22JAN11</td>
</tr>
<tr>
<td>22</td>
<td>OBS22</td>
<td>01JAN11</td>
<td>22</td>
<td>23JAN11</td>
</tr>
<tr>
<td>23</td>
<td>OBS23</td>
<td>01JAN11</td>
<td>23</td>
<td>24JAN11</td>
</tr>
<tr>
<td>24</td>
<td>OBS24</td>
<td>01JAN11</td>
<td>24</td>
<td>25JAN11</td>
</tr>
<tr>
<td>25</td>
<td>OBS25</td>
<td>01JAN11</td>
<td>25</td>
<td>26JAN11</td>
</tr>
</tbody>
</table>

See Also

Functions:
- “INTCK Function” on page 344
- “INTNX Function” on page 370

**INTFMT Function**

Returns a recommended SAS format when a date, time, or datetime interval is specified.

**Category:** Date and Time
Syntax

INTFMT(interval <<multiple,<.shift-index>>>,'size')

Required Arguments

interval

specifies a character constant, a variable, or an expression that contains an interval name such as WEEK, MONTH, or QTR. Interval can appear in uppercase or lowercase.

Multipliers and shift indexes can be used with the basic interval names to construct more complex interval specifications. The general form of an interval name is as follows:

interval<multiple.shift-index>

The three parts of the interval name are as follows:

interval

specifies the name of the basic interval type. For example, YEAR specifies yearly intervals.

multiple

specifies an optional multiplier that sets the interval equal to a multiple of the period of the basic interval type. For example, the interval YEAR2 consists of two-year, or biennial, periods.

See “Incrementing Dates and Times By Using Multipliers and By Shifting Intervals” on page 31 for more information.

shift-index

specifies an optional shift index that shifts the interval to start at a specified subperiod starting point. For example, YEAR.3 specifies yearly periods shifted to start on the first of March of each calendar year and to end in February of the following year.

Restrictions

The shift index cannot be greater than the number of subperiods in the whole interval. For example, you could use YEAR2.24, but YEAR2.25 would be an error because there is no 25th month in a two-year interval.

If the default shift period is the same as the interval type, then only multiperiod intervals can be shifted with the optional shift index. For example, because MONTH type intervals shift by MONTH subperiods by default, monthly intervals cannot be shifted with the shift index. However, bimonthly intervals can be shifted with the shift index, because there are two MONTH intervals in each MONTH2 interval. For example, the interval name MONTH2.2 specifies bimonthly periods starting on the first day of even-numbered months.

See “Incrementing Dates and Times By Using Multipliers and By Shifting Intervals” on page 31 for more information.

'size'

specifies either LONG or SHORT. When a format includes a year value, LONG or L specifies a format that uses a four-digit year. SHORT or S specifies a format that uses a two-digit year.
Details

The INTFMT function returns a recommended format depending on a date, time, or datetime interval for displaying the time ID values that are associated with a time series of a given interval. The valid values of SIZE (LONG, L, SHORT, or S) specify whether to use a two-digit or a four-digit year when the format refers to a SAS date value.

The INTFMT function can also be used with calendar intervals from the retail industry. These intervals are ISO 8601 compliant.

Examples

Example 1

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>fmt1=intfmt('qtr', 's'); put fmt1;</td>
<td>YYQC4.</td>
</tr>
<tr>
<td>fmt2=intfmt('qtr', 'l'); put fmt2;</td>
<td>YYQC6.</td>
</tr>
<tr>
<td>fmt3=intfmt('month', 'l'); put fmt3;</td>
<td>MONYY7.</td>
</tr>
<tr>
<td>fmt4=intfmt('week', 'short'); put fmt4;</td>
<td>WEEKDATX15.</td>
</tr>
<tr>
<td>fmt5=intfmt('week3.2', 'l'); put fmt5;</td>
<td>WEEKDATX17.</td>
</tr>
<tr>
<td>fmt6=intfmt('day', 'long'); put fmt6;</td>
<td>DATE9.</td>
</tr>
<tr>
<td>var1='month2'; fmt7=intfmt(var1, 'long'); put fmt7;</td>
<td>MONYY7.</td>
</tr>
</tbody>
</table>

Example 2

You can also display date and datetime values as strings by using the format that is identified by the INTFMT function. In the following example, INTFIT identifies the intervals of the sashelp.citiwk and sashelp.air data sets. Then INTFMT identifies formats that are based on the intervals. The formats convert the first SAS date value of each data set to a string. The START variable displays the date of the first observation of each data set. This method assumes that the interval of the data set can be identified by examining the first two observations. You can use the TIMESERIES procedure to prepare the data sets for input. More than two observations might be required to identify the difference between a DAY interval and a WEEKDAY interval. This example would need to be modified if the DATE variable contained SAS datetime values.

SAS Language Reference: Concepts For more information, including the variations of the formats produced by different width options, see .
data a(keep=date0 date interval fmt start);
  length start interval fmt $32;
  format date0 date date.;
  set sashelp.air(obs=2) sashelp.citiwk(obs=2);
  date = lag(date);
  if (mod(_n_,2) eq 1) then delete;
  if (mod(_n_,2) eq 0) then interval = intfit(date0, date, 'D');
  if (mod(_n_,2) eq 0) then fmt = intfmt(interval, 'l');
  start = putn(date0, fmt);
run;
proc print;
run;

SAS creates the following output:

Figure 2.19  Displaying DATE and DATETIME Values as Strings

<table>
<thead>
<tr>
<th>Obs</th>
<th>START</th>
<th>INTERVAL</th>
<th>FMT</th>
<th>date0</th>
<th>date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>JAN1949</td>
<td>MONTH</td>
<td>MONYY7.</td>
<td>01JAN49</td>
<td>01FEB49</td>
</tr>
<tr>
<td>2</td>
<td>Sun, 22 Dec 1985</td>
<td>WEEK</td>
<td>WEEKDATX17.</td>
<td>22DEC85</td>
<td>29DEC85</td>
</tr>
</tbody>
</table>

INTGET Function

Returns a time interval based on three date or datetime values.

**Category:** Date and Time

**Syntax**

INTGET(date-1, date-2, date-3)

**Required Argument**

*date*

specifies a SAS date or datetime value.

**Details**

**INTGET Function Intervals**

The INTGET function returns a time interval based on three date or datetime values. The function first determines all possible intervals between the first two dates, and then determines all possible intervals between the second and third dates. If the intervals are the same, INTGET returns that interval. If the intervals for the first and second dates differ, and the intervals for the second and third dates differ, INTGET compares the intervals. If one interval is a multiple of the other, then INTGET returns the smaller of the two intervals. Otherwise, INTGET returns a missing value. INTGET works best with dates generated by the INTNX function whose alignment value is BEGIN.

In the following example, INTGET returns the interval DAY2:

interval=intget('01mar00'd, '03mar00'd, '09mar00'd);
The interval between the first and second dates is DAY2, because the number of days between March 1, 2000, and March 3, 2000, is two. The interval between the second and third dates is DAY6, because the number of days between March 3, 2000, and March 9, 2000, is six. DAY6 is a multiple of DAY2. INTGET returns the smaller of the two intervals.

In the following example, INTGET returns the interval MONTH4:

```
interval=intget('01jan00'd, '01may00'd, '01may01'd);
```

The interval between the first two dates is MONTH4, because the number of months between January 1, 2000, and May 1, 2000, is four. The interval between the second and third dates is YEAR. INTGET determines that YEAR is a multiple of MONTH4 (there are three MONTH4 intervals in YEAR), and returns the smaller of the two intervals.

In the following example, INTGET returns a missing value:

```
interval=intget('01Jan2006'd, '01Apr2006'd, '01Dec2006'd);
```

The interval between the first two dates is MONTH3, and the interval between the second and third dates is MONTH8. INTGET determines that MONTH8 is not a multiple of MONTH3, and returns a missing value.

The intervals that are returned are valid SAS intervals, including multiples of the intervals and shift intervals.

**Note:** If INTGET cannot determine a matching interval, then the function returns a missing value. No message is written to the SAS log.

### Retail Calendar Intervals

The INTGET function can also be used with calendar intervals from the retail industry. These intervals are ISO 8601 compliant.

### Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>interval=intget('01jan00'd, '01jan01'd, '01may01'd); put interval;</td>
<td>MONTH4</td>
</tr>
<tr>
<td>interval=intget('29feb80'd, '28feb82'd, '29feb84'd); put interval;</td>
<td>YEAR2.2</td>
</tr>
<tr>
<td>interval=intget('01feb80'd, '16feb80'd, '01mar80'd); put interval;</td>
<td>SEMIMONTH</td>
</tr>
<tr>
<td>interval=intget('2jan09'd, '2feb10'd, '2mar11'd); put interval;</td>
<td>MONTH13.4</td>
</tr>
<tr>
<td>interval=intget('10feb80'd, '19feb80'd, '28feb80'd); put interval;</td>
<td>DAY9.2</td>
</tr>
<tr>
<td>interval=intget('01apr2006:00:01:02'dt, '01apr2006:00:02:02'dt, '01apr2006:00:03:02'dt); put interval;</td>
<td>MINUTE</td>
</tr>
</tbody>
</table>
INTINDEX Function

Returns the seasonal index when a date, time, or datetime interval and value are specified.

**Category:** Date and Time

**Syntax**

`INTINDEX(interval <multiple.<shift-index>>>, date-value, <seasonality>)`

**Required Arguments**

`interval`

specifies a character constant, a variable, or an expression that contains an interval name such as WEEK, MONTH, or QTR. Interval can appear in uppercase or lowercase.

**TIP** If `interval` is a character constant, then enclose the value in quotation marks.

**TIP** Valid values for `interval` depend on whether `date-value` is a date, a time, or datetime value. For more information, see “Commonly Used Time Intervals” on page 32.

Multipliers and shift indexes can be used with the basic interval names to construct more complex interval specifications. The general form of an interval name is listed below:

`interval<multiple.shift-index>`

The three parts of the interval name are as follows:

`interval`

specifies the name of the basic interval type. For example, YEAR specifies yearly intervals.

`multiple`

specifies an optional multiplier that sets the interval equal to a multiple of the period of the basic interval type. For example, the interval YEAR2 consists of two-year, or biennial, periods.

See “Incrementing Dates and Times By Using Multipliers and By Shifting Intervals” on page 31 for more information.

`shift-index`

specifies an optional shift index that shifts the interval to start at a specified subperiod starting point. For example, YEAR.3 specifies yearly periods shifted to start on the first of March of each calendar year and to end in February of the following year.
Restrictions  The shift index cannot be greater than the number of subperiods in the whole interval. For example, you could use YEAR2.24, but YEAR2.25 would be an error because there is no 25th month in a two-year interval.

If the default shift period is the same as the interval type, then only multiperiod intervals can be shifted with the optional shift index. For example, because MONTH type intervals shift by MONTH subperiods by default, monthly intervals cannot be shifted with the shift index. However, bimonthly intervals can be shifted with the shift index, because there are two MONTH intervals in each MONTH2 interval. For example, the interval name MONTH2.2 specifies bimonthly periods starting on the first day of even-numbered months.

See  “Incrementing Dates and Times By Using Multipliers and By Shifting Intervals” on page 31 for more information.

date-value  specifies a date, time, or datetime value that represents a time period of the given interval.

Optional Argument

seasonality  specifies a number or a cycle.

This argument enables you to have more flexibility in working with dates and time cycles. You can specify whether you want a 52-week or a 53-week seasonality in a year.

Example  In this example, the following functions produce the same result.

```
INTINDEX('MONTH', date, 3);
INTINDEX('MONTH', date, 'QTR');
```

Seasonality in the first example is a number (the number of months), and in the second example seasonality is a cycle (QTR).

Details

INTINDEX Function Intervals

The INTINDEX function returns the seasonal index when you supply an interval and an appropriate date, time, or datetime value. The seasonal index is a number that represents the position of the date, time, or datetime value in the seasonal cycle of the specified interval.

This example returns a value of 12 because there are 12 months in a yearly cycle and December is the 12th month of the year.

```
intindex('month', '01DEC2012'd);
```

In the following examples, INTINDEX returns the same value (1) because both statements have dates that occur in the first quarter of the year 2013.

```
intindex('qtr', '01JAN2013'd);
intindex('qtr', '31MAR2013'd);
```
The following example returns a value of 6 because daily data is weekly periodic and December 7, 2012, is a Friday, the sixth day of the week.

\[
\text{intindex('day', '07DEC2012'd);} 
\]

**How Interval and Date-Time-Value Are Related**

To correctly identify the seasonal index, the interval should agree with the date, time, or datetime value. For example, \text{intindex('month', '01DEC2012'd);} returns a value of 12 because there are 12 months in a yearly interval and December is the 12th month of the year. The MONTH interval requires a SAS date value. The following example returns a value of 6 because there are seven days in a weekly interval and December 7, 2012, is a Friday, the sixth day of the week.

\[
\text{intindex('day', '07DEC2012'd);} 
\]

The DAY interval requires a SAS date value.

This example returns a missing value because the QTR intervals expects the date to be a SAS data value rather than a datetime value.

\[
\text{intindex('qtr', '01JAN2013:00:00:00'dt);} 
\]

This example returns a value of 12. The DTMONTH interval requires a datetime value.

\[
\text{intindex('dtmonth', '01DEC2013:00:00:00'dt);} 
\]

For more information about working with date and time intervals, see “Date and Time Intervals” on page 31.

**Retail Calendar Intervals**

The INTINDEX function can also be used with calendar intervals from the retail industry. These intervals are ISO 8601 compliant. For more information, see “Retail Calendar Intervals: ISO 8601 Compliant” on page 34.

**Seasonality**

Seasonality is a time series concept that measures cyclical variations at different intervals during the year. In specifying seasonality, the time of year is the most common source of the variations. For example, sales of home heating oil are regularly greater in winter than during other times of the year. Often, certain days of the week cause regular fluctuations in daily time series, such as increased spending on leisure activities during weekends. The INTINDEX function uses the concept of seasonality and returns the seasonal index when a date, time, or datetime interval and value are specified. For more information about seasonality and using the forecasting methods in PROC FORECAST, see the SAS/ETS User's Guide.

**Comparisons**

The INTINDEX function returns the seasonal index whereas the INTCINDEX function returns the cycle index.

In the following example, the INTINDEX function returns a value of 5 because April 4, 2013 is a Thursday, the fifth day of the week.

\[
\text{index=intindex('day', '04APR2013'd);} 
\]

Using the same date, the INTCINDEX function returns a value of 14 because April 4, 2013 is the fourteenth week of the year.

\[
\text{cycle_index=intcindex('day', '04APR2013'd);} 
\]

In this example, the INTINDEX function returns the minute of the hour.
index = intindex('minute', '01Sep2012:06:05:04'dt);

Using the same date and time, the INTCINDEX function returns the hour of the day.
cycle_index = intcindex('minute', '01Sep2012:06:05:04'dt);

In the example intseas('interval');, INTSEAS returns the maximum number that could be returned by intindex('interval', date);

Examples

Example 1: Examples of Using INTINDEX with Two Arguments
The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>interval1=intindex('qtr', '14AUG2013'd); put interval1;</td>
<td>3</td>
</tr>
<tr>
<td>interval2=intindex('dtqtr', '23DEC2013:15:09:19'dt); put interval2;</td>
<td>4</td>
</tr>
<tr>
<td>interval3=intindex('hour', '09:05:15't); put interval3;</td>
<td>10</td>
</tr>
<tr>
<td>interval4=intindex('month', '26FEB2013'd); put interval4;</td>
<td>2</td>
</tr>
<tr>
<td>interval5=intindex('dtmonth', '28MAY2013:05:15:00'dt); put interval5;</td>
<td>5</td>
</tr>
<tr>
<td>interval6=intindex('week', '09SEP2013'd); put interval6;</td>
<td>37</td>
</tr>
<tr>
<td>interval7=intindex('tenday', '16APR2013'd); put interval7;</td>
<td>11</td>
</tr>
</tbody>
</table>

Example 2: Example of Seasonality
SAS uses a default seasonal cycle. For example, the assumption is that monthly data is yearly seasonal. However, monthly data could also have a seasonal cycle of semiyearly. This example shows that to use a third argument, seasonality, enables you to specify the seasonality rather than using the default. It also shows how to handle leap years:

data weekly;
  *do year=2007 to 2012;
  year=2004;
  NewYear=holiday('NEWYEAR', year);
  do i=-5 to 5;
    date=intnx('week', NewYear, i);
    output;
  end;
  *end;
format date date.;
  format NewYear date.;
/* The standard leap week is the first week of year. */
/* An alternative method uses a third argument: leap week is week 53. */
title "Using a Third Argument to Control Weekly Seasonality";
data LeapWeekExample;
  set weekly;
  StandardIndex=intindex('week', date);
  IndexWithLeap=intindex('week', date, 53);
run;
proc print;
run;

/* Using a number and an interval can be equivalent for the third argument. */
title "Using the Third Argument as a Number or Cycle";
data Equiv3rdArg;
  set sashelp.air(obs=12);
  defaultSeasonal=intindex('MONTH', date);
  SeasonalArg12=intindex('MONTH', date, 12);
  SeasonalArgYear=intindex('MONTH', date, 'YEAR');
  format date date.;
run;
proc print;
run;

/* Use the third argument for non-standard seasonality. */
title "Using the Third Argument for Non-Standard Seasonality";
data NonStandardSeasonal;
  set sashelp.air(obs=24);
  /* Standard Index - MONTH is Yearly Seasonal */
  StandardIndex=intindex('MONTH', date);
  SemiYrIndex=intindex('MONTH', date, 'SEMIYR');
  Index6=intindex('MONTH', date, 6);
  format date date.;
run;
proc print;
run;
Figure 2.20  Output from the Seasonality Example

### Using a Third Argument to Control Weekly Seasonality

<table>
<thead>
<tr>
<th>Obs</th>
<th>year</th>
<th>NewYear</th>
<th>l</th>
<th>date</th>
<th>StandardIndex</th>
<th>IndexWithLeap</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2004</td>
<td>01JAN04</td>
<td>-5</td>
<td>23NOV03</td>
<td>48</td>
<td>48</td>
</tr>
<tr>
<td>2</td>
<td>2004</td>
<td>01JAN04</td>
<td>-4</td>
<td>30NOV03</td>
<td>49</td>
<td>49</td>
</tr>
<tr>
<td>3</td>
<td>2004</td>
<td>01JAN04</td>
<td>-3</td>
<td>07DEC03</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>4</td>
<td>2004</td>
<td>01JAN04</td>
<td>-2</td>
<td>14DEC03</td>
<td>51</td>
<td>51</td>
</tr>
<tr>
<td>5</td>
<td>2004</td>
<td>01JAN04</td>
<td>-1</td>
<td>21DEC03</td>
<td>52</td>
<td>52</td>
</tr>
<tr>
<td>6</td>
<td>2004</td>
<td>01JAN04</td>
<td>0</td>
<td>28DEC03</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>2004</td>
<td>01JAN04</td>
<td>1</td>
<td>04JAN04</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>2004</td>
<td>01JAN04</td>
<td>2</td>
<td>11JAN04</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>2004</td>
<td>01JAN04</td>
<td>3</td>
<td>18JAN04</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>2004</td>
<td>01JAN04</td>
<td>4</td>
<td>25JAN04</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>11</td>
<td>2004</td>
<td>01JAN04</td>
<td>5</td>
<td>01FEB04</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

### Using the Third Argument as a Number or Cycle

<table>
<thead>
<tr>
<th>Obs</th>
<th>DATE</th>
<th>AIR</th>
<th>defaultSeasonal</th>
<th>SeasonalArg12</th>
<th>SeasonalArgYear</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>01JAN49</td>
<td>112</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>01FEB49</td>
<td>118</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>01MAR49</td>
<td>132</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>01APR49</td>
<td>120</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>01MAY49</td>
<td>121</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>01JUN49</td>
<td>135</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>01JUL49</td>
<td>146</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>01AUG49</td>
<td>146</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>01SEP49</td>
<td>136</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>01OCT49</td>
<td>119</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>01NOV49</td>
<td>104</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>01DEC49</td>
<td>118</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
</tbody>
</table>
INTNEST Function

Calculates the number of whole periods of the smaller interval that will fit into the period of the larger interval.

See Also

Functions:
- “INTCINDEX Function” on page 341
- “INTCYCLE Function” on page 350
- “INTSEAS Function” on page 378

Other References:
- SAS/ETS User's Guide

INTNEST Function

Calculates the number of whole periods of the smaller interval that will fit into the period of the larger interval.

Category: Date and Time

Syntax

INTNEST(interval1, interval2)

Required Arguments

interval1
  specifies the first interval.
interval2
specifies the second interval.

Details
An interval nests within another interval if a whole number of the first interval spans the same time period as the second interval for all time periods. In order to nest, the two intervals must generate beginning and ending dates that align.

If the first interval, interval1, spans a larger time period than the second interval, interval2, then the returned number is positive. If the second interval spans a larger period than the first interval, then the returned number is negative.

The following time series tasks are related to the INTNEST function:

accumulation
if one interval nests into another interval, even with a variable number, accumulation from the smaller time periods into the larger time periods are accomplished with a simple rule. If the intervals do not nest, you should consider transforming a time series from one frequency to another with a more complex rule, for example, an interpolation.

seasonality
many seasonal models require the higher frequency interval nest into the lower frequency seasonal interval with a fixed number of periods.

time reconciliation
time reconciliation requires that the higher frequency interval nest into the lower frequency interval.

The following table contains the types and descriptions of results that are returned by the INTNEST function:

Table 2.2 INTNEST Function Results and Descriptions

<table>
<thead>
<tr>
<th>Result</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Same</td>
<td>The two input intervals define the same time periods for all time periods. INTNEST('MONTH12', 'YEAR')</td>
</tr>
<tr>
<td>1</td>
<td>Variable Number</td>
<td>The first interval contains a whole number of periods of the second interval, but the number varies over time. INTNEST('MONTH', 'DAY')</td>
</tr>
<tr>
<td>-1</td>
<td>Variable Number</td>
<td>The second interval contains a whole number of periods of the first interval, but the number varies over time. INTNEST('DAY', 'YEAR')</td>
</tr>
<tr>
<td>n &gt; 1</td>
<td>Fixed Number</td>
<td>The first interval contains a whole number n periods of the second interval, and that is fixed for all time. INTNEST('WEEK', 'DAY')</td>
</tr>
</tbody>
</table>
### Result | Description | Explanation/Example
---|---|---
\( n < -1 \) | Fixed Number | The second interval contains a whole number \( n \) periods of the first interval, and that is fixed for all time.

\[
\text{INTNEST('DTHOUR', 'DAY')}
\]

**Missing value of M** | Multiple Mismatch | Both intervals cannot nest into the other interval. However, intervals of these types can nest for some multiple values.

\[
\text{INTNEST('SEIMONTH3', 'MONTH')}
\]

**Missing value of S** | Shift Mismatch | Both intervals cannot nest into the other interval. However, if a shift value is changed, then the intervals would be the same or one interval would nest into the other.

\[
\text{INTNEST('SEIMONTH2.2', 'MONTH')}
\]

**Missing value of B** | Base Mismatch | The interval bases define time periods that are so different that nesting is not possible for any multiple or shift. For example, YEAR always begins on January 1 of each year, and is shifted by months. However, YEARV always begins on the Monday on or immediately preceding January 4, and YEARV is shifted by ISO 8601 weeks that begin on Monday. Since January 1 is only a Monday for some years, the intervals will not consistently start on the same day. The same problem exists if the YEAR interval is shifted by months, since the first of a month would not be a Monday for all years.

\[
\text{INTNEST('YEAR', 'YEARV')}
\]

### Example
The following example illustrates the relationship between two intervals that nest and both intervals are either date intervals or datetime intervals. In this example, for each observation, the value calculated for begin1 is the same as begin2, and the value that is calculated for end1 is the same as end2.

```plaintext
/* interval1 and interval2 are any 2 valid intervals */
nest=INTNEST(interval1,interval2);
/* If interval1 and interval2 are date intervals, then start and end are any SAS date values. If interval1 and interval2 are datetime intervals, then start and end are SAS datetime values. This algorithm would need to be modified if a SAS date interval is compared to a SAS datetime interval. */
do date=start to end;
  if ( ( nest = .B ) or
      ( nest = .M ) or
      ( nest = .S ) ) then do;
    /* skip this case as the rule does not apply */
    end;
  else if ( nest = 0 ) then do;
```

**INTNEST Function**

369
begin1=INTNX(interval1,date,0);
begin2=INTNX(interval2,date,0);
end1=INTNX(interval1,date,nest,'E');
end2=INTNX(interval2,date,nest,'E');
end;
else if ( nest = 1 ) then do;
begin1=INTNX(interval1,date,0);
end1=INTNX(interval1,date,0,'E');
n=INTCK(interval2,begin1,end1);
begin2=INTNX(interval2,begin1,0);
end2=INTNX(interval2,begin2,n,'E');
end;
else if ( nest = -1 ) then do;
begin2=INTNX(interval2,date,0);
end2=INTNX(interval2,date,0,'E');
n=INTCK(interval1,begin2,end2);
begin1=INTNX(interval1,begin2,0);
end1=INTNX(interval1,begin1,n,'E');
end;
else if ( nest > 1 ) then do;
begin1=INTNX(interval1,date,0);
begin2=INTNX(interval2,begin1,0);
end1=INTNX(interval1,date,0,'E');
end2=INTNX(interval2,begin2,nest-1,'E');
end;
else if ( nest < 1 ) then do;
begin2=INTNX(interval2,date,0);
begin1=INTNX(interval1,begin2,0);
end1=INTNX(interval1,begin1,(-nest)-1,'E');
end2=INTNX(interval2,date,0,'E');
end;
output;
end;

INTNX Function

Increments a date, time, or datetime value by a given time interval, and returns a date, time, or datetime value.

Category: Date and Time

Syntax

INTNX(interval <multiple><shift-index>, start-from, increment <,'alignment'>)
INTNX(custom-interval, start-from, increment <,'alignment'>)

Required Arguments

interval

specifies a character constant, variable, or expression that contains a time interval such as WEEK, SEMIYEAR, QTR, or HOUR. Interval can appear in uppercase or lowercase.
The type of interval (date, datetime, or time) must match the type of value in `start-from`.

Multipliers and shift indexes can be used with the basic interval names to construct more complex interval specifications. The general form of an interval name is as follows:

```
interval<multiple.shift-index>
```

The three parts of the interval name are listed below:

- **interval**
  specifies the name of the basic interval type. For example, YEAR specifies yearly intervals.

- **multiple**
  specifies an optional multiplier that sets the interval equal to a multiple of the period of the basic interval type. For example, the interval YEAR2 consists of two-year, or biennial, periods.

  See “Incrementing Dates and Times By Using Multipliers and By Shifting Intervals” on page 31 for more information.

- **shift-index**
  specifies an optional shift index that shifts the interval to start at a specified subperiod starting point. For example, YEAR.3 specifies yearly periods shifted to start on the first of March of each calendar year and to end in February of the following year.

  Restrictions
  The shift index cannot be greater than the number of subperiods in the whole interval. For example, you could use YEAR2.24, but YEAR2.25 would be an error because there is no 25th month in a two-year interval.

  If the default shift period is the same as the interval type, then only multiperiod intervals can be shifted with the optional shift index. For example, MONTH type intervals shift by MONTH subperiods by default. Thus, monthly intervals cannot be shifted with the shift index. However, bimonthly intervals can be shifted with the shift index because there are two MONTH intervals in each MONTH2 interval. The interval name MONTH2.2, for example, specifies bimonthly periods starting on the first day of even-numbered months.

  See “Incrementing Dates and Times By Using Multipliers and By Shifting Intervals” on page 31 for more information.

- **start-from**
  specifies a SAS expression that represents a SAS date, time, or datetime value that identifies a starting point.

- **increment**
  specifies a negative, positive, or zero integer that represents the number of date, time, or datetime intervals. *Increment* is the number of intervals to shift the value of *start-from*. 

## Optional Arguments

`alignment`

controls the position of SAS dates within the interval. You must enclose `alignment` in quotation marks. `Alignment` can be one of these values:

**BEGINNING**

specifies that the returned date or datetime value is aligned to the beginning of the interval.

Alias B

**MIDDLE**

specifies that the returned date or datetime value is aligned to the midpoint of the interval, which is the average of the beginning and ending alignment values.

Alias M

**END**

specifies that the returned date or datetime value is aligned to the end of the interval.

Alias E

**SAME**

specifies that the date that is returned has the same alignment as the input date.

Aliases S

<table>
<thead>
<tr>
<th>SAMEDAY</th>
</tr>
</thead>
</table>

See “SAME Alignment” on page 373 for more information.

Default **BEGINNING**

See “Aligning SAS Date Output within Its Intervals” on page 373 for more information.

---

*custom-interval*

specifies an interval that you define.

---

## Details

### The Basics

The INTNX function increments a date, time, or datetime value by intervals such as DAY, WEEK, QTR, and MINUTE, or a custom interval that you define. The increment is based on a starting date, time, or datetime value, and on the number of time intervals that you specify.

The INTNX function returns the SAS date value for the beginning date, time, or datetime value of the interval that you specify in the `start-from` argument. (To convert the SAS date value to a calendar date, use any valid SAS date format, such as the DATE9. format.) The following example shows how to determine the date of the start of the week that is six weeks from the week of October 17, 2003.

```plaintext
x=intnx('week', '17oct03'd, 6);
put x date9. ;
```
INTNX returns the value 23NOV2003.

For more information about working with date and time intervals, see “Date and Time Intervals” on page 31.

**Date and Datetime Intervals**

The intervals that you need to use with SAS datetime values are SAS datetime intervals. Datetime intervals are formed by adding the prefix "DT" to any date interval. For example, MONTH is a SAS date interval, and DTMONTH is a SAS datetime interval. Similarly, YEAR is a SAS date interval, and DTYEAR is a SAS datetime interval.

To ensure correct results with interval functions, use date intervals with date values and datetime intervals with datetime values. SAS does not return an error message if you use a date value with a datetime interval, but the results are incorrect:

```sas
data _null_;  /* The following statement creates expected results. */ date1=intnx('dtday', '01aug11:00:10:48'dt, 1);  /* The following two statements create unexpected results. */ date2=intnx('dtday', '01aug11'd, 1); date3=intnx('dtday', '01aug11:00:10:48'd, 1); put 'Correct Datetime Value   ' date1= datetime19. / 'Incorrect Datetime Value ' date2= datetime19. / 'Incorrect Datetime Value ' date3= datetime19.; run;```

SAS writes the following output to the log:

```
Correct Datetime Value   date1=02AUG2011:00:00:00
Incorrect Datetime Value date2=02JAN1960:00:00:00
Incorrect Datetime Value date3=02JAN1960:00:00:00
```

**Aligning SAS Date Output within Its Intervals**

SAS date values are typically aligned with the beginning of the time interval that is specified with the *interval* argument.

You can use the optional *alignment* argument to specify the alignment of the date that is returned. The values BEGINNING, MIDDLE, or END align the date to the beginning, middle, or end of the interval, respectively.

**SAME Alignment**

If you use the SAME value of the *alignment* argument, then INTNX returns the same calendar date after computing the interval increment that you specified. The same calendar date is aligned based on the interval's shift period, not the interval.

Most of the values of the shift period are equal to their corresponding intervals. The exceptions are the intervals WEEK, WEEKDAY, QTR, SEMIYEAR, YEAR, and their DT counterparts. WEEK and WEEKDAY intervals have a shift period of DAYS; and QTR, SEMIYEAR, and YEAR intervals have a shift period of MONTH. When you use SAME alignment with YEAR, for example, the result is same-day alignment based on MONTH, the interval's shift period. The result is not aligned to the same day of the YEAR interval. If you specify a multiple interval, then the default shift interval is based on the interval, and not on the multiple interval.

When you use SAME alignment for QTR, SEMIYEAR, and YEAR intervals, the computed date is the same number of months from the beginning of the interval as the
input date. The day of the month matches as closely as possible. Because not all months have the same number of days, it is not always possible to match the day of the month.

Alignment Intervals
Use the SAME value of the alignment argument if you want to base the alignment of the computed date on the alignment of the input date:

```plaintext
intnx('week', '15mar2000'd, 1, 'same'); returns 22MAR2000  
intnx('dweek', '15mar2000:8:45'dt, 1, 'same'); returns 22MAR2000:08:45:00  
intnx('year', '15mar2000'd, 5, 'same'); returns 15MAR2005
```

Adjusting Dates
The INTNX function automatically adjusts for the date if the date in the interval that is incremented does not exist. For example:

```plaintext
intnx('month', '15mar2000'd, 5, 'same'); returns 15AUG2000  
intnx('year', '29feb2000'd, 2, 'same'); returns 28FEB2002  
intnx('month', '31aug2001'd, 1, 'same'); returns 30SEP2001  
intnx('year', '01mar1999'd, 1, 'same'); returns 01MAR2000 (the first day of the 3rd month of the year)
```

In the example `intnx('year', '29feb2000'd, 2);`, the INTNX function returns the value 01JAN2002, which is the beginning of the year two years from the starting date (2000).

In the example `intnx('year', '29feb2000'd, 2, 'same');`, the INTNX function returns the value 28FEB2002. In this case, the starting date begins in the year 2000, the year is two years later (2002), the month is the same (February), and the date is the 28th, because that is the closest date to the 29th in February 2002.

Custom Intervals
A custom interval is defined by a SAS data set. The data set must contain the begin variable, and it can also contain the end and season variables. Each observation represents one interval with the begin variable containing the start of the interval, and the end variable, if present, containing the end of the interval. The intervals must be listed in ascending order. You cannot have gaps between intervals, and intervals cannot overlap.

The SAS system option INTERVALDS= is used to define custom intervals and associate interval data sets with new interval names. The following example shows how to specify the INTERVALDS= system option:

```plaintext
options intervalds=(interval=libref.dataset-name);
```

Argument

- interval
  - specifies the name of an interval. The value of interval is the data set that is named in libref.dataset-name.

- libref.dataset-name
  - specifies the libref and data set name of the file that contains user-supplied holidays.

For more information, see “Custom Time Intervals” on page 34.

Retail Calendar Intervals
The retail industry often accounts for its data by dividing the yearly calendar into four 13-week periods, based on one of the following formats: 4-4-5, 4-5-4, or 5-4-4. The first, second, and third numbers specify the number of weeks in the first, second, and third month of each period, respectively.
### Examples

#### Example 1

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>yr=intnx('year', '05feb94'd, 3); put yr / yr date7.;</td>
<td>13515</td>
</tr>
<tr>
<td></td>
<td>01JAN97</td>
</tr>
<tr>
<td>x=intnx('month', '05jan95'd, 0); put x / x date7.;</td>
<td>12784</td>
</tr>
<tr>
<td></td>
<td>01JAN95</td>
</tr>
<tr>
<td>next=intnx('semiyear', '01jan97'd, 1); put next / next date7.;</td>
<td>13696</td>
</tr>
<tr>
<td></td>
<td>01JUL97</td>
</tr>
<tr>
<td>past=intnx('month2', '01aug96'd, -1); put past / past date7.;</td>
<td>13270</td>
</tr>
<tr>
<td></td>
<td>01MAY96</td>
</tr>
<tr>
<td>sm=intnx('semimonth2.2', '01apr97'd, 4); put sm / sm date7.;</td>
<td>13711</td>
</tr>
<tr>
<td></td>
<td>16JUL97</td>
</tr>
<tr>
<td>x='month'; date='1jun1990'd; nextmon=intnx(x, date, 1); put nextmon / nextmon date7.;</td>
<td>11139</td>
</tr>
<tr>
<td></td>
<td>01JUL90</td>
</tr>
<tr>
<td>x1='month     '; x2=trim(x1); date='1jun1990'd - 100; nextmonth=intnx(x2, date, 1); put nextmonth / nextmonth date7.;</td>
<td>11017</td>
</tr>
<tr>
<td></td>
<td>01MAR90</td>
</tr>
</tbody>
</table>

The following examples show the results of advancing a date by using the optional alignment argument.

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>date1=intnx('month', '01jan95'd, 5, 'beginning'); put date1 / date1 date7.;</td>
<td>12935</td>
</tr>
<tr>
<td></td>
<td>01JUN95</td>
</tr>
<tr>
<td>date2=intnx('month', '01jan95'd, 5, 'middle'); put date2 / date2 date7.;</td>
<td>12949</td>
</tr>
<tr>
<td></td>
<td>15JUN95</td>
</tr>
<tr>
<td>date3=intnx('month', '01jan95'd, 5, 'end'); put date3 / date3 date7.;</td>
<td>12964</td>
</tr>
<tr>
<td></td>
<td>30JUN95</td>
</tr>
<tr>
<td>date4=intnx('month', '01jan95'd, 5, 'sameday'); put date4 / date4 date7.;</td>
<td>12935</td>
</tr>
<tr>
<td></td>
<td>01JUN95</td>
</tr>
</tbody>
</table>
Example 2: Example of Using Custom Intervals

The following example uses the *custom-interval* form of the INTNX function to increment a date, time, or datetime value by a given time interval.

```sas
options intervalds=(weekdaycust=dstest);
data dstest;
  format begin end date9.;
  begin='01jan2008'd; end='01jan2008'd; output;
  begin='02jan2008'd; end='02jan2008'd; output;
  begin='03jan2008'd; end='03jan2008'd; output;
  begin='04jan2008'd; end='06jan2008'd; output;
  begin='07jan2008'd; end='07jan2008'd; output;
  begin='08jan2008'd; end='08jan2008'd; output;
  begin='09jan2008'd; end='09jan2008'd; output;
  begin='10jan2008'd; end='10jan2008'd; output;
  begin='11jan2008'd; end='11jan2008'd; output;
  begin='12jan2008'd; end='12jan2008'd; output;
  begin='13jan2008'd; end='13jan2008'd; output;
  begin='14jan2008'd; end='14jan2008'd; output;
  begin='15jan2008'd; end='15jan2008'd; output;
run;

data _null_;%baseline
  format start date9. endcustom date9.;
  start='01jan2008'd;
  do i=0 to 9;
    endcustom=intnx('weekdaycust', start, i);
    put endcustom;
  end;
run;
```
SAS writes the following output to the log:

<table>
<thead>
<tr>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>01JAN2008</td>
</tr>
<tr>
<td>02JAN2008</td>
</tr>
<tr>
<td>03JAN2008</td>
</tr>
<tr>
<td>04JAN2008</td>
</tr>
<tr>
<td>07JAN2008</td>
</tr>
<tr>
<td>08JAN2008</td>
</tr>
<tr>
<td>09JAN2008</td>
</tr>
<tr>
<td>10JAN2008</td>
</tr>
<tr>
<td>11JAN2008</td>
</tr>
<tr>
<td>14JAN2008</td>
</tr>
</tbody>
</table>

**See Also**

**Functions:**
- “INTCK Function” on page 344
- “INTSHIFT Function” on page 382

**System Options:**
- “INTERVALDS= System Option” in SAS Viya System Options: Reference

---

**INTRR Function**

Returns the internal rate of return as a fraction.

**Category:** Financial

**Syntax**

`INTRR(frequency, c0, c1, ..., cn)`

**Required Arguments**

- `frequency` is numeric, the number of payments over a specified base period of time that is associated with the desired internal rate of return.

  **Range**
  
  \( frequency > 0 \)

  **Tip**
  
  The case `frequency = 0` is a flag to allow continuous compounding.

- `c0, c1, ..., cn` are numeric, the optional cash payments.

**Details**

The INTRR function returns the internal rate of return over a specified base period of time for the set of cash payments `c0, c1, ..., cn`. The time intervals between any two consecutive payments are assumed to be equal. The argument `frequency > 0` describes the number of payments that occur over the specified base period of time. The number of notes issued from each instance is limited.
The internal rate of return is the interest rate such that the sequence of payments has a 0 net present value. (See the “NETPV Function” on page 445.) It is given by

\[ r = \begin{cases} \frac{1}{x^{\frac{freq}{x}}} - 1 & \text{freq} > 0 \\
-\log_e(x) & \text{freq} = 0 \end{cases} \]

where \( x \) is the real root of the polynomial.

\[ \sum_{i=0}^{n} c_i x^i = 0 \]

In the case of multiple roots, one real root is returned and a warning is issued concerning the non-uniqueness of the returned internal rate of return. Depending on the value of payments, a root for the equation does not always exist. In that case, a missing value is returned.

Missing values in the payments are treated as 0 values. When \( \text{frequency} > 0 \), the computed rate of return is the effective rate over the specified base period. To compute a quarterly internal rate of return (the base period is three months) with monthly payments, set \( \text{frequency} \) to 3.

If \( \text{frequency} \) is 0, continuous compounding is assumed and the base period is the time interval between two consecutive payments. The computed internal rate of return is the nominal rate of return over the base period. To compute with continuous compounding and monthly payments, set \( \text{frequency} \) to 0. The computed internal rate of return will be a monthly rate.

**Comparisons**

The IRR function is identical to INTRR, except for in the IRR function, the internal rate of return is a percentage.

**Example**

For an initial outlay of $400 and expected payments of $100, $200, and $300 over the following three years, the annual internal rate of return can be expressed as

\[ \text{rate} = \text{intrr}(1, -400, 100, 200, 300); \]

The value returned is 0.19438.

**See Also**

Functions:

- “IRR Function” on page 389

**INTSEAS Function**

Returns the length of the seasonal cycle when a date, time, or datetime interval is specified.

**Category:** Date and Time
Syntax

INTSEAS(interval <<multiple.<shift-index>>> <seasonality>)

Required Argument

interval

specifies a character constant, a variable, or an expression that contains an interval name such as WEEK, MONTH, or QTR. Interval can appear in uppercase or lowercase.

Multipliers and shift indexes can be used with the basic interval names to construct more complex interval specifications. The general form of an interval name is as follows:

interval<multiple.shift-index>

The three parts of the interval name are as follows:

interval

specifies the name of the basic interval type. For example, YEAR specifies yearly intervals.

multiple

specifies an optional multiplier that sets the interval equal to a multiple of the period of the basic interval type. For example, the interval YEAR2 consists of two-year, or biennial, periods.

See “Incrementing Dates and Times By Using Multipliers and By Shifting Intervals” on page 31 for more information.

shift-index

specifies an optional shift index that shifts the interval to start at a specified subperiod starting point. For example, YEAR.3 specifies yearly periods shifted to start on the first of March of each calendar year and to end in February of the following year.

Restrictions

The shift index cannot be greater than the number of subperiods in the whole interval. For example, you could use YEAR2.24, but YEAR2.25 would be an error because there is no 25th month in a two-year interval.

If the default shift period is the same as the interval type, then only multiperiod intervals can be shifted with the optional shift index. For example, because MONTH type intervals shift by MONTH subperiods by default, monthly intervals cannot be shifted with the shift index. However, bimonthly intervals can be shifted with the shift index, because there are two MONTH intervals in each MONTH2 interval. For example, the interval name MONTH2.2 specifies bimonthly periods starting on the first day of even-numbered months.

See “Incrementing Dates and Times By Using Multipliers and By Shifting Intervals” on page 31 for more information.

Optional Argument

seasonality

specifies a number or a cycle.
This argument enables you to have more flexibility in working with dates and time cycles. If there is a 53-week year, you can easily determine the seasonality by using 53 as the value for \textit{seasonality}, as the following example shows:

\texttt{INTSEAS('WEEK', 53)}; By default, \texttt{INTSEAS('WEEK')}; equals 52.

\textbf{Example} 

The function \texttt{INTSEAS('\textit{interval}', \textit{seasonality})};

returns a number when you specify a numeric value for \textit{seasonality}. The function

\texttt{INTSEAS('MONTH', 'QTR')};

returns a value of 3 when you specify the QTR cycle.

\section*{Details}

\textbf{The Basics} 

The \texttt{INTSEAS} function returns the number of intervals in a seasonal cycle. For example, when the interval for a time series is described as monthly, then many procedures use the option \texttt{INTERVAL=MONTH}. Each observation in the data then corresponds to a particular month. Monthly data is considered to be periodic for a one-year period. A year contains 12 months, so the number of intervals (months) in a seasonal cycle (year) is 12.

Quarterly data is also considered to be periodic for a one-year period. A year contains four quarters, so the number of intervals in a seasonal cycle is four.

The periodicity is not always one year. For example, \texttt{INTERVAL=DAY} is considered to have a period of one week. Because there are seven days in a week, the number of intervals in the seasonal cycle is seven.

For more information about working with date and time intervals, see \textit{“Date and Time Intervals” on page 31}. 

\textbf{Retail Calendar Intervals} 

The retail industry often accounts for its data by dividing the yearly calendar into four 13-week periods, based on one of the following formats: 4-4-5, 4-5-4, or 5-4-4. The first, second, and third numbers specify the number of weeks in the first, second, and third month of each period, respectively.

\textbf{Seasonality} 

Seasonality is a time series concept that measures cyclical variations at different intervals during the year. In specifying seasonality, the time of year is the most common source of the variations. For example, sales of home heating oil are regularly greater in winter than during other times of the year. Often, certain days of the week cause regular fluctuations in daily time series, such as increased spending on leisure activities during weekends. The \texttt{INTSEAS} function uses the concept of seasonality and returns the length of the seasonal cycle when a date, time, or datetime interval is specified. For more information about seasonality and forecasting, see the \textit{SAS/ETS User's Guide}. 
Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>cycle_years=intseas('year'); put cycle_years;</td>
<td>1</td>
</tr>
<tr>
<td>cycle_smiyears=intseas('semiyear'); put cycle_smiyears;</td>
<td>2</td>
</tr>
<tr>
<td>cycle_quarters=intseas('quarter'); put cycle_quarters;</td>
<td>4</td>
</tr>
<tr>
<td>cycle_number=intseas('month', 'qtr'); put cycle_number;</td>
<td>3</td>
</tr>
<tr>
<td>cycle_months=intseas('month'); put cycle_months;</td>
<td>12</td>
</tr>
<tr>
<td>cycle_smimonths=intseas('semimonth'); put cycle_smimonths;</td>
<td>24</td>
</tr>
<tr>
<td>cycle_tendays=intseas('tenday'); put cycle_tendays;</td>
<td>36</td>
</tr>
<tr>
<td>cycle_weeks=intseas('week'); put cycle_weeks;</td>
<td>52</td>
</tr>
<tr>
<td>cycle_wkdays=intseas('weekday'); put cycle_wkdays;</td>
<td>5</td>
</tr>
<tr>
<td>cycle_hours=intseas('hour'); put cycle_hours;</td>
<td>24</td>
</tr>
<tr>
<td>cycle_minutes=intseas('minute'); put cycle_minutes;</td>
<td>60</td>
</tr>
<tr>
<td>cycle_month2=intseas('month2.2'); put cycle_month2;</td>
<td>6</td>
</tr>
<tr>
<td>cycle_week2=intseas('week2'); put cycle_week2;</td>
<td>26</td>
</tr>
<tr>
<td>var1='month4.3'; cycle_var1=intseas(var1); put cycle_var1;</td>
<td>3</td>
</tr>
<tr>
<td>cycle_day1=intseas('day1'); put cycle_day1;</td>
<td>7</td>
</tr>
</tbody>
</table>
INTSHIFT Function

Returns the shift interval that corresponds to the base interval.

Category: Date and Time

Syntax

INTSHIFT(interval <<multiple.<shift-index>>>)

Required Arguments

interval specifies a character constant, a variable, or an expression that contains a time interval such as WEEK, SEMIYEAR, QTR, or HOUR. Interval can appear in uppercase or lowercase.

Multipliers and shift indexes can be used with the basic interval names to construct more complex interval specifications. The general form of an interval name is as follows:

interval<multiple.shift-index>

The three parts of the interval name are as follows:

interval specifies the name of the basic interval type. For example, YEAR specifies yearly intervals.

multiple specifies an optional multiplier that sets the interval equal to a multiple of the period of the basic interval type. For example, the interval YEAR2 consists of two-year, or biennial, periods.

See “Incrementing Dates and Times By Using Multipliers and By Shifting Intervals” on page 31 for more information.

shift-index specifies an optional shift index that shifts the interval to start at a specified subperiod starting point. For example, YEAR.3 specifies yearly periods shifted to start on the first of March of each calendar year and to end in February of the following year.

Restrictions The shift index cannot be greater than the number of subperiods in the whole interval. For example, you could use YEAR2.24, but
YEAR2.25 would be an error because there is no 25th month in a two-year interval.

If the default shift period is the same as the interval type, then only multiperiod intervals can be shifted with the optional shift index. For example, because MONTH type intervals shift by MONTH subperiods by default, monthly intervals cannot be shifted with the shift index. However, bimonthly intervals can be shifted with the shift index, because there are two MONTH intervals in each MONTH2 interval. For example, the interval name MONTH2.2 specifies bimonthly periods starting on the first day of even-numbered months.

**Details**

The INTSHIFT function returns the shift interval that corresponds to the base interval. For custom intervals, the value that is returned is the base custom interval name. INTSHIFT ignores multiples of the interval and interval shifts.

The INTSHIFT function can also be used with calendar intervals from the retail industry. These intervals are ISO 8601 compliant.

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>shift1=intshift('year');</td>
<td>MONTH</td>
</tr>
<tr>
<td>put shift1;</td>
<td></td>
</tr>
<tr>
<td>shift2=intshift('dtyear');</td>
<td>DTMONTH</td>
</tr>
<tr>
<td>put shift2;</td>
<td></td>
</tr>
<tr>
<td>shift3=intshift('minute');</td>
<td>DTMINUTE</td>
</tr>
<tr>
<td>put shift3;</td>
<td></td>
</tr>
<tr>
<td>interval='weekdays';</td>
<td>WEEKDAY</td>
</tr>
<tr>
<td>shift4 = intshift(interval);</td>
<td></td>
</tr>
<tr>
<td>put shift4;</td>
<td></td>
</tr>
<tr>
<td>shift5=intshift('weekday5.4');</td>
<td>WEEKDAY</td>
</tr>
<tr>
<td>put shift5;</td>
<td></td>
</tr>
<tr>
<td>shift6=intshift('qtr');</td>
<td>MONTH</td>
</tr>
<tr>
<td>put shift6;</td>
<td></td>
</tr>
<tr>
<td>shift7=intshift('dttenday');</td>
<td>DTTENDAY</td>
</tr>
<tr>
<td>put shift7;</td>
<td></td>
</tr>
</tbody>
</table>
INTTEST Function

Returns 1 if a time interval is valid, and returns 0 if a time interval is invalid.

**Category:** Date and Time

---

**Syntax**

\[ \text{INTTEST}(\text{interval} \ <\ <\ \text{multiple},<\text{shift-index}>>>) \]

**Required Argument**

**interval**

specifies a character constant, variable, or expression that contains an interval name, such as WEEK, MONTH, or QTR. *interval* can appear in uppercase or lowercase.

Multipliers and shift indexes can be used with the basic interval names to construct more complex interval specifications. The general form of an interval name is as follows:

\[ \text{interval}<\text{multiple.shift-index}> \]

Here are the three parts of the interval name:

**interval**

specifies the name of the basic interval type. For example, YEAR specifies yearly intervals.

**multiple**

specifies an optional multiplier that sets the interval equal to a multiple of the period of the basic interval type. For example, YEAR2 consists of two-year, or biennial, periods.

See “Incrementing Dates and Times By Using Multipliers and By Shifting Intervals” on page 31 for more information.

**shift-index**

specifies an optional shift index that shifts the interval to start at a specified subperiod starting point. For example, YEAR.3 specifies yearly periods that are shifted to start on the first of March of each calendar year and to end in February of the following year.

**Restrictions**

The shift index cannot be greater than the number of subperiods in the whole interval. For example, you could use YEAR2.24, but YEAR2.25 is invalid because there is no 25th month in a two-year interval.

If the default shift period is the same as the interval type, then only multiperiod intervals can be shifted with the optional shift index. For example, because MONTH type intervals shift by MONTH subperiods by default, monthly intervals cannot be shifted with the shift index. However, bimonthly intervals can be shifted with the shift index, because there are two MONTH intervals in each MONTH2 interval. For example, the interval name MONTH2.2 specifies bimonthly periods starting on the first day of even-numbered months.
See “Incrementing Dates and Times By Using Multipliers and By Shifting Intervals” on page 31 for more information.

Details

The INTTEST function checks for a valid interval name. This function is useful when checking for valid values of multiple and shift-index.

The INTTEST function can also be used with calendar intervals from the retail industry. These intervals are ISO 8601 compliant.

Example

In the following examples, SAS returns a value of 1 if the interval argument is valid, and 0 if the interval argument is invalid.

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>test1=inttest('month');</td>
<td>1</td>
</tr>
<tr>
<td>put test1;</td>
<td></td>
</tr>
<tr>
<td>test2=inttest('week6.13');</td>
<td>1</td>
</tr>
<tr>
<td>put test2;</td>
<td></td>
</tr>
<tr>
<td>test3=inttest('tenday');</td>
<td>1</td>
</tr>
<tr>
<td>put test3;</td>
<td></td>
</tr>
<tr>
<td>test4=inttest('twoweeks');</td>
<td>0</td>
</tr>
<tr>
<td>put test4;</td>
<td></td>
</tr>
<tr>
<td>var1='hour2.2';</td>
<td></td>
</tr>
<tr>
<td>test5=inttest(var1);</td>
<td>1</td>
</tr>
<tr>
<td>put test5;</td>
<td></td>
</tr>
</tbody>
</table>

INTZ Function

Returns the integer portion of the argument, using zero fuzzing.

Category: Truncation

Syntax

INTZ(argument)

Required Argument

argument

is a numeric constant, variable, or expression.
Details

The following rules apply:

- If the value of the argument is an exact integer, INTZ returns that integer.
- If the argument is positive and not an integer, INTZ returns the largest integer that is less than the argument.
- If the argument is negative and not an integer, INTZ returns the smallest integer that is greater than the argument.

Comparisons

Unlike the INT function, the INTZ function uses zero fuzzing. If the argument is within 1E-12 of an integer, the INT function fuzzes the result to be equal to that integer. The INTZ function does not fuzz the result. Therefore, with the INTZ function you might get unexpected results.

Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>var1=2.1;</td>
<td></td>
</tr>
<tr>
<td>a=intz(var1);</td>
<td></td>
</tr>
<tr>
<td>put a;</td>
<td>2</td>
</tr>
<tr>
<td>var2=-2.4;</td>
<td></td>
</tr>
<tr>
<td>b=intz(var2);</td>
<td></td>
</tr>
<tr>
<td>put b;</td>
<td>-2</td>
</tr>
<tr>
<td>var3=1+1.e-11;</td>
<td></td>
</tr>
<tr>
<td>c=intz(var3);</td>
<td></td>
</tr>
<tr>
<td>put c;</td>
<td>1</td>
</tr>
<tr>
<td>f=intz(-1.6);</td>
<td></td>
</tr>
<tr>
<td>put f;</td>
<td>-1</td>
</tr>
</tbody>
</table>

See Also

Functions:

- “CEIL Function” on page 171
- “CEILZ Function” on page 172
- “FLOOR Function” on page 301
- “FLOORZ Function” on page 302
- “INT Function” on page 340
- “ROUND Function” on page 573
- “ROUNDDZ Function” on page 582
IPMT Function

Returns the interest payment for a given period for a constant payment loan or the periodic savings for a future balance.

Category: Financial

Syntax

IPMT(rate, period, number-of-periods, principal-amount, <future-amount>, <type>)

Required Arguments

rate
specifies the interest rate per payment period.

period
specifies the payment period for which the interest payment is computed.

Requirement Period must be a positive integer value that is less than or equal to the value of number-of-periods.

number-of-periods
specifies the number of payment periods.

Requirement Number-of-periods must be a positive integer value.

principal-amount
specifies the principal amount of the loan. Zero is assumed if a missing value is specified.

Optional Arguments

future-amount
specifies the future amount. Future-amount can be the outstanding balance of a loan after the specified number of payment periods, or the future balance of periodic savings. Zero is assumed if future-amount is omitted or if a missing value is specified.

type
specifies whether the payments occur at the beginning or end of a period. 0 represents the end-of-period payments, and 1 represents the beginning-of-period payments. 0 is assumed if type is omitted or if a missing value is specified.

Example

The interest payment on the first periodic payment of an $8,000 loan, where the nominal annual interest rate is 10% and the end-of-period monthly payments are 36, is computed as follows:

InterestPaid1=IPMT(0.1/12, 1, 36, 8000);

This computation returns a value of 66.66666667.

If the same loan has beginning-of-period payments, then the interest payment can be computed as follows:
• InterestPaid2=IPMT(0.1/12, 1, 36, 8000, 0, 1);
  This computation returns a value of 0.
• InterestPaid3=IPMT(0.1, 3, 3, 8000);
  This computation returns a value of 292.44712991.
• InterestPaid4=IPMT(0.09/12, 359, 360, 125000, 0, 1);
  This computation returns a value of 14.807573663.

### IQR Function

**Returns the interquartile range.**

**Category:** Descriptive Statistics

**Syntax**

\[ \text{IQR}(\text{value}-1 <, \text{value}-2...>) \]

**Required Argument**

\[ \text{value} \]

specifies a numeric constant, variable, or expression for which the interquartile range is to be computed.

**Details**

If all arguments have missing values, the result is a missing value. Otherwise, the result is the interquartile range of the nonmissing values. The formula for the interquartile range is the same as the one that is used in the UNIVARIATE procedure. For more information, see Base SAS Procedures Guide: Statistical Procedures.

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>iqr=iqr(2, 4, 1, 3, 999999);</td>
<td>2</td>
</tr>
<tr>
<td>put iqr;</td>
<td></td>
</tr>
</tbody>
</table>

**See Also**

**Functions:**

- “MAD Function” on page 424
- “PCTL Function” on page 476
IRR Function

Returns the internal rate of return as a percentage.

**Category:** Financial

**Syntax**

\[
\text{IRR}(\text{frequency} , c1, c2<, \ldots , cn>)
\]

**Required Arguments**

- \(\text{frequency}\) is numeric, the number of payments over a specified base period of time that is associated with the desired internal rate of return.
  - **Range** \(\text{frequency} > 0\).
  - **Tip** The case \(\text{frequency} = 0\) is a flag to allow continuous compounding.

- \(c1, c2, \ldots, cn\) are numeric, the optional cash payments.
  - **Requirement** A minimum of two cash payment values are required.

**Details**

The IRR function returns the internal rate of return over a specified base period of time for the set of cash payments \(c1, \ldots, cn\). The time intervals between any two consecutive payments are assumed to be equal. The argument \(\text{frequency} > 0\) describes the number of payments that occur over the specified base period of time. The number of notes issued from each instance is limited.

**Comparisons**

The IRR function is identical to INTRR, except that in the IRR function, the internal rate of return is a percentage.

**Example**

For an initial outlay of $400 and the expected payments of $100, $200, and $300 over the following three years, the annual internal rate of return as a percentage can be expressed as

\[
\text{rate=IRR}(1, -400, 100, 200, 300);
\]

The value that is returned is 19.437709963.

**See Also**

**Functions:**

- “INTRR Function” on page 377
JBESSEL Function

Returns the value of the Bessel function.

Category: Mathematical

Syntax

\[ \text{JBESSEL}(\nu, x) \]

Required Arguments

\( \nu \)

specifies a numeric constant, variable, or expression.

Range \( \nu \geq 0 \)

\( x \)

specifies a numeric constant, variable, or expression.

Range \( x \geq 0 \)

Details

The JBESSEL function returns the value of the Bessel function of order \( \nu \) evaluated at \( x \) (For more information, see Abramowitz and Stegun 1964; Amos, Daniel, and Weston 1977).

Example

The following SAS statement produces this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{x=jbessel(2, 2)};</td>
<td>0.3528340286</td>
</tr>
</tbody>
</table>

JULDATE Function

Returns the Julian date from a SAS date value.

Category: Date and Time

Syntax

\[ \text{JULDATE}(\text{date}) \]
**Required Argument**

date
    specifies a SAS date value.

**Details**

A SAS date value is a number that represents the number of days from January 1, 1960 to a specific date. The JULDATE function converts a SAS date value to a Julian date. If date falls within the 100-year span defined by the system option YEARCUTOFF=, the result has three, four, or five digits. In a five-digit result, the first two digits represent the year, and the next three digits represent the day of the year (1 to 365, or 1 to 366 for leap years). Because leading zeros are dropped from the result, the year portion of a Julian date might be omitted (for years ending in 00), or it might have only one digit (for years ending 01–09). Otherwise, the result has seven digits: the first four digits represent the year, and the next three digits represent the day of the year. For example, if YEARCUTOFF=1926, JULDATE would return 97001 for January 1, 1997, and return 1878365 for December 31, 1878.

**Comparisons**

The function JULDATE7 is similar to JULDATE except that JULDATE7 always returns a four-digit year. Thus, JULDATE7 is year 2000 compliant because it eliminates the need to consider the implications of a two-digit year.

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>julian=juldate('31dec99'd);</td>
<td>99365</td>
</tr>
<tr>
<td>julian=juldate('01jan2099'd);</td>
<td>2099001</td>
</tr>
</tbody>
</table>

**See Also**

Functions:

- “DATEJUL Function” on page 221
- “JULDATE7 Function” on page 391

**JULDATE7 Function**

Returns a seven-digit Julian date from a SAS date value.

**Category:** Date and Time

**Syntax**

JULDATE7(date)
**Required Argument**

*date*

specifies a SAS date value.

**Details**

The JULDATE7 function returns a seven-digit Julian date from a SAS date value. The first four digits represent the year, and the next three digits represent the day of the year.

**Comparisons**

The function JULDATE7 is similar to JULDATE except that JULDATE7 always returns a four-digit year. Thus, JULDATE7 is year 2000 compliant because it eliminates the need to consider the implications of a two-digit year.

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>julian=juldate7('31dec96'd);</td>
<td>1996366</td>
</tr>
<tr>
<td>julian=juldate7('01jan2099'd);</td>
<td>2099001</td>
</tr>
</tbody>
</table>

**See Also**

Functions:

- “JULDATE Function” on page 390

---

**KURTOSIS Function**

Returns the kurtosis.

**Category:** Descriptive Statistics

**Syntax**

\[
\text{KURTOSIS} (\text{argument-1}, \text{argument-2}, \text{argument-3}, \text{argument-4}, ..., \text{argument-n})
\]

**Required Argument**

*argument*

specifies a numeric constant, variable, or expression.

**Details**

At least four nonmissing arguments are required. Otherwise, the function returns a missing value. If all nonmissing arguments have equal values, the kurtosis is
mathematically undefined. The KURTOSIS function returns a missing value and sets _ERROR_ equal to 1.

The argument list can consist of a variable list, which is preceded by OF.

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1=kurtosis(5, 9, 3, 6);</td>
<td>0.928</td>
</tr>
<tr>
<td>x2=kurtosis(5, 8, 9, 6, .);</td>
<td>-3.3</td>
</tr>
<tr>
<td>x3=kurtosis(8, 9, 6, 1);</td>
<td>1.5</td>
</tr>
<tr>
<td>x4=kurtosis(8, 1, 6, 1);</td>
<td>-4.483379501</td>
</tr>
<tr>
<td>x5=kurtosis(of x1-x4);</td>
<td>-5.065692754</td>
</tr>
</tbody>
</table>

**LAG Function**

Returns values from a queue.

**Category:** Special

**Note:** The VARCHAR type is not supported for arguments in the LAG function.

**Syntax**

LAG <n> (argument)

**Required Argument**

argument

specifies a numeric or character constant, variable, or expression.

**Optional Argument**

n

specifies the number of lagged values.

**Details**

**The Basics**

If the LAG function returns a value to a character variable that has not yet been assigned a length, by default the variable is assigned the same length as the variable used in the argument.

The LAG functions, LAG1, LAG2, ..., LAGn return values from a queue. LAG1 can also be written as LAG. A LAGn function stores a value in a queue and returns a value
stored previously in that queue. Each occurrence of a LAG\textsubscript{n} function in a program generates its own queue of values.

The queue for each occurrence of LAG\textsubscript{n} is initialized with \( n \) missing values, where \( n \) is the length of the queue (for example, a LAG2 queue is initialized with two missing values). When an occurrence of LAG\textsubscript{n} is executed, the value at the top of its queue is removed and returned, the remaining values are shifted upward, and the new value of the argument is placed at the bottom of the queue. Hence, missing values are returned for the first \( n \) executions of each occurrence of LAG\textsubscript{n}, after which the lagged values of the argument begin to appear.

Note: Storing values at the bottom of the queue and returning values from the top of the queue occurs only when the function is executed. An occurrence of the LAG\textsubscript{n} function that is executed conditionally stores and returns values only from the observations for which the condition is satisfied.

If the argument of LAG\textsubscript{n} is an array name, a separate queue is maintained for each variable in the array.

Some language elements are supported in SAS but not in CAS. An example is the DATALINES statement. Some examples in this document use DATALINES in SAS to create a data set using the CAS engine. For more information, see “Functions in SAS and CAS” on page 54.

**Memory Limit for the LAG Function**

When the LAG function is compiled, SAS allocates memory in a queue to hold the values of the variable that is listed in the LAG function. For example, if the variable in function LAG100(x) is numeric with a length of 8 bytes, then the memory that is needed is 8 times 100, or 800 bytes. Therefore, the memory limit for the LAG function is based on the memory that SAS allocates, which varies with different operating environments.

**Examples**

**Example 1: Generating Two Lagged Values**

The following program generates two lagged values for each observation.

```sas
data one;
  input x @@;
  y=lag1(x);
  z=lag2(x);
datalines;
1 2 3 4 5 6
;
proc print data=one;
  title 'LAG Output';
run;
```
LAG1 returns one missing value and the values of X (lagged once). LAG2 returns two missing values and the values of X (lagged twice).

**Example 2: Generating Multiple Lagged Values in BY Groups**

The following example shows how to generate up to three lagged values within each BY group.

```plaintext
/**************************************************************************/
/* This program generates up to three lagged values. By increasing the     */
/* size of the array and the number of assignment statements that use      */
/* the LAGn functions, you can generate as many lagged values as needed.  */
/**************************************************************************/
/* Create starting data. */
data old;
  input start end;
datalines;
1 1
1 2
1 3
1 4
1 5
1 6
1 7
2 1
2 2
3 1
3 2
3 3
3 4
3 5;
data new(drop=i count);
  set old;
  by start;
  /* Create and assign values to three new variables. Use ENDLAG1-     */
  /* ENDLAG3 to store lagged values of END, from the most recent to the  */
  /* third preceding value. */
  array x(*) endlag1-endlag3;
  endlag1=lag1(end);
  endlag2=lag2(end);
  endlag3=lag3(end);
```

**Figure 2.21  Output from Generating Two Lagged Values**

| Obs | x  | y  | z
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>.</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>
/* Reset COUNT at the start of each new BY-Group */
if first.start then count=1;
/* On each iteration, set to missing array elements */
/* that have not yet received a lagged value for the */
/* current BY-Group. Increase count by 1. */
do i=count to dim(x);
   x(i)=.;
   count + 1;
run;
proc print;
run;

Figure 2.22  Output from Generating Three Lagged Values

The SAS System

<table>
<thead>
<tr>
<th>Obs</th>
<th>start</th>
<th>end</th>
<th>endlag1</th>
<th>endlag2</th>
<th>endlag3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>.</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>1</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>1</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>.</td>
</tr>
<tr>
<td>13</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

Example 3: Computing the Moving Average of a Variable through the Entire Data Set
The following example computes the moving average of a variable through the entire data set.

data x;
do x=1 to 10;
   output;
   end;
run;
/* Compute the moving average of the entire data set. */
data avg;
retain s 0;
set x;
s=s+x;
a=s/_n_;
Example 4: Computing the Moving Average of a Variable of the Last n Observations

The following example computes the moving average of a variable of the last n observations.

```sas
data x;
  do x=1 to 10;
    output;
  end;
run;
%let n=5;
data avg (drop=s);
  retain s;
  set x;
  s=sum (s, x, -lag&n(x)) ;
  a=s / min(_n_, &n);
run;
proc print;
run;
```
Example 5: Computing the Moving Average of a Variable of the Last n Observations within a BY Group

The following example computes the moving average of a variable of the last n observations within a BY group.

```sas
data x;
  do x=1 to 10;
    output;
  end;
run;
data ds1;
  do patient='A','B','C';
    do month=1 to 7;
      num=int(ranuni(0)*10);
      output;
    end;
  end;
run;
proc sort;
  by patient;
%let n = 4;
data ds2;
  set ds1;
  by patient;
  retain num_sum 0;
  if first.patient then do;
    count=0;
    num_sum=0;
  end;
  count+1;
  last&n=lag&n(num);
  if count gt &n then num_sum=sum(num_sum, num, -last&n);
  else num_sum=sum(num_sum, num);
  if count ge &n then mov_aver=num_sum/&n;
  else mov_aver=.;
run;
proc print;
```
Example 6: Generating a Fibonacci Sequence of Numbers

The following example generates a Fibonacci sequence of numbers. You start with 0 and 1, and then add the two previous Fibonacci numbers to generate the next Fibonacci number.

data _null_;  
put 'Fibonacci Sequence';  
n=1;  
f=1;  
put n= f=;  
do n=2 to 10;  
f=sum(f, lag(f));  
put n= f=;  
end;  
run;
SAS writes the following output to the log:

<table>
<thead>
<tr>
<th>n</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>13</td>
</tr>
<tr>
<td>8</td>
<td>21</td>
</tr>
<tr>
<td>9</td>
<td>34</td>
</tr>
<tr>
<td>10</td>
<td>55</td>
</tr>
</tbody>
</table>

**Example 7: Using Expressions for the LAG Function Argument**

The following program uses an expression for the value of argument and creates a data set that contains the values for X, Y, and Z. LAG dequeues the previous values of the expression and enqueues the current value.

```sas
data one;
  input X @@;
  Y=lag1(x+10);
  Z=lag2(x);
datalines;
1 2 3 4 5 6
;
proc print;
  title 'Lag Output: Using an Expression';
run;
```

**Figure 2.24  Output from the LAG Function: Using an Expression**

<table>
<thead>
<tr>
<th>Obs</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>14</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>15</td>
<td>4</td>
</tr>
</tbody>
</table>

**See Also**

**Functions:**
- “DIF Function” on page 231

**LARGEST Function**

Returns the kth largest nonmissing value.
**Syntax**

\[ \text{LARGEST}(k, \text{value-1} <, \text{value-2} ...>) \]

**Required Arguments**

\( k \)

is a numeric constant, variable, or expression that specifies which value to return.

\( \text{value} \)

specifies the value of a numeric constant, variable, or expression to be processed.

**Details**

If \( k \) is missing, less than zero, or greater than the number of values, the result is a missing value and \_ERROR\_ is set to 1. Otherwise, if \( k \) is greater than the number of nonmissing values, the result is a missing value but \_ERROR\_ is not set to 1.

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>( k=1; )</td>
<td></td>
</tr>
<tr>
<td>largest1=largest{k, 456, 789, .Q, 123};</td>
<td></td>
</tr>
<tr>
<td>put largest1;</td>
<td>789</td>
</tr>
<tr>
<td>( k=2; )</td>
<td></td>
</tr>
<tr>
<td>largest2=largest{k, 456, 789, .Q, 123};</td>
<td></td>
</tr>
<tr>
<td>put largest2;</td>
<td>456</td>
</tr>
<tr>
<td>( k=3; )</td>
<td></td>
</tr>
<tr>
<td>largest3=largest{k, 456, 789, .Q, 123};</td>
<td></td>
</tr>
<tr>
<td>put largest3;</td>
<td>123</td>
</tr>
<tr>
<td>( k=4; )</td>
<td></td>
</tr>
<tr>
<td>largest4=largest{k, 456, 789, .Q, 123};</td>
<td></td>
</tr>
<tr>
<td>put largest4;</td>
<td>.</td>
</tr>
</tbody>
</table>

**See Also**

**Functions:**

- “ORDINAL Function” on page 475
- “PCTL Function” on page 476
- “SMALLEST Function” on page 609
**LBOUND Function**

Returns the lower bound of an array.

**Category:** Array

**Syntax**

\[
\text{LBOUND} \ <n>\ (\text{array-name}) \\
\text{LBOUND}(\text{array-name, bound-n})
\]

**Required Arguments**

- **array-name**
  - is the name of an array that was defined previously in the same DATA step.

- **bound-n**
  - is a numeric constant, variable, or expression that specifies the dimension for which you want to know the lower bound. Use `bound-n` only if `n` is not specified.

**Optional Argument**

- **n**
  - is an integer constant that specifies the dimension for which you want to know the lower bound. If no `n` value is specified, the LBOUND function returns the lower bound of the first dimension of the array.

**Details**

The LBOUND function returns the lower bound of a one-dimensional array or the lower bound of a specified dimension of a multidimensional array. Use LBOUND in array processing to avoid changing the lower bound of an iterative DO group each time you change the bounds of the array. LBOUND and HBOUND can be used together to return the values of the lower and upper bounds of an array dimension.

**Examples**

**Example 1: One-Dimensional Array**

In this example, LBOUND returns the lower bound of the dimension, a value of 2. SAS repeats the statements in the DO loop five times.

```sas
array big{2:6} weight sex height state city;
do i=1bound(big) to hbound(big);
  ...more SAS statements...
end;
```

**Example 2: Multidimensional Array**

This example shows two ways of specifying the LBOUND function for multidimensional arrays. Both methods return the same value for LBOUND, as shown in the table that follows the SAS code example.

```sas
array mult{2:6, 4:13, 2} multi-mult100;
```
LCM Function

Returns the least common multiple.

**Syntax**

\[ \text{LCM}(x_1, x_2, x_3, \ldots, x_n) \]

**Required Argument**

\( x \)

specifies a numeric constant, variable, or expression that has an integer value.

**Details**

The LCM (least common multiple) function returns the smallest multiple that is exactly divisible by every member of a set of numbers. For example, the least common multiple of 12 and 18 is 36.

If any of the arguments are missing, then the returned value is a missing value.

**Example**

The following example returns the smallest multiple that is exactly divisible by the integers 10 and 15.

```sas
data _null_
  x=lcm(10, 15)
  put x=
```

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Alternative Syntax</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LBOUND(MULT)</td>
<td>LBOUND(MULT,1)</td>
<td>2</td>
</tr>
<tr>
<td>LBOUND2(MULT)</td>
<td>LBOUND(MULT,2)</td>
<td>4</td>
</tr>
<tr>
<td>LBOUND3(MULT)</td>
<td>LBOUND(MULT,3)</td>
<td>1</td>
</tr>
</tbody>
</table>

**See Also**

- “DIM Function” on page 234
- “HBOUND Function” on page 320

**LCM Function**

Returns the least common multiple.

**Category:** Mathematical

### Syntax

\[ \text{LCM}(x_1, x_2, x_3, \ldots, x_n) \]

### Required Argument

\( x \)

specifies a numeric constant, variable, or expression that has an integer value.

### Details

The LCM (least common multiple) function returns the smallest multiple that is exactly divisible by every member of a set of numbers. For example, the least common multiple of 12 and 18 is 36.

If any of the arguments are missing, then the returned value is a missing value.

### Example

The following example returns the smallest multiple that is exactly divisible by the integers 10 and 15.

```sas
data _null_
  x=lcm(10, 15)
  put x=
```
run;
SAS writes the following output to the log:

x=10

See Also

Functions:
• “GCD Function” on page 311

LCOMB Function

Computes the logarithm of the COMB function, which is the logarithm of the number of combinations of \( n \) objects taken \( r \) at a time.

**Category:** Combinatorial

**Syntax**

\[
\text{LCOMB}(n, r)
\]

**Required Arguments**

\( n \)

is a nonnegative integer that represents the total number of elements from which the sample is chosen.

\( r \)

is a nonnegative integer that represents the number of chosen elements.

**Restriction** \( r \leq n \)

**Comparisons**

The LCOMB function computes the logarithm of the COMB function.

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x = \text{lcomb}(5000, 500); )</td>
<td>1621.4411361</td>
</tr>
<tr>
<td>put ( x; )</td>
<td></td>
</tr>
<tr>
<td>( y = \text{lcomb}(100, 10); )</td>
<td>30.482323362</td>
</tr>
<tr>
<td>put ( y; )</td>
<td></td>
</tr>
</tbody>
</table>
See Also

Functions:

- “COMB Function” on page 183

LEFT Function

Left-aligns a character string.

Category: Character

Restriction: This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility.

Tip: DBCS equivalent function is KLEFT. See “DBCS Compatibility” on page 405.

Syntax

\[ \text{LEFT}(\text{argument}) \]

Required Argument

\( \text{argument} \)

specifies a character constant, variable, or expression.

Details

The Basics

In a DATA step, if the LEFT function returns a value to a variable that has not previously been assigned a length, then that variable is given the length of the argument.

LEFT returns an argument with leading blanks moved to the end of the value. The argument's length does not change.

DBCS Compatibility

The LEFT function left-aligns a character string. You can use the LEFT function in most cases. If an application can be executed in an ASCII environment, or if the application does not manipulate character strings, then using the LEFT function rather than the KLEFT function.

Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SAS Statement | Result
--- | ---
a=' DUE DATE';
b=left(a);
put b; | DUE DATE

See Also

Functions:
- “COMPRESS Function” on page 191
- “RIGHT Function” on page 571
- “STRIP Function” on page 613
- “TRIM Function” on page 640

**LENGTH Function**

Returns the length of a non-blank character string, excluding trailing blanks, and returns 1 for a blank character string.

**Category:** Character

**Restriction:** This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility.

**Tips:** DBCS equivalent function is KLENGTH.
The LENGTH function returns a length in bytes. The KLENGTH function returns a length in a character-based unit.

**Syntax**

LENGTH(*string*)

*Required Argument*

*string*

specifies a character constant, variable, or expression.

**Details**

The LENGTH function returns an integer that represents the position of the rightmost non-blank character in *string*. If the value of *string* is blank, LENGTH returns a value of 1. If *string* is a numeric constant, variable, or expression (either initialized or uninitialized), SAS automatically converts the numeric value to a right-justified character string by using the BEST12. format. In this case, LENGTH returns a value of 12 and writes a note in the SAS log stating that the numeric values have been converted to character values.
**Comparisons**

- The `LENGTH` and `LENGTHN` functions return the same value for non-blank character strings. `LENGTH` returns a value of 1 for blank character strings, whereas `LENGTHN` returns a value of 0.

- The `LENGTH` function returns the length of a character string, excluding trailing blanks, whereas the `LENGTHC` function returns the length of a character string, including trailing blanks.

- The `LENGTH` function returns the length of a character string, excluding trailing blanks, whereas the `LENGTHM` function returns the amount of memory in bytes that is allocated for a character string.

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>len=length('ABCDEF'); put len;</td>
<td>6</td>
</tr>
<tr>
<td>len2=length(' '); put len2;</td>
<td>1</td>
</tr>
</tbody>
</table>

**See Also**

Functions:

- “LENGTHC Function” on page 407
- “LENGTHM Function” on page 408
- “LENGTHN Function” on page 410

---

**LENGTHC Function**

Returns the length of a character string, including trailing blanks.

**Category:** Character

**Restriction:** This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see [Internationalization Compatibility](#).

**Syntax**

```
LENGTHC(string)
```

**Required Argument**

- `string` specifies a character constant, variable, or expression.
Details

The LENGTHC function returns the number of characters, both blanks and non-blanks, in string. If string is a numeric constant, variable or expression (either initialized or uninitialized), SAS automatically converts the numeric value to a right-justified character string by using the BEST12. format. In this case, LENGTHC returns a value of 12 and writes a note in the SAS log stating that the numeric values have been converted to character values.

Comparisons

• The LENGTHC function returns the length of a character string, including trailing blanks, whereas the LENGTH and LENGTHN functions return the length of a character string, excluding trailing blanks. LENGTHC always returns a value that is greater than or equal to the value of LENGTHN.

• The LENGTHC function returns the length of a character string, including trailing blanks, whereas the LENGTHM function returns the amount of memory in bytes that is allocated for a character string. For fixed-length character strings, LENGTHC and LENGTHM always return the same value. For varying-length character strings, LENGTHC always returns a value that is less than or equal to the value returned by LENGTHM.

Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=lengthc('variable with trailing blanks   '); put x;</td>
<td>32</td>
</tr>
<tr>
<td>length fixed $35;</td>
<td></td>
</tr>
<tr>
<td>fixed='variable with trailing blanks   '; x=lengthc(fixed); put x;</td>
<td>35</td>
</tr>
</tbody>
</table>

See Also

Functions:

• “LENGTH Function” on page 406
• “LENGTHM Function” on page 408
• “LENGTHN Function” on page 410

LENGTHM Function

Returns the amount of memory (in bytes) that is allocated for a character string.

Category: Character
Restriction: This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility.

Syntax

`LENGTHM(string)`

Required Argument

`string`

specifies a character constant, variable, or expression.

Details

The `LENGTHM` function returns an integer that represents the amount of memory in bytes that is allocated for `string`. If `string` is a numeric constant, variable, or expression (either initialized or uninitialized), SAS automatically converts the numeric value to a right-justified character string by using the BEST12. format. In this case, `LENGTHM` returns a value of 12 and writes a note in the SAS log stating that the numeric values have been converted to character values.

Comparisons

The `LENGTHM` function returns the amount of memory in bytes that is allocated for a character string, whereas the `LENGTH`, `LENGTHC`, and `LENGTHN` functions return the length of a character string. `LENGTHM` always returns a value that is greater than or equal to the values that are returned by `LENGTH`, `LENGTHC`, and `LENGTHN`.

Examples

**Example 1: Determining the Amount of Allocated Memory for a Character Expression**

This example determines the amount of memory (in bytes) that is allocated for a buffer that stores intermediate results in a character expression. Because SAS does not know how long the value of the expression `CAT(x, y)` is, SAS allocates memory for values up to 32,767 bytes long.

```sas
data _null_;
x='x';
y='y';
lc=lengthc(cat(x, y));
lm=lengthm(cat(x, y));
put lc= lm=;
run;
```

SAS writes the following output to the log:

```
lc=2 lm=32767
```

**Example 2: Determining the Amount of Allocated Memory for a Variable from an External File**

This example determines the amount of memory (in bytes) that is allocated to a variable that is input into a SAS file from an external file.
data _null_;  
file 'test.txt';  
put 'trailing blanks   ';  
run;  
data test;  
infile 'test.txt';  
input;  
x=lengthm(_infile_);  
put x;  
run;  

SAS writes the following output to the log:

256

See Also

Functions:
• “LENGTH Function” on page 406  
• “LENGTHC Function” on page 407  
• “LENGTHN Function” on page 410

LENGTHN Function

Returns the length of a character string, excluding trailing blanks.

Category: Character

Restriction: This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility.

Syntax

LENGTHN(string)

Required Argument

string

specifies a character constant, variable, or expression.

Details

The LENGTHN function returns an integer that represents the position of the rightmost non-blank character in string. If the value of string is blank, LENGTHN returns a value of 0. If string is a numeric constant, variable, or expression (either initialized or uninitialized), SAS automatically converts the numeric value to a right-justified character string by using the BEST12. format. In this case, LENGTHN returns a value of 12 and writes a note in the SAS log stating that the numeric values have been converted to character values.
Comparisons

- The LENGTHN and LENGTH functions return the same value for non-blank character strings. LENGTH returns a value of 0 for blank character strings, whereas LENGTHN returns a value of 1.

- The LENGTHN function returns the length of a character string, excluding trailing blanks, whereas the LENGTHC function returns the length of a character string, including trailing blanks. LENGTHN always returns a value that is less than or equal to the value returned by LENGTHC.

- The LENGTHN function returns the length of a character string, excluding trailing blanks, whereas the LENGTHM function returns the amount of memory in bytes that is allocated for a character string. LENGTHN always returns a value that is less than or equal to the value returned by LENGTHM.

Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>len=lengthn('ABCDEF'); put len;</td>
<td>6</td>
</tr>
<tr>
<td>len2=lengthn(' '); put len2;</td>
<td>0</td>
</tr>
</tbody>
</table>

See Also

Functions:
- “LENGTH Function” on page 406
- “LENGTHC Function” on page 407
- “LENGTHM Function” on page 408

LFACT Function

Computes the logarithm of the FACT (factorial) function.

**Syntax**

\[
\text{LFACT}(n)
\]

**Required Argument**

\(n\) is an integer that represents the total number of elements from which the sample is chosen.
Details

The LFACT function computes the logarithm of the FACT function.

Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=lfact(5000);</td>
<td>37591.143509</td>
</tr>
<tr>
<td>put x;</td>
<td></td>
</tr>
<tr>
<td>y=lfact(100);</td>
<td>363.73937556</td>
</tr>
<tr>
<td>put y;</td>
<td></td>
</tr>
</tbody>
</table>

See Also

Functions:

- “FACT Function” on page 242

LGAMMA Function

Returns the natural logarithm of the Gamma function.

Category: Mathematical

Syntax

`LGAMMA(argument)`

Required Argument

- **argument**:
  - specifies a numeric constant, variable, or expression.
  - **Range** must be positive.

Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=lgamma(2);</td>
<td>0</td>
</tr>
<tr>
<td>x=lgamma(1.5);</td>
<td>-0.120782238</td>
</tr>
</tbody>
</table>
LOG Function

Returns the natural (base e) logarithm.

**Category:** Mathematical

**Syntax**

\[ \text{LOG}(\text{argument}) \]

**Required Argument**

- \( \text{argument} \) specifies a numeric constant, variable, or expression.

  - **Range:** must be positive.

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x=\text{log}(1.0) );</td>
<td>0</td>
</tr>
<tr>
<td>( x=\text{log}(10.0) );</td>
<td>2.302585093</td>
</tr>
</tbody>
</table>

LOG1PX Function

Returns the log of 1 plus the argument.

**Category:** Mathematical

**Syntax**

\[ \text{LOG1PX}(x) \]

**Required Argument**

- \( x \) specifies a numeric variable, constant, or expression.

**Details**

The LOG1PX function computes the log of 1 plus the argument. The LOG1PX function is mathematically defined by the following equation, where \(-1 < x\):

\[ \text{LOG1PX}(x) = \log(1 + x) \]
When $x$ is close to 0, $\text{LOG1PX}(x)$ can be more accurate than $\text{LOG}(1+x)$.

**Examples**

**Example 1: Computing the Log with the LOG1PX Function**
The following example computes the log of 1 plus the value 0.5.

```sas
data _null_
  x=log1px(0.5);
  put x=;
run;
```

SAS writes the following output to the log:

```
x=0.4054651081
```

**Example 2: Comparing the LOG1PX Function with the LOG Function**
In the following example, the value of X is computed by using the LOG1PX function. The value of Y is computed by using the LOG function.

```sas
data _null_
  x=log1px(1.e-5);
  put x= hex16.;
  y=log(1+1.e-5);
  put y= hex16.;
run;
```

SAS writes the following output to the log:

```
x=3EE4F8AEA9AE7317
y=3EE4F8AEA9AF0A25
```

**See Also**

**Functions:**
- “LOG Function” on page 413

---

**LOG10 Function**

Returns the logarithm to the base 10.

**Category:** Mathematical

**Syntax**

$\text{LOG10}(\text{argument})$

**Required Argument**

$\text{argument}$

specifies a numeric constant, variable, or expression.
Range must be positive.

Example
The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=log10(1.0);</td>
<td>0</td>
</tr>
<tr>
<td>x=log10(10.0);</td>
<td>1</td>
</tr>
<tr>
<td>x=log10(100.0);</td>
<td>2</td>
</tr>
</tbody>
</table>

LOG2 Function
Returns the logarithm to the base 2.

Category: Mathematical

Syntax
LOG2(argument)

Required Argument
argument
specifies a numeric constant, variable, or expression.

Range must be positive.

Example
The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=log2(2.0);</td>
<td>1</td>
</tr>
<tr>
<td>x=log2(0.5);</td>
<td>-1</td>
</tr>
</tbody>
</table>

LOGBETA Function
Returns the logarithm of the beta function.

Category: Mathematical
Syntax

**LOGBETA**($a$, $b$

**Required Arguments**

$a$

is the first shape parameter, where $a > 0$.

$b$

is the second shape parameter, where $b > 0$.

**Details**

The **LOGBETA** function is mathematically given by the equation

$$
\log(\beta(a, b)) = \log(\Gamma(a)) + \log(\Gamma(b)) - \log(\Gamma(a + b))
$$

where $\Gamma(.)$ is the gamma function.

If the expression cannot be computed, **LOGBETA** returns a missing value.

**Example**

In the following example, the first shape parameter has a value of 5 and the second shape parameter has a value of 3.

The following SAS statement produces this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LOGBETA</strong>(5, 3);</td>
<td>-4.653960350</td>
</tr>
</tbody>
</table>

**See Also**

Functions:

- “BETA Function” on page 111

---

**LOGCDF Function**

Returns the logarithm of a left cumulative distribution function.

**Category:** Probability

**See:** “CDF Function” on page 153

**Syntax**

**LOGCDF**(*distribution*, quantile <, parameter-1, ..., parameter-k>)
Required Arguments

'distribution'
is a character constant, variable, or expression that identifies the distribution. Valid distributions are as follows:

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bernoulli</td>
<td>'BERNOULLI'</td>
</tr>
<tr>
<td>Beta</td>
<td>'BETA'</td>
</tr>
<tr>
<td>Binomial</td>
<td>'BINOMIAL'</td>
</tr>
<tr>
<td>Cauchy</td>
<td>'CAUCHY'</td>
</tr>
<tr>
<td>Chi-Square</td>
<td>'CHISQUARE'</td>
</tr>
<tr>
<td>Conway-Maxwell-Poisson</td>
<td>'CONMAXPOI'</td>
</tr>
<tr>
<td>Exponential</td>
<td>'EXONENTIAL'</td>
</tr>
<tr>
<td>F</td>
<td>'F'</td>
</tr>
<tr>
<td>Gamma</td>
<td>'GAMMA'</td>
</tr>
<tr>
<td>Generalized Poisson</td>
<td>'GENPOISSON'</td>
</tr>
<tr>
<td>Geometric</td>
<td>'GEOMETRIC'</td>
</tr>
<tr>
<td>Hypergeometric</td>
<td>'HYPERGEOMETRIC'</td>
</tr>
<tr>
<td>Laplace</td>
<td>'LAPLACE'</td>
</tr>
<tr>
<td>Logistic</td>
<td>'LOGISTIC'</td>
</tr>
<tr>
<td>Lognormal</td>
<td>'LOGNORMAL'</td>
</tr>
<tr>
<td>Negative binomial</td>
<td>'NEGBINOMIAL'</td>
</tr>
<tr>
<td>Normal</td>
<td>'NORMAL'</td>
</tr>
<tr>
<td>Normal mixture</td>
<td>'NORMALMIX'</td>
</tr>
<tr>
<td>Pareto</td>
<td>'PARETO'</td>
</tr>
<tr>
<td>Poisson</td>
<td>'POISSON'</td>
</tr>
<tr>
<td>T</td>
<td>'T'</td>
</tr>
<tr>
<td>Tweedie</td>
<td>'TWEEDIE'</td>
</tr>
</tbody>
</table>
### Distribution and Argument

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform</td>
<td>'UNIFORM'</td>
</tr>
<tr>
<td>Wald (inverse Gaussian)</td>
<td>'WALD'</td>
</tr>
<tr>
<td>Weibull</td>
<td>'WEIBULL'</td>
</tr>
</tbody>
</table>

*Note:* Except for T, F, and NORMALMIX, you can minimally identify any distribution by its first four characters.

A *quantile* is a numeric variable, constant, or expression that specifies the value of a random variable.

#### Optional Argument

*parameter-1, ..., parameter-k* are optional *shape*, *location*, or *scale* parameters appropriate for the specific distribution.

### Details

The LOGCDF function computes the logarithm of a left cumulative distribution function (logarithm of the left side) from various continuous and discrete distributions. For more information, see “CDF Function” on page 153.

For more information about the distributions that are listed in the table, see “PDF Function” on page 477.

### See Also

**Functions:**

- “CDF Function” on page 153
- “LOGPDF Function” on page 418
- “LOGSDF Function” on page 420
- “PDF Function” on page 477
- “QUANTILE Function” on page 549
- “SDF Function” on page 596
- “SQUANTILE Function” on page 1062

### LOGPDF Function

**Returns the logarithm of a probability density (mass) function.**

**Category:** Probability

**Alias:** LOGPMF

**See:** “PDF Function” on page 477
Syntax

`LOGPDF('distribution', quantile, parameter-1, ..., parameter-k)`

**Required Arguments**

`'distribution'` is a character constant, variable, or expression that identifies the distribution. Valid distributions are as follows:

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bernoulli</td>
<td>'BERNOULLI'</td>
</tr>
<tr>
<td>Beta</td>
<td>'BETA'</td>
</tr>
<tr>
<td>Binomial</td>
<td>'BINOMIAL'</td>
</tr>
<tr>
<td>Cauchy</td>
<td>'CAUCHY'</td>
</tr>
<tr>
<td>Chi-Square</td>
<td>'CHISQUARE'</td>
</tr>
<tr>
<td>Conway-Maxwell-Poisson</td>
<td>'CONMAXPOI'</td>
</tr>
<tr>
<td>Exponential</td>
<td>'EXPONENTIAL'</td>
</tr>
<tr>
<td>F</td>
<td>'F'</td>
</tr>
<tr>
<td>Gamma</td>
<td>'GAMMA'</td>
</tr>
<tr>
<td>Generalized Poisson</td>
<td>'GENPOISSON'</td>
</tr>
<tr>
<td>Geometric</td>
<td>'GEOMETRIC'</td>
</tr>
<tr>
<td>Hypergeometric</td>
<td>'HYPERGEOMETRIC'</td>
</tr>
<tr>
<td>Laplace</td>
<td>'LAPLACE'</td>
</tr>
<tr>
<td>Logistic</td>
<td>'LOGISTIC'</td>
</tr>
<tr>
<td>Lognormal</td>
<td>'LOGNORMAL'</td>
</tr>
<tr>
<td>Negative binomial</td>
<td>'NEGBINOMIAL'</td>
</tr>
<tr>
<td>Normal</td>
<td>'NORMAL'</td>
</tr>
<tr>
<td>Normal mixture</td>
<td>'NORMALMIX'</td>
</tr>
<tr>
<td>Pareto</td>
<td>'PARETO'</td>
</tr>
<tr>
<td>Distribution</td>
<td>Argument</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>Poisson</td>
<td>'POISSON'</td>
</tr>
<tr>
<td>T</td>
<td>'T'</td>
</tr>
<tr>
<td>Tweedie</td>
<td>'TWEEDIE'</td>
</tr>
<tr>
<td>Uniform</td>
<td>'UNIFORM'</td>
</tr>
<tr>
<td>Wald (inverse Gaussian)</td>
<td>'WALD'</td>
</tr>
<tr>
<td>Weibull</td>
<td>'WEIBULL'</td>
</tr>
</tbody>
</table>

Note: Except for T, F, and NORMALMIX, you can minimally identify any distribution by its first four characters.

quantile
is a numeric constant, variable, or expression that specifies the value of a random variable.

parameter-1, ..., parameter-k
are optional shape, location, or scale parameters appropriate for the specific distribution.

Details
The LOGPDF function computes the logarithm of the probability density (mass) function from various continuous and discrete distributions. For more information, see “PDF Function” on page 477.

For more information about the distributions that are listed in the table, see “PDF Function” on page 477.

See Also

Functions:
- “CDF Function” on page 153
- “LOGCDF Function” on page 416
- “LOGSDF Function” on page 420
- “PDF Function” on page 477
- “QUANTILE Function” on page 549
- “SDF Function” on page 596
- “SQUANTILE Function” on page 1062

LOGSDF Function
Returns the logarithm of a survival function.
Category: Probability
See: “SDF Function” on page 596

Syntax

\texttt{LOGSDF('distribution', quantile, parameter-1, ..., parameter-k)}

\textbf{Required Arguments}

\textit{'distribution'}

is a character constant, variable, or expression that identifies the distribution. Valid distributions are as follows:

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bernoulli</td>
<td>'BERNOULLI'</td>
</tr>
<tr>
<td>Beta</td>
<td>'BETA'</td>
</tr>
<tr>
<td>Binomial</td>
<td>'BINOMIAL'</td>
</tr>
<tr>
<td>Cauchy</td>
<td>'CAUCHY'</td>
</tr>
<tr>
<td>Chi-Square</td>
<td>'CHISQUARE'</td>
</tr>
<tr>
<td>Conway-Maxwell-Poisson</td>
<td>'CONMAXPOI'</td>
</tr>
<tr>
<td>Exponential</td>
<td>'EXPONENTIAL'</td>
</tr>
<tr>
<td>\text{F}</td>
<td>'F'</td>
</tr>
<tr>
<td>Gamma</td>
<td>'GAMMA'</td>
</tr>
<tr>
<td>Generalized Poisson</td>
<td>'GENPOISSON'</td>
</tr>
<tr>
<td>Geometric</td>
<td>'GEOMETRIC'</td>
</tr>
<tr>
<td>Hypergeometric</td>
<td>'HYPERGEOMETRIC'</td>
</tr>
<tr>
<td>Laplace</td>
<td>'LAPLACE'</td>
</tr>
<tr>
<td>Logistic</td>
<td>'LOGISTIC'</td>
</tr>
<tr>
<td>Lognormal</td>
<td>'LOGNORMAL'</td>
</tr>
<tr>
<td>Negative binomial</td>
<td>'NEGBINOMIAL'</td>
</tr>
<tr>
<td>Normal</td>
<td>'NORMAL'</td>
</tr>
<tr>
<td>Normal mixture</td>
<td>'NORMALMIX'</td>
</tr>
</tbody>
</table>
Note: Except for T, F, and NORMALMIX, you can minimally identify any distribution by its first four characters.

quantile
is a numeric constant, variable, or expression that specifies the value of a random variable.

parameter-1, ..., parameter-k
are optional shape, location, or scale parameters appropriate for the specific distribution.

Details

The LOGSDF function computes the logarithm of the survival function from various continuous and discrete distributions. For more information, see “SDF Function” on page 596.

For more information about the distributions that are listed in the table, see “PDF Function” on page 477.

See Also

Functions:

- “CDF Function” on page 153
- “LOGCDF Function” on page 416
- “LOGPDF Function” on page 418
- “PDF Function” on page 477
- “QUANTILE Function” on page 549
- “SDF Function” on page 596
- “SQUANTILE Function” on page 1062

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pareto</td>
<td>'PARETO'</td>
</tr>
<tr>
<td>Poisson</td>
<td>'POISSON'</td>
</tr>
<tr>
<td>T</td>
<td>'T'</td>
</tr>
<tr>
<td>Tweedie</td>
<td>'TWEEDIE'</td>
</tr>
<tr>
<td>Uniform</td>
<td>'UNIFORM'</td>
</tr>
<tr>
<td>Wald (inverse Gaussian)</td>
<td>'WALD'</td>
</tr>
<tr>
<td>Weibull</td>
<td>'WEIBULL'</td>
</tr>
</tbody>
</table>
LOWCASE Function

Converts all letters in an argument to lowercase.

**Category:** Character

**Restriction:** This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility.

**Syntax**

LOWCASE(argument)

**Required Argument**

*argument*

specifies a character constant, variable, or expression.

**Details**

In a DATA step, if the LOWCASE function returns a value to a variable that has not previously been assigned a length, then that variable is given the length of the argument.

The LOWCASE function copies the character argument, converts all uppercase letters to lowercase letters, and returns the altered value as a result.

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x='INTRODUCTION';</td>
<td></td>
</tr>
<tr>
<td>y=lowcase(x);</td>
<td>introduction</td>
</tr>
<tr>
<td>put y;</td>
<td></td>
</tr>
</tbody>
</table>

**See Also**

Functions:

- “PROPCASE Function” on page 1020
- “UPCASE Function” on page 644

LPERM Function

Computes the logarithm of the PERM function, which is the logarithm of the number of permutations of n objects, with the option of including r number of elements.

**Category:** Combinatorial
Syntax

LPERM(n <, r>)

**Required Argument**

n

is an integer that represents the total number of elements from which the sample is chosen.

**Optional Argument**

r

is an optional integer value that represents the number of chosen elements. If r is omitted, the function returns the factorial of n.

Restriction  \( r \leq n \)

**Details**

The LPERM function computes the logarithm of the PERM function.

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=lperm(5000, 500); put x;</td>
<td>4232.7715946</td>
</tr>
<tr>
<td>y=lperm(100, 10); put y;</td>
<td>45.586735935</td>
</tr>
</tbody>
</table>

**See Also**

Functions:
- “PERM Function” on page 497

---

**MAD Function**

Returns the median absolute deviation from the median.

**Category:** Descriptive Statistics

**Syntax**

MAD(value-1 <, value-2...>)
**Required Argument**

*value*

specifies a numeric constant, variable, or expression of which the median absolute deviation from the median is to be computed.

**Details**

If all arguments have missing values, the result is a missing value. Otherwise, the result is the median absolute deviation from the median of the nonmissing values. The formula for the median is the same as the one that is used in the UNIVARIATE procedure. For more information, see Base SAS Procedures Guide: Statistical Procedures.

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>mad=mad(2, 4, 1, 3, 5, 999999);</td>
<td>1.5</td>
</tr>
<tr>
<td>put mad;</td>
<td></td>
</tr>
</tbody>
</table>

**See Also**

Functions:

- “IQR Function” on page 388
- “MEDIAN Function” on page 434
- “PCTL Function” on page 476

---

**MARGRCLPRC Function**

Calculates call prices for European options on stocks, based on the Margrabe model.

**Category:** Financial

**Syntax**

```
MARGRCLPRC(X, t, X, sigma-1, sigma-2, rho12)
```

**Required Arguments**

*\(X_1\)*

is a nonmissing, positive value that specifies the price of the first asset.

Requirement Specify \(X_1\) and \(X_2\) in the same units.

*t*

is a nonmissing value that specifies the time to expiration, in years.
The MARGRCLPRC function calculates the call price for European options on stocks, based on the Margrabe model. The function is based on the following relationship:

\[ \text{CALL} = X_1 N(d_1) - X_2 N(d_2) \]

**Arguments**

- **\( X_1 \)**
  specifies the price of the first asset.

- **\( X_2 \)**
  specifies the price of the second asset.

- **\( N \)**
  specifies the cumulative normal density function.

\[
\begin{align*}
  d_1 &= \frac{\ln \left( \frac{N_1}{N_2} \right) + \left( \frac{\sigma_1^2 + \sigma_2^2}{2} \right) t}{\sigma \sqrt{t}} \\
  d_2 &= d_1 - \sigma \sqrt{t} \\
  \sigma^2 &= \sigma_{x_1}^2 + \sigma_{x_2}^2 - 2\rho_{x_1,x_2} \sigma_{x_1} \sigma_{x_2}
\end{align*}
\]

The following arguments apply to the preceding equation:

- **\( t \)**
  specifies the time to expiration, in years.

- **\( \sigma_{x_1}^2 \)**
  specifies the variance of the first asset.

- **\( \sigma_{x_2}^2 \)**
  specifies the variance of the second asset.

- **\( \sigma_{x_1} \)**
  specifies the volatility of the first asset.

- **\( \sigma_{x_2} \)**
  specifies the volatility of the second asset.

**Details**

\[ X_2 \]

is a nonmissing, positive value that specifies the price of the second asset.

**Requirement**

Specify \( X_2 \) and \( X_1 \) in the same units.

**\( \sigma_1 \)**

is a nonmissing, positive fraction that specifies the volatility of the first asset.

**\( \sigma_2 \)**

is a nonmissing, positive fraction that specifies the volatility of the second asset.

**\( \rho_{12} \)**

specifies the correlation between the first and second assets, \( \rho_{x_1,x_2} \).

**Range**

between \(-1\) and 1
\[ \rho_{x_1,x_2} \]

specifies the correlation between the first and second assets.

For the special case of \( t=0 \), the following equation is true:

\[ \text{CALL} = \max((X_1 - X_2), 0) \]

\textit{Note:} This function assumes that there are no dividends from the two assets.

For information about the basics of pricing, see “Using Pricing Functions” on page 8.

\textbf{Comparisons}

The MARGRCLPRC function calculates the call price for European options on stocks, based on the Margrabe model. The MARGRPTPRC function calculates the put price for European options on stocks, based on the Margrabe model. These functions return a scalar value.

\textbf{Example}

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>a=margrclprc(15, .5, 13, .06, .05, 1); put a;</td>
<td>2</td>
</tr>
<tr>
<td>b=margrclprc(2, .25, 1, .3, .2, 1); put b;</td>
<td>1</td>
</tr>
</tbody>
</table>

\textbf{See Also}

Functions:
- “MARGRPTPRC Function” on page 427

\textbf{MARGRPTPRC Function}

Calculates put prices for European options on stocks, based on the Margrabe model.

\textbf{Category:} Financial

\textbf{Syntax}

MARGRPTPRC(\( X_1 \), \( t \), \( X_2 \), \( \sigma_1 \), \( \sigma_2 \), \( \rho_{12} \))

\textbf{Required Arguments}

\( X_1 \)

is a nonmissing, positive value that specifies the price of the first asset.
Requirement Specify $X_1$ and $X_2$ in the same units.

$t$

is a nonmissing value that specifies the time to expiration, in years.

$X_2$

is a nonmissing, positive value that specifies the price of the second asset.

Requirement Specify $X_2$ and $X_1$ in the same units.

$\sigma_1$

is a nonmissing, positive fraction that specifies the volatility of the first asset.

$\sigma_2$

is a nonmissing, positive fraction that specifies the volatility of the second asset.

$\rho_{12}$

specifies the correlation between the first and second assets, $\rho_{X_1,X_2}$.

Range between –1 and 1

Details

The MARGRPTPRC function calculates the put price for European options on stocks, based on the Margrabe model. The function is based on the following relationship:

\[
PUT = X_2 N(pd_1) - X_1 N(pd_2)
\]

Arguments

$X_1$

specifies the price of the first asset.

$X_2$

specifies the price of the second asset.

$N$

specifies the cumulative normal density function.

\[
pd_1 = \frac{\ln\left(\frac{X_1}{X_2}\right) + \left(\frac{\sigma_2^2}{2}\right) t}{\sigma_2 \sqrt{t}}
\]

\[
pd_2 = pd_1 - \sigma_1 \sqrt{t}
\]

\[
\sigma^2 = \sigma_{x_1}^2 + \sigma_{x_2}^2 - 2 \rho_{x_1,x_2} \sigma_{x_1} \sigma_{x_2}
\]

The following arguments apply to the preceding equation:

$t$

is a nonmissing value that specifies the time to expiration, in years.

$\sigma_{x_1}^2$

specifies the variance of the first asset.

$\sigma_{x_2}^2$

specifies the variance of the second asset.
\( \sigma_{x_1} \)

specifies the volatility of the first asset.

\( \sigma_{x_2} \)

specifies the volatility of the second asset.

\( \rho_{x_1, x_2} \)

specifies the correlation between the first and second assets.

To view the corresponding CALL relationship, see the “MARGRCLPRC Function” on page 425.

For the special case of \( t=0 \), the following equation is true:

\[
PUT = \max((X_2 - X_1), 0)
\]

**Note:** This function assumes that there are no dividends from the two assets.

For basic information about pricing, see “Using Pricing Functions” on page 8.

**Comparisons**

The MARGRPTPRC function calculates the put price for European options on stocks, based on the Margrabe model. The MARGRCLPRC function calculates the call price for European options on stocks, based on the Margrabe model. These functions return a scalar value.

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a = \text{margrptprc}(2, .25, 3, .06, .2, 1); )</td>
<td>1.0000000001</td>
</tr>
<tr>
<td>put ( a; )</td>
<td></td>
</tr>
<tr>
<td>( b = \text{margrptprc}(3, .25, 4, .05, .3, 1); )</td>
<td>1.0015762491</td>
</tr>
<tr>
<td>put ( b; )</td>
<td></td>
</tr>
</tbody>
</table>

**See Also**

**Functions:**

- “MARGRCLPRC Function” on page 425

---

**MAX Function**

Returns the largest value.

**Category:** Descriptive Statistics
Syntax

MAX(argument-1 <, argument-2, ...>)

Required Argument

argument

specifies a numeric constant, variable, or expression. At least one argument is required. If you use only one argument, then the value of that argument is returned. The argument list can consist of a variable list, which is preceded by OF.

Comparisons

The MAX function returns a missing value (.) only if all arguments are missing.
The MAX operator (<>) returns a missing value only if both operands are missing. In this case, it returns the value of the operand that is higher in the sort order for missing values.

Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=max(8, 3);</td>
<td>8</td>
</tr>
<tr>
<td>x1=max(2, 6, .);</td>
<td>6</td>
</tr>
<tr>
<td>x2=max(2, -3, 1, -1);</td>
<td>2</td>
</tr>
<tr>
<td>x3=max(3, ., -3);</td>
<td>3</td>
</tr>
<tr>
<td>x4=max(of x1-x3);</td>
<td>6</td>
</tr>
<tr>
<td>x5=max(73);</td>
<td>73</td>
</tr>
</tbody>
</table>

MD5 Function

Returns the result of the message digest of a specified string.

Category: Character
Restriction: This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility.

Syntax

MD5(string)
Required Argument

string

specifies a character constant, variable, or expression.

Tip  Enclose a literal string of characters in quotation marks.

Details

Length of Returned Variable
In a DATA step, if the MD5 function returns a value to a variable that has not previously been assigned a length, then that variable is assigned a length of 200 bytes.

The Basics
The MD5 function converts a string, based on the MD5 algorithm, to a 128-bit hash value. This hash value is referred to as a message digest (digital signature), which is nearly unique for each string that is passed to the function.

The MD5 function does not format its own output. Use the $BINARYw. or $HEXw. formats to view readable results.

The Message Digest Algorithm
A message digest results from manipulating and compacting an arbitrarily long stream of binary data. An ideal message digest algorithm never generates the same result for two different sets of input. However, generating such a unique result would require a message digest as long as the input itself. Therefore, MD5 generates a message digest of modest size (16 bytes), created with an algorithm that is designed to make a nearly unique result.

Using the MD5 Function
You can use the MD5 function to track changes in your data sets. The MD5 function can generate a digest of a set of column values in a record in a table. This digest could be treated as the signature of the record, and be used to track changes that are made to the record. If the digest from the new record matches the existing digest of a record in a table, then the two records are the same. If the digest is different, then a column value in the record has changed. The new changed record could then be added to the table along with a new surrogate key because it represents a change to an existing keyed value.

The MD5 function can be useful when you are developing shell scripts or Perl programs for software installation, file comparison, and detection of file corruption and tampering.

You can also use the MD5 function to create a unique identifier for observations to be used as the key of a hash package.

Example: Generating Results with the MD5 Function
This example generates results that are returned by the MD5 function.

```plaintext
data _null_
  y=md5('abc');
  z=md5('access method');
  put y=/ y=$hex32.;
  put z=/ z=$hex32.;
run;
```

The output from this program contains unprintable characters.
MDY Function
Returns a SAS date value from month, day, and year values.

Category: Date and Time

Syntax

```
MDY(month, day, year)
```

Required Arguments

- **month**
  - specifies a numeric constant, variable, or expression that represents an integer from 1 through 12.

- **day**
  - specifies a numeric constant, variable, or expression that represents an integer from 1 through 31.

- **year**
  - specifies a numeric constant, variable, or expression with a value of a two-digit or four-digit integer that represents the year. The YEARCUTOFF= system option defines the year value for two-digit dates.

Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>birthday=mdy(8, 27, 90);</td>
<td>11196</td>
</tr>
<tr>
<td>put birthday;</td>
<td></td>
</tr>
<tr>
<td>put birthday= worddate.;</td>
<td>birthday=August 27, 1990</td>
</tr>
<tr>
<td>anniversary=mdy(7, 11, 2001);</td>
<td>15167</td>
</tr>
<tr>
<td>put anniversary;</td>
<td></td>
</tr>
<tr>
<td>put anniversary=date9.;</td>
<td>anniversary=11JUL2001</td>
</tr>
</tbody>
</table>

See Also

Functions:

- “DAY Function” on page 223
- “MONTH Function” on page 442
- “YEAR Function” on page 690
**MEAN Function**
Returns the arithmetic mean (average).

**Category:** Descriptive Statistics

**Syntax**

```
MEAN(argument-1 <, ... argument-n>)
```

**Required Argument**

*argument*  
specifies a numeric constant, variable, or expression. At least one nonmissing argument is required. Otherwise, the function returns a missing value.

**Tip** The argument list can consist of a variable list, which is preceded by OF.

**Details**

The GEOMEAN function returns the geometric mean, the HARMean function returns the harmonic mean, and the MEDIAN function returns the median of the nonmissing values, whereas the MEAN function returns the arithmetic mean (average).

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1=mean(2, . , . , 6);</td>
<td>4</td>
</tr>
<tr>
<td>x2=mean(1, 2, 3, 2);</td>
<td>2</td>
</tr>
<tr>
<td>x3=mean(of x1-x2);</td>
<td>3</td>
</tr>
</tbody>
</table>

**See Also**

**Functions:**

- “GEOMEAN Function” on page 315
- “GEOMEANZ Function” on page 316
- “HARMean Function” on page 318
- “HARMeanZ Function” on page 319
- “MEDIAN Function” on page 434
MEDIAN Function
Returns the median value.

Category: Descriptive Statistics

Syntax
MEDIAN(value–1 <, value–2, …>)

Required Argument
value
is a numeric constant, variable, or expression.

Details
The MEDIAN function returns the median of the nonmissing values. If all arguments have missing values, the result is a missing value.

Comparisons
The MEDIAN function returns the median of nonmissing values, whereas the MEAN function returns the arithmetic mean (average).

Example
The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=median(2, 4, 1, 3);</td>
<td>2.5</td>
</tr>
<tr>
<td>y=median(5, 8, 0, 3, 4);</td>
<td>4</td>
</tr>
</tbody>
</table>

See Also
Functions:
- “MEAN Function” on page 433

MIN Function
Returns the smallest value.

Category: Descriptive Statistics
Syntax

MIN(argument-1 <, argument-2, ...>)

Required Argument

argument

specifies a numeric constant, variable, or expression. At least one argument is required. If you use only one argument, then the value of that argument is returned. The argument list can consist of a variable list, which is preceded by OF.

Details

The MIN function returns a missing value (.) only if all arguments are missing.
The MIN operator (><) returns a missing value if either operand is missing. In this case, it returns the value of the operand that is lower in the sort order for missing values.

Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=min(7, 4);</td>
<td>4</td>
</tr>
<tr>
<td>x1=min(2, ., 6);</td>
<td>2</td>
</tr>
<tr>
<td>x2=min(2, -3, 1, -1);</td>
<td>-3</td>
</tr>
<tr>
<td>x3=min(0, 4);</td>
<td>0</td>
</tr>
<tr>
<td>x4=min(of x1-x3);</td>
<td>-3</td>
</tr>
<tr>
<td>x5=min(34);</td>
<td>34</td>
</tr>
</tbody>
</table>

MINUTE Function

Returns the minute from a SAS time or datetime value.

Category: Date and Time

Syntax

MINUTE(time | datetime)

Required Arguments

time

is a numeric constant, variable, or expression that specifies a SAS time value.
**datetime**

is a numeric constant, variable, or expression that specifies a SAS datetime value.

**Details**

The MINUTE function returns an integer that represents a specific minute of the hour. MINUTE always returns a positive number in the range of 0 through 59.

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>time='3:19:24't;</td>
<td></td>
</tr>
<tr>
<td>m=minute(time);</td>
<td>19</td>
</tr>
<tr>
<td>put m;</td>
<td></td>
</tr>
</tbody>
</table>

**See Also**

Functions:

- “HOUR Function” on page 325
- “SECOND Function” on page 600

**MISSING Function**

Returns a numeric result that indicates whether the argument contains a missing value.

- **Categories:** Descriptive Statistics
  Character
- **Restriction:** This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility.

**Syntax**

MISSING(*numeric-expression* | *character-expression*)

**Required Arguments**

- **numeric-expression**
  specifies a numeric constant, variable, or expression.
- **character-expression**
  specifies a character constant, variable, or expression.

**Details**

- The MISSING function checks a numeric or character expression for a missing value, and returns a numeric result. If the argument does not contain a missing value,
SAS returns a value of 0. If the argument contains a missing value, SAS returns a value of 1.

- A numeric expression is considered missing if it evaluates to a numeric missing value: .., _, .A, ..., .Z.
- A character expression is considered missing if it evaluates to a string that contains all blanks or has a length of zero.

The example “Example 2: Using the MISSING Function with CAS” on page 437 shows how to run the MISSING function in CAS.

Some language elements are supported in SAS but not in CAS. An example is the DATALINES statement. Some examples in this document use DATALINES in SAS to create a data set using the CAS engine. For more information, see “Functions in SAS and CAS” on page 54.

Comparisons

The MISSING function can have only one argument. The CMISS function can have multiple arguments and returns a count of the missing values. The NMISS function requires numeric arguments and returns the number of missing values in the list of arguments.

Examples

Example 1: Using the MISSING Function with SAS

This example uses the MISSING function to check whether the input variables contain missing values.

```sas
data values;
  input @1 var1 3. @5 var2 3.;
  if missing(var1) then
    do;
      put 'Variable 1 is Missing.';
    end;
  else if missing(var2) then
    do;
      put 'Variable 2 is Missing.';
    end;
  datalines;
  127
  988 195
; run;
```

SAS writes the following output to the log:

```
Variable 2 is Missing.
```

Example 2: Using the MISSING Function with CAS

This example uses the MISSING function to check whether the input variables contain missing values. The first section of code in this example runs in SAS and creates the data set, Values, with the CAS engine. The second section of code runs in CAS, accesses the data in Values that is in CAS, and then creates the Values1 data set.

```sas
data casuser.values;
```
input @1 var1 3. @5 var2 3.;
datalines;
127
988 195;
data casuser.values1;
  set casuser.values;
  if missing(var1) then 
    do;
      put 'Variable 1 is Missing.';
    end;
  else if missing(var2) then 
    do;
      put 'Variable 2 is Missing.';
    end;
run;
SAS writes the following output to the log:

Variable 2 is Missing.

See Also

Functions:
• “CMISS Function” on page 855
• “NMISS Function” on page 446

CALL Routines:
• “CALL MISSING Routine” on page 126

MOD Function
Returns the remainder from the division of the first argument by the second argument, fuzzed to avoid most unexpected floating-point results.

Category: Mathematical

Syntax

MOD(argument-1, argument-2)

Required Arguments

argument-1
  is a numeric constant, variable, or expression that specifies the dividend.

argument-2
  is a numeric constant, variable, or expression that specifies the divisor.

Restriction cannot be 0
Details

The MOD function returns the remainder from the division of argument-1 by argument-2. When the result is nonzero, the result has the same sign as the first argument. The sign of the second argument is ignored.

The computation that is performed by the MOD function is exact if both of the following conditions are true:

• Both arguments are exact integers.
• All integers that are less than either argument have exact 8-byte floating-point representations.

To determine the largest integer for which the computation is exact, execute the following DATA step:

```sas
data _null_
  exactint=constant('exactint');
  put exactint=;
run;
```

Operating Environment Information

For information about the largest integer, see the SAS documentation for your operating environment.

If either of the above conditions is not true, a small amount of numerical error can occur in the floating-point computation. In this case

• MOD returns zero if the remainder is very close to zero or very close to the value of the second argument.
• MOD returns a missing value if the remainder cannot be computed to a precision of approximately three digits or more. In this case, SAS also writes an error message to the log.

Note: Prior to SAS 9, the MOD function did not perform the adjustments to the remainder that were described in the previous paragraph. For this reason, the results of the MOD function in SAS 9 might differ from previous versions.

Comparisons

Here are some comparisons between the MOD and MODZ functions:

• The MOD function performs extra computations, called fuzzing, to return an exact zero when the result would otherwise differ from zero because of numerical error.
• The MODZ function performs no fuzzing.
• Both the MOD and MODZ functions return a missing value if the remainder cannot be computed to a precision of approximately three digits or more.

Example

The following SAS statements produce these results for MOD and MODZ:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1=mod(10, 3);</td>
<td>1.0000</td>
</tr>
<tr>
<td>put x1 9.4;</td>
<td></td>
</tr>
</tbody>
</table>
MODZ Function

Returns the remainder from the division of the first argument by the second argument, using zero fuzzing.

**Category:** Mathematical

**Syntax**

\[
\text{MODZ}(\text{argument-1, argument-2})
\]

**Required Arguments**

- **argument-1** is a numeric constant, variable, or expression that specifies the dividend.
- **argument-2** is a nonzero numeric constant, variable, or expression that specifies the divisor.

**See Also**

**Functions:**
- “INT Function” on page 340
- “INTZ Function” on page 385
- “MODZ Function” on page 440
Details

The MODZ function returns the remainder from the division of \textit{argument-1} by \textit{argument-2}. When the result is nonzero, the result has the same sign as the first argument. The sign of the second argument is ignored.

The computation that is performed by the MODZ function is exact if both of the following conditions are true:

- Both arguments are exact integers.
- All integers that are less than either argument have exact 8-byte floating-point representation.

To determine the largest integer for which the computation is exact, execute the following DATA step:

```sas
data _null_
    exactint=constant('exactint');
    put exactint=;
run;
```

\textit{Operating Environment Information}

For information about the largest integer, see the SAS documentation for your operating environment.

If either of the above conditions is not true, a small amount of numerical error can occur in the floating-point computation. For example, when you use exact arithmetic and the result is zero, MODZ might return a very small positive value or a value slightly less than the second argument.

Comparisons

Here are some comparisons between the MODZ and MOD functions:

- The MODZ function performs no fuzzing.
- The MOD function performs extra computations, called fuzzing, to return an exact zero when the result would otherwise differ from zero because of numerical error.
- Both the MODZ and MOD functions return a missing value if the remainder cannot be computed to a precision of approximately three digits or more.

Example

The following SAS statements produce these results for MOD and MODZ:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1=mod(10, 3); put x1 9.4;</td>
<td>1.0000</td>
</tr>
<tr>
<td>xa=modz(10, 3); put xa 9.4;</td>
<td>1.0000</td>
</tr>
<tr>
<td>x2=mod(.3, -.1); put x2 9.4;</td>
<td>0.0000</td>
</tr>
<tr>
<td>xb=modz(.3, -.1); put xb 9.4;</td>
<td>0.1000</td>
</tr>
<tr>
<td>SAS Statement</td>
<td>Result</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td><code>x3=mod(1.7, .1);</code></td>
<td>0.0000</td>
</tr>
<tr>
<td><code>put x3 9.4;</code></td>
<td></td>
</tr>
<tr>
<td><code>xc=modz(1.7, .1);</code></td>
<td>0.0000</td>
</tr>
<tr>
<td><code>put xc 9.4;</code></td>
<td></td>
</tr>
<tr>
<td><code>x4=mod(.9, .3);</code></td>
<td>0.00000000000000000000</td>
</tr>
<tr>
<td><code>put x4 24.20;</code></td>
<td></td>
</tr>
<tr>
<td><code>xd=modz(.9, .3);</code></td>
<td>0.00000000000000005551</td>
</tr>
<tr>
<td><code>put xd 24.20;</code></td>
<td></td>
</tr>
</tbody>
</table>

### See Also

**Functions:**
- “INT Function” on page 340
- “INTZ Function” on page 385
- “MOD Function” on page 438

### MONTH Function

Returns the month from a SAS date value.

**Category:** Date and Time

**Syntax**

```
MONTH(date)
```

**Required Argument**

`date`

specifies a numeric constant, variable, or expression that represents a SAS date value.

**Details**

The MONTH function returns a numeric value that represents the month from a SAS date value. Numeric values can range from 1 through 12.
Example
The following SAS statements produce this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>date='25jan94'd;</td>
<td></td>
</tr>
<tr>
<td>m=month(date);</td>
<td></td>
</tr>
<tr>
<td>put m;</td>
<td>1</td>
</tr>
</tbody>
</table>

See Also

Functions:
- “DAY Function” on page 223
- “YEAR Function” on page 690

MORT Function
Returns amortization parameters.

Category: Financial

Syntax

MORT(a, p, r, n)

Required Arguments

a
  is numeric, and specifies the initial amount.

p
  is numeric, and specifies the periodic payment.

r
  is numeric, and specifies the periodic interest rate that is expressed as a fraction.

n
  is an integer, and specifies the number of compounding periods.

Range    n ≥ 0

Details

Calculating Results
The MORT function returns the missing argument in the list of four arguments from an amortization calculation with a fixed interest rate that is compounded each period. The arguments are related by the following equation:

\[ p = \frac{ar(1 + r)^n}{(1 + r)^n - 1} \]
One missing argument must be provided. The value is then calculated from the remaining three. No adjustment is made to convert the results to round numbers.

**Restrictions in Calculating Results**
The MORT function returns an invalid argument note to the SAS log and sets _ERROR_ to 1 if one of the following argument combinations is true:

- rate < -1 or n < 0
- principal <= 0 or payment <= 0 or n <= 0
- principal <= 0 or payment <= 0 or rate <= -1
- principal * rate > payment
- principal > payment * n

**Example**
In the following statement, an amount of $50,000 is borrowed for 30 years at an annual interest rate of 10% compounded monthly. The monthly payment can be expressed as follows:

```
payment=mort(50000, . , .10/12, 30*12);
```

The value that is returned is 438.79 (rounded). The second argument has been set to missing, which indicates that the periodic payment is to be calculated. The 10% nominal annual rate has been converted to a monthly rate of 0.10/12. The rate is the fractional (not the percentage) interest rate per compounding period. The 30 years are converted to 360 months.

---

**N Function**
Returns the number of nonmissing numeric values.

**Category:** Descriptive Statistics

**Syntax**

```
N(argument-1 < ..., argument-n>)
```

**Required Argument**

`argument`
specifies a numeric constant, variable, or expression. At least one argument is required. The argument list can consist of a variable list, which is preceded by OF.

**Comparisons**
The N function counts nonmissing values, whereas the NMISS and the CMISS functions count missing values. N requires numeric arguments, whereas CMISS works with both numeric and character values.
Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1=n(1, 0, ., 2, 5, .);</td>
<td>4</td>
</tr>
<tr>
<td>x2=n(1, 2);</td>
<td>2</td>
</tr>
<tr>
<td>x3=n(of x1-x2);</td>
<td>2</td>
</tr>
</tbody>
</table>

NETPV Function

Returns the net present value as a percent.

Category: Financial

Syntax

NETPV(r, frequency, c0, c1, ..., cn)

Required Arguments

r
is numeric, the interest rate over a specified base period of time expressed as a fraction.

Range  \( r \geq 0 \)

frequency
is numeric, the number of payments during the base period of time that is specified with the rate \( r \).

Range  \( frequency > 0 \)

Note  The case \( frequency = 0 \) is a flag to allow continuous discounting.

c0, c1, ..., cn
are numeric cash flows that represent cash outlays (payments) or cash inflows (income) occurring at times 0, 1, ... n. These cash flows are assumed to be equally spaced, beginning-of-period values. Negative values represent payments, positive values represent income, and values of 0 represent no cash flow at a given time. The \( c0 \) argument and the \( c1 \) argument are required.

Details

The NETPV function returns the net present value at time 0 for the set of cash payments \( c0, c1, ..., cn \), with a rate \( r \) over a specified base period of time. The argument \( frequency > 0 \) describes the number of payments that occur over the specified base period of time.

The net present value is given by the equation:
The following relationship applies to the preceding equation:

\[ x = \begin{cases} 
\frac{1}{(1 + r)^{1/freq}} & \text{freq} > 0 \\
\varepsilon^{-r} & \text{freq} = 0 
\end{cases} \]

Missing values in the payments are treated as 0 values. When `frequency`\(>0\), the rate \(r\) is the effective rate over the specified base period. To compute with a quarterly rate (the base period is three months) of 4% with monthly cash payments, set `frequency` to 3 and set \(r\) to .04.

If `frequency` is 0, continuous discounting is assumed. The base period is the time interval between two consecutive payments, and the rate \(r\) is a nominal rate.

To compute with a nominal annual interest rate of 11% discounted continuously with monthly payments, set `frequency` to 0 and set \(r\) to .11/12.

**Example**

For an initial investment of $500 that returns biannual payments of $200, $300, and $400 over the succeeding 6 years and an annual discount rate of 10%, the net present value of the investment can be expressed as follows:

```sas
value=netpv(.10, .5, -500, 200, 300, 400);
```

The value that is returned is 95.982864829.

**See Also**

**Functions:**

- “NPV Function” on page 472

---

**NMISS Function**

Returns the number of missing numeric values.

**Category:** Descriptive Statistics

**Syntax**

\[ \text{NMISS}(\text{argument-1}, \ldots, \text{argument-n}) \]

**Required Argument**

`argument` specifies a numeric constant, variable, or expression. At least one argument is required. The argument list can consist of a variable list, which is preceded by `OF`. 
Details

The NMISS function returns the number of missing values, whereas the N function returns the number of nonmissing values. NMISS requires numeric values, whereas CMISS works with both numeric and character values. NMISS works with multiple numeric values, whereas MISSING works with only one value that can be either numeric or character.

Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1=nmiss(1, 0, ., 2, 5, .);</td>
<td>2</td>
</tr>
<tr>
<td>x2=nmiss(1, 0);</td>
<td>0</td>
</tr>
<tr>
<td>x3=nmiss(of x1-x2);</td>
<td>0</td>
</tr>
</tbody>
</table>

NOMRATE Function

Returns the nominal annual interest rate.

Category: Financial

Syntax

NOMRATE(compounding-interval, rate)

Required Arguments

compounding-interval

is a SAS interval. This value represents how often the returned value is compounded.

rate

is numeric. Rate is the effective annual interest rate (expressed as a percentage) that is compounded at each interval.

Details

The NOMRATE function returns the nominal annual interest rate. NOMRATE computes the nominal annual interest rate that corresponds to an effective annual interest rate.

The following details apply to the NOMRATE function:

- The values for rates must be at least –99.
- In considering an effective interest rate and a compounding interval, if compounding-interval is 'CONTINUOUS', then the value that is returned by NOMRATE equals \( \log(e(1+\frac{\text{rate}}{100})) \).

If compounding-interval is not 'CONTINUOUS', and \( m \) intervals occur in a year, the value that is returned by NOMRATE equals the following:
\[ m \left( 1 + \frac{\text{rate}}{100} \right)^{\frac{1}{m}} - 1 \]

- The following values are valid for compounding-interval:
  - 'CONTINUOUS'
  - 'DAY'
  - 'SEMIMONTH'
  - 'MONTH'
  - 'QUARTER'
  - 'SEMIYEAR'
  - 'YEAR'
- If the interval is 'DAY', then \( m = 365 \).

**Example**
- If an effective rate is 10% when compounded monthly, the corresponding nominal rate can be expressed as follows:
  \[
  \text{effective\_rate1} = \text{NOMRATE('MONTH', 10)};
  \]
- If an effective rate is 10% when compounded quarterly, the corresponding nominal rate can be expressed as follows:
  \[
  \text{effective\_rate2} = \text{NOMRATE('QUARTER', 10)};
  \]

---

**NOTALNUM Function**

Searches a character string for a non-alphanumeric character, and returns the first position at which the character is found.

- **Category:** Character
- **Restriction:** This function is assigned an I18N Level 1 status. If possible, avoid I18N Level 1 functions if you are using a non-English language. Under certain circumstances, the I18N Level 1 functions might not work correctly with Double-Byte Character Set (DBCS) or Multi-Byte Character Set (MBCS) encodings. For more information, see [Internationalization Compatibility](#).

**Syntax**

\[
\text{NOTALNUM(string <, start>)}
\]

**Required Argument**

- **string**
  specifies a character constant, variable, or expression to search.
Optional Argument

start

is an optional numeric constant, variable, or expression with an integer value that specifies the position at which the search should start and the direction in which to search.

Details

The NOTALNUM function searches a string for the first occurrence of any character that is not a digit or an uppercase or lowercase letter. If such a character is found, NOTALNUM returns the position in the string of that character. If no such character is found, NOTALNUM returns a value of 0.

If you use only one argument, NOTALNUM begins the search at the beginning of the string. If you use two arguments, the absolute value of the second argument, start, specifies the position at which to begin the search. The direction in which to search is determined in the following way:

• If the value of start is positive, the search proceeds to the right.
• If the value of start is negative, the search proceeds to the left.
• If the value of start is less than the negative length of the string, the search begins at the end of the string.

NOTALNUM returns a value of zero when one of the following is true:

• The character that you are searching for is not found.
• The value of start is greater than the length of the string.
• The value of start = 0.

Comparisons

The NOTALNUM function searches a character string for a non-alphanumeric character. The ANYALNUM function searches a character string for an alphanumeric character.

Example

The following example uses the NOTALNUM function to search a string from left to right for non-alphanumeric characters.

```sas
data _null_;
  string='Next = Last + 1;';
  j=0;
  do until(j=0);
    j=notalnum(string, j+1);
    if j=0 then put +3 "That's all";
    else do;
      c=substr(string, j, 1);
      put +3 j= c=;
    end;
  end;
run;
```
SAS writes the following output to the log:

```
  j=5  c=
  j=6  c==
  j=7  c=
  j=12 c=
  j=13 c=+
  j=14 c=
  j=16 c=;
  That's all
```

See Also

Functions:
- “ANYALNUM Function” on page 80

---

**NOTALPHA Function**

Searches a character string for a nonalphabetic character, and returns the first position at which the character is found.

- **Category:** Character
- **Restriction:** This function is assigned an I18N Level 1 status. If possible, avoid I18N Level 1 functions if you are using a non-English language. Under certain circumstances, the I18N Level 1 functions might not work correctly with Double-Byte Character Set (DBCS) or Multi-Byte Character Set (MBCS) encodings. For more information, see Internationalization Compatibility.

**Syntax**

```
NOTALPHA(string <, start>)
```

**Required Argument**

- `string` is the character constant, variable, or expression to search.

**Optional Argument**

- `start` is an optional numeric constant, variable, or expression with an integer value that specifies the position at which the search should start and the direction in which to search.

**Details**

The NOTALPHA function searches a string for the first occurrence of any character that is not an uppercase or lowercase letter. If such a character is found, NOTALPHA returns the position in the string of that character. If no such character is found, NOTALPHA returns a value of 0.

If you use only one argument, NOTALPHA begins the search at the beginning of the string. If you use two arguments, the absolute value of the second argument, `start`,
specifies the position at which to begin the search. The direction in which to search is determined in the following way:

- If the value of `start` is positive, the search proceeds to the right.
- If the value of `start` is negative, the search proceeds to the left.
- If the value of `start` is less than the negative length of the string, the search begins at the end of the string.

`NOTALPHA` returns a value of zero when one of the following is true:

- The character that you are searching for is not found.
- The value of `start` is greater than the length of the string.
- The value of `start` = 0.

**Comparisons**

The `NOTALPHA` function searches a character string for a nonalphabetic character. The `ANYALPHA` function searches a character string for an alphabetic character.

**Examples**

**Example 1: Searching a String for Nonalphabetic Characters**

The following example uses the `NOTALPHA` function to search a string from left to right for nonalphabetic characters.

```sas
data _null_;
  string='Next = _n_ + 12E3;';
  j=0;
  do until(j=0);
    j=notalpha(string, j+1);
    if j=0 then put +3 "That's all";
    else do;
      c=substr(string, j, 1);
      put +3 j= c=;
    end;
  end;
run;
```

SAS writes the following output to the log:

```
j=5 c=
j=6 c==
j=7 c=
j=8 c=_
j=10 c=_
j=11 c=
j=12 c=
j=13 c=
j=14 c=1
j=15 c=2
j=17 c=3
j=18 c=;
That's all
```
Example 2: Identifying Control Characters By Using the NOTALPHA Function
You can execute the following program to show the control characters that are identified by the NOTALPHA function.

```sas
data test;
  do dec=0 to 255;
    byte=byte(dec);
    hex=put(dec,hex2.);
    notalpha=notalpha(byte);
    output;
  end;
proc print data=test;
run;
```

See Also

Functions:

- “ANYALPHA Function” on page 82

NOTCNTRL Function

Searches a character string for a character that is not a control character, and returns the first position at which that character is found.

- **Category:** Character
- **Restriction:** This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility.

**Syntax**

`NOTCNTRL(string <, start>)`

**Required Argument**

`string`

is the character constant, variable, or expression to search.

**Optional Argument**

`start`

is an optional numeric constant, variable, or expression with an integer value that specifies the position at which the search should start and the direction in which to search.

**Details**

The NOTCNTRL function searches a string for the first occurrence of a character that is not a control character. If such a character is found, NOTCNTRL returns the position in the string of that character. If no such character is found, NOTCNTRL returns a value of 0.
If you use only one argument, NOTCNTRL begins the search at the beginning of the string. If you use two arguments, the absolute value of the second argument, \texttt{start}, specifies the position at which to begin the search. The direction in which to search is determined in the following way:

- If the value of \texttt{start} is positive, the search proceeds to the right.
- If the value of \texttt{start} is negative, the search proceeds to the left.
- If the value of \texttt{start} is less than the negative length of the string, the search begins at the end of the string.

NOTCNTRL returns a value of zero when one of the following is true:

- The character that you are searching for is not found.
- The value of \texttt{start} is greater than the length of the string.
- The value of \texttt{start} = 0.

**Comparisons**

The NOTCNTRL function searches a character string for a character that is not a control character. The ANYCNTRL function searches a character string for a control character.

**Example**

You can execute the following program to show the control characters that are identified by the NOTCNTRL function.

```plaintext
data test;
do dec=0 to 255;
   byte=byte(dec);
   hex=put(dec, hex2.);
   notcntrl=notcntrl(byte);
   output;
end;
proc print data=test;
run;
```

**See Also**

**Functions:**

- “ANYCNTRL Function” on page 84

\textbf{NOTDIGIT Function}

Searches a character string for any character that is not a digit, and returns the first position at which that character is found.

**Category:** Character

**Restriction:** This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility.
Syntax

NOTDIGIT(string <, start>)

Required Argument

string

is the character constant, variable, or expression to search.

Optional Argument

start

is an optional numeric constant, variable, or expression with an integer value that specifies the position at which the search should start and the direction in which to search.

Details

The NOTDIGIT function searches a string for the first occurrence of any character that is not a digit. If such a character is found, NOTDIGIT returns the position in the string of that character. If no such character is found, NOTDIGIT returns a value of 0.

If you use only one argument, NOTDIGIT begins the search at the beginning of the string. If you use two arguments, the absolute value of the second argument, start, specifies the position at which to begin the search. The direction in which to search is determined in the following way:

• If the value of start is positive, the search proceeds to the right.
• If the value of start is negative, the search proceeds to the left.
• If the value of start is less than the negative length of the string, the search begins at the end of the string.

NOTDIGIT returns a value of zero when one of the following is true:

• The character that you are searching for is not found.
• The value of start is greater than the length of the string.
• The value of start = 0.

Comparisons

The NOTDIGIT function searches a character string for any character that is not a digit. The ANYDIGIT function searches a character string for a digit.

Example

The following example uses the NOTDIGIT function to search for a character that is not a digit.

```
data _null_;  
  string='Next = _n_ +12E3;';  
  j=0;  
  do until(j=0);  
    j=notdigit(string, j+1);  
    if j=0 then put +3 "That's all";  
    else do;  
      c=substr(string, j, 1);  
  end;
```
SAS writes the following output to the log:

<table>
<thead>
<tr>
<th>j</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>N</td>
</tr>
<tr>
<td>2</td>
<td>e</td>
</tr>
<tr>
<td>3</td>
<td>x</td>
</tr>
<tr>
<td>4</td>
<td>t</td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>==</td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>_</td>
</tr>
<tr>
<td>9</td>
<td>n</td>
</tr>
<tr>
<td>10</td>
<td>_</td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>+</td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>E</td>
</tr>
<tr>
<td>18</td>
<td>;</td>
</tr>
</tbody>
</table>

That's all

See Also

Functions:

- “ANYDIGIT Function” on page 86

NOTFIRST Function

Searches a character string for an invalid first character in a SAS variable name under VALIDVARNAME=V7, and returns the first position at which that character is found.

**Category:** Character

**Restriction:** This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility.

**Syntax**

\[
\text{NOTFIRST}(\text{string } \langle, \text{ start}\rangle)
\]

**Required Argument**

**string**

is the character constant, variable, or expression to search.

**Optional Argument**

**start**

is an optional numeric constant, variable, or expression with an integer value that specifies the position at which the search should start and the direction in which to search.
Details

The NOTFIRST function does not depend on the TRANTAB, ENCODING, or LOCALE system options.

The NOTFIRST function searches a string for the first occurrence of any character that is not valid as the first character in a SAS variable name under VALIDVARNAME=V7. These characters are any except the underscore (_) and uppercase or lowercase English letters. If such a character is found, NOTFIRST returns the position in the string of that character. If no such character is found, NOTFIRST returns a value of 0.

If you use only one argument, NOTFIRST begins the search at the beginning of the string. If you use two arguments, the absolute value of the second argument, start, specifies the position at which to begin the search. The direction in which to search is determined in the following way:

- If the value of start is positive, the search proceeds to the right.
- If the value of start is negative, the search proceeds to the left.
- If the value of start is less than the negative length of the string, the search begins at the end of the string.

NOTFIRST returns a value of zero when one of the following is true:

- The character that you are searching for is not found.
- The value of start is greater than the length of the string.
- The value of start = 0.

Comparisons

The NOTFIRST function searches a string for the first occurrence of any character that is not valid as the first character in a SAS variable name under VALIDVARNAME=V7. The ANYFIRST function searches a string for the first occurrence of any character that is valid as the first character in a SAS variable name under VALIDVARNAME=V7.

Example

The following example uses the NOTFIRST function to search a string for any character that is not valid as the first character in a SAS variable name under VALIDVARNAME=V7.

```sas
data _null_;
  string='Next = _n_ + 12E3;';
  j=0;
  do until(j=0);
    j=notfirst(string, j+1);
    if j=0 then put +3 "That's all";
    else do;
      c=substr(string, j, 1);
      put +3 j= c=;
    end;
  end;
run;
```
SAS writes the following output to the log:

```
j=5 c=
j=6 c==
j=7 c=
j=11 c=
j=12 c=+
j=13 c=
j=14 c=1
j=15 c=2
j=17 c=3
j=18 c=;
That's all
```

See Also

Functions:

- “ANYFIRST Function” on page 87

---

**NOTGRAPH Function**

Searches a character string for a non-graphical character, and returns the first position at which that character is found.

- **Category:** Character
- **Restriction:** This function is assigned an I18N Level 1 status. If possible, avoid I18N Level 1 functions if you are using a non-English language. Under certain circumstances, the I18N Level 1 functions might not work correctly with Double-Byte Character Set (DBCS) or Multi-Byte Character Set (MBCS) encodings. For more information, see Internationalization Compatibility.

**Syntax**

```
NOTGRAPH(string <, start>)
```

- **Required Argument**
  
  `string`

  is the character constant, variable, or expression to search.

- **Optional Argument**
  
  `start`

  is an optional numeric constant, variable, or expression with an integer value that specifies the position at which the search should start and the direction in which to search.

**Details**

The NOTGRAPH function searches a string for the first occurrence of a non-graphical character. A graphical character is defined as any printable character other than white space. If such a character is found, NOTGRAPH returns the position in the string of that character. If no such character is found, NOTGRAPH returns a value of 0.
If you use only one argument, NOTGRAPH begins the search at the beginning of the string. If you use two arguments, the absolute value of the second argument, \textit{start}, specifies the position at which to begin the search. The direction in which to search is determined in the following way:

- If the value of \textit{start} is positive, the search proceeds to the right.
- If the value of \textit{start} is negative, the search proceeds to the left.
- If the value of \textit{start} is less than the negative length of the string, the search begins at the end of the string.

NOTGRAPH returns a value of zero when one of the following is true:

- The character that you are searching for is not found.
- The value of \textit{start} is greater than the length of the string.
- The value of \textit{start} = 0.

\section*{Comparisons}

The NOTGRAPH function searches a character string for a non-graphical character. The ANYGRAPH function searches a character string for a graphical character.

\section*{Examples}

\subsection*{Example 1: Searching a String for Non-Graphical Characters}

The following example uses the NOTGRAPH function to search a string for a non-graphical character.

```sas
data _null_;     
  string='Next = _n_ + 12E3;';  
  j=0;                      
  do until(j=0);            
    j=notgraph(string, j+1); 
    if j=0 then put +3 "That's all"; 
    else do; 
      c=substr(string, j, 1); 
      put +3 j= c=; 
    end;  
  end;  
run;
```

SAS writes the following output to the log:

\begin{verbatim}
  j=5  c=  
  j=7  c=  
  j=11 c=  
  j=13 c=  
  That's all
\end{verbatim}

\subsection*{Example 2: Identifying Control Characters By Using the NOTGRAPH Function}

You can execute the following program to show the control characters that are identified by the NOTGRAPH function.

```sas
data test;  
  do dec=0 to 255;  
```

\begin{verbatim}
458 Chapter 2 • Dictionary of Functions and Call Routines for SAS and the CAS Server
If you use only one argument, NOTGRAPH begins the search at the beginning of the string. If you use two arguments, the absolute value of the second argument, \textit{start}, specifies the position at which to begin the search. The direction in which to search is determined in the following way:

- If the value of \textit{start} is positive, the search proceeds to the right.
- If the value of \textit{start} is negative, the search proceeds to the left.
- If the value of \textit{start} is less than the negative length of the string, the search begins at the end of the string.

NOTGRAPH returns a value of zero when one of the following is true:

- The character that you are searching for is not found.
- The value of \textit{start} is greater than the length of the string.
- The value of \textit{start} = 0.

\section*{Comparisons}

The NOTGRAPH function searches a character string for a non-graphical character. The ANYGRAPH function searches a character string for a graphical character.

\section*{Examples}

\subsection*{Example 1: Searching a String for Non-Graphical Characters}

The following example uses the NOTGRAPH function to search a string for a non-graphical character.

```sas
data _null_;  
  string='Next = _n_ + 12E3;';  
  j=0;  
  do until(j=0);  
    j=notgraph(string, j+1);  
    if j=0 then put +3 "That's all";  
    else do;  
      c=substr(string, j, 1);  
      put +3 j= c=;  
    end;  
  end;  
run;
```

SAS writes the following output to the log:

\begin{verbatim}
  j=5  c=  
  j=7  c=  
  j=11 c=  
  j=13 c=  
  That's all
\end{verbatim}

\subsection*{Example 2: Identifying Control Characters By Using the NOTGRAPH Function}

You can execute the following program to show the control characters that are identified by the NOTGRAPH function.

```sas
data test;  
  do dec=0 to 255;  
```
See Also

Functions:
- "ANYGRAPH Function" on page 89

NOTLOWER Function

Searches a character string for a character that is not a lowercase letter, and returns the first position at which that character is found.

**Syntax**

`NOTLOWER(string <, start>)`

**Required Argument**

`string` is the character constant, variable, or expression to search.

**Optional Argument**

`start` is an optional numeric constant, variable, or expression with an integer value that specifies the position at which the search should start and the direction in which to search.

**Details**

The NOTLOWER function searches a string for the first occurrence of any character that is not a lowercase letter. If such a character is found, NOTLOWER returns the position in the string of that character. If no such character is found, NOTLOWER returns a value of 0.

If you use only one argument, NOTLOWER begins the search at the beginning of the string. If you use two arguments, the absolute value of the second argument, `start`, specifies the position at which to begin the search. The direction in which to search is determined in the following way:
If the value of \textit{start} is positive, the search proceeds to the right.

If the value of \textit{start} is negative, the search proceeds to the left.

If the value of \textit{start} is less than the negative length of the string, the search begins at the end of the string.

\textbf{NOTLOWER} returns a value of zero when one of the following is true:

- The character that you are searching for is not found.
- The value of \textit{start} is greater than the length of the string.
- The value of \textit{start} = 0.

\textbf{Comparisons}

The \textbf{NOTLOWER} function searches a character string for a character that is not a lowercase letter. The \textbf{ANYLOWER} function searches a character string for a lowercase letter.

\textbf{Example}

The following example uses the \textbf{NOTLOWER} function to search a string for any character that is not a lowercase letter.

```sas
data _null_;
  string='Next = _n_ + 123E3;';
  j=0;
  do until(j=0);
    j=notlower(string, j+1);
    if j=0 then put +3 "That's all";
    else do;
      c=substr(string, j, 1);
      put +3 j= c=;
    end;
  end;
run;
```

SAS writes the following output to the log:

```
j=1  c=N  
j=5  c=  
j=6  c== 
j=7  c=  
j=8  c=  
j=10 c=  
j=11 c=  
j=12 c=+ 
j=13 c=  
j=14 c=1 
j=15 c=2 
j=16 c=E 
j=17 c=3 
j=18 c=;  
That's all
```

\textbf{See Also}

\textbf{Functions}:
NOTNAME Function

Searches a character string for an invalid character in a SAS variable name under VALIDVARNAME=V7, and returns the first position at which that character is found.

**Category:** Character

**Restriction:** This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility.

**Syntax**

\[
\text{NOTNAME}(\text{string} <, \text{start}>)
\]

**Required Argument**

\text{string}

is the character constant, variable, or expression to search.

**Optional Argument**

\text{start}

is an optional numeric constant, variable, or expression with an integer value that specifies the position at which the search should start and the direction in which to search.

**Details**

The NOTNAME function does not depend on the TRANTAB, ENCODING, or LOCALE system options.

The NOTNAME function searches a string for the first occurrence of any character that is not valid in a SAS variable name under VALIDVARNAME=V7. These characters are any except underscore (_), digits, and uppercase or lowercase English letters. If such a character is found, NOTNAME returns the position in the string of that character. If no such character is found, NOTNAME returns a value of 0.

If you use only one argument, NOTNAME begins the search at the beginning of the string. If you use two arguments, the absolute value of the second argument, \text{start}, specifies the position at which to begin the search. The direction in which to search is determined in the following way:

- If the value of \text{start} is positive, the search proceeds to the right.
- If the value of \text{start} is negative, the search proceeds to the left.
- If the value of \text{start} is less than the negative length of the string, the search begins at the end of the string.

NOTNAME returns a value of zero when one of the following is true:

- The character that you are searching for is not found.
- The value of \text{start} is greater than the length of the string.
- The value of \text{start} = 0.
Comparisons

The NOTNAME function searches a string for the first occurrence of any character that is not valid in a SAS variable name under VALIDVARNAME=V7. The ANYNAME function searches a string for the first occurrence of any character that is valid in a SAS variable name under VALIDVARNAME=V7.

Example

The following example uses the NOTNAME function to search a string for any character that is not valid in a SAS variable name under VALIDVARNAME=V7.

```sas
data _null_;  
   string='Next = _n_ + 12E3;'  
   j=0;  
   do until(j=0);  
      j=notname(string, j+1);  
      if j=0 then put +3 "That's all";  
      else do;  
         c=substr(string, j, 1);  
         put +3 j= c=;  
      end;  
   end;  
run;
```

SAS writes the following output to the log:

```
j=5 c=  
j=6 c==  
j=7 c=  
j=11 c=  
j=12 c+=  
j=13 c=  
j=18 c=;  
That's all
```

See Also

Functions:

- “ANYNAME Function” on page 93

NOTPRINT Function

Searches a character string for a nonprintable character, and returns the first position at which that character is found.

**Category:** Character

**Restriction:** This function is assigned an I18N Level 0 status, and is designed for SBCS data. Do not use this function to process DBCS or MBCS data. For more information, see Internationalization Compatibility.
Syntax

\texttt{NOTPRINT}(string <, \texttt{start}>)

\textbf{Required Argument}

\textit{string}

is the character constant, variable, or expression to search.

\textbf{Optional Argument}

\textit{start}

is an optional numeric constant, variable, or expression with an integer value that specifies the position at which the search should start and the direction in which to search.

\textbf{Details}

The NOTPRINT function searches a string for the first occurrence of a non-printable character. If such a character is found, NOTPRINT returns the position in the string of that character. If no such character is found, NOTPRINT returns a value of 0.

If you use only one argument, NOTPRINT begins the search at the beginning of the string. If you use two arguments, the absolute value of the second argument, \textit{start}, specifies the position at which to begin the search. The direction in which to search is determined in the following way:

- If the value of \textit{start} is positive, the search proceeds to the right.
- If the value of \textit{start} is negative, the search proceeds to the left.
- If the value of \textit{start} is less than the negative length of the string, the search begins at the end of the string.

NOTPRINT returns a value of zero when one of the following is true:

- The character that you are searching for is not found.
- The value of \textit{start} is greater than the length of the string.
- The value of \textit{start} = 0.

\textbf{Comparisons}

The NOTPRINT function searches a character string for a non-printable character. The ANYPRINT function searches a character string for a printable character.

\textbf{Example}

You can execute the following program to show the control characters that are identified by the NOTPRINT function.

```plaintext
data test;
do dec=0 to 255;
   byte=byte(dec);
   hex=put(dec, hex2.);
   notprint=notprint(byte);
   output;
end;
proc print data=test;
```
NOTPUNCT Function

Searches a character string for a character that is not a punctuation character, and returns the first position at which that character is found.

**Category:** Character

**Restriction:** This function is assigned an I18N Level 1 status. If possible, avoid I18N Level 1 functions if you are using a non-English language. Under certain circumstances, the I18N Level 1 functions might not work correctly with Double-Byte Character Set (DBCS) or Multi-Byte Character Set (MBCS) encodings. For more information, see Internationalization Compatibility.

**Syntax**

\[
\text{NOTPUNCT}(\text{string} <, \text{start}>)
\]

**Required Argument**

\text{string}  
is the character constant, variable, or expression to search.

**Optional Argument**

\text{start}  
is an optional numeric constant, variable, or expression with an integer value that specifies the position at which the search should start and the direction in which to search.

**Details**

The NOTPUNCT function searches a string for the first occurrence of a character that is not a punctuation character. If such a character is found, NOTPUNCT returns the position in the string of that character. If no such character is found, NOTPUNCT returns a value of 0.

If you use only one argument, NOTPUNCT begins the search at the beginning of the string. If you use two arguments, the absolute value of the second argument, \text{start}, specifies the position at which to begin the search. The direction in which to search is determined in the following way:

- If the value of \text{start} is positive, the search proceeds to the right.
- If the value of \text{start} is negative, the search proceeds to the left.
- If the value of \text{start} is less than the negative length of the string, the search begins at the end of the string.

NOTPUNCT returns a value of zero when one of the following is true:
• The character that you are searching for is not found.
• The value of start is greater than the length of the string.
• The value of start = 0.

Comparisons
The NOTPUNCT function searches a character string for a character that is not a punctuation character. The ANYPUNCT function searches a character string for a punctuation character.

Examples

Example 1: Searching a String for Characters That Are Not Punctuation Characters
The following example uses the NOTPUNCT function to search a string for characters that are not punctuation characters.

```
data _null_;  
  string='Next = _n_ + 12E3;';  
  j=0;  
  do until(j=0);  
    j=notpunct(string, j+1);  
    if j=0 then put +3 "That's all";  
    else do;  
      c=substr(string, j, 1);  
      put +3 j= c=;  
    end;  
  end;  
run;
```

SAS writes the following output to the log:

```
j=1 c=N  
j=2 c=e  
j=3 c=x  
j=4 c=t  
j=5 c=  
j=7 c=  
j=9 c=n  
j=11 c=  
j=13 c=  
j=14 c=1  
j=15 c=2  
j=16 c=E  
j=17 c=3  
That's all
```

Example 2: Identifying Control Characters By Using the NOTPUNCT Function
You can execute the following program to show the control characters that are identified by the NOTPUNCT function.

```
data test;  
  do dec=0 to 255;  
    byte=byte(dec);  
    hex=put(dec, hex2.);  
  end;  
run;
```
See Also

Functions:
• “ANYPUNCT Function” on page 97

NOTSPACE Function

Searches a character string for a character that is not a whitespace character (blank, horizontal and vertical tab, carriage return, line feed, and form feed), and returns the first position at which that character is found.

Category: Character
Restriction: This function is assigned an I18N Level 1 status. If possible, avoid I18N Level 1 functions if you are using a non-English language. Under certain circumstances, the I18N Level 1 functions might not work correctly with Double-Byte Character Set (DBCS) or Multi-Byte Character Set (MBCS) encodings. For more information, see Internationalization Compatibility.

Syntax

NOTSPACE(string <, start>)

Required Argument

string
is the character constant, variable, or expression to search.

Optional Argument

start
is an optional numeric constant, variable, or expression with an integer value that specifies the position at which the search should start and the direction in which to search.

Details

The NOTSPACE function searches a string for the first occurrence of a character that is not a blank, horizontal tab, vertical tab, carriage return, line feed, or form feed. If such a character is found, NOTSPACE returns the position in the string of that character. If no such character is found, NOTSPACE returns a value of 0.

If you use only one argument, NOTSPACE begins the search at the beginning of the string. If you use two arguments, the absolute value of the second argument, start, specifies the position at which to begin the search. The direction in which to search is determined in the following way:

• If the value of start is positive, the search proceeds to the right.
• If the value of $start$ is negative, the search proceeds to the left.
• If the value of $start$ is less than the negative length of the string, the search begins at the end of the string.

NOTSPACE returns a value of zero when one of the following is true:
• The character that you are searching for is not found.
• The value of $start$ is greater than the length of the string.
• The value of $start = 0$.

Comparisons

The NOTSPACE function searches a character string for the first occurrence of a character that is not a blank, horizontal tab, vertical tab, carriage return, line feed, or form feed. The ANYSPACE function searches a character string for the first occurrence of a character that is a blank, horizontal tab, vertical tab, carriage return, line feed, or form feed.

Examples

Example 1: Searching a String for a Character That Is Not a Whitespace Character

The following example uses the NOTSPACE function to search a string for a character that is not a whitespace character.

```sas
data _null_;
  string='Next = _n_ + 12E3;';
  j=0;
  do until(j=0);
    j=notspace(string, j+1);
    if j=0 then put +3 "That's all";
    else do;
      c=substr(string, j, 1);
      put +3 j= c=;
    end;
  end;
run;
```

SAS writes the following output to the log:

```
j=1 c=N
j=2 c=e
j=3 c=x
j=4 c=t
j=6 c=*
j=8 c=_
j=9 c=n
j=10 c=_
j=12 c=+
j=14 c=1
j=15 c=2
j=16 c=E
j=17 c=3
j=18 c=;
That's all
```
Example 2: Identifying Control Characters By Using the NOTSPACE Function

You can execute the following program to show the control characters that are identified by the NOTSPACE function.

```sas
data test;
  do dec=0 to 255;
    byte=byte(dec);
    hex=put(dec, hex2.);
    notspace=notspace(byte);
    output;
  end;
  proc print data=test;
  run;
```

See Also

Functions:
- “ANYSPACE Function” on page 99

NOTUPPER Function

Searches a character string for a character that is not an uppercase letter, and returns the first position at which that character is found.

- **Category:** Character
- **Restriction:** This function is assigned an I18N Level 1 status. If possible, avoid I18N Level 1 functions if you are using a non-English language. Under certain circumstances, the I18N Level 1 functions might not work correctly with Double-Byte Character Set (DBCS) or Multi-Byte Character Set (MBCS) encodings. For more information, see Internationalization Compatibility.

**Syntax**

```
NOTUPPER(string <, start>)
```

**Required Argument**

- `string` is the character constant, variable, or expression to search.

**Optional Argument**

- `start` is an optional numeric constant, variable, or expression with an integer value that specifies the position at which the search should start and the direction in which to search.

**Details**

The NOTUPPER function searches a string for the first occurrence of a character that is not an uppercase letter. If such a character is found, NOTUPPER returns the position in
the string of that character. If no such character is found, NOTUPPER returns a value of 0.

If you use only one argument, NOTUPPER begins the search at the beginning of the string. If you use two arguments, the absolute value of the second argument, start, specifies the position at which to begin the search. The direction in which to search is determined in the following way:

- If the value of start is positive, the search proceeds to the right.
- If the value of start is negative, the search proceeds to the left.
- If the value of start is less than the negative length of the string, the search begins at the end of the string.

NOTUPPER returns a value of zero when one of the following is true:

- The character that you are searching for is not found.
- The value of start is greater than the length of the string.
- The value of start = 0.

**Comparisons**

The NOTUPPER function searches a character string for a character that is not an uppercase letter. The ANYUPPER function searches a character string for an uppercase letter.

**Example**

The following example uses the NOTUPPER function to search a string for any character that is not an uppercase letter.

```sas
data _null_;  
  string='Next = _n_ + 12E3;';  
  j=0;  
  do until(j=0);  
    j=notupper(string, j+1);  
    if j=0 then put +3 "That's all";  
    else do;  
      c=substr(string, j, 1);  
      put +3 j= c=;  
    end;  
  end;  
run;
```
SAS writes the following output to the log:

```
j=2 c=e
j=3 c=x
j=4 c=t
j=5 c=
j=6 c==
j=7 c=
j=8 c=_
j=9 c=n
j=10 c=_
j=11 c=
j=12 c=+
j=13 c=
j=14 c=1
j=15 c=2
j=17 c=3
j=18 c=;
That's all
```

See Also

Functions:

- “ANYUPPER Function” on page 101

**NOTXDIGIT Function**

Searches a character string for a character that is not a hexadecimal character, and returns the first position at which that character is found.

- **Category:** Character
- **Restriction:** This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility.

**Syntax**

```
NOTXDIGIT(string <, start>)
```

**Required Argument**

`string`

is the character constant, variable, or expression to search.

**Optional Argument**

`start`

is an optional numeric constant, variable, or expression with an integer value that specifies the position at which the search should start and the direction in which to search.
Details

The NOTXDIGIT function searches a string for the first occurrence of any character that is not a digit or an uppercase or lowercase A, B, C, D, E, or F. If such a character is found, NOTXDIGIT returns the position in the string of that character. If no such character is found, NOTXDIGIT returns a value of 0.

If you use only one argument, NOTXDIGIT begins the search at the beginning of the string. If you use two arguments, the absolute value of the second argument, start, specifies the position at which to begin the search. The direction in which to search is determined in the following way:

- If the value of start is positive, the search proceeds to the right.
- If the value of start is negative, the search proceeds to the left.
- If the value of start is less than the negative length of the string, the search begins at the end of the string.

NOTXDIGIT returns a value of zero when one of the following is true:

- The character that you are searching for is not found.
- The value of start is greater than the length of the string.
- The value of start = 0.

Comparisons

The NOTXDIGIT function searches a character string for a character that is not a hexadecimal character. The ANYXDIGIT function searches a character string for a character that is a hexadecimal character.

Example

The following example uses the NOTXDIGIT function to search a string for a character that is not a hexadecimal character.

```plaintext
data _null_;  
string='Next = _n_ + 12E3;';  
j=0;  
do until(j=0);  
  j=notxdigit(string, j+1);  
  if j=0 then put +3 "That's all";  
  else do;  
    c=substr(string, j, 1);  
    put +3 j= c=;  
  end;  
end;  
run;
```
SAS writes the following output to the log:

```
j=1 c=N
j=3 c=X
j=4 c=t
j=5 c=
j=6 c==
j=7 c=
j=8 c=_
j=9 c=n
j=10 c=_
j=11 c=
j=12 c=+
j=13 c=
j=18 c=;
That's all
```

See Also

**Functions:**

- “ANYXDIGIT Function” on page 102

---

**NPV Function**

Returns the net present value with the rate expressed as a percentage.

**Category:** Financial

**Syntax**

\[
NPV(r, frequency, c0, c1, \ldots, cn)
\]

**Required Arguments**

- **r**
  
is numeric, the interest rate over a specified base period of time expressed as a percentage.

- **frequency**
  
is numeric, the number of payments during the base period of time specified with the rate \( r \).

  **Range** \( frequency > 0 \)

  **Note** The case \( frequency = 0 \) is a flag to allow continuous discounting.

- **c0, c1, \ldots, cn**
  
are numeric cash flows that represent cash outlays (payments) or cash inflows (income) occurring at times 0, 1, \ldots, n. These cash flows are assumed to be equally spaced, beginning-of-period values. Negative values represent payments, positive values represent income, and values of 0 represent no cash flow at a given time. The \( c0 \) argument and the \( c1 \) argument are required.
Comparisons
The NPV function is identical to NETPV, except that the \( r \) argument is provided as a percentage.

### NWKDOM Function

Returns the date for the \( n \)th occurrence of a weekday for the specified month and year.

#### Category:
Date and Time

#### Syntax

\[
\text{NWKDOM}(n, \text{weekday}, \text{month}, \text{year})
\]

#### Required Arguments

- **\( n \)**
  - specifies the numeric week of the month that contains the specified day.
  - Range: 1–5
  - Tip: \( N=5 \) indicates that the specified day occurs in the last week of that month. Sometimes \( n=4 \) and \( n=5 \) produce the same results.

- **\( \text{weekday} \)**
  - specifies the number that corresponds to the day of the week.
  - Range: 1–7
  - Tip: Sunday is considered the first day of the week and has a weekday value of 1.

- **\( \text{month} \)**
  - specifies the number that corresponds to the month of the year.
  - Range: 1–12

- **\( \text{year} \)**
  - specifies a four-digit calendar year.

#### Details

The NWKDOM function returns a SAS date value for the \( n \)th weekday of the month and year that you specify. Use any valid SAS date format, such as the DATE9. format, to display a calendar date. You can specify \( n=5 \) for the last occurrence of a particular weekday in the month.

Sometimes \( n=5 \) and \( n=4 \) produce the same result. These results occur when there are only four occurrences of the requested weekday in the month. For example, if the month of January begins on a Sunday, there will be five occurrences of Sunday, Monday, and Tuesday, but only four occurrences of Wednesday, Thursday, Friday, and Saturday. In this case, specifying \( n=5 \) or \( n=4 \) for Wednesday, Thursday, Friday, or Saturday will produce the same result.

If the year is not a leap year, February has 28 days and there are four occurrences of each day of the week. In this case, \( n=5 \) and \( n=4 \) produce the same results for every day.
Comparisons

In the NWKDOM function, the value for weekday corresponds to the numeric day of the week beginning on Sunday. This value is the same value that is used in the WEEKDAY function, where Sunday =1, and so on. The value for month corresponds to the numeric month of the year beginning in January. This value is the same value that is used in the MONTH function, where January =1, and so on.

You can use the NWKDOM function to calculate events that are not defined by the HOLIDAY function. For example, if a university always schedules graduation on the first Saturday in June, then you can use the following statement to calculate the date:

```
UnivGrad=nwkdom(1, 7, 6, year);
```

Examples

Example 1: Returning Date Values

The following example uses the NWKDOM function and returns the date for specific occurrences of a weekday for a specified month and year.

```sas
data _null_; /* Return the date of the third Monday in May 2012. */
a=nwkdom(3, 2, 5, 2012);
/* Return the date of the fourth Wednesday in November 2012. */
b=nwkdom(4, 4, 11, 2012);
/* Return the date of the fourth Saturday in November 2012. */
c=nwkdom(4, 7, 11, 2012);
/* Return the date of the first Sunday in January 2013. */
d=nwkdom(1, 1, 1, 2013);
/* Return the date of the second Tuesday in September 2012. */
e=nwkdom(2, 3, 9, 2012);
/* Return the date of the fifth Thursday in December 2012. */
f=nwkdom(5, 5, 12, 2012);
put a= weekdatx.;
put b= weekdatx.;
put c= weekdatx.;
put d= weekdatx.;
put e= weekdatx.;
put f= weekdatx.;
run;
```

SAS writes the following output to the log:

```
a=Monday, 21 May 2012
b=Wednesday, 28 November 2012
c=Saturday, 24 November 2012
d=Sunday, 6 January 2013
e=Tuesday, 11 September 2012
f=Thursday, 27 December 2012
```

Example 2: Returning the Date of the Last Monday in May

The following example returns the date that corresponds to the last Monday in the month of May in the year 2012.

```sas
data _null_; /* The last Monday in May. */
x=nwkdom(5, 2, 5, 2012);
```
ORDINAL Function

Returns the kth smallest of the missing and nonmissing values.

Category: Descriptive Statistics

Syntax

ORDINAL(k, argument-1, argument-2 <, …argument-n>)

Required Arguments

k

is a numeric constant, variable, or expression with an integer value that is less than or equal to the number of subsequent elements in the list of arguments.

argument

specifies a numeric constant, variable, or expression. At least two arguments are required. An argument can consist of a variable list, preceded by OF.

Details

The ORDINAL function returns the kth smallest value, either missing or nonmissing, among the second through the last arguments.

Comparisons

The ORDINAL function counts both missing and nonmissing values, whereas the SMALLEST function counts only nonmissing values.

See Also

Functions:

- “HOLIDAY Function” on page 323
- “INTNX Function” on page 370
- “MONTH Function” on page 442
- “WEEKDAY Function” on page 687
Example

The following SAS statement produces this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1=ordinal(4, 1, 2, 3, -4, 5, 6, 7);</td>
<td>3</td>
</tr>
</tbody>
</table>

PCTL Function

Returns the percentile that corresponds to the percentage.

**Category:** Descriptive Statistics

**Syntax**

\[
\text{PCTL}\langle n\rangle (\text{percentage}, \text{value–1} <, \text{value–2}, ...)\]

**Required Arguments**

- **percentage** is a numeric constant, variable, or expression that specifies the percentile to be computed.
  
  **Requirement** is numeric where, \(0 \leq \text{percentage} \leq 100\).

- **value** is a numeric variable, constant, or expression.

**Optional Argument**

- **n** is a digit from 1 to 5 which specifies the definition of the percentile to be computed.
  
  **Default** definition 5

**Details**

The PCTL function returns the percentile of the nonmissing values corresponding to the percentage. If \(\text{percentage}\) is missing, less than zero, or greater than 100, the PCTL function generates an error message.

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>lower_quartile=PCTL(25, 2, 4, 1, 3);</td>
<td>1.5</td>
</tr>
<tr>
<td>put lower_quartile;</td>
<td></td>
</tr>
</tbody>
</table>
### PDF Function

Returns a value from a probability density (mass) distribution.

<table>
<thead>
<tr>
<th>Category:</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias:</td>
<td>PMF</td>
</tr>
</tbody>
</table>

#### Syntax

PDF\((distribution, quantile <, parameter-1, ... , parameter-k>)\)

#### Required Arguments

- **distribution**
  - is a character constant, variable, or expression that identifies the distribution. Valid distributions are as follows:

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bernoulli</td>
<td>Bernoulli</td>
</tr>
<tr>
<td>Beta</td>
<td>BETA</td>
</tr>
<tr>
<td>Binomial</td>
<td>BINOMIAL</td>
</tr>
<tr>
<td>Cauchy</td>
<td>CAUCHY</td>
</tr>
<tr>
<td>Chi-Square</td>
<td>CHISQUARE</td>
</tr>
<tr>
<td>Distribution</td>
<td>Argument</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>Conway-Maxwell-Poisson</td>
<td>CONMAXPOI</td>
</tr>
<tr>
<td>Exponential</td>
<td>EXPONENTIAL</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>Gamma</td>
<td>GAMMA</td>
</tr>
<tr>
<td>Generalized Poisson</td>
<td>GENPOISSON</td>
</tr>
<tr>
<td>Geometric</td>
<td>GEOMETRIC</td>
</tr>
<tr>
<td>Hypergeometric</td>
<td>HYPERGEOMETRIC</td>
</tr>
<tr>
<td>Laplace</td>
<td>LAPLACE</td>
</tr>
<tr>
<td>Logistic</td>
<td>LOGISTIC</td>
</tr>
<tr>
<td>Lognormal</td>
<td>LOGNORMAL</td>
</tr>
<tr>
<td>Negative binomial</td>
<td>NEGBINOMIAL</td>
</tr>
<tr>
<td>Normal</td>
<td>NORMAL</td>
</tr>
<tr>
<td>Normal mixture</td>
<td>NORMALMIX</td>
</tr>
<tr>
<td>Pareto</td>
<td>PARETO</td>
</tr>
<tr>
<td>Poisson</td>
<td>POISSON</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>Tweedie</td>
<td>TWEEDIE</td>
</tr>
<tr>
<td>Uniform</td>
<td>UNIFORM</td>
</tr>
<tr>
<td>Wald (inverse Gaussian)</td>
<td>WALD</td>
</tr>
<tr>
<td>Weibull</td>
<td>WEIBULL</td>
</tr>
</tbody>
</table>

**Note**  
Except for T, F, and NORMALMIX, you can minimally identify any distribution by its first four characters.

**quantile**  
is a numeric constant, variable, or expression that specifies the value of the random variable.
Optional Argument

*parameter-1, ..., parameter-k*

are optional numeric constants, variables, or expressions that specify the values of shape, location, or scale parameters that are appropriate for the specific distribution.

See “Details” on page 479 for complete information about these parameters.

Details

**Bernoulli Distribution**

PDF (`BERNOULLI', x, p)

Arguments

- `x` is a numeric constant, variable, or expression that specifies a random variable.
- `p` is a numeric constant, variable, or expression that specifies the probability of success.

Range \( 0 \leq p \leq 1 \)

Details

The PDF function for the Bernoulli distribution returns the probability density function of a Bernoulli distribution, with the probability of success equal to `p`. The PDF function is evaluated at the value `x`. The equation follows:

\[
PDF(BERN', x, p) = \begin{cases} 
0 & x < 0 \\
1 - p & x = 0 \\
0 & 0 < x < 1 \\
p & x = 1 \\
0 & x > 1 
\end{cases}
\]

**Note:** There are no location or scale parameters for this distribution.

**Beta Distribution**

PDF (`BETA', x, a, b < l, r>)

Arguments

- `x` is a numeric constant, variable, or expression that specifies a random variable.
- `a` is a numeric constant, variable, or expression that specifies a shape parameter.

Range \( a > 0 \)

- `b` is a numeric constant, variable, or expression that specifies a shape parameter.

Range \( b > 0 \)
$l$

is a numeric constant, variable, or expression that specifies the left location parameter.

Default 0

$r$

is a numeric constant, variable, or expression that specifies the right location parameter.

Default 1

Range $r > l$

Details

The PDF function for the beta distribution returns the probability density function of a beta distribution, with the shape parameters $a$ and $b$. The PDF function is evaluated at the value $x$. The equation follows:

$$
PDF('BETA', x, a, b, l, r) = \begin{cases} 
0 & x < l \\
\frac{1}{\beta(a, b)} \frac{(x - l)^{a-1}(r - x)^{b-1}}{(r - l)^{a+b-1}} & l \leq x \leq r \\
0 & x > r 
\end{cases}
$$

Note: The quantity $\frac{x - l}{r - l}$ is forced to be $\varepsilon \leq \frac{x - l}{r - l} \leq 1 - 2\varepsilon$.

**Binomial Distribution**

PDF ('BINOMIAL', $m$, $p$, $n$)

**Arguments**

$m$

is an integer random variable that counts the number of successes.

Range $m = 0, 1, ...$

$p$

is a numeric constant, variable, or expression that specifies the probability of success.

Range $0 \leq p \leq 1$

$n$

is an integer parameter that counts the number of independent Bernoulli trials.

Range $n = 0, 1, ...$

Details

The PDF function for the binomial distribution returns the probability density function of a binomial distribution, with the parameters $p$ and $n$, which is evaluated at the value $m$. The equation follows:
PDF(′BINOM′, m, p, n) = \[
\begin{cases}
0 & m < 0 \\
\binom{n}{m} p^m (1 - p)^{n-m} & 0 \leq m \leq n \\
0 & m > n
\end{cases}
\]

Note: There are no location or scale parameters for the binomial distribution.

**Cauchy Distribution**

PDF (′CAUCHY′, x <, θ, λ>)

**Arguments**

\(x\)

is a numeric constant, variable, or expression that specifies a random variable.

\(θ\)

is a numeric constant, variable, or expression that specifies a location parameter.

Default 0

\(λ\)

is a numeric constant, variable, or expression that specifies a scale parameter.

Default 1

Range \(λ > 0\)

**Details**

The PDF function for the Cauchy distribution returns the probability density function of a Cauchy distribution, with the location parameter \(θ\) and the scale parameter \(λ\). The PDF function is evaluated at the value \(x\). The equation follows:

\[
PDF(′CAUCHY′, x, θ, λ) = \frac{1}{\pi} \left( \frac{λ}{λ^2 + (x - θ)^2} \right)
\]

**Chi-Square Distribution**

PDF (′CHISQUARE′, x, df <, nc>)

**Arguments**

\(x\)

is a numeric constant, variable, or expression that specifies a random variable.

\(df\)

is a numeric constant, variable, or expression that specifies the degrees of freedom.

Range \(df > 0\)

\(nc\)

is a numeric constant, variable, or expression that specifies an optional noncentrality parameter.

Range \(nc \geq 0\)

**Details**
The PDF function for the chi-square distribution returns the probability density function of a chi-square distribution, with \( df \) degrees of freedom and the noncentrality parameter \( nc \). The PDF function is evaluated at the value \( x \). This function accepts noninteger degrees of freedom. If \( nc \) is omitted or equal to zero, the value returned is from the central chi-square distribution. The following equation describes the PDF function for the chi-square distribution:

\[
PDF('CHISQ', x, \nu, \lambda) = \begin{cases} 
0 & x < 0 \\
\sum_{j=0}^{\infty} e^{-\frac{\lambda}{2}} \frac{\lambda^{j}}{j!} P_c(x, v + 2j) & x \geq 0 
\end{cases}
\]

In the equation, \( p_c(\cdot, \cdot) \) denotes the density from the central chi-square distribution:

\[
p_c(x, a) = \frac{1}{2^a b} p_g \left( x, \frac{a}{2}, \frac{b}{2} \right)
\]

In the equation, \( p_g(\cdot, \cdot) \) is the density from the gamma distribution, which is given by the following equation:

\[
p_g(y, b) = \frac{1}{\Gamma(b)} e^{-y} y^{b-1}
\]

**Conway-Maxwell-Poisson Distribution**

PDF('CONMAXPOI', y, \lambda, \nu)

**Arguments**

- \( y \) is a numeric constant, variable, or expression that specifies a nonnegative integer representing a count.
- \( \lambda \) is a numeric constant, variable, or expression that specifies a location parameter, similar to the Poisson mean parameter.
- \( \nu \) is a numeric constant, variable, or expression that specifies a dispersion parameter.

**Details**

The Conway-Maxwell-Poisson (CMP) distribution is a generalization of the Poisson distribution that enables you to model underdispersed and overdispersed data. The CMP distribution is defined according to the following equation:

\[
P(Y = y; \lambda, \nu) = \frac{1}{Z(\lambda, \nu) (\nu!)^y} \lambda^y \quad y = 0, 1, 2, ...
\]

The normalization factor is expressed by this equation:

\[
Z(\lambda, \nu) = \sum_{n=0}^{\infty} \frac{\lambda^n}{(n!)^\nu}
\]

\( \lambda \) and \( \nu \) are nonnegative and not simultaneously zero.
The introduction of the additional parameter, \( \nu \), allows for flexibility in modeling the tail behavior of the distribution. If \( \nu = 1 \), the ratio is equal to the rate of decay of the Poisson distribution. If \( \nu < 1 \), the rate of decay decreases, enabling you to model processes that have longer tails than the Poisson distribution (overdispersed data). If \( \nu > 1 \), the rate of decay increases in a nonlinear manner, thus shortening the tail of the distribution (underdispersed data).

There are several special cases of the Conway-Maxwell-Poisson distribution. If \( \lambda < 1 \) and \( \nu \to \infty \), the Conway-Maxwell-Poisson distribution results in the Bernoulli distribution. In this case, the data can take only the values 0 and 1, which represent an extreme underdispersion. If \( \nu = 1 \), the Poisson distribution is recovered with its equidispersion property. When \( \nu = 0 \) and \( \lambda < 1 \), the normalization factor is convergent and forms the following geometric series:

\[
Z(\lambda, 0) = \frac{1}{1 - \lambda}
\]

The probability density function is represented by this equation:

\[
P(Y = y ; \lambda, \nu = 0) = (1 - \lambda)^y
\]

The geometric distribution represents a case of severe overdispersion.

**Mean, Variance, and Dispersion for the Conway-Maxwell-Poisson Model**

The mean and the variance of the Conway-Maxwell-Poisson distribution are defined by the following equations:

\[
E[Y] = \frac{\partial \ln Z}{\partial \ln \lambda}
\]

\[
V[Y] = \frac{\partial^2 \ln Z}{\partial^2 \ln \lambda}
\]

The Conway-Maxwell-Poisson distribution does not have closed-form expressions for its moments in terms of its parameters \( \lambda \) and \( \nu \). However, the moments can be approximated. (For more information about the Conway-Maxwell-Poisson distribution and discrete data, see the References section that is located at the end of this function.) Use asymptotic expressions for \( Z \) to derive \( E(Y) \) and \( V(Y) \) as the following equations show:

\[
E[Y] \approx \lambda^{1/\nu} + \frac{1}{2\nu} - \frac{1}{2}
\]

\[
V[Y] \approx \frac{1}{\nu} \lambda^{1/\nu}
\]

In the Conway-Maxwell-Poisson model, the summation of infinite series is evaluated using a logarithmic expansion. (For more information about the Conway-Maxwell-Poisson distribution and discrete data, see the References section that is located at the end of this function.) The mean and variance are calculated as follows for the Conway-Maxwell-Poisson model:
\[ E\left( Y \right) = \frac{1}{Z(\lambda, v)} \sum_{j=0}^{\infty} j^{\lambda j} / (Y!)^v \]

\[ V\left( Y \right) = \frac{1}{Z(\lambda, v)} \sum_{j=0}^{\infty} j^{2 \lambda j} / (Y!)^v - E(Y)^2 \]

The dispersion is defined as follows:

\[ D\left( Y \right) = \frac{V(Y)}{E(Y)} \]

**Exponential Distribution**

PDF ('EXPONENTIAL', \( x <, \lambda >\))

**Arguments**

\( x \)

is a numeric constant, variable, or expression that specifies a random variable.

\( \lambda \)

is a numeric constant, variable, or expression that specifies a scale parameter.

**Default** 1

**Range** \( \lambda > 0 \)

**Details**

The PDF function for the exponential distribution returns the probability density function of an exponential distribution, with the scale parameter \( \lambda \). The PDF function is evaluated at the value \( x \). The equation follows:

\[ PDF('EXPO', x, \lambda) = \begin{cases} 0 & x < 0 \\ \frac{1}{\lambda} \exp\left( -\frac{x}{\lambda} \right) & x \geq 0 \end{cases} \]

**F Distribution**

PDF ('F', \( x, ndf, ddf <, nc>\))

**Arguments**

\( x \)

is a numeric constant, variable, or expression that specifies a random variable.

\( ndf \)

is a numeric constant, variable, or expression that specifies the numerator degrees of freedom.

**Range** \( ndf \geq 0 \)

\( ddf \)

is a numeric constant, variable, or expression that specifies the denominator degrees of freedom.
nc is a numeric constant, variable, or expression that specifies an optional noncentrality parameter.

\[ \text{Range } \text{nc} \geq 0 \]

**Details**

The PDF function for the F distribution returns the probability density function of an F distribution, with \( ndf \) numerator degrees of freedom, \( ddf \) denominator degrees of freedom, and the noncentrality parameter \( \text{nc} \). The PDF function is evaluated at the value \( x \). This PDF function accepts noninteger degrees of freedom for \( ndf \) and \( ddf \). If \( \text{nc} \) is omitted or equal to zero, the value returned is from a central F distribution. In the following equation, let \( \nu_1 = ndf \), let \( \nu_2 = ddf \), and let \( \lambda = \text{nc} \). The following equation describes the PDF function for the F distribution:

\[
PDF(F', x, \nu_1, \nu_2, \lambda) = \begin{cases} 
0 & x < 0 \\
\sum_{j=0}^{\infty} e^{-\frac{\lambda}{2}} \left( \frac{\lambda}{2} \right)^j \frac{j!}{j!} p_f(f, \nu_1 + 2j, \nu_2) & x \geq 0 
\end{cases}
\]

In the equation, \( p_f(f, u_1, u_2) \) is the density from the central F distribution:

\[
p_f(f, u_1, u_2) = \frac{u_1^u_2}{(u_1 + u_2)^{u_1 + u_2}} \left( \frac{u_1}{2} \right)^{u_1 / 2} \left( \frac{u_2}{2} \right)^{u_2 / 2}
\]

In the equation \( p_b(x, a, b) \) is the density from the standard beta distribution.

*Note:* There are no location or scale parameters for the F distribution.

**Gamma Distribution**

PDF ('GAMMA', \( x, a <, \lambda > \))

**Arguments**

\( x \)

is a numeric constant, variable, or expression that specifies a random variable.

\( a \)

is a numeric constant, variable, or expression that specifies a shape parameter.

\[ \text{Range } a > 0 \]

\( \lambda \)

is a numeric constant, variable, or expression that specifies a scale parameter.

**Details**

The PDF function for the gamma distribution returns the probability density function of a gamma distribution, with the shape parameter \( a \) and the scale parameter \( \lambda \). The PDF function is evaluated at the value \( x \). The equation follows:
Generalized Poisson Distribution

PDF ('GENPOISSON', x, θ, η)

Arguments

x
- is a numeric constant, variable, or expression that specifies an integer random variable.

θ
- is a numeric constant, variable, or expression that specifies a shape parameter.
  Range <10^5 and >0

η
- is a numeric constant, variable, or expression that specifies a shape parameter.
  Range ≥0 and <0.95

Tip
- When η = 0, the distribution is the Poisson distribution with a mean and variance of θ. When η > 0, the mean is \( \theta \div (1 - \eta) \) and the variance is \( \frac{\theta}{(1 - \eta)^3} \).

Details

The probability mass function for the generalized Poisson distribution follows:

\[
f(x; \theta, \eta) = \theta(\theta + \eta x)^{x-1} e^{-\theta - \eta x} / x!, \quad x = 0, 1, 2, ..., \quad \theta > 0, \ 0 \leq \eta < 1
\]

Geometric Distribution

PDF ('GEOMETRIC', m, p)

Arguments

m
- is a numeric constant, variable, or expression that specifies the number of failures before the first success.
  Range \( m \geq 0 \)

p
- is a numeric constant, variable, or expression that specifies a probability of success.
  Range \( 0 \leq p \leq 1 \)

Details

The PDF function for the geometric distribution returns the probability density function of a geometric distribution, with the parameter p. The PDF function is evaluated at the value m. The equation follows:
PDF(GEOM', m, p) = \begin{cases} 
0 & m < 0 \\
p(1 - p)^m & m \geq 0
\end{cases}

Note: There are no location or scale parameters for this distribution.

**Hypergeometric Distribution**

PDF ('HYPER', x, N, R, n <, o>)

**Arguments**

x
- is a numeric constant, variable, or expression that specifies an integer random variable.

N
- is a numeric constant, variable, or expression that specifies an integer population size parameter.
  
  Range \( N = 1, 2, ... \)

R
- is a numeric constant, variable, or expression that specifies an integer number of items in the category of interest.
  
  Range \( R = 0, 1, ..., N \)

n
- is a numeric constant, variable, or expression that specifies an integer sample size parameter.
  
  Range \( n = 1, 2, ..., N \)

o
- is a numeric constant, variable, or expression that specifies an optional odds ratio parameter.
  
  Range \( o > 0 \)

**Details**

The PDF function for the hypergeometric distribution returns the probability density function of an extended hypergeometric distribution, with population size \( N \), number of items \( R \), sample size \( n \), and odds ratio \( o \). The PDF function is evaluated at the value \( x \). If \( o \) is omitted or equal to 1, the value returned is from the usual hypergeometric distribution. The equation follows:

\[
PDF \ ('HYPER', x, N, R, n, o) = \begin{cases} 
0 & x < \max(0, R + n - N) \\
\frac{(R)_{(N-R)}x^x}{\sum_{j=\max(0, R+n-N)}^{\min(R,n)} \binom{R}{j} \binom{N-R}{n-j} o^j} & \max(0, R + n - N) \leq x \leq \min(R, n) \\
0 & x > \min(R, n)
\end{cases}
\]
Laplace Distribution

PDF(‘LAPLACE’, x <, θ, λ>)

Arguments

x
is a numeric constant, variable, or expression that specifies a random variable.

θ
is a numeric constant, variable, or expression that specifies a location parameter.
Default 0

λ
is a numeric constant, variable, or expression that specifies a scale parameter.
Default 1

Range λ > 0

Details

The PDF function for the Laplace distribution returns the probability density function of the Laplace distribution, with the location parameter θ and the scale parameter λ. The PDF function is evaluated at the value x. The equation follows:

$$PDF\left(‘LAPLACE’\,\, x, \theta, \lambda\right) = \frac{1}{2\lambda} \exp\left(-\frac{|x - \theta|}{\lambda}\right)$$

Logistic Distribution

PDF(‘LOGISTIC’, x <, θ, λ>)

Arguments

x
is a numeric constant, variable, or expression that specifies a random variable.

θ
is a numeric constant, variable, or expression that specifies a location parameter.
Default 0

λ
is a numeric constant, variable, or expression that specifies a scale parameter.
Default 1

Range λ > 0

Details

The PDF function for the logistic distribution returns the probability density function of a logistic distribution, with the location parameter θ and the scale parameter λ. The PDF function is evaluated at the value x. The equation follows:

$$PDF\left(‘LOGISTIC’\,\, x, \theta, \lambda\right) = \frac{\exp\left(-\frac{x - \theta}{\lambda}\right)}{\lambda \left(1 + \exp\left(-\frac{x - \theta}{\lambda}\right)\right)^2}$$
Lognormal Distribution
PDF ('LOGNORMAL', \(x < \theta, \lambda\))

Arguments
\(x\)
- is a numeric constant, variable, or expression that specifies a random variable.

\(\theta\)
- is a numeric constant, variable, or expression that specifies a log scale parameter.
\(\exp(\theta)\) is a scale parameter.
Default \(0\)

\(\lambda\)
- is a numeric constant, variable, or expression that specifies a shape parameter.
Default \(1\)

Details
The PDF function for the lognormal distribution returns the probability density function of a lognormal distribution, with the log scale parameter \(\theta\) and the shape parameter \(\lambda\). The PDF function is evaluated at the value \(x\). The equation follows:

\[
PDF('LOGN', x, \theta, \lambda) = \begin{cases} 
0 & x \leq 0 \\
\frac{1}{x\lambda \sqrt{2\pi}} \exp\left(-\frac{(\log(x) - \theta)^2}{2\lambda^2}\right) & x > 0 
\end{cases}
\]

Negative Binomial Distribution
PDF ('NEGBINOMIAL', \(m, p, n\))

Arguments
\(m\)
- is a numeric constant, variable, or expression that specifies a positive integer random variable that counts the number of failures.
Range \(m = 0, 1, ...\)

\(p\)
- is a numeric constant, variable, or expression that specifies a probability of success.
Range \(0 \leq p \leq 1\)

\(n\)
- is a numeric constant, variable, or expression that specifies a value that counts the number of successes.
Range \(n > 0\)

Details
The PDF function for the negative binomial distribution returns the probability density function of a negative binomial distribution, with probability of success \(p\) and number of successes \(n\). The PDF function is evaluated at the value \(m\). The equation follows:
\[
PDF(\text{NEGBI}, m, p, n) = \begin{cases} 
0 & m < 0 \\
\binom{n + m - 1}{n - 1} p^n (1 - p)^m & m \geq 0
\end{cases}
\]

Note: There are no location or scale parameters for the negative binomial distribution.

**Normal Distribution**

PDF(\text{NORMAL}', x, \theta, \lambda)

**Arguments**

- \(x\)
  - is a numeric constant, variable, or expression that specifies a random variable.
- \(\theta\)
  - is a numeric constant, variable, or expression that specifies a location parameter.
  - Default: 0
- \(\lambda\)
  - is a numeric constant, variable, or expression that specifies a scale parameter.
  - Default: 1
  - Range: \(\lambda > 0\)

**Details**

The PDF function for the normal distribution returns the probability density function of a normal distribution, with the location parameter \(\theta\) and the scale parameter \(\lambda\). The PDF function is evaluated at the value \(x\). The equation follows:

\[
PDF(\text{NORMAL}', x, \theta, \lambda) = \frac{1}{\lambda \sqrt{2\pi}} \exp\left( -\frac{(x - \theta)^2}{2\lambda^2} \right)
\]

**Normal Mixture Distribution**

PDF(\text{NORMALMIX}', x, n, p_1, p_2, \ldots, p_n, m_1, m_2, \ldots, m_n, s_1, s_2, \ldots, s_n)

**Arguments**

- \(x\)
  - is a numeric constant, variable, or expression that specifies a random variable.
- \(n\)
  - is a numeric constant, variable, or expression that specifies the integer number of mixtures.
  - Range: \(n = 1, 2, \ldots\)
- \(p_i\)
  - is a list of numeric constants, variables, or expressions that specifies the \(n\) proportions, \(p_1, p_2, \ldots, p_n\), where \(\sum_{i=1}^{n} p_i = 1\).
  - Range: \(p = 0, 1, \ldots\)
\( m \)
is a list of numeric constants, variables, or expressions that specifies the \( n \) means \( m_1, m_2, \ldots, m_n \).

\( s \)
is a list of numeric constants, variables, or expressions that specifies the \( n \) standard deviations \( s_1, s_2, \ldots, s_n \).

Range \( s > 0 \)

Details

The PDF function for the Normal Mixture distribution returns the probability that an observation from a mixture of normal distribution is less than or equal to \( x \). The equation follows:

\[
PDF('NORMALMIX', x, n, p, m, s) = \sum_{i=1}^{i=n} p_i \cdot PDF('NORMAL', x, m_i, s_i)
\]

Weights for the Normal Mixture distribution must be nonnegative. If the sum of the weights does not equal 1, then the weights are treated as relative weights and adjusted so that the sum equals 1.

Note: There are no location or scale parameters for the Normal Mixture distribution.

**Pareto Distribution**

PDF ('PARETO', \( x, a < k > \))

Arguments

\( x \)
is a numeric constant, variable, or expression that specifies a numeric random variable.

\( a \)
is a numeric constant, variable, or expression that specifies a shape parameter.

Range \( a > 0 \)

\( k \)
is a numeric constant, variable, or expression that specifies a scale parameter.

Default \( 1 \)

Range \( k > 0 \)

Details

The PDF function for the Pareto distribution returns the probability density function of a Pareto distribution, with the shape parameter \( a \) and the scale parameter \( k \). The PDF function is evaluated at the value \( x \). The equation follows:

\[
PDF('PARETO', x, a, k) = \begin{cases} 
0 & x < k \\
\frac{a}{\frac{k}{x}}^{a+1} & x \geq k 
\end{cases}
\]
**Poisson Distribution**

**PDF (POISSON', n, m)**

**Arguments**

- **n**
  
  is a numeric constant, variable, or expression that specifies an integer random variable.

  Range: \( n = 0, 1, \ldots \)

- **m**

  is a numeric constant, variable, or expression that specifies a mean parameter.

  Range: \( m > 0 \)

**Details**

The PDF function for the Poisson distribution returns the probability density function of a Poisson distribution, with mean \( m \). The PDF function is evaluated at the value \( n \). The equation follows:

\[
PDF(\text{POISSON}', n, m) = \begin{cases} 
0 & n < 0 \\
e^{-m} \frac{n^m}{n!} & n \geq 0
\end{cases}
\]

**Note:** There are no location or scale parameters for the Poisson distribution.

**T Distribution**

**PDF (T', t, df<, nc>)**

**Arguments**

- **t**

  is a numeric constant, variable, or expression that specifies a random variable.

- **df**

  is a numeric constant, variable, or expression that specifies the degrees of freedom.

  Range: \( df > 0 \)

- **nc**

  is a numeric constant, variable, or expression that specifies an optional noncentrality parameter.

**Details**

The PDF function for the T distribution returns the probability density function of a T distribution, with degrees of freedom \( df \) and the noncentrality parameter \( nc \). The PDF function is evaluated at the value \( x \). This PDF function accepts noninteger degrees of freedom. If \( nc \) is omitted or equal to zero, the value returned is from the central T distribution. In the following equation, let \( \nu = df \) and let \( \delta = nc \).

\[
PDF(\text{T}', t, \nu, \delta) = \frac{1}{2^{\frac{\nu-1}{2}} \Gamma\left(\frac{\nu}{2}\right)} \int_0^\infty x^{\nu-1} e^{-\frac{1}{2}x^2} e^{-\frac{1}{2} \left( \frac{t}{\nu} - \delta \right)^2} \frac{x}{\sqrt{\nu}} dx
\]

**Note:** There are no location or scale parameters for the T distribution.
Tweedie Distribution
PDF (‘TWEEDIE’, y, p <, μ, φ>)

Arguments

**y**
is a numeric constant, variable, or expression that specifies a random variable.

**Range**  
\(y \geq 0\)

**Notes**
This argument is required.

When \(y > 1\), \(y\) is numeric. When \(p = 1\), \(y\) is an integer.

**p**
is a numeric constant, variable, or expression that specifies the power parameter.

**Range**  
\(p \geq 1\)

**Note**
This argument is required.

**μ**
is a numeric constant, variable, or expression that specifies the mean parameter.

**Default**  
1

**Range**  
\(μ > 0\)

**φ**
is a numeric constant, variable, or expression that specifies the dispersion parameter.

**Default**  
1

**Range**  
\(φ > 0\)

Details

The PDF function for the Tweedie distribution returns an exponential dispersion model with variance and mean related by the equation \(\text{variance} = φ \cdot μ^p\).

The equation follows:

\[
\frac{1}{\phi} \sum_{j=1}^{\infty} \frac{y^{-ja} (p-1)^ja}{\phi^{j(1-a)}(2-p)^j j! \Gamma(-ja)} \exp \left( \frac{1}{\phi} \left( y \frac{\mu^{1-p} - 1}{1-p} - \frac{\mu^{2-p} - 1}{2-p} \right) \right)
\]

The following relationship applies to the preceding equation:

\[\alpha = \frac{2-p}{1-p}\]

**Note:** The accuracy of computed Tweedie probabilities is highly dependent on the location in parameter space. Ten digits of accuracy are usually available except when \(p\) is near 2 or \(φ\) is near 0. In that case, the accuracy might be as low as six digits.

**Note:** To avoid issues with numerical data, \(μ\) and \(Φ\) cannot be less than the constant SQRTMACEPS.
**Uniform Distribution**

PDF ('UNIFORM', x <, l, r>)

**Arguments**

- **x**
  is a numeric constant, variable, or expression that specifies a random variable.

- **l**
  is a numeric constant, variable, or expression that specifies the left location parameter.
  Default 0

- **r**
  is a numeric constant, variable, or expression that specifies the right location parameter.
  Default 1
  Range r > l

**Details**

The PDF function for the uniform distribution returns the probability density function of a uniform distribution, with the left location parameter l and the right location parameter r. The PDF function is evaluated at the value x. The equation follows:

\[
PDF('UNIFORM', x, l, r) = \begin{cases} 
0 & x < l \\
\frac{1}{r-l} & l \leq x \leq r \\
0 & x > r
\end{cases}
\]

**Wald (Inverse Gaussian) Distribution**

PDF ('WALD', x, λ <, µ >)

PDF ('IGAUSS', x, λ <, µ >)

**Arguments**

- **x**
  is a numeric constant, variable, or expression that specifies a random variable.

- **λ**
  is a numeric constant, variable, or expression that specifies a shape parameter.
  Range λ > 0

- **µ**
  is a numeric constant, variable, or expression that specifies the mean parameter.
  Default 1
  Range µ > 0

**Details**

The PDF function for the Wald distribution returns the probability density function of a Wald distribution, with the shape parameter λ, which is evaluated at the value x. The equation follows:
\[ f(x) = \left( \frac{\lambda}{2\pi x^3} \right)^{1/2} \exp \left( -\frac{\lambda}{2\mu^2} (x - \mu)^2 \right), \quad x > 0 \]

**Weibull Distribution**

PDF(‘WEIBULL’, \(x, a <, \lambda>\))

**Arguments**

- **\(x\)** is a numeric constant, variable, or expression that specifies a random variable.
- **\(a\)** is a numeric constant, variable, or expression that specifies a shape parameter.
  - **Range** \(a > 0\)
- **\(\lambda\)** is a numeric constant, variable, or expression that specifies a scale parameter.
  - **Default** 1
  - **Range** \(\lambda > 0\)

**Details**

The PDF function for the Weibull distribution returns the probability density function of a Weibull distribution, with the shape parameter \(a\) and the scale parameter \(\lambda\). The PDF function is evaluated at the value \(x\). The equation follows:

\[
PDF(‘WEIBULL’, x, a, \lambda) = \begin{cases} 
0 & x < 0 \\
\exp \left( -\left( \frac{x}{\lambda} \right)^a \right) \frac{a}{\lambda} \left( \frac{x}{\lambda} \right)^{a-1} & x \geq 0 
\end{cases}
\]

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>y=pdf(‘BERN’, 0, .25);</td>
<td>0.75</td>
</tr>
<tr>
<td>y=pdf(‘BERN’, 1, .25);</td>
<td>0.25</td>
</tr>
<tr>
<td>y=pdf(‘BETA’, 0.2, 3, 4);</td>
<td>1.2288</td>
</tr>
<tr>
<td>y=pdf(‘BINOM’, 4, .5, 10);</td>
<td>0.20508</td>
</tr>
<tr>
<td>y=pdf(‘CAUCHY’, 2);</td>
<td>0.063662</td>
</tr>
<tr>
<td>y=pdf(‘CHISQ’, 11.264, 11);</td>
<td>0.081686</td>
</tr>
<tr>
<td>y=pdf(‘CONMAXPOI’, .2, 2.3, .4);</td>
<td>0.0097732635</td>
</tr>
<tr>
<td>SAS Statement</td>
<td>Result</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td><code>y=pdf('EXPO', 1);</code></td>
<td>0.36788</td>
</tr>
<tr>
<td><code>y=pdf('F', 3.32, 2, 3);</code></td>
<td>0.054027</td>
</tr>
<tr>
<td><code>y=pdf('GAMMA', 1, 3);</code></td>
<td>0.18394</td>
</tr>
<tr>
<td><code>y=pdf('GENPOISSON', 9, 1, .7);</code></td>
<td>0.0150130915</td>
</tr>
<tr>
<td><code>y=pdf('GEOMETRIC', 5, .3);</code></td>
<td>y=0.050421</td>
</tr>
<tr>
<td><code>y=pdf('HYPER', 2, 200, 50, 10);</code></td>
<td>0.28685</td>
</tr>
<tr>
<td><code>y=pdf('LAPLACE', 1);</code></td>
<td>0.18394</td>
</tr>
<tr>
<td><code>y=pdf('LOGISTIC', 1);</code></td>
<td>0.19661</td>
</tr>
<tr>
<td><code>y=pdf('LOGNORMAL', 1);</code></td>
<td>0.39894</td>
</tr>
<tr>
<td><code>y=pdf('NEGB', 1, .5, 2);</code></td>
<td>0.25</td>
</tr>
<tr>
<td><code>y=pdf('NORMAL', 1.96);</code></td>
<td>0.058441</td>
</tr>
<tr>
<td><code>y=pdf('NORMALMIX', 2.3, 3, .33, .33, .34, .5, 1.5, 2.5, .79, 1.6, 4.3);</code></td>
<td>0.1166</td>
</tr>
<tr>
<td><code>y=pdf('PARETO', 1, 1);</code></td>
<td>1</td>
</tr>
<tr>
<td><code>y=pdf('POISSON', 2, 1);</code></td>
<td>0.18394</td>
</tr>
<tr>
<td><code>y=pdf('T', .9, 5);</code></td>
<td>0.24194</td>
</tr>
<tr>
<td><code>y=pdf('TWEEDIE', .8, 5);</code></td>
<td>0.7422908236</td>
</tr>
<tr>
<td><code>y=pdf('UNIFORM', 0.25);</code></td>
<td>1</td>
</tr>
<tr>
<td><code>y=pdf('WALD', 1, 2);</code></td>
<td>0.56419</td>
</tr>
<tr>
<td><code>y=pdf('WEIBULL', 1, 2);</code></td>
<td>0.73576</td>
</tr>
</tbody>
</table>

See Also

Functions:

- “CDF Function” on page 153
- “LOGCDF Function” on page 416
- “LOGPDF Function” on page 418
- “LOGSDF Function” on page 420
PERM Function

Computes the number of permutations of \( n \) items that are taken \( r \) at a time.

**Category:** Combinatorial

**Syntax**

\[
\text{PERM}(n, r)
\]

**Required Argument**

\( n \)

is an integer that represents the total number of elements from which the sample is chosen.

**Optional Argument**

\( r \)

is an integer value that represents the number of chosen elements. If \( r \) is omitted, the function returns the factorial of \( n \).

**Restriction**

\( r \leq n \)

**Details**

The mathematical representation of the PERM function is given by the following equation:

\[
\text{PERM}(n, r) = \frac{n!}{(n-r)!}
\]

with \( n \geq 0, r \geq 0, \) and \( n \geq r \).

If the expression cannot be computed, a missing value is returned. For moderately large values, it is sometimes not possible to compute the PERM function.

References

Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=perm(5, 1);</td>
<td>5</td>
</tr>
<tr>
<td>x=perm(5);</td>
<td>120</td>
</tr>
<tr>
<td>x=perm(5, 2)</td>
<td>20</td>
</tr>
</tbody>
</table>

See Also

Functions:
- “COMB Function” on page 183
- “FACT Function” on page 242
- “LPERM Function” on page 423

PMT Function

Returns the periodic payment for a constant payment loan or the periodic savings for a future balance.

Category: Financial

Syntax

PMT(rate, number-of-periods, principal-amount, <future-amount>, <type>)

Required Arguments

rate
specifies the interest rate per payment period.

number-of-periods
specifies the number of payment periods.

Requirement  Number-of-periods must be a positive integer value.

principal-amount
specifies the principal amount of the loan. Zero is assumed if a missing value is specified.

Optional Arguments

future-amount
specifies the future amount. Future-amount can be the outstanding balance of a loan after the specified number of payment periods, or the future balance of periodic savings. Zero is assumed if future-amount is omitted or if a missing value is specified.
**type**

specifies whether the payments occur at the beginning or end of a period. 0 represents the end-of-period payments, and 1 represents the beginning-of-period payments. 0 is assumed if type is omitted or if a missing value is specified.

**Example**

- The monthly payment for a $10,000 loan with a nominal annual interest rate of 8% and 10 end-of-month payments can be computed in the following ways:
  
  \[
  \text{Payment1} = \text{PMT}(0.08/12, 10, 10000, 0, 0); \\
  \text{Payment1} = \text{PMT}(0.08/12, 10, 10000); \\
  \]

  These computations return a value of 1037.0320894.

- If the same loan has beginning-of-period payments, then payment can be computed as follows:

  \[
  \text{Payment2} = \text{PMT}(0.08/12, 10, 10000, 0, 1); \\
  \]

  This computation returns a value of 1030.1643272.

- The payment for a $5,000 loan earning a 12% nominal annual interest rate, that is to be paid back in five monthly payments, is computed as follows:

  \[
  \text{Payment3} = \text{PMT}(0.01/12, 5, 5000); \\
  \]

  This computation returns a value of 1002.5013883.

- The payment for monthly periodic savings that accrue over 18 years at a 6% nominal annual interest rate, and which accumulates $50,000 at the end of the 18 years, is computed as follows:

  \[
  \text{Payment4} = \text{PMT}(0.06/12, 216, 0, 50000, 0); \\
  \]

  This computation returns a value of −129.0811609.

---

**POISSON Function**

Returns the probability from a Poisson distribution.

**Category:** Probability

**See:** “CDF Function” on page 153 , “PDF Function” on page 477

**Syntax**

\[
\text{POISSON}(m, n) \\
\]

**Required Arguments**

\[
m \\

\text{is a numeric mean parameter.} \\
\text{Range} \quad m \geq 0 \\
\]

\[
n \\

\text{is an integer random variable.} \\
\]
Details
The POISSON function returns the probability that an observation from a Poisson distribution, with mean $m$, is less than or equal to $n$. To compute the probability that an observation is equal to a given value, $n$, compute the difference of two probabilities from the Poisson distribution for $n$ and $n-1$.

Example
The following SAS statement produces this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=poisson(1, 2);</td>
<td>0.9196986029</td>
</tr>
</tbody>
</table>

See Also
Functions:
- “CDF Function” on page 153
- “LOGCDF Function” on page 416
- “LOGPDF Function” on page 418
- “LOGSDF Function” on page 420
- “PDF Function” on page 477
- “SDF Function” on page 596

PPMT Function
Returns the principal payment for a given period for a constant payment loan or the periodic savings for a future balance.

Category: Financial

Syntax
\[
\text{PPMT}(\text{rate}, \text{period}, \text{number-of-periods}, \text{principal-amount}, <\text{future-amount}>, <\text{type}>)
\]

Required Arguments
- $\text{rate}$ specifies the interest rate per payment period.
- $\text{period}$ specifies the payment period for which the principal payment is computed. Period must be a positive integer value that is less than or equal to the value of $\text{number-of-periods}$. 
**number-of-periods**

specifies the number of payment periods. *Number-of-periods* must be a positive integer value.

**principal-amount**

specifies the principal amount of the loan. Zero is assumed if a missing value is specified.

**Optional Arguments**

**future-amount**

specifies the future amount. *Future-amount* can be the outstanding balance of a loan after the specified number of payment periods, or the future balance of periodic savings. Zero is assumed if *future-amount* is omitted or if a missing value is specified.

**type**

specifies whether the payments occur at the beginning or end of a period. 0 represents the end-of-period payments, and 1 represents the beginning-of-period payments. 0 is assumed if *type* is omitted or if a missing value is specified.

**Example**

- The principal payment amount of the first monthly periodic payment for a 2-year, $2,000 loan with a nominal annual interest rate of 10%, is computed as follows:

  \[ \text{PrincipalPayment} = \text{PPMT} \left( \frac{0.1}{12}, 1, 24, 2010 \right) \]

  This computation returns a value of 76.001301938.

- The principal payment for a 3-year, $20,000 loan with beginning-of-month payments is computed as follows:

  \[ \text{PrincipalPayment}_2 = \text{PPMT} \left( \frac{0.1}{12}, 1, 36, 2010, 0, 1 \right) \]

  This computation returns a value of 64.321037613 as the principal that was paid with the first payment.

- The principal payment of an end-of-month payment loan with an outstanding balance of $5,000 at the end of three years, is computed as follows:

  \[ \text{PrincipalPayment}_3 = \text{PPMT} \left( \frac{0.1}{12}, 1, 36, 2010, 5000, 0 \right) \]

  This computation returns a value of -71.56222304 as the principal that was paid with the first payment.

---

**PROBBETA Function**

Returns the probability from a beta distribution.

**Category:** Probability

**See:** “CDF Function” on page 153, “PDF Function” on page 477

**Syntax**

\[ \text{PROBBETA}(x, a, b) \]
Required Arguments

\( x \)

is a numeric constant, variable, or expression that specifies the value of a random variable.

Range \( 0 \leq x \leq 1 \)

\( a \)

is a numeric constant, variable, or expression that specifies the value of this shape parameter.

Range \( a > 0 \)

\( b \)

is a numeric constant, variable, or expression that specifies the value of this shape parameter.

Range \( b > 0 \)

Details

The PROBBETA function returns the probability that an observation from a beta distribution, with shape parameters \( a \) and \( b \), is less than or equal to \( x \).

Example

The following SAS statement produces this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x=probbeta(.2, 3, 4);</code></td>
<td>0.09888</td>
</tr>
</tbody>
</table>

See Also

Functions:

- “CDF Function” on page 153
- “LOGCDF Function” on page 416
- “LOGPDF Function” on page 418
- “LOGSDF Function” on page 420
- “PDF Function” on page 477
- “SDF Function” on page 596

PROBBNML Function

Returns the probability from a binomial distribution.

Category: Probability

See: “CDF Function” on page 153, “PDF Function” on page 477
Syntax

PROBBNML\((p, n, m)\)

**Required Arguments**

\(p\)

is a numeric constant, variable, or expression that specifies a probability of success.

Range \(0 \leq p \leq 1\)

\(n\)

is a numeric constant, variable, or expression that specifies an integer number of independent Bernoulli trials.

Range \(n > 0\)

\(m\)

is a numeric constant, variable, or expression that specifies an integer number of successes.

Range \(0 \leq m \leq n\)

**Details**

The PROBBNML function returns the probability that an observation from a binomial distribution, with probability of success \(p\) and number of trials \(n\), produces \(m\) or fewer successes. To compute the probability of exactly \(m\) successes, compute the difference of two probabilities from the binomial distribution for \(m\) and \(m-1\) successes.

**Example**

The following SAS statement produces this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=probbnml(0.5, 10, 4);</td>
<td>0.376953125</td>
</tr>
</tbody>
</table>

**See Also**

**Functions:**

- “CDF Function” on page 153
- “LOGCDF Function” on page 416
- “LOGPDF Function” on page 418
- “LOGSDF Function” on page 420
- “PDF Function” on page 477
- “SDF Function” on page 596
PROBBNRM Function

Returns a probability from a bivariate normal distribution.

**Category:** Probability

**Syntax**

\[
\text{PROBBNRM}(x, y, r)
\]

**Required Arguments**

- \(x\) is a numeric constant, variable, or expression that specifies the value of random variable \(x\).
- \(y\) is a numeric constant, variable, or expression that specifies the value of random variable \(y\).
- \(r\) is a numeric constant, variable, or expression that specifies the value of the correlation coefficient.

**Range** \(-1 \leq r \leq 1\)

**Details**

The PROBBNRM function returns the probability that an observation from a standardized bivariate normal distribution with mean 0, variance 1, and correlation coefficient \(r\), is less than or equal to \((x, y)\). That is, it returns the probability that \(X \leq x\) and \(Y \leq y\). The following equation describes the PROBBNRM function, where \(u\) and \(v\) represent the random variables \(x\) and \(y\), respectively:

\[
\text{PROBBNRM}(x, y, r) = \frac{1}{2\pi \sqrt{1-r^2}} \int_{-\infty}^{x} \int_{-\infty}^{y} \exp\left[-\frac{u^2 - 2ruv + v^2}{2(1-r^2)}\right] \, dv \, du
\]

**Example**

The following SAS statement produces this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>p=probbnrm(.4, -.3, .2);</td>
<td>0.2783183345</td>
</tr>
<tr>
<td>put p;</td>
<td></td>
</tr>
</tbody>
</table>

**See Also**

**Functions:**
- “CDF Function” on page 153
PROBCHI Function

Returns the probability from a chi-square distribution.

**Category:** Probability

**See:** “CDF Function” on page 153, “PDF Function” on page 477

---

**Syntax**

\[
\text{PROBCHI}(x, \, df, \, nc)
\]

**Required Arguments**

\( x \)

- is a numeric constant, variable, or expression that specifies the value of a random variable.
- Range: \( x \geq 0 \)

\( df \)

- is a numeric constant, variable, or expression that specifies the degrees of freedom parameter.
- Range: \( df > 0 \)

**Optional Argument**

\( nc \)

- is a numeric constant, variable, or expression that specifies an optional noncentrality parameter.
- Range: \( nc \geq 0 \)

**Details**

The PROBCHI function returns the probability that an observation from a chi-square distribution, with degrees of freedom \( df \) and noncentrality parameter \( nc \), is less than or equal to \( x \). This function accepts a noninteger degrees of freedom parameter \( df \). If the optional parameter \( nc \) is not specified or has the value 0, the value returned is from the central chi-square distribution.
Example

The following SAS statement produces this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=probchi(11.264, 11);</td>
<td>0.5785813293</td>
</tr>
</tbody>
</table>

See Also

Functions:
- “CDF Function” on page 153
- “LOGCDF Function” on page 416
- “LOGPDF Function” on page 418
- “LOGSDF Function” on page 420
- “PDF Function” on page 477
- “SDF Function” on page 596

PROBF Function

Returns the probability from an $F$ distribution.

**Category:** Probability

**See:** “CDF Function” on page 153, “PDF Function” on page 477

**Syntax**

PROBF($x, ndf, ddf <, nc>)$

**Required Arguments**

$x$

is a numeric constant, variable, or expression that specifies the value of a random variable.

Range $x \geq 0$

$ndf$

is a numeric numerator that specifies the numerator degrees of freedom parameter.

Range $ndf > 0$

$ddf$

is a numeric denominator that specifies the denominator degrees of freedom parameter.

Range $ddf > 0$
**Optional Argument**

\( nc \)

is a numeric constant, variable, or expression that specifies an optional noncentrality parameter.

Range \( nc \geq 0 \)

**Details**

The PROBF function returns the probability that an observation from an \( F \) distribution, with numerator degrees of freedom \( ndf \), denominator degrees of freedom \( ddf \), and noncentrality parameter \( nc \), is less than or equal to \( x \). The PROBF function accepts noninteger degrees of freedom parameters \( ndf \) and \( ddf \). If the optional parameter \( nc \) is not specified or has the value 0, the value returned is from the central \( F \) distribution.

The significance level for an \( F \) test statistic is given by

\[
p = 1 - \text{probf}(x, ndf, ddf);
\]

**Example**

The following SAS statement produces this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x = \text{probf}(3.32, 2, 3) );</td>
<td>0.8263933602</td>
</tr>
</tbody>
</table>

**See Also**

**Functions:**
- “CDF Function” on page 153
- “LOGCDF Function” on page 416
- “LOGPDF Function” on page 418
- “LOGSDF Function” on page 420
- “PDF Function” on page 477
- “SDF Function” on page 596

**PROBGAM Function**

Returns the probability from a gamma distribution.

**Category:** Probability

**See:** “CDF Function” on page 153, “PDF Function” on page 477

**Syntax**

\[
\text{PROBGAM}(x, a)
\]
**Required Arguments**

\( x \)

- is a numeric constant, variable, or expression that specifies the value of a random variable.
- Range \( x \geq 0 \)

\( a \)

- is a numeric constant, variable, or expression that specifies the shape parameter.
- Range \( a > 0 \)

**Details**

The PROBGAM function returns the probability that an observation from a gamma distribution, with shape parameter \( a \), is less than or equal to \( x \).

**Example**

The following SAS statement produces this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x = \text{probgam}(1, 3) );</td>
<td>0.0803013971</td>
</tr>
</tbody>
</table>

**See Also**

Functions:
- “CDF Function” on page 153
- “LOGCDF Function” on page 416
- “LOGPDF Function” on page 418
- “LOGSDF Function” on page 420
- “PDF Function” on page 477
- “SDF Function” on page 596

---

**PROBHYP Function**

Returns the probability from a hypergeometric distribution.

**Category:** Probability

**See:** “CDF Function” on page 153, “PDF Function” on page 477

**Syntax**

\[
\text{PROBHYP}(N, K, n, x <, r>)
\]
Required Arguments

$N$

is a numeric constant, variable, or expression that specifies an integer population size parameter.

Range  \( N \geq 1 \)

$K$

is a numeric constant, variable, or expression that specifies an integer number of items in the category of interest.

Range  \( 0 \leq K \leq N \)

$n$

is a numeric constant, variable, or expression that specifies an integer sample size parameter.

Range  \( 0 \leq n \leq N \)

$x$

is a numeric constant, variable, or expression that specifies an integer random variable.

Range  \( \max(0, K+n-N) \leq x \leq \min(K,n) \)

Optional Argument

$r$

is a numeric constant, variable, or expression that specifies an optional numeric odds ratio parameter.

Range  \( r \geq 0 \)

Details

The PROBHYPR function returns the probability that an observation from an extended or noncentral hypergeometric distribution, with population size \( N \), number of items \( K \), sample size \( n \), and odds ratio \( r \), is less than or equal to \( x \). If the optional parameter \( r \) is not specified or is set to 1, the value returned is from the central hypergeometric distribution.

Example

The following SAS statement produces this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=probhypr(200, 50, 10, 2);</td>
<td>0.5236734081</td>
</tr>
</tbody>
</table>

See Also

Functions:

- “CDF Function” on page 153
PROBIT Function

Returns a quantile from the standard normal distribution.

**Category:** Quantile

**Syntax**

PROBIT(\(p\))

**Required Argument**

\(p\)

is a numeric probability.

**Range** \(0 < p < 1\)

**Details**

The PROBIT function returns the \(p^{th}\) quantile from the standard normal distribution. The probability that an observation from the standard normal distribution is less than or equal to the returned quantile is \(p\).

**CAUTION:**

The result could be truncated to lie between -8.222 and 7.941.

**Note:** PROBIT is the inverse of the PROBNORM function.

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=probit(.025);</td>
<td>-1.959963985</td>
</tr>
<tr>
<td>x=probit(1.e-7);</td>
<td>-5.199337582</td>
</tr>
</tbody>
</table>

**See Also**

**Functions:**

- “CDF Function” on page 153
- “LOGCDF Function” on page 416
PROBMC Function

Returns a probability or a quantile from various distributions for multiple comparisons of means.

**Category:** Probability

**Syntax**

```
PROBMC(distribution, q, prob, df, nparms <, parameters>)
```

**Required Arguments**

**distribution**

is a character constant, variable, or expression that identifies the distribution. The following distributions are valid:

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysis of Means</td>
<td>ANOM</td>
</tr>
<tr>
<td>One-sided Dunnett</td>
<td>DUNNETT1</td>
</tr>
<tr>
<td>Two-sided Dunnett</td>
<td>DUNNETT2</td>
</tr>
<tr>
<td>Maximum Modulus</td>
<td>MAXMOD</td>
</tr>
<tr>
<td>Partitioned Range</td>
<td>PARTRANGE</td>
</tr>
<tr>
<td>Studentized Range</td>
<td>RANGE</td>
</tr>
<tr>
<td>Williams</td>
<td>WILLIAMS</td>
</tr>
</tbody>
</table>

**q**

is a numeric constant, variable, or expression that specifies the quantile from the distribution.

**Restriction**

Either `q` or `prob` can be specified, but not both.

**prob**

is a numeric constant, variable, or expression that specifies the left probability from the distribution.

**Restriction**

Either `prob` or `q` can be specified, but not both.
df
is a numeric constant, variable, or expression that specifies the degrees of freedom.

Note: A missing value is interpreted as an infinite value.

nparms
is a numeric constant, variable, or expression that specifies the number of treatments.

Note: For DUNNETT1 and DUNNETT2, the control group is not counted.

Optional Argument
parameters
is an optional set of nparms parameters that must be specified to handle unequal sample sizes. The meaning of parameters depends on the value of distribution. If parameters is not specified, equal sample sizes are assumed, which is usually the case for a null hypothesis.

Details

Overview
The PROBMC function returns the probability or the quantile from various distributions with finite and infinite degrees of freedom for the variance estimate.

The prob argument is the probability that the random variable is less than q. Therefore, p-values can be computed as 1 – prob. For example, to compute the critical value for a 5% significance level, set prob = 0.95.

The precision of the computed probability is $O(10^{-8})$ (absolute error); the precision of the computed quantile is $O(10^{-5})$.

Note: The studentized range is not computed for finite degrees of freedom and unequal sample sizes.

Note: Williams’ test is computed only for equal sample sizes.

Some language elements are supported in SAS but not in CAS. An example is the DATALINES statement. Some examples in this document use DATALINES in SAS to create a data set using the CAS engine. For more information, see “Functions in SAS and CAS” on page 54.

Formulas and Parameters
The equations listed here define expressions that are used in equations that relate the probability, prob, and the quantile, q, for different distributions and different situations within each distribution. For these equations, let $\nu$ be the degrees of freedom, df.

\[
d\mu_{\nu}(x) = \frac{\sqrt{2}^{\nu} \Gamma(\frac{\nu}{2})}{\Gamma(\frac{\nu}{2} - \frac{1}{2})} x^{\nu - 1} e^{-\frac{x^2}{2}} dx
\]

\[
\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}
\]

\[
\Phi(x) = \int_{-\infty}^{x} \phi(u) du
\]
Computing the Analysis of Means

Analysis of Means (ANOM) applies to data that is organized as $k$ (Gaussian) samples; the $i$th sample is size $n_i$. Let $I_1 = \cdots = I_k$. The distribution function $[1, 2, 3, 4, 5]$ is the CDF for the maximum absolute of a $k$-dimensional multivariate $T$ vector, with $\nu$ degrees of freedom, and an associated correlation matrix $\rho_{ij} = -\rho_i \rho_j$. This equation can be written as

$$
prob = r(t_1 < h, t_2 < h, \ldots, t_k < h) = \int_0^\infty \left[ \int_0^\infty \prod_{j=0}^{k} g(sh, y, \alpha_j) \phi(y) dy \right] d\mu_v(s)
$$

The following relationship applies to the preceding equation:

$$
g(sh, y, \alpha_j) = \Phi\left( \frac{sh - ya_j}{\sqrt{1 + a_j^2}} \right) - \Phi\left( \frac{-sh - ya_j}{\sqrt{1 + a_j^2}} \right)
$$

$\Gamma(\cdot)$, $\phi(\cdot)$, and $\Phi(\cdot)$, are the gamma function, the density, and the CDF from the standard normal distribution, respectively.

For $\nu = \infty$, the distribution reduces to:

$$
r(t_1 < h, t_2 < h, \ldots, t_k < h) = \int_0^\infty \prod_{j=0}^{k} g(h, y, \alpha_j) \phi(y) dy
$$

The following relationship applies to the preceding equation:

$$
g(h, y, \alpha_j) = \Phi\left( \frac{h - ya_j}{\sqrt{1 + a_j^2}} \right) - \Phi\left( \frac{-h - ya_j}{\sqrt{1 + a_j^2}} \right)
$$

For the balanced case, the distribution reduces to the following:

$$
r(t_1 < h, t_2 < h, \ldots, t_n < h) = \int_0^\infty f(h, y, \rho)^n \phi(y) dy
$$

The following relationship applies to the preceding equation:

$$
f(h, y, \rho) = \Phi\left( \frac{h - y\sqrt{\rho}}{\sqrt{1 + \rho}} \right) - \Phi\left( \frac{-h - y\sqrt{\rho}}{\sqrt{1 + \rho}} \right)
$$

and $\rho = \frac{1}{n - 1}$

Here is the syntax for this distribution:

$s = \text{probmc('anom', q, p, nu, n, <alpha_1, ..., alpha_n>)}$;

Arguments
is a numeric value with the returned result.

$q$

is a numeric value that denotes the quantile.

$p$

is a numeric value that denotes the probability. One of $p$ and $q$ must be missing.

$nu$

is a numeric value that denotes the degrees of freedom.

$n$

is a numeric value that denotes the number of samples.

\(\alpha_i, i=1, ..., k\)

are optional numeric values that denote the alpha values from the first equation of this distribution. See “Computing the Analysis of Means” on page 513.

**Many-One t-Statistics: Dunnett’s One-Sided Test**

- This case relates the probability, $prob$, and the quantile, $q$, for the unequal case with finite degrees of freedom. The parameters are $\lambda_1, ..., \lambda_k$, the value of $n parms$ is set to $k$, and the value of $df$ is set to $\nu$. The equation follows:

$$prob = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \phi(y) \prod_{i=1}^{k} \Phi \left( \frac{\lambda_i y + qx}{\sqrt{1 - \lambda_i^2}} \right) dy \, du(x)$$

- This case relates the probability, $prob$, and the quantile, $q$, for the equal case with finite degrees of freedom. No parameters are passed ($\lambda = \frac{1}{\sqrt{2}}$), the value of $n parms$ is set to $k$, and the value of $df$ is set to $\nu$. The equation follows:

$$prob = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \phi(y) \left[ \Phi \left( y + \sqrt{2q} \right) \right]^k dy \, du(x)$$

- This case relates the probability, $prob$, and the quantile, $q$, for the unequal case with infinite degrees of freedom. The parameters are $\lambda_1, ..., \lambda_k$, the value of $n parms$ is set to $k$, and the value of $df$ is set to missing. The equation follows:

$$prob = \int_{-\infty}^{\infty} \phi(y) \prod_{i=1}^{k} \Phi \left( \frac{\lambda_i y + q}{\sqrt{1 - \lambda_i^2}} \right) dy$$

- This case relates the probability, $prob$, and the quantile, $q$, for the equal case with infinite degrees of freedom. No parameters are passed ($\lambda = \frac{1}{\sqrt{2}}$), the value of $n parms$ is set to $k$, and the value of $df$ is set to missing. The equation follows:

$$prob = \int_{-\infty}^{\infty} \phi(y) \left[ \Phi \left( y + \sqrt{2q} \right) \right]^k dy$$
Many-One t-Statistics: Dunnett’s Two-Sided Test

- This case relates the probability, \( \text{prob} \), and the quantile, \( q \), for the unequal case with finite degrees of freedom. The parameters are \( \lambda_1, \ldots, \lambda_k \), the value of \( \text{nparms} \) is set to \( k \), and the value of \( df \) is set to \( \nu \). The equation follows:

\[
\text{prob} = \int_0^\infty \int_{-\infty}^\infty \phi(y) \prod_{i=1}^k \left[ \frac{\lambda_i y + qx}{\sqrt{1 - \lambda_i^2}} - \frac{\lambda_i y - qx}{\sqrt{1 - \lambda_i^2}} \right] dy d\mu(x)
\]

- This case relates the probability, \( \text{prob} \), and the quantile, \( q \), for the equal case with finite degrees of freedom. No parameters are passed, the value of \( \text{nparms} \) is set to \( k \), and the value of \( df \) is set to \( \nu \). The equation follows:

\[
\text{prob} = \int_0^\infty \int_{-\infty}^\infty \phi(y) \left[ \Phi\left(y + \sqrt{2qx}\right) - \Phi\left(y - \sqrt{2qx}\right) \right]^k dy d\mu(x)
\]

- This case relates the probability, \( \text{prob} \), and the quantile, \( q \), for the unequal case with infinite degrees of freedom. The parameters are \( \lambda_1, \ldots, \lambda_k \), the value of \( \text{nparms} \) is set to \( k \), and the value of \( df \) is set to missing. The equation follows:

\[
\text{prob} = \int_{-\infty}^\infty \phi(y) \prod_{i=1}^k \left[ \frac{\lambda_i y + q}{\sqrt{1 - \lambda_i^2}} - \frac{\lambda_i y - q}{\sqrt{1 - \lambda_i^2}} \right] dy
\]

- This case relates the probability, \( \text{prob} \), and the quantile, \( q \), for the equal case with infinite degrees of freedom. No parameters are passed, the value of \( \text{nparms} \) is set to \( k \), and the value of \( df \) is set to missing. The equation follows:

\[
\text{prob} = \int_{-\infty}^\infty \phi(y) \left[ \Phi\left(y + \sqrt{2q}\right) - \Phi\left(y - \sqrt{2q}\right) \right]^k dy
\]

Computing the Partitioned Range

RANGE applies to the distribution of the studentized range for \( n \) group means. PARTRANGE applies to the distribution of the partitioned studentized range. Let the \( n \) groups be partitioned into \( k \) subsets of size \( n_1 + \ldots + n_k = n \). Then the partitioned range is the maximum of the studentized ranges in the respective subsets. The studentization factor is the same in all cases.

\[
\text{prob} = \int_0^\infty \prod_{i=1}^k \left[ \int_{-\infty}^{\infty} k\phi(y)(\Phi(y) - \Phi(y - qx))^k dy \right]^{n_i} d\mu(x)
\]

Here is the syntax for this distribution:

\[
x = \text{probmc}(\text{partrange}', q, p, \nu, k, n_1, \ldots, n_k);
\]

Arguments

- \( x \) is a numeric value with the returned result (either the probability or the quantile).
- \( q \) is a numeric value that denotes the quantile.
\( p \) is a numeric value that denotes the probability. One of \( p \) and \( q \) must be missing.

\( \nu \) is a numeric value that denotes the degrees of freedom.

\( k \) is a numeric value that denotes the number of groups.

\( n_i, i=1, \ldots, k \) are optional numeric values that denote the \( n \) values from the equation in this distribution. See “Computing the Partitioned Range” on page 515.

The Studentized Range

Note: The studentized range is not computed for finite degrees of freedom and unequal sample sizes.

- This case relates the probability, \( \text{prob} \), and the quantile, \( q \), for the equal case with finite degrees of freedom. No parameters are passed, the value of \( \text{nparms} \) is set to \( k \), and the value of \( \text{df} \) is set to \( \nu \). The equation follows:

\[
\text{prob} = \int_{-\infty}^{\infty} \int_{0}^{\infty} k \phi(y) [\Phi(y) - \Phi(y - q)]^{k-1} dy \, du_{\nu}(x)
\]

- This case relates the probability, \( \text{prob} \), and the quantile, \( q \), for the unequal case with infinite degrees of freedom. The parameters are \( \sigma_1, \ldots, \sigma_k \), the value of \( \text{nparms} \) is set to \( k \), and the value of \( \text{df} \) is set to missing. The equation follows:

\[
\text{prob} = \int_{-\infty}^{\infty} \left( \sum_{j=1}^{k} \left( \prod_{i=1}^{k} \phi \left( \frac{y}{\sigma_i} \right) - \phi \left( \frac{y-q}{\sigma_i} \right) \right)^{1/2} \right) dy
\]

- This case relates the probability, \( \text{prob} \), and the quantile, \( q \), for the equal case with infinite degrees of freedom. No parameters are passed, the value of \( \text{nparms} \) is set to \( k \), and the value of \( \text{df} \) is set to missing. The equation follows:

\[
\text{prob} = \int_{-\infty}^{\infty} k \phi(y) [\Phi(y) - \Phi(y - q)]^{k-1} dy
\]

The Studentized Maximum Modulus

- This case relates the probability, \( \text{prob} \), and the quantile, \( q \), for the unequal case with finite degrees of freedom. The parameters are \( \sigma_1, \ldots, \sigma_k \), the value of \( \text{nparms} \) is set to \( k \), and the value of \( \text{df} \) is set to \( \nu \). The equation follows:

\[
\text{prob} = \int_{0}^{\infty} \prod_{i=1}^{k} \left( \phi \left( \frac{q x}{\sigma_i} \right) - 1 \right) d\mu_{\nu}(x)
\]

- This case relates the probability, \( \text{prob} \), and the quantile, \( q \), for the equal case with finite degrees of freedom. No parameters are passed, the value of \( \text{nparms} \) is set to \( k \), and the value of \( \text{df} \) is set to \( \nu \). The equation follows:

\[
\text{prob} = \int_{0}^{\infty} [2\Phi(q x) - 1]^{k} d\mu_{\nu}(x)
\]
This case relates the probability, \( \text{prob} \), and the quantile, \( q \), for the unequal case with infinite degrees of freedom. The \textit{parameters} are \( \sigma_1, ..., \sigma_k \), the value of \textit{nparms} is set to \( k \), and the value of \( df \) is set to missing. The equation follows:

\[
\text{prob} = \prod_{i=1}^{k} \left( 2 \Phi \left( \frac{q}{\sigma_i} \right) - 1 \right)
\]

This case relates the probability, \( \text{prob} \), and the quantile, \( q \), for the equal case with infinite degrees of freedom. No \textit{parameters} are passed, the value of \textit{nparms} is set to \( k \), and the value of \( df \) is set to missing. The equation follows:

\[
\text{prob} = \left[ 2 \Phi(q) - 1 \right]^k
\]

**Williams' Test**

PROBMC computes the probabilities or quantiles from the distribution defined in Williams (1971, 1972). (See “References” on page 1111.) It arises when you compare the dose treatment means with a control mean to determine the lowest effective dose of treatment.

Note: Williams' Test is computed only for equal sample sizes.

Let \( X_1, X_2, ..., X_k \) be identical independent \( N(0,1) \) random variables. Let \( Y_k \) denote their average given by

\[
Y_k = \frac{X_1 + X_2 + ... + X_k}{k}
\]

It is required to compute the distribution of

\[
(Y_k - Z) / S
\]

**Arguments**

\( Y_k \)

is as defined previously.

\( Z \)

is an \( N(0,1) \) independent random variable.

\( S \)

is such that \( \frac{1}{2} \nu S^2 \) is a \( \chi^2 \) variable with \( \nu \) degrees of freedom.

As described in Williams (1971) (see “References” on page 1111), the full computation is extremely lengthy and is carried out in three stages.

1. Compute the distribution of \( Y_k \). It is the fundamental (expensive) part of this operation and it can be used to find both the density and the probability of \( Y_k \). Let \( U_i \) be defined as

\[
U_i = \frac{X_1 + X_2 + ... + X_i}{i}, \quad i = 1, 2, ..., k
\]

You can write a recursive expression for the probability of \( Y_k > d \), with \( d \) as any real number.
\[ \Pr(Y_k > d) = \Pr(U_1 > d) + \Pr(U_2 > d, U_1 < d) + \Pr(U_3 > d, U_2 < d, U_1 < d) + \ldots + \Pr(U_k > d, U_{k-1} < d, \ldots, U_1 < d) = \Pr(Y_{k-1} > d) + \Pr(X_k + (k-1)U_{k-1} > kd) \]

To compute this probability, start from an \(N(0,1)\) density function

\[ D(U_1 = x) = \phi(x) \]

and recursively compute the convolution

\[ D(U_k = x, U_{k-1} < d, \ldots, U_1 < d) = \]

\[ \int_{-\infty}^{d} D(U_{k-1} = y, U_{k-2} < d, \ldots, U_1 < d)(k-1)\phi(kx - (k-1)y)dy \]

From this sequential convolution, it is possible to compute all the elements of the recursive equation for \(\Pr(Y_k < d)\), shown previously.

2. Compute the distribution of \(Y_k - Z\). This computation involves another convolution to compute the probability

\[ \Pr((Y_k - Z) > d) = \int_{-\infty}^{\infty} \Pr(Y_k > \sqrt{2d} + y)\phi(y)dy \]

3. Compute the distribution of \((Y_k - Z)/S\). This computation involves another convolution to compute the probability

\[ \Pr((Y_k - Z) > ts) = \int_{0}^{\infty} \Pr((Y_k - Z) > ty)d\mu_v(y) \]

The third stage is not needed when \(v = \infty\). Due to the complexity of the operations, this lengthy algorithm is replaced by a much faster one when \(k \leq 15\) for both finite and infinite degrees of freedom \(v\). For \(k \geq 16\), the lengthy computation is carried out. It is extremely expensive and very slow due to the complexity of the algorithm.

**Comparisons**

The MEANS statement in the GLM Procedure of SAS/STAT Software computes the following tests:

- Dunnett's one-sided test
- Dunnett's two-sided test
- Studentized Range
Examples

Example 1: Computing Probabilities By Using PROBMC
This example shows how to compute probabilities.

```
data probs;
  array par{5};
  par{1}=.5;
  par{2}=.51;
  par{3}=.55;
  par{4}=.45;
  par{5}=.2;
  df=40;
  q=1;
  do test="dunnett1","dunnett2", "maxmod";
    prob=probmc(test, q, ., df, 5, of par1–par5);
    put test $10. df q e18.13 prob e18.13;
  end;
run;
```

SAS writes the following results to the log:

```
DUNNETT1  40  1.00000000000E+00 4.82992196083E-01
DUNNETT2  40  1.00000000000E+00 1.64023105316E-01
MAXMOD    40  1.00000000000E+00 8.02784203408E-01
```

Example 2: Computing the Analysis of Means
```
data _null_;                                 put q1=;
  q1=probmc('anom', ., 0.9, ., 20);                  put q2=;
  q2=probmc('anom', ., 0.9, 20, 5, 0.1, 0.1, 0.1);  put q3=;
  q3=probmc('anom', ., 0.9, 20, 5, 0.5, 0.5, 0.5, 0.5);  put q4=;
  q4=probmc('anom', ., 0.9, 20, 5, 0.1, 0.2, 0.3, 0.4, 0.5);  put q4=;
run;
```

SAS writes the following results to the log:

```
q1=2.7895061016
q2=2.4549961967
q3=2.4549961967
q4=2.4532319994
```

Example 3: Comparing Means
This example shows how to compare group means to find where the significant differences lie. The data for this example is taken from a paper by Duncan (1955), and can also be found in Hochberg and Tamhane (1987). (See the References section at the end of this function.)

The following values are the group means:

- 49.6
- 71.2
- 67.6
- 61.5
- 71.3
For this data, the mean square error is \( s^2 = 79.64 \) (\( s = 8.924 \)) with \( v = 30 \).

```
data duncan;
array tr{7}$;
array mu{7};
n=7;
do i=1 to n;
   input tr{i} $1. mu{i};
end;
input df s alpha;
prob=1-alpha;
/* compute the interval */
x=probmc("RANGE", ., prob, df, 7);
w=x * s / sqrt(6);
/* compare the means */
do i=1 to n;
do j=i + 1 to n;
   dmean = abs(mu{i} - mu{j});
   if dmean >= w then do;
      put tr{i} tr{j} dmean;
   end;
end;
end;
datalines;
A 49.6
B 71.2
C 67.6
D 61.5
E 71.3
F 58.1
G 61.0
30 8.924 .05
;
run;
```

SAS writes the following results to the log:

```
A B 21.6
A C 18
A E 21.7
```

**Example 4**

```
data _null_;  
q1=probmc('partrange', ., 0.9, ., 4, 3, 4, 5, 6); put q1=;
q2=probmc('partrange', ., 0.9, 12, 4, 3, 4, 5, 6); put q2=;
run;
```

SAS writes the following results to the log:

```
q1=4.1022397989
q2=4.7888626338
```
Example 5: Computing Confidence Intervals

This example shows how to compute 95% one-sided and two-sided confidence intervals of Dunnett’s test. This example and the data come from Dunnett (1955), and can also be found in Hochberg and Tamhane (1987). (See the References section at the end of this function.) The data are blood count measurements on three groups of animals. As shown in the following table, the third group serves as the control, and the first two groups were treated with different drugs. The numbers of animals in these three groups are unequal.

<table>
<thead>
<tr>
<th>Treatment Group:</th>
<th>Drug A</th>
<th>Drug B</th>
<th>Control</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>9.76</td>
<td>12.80</td>
<td>7.40</td>
</tr>
<tr>
<td></td>
<td>8.80</td>
<td>9.68</td>
<td>8.50</td>
</tr>
<tr>
<td></td>
<td>7.68</td>
<td>12.16</td>
<td>7.20</td>
</tr>
<tr>
<td></td>
<td>9.36</td>
<td>9.20</td>
<td>8.24</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Group Mean</td>
<td>8.90</td>
<td>10.88</td>
<td>8.25</td>
</tr>
<tr>
<td>n</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

The mean square error $s^2=1.3805$ ($s=1.175$) with $v = 12$.

```plaintext
data a;
array drug{3}$;
array count{3};
array mu{3};
array lambda{2};
array delta{2};
array left{2};
array right{2};
/* input the table */
do i=1 to 3;
   input drug{i} count{i} mu{i};
end;
/* input the alpha level, */
/* the degrees of freedom, */
/* and the mean square error */
input alpha df s;
/* from the sample size, */
/* compute the lambdas */
do i=1 to 2;
   lambda{i}=sqrt(count{i}/
   (count{i} + count{3}));
end;
/* run the one-sided Dunnett's test */
test="dunnett1";
x=probmc(test, ., 1 - alpha, df,
```

PROBMC Function 521
2, of lambda1-lambda2);

do i=1 to 2;
  delta{i}=x * s * 
    sqrt(1/count{i} + 1/count{3});
  left{i}=mu{i} - mu{3} - delta{i};
end;

put test $10. x left{1} left{2};
/* run the two-sided Dunnett's test */
test="dunnett2";
x=probmc(test, ., 1 - alpha, df,
  2, of lambda1-lambda2);

do i=1 to 2;
  delta{i}=x * s * 
    sqrt(1/count{i} + 1/count{3});
  left{i}=mu{i} - mu{3} - delta{i};
  right{i}=mu{i} - mu{3} + delta{i};
end;

put test $10. left{1} right{1};
put test $10. left{2} right{2};

datalines;
A   4  8.90
B   5 10.88
C   6  8.25
0.05 12 1.175
;
run;

SAS writes the following results to the log:

DUNNETT1  2.1210448226 -0.958726041 1.1208812046
DUNNETT2  -1.256408109 2.5564081095
DUNNETT2  0.8416306717 4.4183693283

Example 6: Computing Williams' Test
In the following example, a substance has been tested at seven levels in a randomized block design of eight blocks. The observed treatment means are as follows:

<table>
<thead>
<tr>
<th>Treatment</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>X₀</td>
<td>10.4</td>
</tr>
<tr>
<td>X₁</td>
<td>9.9</td>
</tr>
<tr>
<td>X₂</td>
<td>10.0</td>
</tr>
<tr>
<td>X₃</td>
<td>10.6</td>
</tr>
<tr>
<td>X₄</td>
<td>11.4</td>
</tr>
<tr>
<td>X₅</td>
<td>11.9</td>
</tr>
<tr>
<td>X₆</td>
<td>11.7</td>
</tr>
</tbody>
</table>
The mean square, with \((7 – 1)(8 – 1) = 42\) degrees of freedom, is \(s^2 = 1.16\).

Determine the maximum likelihood estimates \(M_i\) through the averaging process.

- Because \(X_0 > X_1\), form \(X_{0,1} = (X_0 + X_1) / 2 = 10.15\).
- Because \(X_{0,1} > X_2\), form \(X_{0,1,2} = (X_0 + X_1 + X_2) / 3 = (2X_{0,1} + X_2) / 3 = 10.1\).
- \(X_{0,1,2} < X_3 < X_4\)
- Because \(X_3 > X_5\), form \(X_{5,6} = (X_3 + X_6) / 2 = 11.8\).

Now the order restriction is satisfied.

Here are the maximum likelihood estimates under the alternative hypothesis:

- \(M_0 = M_1 = M_2 = X_{0,1,2} = 10.1\)
- \(M_3 = X_3 = 10.6\)
- \(M_4 = X_4 = 11.4\)
- \(M_5 = M_6 = X_{5,6} = 11.8\)

Now compute \(t = (11.8 - 10.4) / \sqrt{2(1.16)^2 / 8} = 2.60\) and the probability that corresponds to \(k = 6\), \(\nu = 42\), and \(t = 2.60\) is \(0.9924467341\), which shows strong evidence that there is a response to the substance. You can also compute the quantiles for the upper 5% and 1% tails, as shown in the following table.

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>prob=probmc(&quot;williams&quot;, 2.6, ., 42, 6);</td>
<td>0.9924466872</td>
</tr>
<tr>
<td>quant5=probmc(&quot;williams&quot;, ., .95, 42, 6);</td>
<td>1.806562536</td>
</tr>
<tr>
<td>quant1=probmc(&quot;williams&quot;, ., .99, 42, 6);</td>
<td>2.490908273</td>
</tr>
</tbody>
</table>

See Also

Functions:

- “CDF Function” on page 153
- “LOGCDF Function” on page 416
- “LOGPDF Function” on page 418
- “LOGSDF Function” on page 420
- “PDF Function” on page 477
- “SDF Function” on page 596

References

PROBNEGB Function

Returns the probability from a negative binomial distribution.

Category: Probability
See: “CDF Function” on page 153

Syntax

PROBNEGB(p, n, m)

Required Arguments

p

is a numeric constant, variable, or expression that specifies the probability of success.

Range

0 ≤ p ≤ 1

n

is a numeric constant, variable, or expression that specifies an integer number of successes.

Range

n ≥ 1

m

is a numeric constant, variable, or expression that specifies an integer number of failures.

Range

m ≥ 0

Details

The PROBNEGB function returns the probability that an observation from a negative binomial distribution, with probability of success p and number of successes n, is less
than or equal to $m$. This result is the probability of $m$ failures occurring before the $n$th success.

To compute the probability that an observation is equal to a given value $m$, compute the difference of two probabilities from the negative binomial distribution for $m$ and $m-1$.

**Example**

The following SAS statement produces this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x=probnegb(0.5, 2, 1);</code></td>
<td>0.5</td>
</tr>
</tbody>
</table>

**See Also**

Functions:

- “CDF Function” on page 153
- “LOGCDF Function” on page 416
- “LOGPDF Function” on page 418
- “LOGSDF Function” on page 420
- “PDF Function” on page 477
- “SDF Function” on page 596

**PROBNORM Function**

Returns the probability from the standard normal distribution.

**Category:** Probability

**Syntax**

`PROBNORM(x)`

**Required Argument**

$x$

is a numeric constant, variable, or expression that specifies a random variable.

**Details**

The PROBNORM function returns the probability that an observation from the standard normal distribution is less than or equal to $x$.

*Note:* PROBNORM is the inverse of the PROBIT function.
Example
The following SAS statement produces this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=probnorm(1.96);</td>
<td>0.9750021049</td>
</tr>
</tbody>
</table>

See Also

Functions:
- “CDF Function” on page 153
- “LOGCDF Function” on page 416
- “LOGPDF Function” on page 418
- “LOGSDF Function” on page 420
- “PDF Function” on page 477
- “SDF Function” on page 596

PROBT Function
Returns the probability from a $t$ distribution.

**Category:** Probability

**See:** “CDF Function” on page 153, “PDF Function” on page 477

**Syntax**

\[ \text{PROBT}(x, df, nc) \]

**Required Arguments**

$x$

is a numeric constant, variable, or expression that specifies a random variable.

$df$

is a numeric constant, variable, or expression that specifies the degrees of freedom.

Range $df > 0$

**Optional Argument**

$nc$

is a numeric constant, variable, or expression that specifies an optional noncentrality parameter.

**Details**

The PROBT function returns the probability that an observation from a Student's $t$ distribution, with degrees of freedom $df$ and noncentrality parameter $nc$, is less than or
equal to x. This function accepts a noninteger degree of freedom parameter df. If the optional parameter, nc, is not specified or has the value 0, the value that is returned is from the central Student's t distribution.

The significance level of a two-tailed t test is given by

\[ p = (1 - \text{probt}(\text{abs}(x), df)) \times 2; \]

**Example**

The following SAS statement produces this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x = \text{probt}(0.9, 5); )</td>
<td>0.7953143998</td>
</tr>
</tbody>
</table>

**See Also**

**Functions:**
- “CDF Function” on page 153
- “LOGCDF Function” on page 416
- “LOGPDF Function” on page 418
- “LOGSDF Function” on page 420
- “PDF Function” on page 477
- “SDF Function” on page 596

---

**PRXCHANGE Function**

Performs a pattern-matching replacement.

**Category:** Character String Matching

**Restrictions:**
This function is assigned an I18N Level 0 status, and is designed for SBCS data. Do not use this function to process DBCS or MBCS data. For more information, see Internationalization Compatibility.

If you use the *regular-expression-id* argument, you cannot process DBCS and MBCS data because the argument requires the PRXPARSE function, which is not DBCS compatible.

**Syntax**

\[
\text{PRXCHANGE}(\text{perl-regular-expression} | \text{regular-expression-id, times, source})
\]

**Required Arguments**

*perl-regular-expression*

specifies a character constant, variable, or expression with a value that is a Perl regular expression.
regular-expression-id

specifies a numeric variable with a value that is a pattern identifier that is returned from the PRXPARSE function.

Restriction

If you use this argument, you must also use the PRXPARSE function.

times

is a numeric constant, variable, or expression that specifies the number of times to search for a match and replace a matching pattern.

Tip

If the value of times is –1, then matching patterns continue to be replaced until the end of source is reached.

source

specifies a character constant, variable, or expression that you want to search.

Details

The Basics

If you use regular-expression-id, the PRXCHANGE function searches the variable source with the regular-expression-id that is returned by PRXPARSE. It returns the value in source with the changes that were specified by the regular expression. If there is no match, PRXCHANGE returns the unchanged value in source.

If you use perl-regular-expression, PRXCHANGE searches the variable source with the perl-regular-expression, and you do not need to call PRXPARSE. You can use PRXCHANGE with a perl-regular-expression in a WHERE clause and in PROC SQL.

For more information about pattern matching, see “Pattern Matching Using Perl Regular Expressions (PRX)” on page 42.

This example, “Example 2: Changing the Order of First and Last Names By Using the DATA Step in SAS and CAS” on page 530 shows how to run the PRXCHANGE function in CAS.

Some language elements are supported in SAS but not in CAS. An example is the DATALINES statement. Some examples in this document use DATALINES in SAS to create a data set using the CAS engine. For more information, see “Functions in SAS and CAS” on page 54.

Compiling a Perl Regular Expression

If perl-regular-expression is a constant or if it uses the /o option, then the Perl regular expression is compiled once and each use of PRXCHANGE reuses the compiled expression. If perl-regular-expression is not a constant and if it does not use the /o option, then the Perl regular expression is recompiled for each call to PRXCHANGE.

Note: The compile-once behavior occurs when you use PRXCHANGE in a DATA step, in a WHERE clause, or in PROC SQL. For all other uses, the perl-regular-expression is recompiled for each call to PRXCHANGE.

Performing a Match

Perl regular expressions consist of characters and special characters that are called metacharacters. When performing a match, SAS searches a source string for a substring that matches the Perl regular expression that you specify.

To view a short list of Perl regular expression metacharacters that you can use when you build your code, see the table Appendix 1, “Tables of Perl Regular Expression (PRX)
Metacharacters,” on page 1113. You can find a complete list of metacharacters on the Perl website.

Comparisons

The PRXCHANGE function is similar to the CALL PRXCHANGE routine. The difference is that the function returns the value of the pattern-matching replacement as a return argument instead of as one of its parameters.

The Perl regular expression (PRX) functions and CALL routines work together to manipulate strings that match patterns. To see a list and short description of these functions and CALL routines, see the Character String Matching category in “SAS Functions and CALL Routines by Category” on page 701.

Examples

Example 1: Changing the Order of First and Last Names By Using the DATA Step in SAS

The following example uses the DATA step to change the order of first and last names.

```sas
/* Create a data set that contains a list of names. */
data ReversedNames;
  input name & $32.;
datalines;
Jones, Fred
Kavich, Kate
Turley, Ron
Dulix, Yolanda
;
/* Reverse last and first names with a DATA step. */
data names;
  set ReversedNames;
  name=prxchange('s/\w+, \w+/\$2 \$1/', -1, name);
run;
proc print data=names;
run;
```

Output 2.7  Results from the DATA Step

<table>
<thead>
<tr>
<th>Obs</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Fred Jones</td>
</tr>
<tr>
<td>2</td>
<td>Kate Kavich</td>
</tr>
<tr>
<td>3</td>
<td>Ron Turley</td>
</tr>
<tr>
<td>4</td>
<td>Yolanda Dulix</td>
</tr>
</tbody>
</table>
Example 2: Changing the Order of First and Last Names By Using the DATA Step in SAS and CAS

This example uses the DATA step to change the order of first and last names. The first section of code in this example runs in SAS and creates the data set, ReversedNames, with the CAS engine. The second section of code runs in CAS, accesses the data in ReversedNames that is in CAS, and then creates the names data set.

```
data casuser.ReversedNames;
  input name & $32.;
  datalines;
  Jones, Fred
  Kavich, Kate
  Turley, Ron
  Dulix, Yolanda
;
  data casuser.names;
    set casuser.ReversedNames;
    name=prxchange('s/\w+/\w+/$2 $1/', -1, name);
    run;
proc print data=casuser.names;
run;
```

Here is the output from running the code in CAS.

<table>
<thead>
<tr>
<th>Obs</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Fred Jones</td>
</tr>
<tr>
<td>2</td>
<td>Kate Kavich</td>
</tr>
<tr>
<td>3</td>
<td>Ron Turley</td>
</tr>
<tr>
<td>4</td>
<td>Yolanda Dulix</td>
</tr>
</tbody>
</table>

Example 3: Changing the Order of First and Last Names By Using PROC SQL

The following example uses PROC SQL to change the order of first and last names.

```
data ReversedNames;
  input name & $32.;
  datalines;
  Jones, Fred
  Kavich, Kate
  Turley, Ron
  Dulix, Yolanda
;
  proc sql;
    create table names as
      select prxchange('s/(\w+), (\w+)/$2 $1/', -1, name) as name
      from ReversedNames;
    quit;
  proc print data=names;
```

The SAS System
### Example 4: Matching Rows That Have the Same Name

The following example compares the names in two data sets, and writes those names that are common to both data sets.

```sas
data names;
  input name $32.;
datalines;
  Ron Turley
  Judy Donnelly
  Kate Kavich
  Tully Sanchez
;
data ReversedNames;
  input name $32.;
datalines;
  Jones, Fred
  Kavich, Kate
  Turley, Ron
  Dulix, Yolanda
;
proc sql;
  create table NewNames as
  select a.name from names as a, ReversedNames as b
  where a.name=prxchange('s/(\w+), (\w+)/$2 $1/', -1, b.name);
quit;
proc print data=NewNames;
run;
```

#### Output 2.8  Results from PROC SQL

<table>
<thead>
<tr>
<th>Obs</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Fred Jones</td>
</tr>
<tr>
<td>2</td>
<td>Kate Kavich</td>
</tr>
<tr>
<td>3</td>
<td>Ron Turley</td>
</tr>
<tr>
<td>4</td>
<td>Yolanda Dulix</td>
</tr>
</tbody>
</table>
### Example 5: Changing Lowercase Text to Uppercase

The following example uses the \U, \L, and \E metacharacters to change the case of a string of text. Case modifications do not nest. In this example, note that "bear" does not convert to uppercase letters because the \E metacharacter ends all case modifications.

```sas
data _null_;  
  length txt $32;  
  txt=prxchange ('s/(big)(black)(bear)/\U$1\L$2\E$3/', 1, 'bigblackbear');  
  put txt=;  
run;
```

SAS writes the following output to the log:

```
txt=BIGblackbear
```

### Example 6: Changing a Matched Pattern to a Fixed Value

This example locates a pattern in a variable and replaces the variable with a predefined value. The example uses a DATA step to find phone numbers and replace them with an informational message.

```sas
/* Create data set that contains confidential information. */
data a;  
  input text $80.;  
  datalines;  
The phone number for Ed is (801)443-9876 but not until tonight.  
He can be reached at (910)998-8762 tomorrow for testing purposes.  
;  
run;  
/* Locate confidential phone numbers and replace them with message */  
/* indicating that they have been removed. */
data b;  
  set a;  
  text=prxchange('s/\{[2-9]\d\d\} ?\{[2-9]\d\d-\d\d\d\d\}/*PHONE NUMBER REMOVED*/', -1, text);  
  put text=;  
run;  
proc print data=b;  
run;
```
Output 2.10  Results from Changing a Matched Pattern to a Fixed Value

<table>
<thead>
<tr>
<th>Obs</th>
<th>text</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The phone number for Ed is <em>PHONE NUMBER REMOVED</em> but not until tonight.</td>
</tr>
<tr>
<td>2</td>
<td>He can be reached at <em>PHONE NUMBER REMOVED</em> tomorrow for testing purposes.</td>
</tr>
</tbody>
</table>

The SAS System

See Also

Functions:
- “PRXMATCH Function” on page 533
- “PRXPAREN Function” on page 1022
- “PRXPARSE Function” on page 538
- “PRXPOSN Function” on page 541

CALL Routines:
- “CALL PRXCHANGE Routine” on page 782
- “CALL PRXDEBUG Routine” on page 785
- “CALL PRXFREE Routine” on page 127
- “CALL PRXNEXT Routine” on page 787
- “CALL PRXPOSN Routine” on page 789
- “CALL PRXSUBSTR Routine” on page 792

PRXMATCH Function

Searches for a pattern match and returns the position at which the pattern is found.

Category:  Character String Matching

Restrictions:  This function is assigned an I18N Level 0 status, and is designed for SBCS data. Do not use this function to process DBCS or MBCS data. For more information, see Internationalization Compatibility.

If you use the regular-expression-id argument, you cannot process DBCS and MBCS data because the argument requires the PRXPARSE function, which is not DBCS compatible.

Syntax

PRXMATCH(regular-expression-id | perl-regular-expression, source)

Required Arguments

regular-expression-id  specifies a numeric variable with a value that is a pattern identifier that is returned from the PRXPARSE function.
Restriction
If you use this argument, you must also use the PRXPARSE function.

* perl-regular-expression
specifies a character constant, variable, or expression with a value that is a Perl regular expression.

* source
specifies a character constant, variable, or expression that you want to search.

Details

The Basics
If you use regular-expression-id, then the PRXMATCH function searches source with the regular-expression-id that is returned by PRXPARSE, and returns the position at which the string begins. If there is no match, PRXMATCH returns a zero.

If you use perl-regular-expression, PRXMATCH searches source with the perl-regular-expression, and you do not need to call PRXPARSE.

You can use PRXMATCH with a Perl regular expression in a WHERE clause and in PROC SQL. For more information about pattern matching, see “Pattern Matching Using Perl Regular Expressions (PRX)” on page 42.

Some language elements are supported in SAS but not in CAS. An example is the DATALINES statement. Some examples in this document use DATALINES in SAS to create a data set using the CAS engine. For more information, see “Functions in SAS and CAS” on page 54.

Compiling a Perl Regular Expression
If perl-regular-expression is a constant or if it uses the /o option, then the Perl regular expression is compiled once and each use of PRXMATCH reuses the compiled expression. If perl-regular-expression is not a constant and if it does not use the /o option, then the Perl regular expression is recompiled for each call to PRXMATCH.

* Note: The compile-once behavior occurs when you use PRXMATCH in a DATA step, in a WHERE clause, or in PROC SQL. For all other uses, the perl-regular-expression is recompiled for each call to PRXMATCH.

Comparisons
The Perl regular expression (PRX) functions and CALL routines work together to manipulate strings that match patterns. To see a list and short description of these functions and CALL routines, see the Character String Matching category in “SAS Functions and CALL Routines by Category” on page 701.

The example, “Example 5: Extracting a ZIP Code By Using the DATA Step in SAS and CAS” on page 536 shows how to run the PRXMATCH function in CAS.

Examples

**Example 1: Finding the Position of a Substring By Using PRXPARSE**
The following example searches a string for a substring, and returns its position in the string.

/* For release 9.0: the following example makes a call to PRXPARSE. */
/* For release 9.1, no call is required. */

data _null_;  /* Use PRXPARSE to compile the Perl regular expression. */
patternID=prxparse('/world/');  /* Use PRXMATCH to find the position of the pattern match. */
position=prxmatch(patternID, 'Hello world!');
put position=;
run;

SAS writes the following output to the log:

position=7

Example 2: Finding the Position of a Substring By Using a Perl Regular Expression

The following example uses a Perl regular expression to search a string (Hello world) for a substring (world) and to return the position of the substring in the string.

data _null_;  /* Use PRXMATCH to find the position of the pattern match. */
position=prxmatch('/world/', 'Hello world!');
put position=;
run;

SAS writes the following output to the log:

position=7

Example 3: Finding the Position of a Substring in a String: A Complex Example

The following example uses several Perl regular expression functions and a CALL routine to find the position of a substring in a string.

data _null_;  if _N_=1 then do;
retain PerlExpression;
pattern="/\d+:\d+(?:\.\d+)\?/";
PerlExpression=prxparse(pattern);
end;

array match[3] $ 8;
input minsec $80.;
position=prxmatch(PerlExpression, minsec);
if position ^= 0 then do;
do i=1 to prxparen(PerlExpression);
call prxposn(PerlExpression, i, start, length);
if start ^= 0 then
match[i]=substr(minsec, start, length);
end;
if ^missing(match[3]) then
put "", " match[3] "milliseconds";
end;
Example 4: Extracting a ZIP Code By Using the DATA Step
The following example uses a DATA step to search each observation in a data set for a nine-digit ZIP code, and writes those observations to the data set ZipPlus4.

```sas
data ZipCodes;
  input name: $16. zip:$10.;
datalines;
Johnathan 32523-2343
Seth 85030
Kim 39204
Samuel 93849-3843
;
/* Extract ZIP+4 ZIP codes with the DATA step. */
data ZipPlus4;
  set ZipCodes;
  where prxmatch('/\d{5}-\d{4}/', zip);
run;
proc print data=ZipPlus4;
run;
```

**Figure 2.25 ZIP Code Output from the DATA Step**

<table>
<thead>
<tr>
<th>Obs</th>
<th>name</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Johnathan</td>
<td>32523-2343</td>
</tr>
<tr>
<td>2</td>
<td>Samuel</td>
<td>93849-3843</td>
</tr>
</tbody>
</table>

Example 5: Extracting a ZIP Code By Using the DATA Step in SAS and CAS
This example uses a DATA step to search each observation in a data set for a nine-digit ZIP code, and writes those observations to the data set ZipPlus4. The first section of code in this example runs in SAS and creates the data set, ZipCodes, with the CAS engine. The second section of code runs in CAS, accesses the data in ZipCodes that is in CAS, and then creates the ZipPlus4 data set.

```sas
data casuser.ZipCodes;
  input name: $16. zip:$10.;
```
data ZipCodes;
  input name: $16. zip:$10.;
datalines;
Johnathan 32523-2343
Seth 85030
Kim 39204
Samuel 93849-3843
;
data casuser.ZipPlus4;
  set casuser.ZipCodes;
  where prxmatch('/\d{5}-\d{4}/', zip);
run;
proc print data=casuser.ZipPlus4;
run;

Output 2.11  ZIP Code Output from the DATA Step Using CAS

<table>
<thead>
<tr>
<th>Obs</th>
<th>name</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Johnathan</td>
<td>32523-2343</td>
</tr>
<tr>
<td>2</td>
<td>Samuel</td>
<td>93849-3843</td>
</tr>
</tbody>
</table>

Example 6: Extracting a ZIP Code By Using PROC SQL
The following example searches each observation in a data set for a nine-digit ZIP code, and writes those observations to the data set ZipPlus4.

data ZipCodes;
  input name: $16. zip:$10.;
datalines;
Johnathan 32523-2343
Seth 85030
Kim 39204
Samuel 93849-3843
;
/* Extract ZIP+4 ZIP codes with PROC SQL. */
proc sql;
  create table ZipPlus4 as
  select * from ZipCodes
  where prxmatch('/\d{5}-\d{4}/', zip);
run;
proc print data=ZipPlus4;
run;
PRXPARSE Function

Compiles a Perl regular expression (PRX) that can be used for pattern matching of a character value.

Category: Character String Matching

Restrictions: This function is assigned an I18N Level 0 status, and is designed for SBCS data. Do not use this function to process DBCS or MBCS data. For more information, see Internationalization Compatibility.

Use with other Perl regular expressions.

PRXPARSE is not DBCS compatible.

Syntax

\[ \text{regular-expression-id=} \text{PRXPARSE}(\text{perl-regular-expression}) \]
Required Arguments

regular-expression-id

is a numeric pattern identifier that is returned by the PRXPARSE function.

perl-regular-expression

specifies a character value that is a Perl regular expression.

Details

The Basics

The PRXPARSE function returns a pattern identifier number that is used by other Perl functions and CALL routines to match patterns. If an error occurs in parsing the regular expression, SAS returns a missing value.

PRXPARSE uses metacharacters in constructing a Perl regular expression. To view a table of common metacharacters, see Appendix 1, “Tables of Perl Regular Expression (PRX) Metacharacters,” on page 1113.

For more information about pattern matching, see “Pattern Matching Using Perl Regular Expressions (PRX)” on page 42.

The example, “Example 2: PRXPARSE Function Using CAS” on page 540 shows how to run the PRXPARSE function in CAS.

Some language elements are supported in SAS but not in CAS. An example is the DATALINES statement. Some examples in this document use DATALINES in SAS to create a data set using the CAS engine. For more information, see “Functions in SAS and CAS” on page 54.

Compiling a Perl Regular Expression

If perl-regular-expression is a constant or if it uses the /o option, the Perl regular expression is compiled only once. Successive calls to PRXPARSE do not cause a recompile, but returns the regular-expression-id for the regular expression that was already compiled. This behavior simplifies the code because you do not need to use an initialization block (IF _N_ =1) to initialize Perl regular expressions.

Note: If you have a Perl regular expression that is a constant, or if the regular expression uses the /o option, then calling PRXFREE to free the memory allocation results in the need to recompile the regular expression the next time it is called by PRXPARSE. The compile-once behavior occurs when you use PRXPARSE in a DATA step. For all other uses, the perl-regular-expression is recompiled for each call to PRXPARSE.

Comparisons

The Perl regular expression (PRX) functions and CALL routines work together to manipulate strings that match patterns. To see a list and short description of these functions and CALL routines, see the Character String Matching category in “SAS Functions and CALL Routines by Category” on page 701.

Examples

Example 1: PRXPARSE Function Using SAS

The following example uses metacharacters and regular characters to construct a Perl regular expression. This example parses addresses and writes formatted results to the SAS log.
data _null_;  
if _N_=1 then 
do;
    retain patternID;
    /* The i option specifies a case insensitive search. */
    pattern="/ave|avenue|dr|drive|rd|road/i";
    patternID=prxparse(pattern);
end;
input street $80.;
call prxsubstr(patternID, street, position, length);
if position ^= 0 then 
do;
    match=substr(street, position, length);
    put match:$QUOTE. "found in " street:$QUOTE.;
end;
datalines;
153 First Street
6789 64th Ave
4 Moritz Road
7493 Wilkes Place

SAS writes the following output to the log:

"Ave" found in "6789 64th Ave"
"Road" found in "4 Moritz Road"

Example 2: PRXPARSE Function Using CAS
This example parses addresses and writes formatted results to the SAS log using CAS. The first section of code in this example runs in SAS and creates the data set, street1, with the CAS engine. The second section of code runs in CAS, accesses the data in street1 that is in CAS, and then creates the street2 data set.

data casuser.street1;
    input street& $32.;
datalines;
153 First Street
6789 64th Ave
4 Moritz Road
7493 Wilkes Place
;

data casuser.street2;
set casuser.street1;
if _N_=1 then 
do;
    retain patternID;
    /* The i option specifies a case insensitive search. */
    pattern="/ave|avenue|dr|drive|rd|road/i";
    patternID=prxparse(pattern);
end;
call prxsubstr(patternID, street, position, length);
if position ^= 0 then 
do;
    match=substr(street, position, length);
    put match:$QUOTE. "found in " street:$QUOTE.;
end;
run;

SAS writes the following output to the log:

```
"Ave" found in "6789 64th Ave"
"Road" found in "4 Moritz Road"
```

See Also

Functions:
- “PRXCHANGE Function” on page 527
- “PRXMATCH Function” on page 533
- “PRXPAREN Function” on page 1022
- “PRXPOSN Function” on page 541

CALL Routines:
- “CALL PRXCHANGE Routine” on page 782
- “CALL PRXDEBUG Routine” on page 785
- “CALL PRXFREE Routine” on page 127
- “CALL PRXNEXT Routine” on page 787
- “CALL PRXPOSN Routine” on page 789
- “CALL PRXSUBSTR Routine” on page 792

PRXPOSN Function

Returns a character string that contains the value for a capture buffer.

**Category:** Character String Matching

**Restrictions:**
This function is assigned an I18N Level 0 status, and is designed for SBCS data. Do not use this function to process DBCS or MBCS data. For more information, see Internationalization Compatibility.

Use with the PRXPARSE function.

Do not use this function to process DBCS and MBCS data, because this routine requires the PRXPARSE function, which is not DBCS compatible.

**Syntax**

```plaintext
PRXPOSN(regular-expression-id, capture-buffer, source)
```

**Required Arguments**

`regular-expression-id`
specifies a numeric variable with a value that is a pattern identifier that is returned by the PRXPARSE function.
capture-buffer

is a numeric constant, variable, or expression that identifies the capture buffer for
which to retrieve a value:

- If the value of capture-buffer is zero, PRXPOSN returns the entire match.
- If the value of capture-buffer is between 1 and the number of open parentheses in
  the regular expression, then PRXPOSN returns the value for that capture buffer.
- If the value of capture-buffer is greater than the number of open parentheses,
  then PRXPOSN returns a missing value.

source

specifies the text from which to extract capture buffers.

Details

The PRXPOSN function uses the results of PRXMATCH, PRXSUBSTR,
PRXCHANGE, or PRXNEXT to return a capture buffer. A match must be found by one
of these functions for PRXPOSN to return meaningful information.

A capture buffer is part of a match, enclosed in parentheses, that is specified in a regular
expression. This function simplifies using capture buffers by returning the text for the
capture buffer directly, and by not requiring a call to SUBSTR as in the case of CALL
PRXPOSN.

For more information about pattern matching, see “Pattern Matching Using Perl Regular
Expressions (PRX)” on page 42.

This example, “Example 2: PRXPOSN Function Using CAS” on page 543 shows how
to run the PRXPOSN function in CAS.

Some language elements are supported in SAS but not in CAS. An example is the
DATALINES statement. Some examples in this document use DATALINES in SAS to
create a data set using the CAS engine. For more information, see “Functions in SAS
and CAS” on page 54.

Comparisons

The PRXPOSN function is similar to the CALL PRXPOSN routine, except that it
returns the capture buffer itself rather than the position and length of the capture buffer.

The Perl regular expression (PRX) functions and CALL routines work together to
manipulate strings that match patterns. To see a list and short description of these
functions and CALL routines, see the Character String Matching category in “SAS
Functions and CALL Routines by Category” on page 701.

Examples

Example 1: Extracting First and Last Names

The following example uses PRXPOSN to extract first and last names from a data set.

data ReversedNames;
  input name & $32.;
  datalines;
  Jones, Fred
  Kavich, Kate
  Turley, Ron
  Dulix, Yolanda
;
data FirstLastNames;
  length first last $ 16;
  keep first last;
  retain re;
  if _N_=1 then
    re=prxpathre('/\(\w+\), \(\w+\)/');
  set ReversedNames;
  if prxmatch(re, name) then
    do;
      last=prxposn(re, 1, name);
      first=prxposn(re, 2, name);
    end;
  run;
  proc print data=FirstLastNames;
  run;

Figure 2.27  Output from PRXPOSN: First and Last Names

The SAS System

<table>
<thead>
<tr>
<th>Obs</th>
<th>first</th>
<th>last</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Fred</td>
<td>Jones</td>
</tr>
<tr>
<td>2</td>
<td>Kate</td>
<td>Kavich</td>
</tr>
<tr>
<td>3</td>
<td>Ron</td>
<td>Turley</td>
</tr>
<tr>
<td>4</td>
<td>Yolanda</td>
<td>Dulix</td>
</tr>
</tbody>
</table>

Example 2: PRXPOSN Function Using CAS

This example uses PRXPOSN to extract first and last names from a data set. The first section of code in this example runs in SAS and creates the data set, ReversedNames, with the CAS engine. The second section of code runs in CAS, accesses the data in ReversedNames that is in CAS, and then creates the names data set.

data casuser.ReversedNames;
  input name & $32.;
  datalines;
  Jones, Fred
  Kavich, Kate
  Turley, Ron
  Dulix, Yolanda
;

data casuser.FirstLastNames;
  set casuser.ReversedNames;
  length first last $ 16;
  keep first last;
  retain re;
  if _N_=1 then
    re=prxpathre('/\(\w+\), \(\w+\)/');
  if prxmatch(re, name) then
    do;
      last=prxposn(re, 1, name);
    end;
Example 3: Extracting Names When Some Names Are Invalid

The following example creates a data set that contains a list of names. Observations that have only a first name or only a last name are invalid. PRXPOSN extracts the valid names from the data set, and writes the names to the data set NEW.

```sas
data old;
  input name $60.;
datalines;
Judith S Reaveley
Ralph F. Morgan
Jess Ennis
Carol Echols
Kelly Hansen Huff
Judith
Nick
Jones
;
data new;
  length first middle last $ 40;
  keep first middle last;
  re=prxparse('/(\S+\s+\([\^\s]+\)+\s+)?(\S+)/o');
  set old;
  if prxmatch(re, name) then do;
    first=prxposn(re, 1, name);
    middle=prxposn(re, 2, name);
    last=prxposn(re, 3, name);
    output;
  end;
run;
proc print data=new;
run;
```

Output 2.12  Output from PRXPOSN: First and Last Names Using CAS

<table>
<thead>
<tr>
<th>Obs</th>
<th>First</th>
<th>Last</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Fred</td>
<td>Jones</td>
</tr>
<tr>
<td>2</td>
<td>Kate</td>
<td>Kavich</td>
</tr>
<tr>
<td>3</td>
<td>Ron</td>
<td>Turley</td>
</tr>
<tr>
<td>4</td>
<td>Yo'anda</td>
<td>Dulix</td>
</tr>
</tbody>
</table>
PUT Function

Returns a value using a specified format.

**Category:** Special

**Restriction:** This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility.

**Syntax**

\[
\text{PUT}(\text{source, format})
\]
**Required Arguments**

source  
Identifies the constant, variable, or expression whose value you want to reformat. The source argument can be character or numeric.

format.  
Contains the SAS format that you want applied to the value that is specified in the source. This argument must be the name of a format with a period and optional width and decimal specifications, not a character constant, variable, or expression. By default, if the source is numeric, the resulting string is right aligned, and if the source is character, the result is left aligned. To override the default alignment, you can add an alignment specification to a format:

- L left aligns the value.  
- C centers the value.  
- R right aligns the value.

**Restriction**  
The format must be of the same type as the source, either character or numeric. That is, if the source is character, the format name must begin with a dollar sign, but if the source is numeric, the format name must not begin with a dollar sign.

**Details**

If the PUT function returns a value to a variable that has not yet been assigned a length, by default the variable length is determined by the width of the format.

Use the PUT function to convert a numeric value to a character value. The PUT function has no effect on which formats are used in PUT statements or which formats are assigned to variables in data sets. You cannot use the PUT function to directly change the type of variable in a data set from numeric to character. However, you can create a new character variable as the result of the PUT function. Then, if needed, use the DROP statement to drop the original numeric variable, followed by the RENAME statement to rename the new variable back to the original variable name.

**Comparisons**

The PUT statement and the PUT function are similar. The PUT function returns a value using a specified format. You must use an assignment statement to store the value in a variable. The PUT statement writes a value to an external destination (either the SAS log or a destination, that you specify).

**Examples**

**Example 1: Converting Numeric Values to Character Value**

In this example, the first statement converts the values of cc, a numeric variable, into the four-character hexadecimal format, and the second statement writes the same value that the PUT function returns.

```
cchex=put(cc,hex4.);
put cc hex4.;
```

If you need to keep the original variable name of cc, but as a character variable, then use the DROP and RENAME statements following the PUT function.
The new chex variable is created as a character variable from the numeric value of the cc variable. The DROP statement prevents the numeric cc variable from being written to the data set, and the RENAME statement renames the new character chex variable back to the name of cc.

Example 2: Using PUT and INPUT Functions
In this example, the PUT function returns a numeric value as a character string. The value 122591 is assigned to the CHARDATE variable. The INPUT function returns the value of the character string as a SAS date value using a SAS date informat. The value 11681 is stored in the SASDATE variable.

```
numdate=122591;
chardate=put(numdate, z6.);
sasdate=input(chardate, mmddyy6.);
```

See Also

Functions:
- “INPUT Function” on page 334
- “INPUTC Function” on page 337
- “INPUTN Function” on page 339
- “PUTC Function” on page 1025
- “PUTN Function” on page 1028

Statements:
- “PUT Statement” in SAS Viya Statements: Reference

PVP Function
Returns the present value for a periodic cash flow stream (such as a bond), with repayment of principal at maturity.

**Category:** Financial

**Syntax**
PVP\( (A, c, n, K, k_{p, y}) \)

**Required Arguments**
\( A \)
- specifies the par value.
  - Range: \( A > 0 \)
\( c \)
- specifies the nominal per-year coupon rate, expressed as a fraction.
Range: $0 \leq c < 1$

$n$ specifies the number of coupons per year.
Range: $n > 0$ and is an integer

$K$ specifies the number of remaining coupons.
Range: $K > 0$ and is an integer

$k_0$ specifies the time from the present date to the first coupon date, expressed in terms of the number of years.
Range: $0 < k_0 \leq \frac{1}{n}$

$y$ specifies the nominal per-year yield-to-maturity, expressed as a fraction.
Range: $y > 0$

Details
The PVP function is based on the relationship

$$P = \sum_{k=1}^{K} \frac{c(k)}{(1 + \frac{y}{n})^k}$$

The following relationships apply to the preceding equation:

- $t_k = nk_0 + k - 1$
- $c(k) = \frac{c_n A}{n} \text{ for } k = 1, \ldots, K - 1$
- $c(K) = \left(1 + \frac{c_n}{n}\right)A$

Example
```
data _null_;  p=pvp(1000, .01, 4, 14, .33/2, .10);  put p;  run;```
SAS writes the following output to the log:

743.168

QTR Function
Returns the quarter of the year from a SAS date value.

Category: Date and Time
Syntax

QTR(date)

Required Argument

date

specifies a numeric constant, variable, or expression that represents a SAS date value.

Details

The QTR function returns a value of 1, 2, 3, or 4 from a SAS date value to indicate the quarter of the year in which a date value falls.

Example

The following SAS statements produce this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x='20jan94'd;</td>
<td></td>
</tr>
<tr>
<td>y=qtr(x);</td>
<td>y=1</td>
</tr>
<tr>
<td>put y=;</td>
<td></td>
</tr>
</tbody>
</table>

See Also

Functions:

- “YYQ Function” on page 695

---

QUANTILE Function

Returns the quantile from a distribution when you specify the left probability (CDF).

Category: Quantile

See: “CDF Function” on page 153

Syntax

QUANTILE(distribution, probability, parameter-1, ..., parameter-k)

Required Arguments

distribution

is a character constant, variable, or expression that identifies the distribution. Valid distributions are as follows:
<table>
<thead>
<tr>
<th>Distribution</th>
<th>Argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bernoulli</td>
<td>BERNULLI</td>
</tr>
<tr>
<td>Beta</td>
<td>BETA</td>
</tr>
<tr>
<td>Binomial</td>
<td>BINOMIAL</td>
</tr>
<tr>
<td>Cauchy</td>
<td>CAUCHY</td>
</tr>
<tr>
<td>Chi-Square</td>
<td>CHISQUARE</td>
</tr>
<tr>
<td>Conway-Maxwell-Poisson</td>
<td>CONMAXPOI</td>
</tr>
<tr>
<td>Exponential</td>
<td>EXPONENTIAL</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>Gamma</td>
<td>GAMMA</td>
</tr>
<tr>
<td>Generalized Poisson</td>
<td>GENPOISSON</td>
</tr>
<tr>
<td>Geometric</td>
<td>GEOMETRIC</td>
</tr>
<tr>
<td>Hypergeometric</td>
<td>HYPERGEOMETRIC</td>
</tr>
<tr>
<td>Laplace</td>
<td>LAPLACE</td>
</tr>
<tr>
<td>Logistic</td>
<td>LOGISTIC</td>
</tr>
<tr>
<td>Lognormal</td>
<td>LOGNORMAL</td>
</tr>
<tr>
<td>Negative binomial</td>
<td>NEGBINOMIAL</td>
</tr>
<tr>
<td>Normal</td>
<td>NORMAL</td>
</tr>
<tr>
<td>Normal mixture</td>
<td>NORMALMIX</td>
</tr>
<tr>
<td>Pareto</td>
<td>PARETO</td>
</tr>
<tr>
<td>Poisson</td>
<td>POISSON</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>Tweedie</td>
<td>TWEEDIE</td>
</tr>
<tr>
<td>Uniform</td>
<td>UNIFORM</td>
</tr>
<tr>
<td>Wald (inverse Gaussian)</td>
<td>WALD</td>
</tr>
<tr>
<td>Weibull</td>
<td>WEIBULL</td>
</tr>
</tbody>
</table>
Note: Except for T, F, and NORMALMIX, you can minimally identify any distribution by its first four characters.

**probability**
- is a numeric constant, variable, or expression that specifies the value of a random variable.

**parameter-1, ..., parameter-k**
- are optional shape, location, or scale parameters appropriate for the specific distribution.

**Details**

The QUANTILE function computes the quantile from the specified continuous or discrete distribution, based on the probability value that is provided. For more information, see “Details” on page 154 in the CDF function.

The Conway-Maxwell-Poisson distribution for the QUANTILE function returns the counts value \( y \) that is the largest integer whose CDF value is less than or equal to \( p \). The syntax for the Conway-Maxwell-Poisson distribution in the QUANTILE function has the following form:

\[
\text{QUANTILE}('CONMAXPOI', p, \lambda, \nu)
\]

- \( p \) is a real number between 0 and 1, inclusively.
- \( \lambda \) is similar to the mean, as in the Poisson distribution.
- \( \nu \) is a dispersion parameter.

For more information, see “Conway-Maxwell-Poisson” distribution in the PDF function on page 482.

For more information about the distributions that are listed in the table, see “PDF Function” on page 477.

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>y=quantile('BERN', .75, .25);</td>
<td>0</td>
</tr>
<tr>
<td>y=quantile('BETA', 0.1, 3, 4);</td>
<td>0.2009088789</td>
</tr>
<tr>
<td>y=quantile('BINOM', .4, .5, 10);</td>
<td>5</td>
</tr>
<tr>
<td>y=quantile('CAUCHY', .85);</td>
<td>1.9626105055</td>
</tr>
<tr>
<td>y=quantile('CHISQ', .6, 11);</td>
<td>11.529833841</td>
</tr>
<tr>
<td>y=quantile('CONMAXPOI', .2, 2.3, .4);</td>
<td>5</td>
</tr>
<tr>
<td>y=quantile('EXPO', .6);</td>
<td>0.9162907319</td>
</tr>
<tr>
<td>SAS Statement</td>
<td>Result</td>
</tr>
<tr>
<td>---------------------------------------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td><code>y=quantile('F', .8, 2, 3);</code></td>
<td>2.8860266073</td>
</tr>
<tr>
<td><code>y=quantile('GAMMA', .4, 3);</code></td>
<td>2.285076904</td>
</tr>
<tr>
<td><code>y=quantile('GENPOISSON', .9, 1, .7);</code></td>
<td>9</td>
</tr>
<tr>
<td><code>y=quantile('HYPER', .5, 200, 50, 10);</code></td>
<td>2</td>
</tr>
<tr>
<td><code>y=quantile('LAPLACE', .8);</code></td>
<td>0.9162907319</td>
</tr>
<tr>
<td><code>y=quantile('LOGISTIC', .7);</code></td>
<td>0.8472978604</td>
</tr>
<tr>
<td><code>y=quantile('LOGNORMAL', .5);</code></td>
<td>1</td>
</tr>
<tr>
<td><code>y=quantile('NEGIB', .5, .5, 2);</code></td>
<td>1</td>
</tr>
<tr>
<td><code>y=quantile('NORMAL', .975);</code></td>
<td>1.9599639845</td>
</tr>
<tr>
<td><code>y=quantile('NORMALMIX', 0.5, 1);</code></td>
<td></td>
</tr>
<tr>
<td><code>y=quantile('PARETO', .01, 1);</code></td>
<td>1.0101010101</td>
</tr>
<tr>
<td><code>y=quantile('POISSON', .9, 1);</code></td>
<td>2</td>
</tr>
<tr>
<td><code>y=quantile('T', .8, 5);</code></td>
<td>0.9195437802</td>
</tr>
<tr>
<td><code>y=quantile('TWEEDIE', .8, 5);</code></td>
<td>1.2611198197</td>
</tr>
<tr>
<td><code>y=quantile('UNIFORM', 0.25);</code></td>
<td>0.25</td>
</tr>
<tr>
<td><code>y=quantile('WALD', .6, 2);</code></td>
<td>0.9526209927</td>
</tr>
<tr>
<td><code>y=quantile('WEIBULL', .6, 2);</code></td>
<td>0.9572307621</td>
</tr>
</tbody>
</table>

**See Also**

**Functions:**

- “CDF Function” on page 153
- “LOGCDF Function” on page 416
- “LOGPDF Function” on page 418
- “LOGSDF Function” on page 420
- “PDF Function” on page 477
- “SDF Function” on page 596
- “SQUANTILE Function” on page 1062
QUOTE Function

Adds double quotation marks to a character value.

**Category:** Character

**Restriction:** This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility.

**Syntax**

\[
\text{QUOTE}(\text{argument-1}, \text{argument-2})
\]

**Required Arguments**

- **argument-1**: specifies a character constant, variable, or expression.

- **argument-2**: specifies a quoting character, which is a single or double quotation mark. Other characters are ignored and the double quotation mark is used. The double quotation mark is the default.

**Details**

**Length of Returned Variable**

In a DATA step, if the QUOTE function returns a value to a variable that has not previously been assigned a length, then that variable is given a length of 200 bytes.

**The Basics**

The QUOTE function adds double quotation marks, the default character, to a character value. If double quotation marks are found within the argument, they are doubled in the output.

The length of the receiving variable must be long enough to contain the argument (including trailing blanks), leading and trailing quotation marks, and any embedded quotation marks that are doubled. For example, if the argument is ABC followed by three trailing blanks, then the receiving variable must have a length of at least eight to hold “ABC###”. (The character # represents a blank space.) If the receiving field is not long enough, the QUOTE function returns a blank string and writes an invalid argument note to the log.

**Example**

The following examples demonstrate the functionality of the QUOTE function:
<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>data <em>null</em>;</td>
<td>a=&quot;A&quot;&quot;B&quot;</td>
</tr>
<tr>
<td>x='A''B';</td>
<td></td>
</tr>
<tr>
<td>a=quote(x);</td>
<td></td>
</tr>
<tr>
<td>put a=;</td>
<td></td>
</tr>
<tr>
<td>run;</td>
<td></td>
</tr>
<tr>
<td>data <em>null</em>;</td>
<td>b=&quot;A'B&quot;</td>
</tr>
<tr>
<td>x='A''B';</td>
<td></td>
</tr>
<tr>
<td>b=quote(x);</td>
<td></td>
</tr>
<tr>
<td>put b=;</td>
<td></td>
</tr>
<tr>
<td>run;</td>
<td></td>
</tr>
<tr>
<td>data <em>null</em>;</td>
<td>c=&quot;Paul's&quot;</td>
</tr>
<tr>
<td>x='Paul''s';</td>
<td></td>
</tr>
<tr>
<td>c=quote(x);</td>
<td></td>
</tr>
<tr>
<td>put c=;</td>
<td></td>
</tr>
<tr>
<td>run;</td>
<td></td>
</tr>
<tr>
<td>data <em>null</em>;</td>
<td>d=&quot;Paul's Catering Service&quot;</td>
</tr>
<tr>
<td>x='Paul''s Catering Service';</td>
<td></td>
</tr>
<tr>
<td>d=quote(trim(x));</td>
<td></td>
</tr>
<tr>
<td>put d=;</td>
<td></td>
</tr>
<tr>
<td>run;</td>
<td></td>
</tr>
<tr>
<td>data <em>null</em>;</td>
<td>e=&quot;Catering Service Center&quot;</td>
</tr>
<tr>
<td>x='Catering Service Center';</td>
<td></td>
</tr>
<tr>
<td>e=quote(trim(x));</td>
<td></td>
</tr>
<tr>
<td>put e=;</td>
<td></td>
</tr>
<tr>
<td>run;</td>
<td></td>
</tr>
<tr>
<td>data <em>null</em>;</td>
<td>f=&quot;Catering Service Center&quot;</td>
</tr>
<tr>
<td>x='Catering Service Center';</td>
<td></td>
</tr>
<tr>
<td>f=quote(x);</td>
<td></td>
</tr>
<tr>
<td>put f=;</td>
<td></td>
</tr>
<tr>
<td>run;</td>
<td></td>
</tr>
<tr>
<td>data <em>null</em>;</td>
<td>g=&quot;Catering Service Center&quot;</td>
</tr>
<tr>
<td>x='Catering Service Center';</td>
<td></td>
</tr>
<tr>
<td>g=quote(x);</td>
<td></td>
</tr>
<tr>
<td>put g=;</td>
<td></td>
</tr>
<tr>
<td>run;</td>
<td></td>
</tr>
<tr>
<td>data <em>null</em>;</td>
<td>h=&quot;Catering Service Center&quot;</td>
</tr>
<tr>
<td>x='Catering Service Center';</td>
<td></td>
</tr>
<tr>
<td>h=quote(x);</td>
<td></td>
</tr>
<tr>
<td>put h=;</td>
<td></td>
</tr>
<tr>
<td>run;</td>
<td></td>
</tr>
<tr>
<td>data <em>null</em>;</td>
<td>&quot;abc&quot;</td>
</tr>
<tr>
<td>x=quote('abc');</td>
<td></td>
</tr>
<tr>
<td>put x=;</td>
<td></td>
</tr>
<tr>
<td>run;</td>
<td></td>
</tr>
</tbody>
</table>
SAS Statement | Result
---|---
data_null;
   x=quote('abc','"');
   put x=;
run;

'abc'

Note: The second argument contains a single quotation mark. In order to be passed to the function in the DATA step, the argument is specified in the DATA step as double quotation mark, single quotation mark, double quotation mark. The argument could have also been specified as four single quotation marks (that is, a quoted string that uses single quotation marks). The quoted value is an escaped single quotation mark represented as two single quotation marks.

RAND Function

Generates random numbers from a distribution that you specify.

Category: Random Number

Syntax

RAND(distribution, parameter-1, ..., parameter-k)

Required Arguments

distribution

is a character constant, variable, or expression that identifies the distribution. Valid distributions are as follows:

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bernoulli</td>
<td>BERNOLLI</td>
</tr>
<tr>
<td>Beta</td>
<td>BETA</td>
</tr>
<tr>
<td>Binomial</td>
<td>BINOMIAL</td>
</tr>
<tr>
<td>Cauchy</td>
<td>CAUCHY</td>
</tr>
<tr>
<td>Chi-Square</td>
<td>CHISQUARE</td>
</tr>
<tr>
<td>Erlang</td>
<td>ERLANG</td>
</tr>
<tr>
<td>Exponential</td>
<td>EXPONENTIAL</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>Distribution</td>
<td>Argument</td>
</tr>
<tr>
<td>-----------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>Gamma</td>
<td>GAMMA</td>
</tr>
<tr>
<td>Geometric</td>
<td>GEOMETRIC</td>
</tr>
<tr>
<td>Hypergeometric</td>
<td>HYPERGEOMETRIC</td>
</tr>
<tr>
<td>Lognormal</td>
<td>LOGNORMAL</td>
</tr>
<tr>
<td>Negative Binomial</td>
<td>NEGBINOMIAL</td>
</tr>
<tr>
<td>Normal</td>
<td>NORMAL</td>
</tr>
<tr>
<td>Poisson</td>
<td>POISSON</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>Tabled</td>
<td>TABLE</td>
</tr>
<tr>
<td>Triangular</td>
<td>TRIANGLE</td>
</tr>
<tr>
<td>Uniform</td>
<td>UNIFORM</td>
</tr>
<tr>
<td>Weibull</td>
<td>WEIBULL</td>
</tr>
</tbody>
</table>

**Note:** Except for T and F, you can minimally identify any distribution by its first four characters.

*parameter-1, …, parameter-k*

are optional numeric constants, variables, or expressions that specify the values of *shape*, *location*, or *scale* parameters appropriate for the specific distribution.

**See**  “Details” on page 556

## Details

### Generating Random Numbers

The RAND function generates random numbers from various continuous and discrete distributions. Wherever possible, the simplest form of the distribution is used.

The RAND function uses the Mersenne-Twister random number generator (RNG) that was developed by Matsumoto and Nishimura (1998). The random number generator has a very long period ($2^{19937} - 1$) and very good statistical properties. The period is a Mersenne prime, which contributes to the naming of the RNG. The algorithm is a twisted generalized feedback shift register (TGFSR) that explains the latter part of the name. The TGFSR gives the RNG a very high order of equidistribution (623-dimensional with 32-bit accuracy), which means that there is a very small correlation between successive vectors of 623 pseudo-random numbers.

The RAND function is started with a single seed. However, the state of the process cannot be captured by a single seed. You cannot stop and restart the generator from its stopping point.
If the initial seed is exactly divisible by 8192, the RAND function uses the 2002 initialization algorithm (Matsumoto and Nishimura 2002). Otherwise, RAND uses the 1998 initialization algorithm.

**Reproducing a Random Number Stream**

If you want to create reproducible streams of random numbers, use the CALL STREAMINIT routine to specify a seed value for random number generation. Use the CALL STREAMINIT routine once per DATA step before any invocation of the RAND function. If you omit the call to the CALL STREAMINIT routine (or if you specify a nonpositive seed value in the CALL STREAMINIT routine), then RAND obtains an initial seed from the Intel RdRand instruction on machines that support the instruction, or from the current date-time value on machines that do not support RdRand. For more information, see CALL STREAMINIT. “Example: Creating a Reproducible Stream of Random Numbers” on page 141

**Duplicate Values in the Mersenne-Twister RNG Algorithm**

The Mersenne-Twister RNG algorithm has an extremely long period, but this does not imply that large random samples are devoid of duplicate values. The RAND function returns at most $2^{32}$ distinct values. In a random uniform sample of size $10^5$, the chance of drawing at least one duplicate is greater than 50%. The expected number of duplicates in a random uniform sample of size $\text{M}$ is approximately $\text{M}^{2/33}$ when $\text{M}$ is much less than $2^{32}$. For example, you should expect about 115 duplicates in a random uniform sample of size $\text{M}=10^9$. These results are consequences of the famous “birthday matching problem” in probability theory.

**Bernoulli Distribution**

$x=\text{RAND}('BERNOULLI', p)$

**Arguments**

$x$

is an observation from the distribution with the following probability density function:

$$f(x) = \begin{cases} 
1 & p = 0, x = 0 \\
1 - p & 0 < p < 1, x = 0, 1 \\
1 & p = 1, x = 1 
\end{cases}$$

Range $x = 0, 1$

$p$

is a numeric constant, variable, or expression that specifies the probability of success.

Range $0 \leq p \leq 1$

**Beta Distribution**

$x=\text{RAND}('BETA', a, b)$

**Arguments**

$x$

is an observation from the distribution with the following probability density function:
\[ f(x) = \frac{\Gamma(a + b)}{\Gamma(a)\Gamma(b)} x^{a-1} (1 - x)^{b-1} \]

Range \( 0 < x < 1 \)

\( a \) is a numeric constant, variable, or expression that specifies a shape parameter.

Range \( a > 0 \)

\( b \) is a numeric constant, variable, or expression that specifies a shape parameter.

Range \( b > 0 \)

**Binomial Distribution**

\( x = \text{RAND}('\text{BINOMIAL}', p, n) \)

**Arguments**

\( x \)

is an integer observation from the distribution with the following probability density function:

\[
\begin{cases} 
  1 & p = 0, x = 0 \\
  \binom{n}{x} p^x (1 - p)^{n-x} & 0 < p < 1, x = 0, \ldots, n \\
  1 & p = 1, x = n 
\end{cases}
\]

Range \( x = 0, 1, \ldots, n \)

\( p \)

is a numeric constant, variable, or expression that specifies the probability of success.

Range \( 0 \leq p \leq 1 \)

\( n \)

is an integer parameter that counts the number of independent Bernoulli trials.

Range \( n = 1, 2, \ldots \)

**Cauchy Distribution**

\( x = \text{RAND('CAUCHY')} \)

**Arguments**

\( x \)

is an observation from the distribution with the following probability density function:

\[
f(x) = \frac{1}{\pi \left(1 + x^2\right)}
\]
Chi-Square Distribution

\( x = \text{RAND}('\text{CHISQUARE}', df) \)

Arguments

- \( x \) is an observation from the distribution with the following probability density function:
  \[
  f(x) = \frac{2^{-df/2}}{I\left(\frac{df}{2}\right)} x^{df/2-1} e^{-x/2}
  \]

Range \( x > 0 \)

- \( df \) is a numeric constant, variable, or expression that specifies the degrees of freedom.
  Range \( df > 0 \)

Erlang Distribution

\( x = \text{RAND}('\text{ERLANG}', a) \)

Arguments

- \( x \) is an observation from the distribution with the following probability density function:
  \[
  f(x) = \frac{1}{I(a)} x^{a-1} e^{-x}
  \]

Range \( x > 0 \)

- \( a \) is a numeric constant, variable, or expression that specifies a shape parameter.
  Range \( a = 1, 2, ... \)

Exponential Distribution

\( x = \text{RAND}('\text{EXPONENTIAL}', \sigma) \)

Arguments

- \( x \) is an observation from the distribution with the following probability density function:
  \[
  f(x) = \frac{e^{-x/\sigma}}{\sigma}
  \]

Range \( x > 0 \)
σ
is a numeric constant, variable, or expression that specifies a shape parameter.

Default  The optional shape parameter σ > 0 has the default value σ = 1.

**F Distribution**

$x = \text{RAND}(F', n, d)$

**Arguments**

$x$
is an observation from the distribution with the following probability density function:

$$f(x) = \frac{I\left(\frac{n + d}{2}\right)}{I\left(\frac{n}{2}\right)f\left(\frac{d}{2}\right)} \frac{n^{n/2} d^{d/2} x^{n/2 - 1}}{(d + nx)^{(n + d)/2}}$$

Range  $x > 0$

$n$
is a numeric constant, variable, or expression that specifies the numerator degrees of freedom.

Range  $n > 0$

$d$
is a numeric constant, variable, or expression that specifies the denominator degrees of freedom.

Range  $d > 0$

**Gamma Distribution**

$x = \text{RAND}(\text{GAMMA}', a <,\lambda>)$

**Arguments**

$x$
is an observation from the distribution with the following probability density function:

$$f(x) = \frac{x^{a-1}}{\lambda^a \Gamma(a)} e^{-x/\lambda}$$

Range  $x > 0$

$a$
is a numeric constant, variable, or expression that specifies a shape parameter.

Range  $a > 0$

$\lambda$
is a numeric constant, variable, or expression that specifies a shape parameter.

Default  The optional shape parameter $\lambda > 0$ has the default value $\lambda = 1$. 
Geometric Distribution  
\[ x = \text{RAND('GEOMETRIC', } p) \]

**Arguments**

- **x**
  - is an integer count that denotes the number of trials that are needed to obtain one success. \( X \) is an integer observation from the distribution with the following probability density function:
  
  \[ f(x) = \begin{cases} 
    (1 - p)^{x-1}p & 0 < p < 1, x = 1, 2, \ldots \\
    1 & p = 1, x = 1 
  \end{cases} \]

  \[ \text{Range} \quad x = 1, 2, \ldots \]

- **p**
  - is a numeric constant, variable, or expression that specifies the probability of success.
  
  \[ \text{Range} \quad 0 < p \leq 1 \]

Hypergeometric Distribution  
\[ x = \text{RAND('HYPER', } N, R, n) \]

**Arguments**

- **x**
  - is an integer observation from the distribution with the following probability density function:
  
  \[ f(x) = \frac{\binom{R}{x} \binom{N - R}{n - x}}{\binom{N}{n}} \]

  \[ \text{Range} \quad x = \max(0, (n - (N - R))), \ldots, \min(n, R) \]

- **N**
  - is a numeric constant, variable, or expression that specifies an integer population size parameter.
  
  \[ \text{Range} \quad N = 1, 2, \ldots \]

- **R**
  - is a numeric constant, variable, or expression that specifies an integer number of items in the category of interest.
  
  \[ \text{Range} \quad R = 0, 1, \ldots, N \]

- **n**
  - is a numeric constant, variable, or expression that specifies an integer sample size parameter.
  
  \[ \text{Range} \quad n = 1, 2, \ldots, N \]
The hypergeometric distribution is a mathematical formalization of an experiment in which you draw \( n \) balls from an urn that contains \( N \) balls, \( R \) of which are red. The hypergeometric distribution is the distribution of the number of red balls in the sample of \( n \).

**Lognormal Distribution**

\[ x = \text{RAND}('\text{LOGNORMAL}', \langle \theta, \lambda \rangle) \]

**Arguments**

- \( x \)
  - is an observation from the distribution with the following probability density function:
    \[
    f(x) = \frac{1}{x \lambda \sqrt{2\pi}} \exp\left(-\frac{(\ln(x) - \theta)^2}{2\lambda^2}\right)
    \]
  - Range \( x > 0 \)

- \( \theta \)
  - is a numeric constant, variable, or expression that specifies a shape parameter.
  - Default The optional log-scale parameter \( \theta \) has the default value \( \theta = 0 \).

- \( \lambda \)
  - is a numeric constant, variable, or expression that specifies a shape parameter.
  - Default The optional log-scale parameter \( \lambda \) has the default value \( \lambda = 1 \).

**Negative Binomial Distribution**

\[ x = \text{RAND}('\text{NEGBINOMIAL}', p, k) \]

**Arguments**

- \( x \)
  - is an integer observation from the distribution with the following probability density function:
    \[
    f(x) = \begin{cases} 
    \binom{x + k - 1}{k - 1} (1 - p)^x p^k & 0 < p < 1, x = 0, 1, ... \\
    1 & p = 1, x = 0 
    \end{cases}
    \]
  - Range \( x = 0, 1, ... \)

- \( k \)
  - is an integer parameter that is the number of successes. However, noninteger \( k \) values are allowed as well.
  - Range \( k = 1, 2, ... \)

- \( p \)
  - is a numeric constant, variable, or expression that specifies the probability of success.
  - Range \( 0 < p \leq 1 \)
The negative binomial distribution is the distribution of the number of failures before \( k \) successes occur in sequential independent trials, all with the same probability of success, \( p \).

**Normal Distribution**

\[ x = \text{RAND}('\text{NORMAL}', \langle \theta, \lambda \rangle) \]

**Arguments**

- \( x \)
  - is an observation from the normal distribution with a mean of \( \theta \) and a standard deviation of \( \lambda \) that has the following probability density function:
  
  \[
  f(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{(x - \theta)^2}{2\lambda^2}\right)
  \]

  **Range** \( -\infty < x < \infty \)

- \( \theta \)
  - is a numeric constant, variable, or expression that specifies a mean parameter.
  - Default \( 0 \)

- \( \lambda \)
  - is a numeric constant, variable, or expression that specifies a standard deviation parameter.
  - Default \( 1 \)
  - **Range** \( \lambda > 0 \)

**Poisson Distribution**

\[ x = \text{RAND}('\text{POISSON}', m) \]

**Arguments**

- \( x \)
  - is an integer observation from the distribution with the following probability density function:
  
  \[
  f(x) = \frac{m^x e^{-m}}{x!}
  \]

  **Range** \( x = 0, 1, ... \)

- \( m \)
  - is a numeric constant, variable, or expression that specifies a mean parameter.
  - **Range** \( m > 0 \)

**T Distribution**

\[ x = \text{RAND}('T', df) \]

**Arguments**
\( x \)

is an observation from the distribution with the following probability density function:

\[
f(x) = \frac{\Gamma\left(\frac{df+1}{2}\right)}{\sqrt{df \pi} \Gamma\left(\frac{df}{2}\right)} \left(1 + \frac{x^2}{df}\right)^{-\frac{df+1}{2}}
\]

Range \(-\infty < x < \infty\)

\( df \)

is a numeric constant, variable, or expression that specifies the degrees of freedom.

Range \(df > 0\)

**Tabled Distribution**

\( x = \text{RAND}('\text{TABLE}', p1, p2, ...) \)

**Arguments**

\( x \)

is an integer observation from one of the following distributions:

If \( \sum_{i=1}^{n} p_i < 1 \), then \( x \) is an observation from this probability density function:

\[
f(i) = p_i, \quad i = 1, 2, \ldots, n
\]

and

\[
f(n + 1) = 1 - \sum_{i=1}^{n} p_i
\]

If \( \sum_{i=1}^{n} p_i \geq 1 \) for some index \( n \), then \( x \) is an observation from this probability density function:

\[
f(i) = p_i, \quad i = 1, 2, \ldots, n - 1
\]

and

\[
f(n) = 1 - \sum_{i=1}^{n-1} p_i
\]

\( p1, p2, ... \)

are numeric probability values.

Range \( 0 \leq p1, p2, ... \leq 1 \)

Restriction The maximum number of probability parameters depends on your operating environment, but the maximum number of parameters is at least 32,767.

The tabled distribution takes on the values \( 1, 2, ..., n \) with specified probabilities.

Note: By using the FORMAT statement, you can map the set \{1, 2, ..., n\} to any set of \( n \) or fewer elements.
**Triangular Distribution**

\[ x = \text{RAND('TRIANGLE', } h) \]

**Arguments**

\( x \)

is an observation from the distribution with the following probability density function:

\[
f(x) = \begin{cases} 
\frac{2x}{h} & 0 \leq x \leq h \\
\frac{2(1-x)}{1-h} & h < x \leq 1 
\end{cases}
\]

In this equation, \( 0 \leq h \leq 1 \).

**Range** \( 0 \leq x \leq 1 \)

**Note** The distribution can be easily shifted and scaled.

\( h \)

is a numeric constant, variable, or expression that specifies the horizontal location of the peak of the triangle.

**Range** \( 0 \leq h \leq 1 \)

**Uniform Distribution**

\[ x = \text{RAND('UNIFORM', } <a, b>) \]

**Arguments**

\( x \)

is an observation from the distribution with the following probability density function:

\[
f(x) = \begin{cases} 
\frac{1}{|b-a|} & \text{if } a = b \\
\frac{1}{b-a} & \text{if } a \neq b 
\end{cases}
\]

**Range** The range of \( x \) is \( a \leq x \leq b \). The parameters \( a \) and \( b \) default values are \( a=0 \) and \( b=1 \). You must specify values for both \( a \) and \( b \) if you do not want to use the default values.

\( a \)

specifies a value in the probability density function.

**Default** 0

\( b \)

specifies a value in the probability density function.

**Default** 1

The uniform random number generator that the RAND function uses is the Mersenne-Twister (Matsumoto and Nishimura 1998). This generator has a period of \( 2^{19937} - 1 \)
and 623-dimensional equidistribution up to 32-bit accuracy. This algorithm underlies the generators for the other available distributions in the RAND function.

**Weibull Distribution**

\[ x = \text{RAND}('\text{WEIBULL}', a, b) \]

**Arguments**

- \( x \)
  is an observation from the distribution with the following probability density function:
  \[ f(x) = \frac{a}{b^a} x^{a-1} e^{-\left(\frac{x}{b}\right)^a} \]
  
  **Range**  \( x \geq 0 \)

- \( a \)
  is a numeric constant, variable, or expression that specifies a shape parameter.
  **Range**  \( a > 0 \)

- \( b \)
  is a numeric constant, variable, or expression that specifies a scale parameter.
  **Range**  \( b > 0 \)

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=rand('BERN', .75);</td>
<td>0</td>
</tr>
<tr>
<td>x=rand('BETA', 3, 0.1);</td>
<td>.99920</td>
</tr>
<tr>
<td>x=rand('BINOM', 0.75, 10);</td>
<td>10</td>
</tr>
<tr>
<td>x=rand('CAUCHY');</td>
<td>-1.41525</td>
</tr>
<tr>
<td>x=rand('CHISQ', 22);</td>
<td>25.8526</td>
</tr>
<tr>
<td>x=rand('ERLANG', 7);</td>
<td>7.67039</td>
</tr>
<tr>
<td>x=rand('EXPO');</td>
<td>1.48847</td>
</tr>
<tr>
<td>x=rand('F', 12, 322);</td>
<td>1.99647</td>
</tr>
<tr>
<td>x=rand('GAMMA', 7.25);</td>
<td>6.59588</td>
</tr>
<tr>
<td>x=rand('GEOM', 0.02);</td>
<td>43</td>
</tr>
<tr>
<td>SAS Statement</td>
<td>Result</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>x=rand('HYPER', 10, 3, 5);</td>
<td>1</td>
</tr>
<tr>
<td>x=rand('LOGN');</td>
<td>0.66522</td>
</tr>
<tr>
<td>x=rand('NEGEB', 0.8, 5);</td>
<td>33</td>
</tr>
<tr>
<td>x=rand('NORMAL');</td>
<td>1.03507</td>
</tr>
<tr>
<td>x=rand('POISSON', 6.1);</td>
<td>6</td>
</tr>
<tr>
<td>x=rand('T', 4);</td>
<td>2.44646</td>
</tr>
<tr>
<td>x=rand('TABLE', .2, .5);</td>
<td>2</td>
</tr>
<tr>
<td>x=rand('TRIANGLE', 0.7);</td>
<td>.63811</td>
</tr>
<tr>
<td>x=rand('UNIFORM');</td>
<td>.96234</td>
</tr>
<tr>
<td>x=rand('WEIB', 0.25, 2.1);</td>
<td>6.55778</td>
</tr>
</tbody>
</table>

**See Also**

**CALL Routines:**

- “CALL STREAMINIT Routine” on page 139

**References**


### RANGE Function

Returns the range of the nonmissing values.

**Category:** Descriptive Statistics

**Syntax**

```
RANGE(argument-1 <, ...argument-n>)
```

**Required Argument**

`argument`

specifies a numeric constant, variable, or expression. At least one nonmissing argument is required. Otherwise, the function returns a missing value. The argument list can consist of a variable list, which is preceded by `OF`.

**Details**

The RANGE function returns the difference between the largest and the smallest of the nonmissing arguments.

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0=range(., .);</td>
<td>.</td>
</tr>
<tr>
<td>x1=range(-2, 6, 3);</td>
<td>8</td>
</tr>
<tr>
<td>x2=range(2, 6 ,3, .);</td>
<td>4</td>
</tr>
<tr>
<td>x3=range(1, 6, 3, 1);</td>
<td>5</td>
</tr>
<tr>
<td>x4=range(of x1-x3);</td>
<td>4</td>
</tr>
</tbody>
</table>

### RANK Function

Returns the position of a character in the ASCII collating sequence.

**Category:** Character
Restriction: This function is assigned an I18N Level 0 status, and is designed for SBCS data. Do not use this function to process DBCS or MBCS data. For more information, see Internationalization Compatibility.

Syntax

RANK(x)

Required Argument

x

specifies a character constant, variable, or expression that contains a character in the ASCII collating sequence. If the length of x is greater than 1, you receive the rank of the first character in the string.

Details

The RANK function returns an integer that represents the position of the character in the ASCII collating sequence. When more than one character is specified, the RANK function returns the position in the ASCII collating sequence for the first character.

Note: Any program that uses the RANK function with characters above ASCII 127 is not portable. (The hexadecimal notation is ‘7F’x.) The program is not portable because these characters are national characters and they vary from country to country.

Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result for ASCII</th>
</tr>
</thead>
<tbody>
<tr>
<td>n=rank(’A’);</td>
<td></td>
</tr>
<tr>
<td>put n;</td>
<td>65</td>
</tr>
</tbody>
</table>

See Also

Functions:

- “BYTE Function” on page 123
- “COLLATE Function” on page 179

**REPEAT Function**

Returns a character value that consists of the first argument repeated n+1 times.

Category: Character

Restriction: This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility.
Syntax

REPEAT(argument, n)

Required Arguments

argument
  specifies a character constant, variable, or expression.

n
  specifies the number of times to repeat argument.

Restriction
  n must be greater than or equal to 0.

Details

In a DATA step, if the REPEAT function returns a value to a variable that has not previously been assigned a length, then that variable is given a length of 200 bytes.

The REPEAT function returns a character value consisting of the first argument repeated n times. Thus, the first argument appears $n+1$ times in the result.

Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=repeat('ONE', 2);</td>
<td>ONEONEONE</td>
</tr>
<tr>
<td>put x;</td>
<td></td>
</tr>
</tbody>
</table>

REVERSE Function

Reverses a character string.

Category: Character

Restriction: This function is assigned an I18N Level 0 status, and is designed for SBCS data. Do not use this function to process DBCS or MBCS data. For more information, see Internationalization Compatibility.


Syntax

REVERSE(argument)

Required Argument

argument
  specifies a character constant, variable, or expression.
Details

In a DATA step, if the REVERSE function returns a value to a variable that has not previously been assigned a length, then that variable is given the length of the first argument.

The last character in the argument becomes the first character in the result, the next-to-last character in the argument becomes the second character in the result, and so on.

Note: Trailing blanks in the argument become leading blanks in the result.

Example

The following SAS statements produce this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>backward=reverse('xyz ');</td>
<td>zyx</td>
</tr>
<tr>
<td>put backward $5.;</td>
<td></td>
</tr>
</tbody>
</table>

RIGHT Function

Right aligns a character expression.

**Category:** Character

**Restriction:** This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility.

**Tip:** DBCS equivalent function is KRIGHT.

**Syntax**

\texttt{RIGHT(argument)}

**Required Argument**

\texttt{argument} specifies a character constant, variable, or expression.

**Details**

In a DATA step, if the RIGHT function returns a value to a variable that has not previously been assigned a length, then that variable is given the length of the first argument.

The RIGHT function returns an argument with trailing blanks moved to the start of the value. The length of the result is the same as the length of the argument.
Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>a='Due Date ';</td>
<td>Due Date</td>
</tr>
<tr>
<td>b=right(a);</td>
<td>Due Date</td>
</tr>
<tr>
<td>put a $10.;</td>
<td>Due Date</td>
</tr>
<tr>
<td>put b $10.;</td>
<td>Due Date</td>
</tr>
</tbody>
</table>

See Also

Functions:

- “COMPRESS Function” on page 191
- “LEFT Function” on page 405
- “TRIM Function” on page 640

RMS Function

Returns the root mean square of the nonmissing arguments.

Category: Descriptive Statistics

Syntax

RMS(argument <, argument, …>)

Required Argument

argument

is a numeric constant, variable, or expression.

Tip: The argument list can consist of a variable list, which is preceded by OF.

Details

The root mean square is the square root of the arithmetic mean of the squares of the values. If all the arguments are missing values, then the result is a missing value. Otherwise, the result is the root mean square of the nonmissing values.

Let $n$ be the number of arguments with nonmissing values, and let $x_1, x_2, \ldots, x_n$ be the values of those arguments. The root mean square is

$$\sqrt{\frac{x_1^2 + x_2^2 + \ldots + x_n^2}{n}}$$
Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1=rms(1, 7);</td>
<td>5</td>
</tr>
<tr>
<td>x2=rms(., 1, 5, 11);</td>
<td>7</td>
</tr>
<tr>
<td>x3=rms(of x1-x2);</td>
<td>6.0827625303</td>
</tr>
</tbody>
</table>

ROUND Function

Rounds the first argument to the nearest multiple of the second argument, or to the nearest integer when the second argument is omitted.

**Category:** Truncation

**Syntax**

```
ROUND(argument <, rounding-unit>)
```

**Required Argument**

*argument*

is a numeric constant, variable, or expression to be rounded.

**Optional Argument**

*rounding-unit*

is a positive, numeric constant, variable, or expression that specifies the rounding unit.

**Details**

**Basic Concepts**

The ROUND function rounds the first argument to a value that is very close to a multiple of the second argument. The result might not be an exact multiple of the second argument.

**Differences between Binary and Decimal Arithmetic**

Computers use binary arithmetic with finite precision. If you work with numbers that do not have an exact binary representation, computers often produce results that differ slightly from the results that are produced with decimal arithmetic.

For example, the decimal values 0.1 and 0.3 do not have exact binary representations. In decimal arithmetic, 3*0.1 is exactly equal to 0.3, but this equality is not true in binary arithmetic. As the following example shows, if you write these two values in SAS, they appear the same. If you compute the difference, however, you can see that the values are different.
data _null_;
  point_three=0.3;
  three_times_point_one=3*0.1;
  difference=point_three - three_times_point_one;
  put point_three= ;
  put three_times_point_one= ;
  put difference= ;
run;

SAS writes the following output to the log:

<table>
<thead>
<tr>
<th>point_three</th>
<th>three_times_point_one</th>
<th>difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3</td>
<td>0.3</td>
<td>-5.55112E-17</td>
</tr>
</tbody>
</table>

The Effects of Rounding

Rounding by definition finds an exact multiple of the rounding unit that is closest to the value to be rounded. For example, 0.33 rounded to the nearest tenth equals 3\*0.1 or 0.3 in decimal arithmetic. In binary arithmetic, 0.33 rounded to the nearest tenth equals 3\*0.1, and not 0.3, because 0.3 is not an exact multiple of one tenth in binary arithmetic.

The ROUND function returns the value that is based on decimal arithmetic, even though this value is sometimes not the exact, mathematically correct result. In the example `ROUND(0.33,0.1)`, ROUND returns 0.3 and not 3\*0.1.

Expressing Binary Values

If the characters "0.3" appear as a constant in a SAS program, the value is computed by the standard informat as 3/10. To be consistent with the standard informat, `ROUND(0.33,0.1)` computes the result as 3/10, and the following statement produces the results that you would expect.

```sas
if round(x, 0.1)=0.3 then
  ... more SAS statements ...
```

However, if you use the variable Y instead of the constant 0.3, as the following statement shows, the results might be unexpected depending on how the variable Y is computed.

```sas
if round(x, 0.1)=y then
  ... more SAS statements ...
```

If SAS reads Y as the characters "0.3" using the standard informat, the result is the same as if a constant 0.3 appeared in the IF statement. If SAS reads Y with a different informat, or if a program other than SAS reads Y, then there is no guarantee that the characters "0.3" would produce a value of exactly 3/10. Imprecision can also be caused by computation involving numbers that do not have exact binary representations, or by porting data sets from one operating environment to another that has a different floating-point representation.

If you know that Y is a decimal number with one decimal place, but are not certain that Y has exactly the same value as would be produced by the standard informat, it is better to use the following statement:

```sas
if round(x, 0.1)=round(y, 0.1) then
  ... more SAS statements ...
```
**Testing for Approximate Equality**

You should not use the `ROUND` function as a general method to test for approximate equality. Two numbers that differ only in the least significant bit can round to different values if one number rounds down and the other number rounds up. Testing for approximate equality depends on how the numbers have been computed. If both numbers are computed to high relative precision, you could test for approximate equality by using the `ABS` and the `MAX` functions, as the following example shows.

```sas
if abs(x-y) <= 1e-12 * max(abs(x), abs(y)) then
    ... more SAS statements ...
```

**Producing Expected Results**

In general, `ROUND(argument, rounding-unit)` produces the result that you expect from decimal arithmetic if the result has no more than nine significant digits and any of the following conditions are true:

- The rounding unit is an integer.
- The rounding unit is a power of 10 greater than or equal to 1e-15. (If the rounding unit is less than one, `ROUND` treats it as a power of 10 if the reciprocal of the rounding unit differs from a power of 10 in at most the three or four least significant bits.)
- The result that you expect from decimal arithmetic has no more than four decimal places.

For example:

```sas
data rounding;
  d1=round(1234.56789, 100)     - 1200;
  d2=round(1234.56789, 10)      - 1230;
  d3=round(1234.56789, 1)       - 1235;
  d4=round(1234.56789, .1)      - 1234.6;
  d5=round(1234.56789, .01)     - 1234.57;
  d6=round(1234.56789, .001)    - 1234.568;
  d7=round(1234.56789, .0001)   - 1234.5679;
  d8=round(1234.56789, .00001)  - 1234.56789;
  d9=round(1234.56789, .1111)   - 1234.5432;
  /* d10 has too many decimal places in the value for rounding-unit. */
  d10 = round(1234.56789, .11111) - 1234.54321;
run;
proc print data=rounding noobs;
run;
```

**Figure 2.29**  Output from Rounding Based on the Value of the Rounding Unit
When the Rounding Unit Is the Reciprocal of an Integer

When the rounding unit is the reciprocal of an integer, the ROUND function computes the result by dividing by the integer. (ROUND treats the rounding unit as a reciprocal of an integer if the reciprocal of the rounding unit differs from an integer in at most the three or four least significant bits.) Therefore, you can safely compare the result from ROUND with the ratio of two integers, but not with a multiple of the rounding unit. For example:

```sas
data rounding2;
  drop pi unit;
  pi=arccos(-1);
  unit=1/7;
  d1=round(pi, unit) - 22/7;
  d2=round(pi, unit) - 22*unit;
run;
proc print data=rounding2 noobs;
run;
```

*Figure 2.30  Output from Rounding by the Reciprocal of an Integer*

Computing Results in Special Cases

The ROUND function computes the result by multiplying an integer by the rounding unit when all of the following conditions are true:

- The rounding unit is not an integer.
- The rounding unit is not a power of 10.
- The rounding unit is not the reciprocal of an integer.
- The result that you expect from decimal arithmetic has no more than four decimal places.

For example:

```sas
data _null_;  
  difference=round(1234.56789, .11111) - 11111*.11111;  
  put difference=;  
run;
```

SAS writes the following output to the log:

```
difference=0
```

Computing Results When the Value Is Halfway between Multiples of the Rounding Unit

When the value to be rounded is approximately halfway between two multiples of the rounding unit, the ROUND function rounds up the absolute value and restores the original sign. For example:
data test;
  do i=8 to 17;
    value=0.5 - 10**(-i);
    round=round(value);
    output;
  end;
  do i=8 to 17;
    value=-0.5 + 10**(-i);
    round=round(value);
    output;
  end;
run;
proc print data=test noobs;
  format value 19.16;
run;

*Figure 2.31  Output from Rounding When Values Are Halfway between Multiples of the Rounding Unit*

<table>
<thead>
<tr>
<th>i</th>
<th>value</th>
<th>round</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>0.499999999900000000</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0.499999999900000000</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0.499999999900000000</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0.499999999900000000</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>0.499999999900000000</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>0.499999999900000000</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>0.499999999900000000</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>0.499999999900000000</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>0.500000000000000000</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>0.500000000000000000</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>-0.499999999900000000</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>-0.499999999900000000</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>-0.499999999900000000</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>-0.499999999900000000</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>-0.499999999900000000</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>-0.499999999900000000</td>
<td>-1</td>
</tr>
<tr>
<td>14</td>
<td>-0.499999999900000000</td>
<td>-1</td>
</tr>
<tr>
<td>15</td>
<td>-0.499999999900000000</td>
<td>-1</td>
</tr>
<tr>
<td>16</td>
<td>-0.500000000000000000</td>
<td>-1</td>
</tr>
<tr>
<td>17</td>
<td>-0.500000000000000000</td>
<td>-1</td>
</tr>
</tbody>
</table>
The approximation is relative to the size of the value to be rounded, and is computed in a manner that is shown in the following DATA step. This DATA step code will not always produce results exactly equivalent to the ROUND function.

```sas
data testfile;
  do i=1 to 17;
    value=0.5 - 10**{-i};
    epsilon=min(1e-6, value * 1e-12);
    temp=value + .5 + epsilon;
    fraction=modz(temp, 1);
    round=temp - fraction;
    output;
  end;
run;
proc print data=testfile noobs;
  format value 19.16;
run;
```

### Comparisons

The ROUND, ROUNDE, and ROUNDZ functions are similar with four exceptions:

- **ROUND** returns the multiple with the larger absolute value when the first argument is approximately halfway between the two nearest multiples of the second argument.

- **ROUNDE** returns an even multiple when the first argument is approximately halfway between the two nearest multiples of the second argument.

- **ROUNDZ** returns an even multiple when the first argument is exactly halfway between the two nearest multiples of the second argument.

- When the rounding unit is less than one and not the reciprocal of an integer, the result that is returned by ROUNDZ might not agree exactly with the result from decimal arithmetic. ROUND and ROUNDE perform extra computations, called fuzzing, to try to make the result agree with decimal arithmetic in the most common situations. ROUNDZ does not fuzz the result.

### Example

The following example compares the results that are returned by the ROUND function with the results that are returned by the ROUNDE function. The output was generated in the Windows operating environment.

```sas
data results;
  do x=0 to 4 by .25;
    Rounde=rounde(x);
    Round=round(x);
    output;
  end;
run;
proc print data=results noobs;
run;
```
See Also

Functions:

- “CEIL Function” on page 171
- “CEILZ Function” on page 172
- “FLOOR Function” on page 301
- “FLOORZ Function” on page 302
- “INT Function” on page 340
- “INTZ Function” on page 385
- “ROUND Function” on page 580
- “ROUNDE Function” on page 582

Figure 2.32  Output That Is Returned by the ROUND and ROUNDE Functions
ROUND Function

Rounds the first argument to the nearest multiple of the second argument, and returns an even multiple when the first argument is halfway between the two nearest multiples.

**Category:** Truncation

**Syntax**

```
ROUND(argu,entn <-, rounding-unit>)
```

**Required Argument**

`argument`

is a numeric constant, variable, or expression to be rounded.

**Optional Argument**

`rounding-unit`

is a positive, numeric constant, variable, or expression that specifies the rounding unit.

**Details**

The ROUND function rounds the first argument to the nearest multiple of the second argument. If you omit the second argument, ROUND uses a default value of 1 for `rounding-unit`.

**Comparisons**

The ROUND, ROUNDE, and ROUNDZ functions are similar with four exceptions:

- ROUND returns the multiple with the larger absolute value when the first argument is approximately halfway between the two nearest multiples of the second argument.
- ROUNDE returns an even multiple when the first argument is approximately halfway between the two nearest multiples of the second argument.
- ROUNDZ returns an even multiple when the first argument is exactly halfway between the two nearest multiples of the second argument.
- When the rounding unit is less than one and not the reciprocal of an integer, the result that is returned by ROUNDZ might not agree exactly with the result from decimal arithmetic. ROUND and ROUNDE perform extra computations, called fuzzing, to try to make the result agree with decimal arithmetic in the most common situations. ROUNDZ does not fuzz the result.

**Example**

The following example compares the results that are returned by the ROUNDE function with the results that are returned by the ROUND function.

```sas
data results;
  do x=0 to 4 by .25;
    Rounde=rounde(x);
    Round=round(x);
  end;
run;
```
Round=round(x);
output;
end;
run;
proc print data=results noobs;
run;

Figure 2.33  Output from the ROUND and ROUND Functions

<table>
<thead>
<tr>
<th>x</th>
<th>Rounde</th>
<th>Round</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.25</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.50</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0.75</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1.00</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1.25</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1.50</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1.75</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2.00</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2.25</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2.50</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2.75</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3.00</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3.25</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3.50</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>3.75</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>4.00</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

See Also

Functions:
ROUNDZ Function

Rounds the first argument to the nearest multiple of the second argument, using zero fuzzing.

**Category:** Truncation

**Syntax**

\[ \text{ROUNDZ}(\text{argument} <, \text{rounding-unit}>) \]

**Required Argument**

*argument*

is a numeric constant, variable, or expression to be rounded.

**Optional Argument**

*rounding-unit*

is a positive, numeric constant, variable, or expression that specifies the rounding unit.

**Details**

The ROUNDZ function rounds the first argument to the nearest multiple of the second argument. If you omit the second argument, ROUNDZ uses a default value of 1 for *rounding-unit*.

**Comparisons**

The ROUND, ROUNDE, and ROUNDZ functions are similar with four exceptions:

- ROUND returns the multiple with the larger absolute value when the first argument is approximately halfway between the two nearest multiples of the second argument.
- ROUNDE returns an even multiple when the first argument is approximately halfway between the two nearest multiples of the second argument.
- ROUNDZ returns an even multiple when the first argument is exactly halfway between the two nearest multiples of the second argument.
- When the rounding unit is less than one and not the reciprocal of an integer, the result that is returned by ROUNDZ might not agree exactly with the result from decimal arithmetic. ROUND and ROUNDE perform extra computations, called...
fuzzing, to try to make the result agree with decimal arithmetic in the most common situations. ROUNDZ does not fuzz the result.

Examples

Example 1: Comparing Results from the ROUNDZ and ROUND Functions

The following example compares the results that are returned by the ROUNDZ and the ROUND function.

data test;
  do i=10 to 17;
    Value=3.5 - 10**(-i);
    Roundz=roundz(value);
    Round=round(value);
    output;
  end;
  do i=16 to 12 by -1;
    value=3.5 + 10**(-i);
    roundz=roundz(value);
    round=round(value);
    output;
  end;
run;
proc print data=test noobs;
  format value 19.16;
run;
Example 2: Output from the ROUNDZ Function

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>var1=223.456; x=roundz(var1, 1); put x * 9.5;</td>
<td>223.00000</td>
</tr>
<tr>
<td>var2=223.456; x=roundz(var2, .01); put x * 9.5;</td>
<td>223.46000</td>
</tr>
<tr>
<td>x=roundz(223.456, 100); put x * 9.5;</td>
<td>200.00000</td>
</tr>
<tr>
<td>x=roundz(223.456); put x * 9.5;</td>
<td>223.00000</td>
</tr>
<tr>
<td>x=roundz(223.456, .3); put x * 9.5;</td>
<td>223.50000</td>
</tr>
</tbody>
</table>
### SAVINGS Function

Returns the balance of a periodic savings by using variable interest rates.

**Category:** Financial

### Syntax

```
SAVINGS(base-date, initial-deposit-date, deposit-amount, deposit-number,
         deposit-interval, compounding-interval, date-1, rate-2 <, date-2, rate-2, …>)
```

### Required Arguments

- **base-date**
  - is a SAS date. The value that is returned is the balance of the savings at base-date.

- **initial-deposit-date**
  - is a SAS date. Initial-deposit-date is the date of the first deposit. Subsequent deposits are at the beginning of subsequent deposit intervals.

- **deposit-amount**
  - is numeric. All deposits are assumed constant. deposit-amount is the value of each deposit.

- **deposit-number**
  - is a positive integer. Deposit-number is the number of deposits.

- **deposit-interval**
  - is a SAS interval. Deposit-interval is the frequency at which deposits are made.

- **compounding-interval**
  - is a SAS interval. Compounding-interval is the compounding interval.

- **date**
  - is a SAS date. Each date is paired with a rate. date is the time at which rate takes effect.

- **rate**
  - is a numeric percentage. Each rate is paired with a date. rate is the interest rate that starts on date.

### Details

The following details apply to the SAVINGS function:

- The values for rates must be between –99 and 120.
- deposit-interval cannot be 'CONTINUOUS'.
- The list of date-rate pairs does not need to be in chronological order.

---

**See Also**

**Functions:**

- “ROUND Function” on page 573
- “ROUNDE Function” on page 580
• When multiple rate changes occur on a single date, the SAVINGS function applies only the final rate that is listed for that date.

• Simple interest is applied for partial periods.

• There must be a valid date-rate pair whose date is at or prior to both the initial-deposit-date and the base-date.

Example

• If you deposit $300 monthly for two years into an account that compounds quarterly at an annual rate of 4%, the balance of the account after five years can be expressed as follows:

```sas
amount_base1=SAVINGS("01jan2005"d, "01jan2000"d, 300, 24,
   "MONTH", "QUARTER", "01jan2000"d, 4.00);
```

• If the interest rate increases by a quarter-point each year, then the balance of the account could be expressed as follows:

```sas
amount_base2=SAVINGS("01jan2005"d, "01jan2000"d, 300, 24,
   "MONTH", "QUARTER", "01jan2000"d, 4.00,
   "01jan2001"d, 4.25, "01jan2002"d, 4.50,
   "01jan2003"d, 4.75, "01jan2004"d, 5.00);
```

• To determine the balance after one year of deposits, the following statement sets `amount_base3` to the desired balance:

```sas
amount_base3=SAVINGS("01jan2001"d, "01jan2000"d, 300, 24,
   "MONTH", "QUARTER", "01jan2000"d, 4);
```

The SAVINGS function ignores deposits after the base date, so the deposits after the reference date do not affect the value that is returned.

---

### SCAN Function

Returns the \( n \)th word from a character string.

**Category:** Character

**Restriction:** This function is assigned an I18N Level 0 status, and is designed for SBCS data. Do not use this function to process DBCS or MBCS data. For more information, see Internationalization Compatibility.

**Tip:** The DBCS equivalent function is KSCAN.

**Syntax**

```sas
SCAN(string, count <, character-list <, modifier>>)
```

**Required Arguments**

- `string` specifies a character constant, variable, or expression.

- `count` is a nonzero numeric constant, variable, or expression that has an integer value. The integer value specifies the number of the word in the character string that you want
SCAN to select. For example, a value of 1 indicates the first word, a value of 2 indicates the second word, and so on. The following rules apply:

- If \textit{count} is positive, SCAN counts words from left to right in the character string.
- If \textit{count} is negative, SCAN counts words from right to left in the character string.

**Optional Arguments**

\textit{character-list} specifies an optional character expression that initializes a list of characters. This list determines which characters are used as the delimiters that separate words. The following rules apply:

- By default, all characters in \textit{character-list} are used as delimiters.
- If you specify the K modifier in the \textit{modifier} argument, then all characters that are not in \textit{character-list} are used as delimiters.

Tip: You can add more characters to \textit{character-list} by using other modifiers.

\textit{modifier} specifies a character constant, variable, or expression in which each non-blank character modifies the action of the SCAN function. Blanks are ignored. Use the following characters as modifiers:

- \textit{a or A} adds alphabetic characters to the list of characters.
- \textit{b or B} scans backward from right to left instead of from left to right, regardless of the sign of the \textit{count} argument.
- \textit{c or C} adds control characters to the list of characters.
- \textit{d or D} adds digits to the list of characters.
- \textit{f or F} adds an underscore and English letters (that is, valid first characters in a SAS variable name by using VALIDVARNAME=V7) to the list of characters.
- \textit{g or G} adds graphic characters to the list of characters. Graphic characters are characters that, when printed, produce an image on paper.
- \textit{h or H} adds a horizontal tab to the list of characters.
- \textit{i or I} ignores the case of the characters.
- \textit{k or K} causes all characters that are not in the list of characters to be treated as delimiters. That is, if K is specified, then characters that are in the list of characters are kept in the returned value rather than being omitted because they are delimiters. If K is not specified, then all characters that are in the list of characters are treated as delimiters.
- \textit{l or L} adds lowercase letters to the list of characters.
- \textit{m or M} specifies that multiple consecutive delimiters, and delimiters at the beginning or end of the \textit{string} argument, refer to words that have a length of zero. If the M modifier is not specified, then multiple consecutive delimiters are treated as one delimiter, and delimiters at the beginning or end of the \textit{string} argument are ignored.
- \textit{n or N} adds digits, an underscore, and English letters (that is, the characters that can appear in a SAS variable name by using VALIDVARNAME=V7) to the list of characters.
o or O processes the charlist and modifier arguments only once, rather than every time the SCAN function is called. Using the O modifier in the DATA step (excluding WHERE clauses), or in the SQL procedure can make SCAN run faster when you call it in a loop where the character-list and modifier arguments do not change. The O modifier applies separately to each instance of the SCAN function in your SAS code, and does not cause all instances of the SCAN function to use the same delimiters and modifiers.

p or P adds punctuation marks to the list of characters.

q or Q ignores delimiters that are inside substrings that are enclosed in quotation marks. If the value of the string argument contains unmatched quotation marks, then scanning from left to right produces different words than scanning from right to left.

r or R removes leading and trailing blanks from the word that SCAN returns. If you specify the Q and R modifiers, the SCAN function first removes leading and trailing blanks from the word. Then, if the word begins with a quotation mark, SCAN also removes one layer of quotation marks from the word.

s or S adds space characters to the list of characters (blank, horizontal tab, vertical tab, carriage return, line feed, and form feed).

t or T trims trailing blanks from the string and charlist arguments. If you want to remove trailing blanks from only one character argument instead of both character arguments, use the TRIM function instead of the SCAN function with the T modifier.

u or U adds uppercase letters to the list of characters.

w or W adds printable (writable) characters to the list of characters.

x or X adds hexadecimal characters to the list of characters.

Tip If the modifier argument is a character constant, enclose the argument in quotation marks. Specify multiple modifiers in a single set of quotation marks. A modifier argument can also be expressed as a character variable or expression.

Details

Definition of “Delimiter” and “Word”

A delimiter is any of several characters that are used to separate words. You can specify the delimiters in the charlist and modifier arguments.

If you specify the Q modifier, delimiters inside substrings that are enclosed in quotation marks are ignored.

In the SCAN function, “word” refers to a substring that has all of these characteristics:

- is bounded on the left by a delimiter or the beginning of the string
- is bounded on the right by a delimiter or the end of the string
- contains no delimiters

A word can have a length of zero if there are delimiters at the beginning or end of the string, or if the string contains two or more consecutive delimiters. However, the SCAN function ignores words that have a length of zero unless you specify the M modifier.
Note: The definition of “word” is the same in the SCAN and COUNTW functions.

Using Default Delimiters in ASCII and EBCDIC Environments
If you use the SCAN function with only two arguments, then the default delimiters depend on whether your computer uses ASCII or EBCDIC characters.

• If your computer uses ASCII characters, the default delimiters are as follows:
  \[\text{blank}!\ %\ &\ ()*+,.-/;<\wedge|\]
  In ASCII environments that do not contain the \^[\wedge] character, the SCAN function uses the \[\sim\] character instead.

• If your computer uses EBCDIC characters, then the default delimiters are as follows:
  \[\text{blank}!\ %\ &\ ()*+,.-/;<\neg|\eshape\]

If you use the modifier argument without specifying any characters as delimiters, then the only delimiters that are used are delimiters that are defined by the modifier argument. In this case, the lists of default delimiters for ASCII and EBCDIC environments are not used. In other words, modifiers add to the list of delimiters that are explicitly specified by the charlist argument. Modifiers do not add to the list of default modifiers.

The Length of the Result
In a DATA step, most variables have a fixed length. If the word returned by the SCAN function is assigned to a variable that has a fixed length greater than the length of the returned word, then the value of that variable is padded with blanks. Macro variables have varying lengths and are not padded with blanks.

The maximum length of the word that is returned by the SCAN function depends on the environment from which it is called:

• In a DATA step, if the SCAN function returns a value to a variable that has not yet been given a length, that variable is given the length of the first argument. This behavior is different from the behavior in previous releases of SAS. In previous releases code that created a variable with a length of 200 might have produced a variable with a length that was greater than expected. If you need the SCAN function to assign to a variable a value that is different from the length of the first argument, use a LENGTH statement for that variable before the statement that uses the SCAN function.

  If you use the SCAN function in an expression that contains operators or other functions, a word that is returned by the SCAN function can have a length of up to 32,767 characters, except in a WHERE clause. In that case, the maximum length is 200 characters.

• In the SQL procedure, or in a WHERE clause in any procedure, the maximum length of a word that is returned by the SCAN function is 200 characters.

• In the macro processor, the maximum length of a word that is returned by the SCAN function is 65,534 characters.

The minimum length of the word that is returned by the SCAN function depends on whether the M modifier is specified. See “Using the SCAN Function with the M Modifier” on page 590. See also “Using the SCAN Function without the M Modifier” on page 590.
Using the SCAN Function with the M Modifier
If you specify the M modifier, the number of words in a string is defined as one plus the number of delimiters in the string. However, if you specify the Q modifier, delimiters that are inside quotation marks are ignored.

If you specify the M modifier, the SCAN function returns a word with a length of zero if one of these conditions is true:
- The string begins with a delimiter and you request the first word.
- The string ends with a delimiter and you request the last word.
- The string contains two consecutive delimiters and you request the word that is between the two delimiters.

Using the SCAN Function without the M Modifier
If you do not specify the M modifier, the number of words in a string is defined as the number of maximal substrings of consecutive non-delimiters. However, if you specify the Q modifier, delimiters that are inside quotation marks are ignored.

If you do not specify the M modifier, the SCAN function acts in these ways:
- ignores delimiters at the beginning or end of the string
- treats two or more consecutive delimiters as if they were a single delimiter

If the string contains no characters other than delimiters, or if you specify a count that is greater in absolute value than the number of words in the string, then the SCAN function returns one of the following items:
- a single blank when you call the SCAN function from a DATA step
- a string with a length of zero when you call the SCAN function from the macro processor

Using Null Arguments
This example scans a string for the first and last words:
- A negative count instructs the SCAN function to scan from right to left.
- Leading and trailing delimiters are ignored because the M modifier is not used.
- In the last observation, all characters in the string are delimiters.

Some language elements are supported in SAS but not in CAS. An example is the DATALINES statement. Some examples in this document use DATALINES in SAS to create a data set using the CAS engine. For more information, see “Functions in SAS and CAS” on page 54.

Examples

Example 1: Using the SCAN Function in SAS and CAS
This example uses the SCAN function to scan first and last names and output the names. This code processes the data in SAS. The DATALINES statement is supported in SAS but not in CAS.
- A negative count instructs the SCAN function to scan from right to left.
- Leading and trailing delimiters are ignored because the M modifier is not used.
- In the last observation, all characters in the string are delimiters.
data firstlast;
  input string $60.;
  First_Word=scan(string, 1);
  Last_Word=scan(string, -1);
datalines;
Jack and Jill
& Bob & Carol & Ted & Alice & Leonardo
! $ % & ( ) * + , - . / ;
;i;;;
proc print data=firstlast;
run;

Since DATALINES is not supported in CAS, you must create a data set in SAS that uses a CAS engine. Then you can run the function in CAS. Here is an example.

Create a data set, casuser.firstlast, in SAS using the CAS engine.

data casuser.firstlast;
  input String $60.;
datalines4;
Jack and Jill
& Bob & Carol & Ted & Alice & Leonardo
! $ % & ( ) * + , - . / ;
;i;;;
run;

Run the code in CAS using the functionality of the SCAN function with the data set, firstlast.

data casuser.firstlastn;
  set casuser.firstlast;
  First_Word=scan(string, 1);
  Last_Word=scan(string, -1);
run;
proc print data=casuser.firstlastn;
run;

The SAS System

<table>
<thead>
<tr>
<th>Obs</th>
<th>String</th>
<th>First_Word</th>
<th>Last_Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Jack and Jill</td>
<td>Jack</td>
<td>Jill</td>
</tr>
<tr>
<td>2</td>
<td>&amp; Bob &amp; Carol &amp; Ted &amp; Alice &amp; Leonardo</td>
<td>Bob</td>
<td>Alice</td>
</tr>
<tr>
<td>3</td>
<td>Leonardo</td>
<td>Leonardo</td>
<td>Leonardo</td>
</tr>
<tr>
<td>4</td>
<td>! $ % &amp; ( ) * + , - . / ;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Example 2: Finding the First and Last Words in a String**

This example scans a string for the first and last words:
**Example 3: Finding All Words in a String without Using the M Modifier**

This example scans a string from left to right until the word that is returned is blank. Because the M modifier is not used, the SCAN function does not return any words that have a length of zero. Because blanks are included among the default delimiters, the SCAN function returns a blank word only when the count exceeds the number of words in the string. Therefore, the loop can be stopped when SCAN returns a blank word.

```sas
data all;
  length word $20;
  drop string;
  string=' The quick brown fox jumps over the lazy dog. ';  
do until(word=' ');
    count+1;
    word=scan(string, count);
    output;
  end;
run;
proc print data=all noobs;
```

<table>
<thead>
<tr>
<th>Obs</th>
<th>String</th>
<th>First_Word</th>
<th>Last_Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Jack and Jill</td>
<td>Jack</td>
<td>Jill</td>
</tr>
<tr>
<td>2</td>
<td>&amp; Bob &amp; Carol &amp; Ted &amp; Alice &amp;</td>
<td>Bob</td>
<td>Alice</td>
</tr>
<tr>
<td>3</td>
<td>Leonardo</td>
<td>Leonardo</td>
<td>Leonardo</td>
</tr>
<tr>
<td>4</td>
<td>!$%() +,-./</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 2.35  Output from Finding the First and Last Words in a String**
Example 4: Finding All Words in a String By Using the M and O Modifiers

This example shows the results of using the M modifier with a comma as a delimiter. With the M modifier, leading, trailing, and multiple consecutive delimiters cause the SCAN function to return words that have a length of zero. Therefore, do not end the loop by testing for a blank word. Instead, use the COUNTW function with the same modifiers and delimiters to count the words in the string.

The O modifier is used for efficiency because the delimiters and modifiers are the same in every call to the SCAN and COUNTW functions.

data comma;
  keep count word;
  length word $30;
  string=',leading, trailing,and multiple,,delimiters,,';
  delim=',';
  modif='mo';
  nwords=countw(string, delim, modif);
  do count=1 to nwords;
    word=scan(string, count, delim, modif);
    output;
  end;
run;
proc print data=comma noobs;
run;
Example 5: Using Comma-Separated Values, Substrings in Quotation Marks, and the O and R Modifiers

This example uses the SCAN function with the O modifier and a comma as a delimiter, both with and without the R modifier.

The O modifier is used for efficiency because in each call of the SCAN or COUNTW function, the delimiters and modifiers do not change. The O modifier applies separately to each of the two instances of the SCAN function:

- The first instance of the SCAN function uses the same delimiters and modifiers every time SCAN is called.
- The second instance of the SCAN function uses the same delimiters and modifiers every time SCAN is called.
- The first instance of the SCAN function does not use the same modifiers as the second instance, but this fact has no bearing on the use of the O modifier.

data test;
  keep count word word_r;
  length word word_r $30;
  string='He said, "She said, "No!"", not "Yes!"';
  delim=',';
  modif='oq';
  nwords=countw(string, delim, modif);
  do count=1 to nwords;
    word=scan(string, count, delim, modif);
    word_r=scan(string, count, delim, modif||'r');
    output;
  end;
run;
proc print data=test noobs;
run;
Example 6: Finding Substrings of Digits By Using the D and K Modifiers

This example finds substrings of digits. The character-list argument is null. Consequently, the list of characters is initially empty. The D modifier adds digits to the list of characters. The K modifier treats all characters that are not in the list as delimiters. Therefore, all characters except digits are delimiters.

```sas
data digits;
  keep count digits;
  length digits $20;
  string='Call (800) 555-1234 now!';
  do until(digits=' ');
    count+1;
    digits=scan(string, count, , 'dko');
    output;
  end;
run;
proc print data=digits noobs;
run;
```

See Also

Functions:

- “COUNTW Function” on page 209
SDF Function

Returns a survival function.

**Category:** Probability

**See:** “CDF Function” on page 153

**Syntax**

\[
\text{SDF}(\text{distribution, quantile, parameter-1, ..., parameter-k})
\]

**Required Arguments**

\textit{distribution} is a character string that identifies the distribution. Valid distributions are as follows:

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bernoulli</td>
<td>BERNOULLI</td>
</tr>
<tr>
<td>Beta</td>
<td>BETA</td>
</tr>
<tr>
<td>Binomial</td>
<td>BINOMIAL</td>
</tr>
<tr>
<td>Cauchy</td>
<td>CAUCHY</td>
</tr>
<tr>
<td>Chi-Square</td>
<td>CHISQUARE</td>
</tr>
<tr>
<td>Conway-Maxwell-Poisson</td>
<td>CONMAXPOI</td>
</tr>
<tr>
<td>Exponential</td>
<td>EXPONENTIAL</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>Gamma</td>
<td>GAMMA</td>
</tr>
<tr>
<td>Generalized Poisson</td>
<td>GENPOISSON</td>
</tr>
<tr>
<td>Geometric</td>
<td>GEOMETRIC</td>
</tr>
<tr>
<td>Hypergeometric</td>
<td>HYPERGEOMETRIC</td>
</tr>
<tr>
<td>Laplace</td>
<td>LAPLACE</td>
</tr>
</tbody>
</table>
### Distribution

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logistic</td>
<td>LOGISTIC</td>
</tr>
<tr>
<td>Lognormal</td>
<td>LOGNORMAL</td>
</tr>
<tr>
<td>Negative binomial</td>
<td>NEGBINOMIAL</td>
</tr>
<tr>
<td>Normal</td>
<td>NORMAL</td>
</tr>
<tr>
<td>Normal mixture</td>
<td>NORMALMIX</td>
</tr>
<tr>
<td>Pareto</td>
<td>PARETO</td>
</tr>
<tr>
<td>Poisson</td>
<td>POISSON</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>Tweedie</td>
<td>TWEEDIE</td>
</tr>
<tr>
<td>Uniform</td>
<td>UNIFORM</td>
</tr>
<tr>
<td>Wald (inverse Gaussian)</td>
<td>WALD</td>
</tr>
<tr>
<td>Weibull</td>
<td>WEIBULL</td>
</tr>
</tbody>
</table>

**Note**  
Except for T, F, and NORMALMIX, you can minimally identify any distribution by its first four characters.

*quantile*  
is a numeric constant, variable, or expression that specifies the value of a random variable.

*parameter-1, ..., parameter-k*  
are optional shape, location, or scale parameters appropriate for the specific distribution.

### Details

The SDF function computes the survival function (upper tail) from various continuous and discrete distributions. For more information, see “CDF Function” on page 153.

The SDF function for the Conway-Maxwell-Poisson distribution has the following form:

\[
\text{SDF('CONMAXPOI', } y, \lambda, \nu) \]

\( y \) is a nonnegative integer that represents counts data. \( \lambda \) is similar to the mean, as in the Poisson distribution. \( \nu \) is a dispersion parameter. The SDF function returns the probability that the counts value is greater than \( y \). For more information, see “Conway-Maxwell-Poisson” distribution in the PDF function on page 482.
### Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>y=sdf('BERN', 0, .25);</td>
<td>0.25</td>
</tr>
<tr>
<td>y=sdf('BETA', 0.2, 3, 4);</td>
<td>0.90112</td>
</tr>
<tr>
<td>y=sdf('BINOM', 4, .5, 10);</td>
<td>0.62305</td>
</tr>
<tr>
<td>y=sdf('CAUCHY', 2);</td>
<td>0.14758</td>
</tr>
<tr>
<td>y=sdf('CHISQ', 11.264, 11);</td>
<td>0.42142</td>
</tr>
<tr>
<td>y=sdf('CONMAXPOI', 12, 2.3, .4);</td>
<td>0.1970513877</td>
</tr>
<tr>
<td>y=sdf('EXPO', 1);</td>
<td>0.36788</td>
</tr>
<tr>
<td>y=sdf('F', 3.32, 2,3);</td>
<td>0.17361</td>
</tr>
<tr>
<td>y=sdf('GAMMA', 1, 3);</td>
<td>0.91970</td>
</tr>
<tr>
<td>y=sdf('GENPOISSON', .9, 1, .7);</td>
<td>0.6321205588</td>
</tr>
<tr>
<td>y=sdf('GEOMETRIC', 5, .3);</td>
<td>y=0.117649</td>
</tr>
<tr>
<td>y=sdf('HYPER', 2, 200, 50, 10);</td>
<td>0.47633</td>
</tr>
<tr>
<td>y=sdf('LAFLACE', 1);</td>
<td>0.18394</td>
</tr>
<tr>
<td>y=sdf('LOGISTIC', 1);</td>
<td>0.26894</td>
</tr>
<tr>
<td>y=sdf('LOGNORMAL', 1);</td>
<td>0.5</td>
</tr>
<tr>
<td>y=sdf('NEGB', 1, .5, 2);</td>
<td>0.5</td>
</tr>
<tr>
<td>y=sdf('NORMAL', 1.96);</td>
<td>0.025</td>
</tr>
<tr>
<td>y=sdf('NORMALMIX', 2.3, 3, .33, .33, .34, .5, 1.5, 2.5, .79, 1.6, 4.3);</td>
<td>0.2819</td>
</tr>
<tr>
<td>y=sdf('PARETO', 1, 1);</td>
<td>1</td>
</tr>
<tr>
<td>y=sdf('POISSON', 2, 1);</td>
<td>0.08030</td>
</tr>
<tr>
<td>y=sdf('T', .9, 5);</td>
<td>0.20469</td>
</tr>
<tr>
<td>y=sdf('TWEEDIE', .8, 5);</td>
<td>0.4082370836</td>
</tr>
<tr>
<td>y=sdf('UNIFORM', 0.25);</td>
<td>0.75</td>
</tr>
</tbody>
</table>
SAS Statement | Result
---|---
y=sdf('WALD', 1, 2); | 0.37230

y=sdf('WEIBULL', 1, 2); | 0.36788

See Also

Functions:

- “CDF Function” on page 153
- “LOGCDF Function” on page 416
- “LOGPDF Function” on page 418
- “LOGSDF Function” on page 420
- “PDF Function” on page 477
- “QUANTILE Function” on page 549
- “SQUANTILE Function” on page 1062

SEC Function

Returns the secant.

**Category:** Trigonometric

**Syntax**

SEC(argument)

**Required Argument**

*argument* specifies a numeric constant, variable, or expression and is expressed in radians.

**Restriction** *argument* cannot be an odd multiple of PI/2.

**Comparisons**

The SEC function is related to the COS function:

\[ \text{sec}(x) = 1/\cos(x) \]
Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=sec(0.5);</td>
<td>1.1394939273</td>
</tr>
<tr>
<td>x=sec(0);</td>
<td>1</td>
</tr>
<tr>
<td>x=sec(3.14159/3);</td>
<td>1.9999969359</td>
</tr>
</tbody>
</table>

See Also

Functions:
- “COS Function” on page 201
- “COT Function” on page 203
- “CSC Function” on page 212
- “SIN Function” on page 605
- “TAN Function” on page 624

SECOND Function

Returns the second from a SAS time or datetime value.

Category: Date and Time

Syntax

SECOND(time | datetime)

Required Arguments

time
is a numeric constant, variable, or expression with a value that represents a SAS time value.

datetime
is a numeric constant, variable, or expression with a value that represents a SAS datetime value.

Details

The SECOND function produces a numeric value that represents a specific second of the minute. The result can be any number that is >= 0 and < 60.

Example
The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>time='3:19:24' t; s=second(time); put s;</td>
<td>24</td>
</tr>
<tr>
<td>time='6:25:65' t; s=second(time); put s;</td>
<td>5</td>
</tr>
<tr>
<td>time='3:19:60' t; s=second(time); put s;</td>
<td>0</td>
</tr>
</tbody>
</table>

See Also

Functions:
- “HOUR Function” on page 325
- “MINUTE Function” on page 435

SIGN Function

Returns the sign of a value.

**Category:** Mathematical

**Syntax**

\[
\text{SIGN}(\text{argument})
\]

**Required Argument**

\(\text{argument}\)

specifies a numeric constant, variable, or expression.

**Details**

The SIGN function returns the following values:

-1 if \(\text{argument} < 0\)

0 if \(\text{argument} = 0\)

1 if \(\text{argument} > 0\).
Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=sign(-5);</td>
<td>-1</td>
</tr>
<tr>
<td>x=sign(5);</td>
<td>1</td>
</tr>
<tr>
<td>x=sign(0);</td>
<td>0</td>
</tr>
</tbody>
</table>

SHA256HEX Function

Returns the result of the message digest of a specified string and converts the string to hexadecimal representation.

**Category:** Character

**Note:** UTF-8 text is recommended for the SHA256HEX function arguments to ensure consistency across encodings. See “Example 2: Using the KCVT Function to Transcode to UTF-8” on page 603 for an example that uses the KCVT function to transcode data to UTF-8.

**Syntax**

SHA256HEX('string', string_indicator)

**Required Arguments**

'\textit{string}'

specifies a character constant, variable, or expression.

\textit{string\_indicator}

indicates whether the argument \textit{string} is regular characters or hexadecimal representation characters.

0 indicates that the expression in the argument \textit{string} is regular characters.

1 indicates that the expression in the argument \textit{string} is hexadecimal representation characters.

**Note:** There must be an even number of hexadecimal representation characters, and they must all be between 0–9, a–f, or A–F. Blanks in the hexadecimal representation string are ignored.

**Details**

**The Basics**

The SHA256HEX function converts a string, based on the SHA256 algorithm, to a 256-bit hash value. Then, the function converts the data to a hexadecimal representation format.
On an ASCII system, 'ABC' is '414243', and the digest is B5D4045C3F466FA91FE2CC6ABE79232A1A57CDF104F7A26E716E0A1E2789DF78.

**Using the SHA256HEX Function**

You can use the SHA256HEX function to track changes in your data sets. The SHA256HEX function can generate a digest of a set of column values in a table record. This digest could be treated as the signature of the record and be used to track changes that are made to the record. If the digest from the new record matches the existing digest of a table record, then the two records are the same. If the digest is different, then a column value in the record has changed. The new changed record could then be added to the table along with a new surrogate key because the record represents a change to an existing keyed value.

The SHA256HEX function can be useful when you are developing shell scripts or Perl programs for software installation, file comparison, and detection of file corruption and tampering.

**Comparisons**

The SHA256 function does not format its own output, so you must use the $BINARY or $HEX formats to view readable results. The SHA256HEX function formats its output, so you do not have to use the $BINARY or $HEX formats.

**Examples**

**Example 1: Generating Results with the SHA256HEX Function**

This example generates results that are returned by the SHA256HEX function.

```plaintext
data _null_;
  y=sha256hex('abc');
  z=sha256hex('access method');
  put y=;
  put z=;
run;
```

The following output is displayed for ASCII systems:

```
y=BA7816BF8F01CFEA414140DE5DA2223B00361A396177A9CB410FF61F20015AD
z=F2758E91725621F59F2F80D15DE8824560E8C471EB440A83BA6D1259B1605915
```

**Example 2: Using the KCVT Function to Transcode to UTF-8**

The following example demonstrates how to use the KCVT function to transcode to UTF-8. The second digest matches between running UTF-8, which is the first section, and EUC-CN, which is the second section.
SHA256HMACHEX Function

Returns the result of the message digest of a specified string by using the Hash-based Message Authentication (HMAC) algorithm.

Category: Character

Notes: The SHA256HMACHEX function verifies the data integrity and authentication of a message.

For more information, see Hash-based message authentication code (HMAC). UTF-8 text is recommended for the SHA256HMACHEX function arguments to ensure consistency across encodings. See “Example 2: Using the KCVT Function to Transcode to UTF-8” on page 603 for an example that uses the KCVT function to transcode data to UTF-8.

Syntax

SHA256HMACHEX('key', 'message' <string indicator>);

Required Arguments

key

specifies a character constant, variable, or expression.

message

specifies a secret key padded to the right with extra zeros to the input block size of the hash function.

See Also

- “SHA256 Function” on page 1045
- “SHA256HMACHEX Function” on page 604
**Optional Argument**

`string_indicator` indicates whether the key and message are provided in hexadecimal representation.

0  the arguments `key` and `message` are not represented in hexadecimal representation.
1  the argument `message` is represented in hexadecimal representation.
2  the argument `key` is represented in hexadecimal representation.
3  the arguments `key` and `message` are represented in hexadecimal representation.

*Note:* This argument is useful when the SHA256HMACHEX function is being called repeatedly and the result of a previous call is used as the key in a subsequent call. The following code demonstrates this functionality:

```plaintext
length digest $64;
digest = sha256machex('mykey','mymessage',0);
digest = sha256machex(digest,'my new message',2);
```

**Details**

The SHA256HMACHEX function converts a string, based on the SHA256 algorithm, to a 256-bit hash value.

For more information, see Hash-based message authentication code (HMAC).

**Example: Generating Results with the SHA256HMACHEX Function**

This example generates results that are returned by the SHA256HMACHEX function.

```plaintext
data _null_;  
digest = SHA256HMACHEX('key', 'The quick brown fox jumps over the lazy dog',0);  
if digest= upcase('f7bc83f430538424b13298e6aa6fb143ef4d59a14946175997479d2c2d1a3cd8')  
then put 'matched';  
else put 'not matched';  
run;
```

The following results are generated from the code:

```
matched
```

**See Also**

“SHA256HEX Function” on page 602

---

**SIN Function**

Returns the sine.

**Category:** Trigonometric
Syntax

\[
\text{SIN}(\text{argument})
\]

**Required Argument**

\text{argument} specifies a numeric constant, variable, or expression and is expressed in radians. If the magnitude of \text{argument} is so great that \text{mod(argument, pi)} is accurate to less than about three decimal places, SIN returns a missing value.

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=sin(0.5);</td>
<td>0.4794255386</td>
</tr>
<tr>
<td>x=sin(0);</td>
<td>0</td>
</tr>
<tr>
<td>x=sin(3.14159/4);</td>
<td>.7071063121</td>
</tr>
</tbody>
</table>

---

**SINH Function**

Returns the hyperbolic sine.

**Category:** Hyperbolic

Syntax

\[
\text{SINH}(\text{argument})
\]

**Required Argument**

\text{argument} specifies a numeric constant, variable, or expression.

**Details**

The SINH function returns the hyperbolic sine of the argument, which is given by

\[
\frac{e^{\text{argument}} - e^{-\text{argument}}}{2}
\]
Example
The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=sinh(0);</td>
<td>0</td>
</tr>
<tr>
<td>x=sinh(1);</td>
<td>1.1752011936</td>
</tr>
<tr>
<td>x=sinh(-1.0);</td>
<td>-1.175201194</td>
</tr>
</tbody>
</table>

SKEWNESS Function
Returns the skewness of the nonmissing arguments.

**Category:** Descriptive Statistics

**Syntax**

SKEWNESS(argument-1, argument-2, argument-3 <,…argument-n>)

**Required Argument**

argument

specifies a numeric constant, variable, or expression.

**Details**

At least three nonmissing arguments are required. Otherwise, the function returns a missing value. If all nonmissing arguments have equal values, the skewness is mathematically undefined. The SKEWNESS function returns a missing value and sets _ERROR_ equal to 1.

The argument list can consist of a variable list, which is preceded by OF.

Example
The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1=skewness(0, 1, 1);</td>
<td>-1.732050808</td>
</tr>
<tr>
<td>x2=skewness(2, 4, 6, 3, 1);</td>
<td>0.5901286564</td>
</tr>
<tr>
<td>x3=skewness(2, 0, 0);</td>
<td>1.7320508076</td>
</tr>
<tr>
<td>x4=skewness(of x1-x3);</td>
<td>-0.953097714</td>
</tr>
</tbody>
</table>
SLEEP Function
Suspends the execution of a program that invokes this function for a period of time.

Syntax
SLEEP(n <, unit>)

Required Argument
n
is a numeric constant, variable, or expression that specifies the number of seconds for which you want to suspend execution of a program. Negative or missing values for n are invalid.

Range  \( n \geq 0 \)

Tip If you use a fraction for the n argument, the unit argument is required if you want to suspend execution for a fraction of a second. For example, SLEEP(.25); does not suspend execution. SLEEP(1.25); suspends execution for 1 second. SLEEP(1.25, 1); suspends execution for 1.25 seconds. SLEEP(.25,1) suspends execution for .25 seconds.

Optional Argument
unit
specifies the unit of time in seconds as an exponent of 10, which is applied to n. For example, 1 corresponds to 1 second; .001 corresponds to 1 millisecond; and 5 corresponds to 5 seconds.

Default 1

Details
The SLEEP function suspends the execution of a program that invokes this function for a period of time that you specify. The program can be a DATA step, macro, IML, SCL, or any program that can invoke a function.

The SLEEP function only suspends the execution of a program and does not return a value.

When the SLEEP function uses the default unit value, a pop-up window appears that indicates how long SAS is going to sleep.

When you submit a program that calls the SLEEP function, the SLEEP window appears, telling you when SAS is going to wake up. You can inhibit the SLEEP window by starting SAS with the NOSLEEPWINDOW system option. Your SAS session remains inactive until the sleep period is over. To cancel the call to the SLEEP function, use the CTRL+BREAK attention sequence.

You should use a null DATA step to call the SLEEP function; follow this DATA step with the rest of the SAS program. Using the SLEEP function in this manner enables you
to use the CTRL+BREAK attention sequence to interrupt the SLEEP function and to continue with the execution of the rest of your SAS program.

Examples

**Example 1: Suspending Execution for a Specified Period of Time**
The following example tells SAS to delay the execution of the DATA step PAYROLL for 20 seconds:

```sas
data payroll;
   time_slept=sleep(20,1);
   ...more SAS statements...
run;
```

**Example 2: Suspending Execution Based on a Calculation of Sleep Time**
The following example tells SAS to suspend the execution of the DATA step BUDGET until March 1, 2013, at 3:00 AM. SAS calculates the length of the suspension based on the target date and the date and time that the DATA step begins to execute.

```sas
data budget;
   sleeptime='01mar2013:03:00'dt-'01mar2013:2:59:30'dt;
   time_calc=sleep(sleeptime,1);
   put 'Calculation of sleep time:';
   put sleeptime='seconds';
run;
```

SAS writes the following results to the log:

```
Calculation of sleep time:
sleeptime=30 seconds
```

See Also

CALL Routines:
• “CALL SLEEP Routine” on page 138

---

**SMALLEST Function**

Returns the \( k \)th smallest nonmissing value.

- **Category:** Descriptive Statistics

**Syntax**

\[ \text{SMALLEST}(k, \text{value-1} <, \text{value-2}<) \]

**Required Arguments**

- \( k \)
  
is a numeric constant, variable, or expression that specifies which value to return.
value
specifies a numeric constant, variable, or expression.

Details
If $k$ is missing, less than zero, or greater than the number of values, the result is a missing value and _ERROR_ is set to 1. Otherwise, if $k$ is greater than the number of nonmissing values, the result is a missing value but _ERROR_ is not set to 1.

Comparisons
The SMALLEST function differs from the ORDINAL function in that the SMALLEST function ignores missing values, but the ORDINAL function counts missing values.

Example
This example compares the values that are returned by the SMALLEST function with values that are returned by the ORDINAL function.

```sas
data comparison;
  label smallest_num='SMALLEST Function' ordinal_num='ORDINAL Function';
do k = 1 to 4;
  smallest_num=smallest(k, 456, 789, .Q, 123);
  ordinal_num=ordinal (k, 456, 789, .Q, 123);
  output;
end;
run;
proc print data=comparison label noobs;
  var k smallest_num ordinal_num;
  title 'Results From the SMALLEST and the ORDINAL Functions';
run;
```

Figure 2.40  Comparison of Values: The SMALLEST and the ORDINAL Functions

Results From the SMALLEST and the ORDINAL Functions

<table>
<thead>
<tr>
<th>k</th>
<th>SMALLEST Function</th>
<th>ORDINAL Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>123</td>
<td>.Q</td>
</tr>
<tr>
<td>2</td>
<td>456</td>
<td>123</td>
</tr>
<tr>
<td>3</td>
<td>789</td>
<td>456</td>
</tr>
<tr>
<td>4</td>
<td>.</td>
<td>789</td>
</tr>
</tbody>
</table>

See Also

Functions:
- “LARGEST Function” on page 400
SQRT Function

Returns the square root of a value.

Category: Mathematical

Syntax

\[
\text{SQRT}(\text{argument})
\]

Required Argument

\text{argument}

specifies a numeric constant, variable, or expression. \text{Argument} must be nonnegative.

Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{x=sqrt}\left(36\right);</td>
<td>6</td>
</tr>
<tr>
<td>\text{x=sqrt}\left(25\right);</td>
<td>5</td>
</tr>
<tr>
<td>\text{x=sqrt}\left(4.4\right);</td>
<td>2.0976176963</td>
</tr>
</tbody>
</table>

STD Function

Returns the standard deviation of the nonmissing arguments.

Category: Descriptive Statistics

Syntax

\[
\text{STD}(\text{argument-1}, \text{argument-2 <,... argument-n>})
\]

Required Argument

\text{argument}

specifies a numeric constant, variable, or expression. At least two nonmissing arguments are required. Otherwise, the function returns a missing value. The argument list can consist of a variable list, which is preceded by \text{OF}. 
Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1 = std(2, 6);</td>
<td>2.8284271247</td>
</tr>
<tr>
<td>x2 = std(2, 6, .);</td>
<td>2.8284271427</td>
</tr>
<tr>
<td>x3 = std(2, 4, 6, 3, 1);</td>
<td>1.923584062</td>
</tr>
<tr>
<td>x4 = std(of x1-x3);</td>
<td>0.5224377453</td>
</tr>
</tbody>
</table>

STDERR Function

Returns the standard error of the mean of the nonmissing arguments.

Category: Descriptive Statistics

Syntax

```
STDERR(argument-1, argument-2 <, …argument-n>)
```

Required Argument

`argument`

specifies a numeric constant, variable, or expression. At least two nonmissing arguments are required. Otherwise, the function returns a missing value. The argument list can consist of a variable list, which is preceded by OF.

Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1 = stderr(2, 6);</td>
<td>2</td>
</tr>
<tr>
<td>x2 = stderr(2, 6, .);</td>
<td>2</td>
</tr>
<tr>
<td>x3 = stderr(2, 4, 6, 3, 1);</td>
<td>0.8602325267</td>
</tr>
<tr>
<td>x4 = stderr(of x1-x3);</td>
<td>0.3799224911</td>
</tr>
</tbody>
</table>
STRIP Function
Returns a character string with all leading and trailing blanks removed.

**Category:** Character  

**Restriction:** This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility.

### Syntax

\[ \text{STRIP}(\text{string}) \]

### Required Argument

**string**  
A character constant, variable, or expression.

### Details

**Length of Returned Variable**
In a DATA step, if the STRIP function returns a value to a variable that has not previously been assigned a length, then that variable is given the length of the argument.

**The Basics**
The STRIP function returns the argument with all leading and trailing blanks removed. If the argument is blank, STRIP returns a string with a length of zero.

Assigning the results of STRIP to a variable does not affect the length of the receiving variable. If the value that is trimmed is shorter than the length of the receiving variable, SAS pads the value with new trailing blanks.

**Note:** The STRIP function is useful for concatenation because the concatenation operator does not remove leading or trailing blanks.

### Comparisons
The following list compares the STRIP function with the TRIM and TRIMN functions:

- For strings that are blank, the STRIP and TRIMN functions return a string with a length of zero, whereas the TRIM function returns a single blank.
- For strings that lack leading blanks, the STRIP and TRIMN functions return the same value.
- For strings that lack leading blanks but have at least one non-blank character, the STRIP and TRIM functions return the same value.

**Note:** \( \text{STRIP}(\text{string}) \) returns the same result as \( \text{TRIMN}(\text{LEFT}(\text{string})) \), but the STRIP function runs faster.
Examples

Example 1: Deleting the Leading and Trailing Blanks Using the STRIP Function in SAS

The following example shows the results of using the STRIP function to delete leading and trailing blanks.

```sas
data lengthn;
  input string $char8.;
  original='*' || string || '*';
  stripped='*' || strip(string) || '*';
datalines;
abcd
abcd
abcd
abcdefgh
x y z
;
proc print data=lengthn;
run;
```

**Figure 2.41** Output from the STRIP Function Running in SAS

---

Example 2: Deleting the Leading and Trailing Blanks Using the STRIP Function in CAS

This example uses the DATA step to delete leading and trailing blanks. The first section of code in this example runs in SAS and creates the data set, lengthn, with the CAS engine. The second section of code runs in CAS, accesses the data in lengthn that is in CAS, and then creates the lengthnn data set.

```sas
data casuser.lengthn;
  input string $char8.;
datalines;
abcd
abcd
abcd
abcdefgh
x y z
;
```

---

### The SAS System

<table>
<thead>
<tr>
<th>Obs</th>
<th>string</th>
<th>original</th>
<th>stripped</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>abcd</td>
<td><em>abcd</em></td>
<td><em>abcd</em></td>
</tr>
<tr>
<td>2</td>
<td>abcd</td>
<td>* abcd*</td>
<td><em>abcd</em></td>
</tr>
<tr>
<td>3</td>
<td>abcd</td>
<td>* abcd*</td>
<td><em>abcd</em></td>
</tr>
<tr>
<td>4</td>
<td>abcdefgh</td>
<td><em>abcdefgh</em></td>
<td><em>abcdefgh</em></td>
</tr>
<tr>
<td>5</td>
<td>xyz</td>
<td>* xyz*</td>
<td><em>xyz</em></td>
</tr>
</tbody>
</table>

---

Example 2: Deleting the Leading and Trailing Blanks Using the STRIP Function in CAS
data casuser.lengthnn;
set casuser.lengthn;
orignal='*' || string || '***';
stripped='***' || strip(string) || '***';
run;

proc print data=casuser.lengthnn;
run;

Output 2.13  Output from the STRIP Function Running in CAS

<table>
<thead>
<tr>
<th>Obs</th>
<th>string</th>
<th>original</th>
<th>stripped</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>abcd</td>
<td><em>abcd</em></td>
<td><em>abcd</em></td>
</tr>
<tr>
<td>2</td>
<td>abcd</td>
<td>* abcd *</td>
<td><em>abcd</em></td>
</tr>
<tr>
<td>3</td>
<td>abcd</td>
<td>* abcd*</td>
<td><em>abcd</em></td>
</tr>
<tr>
<td>4</td>
<td>abcdefg</td>
<td><em>abcede fg</em></td>
<td><em>abcde fg</em></td>
</tr>
<tr>
<td>5</td>
<td>xyz</td>
<td><em>xyz</em></td>
<td><em>xyz</em></td>
</tr>
</tbody>
</table>

See Also

Functions:
- “CAT Function” on page 143
- “CATS Function” on page 145
- “CATT Function” on page 147
- “CATX Function” on page 149
- “LEFT Function” on page 405
- “TRIM Function” on page 640
- “TRIMN Function” on page 642

SUBSTR (left of =) Function

Replaces character value contents.

Category:  Character

Restriction:  This function is assigned an I18N Level 0 status, and is designed for SBCS data. Do not use this function to process DBCS or MBCS data. For more information, see Internationalization Compatibility.

Tip:  DBCS equivalent functions are KSUBSTR and KSUBSTRB.
Syntax

\[ \text{SUBSTR}(\text{variable}, \text{position} <, \text{length}>) = \text{characters-to-replace} \]

Required Arguments

\textit{variable}

specifies a character variable.

\textit{position}

specifies a numeric constant, variable, or expression that is the beginning character position.

\textit{characters-to-replace}

specifies a character constant, variable, or expression that replaces the contents of \textit{variable}.

Tip  Enclose a literal string of characters in quotation marks.

Optional Argument

\textit{length}

specifies a numeric constant, variable, or expression that is the length of the substring that is replaced.

Restriction  \textit{length} cannot be larger than the length of the expression that remains in \textit{variable} after \textit{position}.

Tip  If you omit \textit{length}, SAS uses all of the characters on the right side of the assignment statement to replace the values of \textit{variable}.

Details

If you use an undeclared variable, it is assigned a default length of 8 when the SUBSTR function is compiled.

When you use the SUBSTR function on the left side of an assignment statement, SAS replaces the value of \textit{variable} with the expression on the right side. SUBSTR replaces \textit{length} characters starting at the character that you specify in \textit{position}.

Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{a='KIDNAP'};</td>
<td></td>
</tr>
<tr>
<td>\texttt{substr(a, 1, 3)=CAT;}</td>
<td></td>
</tr>
<tr>
<td>\texttt{put a;}</td>
<td>CATHAP</td>
</tr>
<tr>
<td>\texttt{b=a;}</td>
<td></td>
</tr>
<tr>
<td>\texttt{substr(b, 4)=TY;}</td>
<td></td>
</tr>
<tr>
<td>\texttt{put b;}</td>
<td>CATTY</td>
</tr>
</tbody>
</table>
See Also

Functions:

- “SUBSTR (right of =) Function” on page 617

SUBSTR (right of =) Function

Extracts a substring from an argument.

Category: Character

Restriction: This function is assigned an I18N Level 0 status, and is designed for SBCS data. Do not use this function to process DBCS or MBCS data. For more information, see Internationalization Compatibility.

Tip: DBCS equivalent functions are KSUBSTR and KSUBSTRB.

Syntax

<variable>=SUBSTR(string, position <, length>)

Required Arguments

string
specifies a character constant, variable, or expression.

position
specifies a numeric constant, variable, or expression that is the beginning character position.

Optional Arguments

variable
specifies a valid SAS variable name.

length
specifies a numeric constant, variable, or expression that is the length of the substring to extract.

Interaction
If length is zero, a negative value, or larger than the length of the expression that remains in string after position, SAS extracts the remainder of the expression. SAS also sets _ERROR_ to 1 and prints a note to the log indicating that the length argument is invalid.

Tip
If you omit length, SAS extracts the remainder of the expression.

Details

In a DATA step, if the SUBSTR (right of =) function returns a value to a variable that has not previously been assigned a length, then that variable is given the length of the first argument.

The SUBSTR function returns a portion of an expression that you specify in string. The portion begins with the character that you specify by position, and is the number of characters that you specify in length.
Example

The following SAS statements produce this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>date='06MAY98';</td>
<td>MAY 98</td>
</tr>
<tr>
<td>month=substr(date, 3, 3);</td>
<td>MAY 98</td>
</tr>
<tr>
<td>year=substr(date, 6, 2);</td>
<td>MAY 98</td>
</tr>
<tr>
<td>put @1 month @5 year;</td>
<td>MAY 98</td>
</tr>
</tbody>
</table>

See Also

Functions:

- “SUBPAD Function” on page 1068
- “SUBSTR (left of =) Function” on page 615
- “SUBSTRN Function” on page 618

SUBSTRN Function

Returns a substring, allowing a result with a length of zero.

- **Category:** Character
- **Restriction:** This function is assigned an I18N Level 0 status, and is designed for SBCS data. Do not use this function to process DBCS or MBCS data. For more information, see Internationalization Compatibility.
- **Tip:** KSUBSTR has the same functionality.

**Syntax**

SUBSTRN(*string*, *position* <, *length*>)

**Required Arguments**

- **string**
  - specifies a character or numeric constant, variable, or expression.
  - If *string* is numeric, then it is converted to a character value that uses the BEST32 format. Leading and trailing blanks are removed, and no message is sent to the SAS log.

- **position**
  - is an integer that specifies the position of the first character in the substring.
Optional Argument

**length**

is an integer that specifies the length of the substring. If you do not specify `length`, the SUBSTRN function returns the substring that extends from the position that you specify to the end of the string.

Details

**Length of Returned Variable**

In a DATA step, if the SUBSTRN function returns a value to a variable that has not previously been assigned a length, then that variable is given the length of the first argument.

The Basics

The following information applies to the SUBSTRN function:

- The SUBSTRN function returns a string with a length of zero if either `position` or `length` has a missing value.
- If the position that you specify is non-positive, the result is truncated at the beginning, so that the first character of the result is the first character of the string. The length of the result is reduced accordingly.
- If the length that you specify extends beyond the end of the string, the result is truncated at the end, so that the last character of the result is the last character of the string.

Some language elements are supported in SAS but not in CAS. An example is the DATALINES statement. Some examples in this document use DATALINES in SAS to create a data set using the CAS engine. For more information, see “Functions in SAS and CAS” on page 54.

Using the SUBSTRN Function in a Macro

If you call SUBSTRN by using the %SYSFUNC macro, then the macro processor resolves the first argument (`string`) to determine whether the argument is character or numeric. If you do not want the first argument to be evaluated as a macro expression, use one of the macro-quoting functions in the first argument.

Comparisons

The following table lists comparisons between the SUBSTRN and SUBSTR functions:

<table>
<thead>
<tr>
<th>Condition</th>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>the value of <code>position</code> is nonpositive</td>
<td>SUBSTRN</td>
<td>returns a result beginning at the first character of the string.</td>
</tr>
<tr>
<td>the value of <code>position</code> is nonpositive</td>
<td>SUBSTR</td>
<td>• writes a note to the log stating that the second argument is invalid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• sets <em>ERROR</em> =1.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• returns the substring that extends from the position that you specified to the end of the string.</td>
</tr>
<tr>
<td>Condition</td>
<td>Function</td>
<td>Result</td>
</tr>
<tr>
<td>-----------</td>
<td>----------</td>
<td>--------</td>
</tr>
<tr>
<td>the value of <code>length</code> is nonpositive</td>
<td><code>SUBSTRN</code></td>
<td>returns a result with a length of zero.</td>
</tr>
</tbody>
</table>
| the value of `length` is nonpositive | `SUBSTR` | • writes a note to the log stating that the third argument is invalid.  
• sets `_ERROR_` = 1.  
• returns the substring that extends from the position that you specified to the end of the string. |
| the substring that you specify extends past the end of the string | `SUBSTRN` | truncates the result. |
| the substring that you specify extends past the end of the string | `SUBSTR` | • writes a note to the log stating that the third argument is invalid.  
• sets `_ERROR_` = 1.  
• returns the substring that extends from the position that you specified to the end of the string. |

**Examples**

**Example 1: Manipulating Strings with the `SUBSTRN` Function**

The following example shows how to manipulate strings with the `SUBSTRN` function.

```sas
data test;
  retain string "abcd";
  drop string;
  do Position=-1 to 6;
    do Length=max(-1,-position) to 7-position;
      Result=substrn(string, position, length);
      output;
    end;
  end;
datalines;
abcd
;
proc print noobs data=test;
run;
```
### The SAS System

<table>
<thead>
<tr>
<th>Position</th>
<th>Length</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>3</td>
<td>a</td>
</tr>
<tr>
<td>-1</td>
<td>4</td>
<td>ab</td>
</tr>
<tr>
<td>-1</td>
<td>5</td>
<td>abc</td>
</tr>
<tr>
<td>-1</td>
<td>6</td>
<td>abcd</td>
</tr>
<tr>
<td>-1</td>
<td>7</td>
<td>abcd</td>
</tr>
<tr>
<td>-1</td>
<td>8</td>
<td>abcd</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>a</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>ab</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>abc</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>abcd</td>
</tr>
<tr>
<td>0</td>
<td>6</td>
<td>abcd</td>
</tr>
<tr>
<td>0</td>
<td>7</td>
<td>abcd</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>a</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>ab</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>abc</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>abcd</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>abcd</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>abcd</td>
</tr>
<tr>
<td>2</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>b</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>bc</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>bcd</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>bcd</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>bcd</td>
</tr>
</tbody>
</table>
Example 2: Comparison between the SUBSTR and SUBSTRN Functions
The following example compares the results of using the SUBSTR function and the SUBSTRN function when the first argument is numeric.

```sas
data _null_;  
  substr_result="*" || substr(1234.5678,2,6) || "*";  
  put substr_result=;  
  subtrn_result="*" || substrn(1234.5678,2,6) || "*";  
  put subtrn_result=;  
run;
```

SAS writes the following output to the log:

```
substr_result=*  1234*
substrn_result=*234.56*
```

See Also

Functions:
• “SUBPAD Function” on page 1068
• “SUBSTR (left of =) Function” on page 615
• “SUBSTR (right of =) Function” on page 617

SUM Function
Returns the sum of the nonmissing arguments.

| Category: | Descriptive Statistics |

Syntax

`SUM(argument-1 <, argument-2, ...>)`

Required Argument

`argument`

specifies a numeric constant, variable, or expression. If all the arguments have missing values, then one of the following occurs:

• If you use only one argument, then the value of that argument is returned.
• If you use two or more arguments, then a standard missing value (.) is returned.

Otherwise, the result is the sum of the nonmissing values. The argument list can consist of a variable list, which is preceded by OF.

Example
The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1=sum(4, 9, 3, 8);</td>
<td>24</td>
</tr>
<tr>
<td>x2=sum(4, 9, 3, 8, .);</td>
<td>24</td>
</tr>
<tr>
<td>x3=sum(56);</td>
<td>56</td>
</tr>
<tr>
<td>x1=9; x2=39; x3=sum(of x1-x2);</td>
<td>48</td>
</tr>
<tr>
<td>x1=5; x2=6; x3=4; x4=9; y1=34; y2=12; y3=74; y4=39; result=sum(of x1-x4, of y1-y5);</td>
<td>183</td>
</tr>
<tr>
<td>x1=55; x2=35; x3=6; x4=sum(of x1-x3, 5);</td>
<td>101</td>
</tr>
<tr>
<td>x1=7; x2=7; x5=sum(x1-x2);</td>
<td>0</td>
</tr>
<tr>
<td>y1=20; y2=30; x6=sum(of y:);</td>
<td>50</td>
</tr>
</tbody>
</table>

**SUMABS Function**

Returns the sum of the absolute values of the nonmissing arguments.

**Category:** Descriptive Statistics

**Syntax**

```
SUMABS(value-1 <, value-2 ...>)
```

**Required Argument**

`value`

specifies a numeric expression.

**Details**

If all arguments have missing values, then the result is a missing value. Otherwise, the result is the sum of the absolute values of the nonmissing values.
Examples

Example 1: Calculating the Sum of Absolute Values
The following example returns the sum of the absolute values of the nonmissing arguments.

```sas
data _null_;  
    x=sumabs(1, ., -2, 0, 3, .q, -4);  
    put x=;  
run;
```

SAS writes the following output to the log:

```
x=10
```

Example 2: Calculating the Sum of Absolute Values When You Use a Variable List
The following example uses a variable list and returns the sum of the absolute value of the nonmissing arguments.

```sas
data _null_;  
    x1=1;  
    x2=3;  
    x3=4;  
    x4=3;  
    x5=1;  
    x=sumabs(of x1-x5);  
    put x=;  
run;
```

SAS writes the following output to the log:

```
x=12
```

TAN Function

Returns the tangent.

Category: Trigonometric

Syntax

`TAN(argument)`

Required Argument

`argument` specifies a numeric constant, variable, or expression and is expressed in radians. If the magnitude of `argument` is so great that `mod(argument, pi)` is accurate to less than about three decimal places, `TAN` returns a missing value.

Restriction: cannot be an odd multiple of π /2
Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=tan(0.5);</td>
<td>0.5463024898</td>
</tr>
<tr>
<td>x=tan(0);</td>
<td>0</td>
</tr>
<tr>
<td>x=tan(3.14159/3);</td>
<td>1.7320472695</td>
</tr>
</tbody>
</table>

**TIME Function**

Returns the current time of day as a numeric SAS time value.

**Category:** Date and Time

**Interaction:** If the value of the TIMEZONE= system option is set to a time zone name or time zone ID, the date and time values that are returned for this function are determined by the time zone.

**Syntax**

TIME()

**Examples**

**Example 1: Displaying the Current Time**

SAS assigns CURRENT a SAS time value that corresponds to 14:32:00 if the following statements are executed exactly at 2:32 PM:

```sas
current=time();
put current=time.;
```

**Example 2: Determining a Time Value for an America/Los_Angeles Time Zone**

This example shows how the TIME function returns a value based on the value of the TIMEZONE= system option.

```sas
option timezone='America/Los_Angeles';
data _null_;   
t1=time();
   put t1=nltimap15.;
run;
```

SAS writes the following results to the log:

```
t1=11:10:24 AM
```
Example 3: Determining a Time Value for a Europe/London Time Zone

This example shows how the TIME function returns a value based on the value of the TIMEZONE= system option.

```sas
option timezone='Europe/London';
data _null_
  t2=time();
  put t2=nltimap15.;
run;
```

SAS writes the following results to the log:

```
t2=07:11:08 PM
```

See Also

Functions:

- “DATE Function” on page 220
- “DATETIME Function” on page 222
- “TODAY Function” on page 630

TIMEPART Function

Extracts a time value from a SAS datetime value.

**Category:** Date and Time

**Syntax**

```
TIMEPART(datetime)
```

**Required Argument**

`datetime`

is a numeric constant, variable, or expression that represents a SAS datetime value.

**Example**

SAS assigns TIME a SAS value that corresponds to 10:40:17 if the following statements are executed exactly at 10:40:17 a.m. on any date:

```sas
  datim=datetime();
  time=timepart(datim);
```

TIMEVALUE Function

Returns the equivalent of a reference amount at a base date by using variable interest rates.

**Category:** Financial
Syntax

TIMEVALUE(base-date, reference-date, reference-amount, compounding-interval, date-1, rate-1 <, date-2, rate-2, …>)

Required Arguments

- **base-date** is a SAS date. The value that is returned is the time value of reference-amount at base-date.
- **reference-date** is a SAS date. Reference-date is the date of reference-amount.
- **reference-amount** is numeric. Reference-amount is the amount at reference-date.
- **compounding-interval** is a SAS interval. Compounding-interval is the compounding interval.
- **date** is a SAS date. Each date is paired with a rate. date is the time at which rate takes effect.
- **rate** is a numeric percentage. Each rate is paired with a date. Rate is the interest rate that starts on date.

Details

The following details apply to the TIMEVALUE function:

- The values for rates must be between –99 and 120.
- The list of date-rate pairs does not need to be sorted by date.
- When multiple rate changes occur on a single date, the TIMEVALUE function applies only the final rate that is listed for that date.
- Simple interest is applied for partial periods.
- There must be a valid date-rate pair whose date is at or prior to both the reference-date and the base-date.

Example

- You can express the accumulated value of an investment of $1,000 at a nominal interest rate of 10% compounded monthly for one year as the following:

  ```
  amount_base1=TIMEVALUE("01jan2001"d, "01jan2000"d, 1000, "MONTH", "01jan2000"d, 10);
  ```

- If the interest rate jumps to 20% halfway through the year, the resulting calculation would be as follows:

  ```
  amount_base2=TIMEVALUE("01jan2001"d, "01jan2000"d, 1000, "MONTH", "01jan2000"d, 10, "01jul2000"d, 20);
  ```

- The date-rate pairs do not need to be sorted by date. This flexibility allows amount_base2 and amount_base3 to assume the same value:

  ```
  amount_base3=TIMEVALUE("01jan2001"d, "01jan2000"d, 1000, "MONTH", "01jul2000"d, 20, "01jan2000"d, 10);
  ```
TINV Function

Returns a quantile from the $t$ distribution.

**Category:** Quantile

**Syntax**

$$\text{TINV}(p, df, <nc>)$$

**Required Arguments**

$p$

- is a numeric probability.
- Range: $0 < p < 1$

$df$

- is a numeric degrees of freedom parameter.
- Range: $df > 0$

**Optional Argument**

$nc$

- is an optional numeric noncentrality parameter.

**Details**

The TINV function returns the $p^{th}$ quantile from the Student's $t$ distribution with degrees of freedom $df$ and a noncentrality parameter $nc$. The probability that an observation from a $t$ distribution is less than or equal to the returned quantile is $p$.

TINV accepts a noninteger degree of freedom parameter $df$. If the optional parameter $nc$ is not specified or is 0, the quantile from the central $t$ distribution is returned.

**CAUTION:**

For large values of $nc$, the algorithm can fail. In that case, a missing value is returned.

**Note:** TINV is the inverse of the PROBT function.

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=tinv(.95, 2);</td>
<td>2.9199855804</td>
</tr>
<tr>
<td>x=tinv(.95, 2.5, 3);</td>
<td>11.03833625</td>
</tr>
</tbody>
</table>
See Also

Functions:
• “QUANTILE Function” on page 549

TNONCT Function

Returns the value of the noncentrality parameter from the Student's t distribution.

Category:
Mathematical

Syntax

TNONCT(x, df, prob)

Required Arguments

x
is a numeric random variable.

df
is a numeric degrees of freedom parameter.

Range $df > 0$

prob
is a probability.

Range $0 < prob < 1$

Details

The TNONCT function returns the nonnegative noncentrality parameter from a noncentral t distribution whose parameters are $x$, $df$, and $nc$. A Newton-type algorithm is used to find a root $nc$ of the equation

$$P_t(x | df, nc) - prob = 0$$

The following relationship applies to the preceding equation:

$$P_t(x | df, nc) = \frac{1}{f} \int_{df}^{\infty} \frac{v}{2} - 1 \cdot \int_{-\infty}^{\varepsilon} \varepsilon \cdot \frac{(u - nc)^2}{2} dudv$$

If the algorithm fails to converge to a fixed point, a missing value is returned.

Example

The following example computes the noncentrality parameter from the t distribution.

data work;
    x=2;

df=4;
do nc=1 to 3 by .5;
    prob=probt(x, df, nc);
    ncc=tnonct(x, df, prob);
    output;
end;
run;
proc print;
run;

Figure 2.43  Output from the Computations of the Noncentrality Parameter for the \textit{t} Distribution

\begin{table}
\centering
\begin{tabular}{|c|c|c|c|c|c|}
\hline
Obs & x & df & nc & prob & ncc \\
\hline
1 & 2 & 4 & 1.0 & 0.76457 & 1.00000 \\
2 & 2 & 4 & 1.5 & 0.61893 & 1.50000 \\
3 & 2 & 4 & 2.0 & 0.45567 & 2.00000 \\
4 & 2 & 4 & 2.5 & 0.30115 & 2.50000 \\
5 & 2 & 4 & 3.0 & 0.17702 & 3.00000 \\
\hline
\end{tabular}
\caption{The SAS System}
\end{table}

\textbf{TODAY Function}

Returns the current date as a numeric SAS date value.

\begin{itemize}
    \item \textbf{Category}: Date and Time
    \item \textbf{Alias}: DATE
    \item \textbf{Interaction}: If the value of the \texttt{TIMEZONE=} system option is set to a time zone name or time zone ID, the date and time values that are returned for this function are determined by the time zone.
\end{itemize}

\textbf{Syntax}

\texttt{TODAY()}

\textbf{Details}

The TODAY function produces the current date in the form of a SAS date value, which is the number of days since January 1, 1960.
Examples

Example 1: Using the TODAY Function in an Accounting Example
These statements illustrate a practical use of the TODAY function:

```sas
data _null_;  
tday=today();  
if (tday-datedue)> 15 then  
    do;  
    put 'As of ' tday date9. ' Account #'  
        account 'is more than 15 days overdue.';  
    end;  
run;
```

Example 2: Determining the Current Date for an America/Denver Time Zone
This example shows how the TODAY function returns a value based on the value of the TIMEZONE= system option.

```sas
option timezone='America/Denver';
data _null_;  
d1=today();  
put d1=nldate.;  
run;
```

SAS writes the following results to the log:

```
d1=November 07, 2012
```

Example 3: Determining the Current Date for an Asia/Seoul Time Zone
This example shows how the TODAY function returns a value based on the value of the TIMEZONE= system option.

```sas
option timezone='Asia/Seoul';
data _null_;  
d2=today();  
put d2=nldate.;  
run;
```

SAS writes the following results to the log:

```
d2=November 08, 2012
```

See Also

Functions:
- “DATE Function” on page 220
- “DATETIME Function” on page 222
- “TIME Function” on page 625
TRANSLATE Function

Replaces specific characters in a character expression.

**Category:** Character

**Restriction:** This function is assigned an I18N Level 0 status, and is designed for SBCS data. Do not use this function to process DBCS or MBCS data. For more information, see Internationalization Compatibility.

**Syntax**

```
TRANSLATE(source, to-1, from-1 <, ...to-n, from-n>)
```

**Required Arguments**

- **source**
  specifies a character constant, variable, or expression that contains the original character string.

- **to**
  specifies the characters that you want TRANSLATE to use as substitutes.

- **from**
  specifies the characters that you want TRANSLATE to replace.

**Details**

**General Information**

Values of to and from correspond on a character-by-character basis. TRANSLATE changes the first character of from to the first character of to, and so on.

In a DATA step, if the TRANSLATE function returns a value to a variable that has not previously been assigned a length, then that variable is given the length of the first argument.

The maximum number of pairs of to and from arguments that TRANSLATE accepts depends on which operating environment you use to run SAS. There is no functional difference between using several pairs of short arguments, or fewer pairs of longer arguments.

**TRANSLATE Specifics under UNIX**

Under UNIX, you must specify a pair of arguments that consist of a to argument and a from argument, and you can use a comma as a placeholder.

**Comparisons**

The TRNWRD function differs from TRANSLATE in that it scans for words (or patterns of characters) and replaces those words with a second word (or pattern of characters).

**Note:** The order of arguments is different between the TRNWRD and TRANSLATE functions. In TRNWRD, the second argument is target and the third argument is replacement. In TRANLSATE, the second argument is to and the third argument is from. The character string specified by target is similar to the from argument in...
TRANSLATE. The character string specified in replacement is similar to the to argument in TRANSLATE.

Example
The following SAS statements produce this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=translate('XYZW', 'AB', 'VW'); put x;</td>
<td>XYZB</td>
</tr>
<tr>
<td>string1='AABBAABABB'; y=translate(string1,'12','AB'); put y=;</td>
<td>y=1122112122</td>
</tr>
</tbody>
</table>

See Also

Functions:

- “TRANWRD Function” on page 635

TRANSTRN Function
Replaces or removes all occurrences of a substring in a character string.

<table>
<thead>
<tr>
<th>Category:</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restriction:</td>
<td>This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility.</td>
</tr>
</tbody>
</table>

Syntax
TRANSTRN(source, target, replacement)

Required Arguments

source
specifies a character constant, variable, or expression that you want to translate.

target
specifies a character constant, variable, or expression that is searched for in source.

Requirement  The length for target must be greater than zero.

replacement
specifies a character constant, variable, or expression that replaces target.
Details

Length of Returned Variable
In a DATA step, if the TRANSTRN function returns a value to a variable that has not previously been assigned a length, then that variable is given a length of 200 bytes. You can use the LENGTH statement, before calling TRANSTRN, to change the length of the value.

The Basics
The TRANSTRN function replaces or removes all occurrences of a given substring within a character string. The TRANSTRN function does not remove trailing blanks in the target string and the replacement string. To remove all occurrences of target, specify replacement as TRIMN("")

Some language elements are supported in SAS but not in CAS. An example is the DATALINES statement. Some examples in this document use DATALINES in SAS to create a data set using the CAS engine. For more information, see “Functions in SAS and CAS” on page 54.

Comparisons
The TRANWRD function differs from the TRANSTRN function because TRANSTRN allows the replacement string to have a length of zero. TRANWRD uses a single blank instead when the replacement string has a length of zero.

The TRANSLATE function converts every occurrence of a user-supplied character to another character. TRANSLATE can scan for more than one character in a single call. In doing this scan, however, TRANSLATE searches for every occurrence of any of the individual characters within a string. That is, if any letter (or character) in the target string is found in the source string, it is replaced with the corresponding letter (or character) in the replacement string.

The TRANSTRN function differs from TRANSLATE in that TRANSTRN scans for substrings and replaces those substrings with a second substring.

Examples

Example 1: Replacing All Occurrences of a Word
These statements and these values produce these results:

```sas
name=transtrn(name, "Mrs.", "Ms.");
name=transtrn(name, "Miss", "Ms.");
put name;
```

<table>
<thead>
<tr>
<th>Value</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mrs. Joan Smith</td>
<td>Ms. Joan Smith</td>
</tr>
<tr>
<td>Miss Alice Cooper</td>
<td>Ms. Alice Cooper</td>
</tr>
</tbody>
</table>

Example 2: Removing Blanks from the Search String
In this example, the TRANSTRN function does not replace the source string because the target string contains blanks.
The LENGTH statement pads target with blanks to the length of 10, which causes the
TRANSTRN function to search for the character string ‘FISH ’ in SALELIST.
Because the search fails, this line is written to the SAS log:

CATFISH

You can use the TRIM function to exclude trailing blanks from a target or replacement
variable. Use the TRIM function with target:

salelist=transtrn(salelist,trim(target), replacement);
put salelist;

Now, this line is written to the SAS log:

CATNIP

Example 3: Zero Length in the Third Argument of the TRANSTRN Function

The following example shows the results of the TRANSTRN function when the third
argument, replacement, has a length of zero. In the DATA step, a character constant that
consists of two quotation marks represents a single blank, and not a zero-length string. In
the following example, the results for string1 are different from the results for string2.

```
data _null_;  
  string1='*' || transtrn('abcxabc', 'abc', trimn('')) || '*';  
  put string1=;  
  string2='*' || transtrn('abcxabc', 'abc', '') || '*';  
  put string2=;  
run;  
```

SAS writes the following output to the log:

```
string1=*x*  
string2=* x *
```

See Also

Functions:

- “TRANSLATE Function” on page 632

TRANWRD Function

Replaces all occurrences of a substring in a character string.

Category: Character
Restriction: This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility.

Syntax

TRANWRD(source, target, replacement)

Required Arguments

source
specifies a character constant, variable, or expression that you want to translate.

target
specifies a character constant, variable, or expression that is searched for in source.

replacement
specifies a character constant, variable, or expression that replaces target. When the replacement string has a length of zero, TRANWRD uses a single blank instead.

Details

Length of Returned Variable
In a DATA step, if the TRANWRD function returns a value to a variable that has not previously been assigned a length, then that variable is given a length of 200 bytes. You can use the LENGTH statement, before calling TRANWRD, to change the length of the value.

The Basics
The TRANWRD function copies the value in source to the result string while searching for all non-overlapping substrings in source that are equal to the value in target. Each of these substrings is omitted from the result and the value of replacement is copied in its place. The TRANWRD function does not remove trailing blanks in the target string or the replacement string.

The example, “Example 3: Removing Blanks from the Search String Using the TRANWRD Function with CAS” on page 638 shows how to run the TRANWRD function in CAS.

Some language elements are supported in SAS but not in CAS. An example is the DATALINES statement. Some examples in this document use DATALINES in SAS to create a data set using the CAS engine. For more information, see “Functions in SAS and CAS” on page 54.

Comparisons
The TRANWRD function differs from the TRANSTRN function because TRANSTRN allows the replacement string to have a length of zero. TRANWRD uses a single blank instead when the replacement string has a length of zero.

The TRANSLATE function converts every occurrence of a user-supplied character to another character. TRANSLATE can scan for more than one character in a single call. In doing this scan, however, TRANSLATE searches for every occurrence of any of the individual characters within a string. That is, if any letter (or character) in the target
The TRANWRD function differs from TRANSLATE in that TRANWRD scans for substrings and replaces those substrings with a second substring.

Note: The order of arguments is different between the TRANWRD and TRANSLATE functions. In TRANWRD, the second argument is target and the third argument is replacement. In TRANSLATE, the second argument is from and the third argument is to. The character string specified by target is similar to the from argument in TRANSLATE. The character string specified in replacement is similar to the to argument in TRANSLATE.

Examples

Example 1: Replacing All Occurrences of a Word

These statements and these values produce these results:

```sas
name=tranwrd(name, "Mrs.", "Ms.");
name=tranwrd(name, "Miss", "Ms.");
put name;
```

<table>
<thead>
<tr>
<th>Value</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mrs. Joan Smith</td>
<td>Ms. Joan Smith</td>
</tr>
<tr>
<td>Miss Alice Cooper</td>
<td>Ms. Alice Cooper</td>
</tr>
</tbody>
</table>

Example 2: Removing Blanks from the Search String Using the TRANWRD Function with SAS

In this example, the TRANWRD function does not replace the source string because the target string contains blanks.

```sas
data list;
   input salelist $;
   length target $10 replacement $3;
   target='FISH';
   replacement='NIP';
   salelist=tranwrd(salelist, target, replacement);
   put salelist;
datalines;
CATFISH
;
```

The LENGTH statement pads target with blanks to the length of 10, which causes the TRANWRD function to search for the character string ‘FISH’ in SALELIST. Because the search fails, this line is written to the SAS log:

```
CATFISH
```

You can use the TRIM function to exclude trailing blanks from a target or replacement variable. Use the TRIM function with target:

```sas
salelist=tranwrd(salelist,trim(target), replacement);
put salelist;
```

Now, this line is written to the SAS log:
CATNIP

**Example 3: Removing Blanks from the Search String Using the TRANWRD Function with CAS**

In this example, the TRANWRD function does not replace the source string because the target string contains blanks. The first section of code in this example runs in SAS and creates the data set, list, with the CAS engine. The second section of code runs in CAS, accesses the data in list that is in CAS, and then creates the listn data set.

```sas
data casuser.list;
  input salelist $;
  datalines;
  CATFISH
;

data casuser.listn;
  set casuser.list;
  length target $10 replacement $3;
  target='FISH';
  replacement='NIP';
  salelist=tranwrd(salelist, target, replacement);
  put salelist;
run;
```

The **LENGTH** statement pads **target** with blanks to the length of 10, which causes the **TRANWRD** function to search for the character string **'FISH '** in **SALELIST**. Because the search fails, this line is written to the SAS log:

```
CATFISH
```

You can use the **TRIM** function to exclude trailing blanks from a target or replacement variable. Use the **TRIM** function with **target**:

```sas
salelist=tranwrd(salelist,trim(target), replacement);
put salelist;
```

Now, this line is written to the SAS log:

```
CATNIP
```

**Example 4: Zero Length in the Third Argument of the TRANWRD Function**

The following example shows the results of the TRANWRD function when the third argument, **replacement**, has a length of zero. In this case, TRANWRD uses a single blank. In the **DATA** step, a character constant that consists of two consecutive quotation marks represents a single blank, and not a zero-length string. In this example, the results for **string1** and **string2** are the same:

```sas
data _null_;
  string1='*' || tranwrd('abcxabc', 'abc', trimn('')) || '*';
  put string1=;
  string2='*' || tranwrd('abcxabc', 'abc', '') || '*';
  put string2=;
run;
```

SAS writes the following output to the log:

```
string1=* x *
string2=* x *
```
Example 5: Removing Repeated Commas
You can use the TRANWRD function to remove repeated commas in text, and replace the repeated commas with a single comma. In the following example, the TRANWRD function is used twice: to replace three commas with one comma, and to replace the ending two commas with a period:

```sas
data _null_
   mytxt='If you exercise your power to vote,,,then your opinion will be heard,,';
   newtext=tranwrd(mytxt, ',,,', ',');
   newtext2=tranwrd(newtext, ',,' , '.');
   put // mytxt= / newtext= / newtext2=;
run;
```

SAS writes the following output to the log:

```
mytxt=If you exercise your power to vote,,,then your opinion will be heard,;
newtext=If you exercise your power to vote,then your opinion will be heard,;
newtext2=If you exercise your power to vote,then your opinion will be heard.
```

See Also

Functions:
- “TRANSLATE Function” on page 632

TRIGAMMA Function

Returns the value of the trigamma function.

Category: Mathematical

Syntax

```
TRIGAMMA(argument)
```

Required Argument

`argument`

specifies a numeric constant, variable, or expression.

Restriction

Nonpositive integers are invalid.

Details

The TRIGAMMA function returns the derivative of the DIGAMMA function. For `argument > 0`, the TRIGAMMA function is the second derivative of the LGAMMA function.
Example

The following SAS statement produces this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=trigamma(3);</td>
<td>0.3949340668</td>
</tr>
</tbody>
</table>

TRIM Function

Removes trailing blanks from a character string, and returns one blank if the string is missing.

**Category:** Character

**Restriction:** This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility.

**Tip:** DBCS equivalent function is “KTRIM Function” in SAS Viya National Language Support (NLS): Reference Guide.

**Syntax**

\[ \text{TRIM}(\text{argument}) \]

**Required Argument**

\( \text{argument} \)

specifies a character constant, variable, or expression.

**Details**

**Length of Returned Variable**

In a DATA step, if the TRIM function returns a value to a variable that has not previously been assigned a length, then that variable is given the length of the argument.

**The Basics**

TRIM copies a character argument, removes trailing blanks, and returns the trimmed argument as a result. If the argument is blank, TRIM returns one blank. TRIM is useful for concatenating because concatenation does not remove trailing blanks.

Assigning the results of TRIM to a variable does not affect the length of the receiving variable. If the trimmed value is shorter than the length of the receiving variable, SAS pads the value with new blanks as it assigns it to the variable.

Some language elements are supported in SAS but not in CAS. An example is the DATALINES statement. Some examples in this document use DATALINES in SAS to create a data set using the CAS engine. For more information, see “Functions in SAS and CAS” on page 54.
Comparisons

The TRIM and TRIMN functions are similar. TRIM returns one blank for a blank string. TRIMN returns a string with a length of zero for a blank string.

Examples

Example 1: Using the TRIM Function in SAS

These statements remove trailing blanks and produce these results:

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>data test; input part1 $ 1-10 part2 $ 11-20; hasblank=part1</td>
<td></td>
</tr>
</tbody>
</table>

Example 2: Using the TRIM Function in SAS and CAS

The first section of code in this example runs in SAS and creates the data set, apple, with the CAS engine. The second section of code runs in CAS, and accesses the data set, apple.

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>data casuser.apple; input part1 $ 1-10 part2 $ 11-20; datalines; apple sauce ;</td>
<td>apple sauce applesauce</td>
</tr>
<tr>
<td>data casuser.apple_sauce; set casuser.apple; hasblank=part1</td>
<td></td>
</tr>
</tbody>
</table>

Example 3: Concatenating a Blank Character Expression

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=&quot;A&quot;</td>
<td></td>
</tr>
<tr>
<td>x=&quot; &quot;; y=&quot;&quot;&gt;&quot;</td>
<td></td>
</tr>
</tbody>
</table>
TRIMN Function

Removes trailing blanks from character expressions, and returns a string with a length of zero if the expression is missing.

**Category:** Character

**Restriction:** This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see [Internationalization Compatibility](#).

**Syntax**

```
TRIMN(argument)
```

**Required Argument**

`argument` specifies a character constant, variable, or expression.

**Details**

**Length of Returned Variable**

In a DATA step, if the TRIMN function returns a value to a variable that has not previously been assigned a length, then that variable is given the length of the argument.

Assigning the results of TRIMN to a variable does not affect the length of the receiving variable. If the trimmed value is shorter than the length of the receiving variable, SAS pads the value with new blanks as it assigns it to the variable.

**The Basics**

TRIMN copies a character argument, removes all trailing blanks, and returns the trimmed argument as a result. If the argument is blank, TRIMN returns a string with a length of zero. TRIMN is useful for concatenating because concatenation does not remove trailing blanks.

**Comparisons**

The TRIMN and TRIM functions are similar. TRIMN returns a string with a length of zero for a blank string. TRIM returns one blank for a blank string.

---

**See Also**

**Functions:**

- “COMPRESS Function” on page 191
- “LEFT Function” on page 405
- “RIGHT Function” on page 571
- “TRIMN Function” on page 642
Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=&quot;A&quot;</td>
<td></td>
</tr>
<tr>
<td>put x;</td>
<td></td>
</tr>
<tr>
<td>x=&quot;   &quot;;</td>
<td></td>
</tr>
<tr>
<td>z=&quot;&gt;&quot;</td>
<td></td>
</tr>
<tr>
<td>put z;</td>
<td></td>
</tr>
</tbody>
</table>

See Also

Functions:
- “COMPRESS Function” on page 191
- “LEFT Function” on page 405
- “RIGHT Function” on page 571
- “TRIM Function” on page 640

TRUNC Function

Truncates a numeric value to a specified number of bytes.

**Category:** Truncation

**Syntax**

```
TRUNC(number, length)
```

**Required Arguments**

- `number` specifies a numeric constant, variable, or expression.
- `length` specifies an integer.

**Details**

The TRUNC function truncates a full-length `number` (stored as a double) to a smaller number of bytes, as specified in `length` and pads the truncated bytes with 0s. The truncation and subsequent expansion duplicate the effect of storing numbers in less than full length and then reading them.

**Example**

Here is an example of using the TRUNC function.
data test;
  length x 3;
  x=1/5;
run;
data test2;
  set test;
  if x ne 1/5 then
    put 'x ne 1/5';
  if x eq trunc(1/5,3) then
    put 'x eq trunc(1/5,3)';
run;

SAS writes the following output to the log:

```
x ne 1/5
x eq trunc(1/5,3)
```

The variable X is stored with a length of 3. Therefore, each of the above comparisons is true.

---

**UNIFORM Function**

Returns a random variate from a uniform distribution.

- **Category:** Random Number
- **Alias:** RANUNI
- **See:** “RANUNI Function” on page 1039

---

**UPCASE Function**

Converts all letters in an argument to uppercase.

- **Category:** Character
- **Restriction:** This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility.

**Syntax**

```
UPCASE(argument)
```

**Required Argument**

- **argument**
  - specifies a character constant, variable, or expression.

**Details**

In a DATA step, if the UPCASE function returns a value to a variable that has not previously been assigned a length, then that variable is given the length of the argument.
The `UPCASE` function copies a character argument, converts all lowercase letters to uppercase letters, and returns the altered value as a result.

**Example**

The following SAS statements produce this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>name=upcase('John B. Smith');</code></td>
<td><code>JOHN B. SMITH</code></td>
</tr>
<tr>
<td><code>put name;</code></td>
<td></td>
</tr>
</tbody>
</table>

**See Also**

**Functions:**

- “LOWCASE Function” on page 423
- “PROPCASE Function” on page 1020

---

**USS Function**

Returns the uncorrected sum of squares of the nonmissing arguments.

**Category:** Descriptive Statistics

**Syntax**

\[
\text{USS}(\text{argument-1}, \ldots, \text{argument-n})
\]

**Required Argument**

`argument` specifies a numeric constant, variable, or expression. At least one nonmissing argument is required. Otherwise, the function returns a missing value. If you have more than one argument, the argument list can consist of a variable list, which is preceded by `OF`.

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x1=uss(4, 2, 3.5, 6);</code></td>
<td>68.25</td>
</tr>
<tr>
<td><code>x2=uss(4, 2, 3.5, 6, .);</code></td>
<td>68.25</td>
</tr>
<tr>
<td><code>x3=uss(of x1-x2);</code></td>
<td>9316.125</td>
</tr>
</tbody>
</table>
**UUIDGEN Function**

Returns the short or binary form of a Universally Unique Identifier (UUID).

**Category:** Special

### Syntax

```
UUIDGEN(<maximum-warnings <, binary-result>>)
```

### Required Argument

**maximum-warnings**

specifies an integer value that represents the maximum number of warnings that this function writes to the log.

Default 1

### Optional Argument

**binary-result**

specifies an integer value that indicates whether this function should return a binary result. Nonzero indicates a binary result should be returned. Zero indicates that a character result should be returned.

Default 0

### Details

#### Length of Returned Variable in a DATA Step

If the UUIDGEN function returns a value to a variable that has not previously been assigned a length, then that variable is given a length of 200 bytes.

#### The Basics

The UUIDGEN function returns a UUID (a unique value) for each call. The default result is 36 characters long and it looks like this:

```
5ab6fa40-426b-4375-bb22-2d0291f43319
```

A binary result is 16 bytes long.

---

**VAR Function**

Returns the variance of the nonmissing arguments.

**Category:** Descriptive Statistics

### Syntax

```
VAR(argument-1, argument-2 <, ...argument-n>)
```
**Required Argument**

*argument*

specifies a numeric constant, variable, or expression. At least two nonmissing arguments are required. Otherwise, the function returns a missing value. The argument list can consist of a variable list, which is preceded by `OF`.

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x1=var(4, 2, 3.5, 6);</code></td>
<td>2.7291666667</td>
</tr>
<tr>
<td><code>x2=var(4, 6, .);</code></td>
<td>2</td>
</tr>
<tr>
<td><code>x3=var(of x1-x2);</code></td>
<td>0.2658420139</td>
</tr>
</tbody>
</table>

---

**VARRAY Function**

Returns a value that indicates whether the specified name is an array.

**Category:** Variable Information  
**Restriction:** Use only with the DATA step

**Syntax**

```
VARRAY(name)
```

**Required Argument**

*name*

specifies a name that is expressed as a scalar or as an array reference.

**Restriction**

You cannot use an expression as an argument.

**Details**

VARRAY returns 1 if the given name is an array; it returns 0 if the given name is not an array.

**Comparisons**

- VARRAY returns a value that indicates whether the specified name is an array. VARRAYX returns a value that indicates whether the value of the specified expression is an array.
- VARRAY does not accept an expression as an argument. VARRAYX accepts expressions, but the value of the specified variable cannot denote an array reference.
• Related functions return the value of other variable attributes, such as the variable name, informat, format, among others. For a list, see the Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701.

Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>array x[3] x1-x3;</td>
<td>a=1</td>
</tr>
<tr>
<td>a=varray(x);</td>
<td></td>
</tr>
<tr>
<td>B=varray(x1);</td>
<td>B=0</td>
</tr>
<tr>
<td>put a=;</td>
<td></td>
</tr>
<tr>
<td>put B=;</td>
<td></td>
</tr>
</tbody>
</table>

See Also

Functions:

• Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701

VARRAYX Function

Returns a value that indicates whether the value of the specified argument is an array.

Category: Variable Information

Syntax

VARRAYX(expression)

Required Argument

expression

specifies a character constant, variable, or expression.

Restriction The value of the specified expression cannot denote an array reference.

Details

VARRAYX returns 1 if the value of the given argument is the name of an array; it returns 0 if the value of the given argument is not the name of an array.

Comparisons

• VARRAY returns a value that indicates whether the specified name is the name of an array. VARRAYX returns a value that indicates whether the value of the specified expression is the name of an array.
VARRAY does not accept an expression as an argument. VARRAYX accepts expressions, but the value of the specified variable cannot denote an array reference.

- Related functions return the value of other variable attributes, such as the variable name, informat, format, among others. For a list, see the Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701.

Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>array x(3) x1-x3;</td>
<td></td>
</tr>
<tr>
<td>array vx(4) $6 vx1 vx2 vx3 vx4 ('x' 'x1' 'x2' 'x3');</td>
<td>y=1</td>
</tr>
<tr>
<td>y=varrayx(vx(1));</td>
<td>z=0</td>
</tr>
<tr>
<td>z=varrayx(vx(2));</td>
<td></td>
</tr>
<tr>
<td>put y=;</td>
<td></td>
</tr>
<tr>
<td>put z=;</td>
<td></td>
</tr>
</tbody>
</table>

See Also

Functions:
- Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701

VERIFY Function

Returns the position of the first character in a string that is not in any of several other strings.

**Category:** Character

**Restriction:** This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility.

**Tip:** DBCS equivalent function is KVERIFY.

**Syntax**

`VERIFY(source, excerpt-1 <, ...excerpt-n>)`

**Required Arguments**

- `source`: specifies a character constant, variable, or expression.
- `excerpt`: specifies a character constant, variable, or expression. If you specify more than one excerpt, separate them with a comma.
Details

The VERIFY function returns the position of the first character in *source* that is not present in any excerpt. If VERIFY finds every character in *source* in at least one excerpt, it returns a 0.

Some language elements are supported in SAS but not in CAS. An example is the DATALINES statement. Some examples in this document use DATALINES in SAS to create a data set using the CAS engine. For more information, see “Functions in SAS and CAS” on page 54.

Examples

**Example 1**
The code in this example runs in SAS, creates the data set, scores, and produces these results.

*Table 2.3  Using the VERIFY Function in SAS*

<table>
<thead>
<tr>
<th>Statements</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>data scores;</td>
<td>INVALID Grade=q</td>
</tr>
<tr>
<td>input Grade : $1. @@;</td>
<td></td>
</tr>
<tr>
<td>check='abcdef';</td>
<td></td>
</tr>
<tr>
<td>if verify(grade, check)&gt;0 then</td>
<td></td>
</tr>
<tr>
<td>put @1 'INVALID ' grade=;</td>
<td></td>
</tr>
<tr>
<td>datalines;</td>
<td></td>
</tr>
<tr>
<td>a b c b c d f a a q a b d d b</td>
<td></td>
</tr>
</tbody>
</table>

**Example 2**
The first section of code in this example runs in SAS and creates the data set, scores, with the CAS engine. The second section of code runs in CAS, accesses the scores table that is in CAS, and then creates the verify_scores table. The verify_scores table includes the observations that meet the verification criteria.

*Table 2.4  Using the VERIFY Function in SAS and CAS*

<table>
<thead>
<tr>
<th>Statements</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>data casuser.scores;</td>
<td>INVALID Grade=q</td>
</tr>
<tr>
<td>input Grade : $1. @@;</td>
<td></td>
</tr>
<tr>
<td>datalines;</td>
<td></td>
</tr>
<tr>
<td>a b c b c d f a a q a b d d b</td>
<td></td>
</tr>
</tbody>
</table>

| data casuser.verify_scores;| INVALID Grade=q |
|   set casuser.scores;     |                 |
|   check='abcdef';          |                 |
|   if verify(grade, check)>0 then |
|     put @1 'INVALID ' grade=; |
| run;                       |                 |
VFOMAT Function

Returns the format that is associated with the specified variable.

**Category:** Variable Information

**Restriction:** Use only with the DATA step

### Syntax

\[
\text{VFORMAT}(\text{variable})
\]

### Required Argument

**variable**

specifies a variable that is expressed as a scalar or as an array reference.

**Restriction** You cannot use an expression as an argument.

### Details

If the VFORMAT function returns a value to a variable that has not yet been assigned a length, by default the variable is assigned a length of 200.

VFOMAT returns the complete format name, which includes the width and the period (for example, $CHAR20.)

### Comparisons

- VFORMAT returns the format that is associated with the specified variable. VFORMATX, however, evaluates the argument to determine the variable name. The function then returns the format that is associated with that variable name.

- VFORMAT does not accept an expression as an argument. VFORMATX accepts expressions, but the value of the specified expression cannot denote an array reference.

- Related functions return the value of other variable attributes, such as the variable name, type, length, among others. For a list, see the Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701.
Example

The following SAS statements produce this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>array x[3] x1-x3;</td>
<td>y=BEST6.</td>
</tr>
<tr>
<td>format x1 best6.;</td>
<td></td>
</tr>
<tr>
<td>y=vformat(x[1]);</td>
<td></td>
</tr>
<tr>
<td>put y=;</td>
<td></td>
</tr>
</tbody>
</table>

See Also

Functions:

- Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701

VFORMATD Function

Returns the decimal value of the format that is associated with the specified variable.

**Category:** Variable Information

**Syntax**

VFORMATD(\textit{variable})

**Required Argument**

\textit{variable}

 specifies a variable that is expressed as a scalar or as an array reference.

**Restriction**

You cannot use an expression as an argument.

**Comparisons**

- VFORMATD returns the format decimal value that is associated with the specified variable. VFORMATDX, however, evaluates the argument to determine the variable name. The function then returns the format decimal value that is associated with that variable name.
- VFORMATD does not accept an expression as an argument. VFORMATDX accepts expressions, but the value of the specified expression cannot denote an array reference.
- Related functions return the value of other variable attributes, such as the variable name, type, and length, among others. For a list, see the Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701.
Example

The following SAS statements produce this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>array x(3) x1-x3;</td>
<td></td>
</tr>
<tr>
<td>format x1 comma8.2;</td>
<td></td>
</tr>
<tr>
<td>y=vformatd(x(1));</td>
<td></td>
</tr>
<tr>
<td>put y=;</td>
<td>y=2</td>
</tr>
</tbody>
</table>

See Also

Functions:

- Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701

VFORMATDX Function

Returns the decimal value of the format that is associated with the value of the specified argument.

Category: Variable Information

Syntax

`VFORMATDX(expression)`

Required Argument

expression

specifies a SAS character constant, variable, or expression that evaluates to a variable name.

Restriction

The value of the specified expression cannot denote an array reference.

Details

- VFORMATD returns the format decimal value that is associated with the specified variable. VFORMATDX, however, evaluates the argument to determine the variable name. The function then returns the format decimal value that is associated with that variable name.
- VFORMATD does not accept an expression as an argument. VFORMATDX accepts expressions, but the value of the specified expression cannot denote an array reference.
- Related functions return the value of other variable attributes, such as the variable name, length, type, among others. For a list, see the Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701.
Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>array x(3) x1-x3;</td>
<td>y=2</td>
</tr>
<tr>
<td>format x1 comma8.2;</td>
<td>z=2</td>
</tr>
<tr>
<td>array vx(3) $6 vx1 vx2 vx3</td>
<td></td>
</tr>
<tr>
<td>('x1' 'x2' 'x3');</td>
<td></td>
</tr>
<tr>
<td>y=vformatdx(vx{1});</td>
<td></td>
</tr>
<tr>
<td>z=vformatdx('x'</td>
<td></td>
</tr>
<tr>
<td>put y=;</td>
<td></td>
</tr>
<tr>
<td>put z=;</td>
<td></td>
</tr>
</tbody>
</table>

See Also

Functions:
- Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701

VFORMATN Function

Returns the format name that is associated with the specified variable.

Category: Variable Information

Syntax

VFORMATN(\textit{variable})

Required Argument

\textit{variable} specifies a variable that is expressed as a scalar or as an array reference.

Restriction You cannot use an expression as an argument.

Details

If the VFORMATN function returns a value to a variable that has not yet been assigned a length, by default the variable is assigned a length of 200.

VFORMATN returns only the format name, which does not include the width or the period (for example, $CHAR$).

Comparisons

- VFORMATN returns the format name that is associated with the specified variable. VFORMATNX, however, evaluates the argument to determine the variable name. The function then returns the format name that is associated with that variable name.
• VFORMATN does not accept an expression as an argument. VFORMATNX accepts expressions, but the value of the specified expression cannot denote an array reference.

• Related functions return the value of other variable attributes, such as the variable name, type, and length, among others. For a list, see the Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701.

Example
The following SAS statements produce this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>array x(3) x1-x3;</td>
<td></td>
</tr>
<tr>
<td>format x1 best6.;</td>
<td></td>
</tr>
<tr>
<td>y=vformatn(x(1));</td>
<td></td>
</tr>
<tr>
<td>put y=;</td>
<td></td>
</tr>
<tr>
<td>y=BEST</td>
<td></td>
</tr>
</tbody>
</table>

See Also

Functions:
• Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701

VFORMATNX Function

Returns the format name that is associated with the value of the specified argument.

Category: Variable Information

Syntax

VFORMATNX(expression)

Required Argument

expression
• specifies a character constant, variable, or expression that evaluates to a variable name.

Restriction The value of the specified expression cannot denote an array reference.

Details

If the VFORMATNX function returns a value to a variable that has not yet been assigned a length, by default the variable is assigned a length of 200.

VFORMATNX returns only the format name, which does not include the length or the period (for example, $CHAR).
Comparisons

- **VFORMATN** returns the format name that is associated with the specified variable. **VFORMATNX**, however, evaluates the argument to determine the variable name. The function then returns the format name that is associated with that variable name.

- **VFORMATN** does not accept an expression as an argument. **VFORMATNX** accepts expressions, but the value of the specified expression cannot denote an array reference.

- Related functions return the value of other variable attributes, such as the variable name, length, and type, among others. For a list, see the Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701.

Example

The following SAS statements produce this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>array x(3) x1-x3; format x1 best6.; array vx(3) $6 vx1 vx2 vx3 ('x1' 'x2' 'x3'); y=vformatnx(vx[1]); put y=;</td>
<td>y=BEST</td>
</tr>
</tbody>
</table>

See Also

Functions:

- Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701

VFORMATW Function

Returns the format width that is associated with the specified variable.

**Category:** Variable Information

**Syntax**

**VFORMATW**(variable)

**Required Argument**

- **variable** specifies a variable that is expressed as a scalar or as an array reference.

  **Restriction** You cannot use an expression as an argument.
Comparisons

- **VFORMATW** returns the format width that is associated with the specified variable. **VFORMATWX**, however, evaluates the argument to determine the variable name. The function then returns the format width that is associated with that variable name.

- **VFORMATW** does not accept an expression as an argument. **VFORMATWX** accepts expressions, but the value of the specified expression cannot denote an array reference.

- Related functions return the value of other variable attributes, such as the variable name, type, and length, among others. For a list, see the Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701.

Example

The following SAS statements produce this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>array x(3) x1-x3; format x1 best6.; y=vformatw(x(1)); put y=;</td>
<td>y=6</td>
</tr>
</tbody>
</table>

See Also

**Functions:**

- Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701

**VFORMATWX Function**

Returns the format width that is associated with the value of the specified argument.

- **Category:** Variable Information

**Syntax**

`VFORMATWX(expression)`

**Required Argument**

- **expression** specifies a character constant, variable, or expression that evaluates to a variable name.

**Restriction** The value of the specified expression cannot denote an array reference.
Comparisons

- VFORMATW returns the format width that is associated with the specified variable. VFORMATWX, however, evaluates the argument to determine the variable name. The function then returns the format width that is associated with that variable name.
- VFORMATW does not accept an expression as an argument. VFORMATWX accepts expressions, but the value of the specified expression cannot denote an array reference.
- Related functions return the value of other variable attributes, such as the variable name, length, and type, among others. For a list, see the Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701.

Example

The following SAS statements produce this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>array x(3) x1-x3;</td>
<td></td>
</tr>
<tr>
<td>format x1 best6.;</td>
<td>y=6</td>
</tr>
<tr>
<td>array vx(3) $6 vx1 vx2 vx3</td>
<td></td>
</tr>
<tr>
<td>('x1' 'x2' 'x3');</td>
<td></td>
</tr>
<tr>
<td>y=vformatwx(vx{1});</td>
<td></td>
</tr>
<tr>
<td>put y=;</td>
<td></td>
</tr>
</tbody>
</table>

See Also

Functions:
- Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701

VFORMATX Function

Returns the format that is associated with the value of the specified argument.

Category: Variable Information

Syntax

VFORMATX(expression)

Required Argument

expression

specifies a character constant, variable, or expression that evaluates to a variable name.

Restriction

The value of the specified expression cannot denote an array reference.
Details

If the VFORMATX function returns a value to a variable that has not yet been assigned a
length, by default the variable is assigned a length of 200.

VFORMATX returns the complete format name which includes the width and the period
(for example, $CHAR20.).

Comparisons

• VFORMAT returns the format that is associated with the specified variable.
  VFORMATX, however, evaluates the argument to determine the variable name. The
  function then returns the format that is associated with that variable name.

• VFORMAT does not accept an expression as an argument. VFORMATX accepts
  expressions, but the value of the specified expression cannot denote an array
  reference.

• Related functions return the value of other variable attributes, such as the variable
  name, length, and type, among others. For a list, see the Variable Information
  functions in “SAS Functions and CALL Routines by Category” on page 701.

Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>array x(3) x1-x3;</td>
<td>y=BEST6.</td>
</tr>
<tr>
<td>format x1 best6.;</td>
<td>z=F20.10</td>
</tr>
<tr>
<td>format x2 20.10;</td>
<td></td>
</tr>
<tr>
<td>array vx(3) $6 vx1 vx2 vx3</td>
<td></td>
</tr>
<tr>
<td>('x1' 'x2' 'x3');</td>
<td></td>
</tr>
<tr>
<td>y=vformatx(vx(1));</td>
<td></td>
</tr>
<tr>
<td>z=vformatx(vx(2));</td>
<td></td>
</tr>
<tr>
<td>put y=;</td>
<td></td>
</tr>
<tr>
<td>put z=;</td>
<td></td>
</tr>
</tbody>
</table>

See Also

Functions:

• Variable Information functions in “SAS Functions and CALL Routines by Category”
on page 701

VINARRAY Function

Returns a value that indicates whether the specified variable is a member of an array.

Category: Variable Information

Restriction: Use only with the DATA step
Syntax

VINARRAY(variable)

Required Argument

variable

specifies a variable that is expressed as a scalar or as an array reference.

Restriction

You cannot use an expression as an argument.

Details

VINARRAY returns 1 if the given variable is a member of an array; it returns 0 if the given variable is not a member of an array.

Comparisons

- VINARRAY returns a value that indicates whether the specified variable is a member of an array. VINARRAYX, however, evaluates the argument to determine the variable name. The function then returns a value that indicates whether the variable name is a member of an array.
- VINARRAY does not accept an expression as an argument. VINARRAYX accepts expressions, but the value of the specified expression cannot denote an array reference.
- Related functions return the value of other variable attributes, such as the variable name, informat, and format, among others. For a list, see the Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701.

Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>array x[3] x1-x3;</td>
<td></td>
</tr>
<tr>
<td>y=vinarray(x);</td>
<td>y=0</td>
</tr>
<tr>
<td>Z=vinarray(x1);</td>
<td>z=1</td>
</tr>
<tr>
<td>put y=;</td>
<td></td>
</tr>
<tr>
<td>put Z=;</td>
<td></td>
</tr>
</tbody>
</table>

See Also

Functions:

- Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701

VINARRAYX Function

Returns a value that indicates whether the value of the specified argument is a member of an array.
**Syntax**

VINARRAYX(expression)

**Required Argument**

expression

 specifies a character constant, variable, or expression that evaluates to a variable name.

**Restriction**

The value of the specified expression cannot denote an array reference.

**Details**

VINARRAYX returns 1 if the value of the given argument is a member of an array; it returns 0 if the value of the given argument is not a member of an array.

**Comparisons**

- VINARRAY returns a value that indicates whether the specified variable is a member of an array. VINARRAYX, however, evaluates the argument to determine the variable name. The function then returns a value that indicates whether the variable name is a member of an array.

- VINARRAY does not accept an expression as an argument. VINARRAYX accepts expressions, but the value of the specified expression cannot denote an array reference.

- Related functions return the value of other variable attributes, such as the variable name, informat, and format, among others. For a list, see the Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701.

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>array x(3) x1-x3;</td>
<td>y=0</td>
</tr>
<tr>
<td>array vx(4) $6 vx1 vx2 vx3 vx4</td>
<td>z=1</td>
</tr>
<tr>
<td>('x' 'x1' 'x2' 'x3');</td>
<td></td>
</tr>
<tr>
<td>y=vinarrayx(vx{1});</td>
<td></td>
</tr>
<tr>
<td>z=vinarrayx(vx{2});</td>
<td></td>
</tr>
<tr>
<td>put y=;</td>
<td></td>
</tr>
<tr>
<td>put z=;</td>
<td></td>
</tr>
</tbody>
</table>

**See Also**

**Functions:**

- Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701
VINFORMAT Function
Returns the informat that is associated with the specified variable.

**Syntax**

```
VINFORMAT(variable)
```

**Required Argument**

variable

specifies a variable that is expressed as a scalar or as an array reference.

**Details**

If the VINFORMAT function returns a value to a variable that has not yet been assigned a length, by default the variable is assigned a length of 200.

VINFORMAT returns the complete informat name, which includes the width and the period (for example, $CHAR20.).

**Comparisons**

- VINFORMAT returns the informat that is associated with the specified variable. VINFORMATX, however, evaluates the argument to determine the variable name. The function then returns the informat that is associated with that variable name.
- VINFORMAT does not accept an expression as an argument. VINFORMATX accepts expressions, but the value of the specified expression cannot denote an array reference.
- Related functions return the value of other variable attributes, such as the variable name, type, and length, among others. For a list, see the Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701.

**Example**

The following SAS statements produce this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>informat x $char6.;</td>
<td>y=$CHAR6.</td>
</tr>
<tr>
<td>input x;</td>
<td></td>
</tr>
<tr>
<td>y=vinformat(x);</td>
<td></td>
</tr>
<tr>
<td>put y=;</td>
<td></td>
</tr>
</tbody>
</table>
See Also

Functions:
- Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701

VINFORMATD Function

Returns the decimal value of the informat that is associated with the specified variable.

**Category:** Variable Information

**Syntax**

VINFORMATD(\textit{variable})

**Required Argument**

\textit{variable}

specifies a variable that is expressed as a scalar or as an array reference.

**Restriction** You cannot use an expression as an argument.

**Comparisons**

- VINFORMATD returns the informat decimal value that is associated with the specified variable. VINFORMATDX, however, evaluates the argument to determine the variable name. The function then returns the informat decimal value that is associated with that variable name.
- VINFORMATD does not accept an expression as an argument. VINFORMATDX accepts expressions, but the value of the specified expression cannot denote an array reference.
- Related functions return the value of other variable attributes, such as the variable name, type, and length, among others. For a list, see the Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701.

**Example**

The following SAS statements produce this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>informat x comma8.2;</code></td>
<td></td>
</tr>
<tr>
<td><code>input x;</code></td>
<td></td>
</tr>
<tr>
<td><code>y=vinformatd(x);</code></td>
<td></td>
</tr>
<tr>
<td><code>put y=;</code></td>
<td><code>y=2</code></td>
</tr>
</tbody>
</table>
See Also

Functions:

• Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701

VINFORMATDX Function

Returns the decimal value of the informat that is associated with the value of the specified variable.

Category: Variable Information

Syntax

VINFORMATDX(expression)

Required Argument

expression

specifies a character constant, variable, or expression that evaluates to a variable name.

Restriction

The value of the specified variable cannot denote an array reference.

Comparisons

• VINFORMATD returns the informat decimal value that is associated with the specified variable. VINFORMATDX, however, evaluates the argument to determine the variable name. The function then returns the informat decimal value that is associated with that variable name.

• VINFORMATD does not accept an expression as an argument. VINFORMATDX accepts expressions, but the value of the specified expression cannot denote an array reference.

• Related functions return the value of other variable attributes, such as the variable name, length, and type, among others. For a list, see the Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701.

Example

The following SAS statements produce this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>informat x1 x2 x3 comma9.3;</td>
<td></td>
</tr>
<tr>
<td>input x1 x2 x3;</td>
<td></td>
</tr>
<tr>
<td>array vx(3) $6 vx1 vx2 vx3 ('x1' 'x2' 'x3');</td>
<td></td>
</tr>
<tr>
<td>y=vinformatdx(vx[1]);</td>
<td>y=3</td>
</tr>
<tr>
<td>put y=;</td>
<td></td>
</tr>
</tbody>
</table>
See Also

Functions:

- Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701

VINFORMATN Function

Returns the informat name that is associated with the specified variable.

**Category:** Variable Information

**Syntax**

VINFORMATN(\textit{variable})

**Required Argument**

\textit{variable}

specifies a variable that is expressed as a scalar or as an array reference.

**Restriction**

You cannot use an expression as an argument.

**Details**

If the VINFORMATN function returns a value to a variable that has not yet been assigned a length, by default the variable is assigned a length of 200.

VINFORMATN returns only the informat name, which does not include the width or the period (for example, $\text{CHAR}$).

**Comparisons**

- VINFORMATN returns the informat name that is associated with the specified variable. VINFORMATNX, however, evaluates the argument to determine the variable name. The function then returns the informat name that is associated with that variable name.

- VINFORMATN does not accept an expression as an argument. VINFORMATNX accepts expressions, but the value of the specified expression cannot denote an array reference.

- Related functions return the value of other variable attributes, such as the variable name, type, and length, among others. For a list, see the Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701.

**Example**
The following SAS statements produce this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>informat x $char6.;</td>
<td></td>
</tr>
<tr>
<td>input x;</td>
<td></td>
</tr>
<tr>
<td>y=vinformatn(x);</td>
<td>y=$CHAR</td>
</tr>
<tr>
<td>put y=;</td>
<td></td>
</tr>
</tbody>
</table>

See Also

Functions:

- Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701

**VINFORMATNX Function**

Returns the informat name that is associated with the value of the specified argument.

**Category:** Variable Information

**Syntax**

VINFORMATNX(expression)

**Required Argument**

expression

specifies a character constant, variable, or expression that evaluates to a variable name.

**Restriction**

The value of the specified expression cannot denote an array reference.

**Details**

If the VINFORMATNX function returns a value to a variable that has not yet been assigned a length, by default the variable is assigned a length of 200.

VINFORMATNX returns only the informat name, which does not include the width or the period (for example, $CHAR).

**Comparisons**

- VINFORMATN returns the informat name that is associated with the specified variable. VINFORMATNX, however, evaluates the argument to determine the variable name. The function then returns the informat name that is associated with that variable name.
- VINFORMATN does not accept an expression as an argument. VINFORMATNX accepts expressions, but the value of the specified expression cannot denote an array reference.
• Related functions return the value of other variable attributes, such as the variable name, length, and type, among others. For a list, see the Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701.

Example

The following SAS statements produce this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>informat x1 x2 x3 $char6.;</td>
<td>y=$CHAR</td>
</tr>
<tr>
<td>input x1 x2 x3;</td>
<td></td>
</tr>
<tr>
<td>array vx(3) $6 vx1 vx2 vx3</td>
<td></td>
</tr>
<tr>
<td>('x1' 'x2' 'x3');</td>
<td></td>
</tr>
<tr>
<td>y=vinformatnx(vx[1]);</td>
<td></td>
</tr>
<tr>
<td>put y=;</td>
<td></td>
</tr>
</tbody>
</table>

See Also

Functions:
• Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701

VINFORMATW Function

Returns the informat width that is associated with the specified variable.

| Category: | Variable Information |

Syntax

VINFORMATW(variable)

Required Argument

variable

specifies a variable that is expressed as a scalar or as an array reference.

Restriction

You cannot use an expression as an argument.

Comparisons

• VINFORMATW returns the informat width that is associated with the specified variable. VINFORMATWX, however, evaluates the argument to determine the variable name. The function then returns the informat width that is associated with that variable name.

• VINFORMATW does not accept an expression as an argument. VINFORMATWX accepts expressions, but the value of the specified expression cannot denote an array reference.
- Related functions return the value of other variable attributes, such as the variable name, type, and length, among others. For a list, see the Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701.

**Example**

The following SAS statements produce this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>informat x $char6.; input x; y=vinformatw(x); put y=;</td>
<td>y=6</td>
</tr>
</tbody>
</table>

**See Also**

Functions:

- Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701

**VINFORMATWX Function**

Returns the informat width that is associated with the value of the specified argument.

**Category:** Variable Information

**Syntax**

VINFORMATWX(*expression*)

**Required Argument**

*expression*

specifies a character constant, variable, or expression that evaluates to a variable name.

**Restriction**

The value of the specified expression cannot denote an array reference.

**Comparisons**

- VINFORMATW returns the informat width that is associated with the specified variable. VINFORMATWX, however, evaluates the argument to determine the variable name. The function then returns the informat width that is associated with that variable name.
- VINFORMATW does not accept an expression as an argument. VINFORMATWX accepts expressions, but the value of the specified expression cannot denote an array reference.
• Related functions return the value of other variable attributes, such as the variable name, length, and type, among others. For a list, see the Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701.

Example

The following SAS statements produce this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>informat x1 x2 x3 $char6.; input x1 x2 x3; array vx(3) $6 vx1 vx2 vx3 ('x1' 'x2' 'x3'); y=vinformatwx(vx[1]); put y=;</td>
<td>y=6</td>
</tr>
</tbody>
</table>

See Also

Functions:
• Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701

VINFORMATX Function

Returns the informat that is associated with the value of the specified argument.

Category: Variable Information

Syntax

VINFORMATX(expression)

Required Argument

expression

specifies a character constant, variable, or expression that evaluates to a variable name.

Restriction

The value of the specified expression cannot denote an array reference.

Details

If the VINFORMATX function returns a value to a variable that has not yet been assigned a length, by default the variable is assigned a length of 200.

VINFORMATX returns the complete informat name, which includes the width and the period (for example, $CHAR20.).
Comparisons

• VINFORMAT returns the informat that is associated with the specified variable. VINFORMATX, however, evaluates the argument to determine the variable name. The function then returns the informat that is associated with that variable name.

• VINFORMAT does not accept an expression as an argument. VINFORMATX accepts expressions, but the value of the specified expression cannot denote an array reference.

• Related functions return the value of other variable attributes, such as the variable name, length, and type, among others. For a list, see the Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701.

Example

The following SAS statements produce this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>informat x1 x2 x3 $char6.;</td>
<td>y=$CHAR6.</td>
</tr>
<tr>
<td>input x1 x2 x3;</td>
<td></td>
</tr>
<tr>
<td>array vx(3) $6 vx1 vx2 vx3</td>
<td></td>
</tr>
<tr>
<td>('x1' 'x2' 'x3');</td>
<td></td>
</tr>
<tr>
<td>y=vinformatx(vx(1));</td>
<td></td>
</tr>
<tr>
<td>put y=;</td>
<td></td>
</tr>
</tbody>
</table>

See Also

Functions:

• Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701

VLABEL Function

Returns the label that is associated with the specified variable.

Category: Variable Information

Restriction: Use only with the DATA step

Syntax

VLABEL(variable)

Required Argument

variable

specifies a variable that is expressed as a scalar or as an array reference.

Restriction You cannot use an expression as an argument.
Details

If the VLABEL function returns a value to a variable that has not yet been assigned a length, by default the variable is assigned a length of 200. If there is no label, VLABEL returns the variable name.

Comparisons

- VLABEL returns the label of the specified variable or the name of the specified variable, if no label exists. VLABELX, however, evaluates the argument to determine the variable name. The function then returns the label that is associated with that variable name, or the variable name if no label exists.
- VLABEL does not accept an expression as an argument. VLABELX accepts expressions, but the value of the specified expression cannot denote an array reference.
- VLABEL has the same functionality as CALL LABEL.
- Related functions return the value of other variable attributes, such as the variable name, informat, and format, among others. For a list, see the Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701.

Example

The following SAS statements produce this result:

```
array x(3) x1-x3;
label x1='Test1';
y=vlabel(x(1));
put y=;
```

See Also

Functions:
- Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701

VLABELX Function

Returns the label that is associated with the value of the specified argument.

**Category:** Variable Information

**Syntax**

```
VLABELX(expression)
```
**Required Argument**

*expression*

specifies a character constant, variable, or expression that evaluates to a variable name.

**Restriction**
The value of the specified expression cannot denote an array reference.

**Details**

If the VLABELX function returns a value to a variable that has not yet been assigned a length, by default the variable is assigned a length of 200.

If there is no label, VLABELX returns the variable name.

**Comparisons**

- VLABEL returns the label of the specified variable, or the name of the specified variable if no label exists. VLABELX, however, evaluates the argument to determine the variable name. The function then returns the label that is associated with that variable name, or the variable name if no label exists.
- VLABEL does not accept an expression as an argument. VLABELX accepts expressions, but the value of the specified expression cannot denote an array reference.
- Related functions return the value of other variable attributes, such as the variable name, informat, and format, among others. For a list, see the Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701.

**Example**

The following SAS statements produce this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>array x(3) x1-x3;</td>
<td></td>
</tr>
<tr>
<td>array vx(3) $6 vx1 vx2 vx3</td>
<td></td>
</tr>
<tr>
<td>('x1' 'x2' 'x3');</td>
<td></td>
</tr>
<tr>
<td>label x1='Test1';</td>
<td></td>
</tr>
<tr>
<td>y=vlabelx(vx(1));</td>
<td></td>
</tr>
<tr>
<td>put y=;</td>
<td>y='Test1'</td>
</tr>
</tbody>
</table>

**See Also**

Functions:

- Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701

**VLENGTH Function**

Returns the compile-time (allocated) size of the specified variable.

**Category:** Variable Information
Restriction: Use only with the DATA step

Syntax

VLENGTH(variable)

Required Argument

variable

specifies a variable that is expressed as a scalar or as an array reference.

Restriction You cannot use an expression as an argument.

Comparisons

• LENGTH examines the variable at run time, trimming trailing blanks to determine the length. VLENGTH returns a compile-time constant value, which reflects the maximum length.

• LENGTHC returns the same value as VLENGTH, but LENGTHC can be used in any calling environment and its argument can be any expression.

• VLENGTH returns the length of the specified variable. VLENGTHX, however, evaluates the argument to determine the variable name. The function then returns the compile-time size that is associated with that variable name.

• VLENGTH does not accept an expression as an argument. VLENGTHX accepts expressions, but the value of the specified expression cannot denote an array reference.

• Related functions return the value of other variable attributes, such as the variable name, informat, and format, among others. For a list, see the Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701.

Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>length x $8;</td>
<td></td>
</tr>
<tr>
<td>x='abc';</td>
<td></td>
</tr>
<tr>
<td>y=vlength(x);</td>
<td>y=8</td>
</tr>
<tr>
<td>z=length(x);</td>
<td>z=3</td>
</tr>
<tr>
<td>put y=;</td>
<td></td>
</tr>
<tr>
<td>put z=;</td>
<td></td>
</tr>
</tbody>
</table>

See Also

Functions:

• Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701
VLENGTHX Function

Returns the compile-time (allocated) size for the variable that has a name that is the same as the value of the argument.

Category: Variable Information

Syntax

VLENGTHX(expression)

Required Argument

expression

specifies a character constant, variable, or expression that evaluates to a variable name.

Restriction  The value of the specified expression cannot denote an array reference.

Comparisons

• LENGTH examines the variable at run time, trimming trailing blanks to determine the length. VLENGTHX, however, evaluates the argument to determine the variable name. The function then returns the compile-time size that is associated with that variable name.

• LENGTHC accepts an expression as the argument, but it returns the length of the value of the expression, not the length of the variable that has a name equal to the value of the expression.

• VLENGTH returns the length of the specified variable. VLENGTHX returns the length for the value of the specified expression.

• VLENGTH does not accept an expression as an argument. VLENGTHX accepts expressions, but the value of the specified expression cannot denote an array reference.

• Related functions return the value of other variable attributes, such as the variable name, informat, format, among others. For a list, see the Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701.
Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>length x1 $8;</td>
<td>y=8</td>
</tr>
<tr>
<td>x1='abc';</td>
<td></td>
</tr>
<tr>
<td>array vx(3) $6</td>
<td>z=3</td>
</tr>
<tr>
<td>vx1 vx2 vx3</td>
<td></td>
</tr>
<tr>
<td>('x1' 'x2' 'x3');</td>
<td></td>
</tr>
<tr>
<td>y=vlengthx(vx(1));</td>
<td></td>
</tr>
<tr>
<td>z=length(x1);</td>
<td></td>
</tr>
<tr>
<td>put y=;</td>
<td></td>
</tr>
<tr>
<td>put z=;</td>
<td></td>
</tr>
</tbody>
</table>

See Also

Functions:

- Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701

VNAME Function

Returns the name of the specified variable.

Category: Variable Information
Restriction: Use only with the DATA step

Syntax

VNAME(variable)

Required Argument

variable

specifies a variable that is expressed as a scalar or as an array reference.

Restriction You cannot use an expression as an argument.

Details

If the VNAME function returns a value to a variable that has not yet been assigned a length, by default the variable is assigned a length of 200.

Comparisons

- VNAME returns the name of the specified variable. VNAMEX, however, evaluates the argument to determine a variable name. If the name is a known variable name, the function returns that name. Otherwise, the function returns a blank.
• VNAME does not accept an expression as an argument. VNAMEX accepts expressions, but the value of the specified expression cannot denote an array reference.

• VNAME has the same functionality as CALL VNAME.

• Related functions return the value of other variable attributes, such as the variable label, informat, and format, among others. For a list, see the Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701.

Example

The following SAS statements produce this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>array x(3) x1-x3; y=vname(x1);</td>
<td>y=x1</td>
</tr>
<tr>
<td>put y=;</td>
<td></td>
</tr>
</tbody>
</table>

See Also

Functions:

• Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701

VNAMEX Function

Validates the value of the specified argument as a variable name.

Category: Variable Information

Syntax

VNAMEX(expression)

Required Argument

expression

specifies a character constant, variable, or expression.

Restriction

The value of the specified expression cannot denote an array reference.

Details

If the VNAMEX function returns a value to a variable that has not yet been assigned a length, by default the variable is assigned a length of 200.
Comparisons

- VNAME returns the name of the specified variable. VNAMEX, however, evaluates the argument to determine a variable name. If the name is a known variable name, the function returns that name. Otherwise, the function returns a blank.

- VNAME does not accept an expression as an argument. VNAMEX accepts expressions, but the value of the specified variable cannot denote an array reference.

- Related functions return the value of other variable attributes, such as the variable label, informat, and format, among others. For a list, see the Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701.

Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>array x(3) x1-x3;</td>
<td>y=x1</td>
</tr>
<tr>
<td>array vx(3) $6 vx1 vx2 vx3 ('x1' 'x2' 'x3');</td>
<td>z=x1</td>
</tr>
<tr>
<td>y=vnamex(vx(1));</td>
<td></td>
</tr>
<tr>
<td>z=vnamex('x'</td>
<td></td>
</tr>
<tr>
<td>put y=;</td>
<td></td>
</tr>
<tr>
<td>put z=;</td>
<td></td>
</tr>
</tbody>
</table>

See Also

Functions:

- Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701

VTYPE Function

Returns the type (character or numeric) of the specified variable.

**Category:** Variable Information

**Restriction:** Use only with the DATA step

**Syntax**

VTYPE(\textit{variable})

**Required Argument**

\textit{variable}

specifies a variable that is expressed as a scalar or as an array reference.

**Restriction** You cannot use an expression as an argument.
Details
If the VTYPE function returns a value to a variable that has not yet been assigned a length, by default the variable is assigned a length of 1.

VTYPE returns N for numeric variables and C for character variables.

Comparisons
• VTYPE returns the type of the specified variable. VTYPEX, however, evaluates the argument to determine the variable name. The function then returns the type (character or numeric) that is associated with that variable name.
• VTYPE does not accept an expression as an argument. VTYPEX accepts expressions, but the value of the specified expression cannot denote an array reference.
• Related functions return the value of other variable attributes, such as the variable name, informat, and format, among others. For a list, see the Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701.

Example
The following SAS statements produce this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>array x(3) x1-x3;</td>
<td></td>
</tr>
<tr>
<td>y=vtype(x(1));</td>
<td>y=N</td>
</tr>
<tr>
<td>put y=;</td>
<td></td>
</tr>
</tbody>
</table>

See Also

Functions:
• Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701

VTYPEX Function
Returns the type (character or numeric) for the value of the specified argument.

Category: Variable Information

Syntax
VTYPEX(expression)

Required Argument
expression
specifies a character constant, variable, or expression that evaluates to a variable name.
Restriction

The value of the specified expression cannot denote an array reference.

Details

If the VTYPEEX function returns a value to a variable that has not yet been assigned a length, by default the variable is assigned a length of 1.

VTYPEEX returns N for numeric variables and C for character variables.

Comparisons

• VTYPE returns the type of the specified variable. VTYPEEX, however, evaluates the argument to determine the variable name. The function then returns the type (character or numeric) that is associated with that variable name.

• VTYPE does not accept an expression as an argument. VTYPEEX accepts expressions, but the value of the specified expression cannot denote an array reference.

• Related functions return the value of other variable attributes, such as the variable name, informat, and format, among others. For a list, see the Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701.

Example

The following SAS statements produce this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>array x(3) x1-x3;</td>
<td>y=N</td>
</tr>
<tr>
<td>array vx(3) $6 vx1 vx2 vx3</td>
<td></td>
</tr>
<tr>
<td>('x1' 'x2' 'x3');</td>
<td></td>
</tr>
<tr>
<td>y=vtype(vx(1));</td>
<td></td>
</tr>
<tr>
<td>put y=;</td>
<td></td>
</tr>
</tbody>
</table>

See Also

Functions:

• Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701

VVALUE Function

Returns the formatted value that is associated with the variable that you specify.

Category: Variable Information

Restriction: Use only with the DATA step

Syntax

VVALUE(variable)
**Required Argument**

*variable*

specifies a variable that is expressed as a scalar or as an array reference.

**Restriction**

You cannot use an expression as an argument.

**Details**

If the VVALUE function returns a value to a variable that has not yet been assigned a length, by default the variable is assigned a length of 200.

VVALUE returns a character string that contains the current value of the variable that you specify. The value is formatted using the current format that is associated with the variable.

**Comparisons**

- VVALUE returns the value that is associated with the variable that you specify. VVALUEX, however, evaluates the argument to determine the variable name. The function then returns the value that is associated with that variable name.
- VVALUE does not accept an expression as an argument. VVALUEX accepts expressions, but the value of the expression cannot denote an array reference.
- VVALUE and an assignment statement both return a character string that contains the current value of the variable that you specify. With VVALUE, the value is formatted using the current format that is associated with the variable. With an assignment statement, however, the value is unformatted.
- The PUT function enables you to reformat a specified variable or constant. VVALUE uses the current format that is associated with the variable.

**Example**

The following SAS statements produce this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>y=9999;</td>
<td>9,999.00</td>
</tr>
<tr>
<td>format y comma10.2;</td>
<td></td>
</tr>
<tr>
<td>v=vvalue(y);</td>
<td></td>
</tr>
<tr>
<td>put v;</td>
<td></td>
</tr>
</tbody>
</table>

**See Also**

**Functions:**

- “VVALUEX Function” on page 681
- Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701
**VVALUEX Function**

Returns the formatted value that is associated with the argument that you specify.

**Category:** Variable Information

**Syntax**

VVALUEX(expression)

**Required Argument**

*expression*

specifies a character constant, variable, or expression that evaluates to a variable name.

**Restriction**

The value of the specified expression cannot denote an array reference.

**Details**

If the VVALUEX function returns a value to a variable that has not yet been assigned a length, by default the variable is assigned a length of 200.

VVALUEX returns a character string that contains the current value of the argument that you specify. The value is formatted by using the format that is currently associated with the argument.

**Comparisons**

- VVALUE accepts a variable as an argument and returns the value of that variable. VVALUEX, however, accepts a character expression as an argument. The function then evaluates the expression to determine the variable name and returns the value that is associated with that variable name.

- VVALUE does not accept an expression as an argument, but it does accept array references. VVALUEX accepts expressions, but the value of the expression cannot denote an array reference.

- VVALUEX and an assignment statement both return a character string that contains the current value of the variable that you specify. With VVALUEX, the value is formatted by using the current format that is associated with the variable. With an assignment statement, however, the value is unformatted.

- The PUT function enables you to reformat a specified variable or constant. VVALUEX uses the current format that is associated with the variable.
Example

The following SAS statements produce this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>date1='31mar02'd;</td>
<td>31MAR02</td>
</tr>
<tr>
<td>date2='date1';</td>
<td></td>
</tr>
<tr>
<td>format date1 date7.;</td>
<td></td>
</tr>
<tr>
<td>datevalue=vvaluex(date2);</td>
<td></td>
</tr>
<tr>
<td>put datevalue;</td>
<td></td>
</tr>
</tbody>
</table>

See Also

Functions:

- “VVALUE Function” on page 679
- Variable Information functions in “SAS Functions and CALL Routines by Category” on page 701

WEEK Function

Returns the week-number value.

Category: Date and Time

Syntax

WEEK(<sas-date>, <'descriptor'>)

Optional Arguments

sas-date

specifies the SAS data value. If the sas-date argument is not specified, the WEEK function returns the week-number value of the current date.

descriptor

specifies the value of the descriptor. The following descriptors can be specified in uppercase or lowercase characters.

U

specifies the number-of-the-week within the year. Sunday is considered the first day of the week. The number-of-the-week value is represented as a decimal number in the range 0–53. Week 53 has no special meaning. The value of week('31dec2006'd, 'u') is 53. U is the default value.

Tip  The U and W descriptors are similar, except that the U descriptor considers Sunday as the first day of the week, and the W descriptor considers Monday as the first day of the week.

See  “The U Descriptor” on page 683
specifies the number-of-the-week whose value is represented as a decimal number in the range 1–53. Monday is considered the first day of the week and week 1 of the year is the week that includes both January 4 and the first Thursday of the year. If the first Monday of January is the 2nd, 3rd, or 4th, the preceding days are part of the last week of the preceding year.

See “The V Descriptor” on page 683

specifies the number-of-the-week within the year. Monday is considered the first day of the week. The number-of-the-week value is represented as a decimal number in the range 0–53. Week 53 has no special meaning. The value of \( \text{week('31dec2006'd, 'w')} \) is 52.

Tip The U and W descriptors are similar except that the U descriptor considers Sunday as the first day of the week, and the W descriptor considers Monday as the first day of the week.

See “The W Descriptor” on page 684

Details

The Basics
The WEEK function reads a SAS date value and returns the week number. The WEEK function is not dependent on locale, and uses only the Gregorian calendar in its computations.

The U Descriptor
The WEEK function with the U descriptor reads a SAS date value and returns the number of the week within the year. The number-of-the-week value is represented as a decimal number in the range 0–53, with a leading zero and maximum value of 53. Week 0 means that the first day of the week occurs in the preceding year. The fifth week of the year is represented as 05.

Sunday is considered the first day of the week. For example, the value of \( \text{week('01jan2007'd, 'u')} \) is 0.

The V Descriptor
The WEEK function with the V descriptor reads a SAS date value and returns the week number. The number-of-the-week is represented as a decimal number in the range 01–53. The decimal number has a leading zero and a maximum value of 53. Weeks begin on a Monday, and week 1 of the year is the week that includes both January 4 and the first Thursday of the year. If the first Monday of January is the 2nd, 3rd, or 4th, the preceding days are part of the last week of the preceding year. In the following example, 01jan2006 and 30dec2005 occur in the same week. The first day (Monday) of that week is 26dec2005. Therefore, \( \text{week('01jan2006'd, 'v')} \) and \( \text{week('30dec2005'd, 'v')} \) both return a value of 52. This means that both dates occur in week 52 of the year 2005.
The W Descriptor
The WEEK function with the W descriptor reads a SAS date value and returns the number of the week within the year. The number-of-the-week value is represented as a decimal number in the range 0–53, with a leading zero and maximum value of 53. Week 0 means that the first day of the week occurs in the preceding year. The fifth week of the year would be represented as 05.

Monday is considered the first day of the week. Therefore, the value of
\texttt{week('01jan2007'd, 'w')} is 1.

Comparisons of Descriptors
U is the default descriptor. Its range is 0-53, and the first day of the week is Sunday. The V descriptor has a range of 1-53 and the first day of the week is Monday. The W descriptor has a range of 0-53 and the first day of the week is Monday.

The following list describes the descriptors and an associated week:

- Week 0:
  - U indicates the days in the current Gregorian year before week 1.
  - V does not apply.
  - W indicates the days in the current Gregorian year before week 1.

- Week 1:
  - U begins on the first Sunday in a Gregorian year.
  - V begins on the Monday between December 29 of the previous Gregorian year and January 4 of the current Gregorian year. The first ISO week can span the previous and current Gregorian years.
  - W begins on the first Monday in a Gregorian year.

- End of Year Weeks:
  - U specifies that the last week (52 or 53) in the year can contain less than 7 days. A Sunday to Saturday period that spans 2 consecutive Gregorian years is designated as 52 and 0 or 53 and 0.
  - V specifies that the last week (52 or 53) of the ISO year contains 7 days. However, the last week of the ISO year can span the current Gregorian and next Gregorian year.
  - W specifies that the last week (52 or 53) in the year can contain less than 7 days. A Monday to Sunday period that spans two consecutive Gregorian years is designated as 52 and 0 or 53 and 0.

Example
The following example shows the values of the U, V, and W descriptors for dates near the end of certain years and the beginning of the next year. Examining the full data set illustrates how the behavior can differ between the various descriptors depending on the day of the week for January 1. The output displays the first 20 observations:

\begin{verbatim}
title 'Values of the U, V, and W Descriptors';
data a(drop=i date0 date1 y);
date0='20dec2005'd;
do y=0 to 5;
   date1=intnx("YEAR", date0, y, 's');
\end{verbatim}
do i=0 to 20;
    date=intnx("DAY", date1, i);
    year=YEAR(date);
    week=week(date);
    week_u=week(date, 'u');
    week_v=week(date, 'v');
    week_w=week(date, 'w');
    output;
end;
end;
format date WEEKDATX17.;
run;
proc print;
run;
# Values of the U, V, and W Descriptors

<table>
<thead>
<tr>
<th>Obs</th>
<th>date</th>
<th>year</th>
<th>week</th>
<th>week_u</th>
<th>week_v</th>
<th>week_w</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tue, 20 Dec 2005</td>
<td>2005</td>
<td>51</td>
<td>51</td>
<td>51</td>
<td>51</td>
</tr>
<tr>
<td>2</td>
<td>Wed, 21 Dec 2005</td>
<td>2005</td>
<td>51</td>
<td>51</td>
<td>51</td>
<td>51</td>
</tr>
<tr>
<td>3</td>
<td>Thu, 22 Dec 2005</td>
<td>2005</td>
<td>51</td>
<td>51</td>
<td>51</td>
<td>51</td>
</tr>
<tr>
<td>4</td>
<td>Fri, 23 Dec 2005</td>
<td>2005</td>
<td>51</td>
<td>51</td>
<td>51</td>
<td>51</td>
</tr>
<tr>
<td>5</td>
<td>Sat, 24 Dec 2005</td>
<td>2005</td>
<td>51</td>
<td>51</td>
<td>51</td>
<td>51</td>
</tr>
<tr>
<td>6</td>
<td>Sun, 25 Dec 2005</td>
<td>2005</td>
<td>52</td>
<td>52</td>
<td>51</td>
<td>51</td>
</tr>
<tr>
<td>7</td>
<td>Mon, 26 Dec 2005</td>
<td>2005</td>
<td>52</td>
<td>52</td>
<td>52</td>
<td>52</td>
</tr>
<tr>
<td>8</td>
<td>Tue, 27 Dec 2005</td>
<td>2005</td>
<td>52</td>
<td>52</td>
<td>52</td>
<td>52</td>
</tr>
<tr>
<td>9</td>
<td>Wed, 28 Dec 2005</td>
<td>2005</td>
<td>52</td>
<td>52</td>
<td>52</td>
<td>52</td>
</tr>
<tr>
<td>10</td>
<td>Thu, 29 Dec 2005</td>
<td>2005</td>
<td>52</td>
<td>52</td>
<td>52</td>
<td>52</td>
</tr>
<tr>
<td>11</td>
<td>Fri, 30 Dec 2005</td>
<td>2005</td>
<td>52</td>
<td>52</td>
<td>52</td>
<td>52</td>
</tr>
<tr>
<td>12</td>
<td>Sat, 31 Dec 2005</td>
<td>2005</td>
<td>52</td>
<td>52</td>
<td>52</td>
<td>52</td>
</tr>
<tr>
<td>13</td>
<td>Sun, 1 Jan 2006</td>
<td>2006</td>
<td>1</td>
<td>1</td>
<td>52</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>Mon, 2 Jan 2006</td>
<td>2006</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>Tue, 3 Jan 2006</td>
<td>2006</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>Wed, 4 Jan 2006</td>
<td>2006</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>Thu, 5 Jan 2006</td>
<td>2006</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>18</td>
<td>Fri, 6 Jan 2006</td>
<td>2006</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>Sat, 7 Jan 2006</td>
<td>2006</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>Sun, 8 Jan 2006</td>
<td>2006</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>21</td>
<td>Mon, 9 Jan 2006</td>
<td>2006</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>22</td>
<td>Wed, 20 Dec 2006</td>
<td>2006</td>
<td>51</td>
<td>51</td>
<td>51</td>
<td>51</td>
</tr>
<tr>
<td>23</td>
<td>Thu, 21 Dec 2006</td>
<td>2006</td>
<td>51</td>
<td>51</td>
<td>51</td>
<td>51</td>
</tr>
<tr>
<td>24</td>
<td>Fri, 22 Dec 2006</td>
<td>2006</td>
<td>51</td>
<td>51</td>
<td>51</td>
<td>51</td>
</tr>
<tr>
<td>25</td>
<td>Sat, 23 Dec 2006</td>
<td>2006</td>
<td>51</td>
<td>51</td>
<td>51</td>
<td>51</td>
</tr>
<tr>
<td>26</td>
<td>Sun, 24 Dec 2006</td>
<td>2006</td>
<td>52</td>
<td>52</td>
<td>51</td>
<td>51</td>
</tr>
<tr>
<td>27</td>
<td>Mon, 25 Dec 2006</td>
<td>2006</td>
<td>52</td>
<td>52</td>
<td>52</td>
<td>52</td>
</tr>
</tbody>
</table>
See Also

Functions:
- “INTNX Function” on page 370

Formats:
- “WEEKU Format” in SAS Viya Formats and Informats: Reference
- “WEEKV Format” in SAS Viya Formats and Informats: Reference
- “WEEKW Format” in SAS Viya Formats and Informats: Reference

Informats:
- “WEEKU Informat” in SAS Viya Formats and Informats: Reference
- “WEEKV Informat” in SAS Viya Formats and Informats: Reference
- “WEEKW Informat” in SAS Viya Formats and Informats: Reference

---

WEEKDAY Function
From a SAS date value, returns an integer that corresponds to the day of the week.

**Category:** Date and Time

**Syntax**

```
WEEKDAY(date)
```

**Required Argument**

*date*

specifies a SAS expression that represents a SAS date value.

**Details**

The WEEKDAY function produces an integer that represents the day of the week, where 1=Sunday, 2=Monday, ..., 7=Saturday.

**Example**

The following SAS statements produce this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=weekday('16mar97'd);</td>
<td>1</td>
</tr>
<tr>
<td>put x;</td>
<td>1</td>
</tr>
</tbody>
</table>
WHICH Function

Searches for a character value that is equal to the first argument, and returns the index of the first matching value.

**Category:** Search

**Restriction:**
This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility.

**Syntax**

```
WHICHC(string, value-1 <, value-2, ...>)
```

**Required Arguments**

- `string` is a character constant, variable, or expression that specifies the value to search for.
- `value` is a character constant, variable, or expression that specifies the value to be searched.

**Details**

The WHICHC function searches the second and subsequent arguments for a value that is equal to the first argument, and returns the index of the first matching value.

If `string` is missing, then WHICHC returns a missing value. Otherwise, WHICHC compares the value of `string` with `value-1`, `value-2`, and so on, in sequence. If argument `value-i` equals `string`, then WHICHC returns the positive integer `i`. If `string` does not equal any subsequent argument, then WHICHC returns 0.

Using WHICHC is useful when the values that are being searched are subject to frequent change. If you need to perform many searches without changing the values that are being searched, using the HASH object is much more efficient.

**Example**

The following example searches the array for the first argument and returns the index of the first matching value.

```sas
data _null_;
  array fruit (*) $12 fruit1-fruit3 ('watermelon' 'apple' 'banana');
  x1=whichc('watermelon', of fruit[*]);
  x2=whichc('banana', of fruit[*]);
  x3=whichc('orange', of fruit[*]);
  put x1= / x2= / x3=;
run;
```

SAS writes the following output to the log:

```
x1=1
x2=3
x3=0
```
WHICHN Function

Searches for a numeric value that is equal to the first argument, and returns the index of the first matching value.

Category: Search

Syntax

WHICHN(argument, value-1 <, value-2, ...>)

Required Arguments

argument
is a numeric constant, variable, or expression that specifies the value to search for.

value
is a numeric constant, variable, or expression that specifies the value to be searched.

Details

The WHICHN function searches the second and subsequent arguments for a value that is equal to the first argument, and returns the index of the first matching value.

If argument is missing, then WHICHN returns a missing value. Otherwise, WHICHN compares the value of argument with value-1, value-2, and so on, in sequence. If value-i equals argument, then WHICHN returns the positive integer i. If argument does not equal any subsequent argument, then WHICHN returns 0.

Using WHICHN is useful when the values that are being searched are subject to frequent change. If you need to perform many searches without changing the values that are being searched, using the HASH object is much more efficient.

Example

The following example searches the array for the first argument and returns the index of the first matching value.

```
data _null_; array dates[*] Columbus Hastings Nicea US_Independence missing Magna_Carta Gutenberg (1492 1066 325 1776 . 1215 1450); x0=whichn(., of dates[*]); x1=whichn(1492, of dates[*]); x2=whichn(1066, of dates[*]); x3=whichn(1450, of dates[*]); x4=whichn(1000, of dates[*]); put x0= / x1= / x2= / x3= / x4=; run;
```
SAS writes the following results to the log:

```
x0=.
x1=1
x2=2
x3=7
x4=0
```

See Also

Functions:
- “WHICHC Function” on page 688

---

**YEAR Function**

Returns the year from a SAS date value.

**Category:** Date and Time

**Syntax**

```
YEAR(date)
```

**Required Argument**

date

specifies a SAS expression that represents a SAS date value.

**Details**

The YEAR function produces a four-digit numeric value that represents the year.

**Example**

The following SAS statements produce this result:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>date='25dec97'd;</td>
<td></td>
</tr>
<tr>
<td>y=year(date);</td>
<td></td>
</tr>
<tr>
<td>put y;</td>
<td>1997</td>
</tr>
</tbody>
</table>

See Also

Functions:
- “DAY Function” on page 223
- “MONTH Function” on page 442
YIELDP Function

Returns the yield-to-maturity for a periodic cash flow stream, such as a bond.

**Category:** Financial

**Syntax**

\[
\text{YIELDP}(A, c, n, K, k_0, p)
\]

**Required Arguments**

\(A\)

specifies the face value.

Range \( A > 0 \)

\(c\)

specifies the nominal annual coupon rate, expressed as a fraction.

Range \( 0 \leq c < 1 \)

\(n\)

specifies the number of coupons per year.

Range \( n > 0 \) and is an integer

\(K\)

specifies the number of remaining coupons from settlement date to maturity.

Range \( K > 0 \) and is an integer

\(k_0\)

specifies the time from settlement date to the next coupon as a fraction of the annual basis.

Range \( 0 < k_0 \leq \frac{1}{n} \)

\(p\)

specifies the price with accrued interest.

Range \( p > 0 \)

**Details**

The YIELDP function is based on the following relationship:

\[
P = \sum_{k=1}^{K} c(k) \frac{1}{(1 + \frac{k}{n})^k}
\]

The following relationships apply to the preceding equation:
• \( t_k = nk_0 + k - 1 \)

• \( c(k) = \frac{c}{n} A \text{ for } k = 1, \ldots, K - 1 \)

• \( c(K) = \left( 1 + \frac{c}{n} \right) A \)

The YIELDP function solves for \( y \).

**Example**

In the following example, the YIELDP function returns the yield-to-maturity of a bond that has a face value of 1000, an annual coupon rate of 0.01, 4 coupons per year, and 14 remaining coupons. The time from settlement date to next coupon date is 0.165, and the price with accrued interest is 800.

```sas
data _null_
  y=yieldp(1000, .01, 4, 14, .165, 800)
  put y;
run;

SAS writes the following output to the log:

0.0775031248
```

**YRDIF Function**

Returns the difference in years between two dates according to specified day count conventions; returns a person’s age.

**Category:** Date and Time

**Syntax**

`YRDIF(start-date, end-date, <basis>)`

**Required Arguments**

- `start-date` specifies a SAS date value that identifies the starting date.
- `end-date` specifies a SAS date value that identifies the ending date.

**Optional Argument**

- `basis` identifies a character constant or variable that describes how SAS calculates a date difference or a person’s age. The following character strings are valid:
  - `'30/360'` specifies a 30-day month and a 360-day year in calculating the number of years. Each month is considered to have 30 days, and each year 360 days, regardless of the actual number of days in each month or year.
Alias '360'

Tip If either date falls at the end of a month, it is treated as if it were the last day of a 30-day month.

'ACT/ACT'
uses the actual number of days between dates in calculating the number of years. SAS calculates this value as the number of days that fall in 365-day years divided by 365 plus the number of days that fall in 366-day years divided by 366.

Alias 'Actual'

'ACT/360'
uses the actual number of days between dates in calculating the number of years. SAS calculates this value as the number of days divided by 360, regardless of the actual number of days in each year.

'ACT/365'
uses the actual number of days between dates in calculating the number of years. SAS calculates this value as the number of days divided by 365, regardless of the actual number of days in each year.

'AGE'
specifies that a person’s age is computed.

If you do not specify a third argument, AGE becomes the default value for basis.

Details

Using YRDIF in Financial Applications

The Basics
The YRDIF function can be used in calculating interest for fixed income securities when the third argument, basis, is present. YRDIF returns the difference between two dates according to specified day count conventions.

Calculations That Use ACT/ACT Basis
In YRDIF calculations that use the ACT/ACT basis, both a 365–day year and 366–day year are taken into account. For example, if \( n_{365} \) equals the number of days between the start and end dates in a 365–day year, and \( n_{366} \) equals the number of days between the start and end dates in a 366–day year, the YRDIF calculation is computed as \( YRDIF = \frac{n_{365}}{365.0} + \frac{n_{366}}{366.0} \). This calculation corresponds to the commonly understood ACT/ACT day count basis that is documented in the financial literature. The values for basis also include 30/360, ACT/360, and ACT/365. Each has well-defined meanings that must be conformed to in calculating interest payments for specific financial instruments.

Computing a Person’s Age
The YRDIF function can compute a person’s age. The first two arguments, start-date and end-date, are required. If the value of basis is AGE, then YRDIF computes the age. The age computation takes into account leap years. No other values for basis are valid when computing a person’s age.
Examples

**Example 1: Calculating a Difference in Years Based on Basis**
In the following example, YRDIF returns the difference in years between two dates based on each of the options for basis.

```sas
data _null_;  
  sdate='16oct1998'd;  
  edate='16feb2010'd;  
  y30360=yrdif(sdate, edate, '30/360');  
  yactact=yrdif(sdate, edate, 'ACT/ACT');  
  yact360=yrdif(sdate, edate, 'ACT/360');  
  yact365=yrdif(sdate, edate, 'ACT/365');  
  put y30360= / yactact= / yact360= / yact365= ;  
run;
```

SAS writes the following output to the log:

```
y30360=11.333333333
yactact=11.336986301
yact360=11.502777778
yact365=11.345205479
```

**Example 2: Calculating a Person's Age**
You can calculate a person’s age by using three arguments in the YRDIF function. The third argument, *basis*, must have a value of AGE:

```sas
data _null_;  
  sdate='16oct1998'd;  
  edate='16feb2010'd;  
  age=yrdif(sdate, edate, 'AGE');  
  put age= 'years';  
run;
```

SAS writes the following output to the log:

```
age=11.336986301 years
```

**See Also**

**Functions:**
- “DATDIF Function” on page 217

**References**


YYQ Function

Returns a SAS date value from year and quarter year values.

**Category:** Date and Time

**Syntax**

```
YYQ(year, quarter)
```

**Required Arguments**

- `year` specifies a two-digit or four-digit integer that represents the year. The YEARCUTOFF= system option defines the year value for two-digit dates.
- `quarter` specifies the quarter of the year (1, 2, 3, or 4).

**Details**

The YYQ function returns a SAS date value that corresponds to the first day of the specified quarter. If either `year` or `quarter` is missing, or if the quarter value is not valid, the result is missing.

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>DateValue=yyq(2001, 3); put DateValue;</td>
<td>15157</td>
</tr>
<tr>
<td>put DateValue date7.;</td>
<td>01JUL01</td>
</tr>
<tr>
<td>put DateValue date9.;</td>
<td>01JUL2001</td>
</tr>
<tr>
<td>StartOfQtr=yyq(99, 4); put StartOfQtr;</td>
<td>14518</td>
</tr>
<tr>
<td>put StartOfQtr=worddate.;</td>
<td>StartOfQtr=October 1, 1999</td>
</tr>
</tbody>
</table>

**See Also**

- "QTR Function" on page 548
- "YEAR Function" on page 690

**System Options:**

- "YEARCUTOFF= System Option" in *SAS Viya System Options: Reference*
Chapter 3
Dictionary of Functions and CALL Routines for SAS Only
CALL RANUNI Routine ........................................ 814
CALL SCAN Routine ........................................ 816
CALL RANCOMB Routine .................................... 827
CALL RANPERK Routine ..................................... 829
CALL RANPERM Routine .................................... 831
CALL SET Routine ........................................... 833
CALL SOFTMAX Routine .................................... 835
CALL SORTC Routine ........................................ 836
CALL SORTN Routine ........................................ 837
CALL STDIZE Routine ...................................... 838
CALL SYMPUT Routine ..................................... 841
CALL SYMPUTX Routine .................................... 842
CALL SYSTEM Routine ..................................... 844
CALL TANH Routine ......................................... 845
CATQ Function ................................................ 846
CEXIST Function .............................................. 850
CHAR Function ............................................... 851
CINV Function ................................................ 853
CLOSE Function .............................................. 854
CMISS Function .............................................. 855
COMPGED Function ......................................... 856
COMPLEV Function ......................................... 862
CUROBS Function ............................................ 864
DACCDB Function ............................................ 865
DACCDBSL Function ......................................... 866
DACCSDL Function ........................................... 867
DACCSSYD Function ......................................... 868
DACCTAB Function ......................................... 869
DCLOSE Function ............................................. 870
DCREATE Function .......................................... 872
DEPDB Function ............................................... 873
DEPDBSL Function ........................................... 874
DEPSL Function ............................................... 875
DEPSYD Function ............................................ 876
DEPTAB Function ............................................. 876
DINFO Function .............................................. 877
DNUM Function ............................................... 880
DOPEN Function .............................................. 881
DOPTNAME Function ......................................... 882
DOPTNUM Function ........................................... 884
DOSUBL Function ............................................ 885
DREAD Function .............................................. 887
DROPNOTE Function ......................................... 888
DSNAME Function ............................................ 889
ENVLEN Function ............................................ 890
EUCLID Function ............................................. 891
EXIST Function ............................................... 892
FAPPEND Function .......................................... 895
FCLOSE Function ............................................. 896
FCOL Function ............................................... 897
FCOPY Function .............................................. 898
FDELETE Function ........................................... 902
FETCH Function ............................................... 903
FETCHOBS Function ......................................... 905
FEXIST Function .............................................. 906
FGET Function ................................................. 907
PATHNAME Function .................................................. 1013
PEEK Function ...................................................... 1014
PEEK Function ...................................................... 1016
PEEKCLONG Function ............................................. 1017
PEEKLONG Function ................................................. 1018
POINT Function ..................................................... 1019
PROPCASE Function ................................................. 1020
PRXPAREN Function ............................................... 1022
PTRLONGADD Function ............................................ 1024
PUTC Function ....................................................... 1025
PUTN Function ....................................................... 1028
RANBIN Function ................................................... 1030
RANCAU Function .................................................. 1031
RANEXP Function ................................................... 1032
RANGAM Function .................................................. 1033
RANNOR Function .................................................. 1035
RANPOI Function ................................................... 1036
RANTBL Function ................................................... 1037
RANTRI Function ................................................... 1038
RANUNI Function ................................................... 1039
RENAME Function .................................................. 1040
RESOLVE Function .................................................. 1042
REWIND Function .................................................... 1043
SAVING Function .................................................... 1044
SHA256 Function ..................................................... 1045
SOAPWEB Function .................................................. 1046
SOAPWEBMETA Function .......................................... 1048
SOAPWIPSERVICE Function ....................................... 1050
SOAPWIPSRS Function ............................................. 1052
SOAPWS Function .................................................. 1055
SOAPWSMETA Function ............................................ 1056
SOUNDEX Function .................................................. 1058
SPEDIS Function ..................................................... 1059
SQUANTILE Function ............................................... 1062
STFIPS Function ..................................................... 1064
STNAME Function ................................................... 1066
STNAMEL Function .................................................. 1067
SUBPAD Function .................................................... 1068
SYMEXIST Function .................................................. 1069
SYMGET Function .................................................... 1070
SYMGLOBAL Function ............................................. 1071
SYMLOCAL Function .................................................. 1071
SYSEXIST Function .................................................. 1072
SYSGET Function .................................................... 1073
SYSMSG Function .................................................... 1074
SYSPARM Function .................................................. 1075
SYSPROCESSID Function .......................................... 1076
SYSPROCESSNAME Function ...................................... 1077
SYSPROD Function ................................................... 1078
SYSRC Function ...................................................... 1079
TANH Function ........................................................ 1079
TYPEOF Function ..................................................... 1080
TZONEID Function .................................................... 1080
TZONENAME Function .............................................. 1082
TZONEOFF Function .................................................. 1083
TZONES2U Function .................................................. 1085
SAS Functions and CALL Routines Documented in Other SAS Publications

Functions and CALL routines with related subject matter are also documented in the following publications.

- SAS Cloud Analytic Services: Programming and Reference
- SAS Macro Language: Reference

SAS Functions and CALL Routines by Category

Here are the categories for SAS functions and CALL routines:

- **Arithmetic**: returns the result of a division that handles special missing values for ODS output.
- **Array**: returns information about arrays.
- **Bitwise Logical Operations**: returns the bitwise logical result for an argument.
- **Character**: returns information based on character data.
- **Character String Matching**: returns information from Perl regular expressions.
- **Combinatorial**: generates combinations and permutations.
- **Date and Time**: returns date and time values, including time intervals.
- **Descriptive Statistics**: returns statistical values, such as mean, median, and standard deviation.
- **Distance**: returns the geodetic distance.
- **External Files**: returns information that is associated with external files.
External Routines returns a character or numeric value, or calls a routine without any return code.

Financial calculates financial values such as interest, periodic payments, depreciation, and prices for European options on stocks.

Hyperbolic performs hyperbolic calculations such as sine, cosine, and tangent.

Macro assigns a value to a macro variable, returns the value of a macro variable, determines whether a macro variable is global or local in scope, and identifies whether a macro variable exists.

Mathematical performs mathematical calculations such as factorials, absolute value, fuzzy comparisons, and logarithms.

Numeric returns a numeric value based on whether an expression is true, false, or missing, or determines whether a software image exists in the installed version of SAS.

Probability returns probability calculations, such as from a chi-square or Poisson distribution.

Quantile returns a quantile from specific distributions.

Random Number returns random variates from specific distributions.

SAS File I/O returns information about SAS files.

Search searches for character or numeric values.

Sort sorts the values of character or numeric arguments.

Special returns and stores memory addresses, writes a value directly into memory, suspends execution of a program, submits an operating-environment command for execution, returns the value of a SAS system or graphics option, specifies formats and informats at run time, returns the system return code, returns the UUID, determines whether a product is licensed, as well as returns other information about SAS processing.

State and ZIP Code returns ZIP codes, FIPS codes, state and city names, postal codes, and the geodetic distance between ZIP codes.

Trigonometric returns trigonometric values such as sine, cosine, and tangent.

Truncation truncates numeric values and returns numeric values, often using fuzzing, or zero fuzzing.

Variable Control assigns variable labels, links SAS data set variables to DATA step or macro variables, and assigns variable names.

Variable Information returns a name, type, length, informat name, label, and other variable information.

Web Service calls a web service or a SAS registered web service.

Web Tools encodes and decodes a string of data.

The following table lists SAS functions and CALL routines according to category:
<table>
<thead>
<tr>
<th>Category</th>
<th>Language Elements</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character</td>
<td>CALL CATS Routine (p. 736)</td>
<td>Removes leading and trailing blanks, and returns a concatenated character string.</td>
</tr>
<tr>
<td></td>
<td>CALL CATT Routine (p. 738)</td>
<td>Removes trailing blanks, and returns a concatenated character string.</td>
</tr>
<tr>
<td></td>
<td>CALL CATX Routine (p. 740)</td>
<td>Removes leading and trailing blanks, inserts delimiters, and returns a concatenated character string.</td>
</tr>
<tr>
<td></td>
<td>CALL COMPCOST Routine (p. 742)</td>
<td>Sets the costs of operations for later use by the COMPGED function</td>
</tr>
<tr>
<td></td>
<td>CALL SCAN Routine (p. 816)</td>
<td>Returns the position and length of the nth word from a character string.</td>
</tr>
<tr>
<td></td>
<td>CATQ Function (p. 846)</td>
<td>Concatenates character or numeric values by using a delimiter to separate items and by adding quotation marks to strings that contain the delimiter.</td>
</tr>
<tr>
<td></td>
<td>CHAR Function (p. 851)</td>
<td>Returns a single character from a specified position in a character string.</td>
</tr>
<tr>
<td></td>
<td>COMPGED Function (p. 856)</td>
<td>Returns the generalized edit distance between two strings.</td>
</tr>
<tr>
<td></td>
<td>COMPLEV Function (p. 862)</td>
<td>Returns the Levenshtein edit distance between two strings.</td>
</tr>
<tr>
<td></td>
<td>FIRST Function (p. 921)</td>
<td>Returns the first character in a character string.</td>
</tr>
<tr>
<td></td>
<td>IFC Function (p. 971)</td>
<td>Returns a character value based on whether an expression is true, false, or missing.</td>
</tr>
<tr>
<td></td>
<td>MVALID Function (p. 1001)</td>
<td>Checks the validity of a character string for use as a SAS member name.</td>
</tr>
<tr>
<td></td>
<td>NLITERAL Function (p. 1004)</td>
<td>Converts a character string that you specify to a SAS name literal.</td>
</tr>
<tr>
<td></td>
<td>NVALID Function (p. 1008)</td>
<td>Checks the validity of a character string for use as a SAS variable name.</td>
</tr>
<tr>
<td></td>
<td>PROPCASE Function (p. 1020)</td>
<td>Converts all words in an argument to proper case.</td>
</tr>
<tr>
<td></td>
<td>SHA256 Function (p. 1045)</td>
<td>Returns the result of the message digest of a specified string.</td>
</tr>
<tr>
<td></td>
<td>SOUNDEX Function (p. 1058)</td>
<td>Encodes a string to facilitate searching.</td>
</tr>
<tr>
<td></td>
<td>SPEDIS Function (p. 1059)</td>
<td>Determines the likelihood of two words matching, expressed as the asymmetric spelling distance between the two words.</td>
</tr>
<tr>
<td></td>
<td>SUBPAD Function (p. 1068)</td>
<td>Returns a substring that has a length that you specify, using blank padding if necessary.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Character String Matching</td>
<td>TYPEOF Function (p. 1080)</td>
<td>Returns a value that indicates whether the argument is character or numeric.</td>
</tr>
<tr>
<td></td>
<td>CALL PRXCHANGE Routine (p. 782)</td>
<td>Performs a pattern-matching replacement.</td>
</tr>
<tr>
<td></td>
<td>CALL PRXDEBUG Routine (p. 785)</td>
<td>Enables Perl regular expressions in a DATA step to send debugging output to the SAS log.</td>
</tr>
<tr>
<td></td>
<td>CALL PRXNEXT Routine (p. 787)</td>
<td>Returns the position and length of a substring that matches a pattern, and iterates over multiple matches within one string.</td>
</tr>
<tr>
<td></td>
<td>CALL PRXPOSN Routine (p. 789)</td>
<td>Returns the start position and length for a capture buffer.</td>
</tr>
<tr>
<td></td>
<td>CALL PRXSUBSTR Routine (p. 792)</td>
<td>Returns the position and length of a substring that matches a pattern.</td>
</tr>
<tr>
<td></td>
<td>PRXPAREN Function (p. 1022)</td>
<td>Returns the last bracket match for which there is a match in a pattern.</td>
</tr>
<tr>
<td>Combinatorial</td>
<td>ALLCOMB Function (p. 716)</td>
<td>Generates all combinations of the values of n variables taken k at a time in a minimal change order.</td>
</tr>
<tr>
<td></td>
<td>ALLPERM Function (p. 718)</td>
<td>Generates all permutations of the values of several variables in a minimal change order.</td>
</tr>
<tr>
<td></td>
<td>CALL ALLCOMB Routine (p. 727)</td>
<td>Generates all combinations of the values of n variables taken k at a time in a minimal change order.</td>
</tr>
<tr>
<td></td>
<td>CALL ALLCOMBI Routine (p. 730)</td>
<td>Generates all combinations of the indices of n objects taken k at a time in a minimal change order.</td>
</tr>
<tr>
<td></td>
<td>CALL ALLPERM Routine (p. 733)</td>
<td>Generates all permutations of the values of several variables in a minimal change order.</td>
</tr>
<tr>
<td></td>
<td>CALL GRAYCODE Routine (p. 746)</td>
<td>Generates all subsets of n items in a minimal change order.</td>
</tr>
<tr>
<td></td>
<td>CALL LEXCOMB Routine (p. 760)</td>
<td>Generates all distinct combinations of the nonmissing values of n variables taken k at a time in lexicographic order.</td>
</tr>
<tr>
<td></td>
<td>CALL LEXCOMBI Routine (p. 764)</td>
<td>Generates all combinations of the indices of n objects taken k at a time in lexicographic order.</td>
</tr>
<tr>
<td></td>
<td>CALL LEXPERK Routine (p. 766)</td>
<td>Generates all distinct permutations of the nonmissing values of n variables taken k at a time in lexicographic order.</td>
</tr>
<tr>
<td></td>
<td>CALL LEXPERM Routine (p. 770)</td>
<td>Generates all distinct permutations of the nonmissing values of several variables in lexicographic order.</td>
</tr>
<tr>
<td></td>
<td>CALL RANCOMB Routine (p. 827)</td>
<td>Permutes the values of the arguments, and returns a random combination of k out of n values.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CALL RANPERK</td>
<td>Routine (p. 829)</td>
<td>Permutes the values of the arguments, and returns a random permutation of k out of n values.</td>
</tr>
<tr>
<td>CALL RANPERM</td>
<td>Routine (p. 831)</td>
<td>Randomly permutes the values of the arguments.</td>
</tr>
<tr>
<td>GRAYCODE</td>
<td>Function (p. 946)</td>
<td>Generates all subsets of n items in a minimal change order.</td>
</tr>
<tr>
<td>LEXCOMB</td>
<td>Function (p. 977)</td>
<td>Generates all distinct combinations of the nonmissing values of n variables taken k at a time in lexicographic order.</td>
</tr>
<tr>
<td>LEXCOMBI</td>
<td>Function (p. 979)</td>
<td>Generates all combinations of the indices of n objects taken k at a time in lexicographic order.</td>
</tr>
<tr>
<td>LEXPERK</td>
<td>Function (p. 981)</td>
<td>Generates all distinct permutations of the nonmissing values of n variables taken k at a time in lexicographic order.</td>
</tr>
<tr>
<td>LEXPERM</td>
<td>Function (p. 984)</td>
<td>Generates all distinct permutations of the nonmissing values of several variables in lexicographic order.</td>
</tr>
<tr>
<td>Date and Time</td>
<td>CALL IS8601_CONVERT Routine (p. 750)</td>
<td>Converts an ISO 8601 interval to datetime and duration values, or converts datetime and duration values to an ISO 8601 interval.</td>
</tr>
<tr>
<td></td>
<td>HOLIDAYCK Function (p. 949)</td>
<td>Returns the number of occurrences of the holiday value between date1 and date2.</td>
</tr>
<tr>
<td></td>
<td>HOLIDAYCOUNT Function (p. 952)</td>
<td>Returns the number of holidays defined for a SAS date value.</td>
</tr>
<tr>
<td></td>
<td>HOLIDAYNAME Function (p. 955)</td>
<td>Returns the name of the holiday that corresponds to the SAS date or a blank string if a holiday is not defined for the SAS date.</td>
</tr>
<tr>
<td></td>
<td>HOLIDAYNX Function (p. 958)</td>
<td>Returns the nth occurrence of the holiday relative to the date argument.</td>
</tr>
<tr>
<td></td>
<td>HOLIDAYNY Function (p. 960)</td>
<td>Returns the nth occurrence of the holiday for the year.</td>
</tr>
<tr>
<td></td>
<td>HOLIDAYTEST Function (p. 964)</td>
<td>Returns 1 if the holiday occurs on the SAS date value.</td>
</tr>
<tr>
<td></td>
<td>TZONEID Function (p. 1080)</td>
<td>Returns the current time zone ID.</td>
</tr>
<tr>
<td></td>
<td>TZONENAME Function (p. 1082)</td>
<td>Returns the current standard or daylight savings time, time zone name.</td>
</tr>
<tr>
<td></td>
<td>TZONEOFF Function (p. 1083)</td>
<td>Returns the user time zone offset.</td>
</tr>
<tr>
<td></td>
<td>TZONEES2U Function (p. 1085)</td>
<td>Converts a SAS date time value to a UTC date time value.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Category</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TZONEU2S Function (p. 1086)</td>
<td></td>
<td>Converts a UTC date time value to a SAS date time value.</td>
</tr>
<tr>
<td>Descriptive</td>
<td>CMISS Function (p. 855)</td>
<td>Counts the number of missing arguments.</td>
</tr>
<tr>
<td>Statistics</td>
<td>EUCLID Function (p. 891)</td>
<td>Returns the Euclidean norm of the nonmissing arguments.</td>
</tr>
<tr>
<td></td>
<td>LPNORM Function (p. 990)</td>
<td>Returns the Lp norm of the second argument and subsequent nonmissing arguments.</td>
</tr>
<tr>
<td>Distance</td>
<td>ZIPCITYDISTANCE Function (p. 1100)</td>
<td>Returns the geodetic distance between two ZIP code locations.</td>
</tr>
<tr>
<td>External Files</td>
<td>DCLOSE Function (p. 870)</td>
<td>Closes a directory that was opened by the DOPEN function.</td>
</tr>
<tr>
<td></td>
<td>DCREATE Function (p. 872)</td>
<td>Returns the complete pathname of a new, external directory.</td>
</tr>
<tr>
<td></td>
<td>DINFO Function (p. 877)</td>
<td>Returns information about a directory.</td>
</tr>
<tr>
<td></td>
<td>DNUM Function (p. 880)</td>
<td>Returns the number of members in a directory.</td>
</tr>
<tr>
<td></td>
<td>DOPEN Function (p. 881)</td>
<td>Opens a directory, and returns a directory identifier value.</td>
</tr>
<tr>
<td></td>
<td>DOPTNAME Function (p. 882)</td>
<td>Returns directory attribute information.</td>
</tr>
<tr>
<td></td>
<td>DOPTNUM Function (p. 884)</td>
<td>Returns the number of information items that are available for a directory.</td>
</tr>
<tr>
<td></td>
<td>DREAD Function (p. 887)</td>
<td>Returns the name of a directory member.</td>
</tr>
<tr>
<td></td>
<td>DROPNOTE Function (p. 888)</td>
<td>Deletes a note marker from a SAS data set or an external file.</td>
</tr>
<tr>
<td></td>
<td>FAPPEND Function (p. 895)</td>
<td>Appends the current record to the end of an external file.</td>
</tr>
<tr>
<td></td>
<td>FCLOSE Function (p. 896)</td>
<td>Closes an external file, directory, or directory member.</td>
</tr>
<tr>
<td></td>
<td>FCOL Function (p. 897)</td>
<td>Returns the current column position in the File Data Buffer (FDB).</td>
</tr>
<tr>
<td></td>
<td>FDELETE Function (p. 902)</td>
<td>Deletes an external file or an empty directory.</td>
</tr>
<tr>
<td></td>
<td>FEXIST Function (p. 906)</td>
<td>Verifies the existence of an external file that is associated with a fileref.</td>
</tr>
<tr>
<td></td>
<td>FGET Function (p. 907)</td>
<td>Copies data from the File Data Buffer (FDB) into a variable.</td>
</tr>
<tr>
<td></td>
<td>FILEEXIST Function (p. 909)</td>
<td>Verifies the existence of an external file by its physical name.</td>
</tr>
<tr>
<td></td>
<td>FILENAME Function (p. 910)</td>
<td>Assigns or deassigns a fileref to an external file, directory, or output device.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>FILEREF Function</td>
<td>(p. 913) Verifies whether a fileref has been assigned for the current SAS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>session.</td>
</tr>
<tr>
<td></td>
<td>FINFO Function</td>
<td>(p. 914) Returns the value of a file information item.</td>
</tr>
<tr>
<td></td>
<td>FNOTE Function</td>
<td>(p. 924) Identifies the last record that was read, and returns a value that</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the FPOINT function can use.</td>
</tr>
<tr>
<td></td>
<td>FOPEN Function</td>
<td>(p. 925) Opens an external file and returns a file identifier value.</td>
</tr>
<tr>
<td></td>
<td>FOPTNAME Function</td>
<td>(p. 928) Returns the name of an item of information about an external file.</td>
</tr>
<tr>
<td></td>
<td>FOPTNUM Function</td>
<td>(p. 931) Returns the number of information items, such as filename or</td>
</tr>
<tr>
<td></td>
<td></td>
<td>record length, that are available for an external file.</td>
</tr>
<tr>
<td></td>
<td>FPOINT Function</td>
<td>(p. 932) Positions the read pointer on the next record to be read.</td>
</tr>
<tr>
<td></td>
<td>FPOS Function</td>
<td>(p. 934) Sets the position of the column pointer in the File Data Buffer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(FDB).</td>
</tr>
<tr>
<td></td>
<td>FPUT Function</td>
<td>(p. 936) Moves data to the File Data Buffer (FDB) of an external file,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>starting at the FDB's current column position.</td>
</tr>
<tr>
<td></td>
<td>FREAD Function</td>
<td>(p. 937) Reads a record from an external file into the File Data Buffer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(FDB).</td>
</tr>
<tr>
<td></td>
<td>FREWIND Function</td>
<td>(p. 938) Positions the file pointer to the start of the file.</td>
</tr>
<tr>
<td></td>
<td>FRLEN Function</td>
<td>(p. 940) Returns the size of the last record that was read, or, if the file</td>
</tr>
<tr>
<td></td>
<td></td>
<td>opened for output, returns the current record size.</td>
</tr>
<tr>
<td></td>
<td>FSEP Function</td>
<td>(p. 941) Sets the token delimiters for the FGET function.</td>
</tr>
<tr>
<td></td>
<td>FWRITE Function</td>
<td>(p. 942) Writes a record to an external file.</td>
</tr>
<tr>
<td></td>
<td>MOPEN Function</td>
<td>(p. 996) Opens a file by directory ID and member name, and returns either</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the file identifier or a 0.</td>
</tr>
<tr>
<td></td>
<td>PATHNAME Function</td>
<td>(p. 1013) Returns the physical name of an external file or a SAS library,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>or returns a blank.</td>
</tr>
<tr>
<td></td>
<td>RENAME Function</td>
<td>(p. 1040) Renames a member of a SAS library, an entry in a SAS catalog, an</td>
</tr>
<tr>
<td></td>
<td></td>
<td>external file, or a directory.</td>
</tr>
<tr>
<td></td>
<td>SYSMMSG Function</td>
<td>(p. 1074) Returns error or warning message text from processing the last</td>
</tr>
<tr>
<td></td>
<td></td>
<td>data set or external file function.</td>
</tr>
<tr>
<td></td>
<td>SYSRC Function</td>
<td>(p. 1079) Returns a system error number.</td>
</tr>
<tr>
<td></td>
<td>CALL MODULE Routine</td>
<td>(p. 775) Calls an external routine without any return code.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Module</td>
<td>MODULE Function (p. 993)</td>
<td>Calls a specific routine or module that resides in an external dynamic link library (DLL).</td>
</tr>
<tr>
<td></td>
<td>MODULEC Function (p. 995)</td>
<td>Calls an external routine and returns a character value.</td>
</tr>
<tr>
<td></td>
<td>MODULEN Function (p. 995)</td>
<td>Calls an external routine and returns a numeric value.</td>
</tr>
<tr>
<td>Financial</td>
<td>DACCDB Function (p. 865)</td>
<td>Returns the accumulated declining balance depreciation.</td>
</tr>
<tr>
<td></td>
<td>DACCDBSL Function (p. 866)</td>
<td>Returns the accumulated declining balance with conversion to a straight-line depreciation.</td>
</tr>
<tr>
<td></td>
<td>DACCSL Function (p. 867)</td>
<td>Returns the accumulated straight-line depreciation.</td>
</tr>
<tr>
<td></td>
<td>DACCSYD Function (p. 868)</td>
<td>Returns the accumulated sum-of-years-digits depreciation.</td>
</tr>
<tr>
<td></td>
<td>DACCTAB Function (p. 869)</td>
<td>Returns the accumulated depreciation from specified tables.</td>
</tr>
<tr>
<td></td>
<td>DEPDB Function (p. 873)</td>
<td>Returns the declining balance depreciation.</td>
</tr>
<tr>
<td></td>
<td>DEPDBSL Function (p. 874)</td>
<td>Returns the declining balance with conversion to a straight-line depreciation.</td>
</tr>
<tr>
<td></td>
<td>DEPSL Function (p. 875)</td>
<td>Returns the straight-line depreciation.</td>
</tr>
<tr>
<td></td>
<td>DEPSYD Function (p. 876)</td>
<td>Returns the sum-of-years-digits depreciation.</td>
</tr>
<tr>
<td></td>
<td>DEPTAB Function (p. 876)</td>
<td>Returns the depreciation from specified tables.</td>
</tr>
<tr>
<td></td>
<td>SAVING Function (p. 1044)</td>
<td>Returns the future value of a periodic saving.</td>
</tr>
<tr>
<td>Hyperbolic</td>
<td>TANH Function (p. 1079)</td>
<td>Returns the hyperbolic tangent.</td>
</tr>
<tr>
<td>Macro</td>
<td>CALL EXECUTE Routine (p. 745)</td>
<td>Resolves the argument, and issues the resolved value for execution at the next step boundary.</td>
</tr>
<tr>
<td></td>
<td>CALL SYMPUT Routine (p. 841)</td>
<td>Assigns DATA step information to a macro variable.</td>
</tr>
<tr>
<td></td>
<td>CALL SYMPUTX Routine (p. 842)</td>
<td>Assigns a value to a macro variable, and removes both leading and trailing blanks.</td>
</tr>
<tr>
<td></td>
<td>DOSUBL Function (p. 885)</td>
<td>Imports macro variables from the calling environment, and exports macro variables back to the calling environment.</td>
</tr>
<tr>
<td></td>
<td>RESOLVE Function (p. 1042)</td>
<td>Returns the resolved value of the argument after the argument has been processed by the macro facility.</td>
</tr>
<tr>
<td></td>
<td>SYMEXIST Function (p. 1069)</td>
<td>Returns an indication of the existence of a macro variable.</td>
</tr>
<tr>
<td></td>
<td>SYMGET Function (p. 1070)</td>
<td>Returns the value of a macro variable during DATA step execution.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Category</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Language</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Elements</td>
<td>SYMGLOBL Function (p.</td>
<td>Returns an indication of whether a macro variable is in global scope to the</td>
</tr>
<tr>
<td></td>
<td>1071)</td>
<td>DATA step during DATA step execution.</td>
</tr>
<tr>
<td></td>
<td>SYMLOCAL Function (p.</td>
<td>Returns an indication of whether a macro variable is in local scope to the</td>
</tr>
<tr>
<td></td>
<td>1071)</td>
<td>DATA step during DATA step execution.</td>
</tr>
<tr>
<td></td>
<td>Mathemtical</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CALL LOGISTIC Routine (p.</td>
<td>Applies the logistic function to each argument.</td>
</tr>
<tr>
<td></td>
<td>774)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CALL SOFTMAX Routine (p.</td>
<td>Returns the softmax value.</td>
</tr>
<tr>
<td></td>
<td>835)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CALL STDIZE Routine (p.</td>
<td>Standardizes the values of one or more variables.</td>
</tr>
<tr>
<td></td>
<td>838)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CALL TANH Routine (p.</td>
<td>Returns the hyperbolic tangent.</td>
</tr>
<tr>
<td></td>
<td>845)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LOGISTIC Function (p.</td>
<td>Returns the logistic transformation of the argument.</td>
</tr>
<tr>
<td></td>
<td>990)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MSPLINT Function (p.</td>
<td>Returns the ordinate of a monotonicity-preserving interpolating spline.</td>
</tr>
<tr>
<td></td>
<td>999)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Numeric</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IFN Function (p. 973)</td>
<td>Returns a numeric value based on whether an expression is true, false, or</td>
</tr>
<tr>
<td></td>
<td></td>
<td>missing.</td>
</tr>
<tr>
<td></td>
<td>MODEEXIST Function (p.</td>
<td>Determines whether a software image exists in the version of SAS you have</td>
</tr>
<tr>
<td></td>
<td>992)</td>
<td>installed.</td>
</tr>
<tr>
<td></td>
<td>Quantile</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CINV Function (p. 853)</td>
<td>Returns a quantile from the chi-square distribution.</td>
</tr>
<tr>
<td></td>
<td>FINV Function (p. 916)</td>
<td>Returns a quantile from the F distribution.</td>
</tr>
<tr>
<td></td>
<td>SQUANTILE Function (p.</td>
<td>Returns the quantile from a distribution when you specify the right</td>
</tr>
<tr>
<td></td>
<td>1062)</td>
<td>probability (SDF).</td>
</tr>
<tr>
<td></td>
<td>Random Number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CALL RANBIN Routine (p.</td>
<td>Returns a random variate from a binomial distribution.</td>
</tr>
<tr>
<td></td>
<td>794)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CALL RANCAU Routine (p.</td>
<td>Returns a random variate from a Cauchy distribution.</td>
</tr>
<tr>
<td></td>
<td>797)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CALL RANEXP Routine (p.</td>
<td>Returns a random variate from an exponential distribution.</td>
</tr>
<tr>
<td></td>
<td>800)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CALL RANGAM Routine (p.</td>
<td>Returns a random variate from a gamma distribution.</td>
</tr>
<tr>
<td></td>
<td>802)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CALL RANNOR Routine (p.</td>
<td>Returns a random variate from a normal distribution.</td>
</tr>
<tr>
<td></td>
<td>805)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CALL RANPOI Routine (p.</td>
<td>Returns a random variate from a Poisson distribution.</td>
</tr>
<tr>
<td></td>
<td>807)</td>
<td></td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CALL</td>
<td>RANTBL Routine (p. 809)</td>
<td>Returns a random variate from a tabled probability distribution.</td>
</tr>
<tr>
<td>CALL</td>
<td>RANTRI Routine (p. 812)</td>
<td>Returns a random variate from a triangular distribution.</td>
</tr>
<tr>
<td>CALL</td>
<td>RANUNI Routine (p. 814)</td>
<td>Returns a random variate from a uniform distribution.</td>
</tr>
<tr>
<td>NORMAL</td>
<td>Function (p. 1006)</td>
<td>Returns a random variate from a normal, or Gaussian, distribution.</td>
</tr>
<tr>
<td>RANBIN</td>
<td>Function (p. 1030)</td>
<td>Returns a random variate from a binomial distribution.</td>
</tr>
<tr>
<td>RANCAU</td>
<td>Function (p. 1031)</td>
<td>Returns a random variate from a Cauchy distribution.</td>
</tr>
<tr>
<td>RANEXP</td>
<td>Function (p. 1032)</td>
<td>Returns a random variate from an exponential distribution.</td>
</tr>
<tr>
<td>RANGAM</td>
<td>Function (p. 1033)</td>
<td>Returns a random variate from a gamma distribution.</td>
</tr>
<tr>
<td>RANNOR</td>
<td>Function (p. 1035)</td>
<td>Returns a random variate from a normal distribution.</td>
</tr>
<tr>
<td>RANPOI</td>
<td>Function (p. 1036)</td>
<td>Returns a random variate from a Poisson distribution.</td>
</tr>
<tr>
<td>RANTBL</td>
<td>Function (p. 1037)</td>
<td>Returns a random variate from a tabled probability distribution.</td>
</tr>
<tr>
<td>RANTRI</td>
<td>Function (p. 1038)</td>
<td>Returns a random variate from a triangular distribution.</td>
</tr>
<tr>
<td>RANUNI</td>
<td>Function (p. 1039)</td>
<td>Returns a random variate from a uniform distribution.</td>
</tr>
<tr>
<td>SAS File I/O</td>
<td>ATTRC Function (p. 720)</td>
<td>Returns the value of a character attribute for a SAS data set.</td>
</tr>
<tr>
<td></td>
<td>ATTRN Function (p. 723)</td>
<td>Returns the value of a numeric attribute for a SAS data set.</td>
</tr>
<tr>
<td></td>
<td>CEXIST Function (p. 850)</td>
<td>Verifies the existence of a SAS catalog or SAS catalog entry.</td>
</tr>
<tr>
<td></td>
<td>CLOSE Function (p. 854)</td>
<td>Closes a SAS data set.</td>
</tr>
<tr>
<td></td>
<td>CUROBS Function (p. 864)</td>
<td>Returns the observation number of the current observation.</td>
</tr>
<tr>
<td></td>
<td>DROPNOTE Function (p. 888)</td>
<td>Deletes a note marker from a SAS data set or an external file.</td>
</tr>
<tr>
<td></td>
<td>DSNAME Function (p. 889)</td>
<td>Returns the SAS data set name that is associated with a data set identifier.</td>
</tr>
<tr>
<td></td>
<td>ENVLEN Function (p. 890)</td>
<td>Returns the length of an environment variable.</td>
</tr>
<tr>
<td></td>
<td>EXIST Function (p. 892)</td>
<td>Verifies the existence of a SAS library member.</td>
</tr>
<tr>
<td></td>
<td>FCOPY Function (p. 898)</td>
<td>Copies records from one fileref to another fileref, and returns a value that indicates whether the records were successfully copied.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>FETCH Function (p. 903)</td>
<td>Reads the next non-deleted observation from a SAS data set into the Data Set Data Vector (DDV).</td>
<td></td>
</tr>
<tr>
<td>FETCHOBS Function (p. 905)</td>
<td>Reads a specified observation from a SAS data set into the Data Set Data Vector (DDV).</td>
<td></td>
</tr>
<tr>
<td>GETVARC Function (p. 944)</td>
<td>Returns the value of a SAS data set character variable.</td>
<td></td>
</tr>
<tr>
<td>GETVARN Function (p. 945)</td>
<td>Returns the value of a SAS data set numeric variable.</td>
<td></td>
</tr>
<tr>
<td>IORCMSG Function (p. 975)</td>
<td>Returns a formatted error message for <em>IORC</em>.</td>
<td></td>
</tr>
<tr>
<td>LIBNAME Function (p. 986)</td>
<td>Assigns or clears a libref for a SAS library.</td>
<td></td>
</tr>
<tr>
<td>LIBREF Function (p. 989)</td>
<td>Verifies that a libref has been assigned.</td>
<td></td>
</tr>
<tr>
<td>NOTE Function (p. 1006)</td>
<td>Returns an observation ID for the current observation of a SAS data set.</td>
<td></td>
</tr>
<tr>
<td>OPEN Function (p. 1010)</td>
<td>Opens a SAS data set.</td>
<td></td>
</tr>
<tr>
<td>PATHNAME Function (p. 1013)</td>
<td>Returns the physical name of an external file or a SAS library, or returns a blank.</td>
<td></td>
</tr>
<tr>
<td>POINT Function (p. 1019)</td>
<td>Locates an observation that is identified by the NOTE function.</td>
<td></td>
</tr>
<tr>
<td>RENAME Function (p. 1040)</td>
<td>Renames a member of a SAS library, an entry in a SAS catalog, an external file, or a directory.</td>
<td></td>
</tr>
<tr>
<td>REWIND Function (p. 1043)</td>
<td>Positions the data set pointer at the beginning of a SAS data set.</td>
<td></td>
</tr>
<tr>
<td>SYSEXIST Function (p. 1072)</td>
<td>Returns a value that indicates whether an operating-environment variable exists in your environment.</td>
<td></td>
</tr>
<tr>
<td>SYMMSG Function (p. 1074)</td>
<td>Returns error or warning message text from processing the last data set or external file function.</td>
<td></td>
</tr>
<tr>
<td>SYSRC Function (p. 1079)</td>
<td>Returns a system error number.</td>
<td></td>
</tr>
<tr>
<td>VARFMT Function (p. 1089)</td>
<td>Returns the format that is assigned to a SAS data set variable.</td>
<td></td>
</tr>
<tr>
<td>VARINFMT Function (p. 1091)</td>
<td>Returns the informat that is assigned to a SAS data set variable.</td>
<td></td>
</tr>
<tr>
<td>VARLABEL Function (p. 1092)</td>
<td>Returns the label that is assigned to a SAS data set variable.</td>
<td></td>
</tr>
<tr>
<td>VARLEN Function (p. 1093)</td>
<td>Returns the length of a SAS data set variable.</td>
<td></td>
</tr>
<tr>
<td>VARNAME Function (p. 1094)</td>
<td>Returns the name of a SAS data set variable.</td>
<td></td>
</tr>
<tr>
<td>VARNUM Function (p. 1095)</td>
<td>Returns the number of a variable's position in a SAS data set.</td>
<td></td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>VARTYPE</strong> Function (p. 1097)</td>
<td>Returns the data type of a SAS data set variable.</td>
<td></td>
</tr>
<tr>
<td><strong>Sort</strong></td>
<td>CALL SORTC Routine (p. 836)</td>
<td>Sorts the values of character arguments.</td>
</tr>
<tr>
<td></td>
<td>CALL SORTN Routine (p. 837)</td>
<td>Sorts the values of numeric arguments.</td>
</tr>
<tr>
<td><strong>Special</strong></td>
<td>ADDR Function (p. 714)</td>
<td>Returns the memory address of a variable on a 32-bit platform.</td>
</tr>
<tr>
<td></td>
<td>ADDRLONG Function (p. 715)</td>
<td>Returns the memory address of a variable on 32-bit and 64-bit platforms.</td>
</tr>
<tr>
<td></td>
<td>CALL POKE Routine (p. 780)</td>
<td>Writes a value directly into memory on a 32-bit platform.</td>
</tr>
<tr>
<td></td>
<td>CALL POKELONG Routine (p. 781)</td>
<td>Writes a value directly into memory on 32-bit and 64-bit platforms.</td>
</tr>
<tr>
<td></td>
<td>CALL SYSTEM Routine (p. 844)</td>
<td>Submits an operating environment command for execution.</td>
</tr>
<tr>
<td></td>
<td>FMTINFO Function (p. 922)</td>
<td>Retrieves information about a format or informat.</td>
</tr>
<tr>
<td></td>
<td>PEEK Function (p. 1014)</td>
<td>Stores the contents of a memory address in a numeric variable on a 32-bit platform.</td>
</tr>
<tr>
<td></td>
<td>PEEKC Function (p. 1016)</td>
<td>Stores the contents of a memory address in a character variable on a 32-bit platform.</td>
</tr>
<tr>
<td></td>
<td>PEEKCLONG Function (p. 1017)</td>
<td>Stores the contents of a memory address in a character variable on 32-bit and 64-bit platforms.</td>
</tr>
<tr>
<td></td>
<td>PEEKLONG Function (p. 1018)</td>
<td>Stores the contents of a memory address in a numeric variable on 32-bit and 64-bit platforms.</td>
</tr>
<tr>
<td></td>
<td>PTRLONGADD Function (p. 1024)</td>
<td>Returns the pointer address as a character variable on 32-bit and 64-bit platforms.</td>
</tr>
<tr>
<td></td>
<td>PUTC Function (p. 1025)</td>
<td>Enables you to specify a character format at run time.</td>
</tr>
<tr>
<td></td>
<td>PUTN Function (p. 1028)</td>
<td>Enables you to specify a numeric format at run time.</td>
</tr>
<tr>
<td></td>
<td>SYSEXIST Function (p. 1072)</td>
<td>Returns a value that indicates whether an operating-environment variable exists in your environment.</td>
</tr>
<tr>
<td></td>
<td>SYSGET Function (p. 1073)</td>
<td>Returns the value of the specified operating environment variable.</td>
</tr>
<tr>
<td></td>
<td>SYSPARM Function (p. 1075)</td>
<td>Returns the system parameter string.</td>
</tr>
<tr>
<td></td>
<td>SYSPROCESSID Function (p. 1076)</td>
<td>Returns the process ID of the current process.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SYSPROCESSNAME Function (p. 1077)</td>
<td>Returns the process name that is associated with a given process ID, or returns the name of the current process.</td>
<td></td>
</tr>
<tr>
<td>SYSPROD Function (p. 1078)</td>
<td>Determines whether a product is licensed.</td>
<td></td>
</tr>
<tr>
<td>State and ZIP code</td>
<td>FIPNAME Function (p. 917)</td>
<td>Converts two-digit FIPS codes to uppercase state names.</td>
</tr>
<tr>
<td></td>
<td>FIPNAMEL Function (p. 918)</td>
<td>Converts two-digit FIPS codes to mixed case state names.</td>
</tr>
<tr>
<td></td>
<td>FIPSTATE Function (p. 920)</td>
<td>Converts two-digit FIPS codes to two-character state postal codes.</td>
</tr>
<tr>
<td></td>
<td>STFIPS Function (p. 1064)</td>
<td>Converts state postal codes to FIPS state codes.</td>
</tr>
<tr>
<td></td>
<td>STNAME Function (p. 1066)</td>
<td>Converts state postal codes to uppercase state names.</td>
</tr>
<tr>
<td></td>
<td>STNAMEL Function (p. 1067)</td>
<td>Converts state postal codes to mixed case state names.</td>
</tr>
<tr>
<td></td>
<td>ZIPCITY Function (p. 1098)</td>
<td>Returns a city name and the two-character postal code that corresponds to a ZIP code.</td>
</tr>
<tr>
<td></td>
<td>ZIPCITYDISTANCE Function (p. 1100)</td>
<td>Returns the geodetic distance between two ZIP code locations.</td>
</tr>
<tr>
<td></td>
<td>ZIPFIPS Function (p. 1101)</td>
<td>Converts ZIP codes to two-digit FIPS codes.</td>
</tr>
<tr>
<td></td>
<td>ZIPNAME Function (p. 1103)</td>
<td>Converts ZIP codes to uppercase state names.</td>
</tr>
<tr>
<td></td>
<td>ZIPNAMEL Function (p. 1105)</td>
<td>Converts ZIP codes to mixed case state names.</td>
</tr>
<tr>
<td></td>
<td>ZIPSTATE Function (p. 1106)</td>
<td>Converts ZIP codes to two-character state postal codes.</td>
</tr>
<tr>
<td>Variable Control</td>
<td>CALL SET Routine (p. 833)</td>
<td>Links SAS data set variables to DATA step or macro variables that have the same name and data type.</td>
</tr>
<tr>
<td>Web Service</td>
<td>SOAPWEB Function (p. 1046)</td>
<td>Calls a web service by using basic web authentication; credentials are provided in the arguments.</td>
</tr>
<tr>
<td></td>
<td>SOAPWEBMETA Function (p. 1048)</td>
<td>Calls a web service by using basic web authentication; credentials for the authentication domain are retrieved from metadata.</td>
</tr>
<tr>
<td></td>
<td>SOAPWIPSERVICE Function (p. 1050)</td>
<td>Calls a SAS registered web service by using WS-Security authentication; credentials are provided in the arguments.</td>
</tr>
<tr>
<td></td>
<td>SOAPWIPSRS Function (p. 1052)</td>
<td>Calls a SAS registered web service by using WS-Security authentication; credentials are provided in the arguments.</td>
</tr>
<tr>
<td></td>
<td>SOAPWS Function (p. 1055)</td>
<td>Calls a web service by using WS-Security authentication; credentials are provided in the arguments.</td>
</tr>
<tr>
<td></td>
<td>SOAPWSMETA Function (p. 1056)</td>
<td>Calls a web service by using WS-Security authentication; credentials for the provided authentication domain are retrieved from metadata.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Web Tools</td>
<td>HTMLDECODE Function (p. 967)</td>
<td>Decodes a string that contains HTML numeric character references or HTML character entity references, and returns the decoded string.</td>
</tr>
<tr>
<td></td>
<td>HTMLENCODE Function (p. 968)</td>
<td>Encodes characters using HTML character entity references, and returns the encoded string.</td>
</tr>
<tr>
<td></td>
<td>URLDECODE Function (p. 1087)</td>
<td>Returns a string that was decoded using the URL escape syntax.</td>
</tr>
<tr>
<td></td>
<td>URLENCODE Function (p. 1088)</td>
<td>Returns a string that was encoded using the URL escape syntax.</td>
</tr>
</tbody>
</table>

## Dictionary

### ADDR Function

Returns the memory address of a variable on a 32-bit platform.

**Category:** Special  
**Restrictions:** Use on 32-bit platforms only.  
This function is not valid on the CAS server.  
**Interaction:** When a SAS server is in a locked-down state, the ADDR function does not execute.

**Syntax**

ADDR(\textit{variable})

**Required Argument**

\textit{variable}  
specifies a variable name.

**Details**

The value that is returned is numeric. Because the storage location of a variable can vary from one execution to the next, the value that is returned by ADDR can vary. The ADDR function is used mostly in combination with the PEEK and PEEKC functions and the CALL POKE routine.

If you attempt to use the ADDR function on 64-bit platforms, SAS writes a message to the log stating that this restriction applies. If you have legacy applications that use ADDR, change the applications and use ADDRLONG instead. You can use ADDRLONG on 32-bit and 64-bit platforms.
Comparisons

The ADDR function returns the memory address of a variable on a 32-bit platform. ADDRLONG returns the memory address of a variable on 32-bit and 64-bit platforms.

Note: As a best practice, use ADDRLONG instead of ADDR because ADDRLONG can be used on 32-bit and 64-bit platforms.

Example

This example returns the address at which the variable FIRST is stored.

```
data numlist;
  first=3;
  x=addr(first);
run;
```

See Also

Functions:
- “ADDRLONG Function” on page 715
- “PEEK Function” on page 1014
- “PEEKC Function” on page 1016

CALL Routines:
- “CALL POKE Routine” on page 780

ADDRLONG Function

Returns the memory address of a variable on 32-bit and 64-bit platforms.

Category: Special

Restriction: This function is not valid on the CAS server.

Interaction: When a SAS server is in a locked-down state, the ADDRLONG function does not execute.

Syntax

```
ADDRLONG(variable)
```

Required Argument

variable

specifies a variable.

Details

The return value is a character string that contains the binary representation of the address. To display this value, use the $HEXw. format to convert the binary value to its hexadecimal equivalent. If you store the result in a variable, that variable should be a
character variable with a length of at least eight characters for portability. If you assign
the result to a variable that does not yet have a length defined, that variable is assigned a
length of 20 characters.

Example

This example returns the pointer address for the variable ITEM, and formats the value.

```
data characterlist;
  item=6345;
  x=addrlong(item);
  put x $hex16.;
run;
```

SAS writes the following results to the log:

```
480063B020202020
```

---

ALLCOMB Function

Generates all combinations of the values of \( n \) variables taken \( k \) at a time in a minimal change order.

**Category:** Combinatorial

**Restrictions:**
- This function is not valid on the CAS server.
- The ALLCOMB function cannot be executed when you use the %SYSFUNC macro.

**Syntax**

```
ALLCOMB(count, k, variable-1, ..., variable-n)
```

**Required Arguments**

- **count**
  - specifies an integer variable that is assigned values from 1 to the number of
    combinations in a loop.

- **k**
  - specifies an integer constant, variable, or expression between 1 and \( n \), inclusive, that
    specifies the number of items in each combination.

- **variable**
  - specifies either all numeric variables, or all character variables that have the same
    length. The values of these variables are permuted.

**Restriction**

Specify no more than 33 items. If you need to find combinations of more than 33 items, use the CALL ALLCOMBI routine.

**Requirement**

Initialize these variables before executing the ALLCOMB function.

**Tip**

After executing ALLCOMB, the first \( k \) variables contain the values in one combination.
Details

Use the ALLCOMB function in a loop where the first argument to ALLCOMB accepts each integral value from 1 to the number of combinations, and where \( k \) is constant. The number of combinations can be computed by using the COMB function. On the first execution, the argument types and lengths are checked for consistency. On each subsequent execution, the values of two variables are interchanged.

For the ALLCOMB function, the following actions occur:

- On the first execution, ALLCOMB returns 0.
- If the values of variable-\( i \) and variable-\( j \) were interchanged, where \( i < j \), then ALLCOMB returns \( i \).
- If no values were interchanged because all combinations were already generated, then ALLCOMB returns \(-1\).

If you execute the ALLCOMB function with the first argument out of sequence, the results are not useful. In particular, if you initialize the variables and then immediately execute the ALLCOMB function with a first argument of \( j \), then you do not get the \( j \)th combination (except when \( j \) is 1). To get the \( j \)th combination, you must execute ALLCOMB \( j \) times. The first argument takes values from 1 through \( j \) in that exact order.

Comparisons

SAS provides four functions or CALL routines for generating combinations:

- **ALLCOMB** generates all possible combinations of the values, missing or nonmissing, of \( N \) variables. The values can be any numeric or character values. Each combination is formed from the previous combination by removing one value and inserting another value.
- **LEXCOMB** generates all distinct combinations of the nonmissing values of several variables. The values can be any numeric or character values. The combinations are generated in lexicographic order.
- **ALLCOMBI** generates all combinations of the indices of \( N \) items, where indices are integers from 1 to \( N \). Each combination is formed from the previous combination by removing one index and inserting another index.
- **LEXCOMBI** generates all combinations of the indices of \( N \) items, where indices are integers from 1 to \( N \). The combinations are generated in lexicographic order.

ALLCOMBI is the fastest of these functions and CALL routines. LEXCOMB is the slowest.

Example

Here is an example of the ALLCOMB function.

```sas
data _null_;  
array x[5] $3 ('ant' 'bee' 'cat' 'dog' 'ewe');  
n=dim(x);  
k=3;  
ncomb=comb(n, k);  
do j=1 to ncomb+1;  
   rc=allcomb(j, k, of x[*]);  
   put j 5. +3 x1-x3 +3 rc=;  
end;  
run;
```
SAS writes the following output to the log:

```
1   ant bee cat    rc=0
2   ant bee ewe    rc=3
3   ant bee dog    rc=3
4   ant cat dog    rc=2
5   ant cat ewe    rc=3
6   ant dog ewe    rc=2
7   bee dog ewe    rc=1
8   bee dog cat    rc=3
9   bee ewe cat    rc=2
10  dog ewe cat    rc=1
11  dog ewe cat    rc=-1
```

See Also

CALL Routines:

- “CALL ALLCOMB Routine” on page 727

### ALLPERM Function

Generates all permutations of the values of several variables in a minimal change order.

**Category:** Combinatorial

**Restriction:** This function is not valid on the CAS server.

### Syntax

```
ALLPERM(count, variable-1 <,variable-2 ...>)
```

### Required Arguments

- **count**
  - specifies a variable with an integer value that ranges from 1 to the number of permutations.

- **variable**
  - specifies either all numeric variables, or all character variables that have the same length. The values of these variables are permuted.

**Restriction**

Specify no more than 18 variables.

**Requirement**

Initialize these variables before you execute the ALLPERM function.

### Details

**The Basics**

Use the ALLPERM function in a loop where the first argument to ALLPERM accepts each integral value from 1 to the number of permutations. On the first execution, the argument types and lengths are checked for consistency. On each subsequent execution, the values of two consecutive variables are interchanged.
Note: You can compute the number of permutations by using the PERM function. For more information, see the “PERM Function” on page 497. For the ALLPERM function, the following values are returned:

- 0 if count=1
- J if the values of variable-J and variable-K are interchanged, where K=J+1
- -1 if count>N!

If you use the ALLPERM function and the first argument is out of sequence, the results are not useful. For example, if you initialize the variables and then immediately execute the ALLPERM function with a first argument of K, your result is not the Kth permutation (except when K is 1). To get the Kth permutation, you must execute the ALLPERM function K times. The first argument takes values from 1 through K in that exact order.

ALLPERM always produces N! permutations even if some of the variables have equal values or missing values. If you want to generate only the distinct permutations when there are equal values, or if you want to omit missing values from the permutations, use the LEXPERM function instead.

Note: The ALLPERM function cannot be executed when you use the %SYSFUNC macro.

Comparisons

SAS provides three functions or CALL routines for generating all permutations:

- ALLPERM generates all possible permutations of the values, missing or non-missing, of several variables. Each permutation is formed from the previous permutation by interchanging two consecutive values.
- LEXPERM generates all distinct permutations of the non-missing values of several variables. The permutations are generated in lexicographic order.
- LEXPERK generates all distinct permutations of K of the non-missing values of N variables. The permutations are generated in lexicographic order.

ALLPERM is the fastest of these functions and CALL routines. LEXPERK is the slowest.

Example

The following example generates permutations of given values by using the ALLPERM function.

```sas
data _null_;
array x [4] $3 ('ant' 'bee' 'cat' 'dog');
array n dim(x);
array nfact fact(n);
do i=1 to nfact+1;
   change=allperm(i, of x[*]);
   put i 5. +2 change +2 x[*];
end;
run;
```
SAS writes the following output to the log:

```
1  0   ant bee cat dog
2  3   ant bee dog cat
3  2   ant dog bee cat
4  1   dog ant bee cat
5  3   dog ant cat bee
6  1   ant dog cat bee
7  2   ant cat dog bee
8  3   ant cat bee dog
9  1   cat ant bee dog
10 3   cat ant dog bee
11 2   cat dog ant bee
12 1   dog cat ant bee
13 3   dog cat bee ant
14 1   cat dog bee ant
15 2   cat bee dog ant
16 3   cat bee ant dog
17 1   bee cat ant dog
18 3   bee cat dog ant
19 2   bee dog cat ant
20 1   dog bee cat ant
21 3   dog bee ant cat
22 1   bee dog ant cat
23 2   bee ant dog cat
24 3   bee ant cat dog
25 -1   bee ant cat dog
```

See Also

Functions:
- “LEXPERM Function” on page 984

CALL Routines:
- “CALL ALLPERM Routine” on page 733
- “CALL RANPERK Routine” on page 829
- “CALL RANPERM Routine” on page 831

ATTRC Function

Returns the value of a character attribute for a SAS data set.

- **Category:** SAS File I/O
- **Restriction:** This function is not valid on the CAS server.

**Syntax**

```
ATTRC(data-set-id,attribute-name)
```

**Required Arguments**

- **data-set-id**
  specifies the data set identifier that the OPEN function returns.
**attribute-name**

is the name of a SAS data set attribute. If the value of *attribute-name* is invalid, a missing value is returned. Here is a list of SAS data set attribute names and their values:

**CHARSET**

returns a value for the character set of the computer that created the data set.

- empty string data set not sorted
- ASCII ASCII character set
- EBCDIC EBCDIC character set
- ANSI OS/2 ANSI standard ASCII character set
- OEM OS/2 OEM code format

**COMPRESS**

returns a value that specifies how a data set is compressed.

- CHAR specifies that the data set is compressed as character data.
- BINARY specifies that the data set is compressed as binary data.
- NO specifies that the data set is not compressed.

**DATAREP**

returns a value that indicates whether the data set is in a native format.

- NATIVE indicates that the data set is in native format.
- FOREIGN indicates that the data set is in a foreign format.

**ENCRYPT**

returns 'YES' or 'NO' depending on whether the SAS data set is encrypted.

**ENGINE**

returns the name of the engine that is used to access the data set.

**LABEL**

returns the label assigned to the data set.

**LIB**

returns the libref of the SAS library in which the data set resides.

**MEM**

returns the SAS data set name.

**MODE**

returns the mode in which the SAS data set was opened, such as:

- I INPUT mode allows random access if the engine supports it. Otherwise, it defaults to IN mode.
- IN INPUT mode reads sequentially and allows revisiting observations.
- IS INPUT mode reads sequentially but does not allow revisiting observations.
- N NEW mode creates a new data set.
- U UPDATE mode allows random access if the engine supports it. Otherwise, it defaults to UN mode.
- UN UPDATE mode reads sequentially and allows revisiting observations.
US  UPDATE mode reads sequentially but does not allow revisiting observations.

V  UTILITY mode allows modification of variable attributes and indexes associated with the data set.

MTYPE
   returns the SAS library member type.

SORTEDBY
   returns an empty string if the data set is not sorted. Otherwise, it returns the names of the BY variables in the standard BY statement format.

SORTLVL
   returns a value that indicates how a data set was sorted:

   Empty string  Data set is not sorted.
   WEAK          Sort order of the data set was established by the user (for example, through the SORTEDBY data set option). The system cannot validate its correctness, so the order of observations cannot be depended on.
   STRONG        Sort order of the data set was established by the software (for example, through PROC SORT or the OUT= option in the CONTENTS procedure).

SORTSEQ
   returns an empty string if the data set is sorted on the native computer or if the sort collating sequence is the default for the operating environment. Otherwise, it returns the name of the alternate collating sequence used to sort the file.

TYPE
   returns the SAS data set type.

Examples

Example 1: Writing a Message about Input Sequential Mode to the SAS Log
This example generates a message if the SAS data set has not been opened in INPUT SEQUENTIAL mode. The message is written to the SAS log as follows:

```sas
%let mode=%sysfunc(attrc(&dsid,MODE));
%if &mode ne IS %then
   %put Data set has not been opened in INPUT SEQUENTIAL mode.;
```

Example 2: Testing Whether a Data Set Has Been Sorted
This example tests whether a data set has been sorted and writes the result to the SAS log:

```sas
data _null_;
   dsid=open("sasdata.sortcars", "i");
   charset=attrc(dsid, "CHARSET");
   if charset = "" then
      put "Data set has not been sorted.";
   else put "Data set sorted with " charset "character set.";
   rc=close(dsid);
```
ATTRN Function

Returns the value of a numeric attribute for a SAS data set.

**Category:** SAS File I/O  
**Restriction:** This function is not valid on the CAS server.

**Syntax**

ATTRN(*data-set-id*,*attribute-name*)

**Required Arguments**

*data-set-id*

specifies the data set identifier that the OPEN function returns.

*attribute-name*

is the name of the SAS data set attribute whose numeric value is returned. If the value of *attribute-name* is invalid, a missing value is returned. Here is a list of SAS data set attribute names and their values:

- **ALTERPW**
  
  specifies whether a password is required to alter the data set.
  
  1 the data set is Alter protected.
  
  0 the data set is not Alter protected.

- **ANOBS**
  
  specifies whether the engine knows the number of observations.
  
  1 the engine knows the number of observations.
  
  0 the engine does not know the number of observations.

- **ANY**
  
  specifies whether the data set has observations or variables.
  
  −1 the data set has no observations or variables.
  
  0 the data set has no observations.
  
  1 the data set has observations and variables.

**Alias**

VAROBS

ARAND

specifies whether the engine supports random access.

See Also

**Functions**

- “ATTRN Function” on page 723
- “OPEN Function” on page 1010
the engine supports random access.
0 the engine does not support random access.

Alias RANDOM

ARWU
specifies whether the engine can manipulate files.
1 the engine is not read-only. It can create or update SAS files.
0 the engine is read-only.

AUDIT
specifies whether logging to an audit file is enabled.
1 logging is enabled.
0 logging is suspended.

AUDIT_DATA
specifies whether after-update record images are stored.
1 after-update record images are stored.
0 after-update record images are not stored.

AUDIT BEFORE
specifies whether before-update record images are stored.
1 before-update record images are stored.
0 before-update record images are not stored.

AUDIT_ERROR
specifies whether unsuccessful after-update record images are stored.
1 unsuccessful after-update record images are stored.
0 unsuccessful after-update record images are not stored.

CRDTE
specifies the date on which the data set was created. The value that is returned is the internal SAS datetime value for the creation date.

Tip Use the DATETIME. format to display this value.

ICONST
returns information about the existence of integrity constraints for a SAS data set.
0 no integrity constraints.
1 one or more general integrity constraints.
2 one or more referential integrity constraints.
3 both one or more general integrity constraints and one or more referential integrity constraints.

INDEX
specifies whether the data set supports indexing.
1 indexing is supported.
indexing is not supported.

**ISINDEX**
specifies whether the data set is indexed.

- 1 at least one index exists for the data set.
- 0 the data set is not indexed.

**ISSUBSET**
specifies whether the data set is a subset.

- 1 at least one WHERE clause is active.
- 0 no WHERE clause is active.

**LRECL**
specifies the logical record length.

**LRID**
specifies the length of the record ID.

**MAXGEN**
specifies the maximum number of generations.

**MAXRC**
specifies whether an application checks return codes.

- 1 an application checks return codes.
- 0 an application does not check return codes.

**MODTE**
specifies the last date and time that the data set was modified. The value returned is the internal SAS datetime value.

**Tip** Use the DATETIME. format to display this value.

**NDEL**
specifies the number of observations in the data set that are marked for deletion.

**NEXTGEN**
specifies the next generation number to generate.

**NLOBS**
specifies the number of logical observations (the observations that are not marked for deletion). An active WHERE clause does not affect this number.

- -1 the number of observations is not available.

**NLOBSF**
specifies the number of logical observations (the observations that are not marked for deletion) by forcing each observation to be read and by taking the FIRSTOBS system option, the OBS system option, and the WHERE clauses into account.

**Tip** Passing NLOBSF to ATTRN requires the engine to read every observation from the data set that matches the WHERE clause. Based on the file type and file size, reading these observations can be a time-consuming process.
NOBS
specifies the number of physical observations (including the observations that are marked for deletion). An active WHERE clause does not affect this number.

-1  the number of observations is not available.

NVARS
specifies the number of variables in the data set.

PW
specifies whether a password is required to access the data set.

1  the data set is protected.
0  the data set is not protected.

RADIX
specifies whether access by observation number (radix addressability) is allowed.

1  access by observation number is allowed.
0  access by observation number is not allowed.

Note: A data set that is accessed by a tape engine is index addressable although it cannot be accessed by an observation number.

READPW
specifies whether a password is required to read the data set.

1  the data set is Read protected.
0  the data set is not Read protected.

REUSE
specifies whether new observations can be written to free space in compressed SAS data sets.

1  free space can be reused.
0  free space cannot be reused.

TAPE
specifies the status of the data set tape.

1  the data set is a sequential file.
0  the data set is not a sequential file.

WHSTMT
specifies the active WHERE clauses.

0  no WHERE clause is active.
1  a permanent WHERE clause is active.
2  a temporary WHERE clause is active.
3  both permanent and temporary WHERE clauses are active.

WRITEPW
specifies whether a password is required to write to the data set.

1  the data set is Write protected.
0  the data set is not Write protected.
Examples

**Example 1: Checking for an Active WHERE Clause**
This example checks whether a WHERE clause is currently active for a data set.

```sas
%let iswhere=%sysfunc(attrn(&dsid,whstmt));
%if &iswhere %then
  %put A WHERE clause is currently active.;
```

**Example 2: Checking for an Indexed Data Set**
This example checks whether a data set is indexed.

```sas
data _null_;  
dsid=open("mydata");  
isindex=attrn(dsid, "isindex");  
if isindex then put "data set is indexed";
else put "data set is not indexed";
run;
```

**Example 3: Checking a Data Set for Password Protection**
This example checks whether a data set is protected with a password.

```sas
data _null_;  
dsid=open("mydata");  
pw=attrn(dsid, "pw");  
if pw then put "data set is protected";
run;
```

**See Also**

Functions:

- “ATTRC Function” on page 720
- “OPEN Function” on page 1010

---

**CALL ALLCOMB Routine**

Generates all combinations of the values of \( n \) variables taken \( k \) at a time in a minimal change order.

**Function Details**

- **Category:** Combinatorial
- **Restriction:** This function is not valid on the CAS server.
- **Note:** Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

**Syntax**

```
CALL ALLCOMB(count, k, variable-1, ..., variable-n);
```
Required Arguments

**count**
- specifies an integer variable that is assigned from 1 to the number of combinations in a loop.

**k**
- specifies an integer constant, variable, or expression between 1 and \( n \), inclusive, that specifies the number of items in each combination.

**variable**
- specifies either all numeric variables, or all character variables that have the same length. The values of these variables are permuted.

**Restriction**
- Specify no more than 33 items. If you need to find combinations of more than 33 items, use the CALL ALLCOMBI routine.

**Requirement**
- Initialize these variables before calling the ALLCOMB routine.

**Tip**
- After calling the ALLCOMB routine, the first \( k \) variables contain the values in one combination.

Details

**CALL ALLCOMB Processing**
Use the CALL ALLCOMB routine in a loop where the first argument to CALL ALLCOMB accepts each integral value from 1 to the number of combinations, and where \( k \) is constant. The number of combinations can be computed by using the COMB function. On the first call, the argument types and lengths are checked for consistency. On each subsequent call, the values of two variables are interchanged.

If you call the ALLCOMB routine with the first argument out of sequence, the results are not useful. In particular, if you initialize the variables and then immediately call ALLCOMB with a first argument of \( j \), then you do not get the \( j \)th combination (except when \( j = 1 \)). To get the \( j \)th combination, you must call ALLCOMB \( j \) times. The first argument takes values from 1 through \( j \) in that exact order.

**Using the CALL ALLCOMB Routine with Macros**
You can call the ALLCOMB routine when you use the %SYSCALL macro. In this case, the variable arguments are not required to be the same type or length. If %SYSCALL identifies an argument as numeric, then %SYSCALL reformats the returned value.

If an error occurs during the execution of the CALL ALLCOMB routine, then both of the following values are set:

- &SYSERR is assigned a value that is greater than 4.
- &SYSINFO is assigned a value that is less than –100.

If there are no errors, then &SYSERR is set to zero, and &SYSINFO is set to one of the following values:

- 0 if \( count = 1 \)
- \( j \) if the values of variable-\( j \) and variable-\( k \) were interchanged, where \( j < k \)
- –1 if no values were interchanged because all distinct combinations were already generated
Comparisons

SAS provides four functions or CALL routines for generating combinations:

- **ALLCOMB** generates all possible combinations of the values, missing or nonmissing, of n variables. The values can be any numeric or character values. Each combination is formed from the previous combination by removing one value and inserting another value.

- **LEXCOMB** generates all distinct combinations of the nonmissing values of several variables. The values can be any numeric or character values. The combinations are generated in lexicographic order.

- **ALLCOMBI** generates all combinations of the indices of n items, where indices are integers from 1 to n. Each combination is formed from the previous combination by removing one index and inserting another index.

- **LEXCOMBI** generates all combinations of the indices of n items, where indices are integers from 1 to n. The combinations are generated in lexicographic order.

ALLCOMBI is the fastest of these functions and CALL routines. LEXCOMB is the slowest.

Examples

**Example 1: Using CALL ALLCOMB in a DATA Step**

The following is an example of the CALL ALLCOMB routine that is used with the DATA step.

```sas
data _null_;
  array x[5] $3 ('ant' 'bee' 'cat' 'dog' 'ewe');
  n=dim(x);
  k=3;
  ncomb=comb(n, k);
  do j=1 to ncomb+1;
    call allcomb(j, k, of x[*]);
    put j 5. +3 x1-x3;
  end;
run;
```

SAS writes the following output to the log:

```
1   ant bee cat
2   ant bee ewe
3   ant bee dog
4   ant cat dog
5   ant cat ewe
6   ant dog ewe
7   bee dog ewe
8   bee dog cat
9   bee ewe cat
10  dog ewe cat
11  dog ewe cat
```

**Example 2: Using CALL ALLCOMB with Macros and Displaying the Return Code**

The following is an example of the CALL ALLCOMB routine that is used with macros. The output includes values for the %SYSINFO macro.
SAS writes the following output to the log:

1: ant -0.1234 sysinfo=0
2: ant zebra sysinfo=2
3: ant hippopotamus sysinfo=2
4: ant 10000000000 sysinfo=2
5: -0.1234 10000000000 sysinfo=1
6: -0.1234 zebra sysinfo=2
7: -0.1234 hippopotamus sysinfo=2
8: 10000000000 hippopotamus sysinfo=1
9: 10000000000 zebra sysinfo=2
10: hippopotamus zebra sysinfo=1
11: hippopotamus zebra sysinfo=-1

See Also

Functions:
- “ALLCOMB Function” on page 716

CALL ALLCOMBI Routine

Generates all combinations of the indices of n objects taken k at a time in a minimal change order.

**Category:** Combinatorial

**Restriction:** This function is not valid on the CAS server.

**Note:** Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

**Syntax**

```
CALL ALLCOMBI(N, K, index-1, ..., index-K, <, index-added, index-removed>);
```
**Required Arguments**

\( N \)

is a numeric constant, variable, or expression that specifies the total number of objects.

\( K \)

is a numeric constant, variable, or expression that specifies the number of objects in each combination.

\( \text{index} \)

is a numeric variable that contains indices of the objects in the returned combination. Indices are integers between 1 and \( N \) inclusive.

**Tip**
If \( \text{index-}1 \) is missing or zero, then ALLCOMBI initializes the indices to \( \text{index-}1=1 \) through \( \text{index-}K=K \). Otherwise, ALLCOMBI creates a new combination by removing one index from the combination and adding another index.

**Optional Arguments**

\( \text{index-added} \)

is a numeric variable in which ALLCOMBI returns the value of the index that was added.

\( \text{index-removed} \)

is a numeric variable in which ALLCOMBI returns the value of the index that was removed.

**Details**

**CALL ALLCOMBI Processing**

Before you make the first call to ALLCOMBI, complete one of the following tasks:

- Set \( \text{index-}1 \) equal to zero or to a missing value.
- Initialize \( \text{index-}1 \) through \( \text{index-}K \) to distinct integers between 1 and \( N \) inclusive.

The number of combinations of \( N \) objects taken \( K \) at a time can be computed as \( \text{COMB}(N,K) \). To generate all combinations of \( N \) objects taken \( K \) at a time, call ALLCOMBI in a loop that executes \( \text{COMB}(N,K) \) times.

**Using the CALL ALLCOMBI Routine with Macros**

If you call ALLCOMBI from the macro processor with \%SYSCALL, then you must initialize all arguments to numeric values. \&SYSCALL reformats the values that are returned.

If an error occurs during the execution of the CALL ALLCOMBI routine, then both of the following values are set:

- \&SYSERR is assigned a value that is greater than 4.
- \&SYSINFO is assigned a value that is less than –100.

If there are no errors, then \&SYSERR and \&SYSINFO are set to zero.

**Comparisons**

The CALL ALLCOMBI routine generates all combinations of the indices of \( N \) objects taken \( K \) at a time in a minimal change order. The CALL ALLCOMB routine generates
all combinations of the values of N variables taken K at a time in a minimal change order.

Examples

Example 1: Using CALL ALLCOMBI in a DATA Step

The following is an example of the CALL ALLCOMBI routine that is used in a DATA step.

```sas
data _null_;
array x[5] $3 ('ant' 'bee' 'cat' 'dog' 'ewe');
array c[3] $3;
array i[3];
n=dim(x);
k=dim(i);
i[1]=0;
ncomb=comb(n, k);   /* The one extra call goes back */
do j=1 to ncomb+1;  /* to the first combination. */
   call allcombi(n, k, of i[*], add, remove);
   do h=1 to k;
      c[h]=x[i[h]];
   end;
   put @4 j= @10 'i= ' i[*] +3 'c= ' c[*] +3 add= remove=;
end;
run;
```

SAS writes the following output to the log:

```
j=1   i= 1 2 3   c= ant bee cat   add=0 remove=0
j=2   i= 1 3 4   c= ant cat dog   add=4 remove=2
j=3   i= 2 3 4   c= bee cat dog   add=2 remove=1
j=4   i= 1 2 4   c= ant bee dog   add=1 remove=3
j=5   i= 1 4 5   c= ant dog ewe   add=5 remove=2
j=6   i= 2 4 5   c= bee dog ewe   add=2 remove=1
j=7   i= 3 4 5   c= cat dog ewe   add=3 remove=2
j=8   i= 1 3 5   c= ant cat ewe   add=1 remove=4
j=9   i= 2 3 5   c= bee cat ewe   add=2 remove=1
j=10  i= 1 2 5   c= ant bee ewe   add=1 remove=3
j=11  i= 1 2 3   c= ant bee cat   add=3 remove=5
```

Example 2: Using CALL ALLCOMBI with Macros

The following is an example of the CALL ALLCOMBI routine that is used with macros.

```sas
%macro test;
%let x1=0;
%let x2=0;
%let x3=0;
%let add=0;
%let remove=0;
%let n=5;
%let k=3;
%let ncomb=%sysfunc(comb(&n, &k));
%do j=1 %to &ncomb;
   %syscall allcombi(n, k, x1, x2, x3, add, remove);
   %let jfmt=%qsysfunc(putn(&j, 5.));
   %put &jfmt: &x1 &x2 &x3 add= &add= remove=&remove;
%end;
%mend test;
```

```
test;
```

SAS writes the following output to the log:

```
j=1   x1= 0 x2= 0 x3= 0 add= 0 remove= 0
j=2   x1= 1 x2= 0 x3= 0 add= 4 remove= 2
j=3   x1= 0 x2= 1 x3= 0 add= 2 remove= 1
j=4   x1= 1 x2= 1 x3= 0 add= 1 remove= 3
j=5   x1= 0 x2= 0 x3= 1 add= 5 remove= 2
j=6   x1= 1 x2= 2 x3= 0 add= 2 remove= 1
j=7   x1= 0 x2= 1 x3= 1 add= 3 remove= 2
j=8   x1= 1 x2= 0 x3= 1 add= 1 remove= 4
j=9   x1= 0 x2= 1 x3= 1 add= 2 remove= 1
j=10  x1= 0 x2= 0 x3= 1 add= 1 remove= 3
j=11  x1= 0 x2= 0 x3= 1 add= 3 remove= 5
```
SAS writes the following output to the log:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>add</th>
<th>remove</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>

See Also

CALL Routines:

- “CALL ALLCOMB Routine” on page 727

CALL ALLPERM Routine

Generates all permutations of the values of several variables in a minimal change order.

- **Category:** Combinatorial
- **Restriction:** This function is not valid on the CAS server.
- **Note:** Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

**Syntax**

```
CALL ALLPERM(count, variable-1 <, variable-2 ...>);
```

**Required Arguments**

- **count**
  - specifies an integer variable that ranges from 1 to the number of permutations.
- **variable**
  - specifies either all numeric variables, or all character variables that have the same length. The values of these variables are permuted.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Specify no more than 18 variables.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>Initialize these variables before you call the ALLPERM routine.</td>
</tr>
</tbody>
</table>

**Details**

**CALL ALLPERM Processing**

Use the CALL ALLPERM routine in a loop where the first argument to CALL ALLPERM takes each integral value from 1 to the number of permutations. On the first
call, the argument types and lengths are checked for consistency. On each subsequent
call, the values of two consecutive variables are interchanged.

Note: You can compute the number of permutations by using the PERM function. See
PERM Function on page 497 for more information.

If you call the ALLPERM routine and the first argument is out of sequence, the results
are not useful. In particular, if you initialize the variables and then immediately call the
ALLPERM routine with a first argument of K, your result is not the Kth permutation
(except when K is 1). To get the Kth permutation, you must call the ALLPERM routine
K times. The first argument takes values from 1 through K in that exact order.

ALLPERM always produces N! permutations even if some of the variables have equal
values or missing values. If you want to generate only the distinct permutations when
there are equal values, or if you want to omit missing values from the permutations, use
the LEXPERM function instead.

**Using the CALL ALLPERM Routine with Macros**

You can call the ALLPERM routine when you use the %SYSCALL macro. In this case,
the variable arguments are not required to be the same type or length. If %SYSCALL
identifies an argument as numeric, then %SYSCALL reformats the returned value.

If an error occurs during the execution of the CALL ALLPERM routine, then both of the
following values are set:

- &SYSERR is assigned a value that is greater than 4.
- &SYSINFO is assigned a value that is less than -100.

If there are no errors, then &SYSERR is set to zero, and &SYSINFO is set to one of the
following values:

- 0 if count=1
- 1 if 1<count<=N! and the values of variable-J and variable-K were interchanged,
  where J+1=K
- -1 if count>N!

**Comparisons**

SAS provides three functions or CALL routines for generating all permutations:

- ALLPERM generates all possible permutations of the values, missing or nonmissing,
  of several variables. Each permutation is formed from the previous permutation by
  interchanging two consecutive values.

- LEXPERM generates all distinct permutations of the nonmissing values of several
  variables. The permutations are generated in lexicographic order.

- LEXPERK generates all distinct permutations of K of the nonmissing values of N
  variables. The permutations are generated in lexicographic order.

ALLPERM is the fastest of these functions and CALL routines. LEXPERK is the
slowest.

**Examples**

**Example 1: Using CALL ALLPERM in a DATA Step**

The following example generates permutations of given values by using the CALL
ALLPERM routine.
data _null_; array x [4] $3 ('ant' 'bee' 'cat' 'dog'); n=dim(x); nfact=fact(n); do i=1 to nfact; call allperm(i, of x[*]); put i 5. +2 x[*]; end; run;

SAS writes the following output to the log:

```
1  ant bee cat dog
2  ant bee dog cat
3  ant dog bee cat
4  dog ant bee cat
5  dog ant cat bee
6  ant dog cat bee
7  ant cat dog bee
8  ant cat bee dog
9  cat ant bee dog
10  cat ant dog bee
11  cat dog ant bee
12  dog cat ant bee
13  dog cat bee ant
14  cat dog bee ant
15  cat bee dog ant
16  cat bee ant dog
17  bee cat ant dog
18  bee cat dog ant
19  bee dog cat ant
20  dog bee cat ant
21  dog bee ant cat
22  bee dog ant cat
23  bee ant dog cat
24  bee ant cat dog
```

**Example 2: Using CALL ALLPERM with Macros**

Here is an example of the CALL ALLPERM routine that is used with macros. The output includes values for the %SYSINFO macro.

```
%macro test;
%let x1=ant;
%let x2=-1.234;
%let x3=1e10;
%let x4=hippopotamus;
%let nperm=%sysfunc(perm(4));
%do j=1 %to &nperm+1;
  %syscall allperm(j, &x1, &x2, &x3, &x4);
  %let jfmt=%qsysfunc(putn(&j, 5.));
  %put &jfmt: &x1 &x2 &x3 &x4 sysinfo=&sysinfo;
%end;
%mend;

%test;
```
SAS writes the following output to the log:

```
1:   ant -0.1234 10000000000 hippopotamus sysinfo=0
2:   ant -0.1234 hippopotamus 10000000000 sysinfo=3
3:   ant hippopotamus -0.1234 10000000000 sysinfo=2
4:   hippopotamus ant -0.1234 10000000000 sysinfo=1
5:   hippopotamus ant 10000000000 -0.1234 sysinfo=3
6:   ant hippopotamus 10000000000 -0.1234 sysinfo=1
7:   ant 10000000000 hippopotamus -0.1234 sysinfo=2
8:   ant 10000000000 -0.1234 hippopotamus sysinfo=3
9:   10000000000 ant -0.1234 hippopotamus sysinfo=1
10:   10000000000 ant hippopotamus -0.1234 sysinfo=3
11:   10000000000 hippopotamus ant -0.1234 sysinfo=2
12:   hippopotamus 10000000000 ant -0.1234 sysinfo=1
13:   hippopotamus 10000000000 -0.1234 ant sysinfo=3
14:   10000000000 hippopotamus -0.1234 ant sysinfo=1
15:   10000000000 -0.1234 hippopotamus ant sysinfo=2
16:   -0.1234 10000000000 ant hippopotamus sysinfo=1
17:   -0.1234 10000000000 hippopotamus ant sysinfo=3
18:   -0.1234 hippopotamus 10000000000 ant sysinfo=2
19:   hippopotamus -0.1234 10000000000 ant sysinfo=1
20:   -0.1234 hippopotamus 10000000000 ant sysinfo=2
21:   hippopotamus -0.1234 ant 10000000000 sysinfo=3
22:   -0.1234 hippopotamus ant 10000000000 sysinfo=1
23:   -0.1234 ant hippopotamus 10000000000 sysinfo=2
24:   -0.1234 ant 10000000000 hippopotamus sysinfo=3
25:   -0.1234 ant 10000000000 hippopotamus sysinfo=-1
```

### See Also

**Functions:**

- “ALLPERM Function” on page 718
- “LEXPERM Function” on page 984

**CALL Routines:**

- “CALL RANPERK Routine” on page 829
- “CALL RANPERM Routine” on page 831

### CALL CATS Routine

Removes leading and trailing blanks, and returns a concatenated character string.

**Category:** Character

**Restriction:** This function is not valid on the CAS server.

**Note:** Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

### Syntax

```
CALL CATS(result <, item-1, ..., item-n>);
```
**Required Argument**

**result**

specifies a character variable.

**Restriction**

The CALL CATS routine accepts only a character variable as a valid argument for `result`. Do not use a constant or a SAS expression because CALL CATS is unable to update these arguments.

---

**Optional Argument**

**item**

specifies a constant, variable, or expression, either character or numeric. If `item` is numeric, then its value is converted to a character string using the `BESTw.` format. In this case, SAS does not write a note to the log.

---

**Details**

The CALL CATS routine returns the result in the first argument, `result`. The routine appends the values of the arguments that follow to `result`. If the length of `result` is not large enough to contain the entire result, SAS does the following:

- writes a warning message to the log stating that the result was truncated
- writes a note to the log that shows the location of the function call and lists the argument that caused the truncation, except in SQL or in a WHERE clause
- sets `_ERROR_` to 1 in the DATA step, except in a WHERE clause

The CALL CATS routine removes leading and trailing blanks from numeric arguments after it formats the numeric value with the `BESTw.` format.

---

**Comparisons**

The results of the CALL CATS, CALL CATT, and CALL CATX routines are usually equivalent to statements that use the concatenation operator (||) and the TRIM and LEFT functions. However, using the CALL CATS, CALL CATT, and CALL CATX routines is faster than using TRIM and LEFT.

The following table shows statements that are equivalent to CALL CATS, CALL CATT, and CALL CATX. The variables X1 through X4 specify character variables, and SP specifies a separator, such as a blank or comma.

<table>
<thead>
<tr>
<th>CALL Routine</th>
<th>Equivalent Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL CATS (OF X1-X4);</td>
<td>X1=TRIM(LEFT(X1))</td>
</tr>
<tr>
<td>CALL CATT (OF X1-X4);</td>
<td>X1=TRIM(X1)</td>
</tr>
<tr>
<td>CALL CATX (SP, OF X1-X4); *</td>
<td>X1=TRIM(LEFT(X1))</td>
</tr>
</tbody>
</table>

*Note:* If any of the arguments is blank, the results that are produced by CALL CATX differ slightly from the results that are produced by the concatenated code. In this case, CALL CATX omits the corresponding separator. For example, **CALL CATX ("+", "X", " ", "Z", " ");** produces `X+Z`. 
Example

The following example shows how the CALL CATS routine concatenates strings.

```sas
data _null_;  
  length answer $ 36;  
  x='Athens is t  ';  
  y=' he Olym  ';  
  z='   pic site for 2004. ';  
  call cats(answer, x, y, z);  
  put answer;  
run;
```

SAS writes the following output to the log:

```
Athens is the Olympic site for 2004.
```

See Also

Functions:
- “CAT Function” on page 143
- “CATQ Function” on page 846
- “CATS Function” on page 145
- “CATT Function” on page 147
- “CATX Function” on page 149

CALL Routines:
- “CALL CATT Routine” on page 738
- “CALL CATX Routine” on page 740

CALL CATT Routine

Removes trailing blanks, and returns a concatenated character string.

**Category:** Character

**Restriction:** This function is not valid on the CAS server.

**Note:** Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

**Syntax**

```
CALL CATT(result <, item-1, ..., item-n>);
```

**Required Argument**

`result`

specifies a character variable.
Restriction  The CALL CATT routine accepts only a character variable as a valid argument for `result`. Do not use a constant or a SAS expression because CALL CATT is unable to update these arguments.

Optional Argument

`item` specifies a constant, variable, or expression, either character or numeric. If `item` is numeric, then its value is converted to a character string using the `BESTw.` format. In this case, leading blanks are removed and SAS does not write a note to the log.

Details

The CALL CATT routine returns the result in the first argument, `result`. The routine appends the values of the arguments that follow to `result`. If the length of `result` is not large enough to contain the entire result, SAS does the following:

- writes a warning message to the log stating that the result was truncated
- writes a note to the log that shows the location of the function call and lists the argument that caused the truncation, except in SQL or in a WHERE clause
- sets `_ERROR_` to 1 in the DATA step, except in a WHERE clause

The CALL CATT routine removes leading and trailing blanks from numeric arguments after it formats the numeric value with the `BESTw.` format.

Comparisons

The results of the CALL CATS, CALL CATT, and CALL CATX routines are usually equivalent to statements that use the concatenation operator (||) and the TRIM and LEFT functions. However, using the CALL CATS, CALL CATT, and CALL CATX routines is faster than using TRIM and LEFT.

The following table shows statements that are equivalent to CALL CATS, CALL CATT, and CALL CATX. The variables X1 through X4 specify character variables, and SP specifies a separator, such as a blank or comma.

<table>
<thead>
<tr>
<th>CALL Routine</th>
<th>Equivalent Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL CATS(OF X1-X4);</td>
<td>`X1=TRIM(LEFT(X1))</td>
</tr>
<tr>
<td>CALL CATT(OF X1-X4);</td>
<td>`X1=TRIM(X1)</td>
</tr>
<tr>
<td>CALL CATX(SP, OF X1-X4); *</td>
<td>`X1=TRIM(LEFT(X1))</td>
</tr>
</tbody>
</table>

Note: If any of the arguments is blank, the results that are produced by CALL CATX differ slightly from the results that are produced by the concatenated code. In this case, CALL CATX omits the corresponding separator. For example, `CALL CATX ("+","X"," ", "Z"," ");` produces `X+Z`.

Example

The following example shows how the CALL CATT routine concatenates strings.
data _null_;  
   length answer $ 36;  
   x='London is t  ';  
   y='he Olym  ';  
   z='pic site for 2012. ';  
   call catt(answer, x, y, z);  
   put answer;  
run;  

SAS writes the following output to the log:

   London is the Olympic site for 2012.

See Also

Functions:
- “CAT Function” on page 143
- “CATQ Function” on page 846
- “CATS Function” on page 145
- “CATT Function” on page 147
- “CATX Function” on page 149

CALL Routines:
- “CALL CATS Routine” on page 736
- “CALL CATX Routine” on page 740

---

CALL CATX Routine

Removes leading and trailing blanks, inserts delimiters, and returns a concatenated character string.

**Category:** Character

**Restriction:** This function is not valid on the CAS server.

**Note:** Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

**Syntax**

`CALL CATX(delimiter, result <, item-1 , ..., item-n>);`

**Required Arguments**

- `delimiter`
  - specifies a character string that is used as a delimiter between concatenated strings.

- `result`
  - specifies a character variable.
Restriction

The CALL CATX routine accepts only a character variable as a valid argument for `result`. Do not use a constant or a SAS expression because CALL CATX is unable to update these arguments.

Optional Argument

`item`

specifies a constant, variable, or expression, either character or numeric. If `item` is numeric, then its value is converted to a character string using the BESTw. format. In this case, SAS does not write a note to the log.

Details

The CALL CATX routine returns the result in the second argument, `result`. The routine appends the values of the arguments that follow to `result`. If the length of `result` is not large enough to contain the entire result, SAS does the following:

- writes a warning message to the log stating that the result was truncated
- writes a note to the log that shows the location of the function call and lists the argument that caused the truncation, except in SQL or in a WHERE clause
- sets `_ERROR_` to 1 in the DATA step, except in a WHERE clause

The CALL CATX routine removes leading and trailing blanks from numeric arguments after formatting the numeric value with the BESTw. format.

Comparisons

The results of the CALL CATS, CALL CATT, and CALL CATX routines are usually equivalent to statements that use the concatenation operator (||) and the TRIM and LEFT functions. However, using the CALL CATS, CALL CATT, and CALL CATX routines is faster than using TRIM and LEFT.

The following table shows statements that are equivalent to CALL CATS, CALL CATT, and CALL CATX. The variables `X1` through `X4` specify character variables, and `SP` specifies a delimiter, such as a blank or comma.

<table>
<thead>
<tr>
<th>CALL Routine</th>
<th>Equivalent Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL CATS(OF X1-X4);</td>
<td>`X1=TRIM(LEFT(X1))</td>
</tr>
<tr>
<td>CALL CATT(OF X1-X4);</td>
<td>`X1=TRIM(X1)</td>
</tr>
<tr>
<td>CALL CATX(SP, OF X1-X4); *</td>
<td>`X1=TRIM(LEFT(X1))</td>
</tr>
</tbody>
</table>

Note: If any of the arguments are blank, the results that are produced by CALL CATX differ slightly from the results that are produced by the concatenated code. In this case, CALL CATX omits the corresponding delimiter. For example, `CALL CATX("+", newvar, "X"," ", "Z"," ");` produces `X+Z`.

Example

The following example shows how the CALL CATX routine concatenates strings.
data _null_;  
  length answer $ 50;  
  separator='%%$%%';  
  x='Athens is t';  
  y='he Olym';  
  z='pic site for 2004. ';  
  call catx(separator, answer, x, y, z);  
  put answer;  
run;

SAS writes the following output to the log:

Athens is t%%$%%he Olym%%$%%pic site for 2004.

See Also

Functions:
- “CAT Function” on page 143
- “CATQ Function” on page 846
- “CATS Function” on page 145
- “CATT Function” on page 147
- “CATX Function” on page 149

CALL Routines:
- “CALL CATS Routine” on page 736
- “CALL CATT Routine” on page 738

CALL COMPCOST Routine

Sets the costs of operations for later use by the COMPGED function

Category: Character

Restrictions: Use with the COMPGED function
This function is not valid on the CAS server.

Interaction: When invoked by the %SYSCALL macro statement, CALL COMPCOST removes quotation marks from its arguments. For more information, see “Using CALL Routines and the %SYSCALL Macro Statement” on page 9.

Note: Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

Syntax

CALL COMPCOST(operation-1, value-1 <, operation-2, value-2 ..>);
**Required Arguments**

*operation*  
is a character constant, variable, or expression that specifies an operation that is performed by the COMPGED function.

*value*  
is a numeric constant, variable, or expression that specifies the cost of the operation that is indicated by the preceding argument.

**Restriction**  
Must be an integer that ranges from –32767 through 32767, or a missing value

**Details**

**Computing the Cost of Operations**

Each argument that specifies an operation must have a value that is a character string. The character string corresponds to one of the terms that are used to denote an operation that the COMPGED function performs. See “Computing the Generalized Edit Distance” on page 857 to view a table of operations that the COMPGED function uses.

The character strings that specify operations can be in uppercase, lowercase, or mixed case. Blanks are ignored. Each character string must end with an equal sign (=). Valid values for operations, and the default cost of the operations are listed in the following table.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Default Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPEND=</td>
<td>very large</td>
</tr>
<tr>
<td>BLANK=</td>
<td>very large</td>
</tr>
<tr>
<td>DELETE=</td>
<td>100</td>
</tr>
<tr>
<td>DOUBLE=</td>
<td>very large</td>
</tr>
<tr>
<td>FDELETE=</td>
<td>equal to DELETE</td>
</tr>
<tr>
<td>FINSERT=</td>
<td>equal to INSERT</td>
</tr>
<tr>
<td>FREPLACE=</td>
<td>equal to REPLACE</td>
</tr>
<tr>
<td>INSERT=</td>
<td>100</td>
</tr>
<tr>
<td>MATCH=</td>
<td>0</td>
</tr>
<tr>
<td>PUNCTUATION=</td>
<td>very large</td>
</tr>
<tr>
<td>REPLACE=</td>
<td>100</td>
</tr>
<tr>
<td>SINGLE=</td>
<td>very large</td>
</tr>
<tr>
<td>SWAP=</td>
<td>very large</td>
</tr>
</tbody>
</table>
If an operation does not appear in the call to the COMPCOST routine, or if the operation appears and is followed by a missing value, then that operation is assigned a default cost.
A “very large” cost indicates a cost that is sufficiently large that the COMPGED function does not use the corresponding operation.

After your program calls the COMPCOST routine, the costs that are specified remain in effect until your program calls the COMPCOST routine again, or until the step that contains the call to COMPCOST terminates.

**Abbreviating Character Strings**
You can abbreviate character strings. That is, you can use the first one or more letters of a specific operation rather than use the entire term. You must, however, use as many letters as necessary to uniquely identify the term. For example, you can specify the INSERT= operation as “in=”, and the REPLACE= operation as “r=”. To specify the DELETE= operation or the DOUBLE= operation, you must use the first two letters because both DELETE= and DOUBLE= begin with “d”. The character string must always end with an equal sign.

**Example**
The following example calls the COMPCOST routine to compute the generalized edit distance for the operations that are specified.

```sas
options pageno=1 nodate linesize=80 pagesize=60;
data test;
  length String $8 Operation $40;
  if _n_ = 1 then call compcost('insert=', 10, 'DEL=', 11, 'r=', 12);
  input String Operation;
  GED=compged(string, 'baboon');
datalines;
baboon match
xbaboon insert
babon delete
baXoon replace
;
proc print data=test label;
  label GED='Generalized Edit Distance';
  var String Operation GED;
run;
```

The following output shows the results.
CALL EXECUTE Routine

Resolves the argument, and issues the resolved value for execution at the next step boundary.

**Category:** Macro

**Restriction:** This function is not valid on the CAS server.

**Note:** Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

**Syntax**

CALL EXECUTE(argument);

**Required Argument**

*argument*

specifies a character expression or a constant that yields a macro invocation or a SAS statement. *Argument* can be:

- a character string, enclosed in quotation marks.
- the name of a DATA step character variable. Do not enclose the name of the DATA step variable in quotation marks.

---

**See Also**

**Functions:**

- “COMPARE Function” on page 184
- “COMPGED Function” on page 856
- “COMPLEV Function” on page 862
a character expression that the DATA step resolves to a macro text expression or a SAS statement.

Details

If argument resolves to a macro invocation, the macro executes immediately and DATA step execution pauses while the macro executes. If argument resolves to a SAS statement or if execution of the macro generates SAS statements, the statement(s) execute after the end of the DATA step that contains the CALL EXECUTE routine. CALL EXECUTE is fully documented in SAS Viya Macro Language: Reference.

CALL GRAYCODE Routine

Generates all subsets of $n$ items in a minimal change order.

**Category:** Combinatorial

**Restriction:** This function is not valid on the CAS server.

**Note:** Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

**Syntax**

CALL GRAYCODE($k$, numeric-variable-1, ..., numeric-variable-$n$);

CALL GRAYCODE($k$, character-variable <, $n$ <, in-out> >);

**Required Arguments**

$k$

specifies a numeric variable. Initialize $k$ to either of the following values before executing the CALL GRAYCODE routine:

- a negative number to cause CALL GRAYCODE to initialize the subset to be empty
- the number of items in the initial set indicated by numeric-variable-1 through numeric-variable-$n$, or character-variable, which must be an integer value between 0 and N inclusive

The value of $k$ is updated when CALL GRAYCODE is executed. The value that is returned is the number of items in the subset.

numeric-variable

specifies numeric variables that have values of 0 or 1 which are updated when CALL GRAYCODE is executed. A value of 1 for numeric-variable-$j$ indicates that the $j^{th}$ item is in the subset. A value of 0 for numeric-variable-$j$ indicates that the $j^{th}$ item is not in the subset.

If you assign a negative value to $k$ before you execute CALL GRAYCODE, then you do not need to initialize numeric-variable-1 through numeric-variable-$n$ before executing CALL GRAYCODE unless you want to suppress the note about uninitialized variables.
If you assign a value between 0 and $n$ inclusive to $k$ before you execute CALL GRAYCODE, then you must initialize $\text{numeric-variable-1}$ through $\text{numeric-variable-n}$ to $k$ values of 1 and $n-k$ values of 0.

**character-variable**

specifies a character variable that has a length of at least $n$ characters. The first $n$ characters indicate which items are in the subset. By default, an "I" in the $j$th position indicates that the $j$th item is in the subset, and an "O" in the $j$th position indicates that the $j$th item is out of the subset. You can change the two characters by specifying the _in-out_ argument.

If you assign a negative value to $k$ before you execute CALL GRAYCODE, then you do not need to initialize _character-variable_ before executing CALL GRAYCODE unless you want to suppress the note about an uninitialized variable.

If you assign a value between 0 and $n$ inclusive to $k$ before you execute CALL GRAYCODE, then you must initialize _character-variable_ to $k$ characters that indicate an item is in the subset, and $k$-characters that indicate an item is out of the subset.

**Optional Arguments**

$n$

specifies a numeric constant, variable, or expression. By default, $n$ is the length of _character-variable_.

_in-out_

specifies a character constant, variable, or expression. The default value is "IO." The first character is used to indicate that an item is in the subset. The second character is used to indicate that an item is out of the subset.

**Details**

**Using CALL GRAYCODE in a DATA Step**

When you execute the CALL GRAYCODE routine with a negative value of $k$, the subset is initialized to be empty.

When you execute the CALL GRAYCODE routine with an integer value of $k$ between 0 and $n$ inclusive, one item is either added to the subset or removed from the subset, and the value of $k$ is updated to equal the number of items in the subset.

To generate all subsets of $n$ items, you can initialize $k$ to a negative value and execute CALL GRAYCODE in a loop that iterates $2^n$ times. If you want to start with a non-empty subset, then initialize $k$ to be the number of items in the subset, initialize the other arguments to specify the desired initial subset, and execute CALL GRAYCODE in a loop that iterates $2^n-1$ times. The sequence of subsets that are generated by CALL GRAYCODE is cyclical, so you can begin with any subset that you want.

**Using the CALL GRAYCODE Routine with Macros**

You can call the GRAYCODE routine when you use the `%SYSCALL` macro. Differences exist when you use CALL GRAYCODE in a DATA step and when you use the routine with macros. The following list describes usage with macros:

- All arguments must be initialized to nonblank values.
- If you use the _character-variable_ argument, then it must be initialized to a nonblank, nonnumeric character string that contains at least $n$ characters.
• If you use the in-out argument, then it must be initialized to a string that contains two characters that are not blanks, digits, decimal points, or plus and minus signs.

If %SYSCALL identifies an argument as being the wrong type, or if %SYSCALL is unable to identify the type of argument, then &SYSERR and &SYSINFO are not set.

Otherwise, if an error occurs during the execution of the CALL GRAYCODE routine, then both of the following values are set:

• &SYSERR is assigned a value that is greater than 4.
• &SYSINFO is assigned a value that is less than –100.

If there are no errors, then &SYSERR is set to zero, and &SYSINFO is set to one of the following values:

• 0 if the value of \( k \) on input is negative
• the index of the item that was added or removed from the subset if the value of \( k \) on input is a valid nonnegative integer.

Examples

**Example 1: Using a Character Variable and Positive Initial \( k \) with CALL GRAYCODE**

The following example uses the CALL GRAYCODE routine to generate subsets in a minimal change order.

```sas
data _null_;
x='++++';
n=length(x);
k=countc(x, '+');
put '    1' +3 k= +2 x=;
nsubs=2**n;
do i=2 to nsubs;
call graycode(k, x, n, '+-');
put i 5. +3 k= +2 x=;
end;
run;
```

SAS writes the following output to the log:

```
1   k=4   x=++++
2   k=3   x=+++ 
3   k=2   x=--  
4   k=3   x=+++ 
5   k=2   x=---- 
6   k=1   x=--- 
7   k=0   x=---- 
8   k=1   x=--- 
9   k=2   x=++- 
10  k=1   x=--- 
11  k=2   x=--- 
12  k=3   x=++- 
13  k=2   x=--  
14  k=1   x=---- 
15  k=2   x=--- 
16  k=3   x=+++ 
```
**Example 2: Using %SYSCALL with Numeric Variables and Negative k**

The following example uses the %SYSCALL macro with numeric variables to generate subsets in a minimal change order.

```sas
%macro test;
  %let n=3;
  %let x1=.;
  %let x2=.;
  %let x3=.;
  %let k=-1;
  %let nsubs=%eval(2**&n + 1);
  %put nsubs=&nsubs k=&k x: &x1 &x2 &x3;
  %do j=1 %to &nsubs;
    %syscall graycode(k, x1, x2, x3);
    %put &j: k=&k x: &x1 &x2 &x3 sysinfo=&sysinfo;
  %end;
%mend;
%test;
```

SAS writes the following output to the log:

```
nsubs=9 k=-1 x: . . .
1: k=0 x: 0 0 0 sysinfo=0
2: k=1 x: 1 0 0 sysinfo=1
3: k=2 x: 1 1 0 sysinfo=2
4: k=1 x: 0 1 0 sysinfo=1
5: k=2 x: 0 1 1 sysinfo=3
6: k=3 x: 1 1 1 sysinfo=1
7: k=2 x: 1 0 1 sysinfo=2
8: k=1 x: 0 0 1 sysinfo=1
9: k=0 x: 0 0 0 sysinfo=3
```

**Example 3: Using %SYSCALL with a Character Variable and Negative k**

The following example uses the %SYSCALL macro with a character variable to generate subsets in a minimal change order.

```sas
%macro test(n);
  %*** Initialize the character variable to a sufficiently long nonblank, nonnumeric value. ;
  %let x=%sysfunc(repeat(_, &n-1));
  %let k=-1;
  %let nsubs=%eval(2**&n + 1);
  %put nsubs=&nsubs k=&k x="&x";
  %do j=1 %to &nsubs;
    %syscall graycode(k, x, n);
    %put &j: k=&k x="&x" sysinfo=&sysinfo;
  %end;
%mend;
%test(3);
```
SAS writes the following output to the log:

```
nsubs=9  k=-1  x="___"
1: k=0  x="OOO"  sysinfo=0
2: k=1  x="IOO"  sysinfo=1
3: k=2  x="IIO"  sysinfo=2
4: k=1  x="OIO"  sysinfo=1
5: k=2  x="OII"  sysinfo=3
6: k=3  x="III"  sysinfo=1
7: k=2  x="IOI"  sysinfo=2
8: k=1  x="OOI"  sysinfo=1
9: k=0  x="OOO"  sysinfo=3
```

See Also

Functions:

- “GRAYCODE Function” on page 946

CALL IS8601_CONVERT Routine

Converts an ISO 8601 interval to datetime and duration values, or converts datetime and duration values to an ISO 8601 interval.

Category: Date and Time

Restriction: This function is not valid on the CAS server.

Note: Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

Syntax

```
CALL IS8601_CONVERT(convert-from, convert-to, <from-variables>, <to-variables>, <date-time-replacements>);
```

Required Arguments

**convert-from**

specifies a keyword in single quotation marks that indicates the source for the conversion, such as a date, a datetime and duration, a duration, or an interval value. Convert-from can have one of the following values:

'mdn'

specifies a date value, where n is a value from 1 to 6. The default value is 1. N is the number of components in the from-variables or to-variables arguments. The following components are valid:

- year
- month
- day
- hour
- minute
• second

'dtn'
specifies a datetime value, where \( n \) is a value from 1 to 6. The default value is 1. \( N \) is the number of components in the from-variables or to-variables arguments. The following components are valid:

- year
- month
- day
- hour
- minute
- second

'dt/dt'
specifies that the source value for the conversion is a datetime/datetime value.

'dt/du'
specifies that the source value for the conversion is a datetime/duration value.

'dun'
specifies that the source value for the conversion is a duration value, where \( n \) is a value from 1 to 6. The default value is 1. \( N \) is the number of components in the from-variables or to-variables arguments. The following components are valid:

- year
- month
- day
- hour
- minute
- second

'du/dt'
specifies that the source value for the conversion is a duration/datetime value.

'intvl'
specifies that the source value for the conversion is an interval value.

**convert-to**
specifies a keyword in single quotation marks that indicates the results of the conversion. *Convert-to* can have one of the following values:

'dn'
specifies a date value, where \( n \) is a value from 1 to 6. The default value is 1. \( N \) is the number of components in the from-variables or to-variables arguments. The following components are valid:

- year
- month
- day
- hour
- minute
- second
'dt' specifies a datetime value, where \( n \) is a value from 1 to 6. The default value is 1. \( N \) is the number of components in the `from-variables` or `to-variables` arguments. The following components are valid:

- year
- month
- day
- hour
- minute
- second

'dt/dt' specifies to create a datetime/datetime interval.

dt/du' specifies to create a datetime/duration interval.

'du' specifies to create a duration, where \( n \) is a value from 1 to 6. The default value is 1. \( N \) is the number of components in the `from-variables` or `to-variables` arguments. The following components are valid:

- year
- month
- day
- hour
- minute
- second

'du/dt' specifies to create a duration/datetime interval.

'end' specifies to create a value that is the ending datetime or duration of an interval value.

'intvl' specifies to create an interval value.

'start' specifies to create a value that is the beginning datetime or duration of an interval value.

**Optional Arguments**

`from-variable` specifies one or two variables that contain the source value. Specify one variable for an interval value, and specify two variables, one each, for datetime and duration values. The datetime and duration values are interval components where the first value is the beginning value of the interval and the second value is the ending value of the interval.

**Requirements** An integer variable must be at least a 16-byte character variable whose value is read by the $N8601Bw.d informat or the $N8601Ew.d informat. The integer variable can also be an integer.
value that is returned by invoking the CALL IS8601_CONVERT routine.

A datetime value must be a SAS datetime value or an 8-byte character value that is read by the $N8601Bw.d informat or the $N8601Ew.d informat, or by invoking the CALL IS8601_CONVERT routine.

A duration value must be a numeric value that represents the number of seconds in the duration, or an 8-byte character value whose value is read by the $N8601Bw.d informat or the $N8601Ew.d informat, or by invoking the CALL IS8601_CONVERT routine.

**to-variable**

specifies one or two variables that contain converted values. Specify one variable for an interval value, and specify two variables, one each, for datetime and duration values.

**Requirement**
The interval variable must be a character variable that is at least 16 bytes.

**Tip**
The datetime and duration variables can be numeric or character. To avoid losing precision of a numeric value, the length of a numeric variable must be at least eight characters. Datetime and duration character variables must be at least 16 bytes; they are padded with blank characters for values that are less than the length of the variable.

**date-time-replacements**

specifies date or time component values to use when a month, day, or time component is omitted from an interval, datetime, or duration value. The *date-time-replacements* argument is specified as a series of numbers that are separated by commas and represent components in this order: year, month, day, hour, minute, and second. Components of *date-time-replacements* can be omitted only in the reverse order: second, minute, hour, day, month, and year. If no substitute values are specified, the conversion is performed using default values.

**Default**
The following default values for date and time components are omitted:

```
1 month
1 day
0 hour
0 minute
0 second
```

**Requirement**
A year component must be part of the datetime or duration value and therefore is not valid in *date-time-replacements*. A comma is required as a placeholder for the year in *date-time-replacements*. For example, in the replacement value string, ,9,4,,2,, the first comma is a placeholder for a year value.
Details

The Basics
ISO 8601 representations of date, time, and datetime values within the ISO 8601 standards consist of basic and extended notations. A value is basic when delimiters that separate the various components within a value are omitted. A value is extended when delimiters are used to separate those components. A set of formats and informats are used with ISO 8601 date, time, and datetime values. For more information, see “Working with Dates and Times By Using the ISO 8601 Basic and Extended Notations” in *SAS Viya Formats and Informats: Reference* and “Reading Dates and Times By Using the ISO 860 Basic and Extended Notations” in *SAS Viya Formats and Informats: Reference*.

After your values are stored as SAS variables, you can calculate intervals, durations, and datetime values with the CALL IS8601_CONVERT routine. This routine enables you to convert an ISO 8601 interval to datetime and duration values, or to convert datetime and duration values to an ISO 8601 interval. The CALL IS8601_CONVERT routine accepts missing values in the \(dn\), \(dt\n\), and \(dun\) arguments. You can also use the CALL IS8601_CONVERT routine to perform calculations with dates and times.

You can use the year, month, day, hour, minute, and second components in any order.

How Arguments in the CALL IS8601_CONVERT Routine Are Used
The first argument to the CALL IS8601_CONVERT routine, \(convert-from\), can be one or two values. The number of values is based on how many variables you provide to the routine for an expected result. For example, datetime and duration values can be specified with an expected output of an interval. In this example, the supplied first argument would be \(dt/dun\). If two datetime values are supplied with an expected duration as output, the first argument would be \(dt/dt\).

The second argument to the CALL IS8601_CONVERT routine, \(convert-to\), can also be one or two values. The values depend on the type of result that you expect SAS to compute using the input that you supply in the first argument.

*Note:* The minimal length for durations is 16, and the minimum length for intervals is 32.

Examples

Example 1: Using the CALL IS8601_CONVERT Routine
This DATA step uses the CALL IS8601_CONVERT routine to perform these tasks:

- create an interval by using datetime and duration values
- create datetime and duration values from an interval that was created using the CALL IS8601_CONVERT routine
- create an interval from datetime and duration values by using replacement values for omitted date and time components in the datetime value

For easier reading, numeric variables end with an N, and character variables end with a C.

```sas
data _null_;  /* Declare variable length and type. Character datetime and duration */  /* values must be at least 16 characters. To avoid losing precision, */  /* the numeric datetime value has a length of 8. */  /* */
```
CALL IS8601_CONVERT Routine

length dtN duN 8 dtC duC $16 intervalC $32;
/* Assign a numeric datetime value and a character duration value. */
dtN='15Sep2011:09:00:00'dt;
duC=input('P2y3m4dT5h6m7s', $n8601b.);
put dtN=;
pduC=;
/* Create an interval from a datetime and duration value and format the */
/* interval using the ISO 8601 extended notation for character values. */
call is8601_convert('dt/du', 'intvl', dtN, duC, intervalC);
put '** Character interval created from datetime and duration values **/';
pput intervalC $n8601e.;
p'';
/* Create numeric datetime and duration values from an interval and */
/* format the values using the ISO 8601 extended notation for numeric */
/* values. */
call is8601_convert('intvl', 'dt/du', intervalC, dtN, duN);
put '** Character datetime and duration created from an interval **/';
pdtN=;
put duN=;
/* Assign a new datetime value with omitted components. */
dtC=input('2012---15T10:-:-', $n8601b.);
put '** This datetime is a character value. **';
pdtC $n8601h.;
p'';
/* Create an interval by reading a datetime value that has omitted date */
/* and time components. Use replacement values for the month, minutes, */
/* and seconds. */
call is8601_convert('du/dt', 'intvl', duC, dtC, intervalC, , 7, , , 35, 45);
put '** Interval created using a datetime with omitted values, **/'
put '** inserting replacement values for month (7), minute (35) **/';
pput intervalC $n8601e.;
p'';
run;
SAS writes the following results to the log:

| dtN=1631696400 |
| duC=0002304050607FFC |
| ** Character interval created from datetime and duration values **/ |
| 2011-09-15T09:00:00.000/P2Y3M4DT5H6M7S |
| ** Character datetime and duration created from an interval **/ |
| dtN=1631696400 |
| duN=71211967 |
| ** This datetime is a character value. ** |
| 2012---15T10:-:- |
| ** Interval created using a datetime with omitted values, **/ |
| ** inserting replacement values for month (7), minute (35) **/ |
| ** seconds (45). **/ |
| P2Y3M4DT5H6M7S/2012-07-15T10:35:45 |

Example 2: Finding an Interval Value
This example writes duration and datetime values to the log.

data _null_;
length mynew $32;
x='P8w';
y='11feb2012:12:35:22'dt;
call is8601_convert('du/dt', 'intvl', x, y, mynew);
put mynew=$n8601e.);
run;

SAS writes the following results to the log:

```
mynew=P8W/2012-02-11T12:35:22.000
```

**Example 3: Finding the Start Date of an Interval**

This example returns the start date of an interval.

```
data temp;
length dt $32;
x='P6w';
y='11feb2012:11:13:22'dt;
call is8601_convert('du/dt', 'start', x, y, dt);
put dt=$n8601e.);
run;
```

SAS writes the following results to the log:

```
dt=2011-12-31T00:00:00.000
```

In this example, P6w specifies a six-week duration, and y is the date from which start is calculated. start is the beginning duration of an interval. This datetime value is six weeks before y.

**Example 4: Finding an Interval Using Duration and Datetime Values**

This example uses duration and datetime values to find an interval.

```
data _null_;
infile datalines;
input mo d yr hour min sec;
length mynew $32;
x='P8w';
call is8601_convert('du/dt6', 'intvl', x, mo, d, yr, hour, min, sec, mynew);
put mynew=$n8601e.);
datalines;
02 22 2011 10 30 45
12 13 2010 12 35 25
03 26 2010 10 10 10
;
run;
```

SAS writes the following results to the log:

```
mynew=P8W/0002-14-91T10:30:45.000
mynew=P8W/0012-13-90T12:35:25.000
mynew=P8W/0003-14-90T10:10:10.000
```

**Example 5: Finding the Start Date for Multiple Intervals**

This example uses data lines as input to find the start date for an interval.
data temp;
  input y mo d h min s;
  length mynew $16;
  /* This value is the duration. */
  x='P2w';
  /* This CALL routine uses the date and time values that are */
  /* listed in the data lines to create a DATETIME value. */
  call is8601_convert('dt6', 'dt', y, mo, d, h, min, s, dt);
  put dt=datetime.;
  /* This CALL routine uses the DATETIME value that was previously */
  /* created. */
  call is8601_convert('du/dt', 'start', x, dt, mynew);
  put mynew=$n8601e.;
datalines;
  2011 6 7 10 15 20
  2011 12 5 10 25 45
  2011 6 30 10 32 20
; SAS writes the following results to the log:

dt=07JUN11:10:15:20
mynew=2011-05-24T00:00:00.000
dt=05DEC11:10:25:45
mynew=2011-11-21T00:00:00.000
dt=30JUN11:10:32:20
mynew=2011-06-16T00:00:00.000

Example 6: Converting a Duration to a SAS Time
In this example, you supply the duration value P8w, which specifies eight weeks. This type of value is often coupled with a datetime value to produce a duration, but the value can be used alone and converted to a SAS time value. There is no SAS informat to convert the P8w value to a SAS time value, but you can use the CALL IS8601_CONVERT routine to convert the value.

data a;
  x='P8w';
  call is8601_convert('du', 'du', x, mynew);
  put mynew=time8.;
run;

The value of Mynew is 1344:00, which indicates 1344 hours.

In this example, the following statements apply:

• The X variable is a duration. Therefore, 'du' is the first argument.
• A time value is expected as output, but there is no convert-to value for time. As a result, the value for 'du' is used for duration. Both arguments require single quotation marks.
• X is the variable name that is supplied, and Mynew is the variable that is being created.
• Because the result is a SAS time value, you can use a SAS time format (TIME8.).
Example 7: Converting a Duration Value and a Datetime Value to an Interval

This example takes the previous example one step further, by using the same duration value with a datetime value to create an interval as output. The following DATA step is based on an event that lasts for eight weeks, ending on February 11, 2012, at 12:22 p.m.

```sas
data a;
  length mynew $32;
  x='P8w';
  y='11feb2012:12:22'dt;
  call is8601_convert('du/dt', 'intvl', x, y, mynew);
  put mynew=$n8601e.;
run;
```

The value of Mynew is P8W/2012-02-11T12:22:00.000.

In this example, the following statements apply:

- The first argument, 'du/dt', to the CALL IS8601_CONVERT routine indicates the types of variables that are being passed in for conversion. The value 'du/dt' specifies that two variables are being passed: one is a duration value, X, and the other is a datetime value, Y. Because the duration value comes before the datetime value, the datetime value is assumed to be the end of the interval.

- The result that you want from this DATA step is an interval. Therefore, the second argument is 'intvl'.

- The remaining arguments name the incoming variables, X and Y, and the new variable, Mynew, that is being created. The order of these variables must match the types of variables that are specified in the first argument. For example, if X and Y are reversed, the following note appears in the log:

```
NOTE: Invalid argument to function IS8601_CONVERT('du/dt','intvl',1644582120, 'P8w','') at line 770 column 9.
mynew=**************************************************
mynew= x=P8w y=1644582120 _ERROR_=1 _N_=1
```

Example 8: Computing the Start Datetime of an Interval When You Have a Duration and a Datetime

"Example 7: Converting a Duration Value and a Datetime Value to an Interval" on page 758 computes an interval when datetime and duration values are supplied. This example computes the starting datetime for an interval when a duration and datetime value are supplied.

```sas
data a;
  length mynew $16;
  /* If you omit the LENGTH statement, replace the PUT statement */
  /* with 'put mynew datetime22.;'. */
  x='P8w';
  y='11feb2012:12:22'dt;
  call is8601_convert('du/dt', 'start', x, y, mynew);
  put mynew=$n8601e.;
run;
```

The value of Mynew is 2011-12-17T00:00:00.000.

In this example, the following statements apply:

- The CALL IS8601_CONVERT routine uses the argument 'start' as the second argument in order to compute the starting datetime value for the interval.
Because the SAS datetime value is computed, you can remove the LENGTH statement so that Mynew is created as a numeric variable. If you remove the LENGTH statement, add a PUT statement that specifies the DATETIME22. format. It is also valid to leave the LENGTH statement and create variable output by using the $N8601Ew. format, as shown in the second PUT statement.

**Example 9: Converting a Duration of One Type to a Duration of a Different Type**

If you receive data that is represented in hours, you can perform a conversion similar to the previous example to obtain output as a duration.

```sas
data _null_;  
x='1271:59:00';  
time=input(x, time10.);  
length dur $16;  
call is8601_convert('du', 'du', time, dur);  
put dur=$n8601e.;  
run;
```

The value of dur is P0Y1M22DT23H59M0.0S.

In this example, the following statements apply:

- The TIMEw.d format reads the time value into SAS.
- The CALL IS8601_CONVERT routine converts the time value to a duration value.

**Example 10: Converting Two Datetime Values to a Duration**

This example determines the amount of time between two datetime values, and creates a duration as output.

```sas
data a;  
    length dur $16;  
    start='02apr2012:12:30:22'dt;  
    end='08apr2012:14:32:22'dt;  
    call is8601_convert('dt/dt', 'du', start, end, dur);  
    put dur=$n8601e.;  
run;
```

The value of dur is P6DT2H2M.

In this example, the following statements apply:

- The CALL IS8601_CONVERT routine uses the following arguments:
  - 'dt/dt' indicates that two datetime values are being passed into the function.
  - 'du' indicates that a duration value is the expected outcome.
  - 'start' is the name of the first datetime variable.
  - 'end' is the name of the second datetime variable.
  - 'dur' is the name of the desired output variable.
- A LENGTH statement is required in order to specify that Dur is a character variable and that the length should be 16. If you create an interval, a length of 32 is required.
- The routine creates an indecipherable hexadecimal value. Therefore, it is necessary to format the value with one of the $N8601 formats; in this case, $N8601Ew.d.
Example 11: Converting an Interval to a Datetime Value and a Duration

You can specify an interval value as two datetime values that are separated by a forward slash (/). The slash separates the beginning and ending values for an event, as shown in this example.

```sas
data _null_;
  length Final2 $16;
  int=input('2012-03-15T14:32:00/2012-03-29T09:45:00', $n8601e40.);
  call is8601_convert('intvl', 'dt/du', int, Final1, Final2);
  put Final1= / Final2=$n8601e.;
run;
```

The resulting values for Final1 and Final2 are listed here:

- Final1=1647441120
- Final2=P13DT19H13M

In this example, the following statements apply:

- The datetime values are specified inside single quotation marks so that the INPUT function and the $N8601EW informat can convert the values into an interval value in the variable `Int`.
- The CALL IS8601_CONVERT routine converts that interval value to two variables: Final1 is a datetime variable, and Final2 is a duration variable.
- Because a SAS datetime variable (Final1) is a numeric variable that can be stored in 8 bytes, a length value is not required for this variable in the LENGTH statement. However, a duration requires a length of 16. Therefore, a length is specified for Final2 in the LENGTH statement.
- Because Final2 is a duration, the $N8601EW format is used to write its value to the log so that the output is understandable.

If the first datetime value in the example was missing the month value (2012-03-15T14:32:00), you could supply that value by specifying the value in the last arguments of the CALL routine, as shown here:

```sas
call is8601_convert('intvl', 'dt/du', int, Final1, Final2, , 2);
```

The month is specified here as 2, and the two consecutive commas preceding that value indicate that a year value was not supplied. Therefore, a datetime and a duration are computed based on the date 2012-02-15T14:32:00 and the other date that is specified in the code. In this case, the output from the PUT statement for each variable is as follows:

- Final1=1644935520
- Final2=P1M13DT19H13M

CALL LEXCOMB Routine

Generates all distinct combinations of the nonmissing values of n variables taken k at a time in lexicographic order.

Category: Combinatorial

Restriction: This function is not valid on the CAS server.
Interaction: When invoked by the %SYSCALL macro statement, CALL LEXCOMB removes the quotation marks from its arguments. For more information, see Using CALL Routines and the %SYSCALL Macro Statement on page 9.

Note: Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

Syntax

CALL LEXCOMB(count, k, variable-1, ..., variable-n);

Required Arguments

**count**

specifies an integer value that is assigned values from 1 to the number of combinations in a loop.

**k**

specifies an integer constant, variable, or expression between 1 and \( n \), inclusive, that specifies the number of items in each combination.

**variable**

specifies either all numeric variables, or all character variables that have the same length. The values of these variables are permuted.

Requirement: Initialize these variables before you call the LEXCOMB routine.

Tip: After calling LEXCOMB, the first \( k \) variables contain the values in one combination.

Details

The Basics

Use the CALL LEXCOMB routine in a loop where the first argument to CALL LEXCOMB takes each integral value from 1 to the number of distinct combinations of the nonmissing values of the variables. In each call to LEXCOMB within this loop, \( k \) should have the same value.

Number of Combinations

When all of the variables have nonmissing, unequal values, then the number of combinations is \( \text{COMB}(n, k) \). If the number of variables that have missing values is \( m \), and all the nonmissing values are unequal, then LEXCOMB produces \( \text{COMB}(n-m, k) \) combinations because the missing values are omitted from the combinations.

When some of the variables have equal values, the exact number of combinations is difficult to compute. If you cannot compute the exact number of combinations, use the LEXCOMB function instead of the CALL LEXCOMB routine.

CALL LEXCOMB Processing

On the first call to the LEXCOMB routine, the following actions occur:

- The argument types and lengths are checked for consistency.
- The \( m \) missing values are assigned to the last \( m \) arguments.
• The \( n-m \) nonmissing values are assigned in ascending order to the first \( n-m \) arguments following \( \text{count} \).

On subsequent calls, up to and including the last combination, the next distinct combination of the nonmissing values is generated in lexicographic order.

If you call the LEXCOMB routine with the first argument out of sequence, then the results are not useful. In particular, if you initialize the variables and then immediately call the LEXCOMB routine with a first argument of \( j \), you will not get the \( j^{th} \) combination (except when \( j \) is 1). To get the \( j^{th} \) combination, you must call the LEXCOMB routine \( j \) times, and the first argument must take values from 1 through \( j \) in that exact order.

**Using the CALL LEXCOMB Routine with Macros**

You can call the LEXCOMB routine when you use the %SYSCALL macro. In this case, the \( \text{variable} \) arguments are not required to be the same length, but they are required to be the same type. If %SYSCALL identifies an argument as numeric, then %SYSCALL reformats the returned value.

If an error occurs during the execution of the CALL LEXCOMB routine, then both of the following values are set:

• \&SYSERR is assigned a value that is greater than 4.
• \&SYSINFO is assigned a value that is less than –100.

If there are no errors, then \&SYSERR is set to zero, and \&SYSINFO is set to one of the following values:

• 1 if \( \text{count}=1 \) and at lease one variable has a nonmissing value
• 1 if the value of \( \text{variable-1} \) changed
• \( j \) if \( \text{variable-1} \) through \( \text{variable-i} \) did not change, but \( \text{variable-j} \) did change, where \( j\) = \( i+1 \)
• –1 if all distinct combinations have already been generated

**Comparisons**

The CALL LEXCOMB routine generates all distinct combinations of the nonmissing values of \( n \) variables taken \( k \) at a time in lexicographic order. The CALL ALLCOMB routine generates all combinations of the values of \( n \) variables taken \( k \) at a time in a minimal change order.

**Examples**

**Example 1: Using CALL LEXCOMB in a DATA Step**

The following example calls the LEXCOMB routine to generate distinct combinations in lexicographic order.

```sas
data _null_;
  array x[5] $3 ('ant' 'bee' 'cat' 'dog' 'ewe');
  n=dim(x);
  k=3;
  ncomb=comb(n, k);
  do j=1 to ncomb;
    call lexcomb(j, k, of x[*]);
    put j 5. +3 x1-x3;
  end;
```
run;

SAS writes the following output to the log:

```
1   ant bee cat
2   ant bee dog
3   ant bee ewe
4   ant cat dog
5   ant cat ewe
6   ant dog ewe
7   bee cat dog
8   bee cat ewe
9   bee dog ewe
10  cat dog ewe
```

**Example 2: Using CALL LEXCOMB with Macros**

The following is an example of the CALL LEXCOMB routine that is used with macros. The output includes values for the %SYSINFO macro.

```sas
%macro test;
  %let x1=ant;
  %let x2=baboon;
  %let x3=baboon;
  %let x4=hippopotamus;
  %let x5=zebra;
  %let k=2;
  %let ncomb=%sysfunc(comb(5, &k));
  %do j=1 %to &ncomb;
    %syscall lexcomb(j, k, x1, x2, x3, x4, x5);
    %let jfmt=%qsysfunc(putn(&j, 5. ));
    %let pad=%qsysfunc(repeat(%str( ), 20-%length(&x1 &x2)});
    %put &jfmt: &x1 &x2 &pad sysinfo=&sysinfo;
    %if &sysinfo < 0 %then %let j=%eval(&ncomb+1);  
  %end;
%mend;
%test
```

SAS writes the following output to the log:

```
1: ant baboon             sysinfo=1
2: ant hippopotamus       sysinfo=2
3: ant zebra              sysinfo=2
4: baboon baboon          sysinfo=1
5: baboon hippopotamus    sysinfo=2
6: baboon zebra           sysinfo=2
7: hippopotamus zebra     sysinfo=1
8: hippopotamus zebra     sysinfo=-1
```

**See Also**

**Functions:**
- “LEXCOMB Function” on page 977

**CALL Routines:**
- “CALL ALLCOMB Routine” on page 727
CALL LEXCOMBI Routine

Generates all combinations of the indices of $n$ objects taken $k$ at a time in lexicographic order.

<table>
<thead>
<tr>
<th>Category:</th>
<th>Combinatorial</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restriction:</td>
<td>This function is not valid on the CAS server.</td>
</tr>
<tr>
<td>Note:</td>
<td>Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.</td>
</tr>
</tbody>
</table>

Syntax

CALL LEXCOMBI$(n, k, \text{index-1}, \ldots, \text{index-k})$;

Required Arguments

$n$

is a numeric constant, variable, or expression that specifies the total number of objects.

$k$

is a numeric constant, variable, or expression that specifies the number of objects in each combination.

$\text{index}$

is a numeric variable that contains indices of the objects in the combination that is returned. Indices are integers between 1 and $n$, inclusive.

Tip: If $\text{index-1}$ is missing or zero, then the CALL LEXCOMBI routine initializes the indices to $\text{index-1}=1$ through $\text{index-k}=k$. Otherwise, CALL LEXCOMBI creates a new combination by removing one index from the combination and adding another index.

Details

CALL LEXCOMBI Processing

Before the first call to the LEXCOMBI routine, complete one of the following tasks:

- Set $\text{index-1}$ equal to zero or to a missing value.
- Initialize $\text{index-1}$ through $\text{index-k}$ to distinct integers between 1 and $n$ inclusive.

The number of combinations of $n$ objects taken $k$ at a time can be computed as COMB$(n,k)$. To generate all combinations of $n$ objects taken $k$ at a time, call LEXCOMBI in a loop that executes COMB$(n,k)$ times.

Using the CALL LEXCOMBI Routine with Macros

If you call the LEXCOMBI routine from the macro processor with %SYSCALL, then you must initialize all arguments to numeric values. %SYSCALL reformats the values that are returned.

If an error occurs during the execution of the CALL LEXCOMBI routine, then both of the following values are set:
CALL LEXCOMBI Routine

- &SYSERR is assigned a value that is greater than 4.
- &SYSINFO is assigned a value that is less than –100.

If there are no errors, then &SYSERR is set to zero, and &SYSINFO is set to one of the following values:

- 1 if the value of variable-1 changed
- \( j \) if variable-1 through variable-\( i \) did not change, but variable-\( j \) did change, where \( j = \) \( i + 1 \)
- –1 if all distinct combinations have already been generated

Comparisons

The CALL LEXCOMBI routine generates all combinations of the indices of \( n \) objects taken \( k \) at a time in lexicographic order. The CALL ALLCOMBI routine generates all combinations of the indices of \( n \) objects taken \( k \) at a time in a minimum change order.

Examples

Example 1: Using the CALL LEXCOMBI Routine with the DATA Step

The following example uses the CALL LEXCOMBI routine to generate combinations of indices in lexicographic order.

```sas
data _null_;  array x[5] $3 ('ant' 'bee' 'cat' 'dog' 'ewe');  array c[3] $3;  array i[3];  n=dim(x);  k=dim(i);  i[1]=0;  ncomb=comb(n, k);  do j=1 to ncomb;  call lexcombi(n, k, of i[*]);  do h=1 to k;  c[h]=x[i[h]];  end;  put @4 j= @10 'i= ' i[*] +3 'c= ' c[*];  end;  run;
```

SAS writes the following output to the log:

```plaintext
j=1  i= 1 2 3  c= ant bee cat
j=2  i= 1 2 4  c= ant bee dog
j=3  i= 1 2 5  c= ant bee ewe
j=4  i= 1 3 4  c= ant cat dog
j=5  i= 1 3 5  c= ant cat ewe
j=6  i= 1 4 5  c= ant dog ewe
j=7  i= 2 3 4  c= bee cat dog
j=8  i= 2 3 5  c= bee cat ewe
j=9  i= 2 4 5  c= bee dog ewe
j=10  i= 3 4 5  c= cat dog ewe
```
Example 2: Using the CALL LEXCOMBI Routine with Macros and Displaying the Return Code

The following example uses the CALL LEXCOMBI routine with macros. The output includes values for the %SYSINFO macro.

```sas
%macro test;
    %let x1=0;
    %let x2=0;
    %let x3=0;
    %let n=5;
    %let k=3;
    %let ncomb=%sysfunc(comb(&n, &k));
    %do j=1 %to &ncomb+1;
        %syscall lexcombi(n, k, x1, x2, x3);
        %let jfmt=%qsysfunc(putn(&j, 5.));
        %let pad=%qsysfunc(repeat(%str(), 6-%length(&x1 &x2 &x3)));
        %put &jfmt: &x1 &x2 &x3 &pad sysinfo=&sysinfo;
    %end;
%mend;
%test
```

SAS writes the following output to the log:

```
1: 1 2 3  sysinfo=1
2: 1 2 4  sysinfo=3
3: 1 2 5  sysinfo=3
4: 1 3 4  sysinfo=2
5: 1 3 5  sysinfo=3
6: 1 4 5  sysinfo=2
7: 2 3 4  sysinfo=1
8: 2 3 5  sysinfo=3
9: 2 4 5  sysinfo=2
10: 3 4 5  sysinfo=1
11: 3 4 5  sysinfo=-1
```

See Also

CALL Routines:

- “CALL LEXCOMB Routine” on page 760
- “CALL ALLCOMBI Routine” on page 730

CALL LEXPERK Routine

Generates all distinct permutations of the nonmissing values of \(n\) variables taken \(k\) at a time in lexicographic order.

**Category:** Combinatorial

**Restriction:** This function is not valid on the CAS server.

**Interaction:** When invoked by the %SYSCALL macro statement, CALL LEXPERK removes the quotation marks from its arguments. For more information, see Using CALL Routines and the %SYSCALL Macro Statement on page 9.
Note: Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

Syntax

CALL LEXPERK(count, k, variable-1, ..., variable-n);

Required Arguments

**count**
- specifies an integer variable that is assigned a value from 1 to the number of permutations in a loop.

**k**
- specifies an integer constant, variable, or expression between 1 and \( n \), inclusive, that specifies the number of items in each permutation.

**variable**
- specifies either all numeric variables, or all character variables that have the same length. The values of these variables are permuted.

Requirement Initialize these variables before you call the LEXPERK routine.

Tip After calling LEXPERK, the first \( k \) variables contain the values in one permutation.

Details

**The Basics**
Use the CALL LEXPERK routine in a loop where the first argument to CALL LEXPERK accepts each integral value from 1 to the number of distinct permutations of \( k \) nonmissing values of the variables. In each call to LEXPERK within this loop, \( k \) should have the same value.

**Number of Permutations**
When all of the variables have nonmissing, unequal values, the number of permutations is \( \text{PERM}(n,k) \). If the number of variables that have missing values is \( m \), and all the nonmissing values are unequal, CALL LEXPERK produces \( \text{PERM}(n-m,k) \) permutations because the missing values are omitted from the permutations. When some of the variables have equal values, the exact number of permutations is difficult to compute. If you cannot compute the exact number of permutations, use the LEXPERK function instead of the CALL LEXPERK routine.

**CALL LEXPERK Processing**
On the first call to the LEXPERK routine, the following actions occur:

- The argument types and lengths are checked for consistency.
- The \( m \) missing values are assigned to the last \( m \) arguments.
- The \( n-m \) nonmissing values are assigned in ascending order to the first \( n-m \) arguments following \( count \).

On subsequent calls, up to and including the last permutation, the next distinct permutation of \( k \) nonmissing values is generated in lexicographic order.
If you call the LEXPERK routine with the first argument out of sequence, then the results are not useful. In particular, if you initialize the variables and then immediately call the LEXPERK routine with a first argument of \( j \), you do not get the \( j \)th permutation (except when \( j = 1 \)). To get the \( j \)th permutation, you must call LEXPERK \( j \) times. The first argument takes values from 1 through \( j \) in that exact order.

**Using the CALL LEXPERK Routine with Macros**
You can call the LEXPERK routine when you use the `%SYSCALL` macro. In this case, the variable arguments are not required to be the same length, but they are required to be the same type. If `%SYSCALL` identifies an argument as numeric, then `%SYSCALL` reformats the returned value.

If an error occurs during the execution of the CALL LEXPERK routine, then both of the following values are set:

- &SYSERR is assigned a value that is greater than 4.
- &SYSINFO is assigned a value that is less than –100.

If there are no errors, then &SYSERR is set to zero, and &SYSINFO is set to one of the following values:

- 1 if \( count = 1 \) and at least one variable has a nonmissing value
- 1 if \( count > 1 \) and the value of \( \text{variable-1} \) changed
- \( j \) if \( count > 1 \) and \( \text{variable-1} \) through \( \text{variable-i} \) did not change, but \( \text{variable-j} \) did change, where \( j = i + 1 \)
- –1 if all distinct permutations were already generated

**Comparisons**
The CALL LEXPERK routine generates all distinct permutations of the nonmissing values of \( n \) variables taken \( k \) at a time in lexicographic order. The CALL ALLPERM routine generates all permutations of the values of several variables in a minimal change order.

**Examples**

**Example 1: Using CALL LEXPERK in a DATA Step**
The following is an example of the CALL LEXPERK routine.

```sas
data _null_;
array x[5] $3 ('V' 'W' 'X' 'Y' 'Z');
n=dim(x);
k=3;
nperm=perm(n, k);
do j=1 to nperm;
call lexperk(j, k, of x[*]);
   put j 5. +3 x1-x3;
end;
run;
```
SAS writes the following output to the log:

<table>
<thead>
<tr>
<th></th>
<th>V</th>
<th>W</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>V</td>
<td>W</td>
<td>Y</td>
</tr>
<tr>
<td>3</td>
<td>V</td>
<td>W</td>
<td>Z</td>
</tr>
<tr>
<td>4</td>
<td>V</td>
<td>X</td>
<td>W</td>
</tr>
<tr>
<td>5</td>
<td>V</td>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td>6</td>
<td>V</td>
<td>X</td>
<td>Z</td>
</tr>
<tr>
<td>7</td>
<td>V</td>
<td>Y</td>
<td>W</td>
</tr>
<tr>
<td>8</td>
<td>V</td>
<td>Y</td>
<td>X</td>
</tr>
<tr>
<td>9</td>
<td>V</td>
<td>Y</td>
<td>Z</td>
</tr>
<tr>
<td>10</td>
<td>V</td>
<td>Z</td>
<td>W</td>
</tr>
<tr>
<td>11</td>
<td>V</td>
<td>Z</td>
<td>X</td>
</tr>
<tr>
<td>12</td>
<td>V</td>
<td>Z</td>
<td>Y</td>
</tr>
<tr>
<td>13</td>
<td>W</td>
<td>V</td>
<td>X</td>
</tr>
<tr>
<td>14</td>
<td>W</td>
<td>V</td>
<td>Y</td>
</tr>
<tr>
<td>15</td>
<td>W</td>
<td>V</td>
<td>Z</td>
</tr>
<tr>
<td>16</td>
<td>W</td>
<td>X</td>
<td>V</td>
</tr>
<tr>
<td>17</td>
<td>W</td>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td>18</td>
<td>W</td>
<td>X</td>
<td>Z</td>
</tr>
<tr>
<td>19</td>
<td>W</td>
<td>Y</td>
<td>V</td>
</tr>
<tr>
<td>20</td>
<td>W</td>
<td>Y</td>
<td>X</td>
</tr>
<tr>
<td>21</td>
<td>W</td>
<td>Y</td>
<td>Z</td>
</tr>
<tr>
<td>22</td>
<td>W</td>
<td>Z</td>
<td>V</td>
</tr>
<tr>
<td>23</td>
<td>W</td>
<td>Z</td>
<td>X</td>
</tr>
<tr>
<td>24</td>
<td>W</td>
<td>Z</td>
<td>Y</td>
</tr>
<tr>
<td>25</td>
<td>X</td>
<td>V</td>
<td>W</td>
</tr>
<tr>
<td>26</td>
<td>X</td>
<td>V</td>
<td>Y</td>
</tr>
<tr>
<td>27</td>
<td>X</td>
<td>V</td>
<td>Z</td>
</tr>
<tr>
<td>28</td>
<td>X</td>
<td>W</td>
<td>V</td>
</tr>
<tr>
<td>29</td>
<td>X</td>
<td>W</td>
<td>Y</td>
</tr>
<tr>
<td>30</td>
<td>X</td>
<td>W</td>
<td>Z</td>
</tr>
<tr>
<td>31</td>
<td>X</td>
<td>Y</td>
<td>V</td>
</tr>
<tr>
<td>32</td>
<td>X</td>
<td>Y</td>
<td>W</td>
</tr>
<tr>
<td>33</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
</tr>
<tr>
<td>34</td>
<td>X</td>
<td>Z</td>
<td>V</td>
</tr>
<tr>
<td>35</td>
<td>X</td>
<td>Z</td>
<td>W</td>
</tr>
<tr>
<td>36</td>
<td>X</td>
<td>Z</td>
<td>Y</td>
</tr>
<tr>
<td>37</td>
<td>Y</td>
<td>V</td>
<td>W</td>
</tr>
<tr>
<td>38</td>
<td>Y</td>
<td>V</td>
<td>X</td>
</tr>
<tr>
<td>39</td>
<td>Y</td>
<td>V</td>
<td>Z</td>
</tr>
<tr>
<td>40</td>
<td>Y</td>
<td>W</td>
<td>V</td>
</tr>
<tr>
<td>41</td>
<td>Y</td>
<td>W</td>
<td>X</td>
</tr>
<tr>
<td>42</td>
<td>Y</td>
<td>W</td>
<td>Z</td>
</tr>
<tr>
<td>43</td>
<td>Y</td>
<td>X</td>
<td>V</td>
</tr>
<tr>
<td>44</td>
<td>Y</td>
<td>X</td>
<td>W</td>
</tr>
<tr>
<td>45</td>
<td>Y</td>
<td>X</td>
<td>Z</td>
</tr>
<tr>
<td>46</td>
<td>Y</td>
<td>Z</td>
<td>V</td>
</tr>
<tr>
<td>47</td>
<td>Y</td>
<td>Z</td>
<td>W</td>
</tr>
<tr>
<td>48</td>
<td>Y</td>
<td>Z</td>
<td>X</td>
</tr>
<tr>
<td>49</td>
<td>Z</td>
<td>V</td>
<td>W</td>
</tr>
<tr>
<td>50</td>
<td>Z</td>
<td>V</td>
<td>X</td>
</tr>
<tr>
<td>51</td>
<td>Z</td>
<td>V</td>
<td>Y</td>
</tr>
<tr>
<td>52</td>
<td>Z</td>
<td>W</td>
<td>V</td>
</tr>
<tr>
<td>53</td>
<td>Z</td>
<td>W</td>
<td>X</td>
</tr>
<tr>
<td>54</td>
<td>Z</td>
<td>W</td>
<td>Y</td>
</tr>
<tr>
<td>55</td>
<td>Z</td>
<td>X</td>
<td>V</td>
</tr>
<tr>
<td>56</td>
<td>Z</td>
<td>X</td>
<td>W</td>
</tr>
<tr>
<td>57</td>
<td>Z</td>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td>58</td>
<td>Z</td>
<td>Y</td>
<td>V</td>
</tr>
<tr>
<td>59</td>
<td>Z</td>
<td>Y</td>
<td>W</td>
</tr>
<tr>
<td>60</td>
<td>Z</td>
<td>Y</td>
<td>X</td>
</tr>
</tbody>
</table>
Example 2: Using CALL LEXPERK with Macros

The following is an example of the CALL LEXPERK routine that is used with macros. The output includes values for the %SYSINFO macro.

```sas
%macro test;
  %let x1=ant;
  %let x2=baboon;
  %let x3=baboon;
  %let x4=hippopotamus;
  %let x5=zebra;
  %let k=2;
  %let nperk=%sysfunc(perm(5, &k));
  %do j=1 %to &nperk;
    %syscall lexperk(j, k, x1, x2, x3, x4, x5);
    %let jfmt=%qsysfunc(putn(&j, 5.));
    %let pad=%qsysfunc(repeat(%str(), 20-%length(&x1 &x2)));
    %put &jfmt: &x1 &x2 &pad sysinfo=&sysinfo;
    %if &sysinfo<0 %then %let j=%eval(&nperk+1);
  %end;
%mend;
%test
```

SAS writes the following output to the log:

```
1: ant baboon  sysinfo=1
2: ant hippopotamus  sysinfo=2
3: ant zebra  sysinfo=2
4: baboon ant  sysinfo=1
5: baboon baboon  sysinfo=2
6: baboon hippopotamus  sysinfo=2
7: baboon zebra  sysinfo=2
8: hippopotamus ant  sysinfo=1
9: hippopotamus baboon  sysinfo=2
10: hippopotamus zebra  sysinfo=2
11: zebra ant  sysinfo=1
12: zebra baboon  sysinfo=2
13: zebra hippopotamus  sysinfo=2
14: zebra hippopotamus  sysinfo=-1
```

See Also

Functions:
- “LEXPERM Function” on page 984

CALL Routines:
- “CALL ALLPERM Routine” on page 733
- “CALL RANPERK Routine” on page 829
- “CALL RANPERM Routine” on page 831

CALL LEXPERM Routine

Generates all distinct permutations of the nonmissing values of several variables in lexicographic order.

Category: Combinatorial
Restriction: This function is not valid on the CAS server.

Interaction: When invoked by the %SYSCALL macro statement, CALL LEXPRM removes the quotation marks from its arguments. For more information, see Using CALL Routines and the %SYSCALL Macro Statement on page 9.

Note: Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

Syntax

CALL LEXPRM(count, variable-1 <, ..., variable-N>);

Required Arguments

count specifies a numeric variable that has an integer value that ranges from 1 to the number of permutations.

variable specifies either all numeric variables, or all character variables that have the same length. The values of these variables are permuted by LEXPRM.

Requirement Initialize these variables before you call the LEXPRM routine.

Details

Determine the Number of Distinct Permutations

These variables are defined for use in the equation that follows:

\[ P = \frac{(N_1 + N_2 + \ldots + N_d)!}{N_1! N_2! \ldots N_d!} \leq N! \]

CALL LEXPRM Processing

Use the CALL LEXPRM routine in a loop where the argument count accepts each integral value from 1 to P. You do not need to compute P provided you exit the loop when CALL LEXPRM returns a value that is less than zero.

For 1=count<P, the following actions occur:

- The argument types and lengths are checked for consistency.
• The M missing values are assigned to the last M arguments.
• The N-M nonmissing values are assigned in ascending order to the first N-M arguments following count.
• CALL LEXPERM returns 1.

For 1<count<=P, the following actions occur:
• The next distinct permutation of the nonmissing values is generated in lexicographic order.
• If variable-1 through variable-I did not change, but variable-J did change, where J=I+1, then CALL LEXPERM returns J.

For count>P, CALL LEXPERM returns −1.

If the CALL LEXPERM routine is executed with the first argument out of sequence, the results might not be useful. In particular, if you initialize the variables and then immediately execute CALL LEXPERM with a first argument of K, you do not get the Kth permutation (except when K is 1). To get the Kth permutation, you must execute CALL LEXPERM K times. The first argument accepts values from 1 through K in that exact order.

Using the CALL LEXPERM Routine with Macros
You can call the LEXPERM routine when you use the %SYSCALL macro. In this case, the variable arguments are not required to be the same length, but they must be the same type. If %SYSCALL identifies an argument as numeric, then %SYSCALL reformats the returned value.

If an error occurs during the execution of the CALL LEXPERM routine, then both of the following values are set:
• &SYSERR is assigned a value that is greater than 4.
• &SYSINFO is assigned a value that is less than −100.

If there are no errors, then &SYSERR is set to zero, and &SYSINFO is set to one of the following values:
• 1 if 1<count<P
• 1 if 1<count<=P and the value of variable-1 changed
• J if 1<count<=P and variable-1 through variable-I did not change, but variable-J did change, where J=I+1
• −1 if count>P

Comparisons
SAS provides three functions or CALL routines for generating all permutations:
• ALLPERM generates all possible permutations of the values, missing or nonmissing, of several variables. Each permutation is formed from the previous permutation by interchanging two consecutive values.
• LEXPERM generates all distinct permutations of the nonmissing values of several variables. The permutations are generated in lexicographic order.
• LEXPERK generates all distinct permutations of K of the nonmissing values of N variables. The permutations are generated in lexicographic order.

ALLPERM is the fastest of these functions and CALL routines. LEXPERK is the slowest.
Examples

Example 1: Using CALL LEXPERM in a DATA Step
The following example uses the DATA step to generate all distinct permutations of the nonmissing values of several variables in lexicographic order.

```sas
data _null_;  
array x[4] $3 ('ant' 'bee' 'cat' 'dog');  
n=dim(x);  
nfact=fact(n);  
do i=1 to nfact;  
call lexperm(i, of x[*]);  
    put i 5. +2 x[*];  
end;  
run;
```

SAS writes the following output to the log:

```
1  ant bee cat dog  
2  ant bee dog cat  
3  ant cat bee dog  
4  ant cat dog bee  
5  ant dog bee cat  
6  ant dog cat bee  
7  bee ant cat dog  
8  bee ant dog cat  
9  bee cat ant dog  
10  bee cat dog ant  
11  bee dog ant cat  
12  bee dog cat ant  
13  cat ant bee dog  
14  cat ant dog bee  
15  cat bee ant dog  
16  cat bee dog ant  
17  cat dog ant bee  
18  cat dog bee ant  
19  dog ant bee cat  
20  dog ant cat bee  
21  dog bee ant cat  
22  dog bee cat ant  
23  dog cat ant bee  
24  dog cat bee ant
```

Example 2: Using CALL LEXPERM with Macros
The following is an example of the CALL LEXPERM routine that is used with macros. The output includes values for the %SYSINFO macro.

```sas
%macro test;  
    %let x1=ant;  
    %let x2=baboon;  
    %let x3=baboon;  
    %let x4=hippopotamus;  
    %let n=4;  
    %let nperm=%sysfunc(perm(4));  
    %do j=1 %to &nperm;  
        %syscall lexperm(j, x1, x2, x3, x4);  
        %let jfmt=%qsysfunc(putn(&j, 5.));  
        %put &jfmt: &x1 &x2 &x3 &x4 sysinfo=&sysinfo;  
        %if &sysinfo<0 %then %let j=%eval(&nperm+1);  
    %end;
```

CALL LEXPERM Routine
773
SAS writes the following output to the log:

1: ant baboon baboon hippopotamus sysinfo=1
2: ant baboon hippopotamus baboon sysinfo=3
3: ant hippopotamus baboon baboon sysinfo=2
4: baboon ant baboon hippopotamus sysinfo=1
5: baboon ant hippopotamus baboon sysinfo=3
6: baboon baboon ant hippopotamus sysinfo=2
7: baboon baboon hippopotamus ant sysinfo=3
8: baboon hippopotamus ant baboon sysinfo=2
9: baboon hippopotamus baboon ant sysinfo=3
10: hippopotamus ant baboon baboon sysinfo=1
11: hippopotamus baboon ant baboon sysinfo=2
12: hippopotamus baboon baboon ant sysinfo=3
13: hippopotamus baboon baboon ant sysinfo=-1

See Also

Functions:
- “LEXPERK Function” on page 981
- “LEXPERM Function” on page 984

CALL Routines:
- “CALL ALLPERM Routine” on page 733
- “CALL RANPERK Routine” on page 829
- “CALL RANPERM Routine” on page 831

CALL LOGISTIC Routine

Applies the logistic function to each argument.

Category: Mathematical
Restriction: This function is not valid on the CAS server.
Note: Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

Syntax

CALL LOGISTIC(argument<, argument, ...>);

Required Argument

argument

is a numeric variable.
Restriction

The CALL LOGISTIC routine accepts only variables as valid arguments. Do not use a constant or a SAS expression because the CALL routine is unable to update these arguments.

Details

The CALL LOGISTIC routine replaces each argument with the logistic value of that argument. For example, an arbitrary argument $x_j$ is replaced by

$$
\frac{e^{x_j}}{1 + e^{x_j}}
$$

If any argument contains a missing value, then CALL LOGISTIC returns missing values for all the arguments.

Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=0.5;</td>
<td>x=0.6224593312</td>
</tr>
<tr>
<td>y=-0.5;</td>
<td>y=0.3775406688</td>
</tr>
<tr>
<td>call logistic(x, y);</td>
<td></td>
</tr>
<tr>
<td>put x= y=;</td>
<td></td>
</tr>
</tbody>
</table>

CALL MODULE Routine

Calls an external routine without any return code.

Category: External Routines

Restriction: This function is not valid on the CAS server.

Interaction: When a SAS server is in a locked-down state, the CALL MODULE routine does not execute.

Note: Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

Syntax

CALL MODULE(<control>, module, argument-1, argument-2 ..., argument-n);
number=MODULEN(<control>, module, argument-1, argument-2 ..., argument-n);
character=MODULEC(<control>, module, argument-1 ..., argument-2, argument-n);
CALL MODULEI(<control>, module, argument-1, argument-2 ..., argument-n);
number=MODULEIN(<control>, module, argument-1, argument-2 ..., argument-n);
character=MODULEIC(<control>, module, argument-1, argument-2 ..., argument-n);
**Required Arguments**

*module*

specifies the name of the external module to use. The *module* can be specified as a shared library with the routine name or ordinal value, separated by a comma. You do not need to specify the shared library name if you specified the MODULE attribute for the routine in the SASCBTBL attribute table, as long as the routine name is unique (that is, no other routine has the same name in the attribute file).

The module must reside in a shared library, and it must be able to be called externally. Although the shared library name is not case sensitive, the routine name is based on the restraints of the routine’s implementation language, so the routine name is case sensitive.

If the shared library supports ordinal-value naming, you can provide the shared library name followed by a decimal number, such as 'XYZ,30'.

You can specify *module* as a SAS character expression instead of as a constant. Most often, it is passed as a constant.

*argument-1, argument-2, ..., argument-n*

specifies the arguments to pass to the requested routine. Use the proper attributes for the arguments (that is, numeric arguments for numeric attributes and character arguments for character attributes).

**CAUTION:**

Be sure to use the correct arguments and attributes. If you use incorrect arguments or attributes for a shared library function, you can cause SAS to crash or you might see unexpected results.

**Optional Argument**

*control*

is an optional control string whose first character must be an asterisk (*), followed by any combination of the following characters:

I prints the hexadecimal representations of all arguments to the MODULE function and to the requested shared library routine before and after the shared library routine is called. You can use this option to help diagnose problems that are caused by incorrect arguments or attribute tables. If you specify the I option, the E option is implied.

E prints detailed error messages. Without the E option (or the I option, which supersedes it), the only error message that the MODULE function generates is "Invalid argument to function," which is usually not enough information to determine the cause of the error.

Sx uses x as a separator character to separate field definitions. You can then specify x in the argument list as its own character argument to serve as a delimiter for a list of arguments that you want to group together as a single structure. Use this option only if you do not supply an entry in the SASCBTBL attribute table. If you do supply an entry for this module in the SASCBTBL attribute table, you should use the FDSTART option in the ARG statement in the table to separate structure definitions.

H provides brief help information about the syntax of the MODULE routines, the attribute file format, and the suggested SAS formats and informats.

For example, the control string "*IS/" specifies that parameter lists be printed and that the string '/' is to be treated as a separator character in the argument list.
Details

Using IML Functions with the CALL MODULE Routine
The following functions permit vector and matrix arguments. You can use them only within the IML procedure:

- CALL MODULEI
- MODULEIN
- MODULEIC

For more information, see the SAS/IML Studio: User's Guide.

Working with Attribute Tables
The CALL MODULE routine executes a routine that resides in an external library. The routine uses the required module-name argument to identify the name of the library, as well as optional arguments.

CALL MODULE builds a parameter list by using the information in the arguments and a routine description and argument attribute table that you define in a separate file. The attribute table is a sequential text file that contains descriptions of the routines that you can invoke with the CALL MODULE routine. The purpose of the table is to define how CALL MODULE interprets its supplied arguments when it builds a parameter list to pass to the external routine. The attribute table contains a description for each external routine that you intend to call, and descriptions of each argument associated with that routine.

Before you invoke CALL MODULE, you must define the fileref of SASCBTBL to point to the external file that contains the attribute table. You can name the file whatever you want when you create it. In this way, you use SAS variables and formats as arguments to CALL MODULE and ensure that these arguments are properly converted before being passed to the external routine. If you do not define this fileref, CALL MODULE calls the requested routine without altering the arguments.

CAUTION:
Using the CALL MODULE routine without a defined attribute table can cause the SAS System to fail or force you to reset your computer. You must use an attribute table for all external functions that you want to invoke.

Using Special Formats with the CALL MODULE Routine

Numeric Alignment
By default, the CALL MODULE routine forces numeric alignment for fields that are defined with the IBw., PIBw., and RBw. formats within the SASCBTBL table. Therefore, the fields are padded to provide an appropriate 4-byte or 8-byte boundary. Most current compilers expect this alignment.

New formats were created specifically for the CALL MODULE routine to accommodate routines that were built with a compiler that does not expect alignment. These new formats are IBUNALNw., PIBUNALNw., and RBUNALNw. Use these formats instead of the IBw., PIBw., and RBw. formats.

The IBUNALNw., PIBUNALNw. and RBUNALNw. Formats
The IBUNALNw., PIBUNALNw., and RBUNALNw. formats are used only with the CALL MODULE routine. They were created because the default behavior within an FDSTART structure forces alignment when you use the IBw. and PIBw. formats. The IBUNALNw., PIBUNALNw., and RBUNALNw. formats do not align data.
This example shows a `SASCBTBL` definition, which uses the `IBw.` format with the `FDSTART` structure:

```sas
filename sascbtbl temp;
data _null_;   
  file sascbtbl; 
  input;   
  put _infile_;   
  datalines4;   
routine xyz minarg=2 maxarg=2;   
arg 1 char format=$3. fdstart;   
arg 2 num format=ib4.;   
;;;   
```

This example shows a DATA step that uses the CALL MODULE routine based on the preceding `SASCBTBL` definition:

```sas
data _null_;   
x='abc';   
y=1;   
call module('xyz', x, y);   
run;   
```

When `XYZ` is called, it receives a single argument, which is the address of an aggregate structure. The aggregate structure contains the three-bytes `'abc'` followed by an alignment byte, which is then followed by the value 1 from the `IB4.` format (`00000001`x in a big endian environment and `01000000`x in a little endian environment). If the `XYZ` routine is not expecting alignment, it misreads the numeric value.

This example shows an `SASCBTBL` definition, which uses the `IBUNALNW.` format with the `FDSTART` structure:

```sas
filename sascbtbl temp;
data _null_;   
  file sascbtbl; 
  input;   
  put _infile_;   
  datalines4;   
routine xyz minarg=2 maxarg=2;   
arg 1 char format=$3. fdstart;   
arg 2 num format=ibunaln4.;   
;;;   
```

`XYZ` is given the address of an aggregate structure that contains `'abc'` followed immediately by the value of 1, with no intervening alignment bytes.

Alignment occurs when numeric formats `IBw`, `PIBw`, or `RBw` are used. If the width is 2, the alignment occurs on an even byte boundary. If the width is 4, the alignment occurs on a 4-byte boundary. If the width is 8, the alignment occurs on an 8-byte boundary.

**Comparisons**

- The `MODULEN` and `MODULEC` functions return a number and a character, respectively. The CALL MODULE routine does not return a value.
- The CALL `MODULEI` routine and the `MODULEIC` and `MODULEIN` functions allow vector and matrix arguments. Their return values are scalar. You can invoke CALL `MODULEI`, `MODULEIC`, and `MODULEIN` routines only from the IML procedure.
Examples

Example 1: Using the CALL MODULE Routine

This example calls the XYZ routine. Use the following attribute table:

```
routine xyz minarg=2 maxarg=2;
arg 1 input num byvalue format=ib4.;
arg 2 output char format=$char10.;
```

Here is the sample SAS code that calls the XYZ function:

```
data _null_;  
call module('xyz', 1, x);  
run;
```

Example 2: Using the MODULEIN Function in the IML Procedure

This example invokes the CHANGI routine from the TRYMOD.DLL module on a Windows platform. Use the following attribute table:

```
routine changi module=trymod returns=long;  
arg 1 input num format=ib4. byvalue;  
arg 2 update num format=ib4.;
```

This PROC IML code calls the CHANGI function:

```
proc iml;  
x1=J(4, 5, 0);  
do i=1 to 4;  
  do j=1 to 5;  
    x1[i,j]=i*10+j+3;  
  end;  
end;  
y1=x1;  
x2=x1;  
y2=y1;  
rc=modulein("i", 'changi', 6, x2);
```

Example 3: Using the MODULEN Function

This example calls the BEEP routine, which is part of the Win32 API in the KERNEL32 Dynamic Link Library on a Windows platform. Use the following attribute table:

```
routine Beep
  minarg=2  
  maxarg=2
  stackpop=called
  callseq=byvalue
  module=kernel32;
arg 1 num format=pib4.;
arg 2 num format=pib4.;
```

Assume that you name the attribute table file 'myatttbl.dat'. Here is the sample SAS code that calls the BEEP function:

```
filename sascbtbl 'myatttbl.dat';
data _null_;  
  rc=modulen("e", "Beep", 1380, 1000);  
run;
```

The preceding code causes the computer speaker to beep.
CALL POKE Routine

Writes a value directly into memory on a 32-bit platform.

**Category:** Special

**Restrictions:** Use on 32-bit platforms only.
This function is not valid on the CAS server.

**Interaction:** When a SAS server is in a locked-down state, the CALL POKE routine does not execute.

**Note:** Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

**Syntax**

\[
\text{CALL POKE}(\text{source}, \text{pointer} \,<, \text{length} \,<, \text{floating-point}>);
\]

**Required Arguments**

- **source**
  specifies a constant, variable, or expression that contains a value to write into memory.

- **pointer**
  specifies a numeric expression that contains the virtual address of the data that the CALL POKE routine alters.

**Optional Arguments**

- **length**
  specifies a numeric constant, variable, or expression that contains the number of bytes to write from the \text{source} to the address that is indicated by \text{pointer}. If you omit \text{length}, the action that the CALL POKE routine takes depends on whether \text{source} is a character value or a numeric value:
  - If \text{source} is a character value, the CALL POKE routine copies the entire value of \text{source} to the specified memory location.
  - If \text{source} is a numeric value, the CALL POKE routine converts \text{source} to a long integer and writes into memory the number of bytes that constitute a pointer.

- **floating-point**
  specifies that the value of \text{source} is stored as a floating-point number. The value of \text{floating-point} can be any number.

**See Also**

Functions:
- “MODULEC Function” on page 995
- “MODULEN Function” on page 995
Tip If you do not use the floating-point argument, then source is stored as an integer value.

Details

**CAUTION:**

The CALL POKE routine is intended only for experienced programmers in specific cases. If you use this routine, use extreme caution in your programming and in your typing. Typing directly into memory can cause devastating problems. This routine bypasses the normal safeguards that prevent you from destroying a vital element in your SAS session or in another piece of software that is active at the time.

If you do not have access to the memory location that you specify, the CALL POKE routine returns an invalid argument error.

If you attempt to use the CALL POKE routine on 64-bit platforms, SAS writes a message to the log stating that this action cannot be completed. If you have legacy applications that use CALL POKE, change the applications and use CALL POKELONG instead. You can use CALL POKELONG on 32-bit and 64-bit platforms.

If you use the fourth argument, a floating-point number is assumed to be the value that is stored. If you do not use the fourth argument, an integer value is assumed to be stored.

See Also

Functions:

- “ADDR Function” on page 714
- “PEEK Function” on page 1014
- “PEEKC Function” on page 1016

CALL Routines:

- “CALL POKELONG Routine” on page 781

**CALL POKELONG Routine**

Writes a value directly into memory on 32-bit and 64-bit platforms.

**Category:** Special

**Restriction:** This function is not valid on the CAS server.

**Interaction:** When a SAS server is in a locked-down state, the CALL POKELONG routine does not execute.

**Note:** Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

**Syntax**

```call pokelong(source, pointer <, length> <, floating-point>);```
Required Arguments

source
specifies a character constant, variable, or expression that contains a value to write into memory.

pointer
specifies a character string that contains the virtual address of the data that the CALL POKELONG routine alters.

Optional Arguments

length
specifies a numeric SAS expression that contains the number of bytes to write from the source to the address that is indicated by the pointer. If you omit length, the CALL POKELONG routine copies the entire value of source to the specified memory location.

floating-point
specifies that the value of source is stored as a floating-point number. The value of floating-point can be any number.

Tip If you do not use the floating-point argument, then source is stored as an integer value.

Details

CAUTION: The CALL POKELONG routine is intended only for experienced programmers in specific cases. If you use this routine, use extreme caution in your programming and in your typing. Typing directly into memory can cause devastating problems. This routine bypasses the normal safeguards that prevent you from destroying a vital element in your SAS session or in another piece of software that is active at the time.

If you do not have access to the memory location that you specify, the CALL POKELONG routine returns an Invalid argument error.

If you use the fourth argument, a floating-point number is assumed to be the value that is stored. If you do not use the fourth argument, an integer value is assumed to be stored.

See Also

CALL Routines:
• “CALL POKE Routine” on page 780

CALL PRXCHANGE Routine

Performs a pattern-matching replacement.

Category: Character String Matching

Restrictions: Use with the PRXPARSE function.
This function is not valid on the CAS server.
Do not use this function to process DBCS and MBCS data, because this CALL routine requires the PRXPARSE function, which is not DBCS compatible.
**Interaction:** When invoked by the %SYSCALL macro statement, CALL PRXCHANGE removes the quotation marks from its arguments. For more information, see Using CALL Routines and the %SYSCALL Macro Statement on page 9.

**Note:** Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

---

**Syntax**

CALL PRXCHANGE((regular-expression-id, times, old-string <, new-string <, result-length <, truncation-value <, number-of-changes> >)>

**Required Arguments**

*regular-expression-id*

specifies a numeric variable with a value that is a pattern identifier that is returned from the PRXPARSE function.

*times*

is a numeric constant, variable, or expression that specifies the number of times to search for a match and replace a matching pattern.

**Tip** If the value of *times* is -1, then all matching patterns are replaced.

*old-string*

specifies the character expression on which to perform a search and replace.

**Tip** All changes are made to *old-string* if you do not use the *new-string* argument.

**Optional Arguments**

*new-string*

specifies a character variable in which to place the results of the change to *old-string*.

**Tip** If you use the *new-string* argument in the call to the PRXCHANGE routine, then *old-string* is not modified.

*result-length*

is a numeric variable with a return value that is the number of characters that are copied into the result.

**Tip** Trailing blanks in the value of *old-string* are not copied to *new-string*, and are therefore not included as part of the length in *result-length*.

*truncation-value*

is a numeric variable with a returned value that is either 0 or 1, depending on the result of the change operation:

0 if the entire replacement result is not longer than the length of *new-string*.

1 if the entire replacement result is longer than the length of *new-string*.

*number-of-changes*

is a numeric variable with a returned value that is the total number of replacements that were made. If the result is truncated when it is placed into *new-string*, the value of *number-of-changes* is not changed.
Details
The CALL PRXCHANGE routine matches and replaces a pattern. If the value of times is -1, the replacement is performed as many times as possible.

For more information about pattern matching, see Pattern Matching Using Perl Regular Expressions (PRX) on page 42.

Comparisons
The CALL PRXCHANGE routine is similar to the PRXCHANGE function. The difference is that the CALL routine returns the value of the pattern-matching replacement as one of its parameters instead of as a return argument.

The Perl regular expression (PRX) functions and CALL routines work together to manipulate strings that match patterns. To see a list and short description of these functions and CALL routines, see the Character String Matching category in “SAS Functions and CALL Routines by Category” on page 701.

Example
The following example replaces all occurrences of cat, rat, or bat with the value TREE.

```sas
data _null_; /* Use a pattern to replace all occurrences of cat, */ /* rat, or bat with the value TREE. */ length text $46; RegularExpressionId = prxparse('s/[crb]at/tree/'); text = 'The woods have a bat, cat, bat, and a rat!'; /* Use CALL PRXCHANGE to perform the search and replace. */ /* Because the argument times has a value of -1, the */ /* replacement is performed as many times as possible. */ call prxchange(RegularExpressionId, -1, text); put text; run;
```

SAS writes the following output to the log:

```
The woods have a tree, tree, tree, and a tree!
```

See Also

Functions:
- “PRXCHANGE Function” on page 527
- “PRXPAREN Function” on page 1022
- “PRXPARSE Function” on page 538
- “PRXMATCH Function” on page 533
- “PRXPOSN Function” on page 541

CALL Routines:
- “CALL PRXDEBUG Routine” on page 785
- “CALL PRXFREE Routine” on page 127
CALL PRXDEBUG Routine

Enables Perl regular expressions in a DATA step to send debugging output to the SAS log.

**Category:** Character String Matching

**Restrictions:**
This function is not valid on the CAS server.

Use with the CALL PRXCHANGE, CALL PRXFREE, CALL PRXNEXT, CALL PRXPOSN, CALL PRXSUBSTR, PRXPARSE, PRXPAREN, and PRXMATCH functions and CALL routines. The PRXPARSE function is not DBCS compatible.

**Note:** Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

**Syntax**

CALL PRXDEBUG(on-off);

**Required Argument**

*on-off*

specifies a numeric constant, variable, or expression. If the value of *on-off* is positive and nonzero, then debugging is turned on. If the value of *on-off* is zero, then debugging is turned off.

**Details**

The CALL PRXDEBUG routine provides information about how a Perl regular expression is compiled, and about which steps are taken when a pattern is matched to a character value.

You can turn debugging on and off multiple times in your program if you want to see debugging output for particular Perl regular expression function calls.

For more information about pattern matching, see Pattern Matching Using Perl Regular Expressions (PRX) on page 42.

**Comparisons**

The Perl regular expression (PRX) functions and CALL routines work together to manipulate strings that match patterns. To see a list and short description of these functions and CALL routines, see the Character String Matching category in “SAS Functions and CALL Routines by Category” on page 701.

**Example**

The following example produces debugging output.

```sas
data _null_;  /* Turn the debugging option on. */
```
call prxdebug(1);
putlog 'PRXPARSE: ';
re = prxparse('/[bc]d(ef*g)+h[ij]k$/');
putlog 'PRXMATCH: ';
pos = prxmatch(re, 'abcdefg_gh_');
/* Turn the debugging option off. */
call prxdebug(0);
run;

SAS writes the following output to the log:

<table>
<thead>
<tr>
<th>PRXPARSE:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compiling REx '[bc]d(ef*g)+h[ij]k$'</td>
</tr>
<tr>
<td>size 41 first at 1</td>
</tr>
<tr>
<td>rarest char g at 0</td>
</tr>
<tr>
<td>rarest char d at 0</td>
</tr>
<tr>
<td>1: ANYOF<a href="10">bc</a></td>
</tr>
<tr>
<td>10: EXACT &lt;d&gt;(12)</td>
</tr>
<tr>
<td>12: CURLYX[0] {1,32767}(26)</td>
</tr>
<tr>
<td>14: OPEN1(16)</td>
</tr>
<tr>
<td>16: EXACT &lt;e&gt;(18)</td>
</tr>
<tr>
<td>18: STAR(21)</td>
</tr>
<tr>
<td>19: EXACT &lt;f&gt;(0)</td>
</tr>
<tr>
<td>21: EXACT &lt;g&gt;(23)</td>
</tr>
<tr>
<td>23: CLOSE1(25)</td>
</tr>
<tr>
<td>25: WHILEM<a href="0">1/1</a></td>
</tr>
<tr>
<td>26: NOTHING(27)</td>
</tr>
<tr>
<td>27: EXACT &lt;h&gt;(29)</td>
</tr>
<tr>
<td>29: ANYOF<a href="38">ij</a></td>
</tr>
<tr>
<td>38: EXACT &lt;k&gt;(40)</td>
</tr>
<tr>
<td>40: EOL(41)</td>
</tr>
<tr>
<td>41: END(0)</td>
</tr>
<tr>
<td>anchored 'de' at 1 floating 'gh' at 3..2147483647 (checking floating)</td>
</tr>
<tr>
<td>stclass 'ANYOF[bc]' minlen 7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PRXMATCH:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guessing start of match, REx '[bc]d(ef*g)+h[ij]k$' against 'abcdefg_gh_'...</td>
</tr>
<tr>
<td>Did not find floating substr 'gh'...</td>
</tr>
<tr>
<td>Match rejected by optimizer</td>
</tr>
</tbody>
</table>

The following items correspond to the lines that are numbered in the SAS log that is shown above.

1. This line shows the precompiled form of the Perl regular expression.
2. Size specifies a value in arbitrary units of the compiled form of the Perl regular expression. 41 is the label ID of the first node that performs a match.
3. This line begins a list of program nodes in compiled form for regular expressions.
4. These two lines provide optimizer information. In the example above, the optimizer found that the match should contain the substring de at offset 1, and the substring gh at an offset between 3 and infinity. To rule out a pattern match quickly, Perl checks substring gh before it checks substring de.

The optimizer might use the information that the match begins at the first ID (line 2), with a character class (line 5), and cannot be shorter than seven characters (line 6).

See Also

Functions:
- “PRXCHANGE Function” on page 527
- “PRXPAREN Function” on page 1022
CALL PRXNEXT Routine

Returns the position and length of a substring that matches a pattern, and iterates over multiple matches within one string.

**Category:** Character String Matching

**Restrictions:** Use with the PRXPARSE function. This function is not valid on the CAS server. Do not use this function to process DBCS and MBCS data, because this CALL routine requires the PRXPARSE function, which is not DBCS compatible.

**Interaction:** When invoked by the %SYSCALL macro statement, CALL PRXNEXT removes the quotation marks from arguments. For more information, see Using CALL Routines and the %SYSCALL Macro Statement on page 9.

**Note:** Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

**Syntax**

CALL PRXNEXT*(regular-expression-id, start, stop, source, position, length)*;

**Required Arguments**

*regular-expression-id*

specifies a numeric variable with a value that is the identification number that is returned by the PRXPARSE function.

*start*

is a numeric variable that specifies the position at which to start the pattern matching in *source*. If the match is successful, CALL PRXNEXT returns a value of *position* + MAX(1, *length*). If the match is not successful, the value of *start* is not changed.

*stop*

is a numeric constant, variable, or expression that specifies the last character to use in *source*. If *stop* is -1, then the last character is the last non-blank character in *source*. 
source
specifies a character constant, variable, or expression that you want to search.

position
is a numeric variable with a returned value that is the position in source at which the pattern begins. If no match is found, CALL PRXNEXT returns zero.

length
is a numeric variable with a returned value that is the length of the string that is matched by the pattern. If no match is found, CALL PRXNEXT returns zero.

Details
The CALL PRXNEXT routine searches the variable source with a pattern. It returns the position and length of a pattern match that is located between the start and stop positions in source. Because the value of the start parameter is updated to be the position of the next character that follows a match, CALL PRXNEXT enables you to search a string for a pattern multiple times in succession.

For more information about pattern matching, see Pattern Matching Using Perl Regular Expressions (PRX) on page 42.

Comparisons
The Perl regular expression (PRX) functions and CALL routines work together to manipulate strings that match patterns. To see a list and short description of these functions and CALL routines, see the Character String Matching category in “SAS Functions and CALL Routines by Category” on page 701.

Example
The following example finds all instances of cat, rat, or bat in a text string.

data _null_;  
  ExpressionID = prxparse('/[crb]at/');  
  text = 'The woods have a bat, cat, and a rat!';  
  start = 1;  
  stop = length(text);  
  /* Use PRXNEXT to find the first instance of the pattern, */  
  /* then use DO WHILE to find all further instances. */  
  /* PRXNEXT changes the start parameter so that searching */  
  /* begins again after the last match. */  
  call prxnext(ExpressionID, start, stop, text, position, length);  
  do while (position > 0);  
    found = substr(text, position, length);  
    put found= position= length=;  
    call prxnext(ExpressionID, start, stop, text, position, length);  
  end;  
run;

SAS writes the following output to the log:

found=bat position=18 length=3
found=cat position=23 length=3
found=rat position=34 length=3
CALL PRXPOSN Routine

Returns the start position and length for a capture buffer.

**Category:** Character String Matching

**Restrictions:**
- Use with the PRXPARSE function.
- This function is not valid on the CAS server.
- Do not use this function to process DBCS and MBCS data, because this CALL routine requires the PRXPARSE function, which is not DBCS compatible.

**Note:** Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

**Syntax**

```call
CALL PRXPOSN(regular-expression-id, capture-buffer, start <, length>);
```

**Required Arguments**

- `regular-expression-id`
  - specifies a numeric variable with a value that is a pattern identifier that is returned by the PRXPARSE function.

- `capture-buffer`
  - is a numeric constant, variable, or expression with a value that identifies the capture buffer from which to retrieve the start position and length:
    - If the value of `capture-buffer` is zero, CALL PRXPOSN returns the start position and length of the entire match.
    - If the value of `capture-buffer` is between 1 and the number of open parentheses, CALL PRXPOSN returns the start position and length for that capture buffer.
• If the value of `capture-buffer` is greater than the number of open parentheses, CALL PRXPOSN returns missing values for the start position and length.

`start` is a numeric variable with a returned value that is the position at which the capture buffer is found:

• If the value of `capture-buffer` is not found, CALL PRXPOSN returns a zero value for the start position.
• If the value of `capture-buffer` is greater than the number of open parentheses in the pattern, CALL PRXPOSN returns a missing value for the start position.

**Optional Argument**

`length` is a numeric variable with a returned value that is the pattern length of the previous pattern match:

• If the pattern match is not found, CALL PRXPOSN returns a zero value for the length.
• If the value of `capture-buffer` is greater than the number of open parentheses in the pattern, CALL PRXPOSN returns a missing value for length.

**Details**

The CALL PRXPOSN routine uses the results of PRXMATCH, PRXSUBSTR, PRXCHANGE, or PRXNEXT to return a capture buffer. A match must be found by one of these functions for the CALL PRXPOSN routine to return meaningful information.

A capture buffer is part of a match, enclosed in parentheses, that is specified in a regular expression. CALL PRXPOSN does not return the text for the capture buffer directly. It requires a call to the SUBSTR function to return the text.

For more information about pattern matching, see Pattern Matching Using Perl Regular Expressions (PRX) on page 42.

**Comparisons**

The CALL PRXPOSN routine is similar to the PRXPOSN function, except that CALL PRXPOSN returns the position and length of the capture buffer rather than the capture buffer itself.

The Perl regular expression (PRX) functions and CALL routines work together to manipulate strings that match patterns. To see a list and short description of these functions and CALL routines, see the Character String Matching category in “SAS Functions and CALL Routines by Category” on page 701.

**Examples**

**Example 1: Finding Submatches within a Match**

The following example searches a regular expression and calls the PRXPOSN routine to find the position and length of three submatches.

```sas
data _null_;  
patternID = prxparse('/\d\d:\d\d\:am|pm/');  
text = 'The time is 09:56am.';  
if prxmatch(patternID, text) then do;
```
call prxposn(patternID, 1, position, length);
hour = substr(text, position, length);
call prxposn(patternID, 2, position, length);
minute = substr(text, position, length);
call prxposn(patternID, 3, position, length);
ampm = substr(text, position, length);
put hour= minute= ampm=;
put text=;
end;
run;

SAS writes the following output to the log:

| hour=09 minute=56 ampm=am |
| text=The time is 09:56am. |

**Example 2: Parsing Time Data**
The following example parses time data and writes the results to the SAS log.

data _null_; if _N_ = 1 then do; retain patternID; pattern = "/(\d+):(.\d\d)(?:\.(\d+))?/"; patternID = prxparse(pattern); end;
array match[3] $ 8;
input minsec $80.;
position = prxmatch(patternID, minsec);
if position ^= 0 then do;
    do i = 1 to prxparen(patternID);
call prxposn(patternID, i, start, length);
    if start ^= 0 then
        match[i] = substr(minsec, start, length);
    end;
    if ^missing(match[3]) then
        put ", " match[3] "milliseconds";
    end;
datalines;
14:56.456
45:32
;
SAS writes the following output to the log:

| 14 minutes, 56 seconds, 456 milliseconds |
| 45 minutes, 32 seconds |

**See Also**

**Functions:**
- “PRXCHANGE Function” on page 527
CALL PRXSUBSTR Routine

Returns the position and length of a substring that matches a pattern.

**Category:** Character String Matching

**Restrictions:**
- Use with the PRXPARSE function.
- This function is not valid on the CAS server.
- Do not use this function to process DBCS and MBCS data, because this CALL routine requires the PRXPARSE function, which is not DBCS compatible.

**Interaction:**
When invoked by the %SYSCALL macro statement, CALL PRXSUBSTR removes the quotation marks from its arguments. For more information, see Using CALL Routines and the %SYSCALL Macro Statement on page 9.

**Note:** Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

**Syntax**

```
CALL PRXSUBSTR(regular-expression-id, source, position <, length>);
```

**Required Arguments**

- **regular-expression-id** specifies a numeric variable with a value that is an identification number that is returned by the PRXPARSE function.
- **source** specifies a character constant, variable, or expression that you want to search.
- **position** is a numeric variable with a returned value that is the position in `source` where the pattern begins. If no match is found, CALL PRXSUBSTR returns zero.
**Optional Argument**

*length*

is a numeric variable with a returned value that is the length of the substring that is matched by the pattern. If no match is found, CALL PRXSUBSTR returns zero.

**Details**

The CALL PRXSUBSTR routine searches the variable *source* with the pattern from PRXPARSE, returns the position of the start of the string, and if specified, returns the length of the string that is matched. By default, when a pattern matches more than one character that begins at a specific position, CALL PRXSUBSTR selects the longest match.

For more information about pattern matching, see Pattern Matching Using Perl Regular Expressions (PRX) on page 42.

**Comparisons**

CALL PRXSUBSTR performs the same matching as PRXMATCH, but CALL PRXSUBSTR also enables you to use the *length* argument to receive more information about the match.

The Perl regular expression (PRX) functions and CALL routines work together to manipulate strings that match patterns. To see a list and short description of these functions and CALL routines, see the Character String Matching category in “SAS Functions and CALL Routines by Category” on page 701.

**Examples**

**Example 1: Finding the Position and Length of a Substring**

The following example searches a string for a substring, and returns its position and length in the string.

```sas
data _null_; /* Use PRXPARSE to compile the Perl regular expression. */
   patternID = prxparse('/world/');
   /* Use PRXSUBSTR to find the position and length of the string. */
   call prxsubstr(patternID, 'Hello world!', position, length);
   put position= length=;
run;
```

SAS writes the following output to the log:

```
position=7 length=5
```

**Example 2: Finding a Match in a Substring**

The following example searches for addresses that contain avenue, drive, or road, and extracts the text that was found.

```sas
data _null_; 
   if _N_ = 1 then do;
      retain ExpressionID;
      /* The i option specifies a case insensitive search. */
      pattern = "/*aven*/|drive|rd|road/i*;
      ExpressionID = prxparse(pattern);
```
end;
input street $80.;
call prxsubstr(ExpressionID, street, position, length);
if position ^= 0 then
do;
   match = substr(street, position, length);
   put match:$QUOTE. "found in " street:$QUOTE.;
end;
datalines;
153 First Street
6789 64th Ave
4 Moritz Road
7493 Wilkes Place
;
run;

SAS writes the following output to the log:

"Ave" found in "6789 64th Ave"
"Road" found in "4 Moritz Road"

See Also

Functions:
• "PRXCHANGE Function" on page 527
• "PRXMATCH Function" on page 533
• "PRXPAREN Function" on page 1022
• "PRXPARSE Function" on page 538
• "PRXPOSN Function" on page 541

CALL Routines:
• "CALL PRXCHANGE Routine" on page 782
• "CALL PRXDEBUG Routine" on page 785
• "CALL PRXFREE Routine" on page 127
• "CALL PRXNEXT Routine" on page 787
• "CALL PRXPOSN Routine" on page 789

CALL RANBIN Routine

Returns a random variate from a binomial distribution.

Category: Random Number

Restriction: This function is not valid on the CAS server.

Note: Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.
CALL RANBIN Routine

**Syntax**

CALL RANBIN(\(seed, n, p, x\));

**Required Arguments**

**seed**

is the seed value. A new value for \(seed\) is returned each time CALL RANBIN is executed.

- **Range** \(seed < 2^{31} - 1\)

**Note**

If \(seed \leq 0\), the time of day is used to initialize the seed stream.

**See**

Seed Values on page 11 and Comparison of Seed Values in Random-Number Functions and CALL Routines on page 15 for more information about seed values.

**n**

is an integer number of independent Bernoulli trials.

- **Range** \(n > 0\)

**p**

is a numeric probability of success parameter.

- **Range** \(0 < p < 1\)

**x**

is a numeric SAS variable. A new value for the random variate \(x\) is returned each time CALL RANBIN is executed.

**Details**

The CALL RANBIN routine updates \(seed\) and returns a variate \(x\) that is generated from a binomial distribution with mean \(np\) and variance \(np(1-p)\). If \(n \leq 50\), \(np \leq 5\), or \(n(1-p) \leq 5\), SAS uses an inverse transform method applied to a RANUNI uniform variate. If \(n > 50\), \(np > 5\), and \(n(1-p) > 5\), SAS uses the normal approximation to the binomial distribution. In that case, the Box-Muller transformation of RANUNI uniform variates is used.

By adjusting the seeds, you can force streams of variates to agree or disagree for some or all of the observations in the same, or in subsequent, DATA steps.

For a discussion and example of an effective use of the random number CALL routines, see Starting, Stopping, and Restarting a Stream on page 26.

**Comparisons**

The CALL RANBIN routine gives greater control of the seed and random number streams than does the RANBIN function.

**Example**

The following example uses the CALL RANBIN routine:

```sas
data u1 (keep=x);
  seed=104;
```
do i=1 to 5;
   call ranbin(seed, 2000, 0.2 ,x);
   output;
end;
call symputx('seed', seed);
run;

data u2 (keep=x);
   seed=&seed;
   do i=1 to 5;
      call ranbin(seed, 2000, 0.2 ,x);
      output;
   end;
run;

data all;
   set u1 u2;
   z=ranbin(104, 2000, 0.2);
run;

proc print label;
   label x='Separate Streams' z='Single Stream';
run;

Figure 3.2  Output from the CALL RANBIN Routine

<table>
<thead>
<tr>
<th>Obs</th>
<th>Separate Streams</th>
<th>Single Stream</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>423</td>
<td>423</td>
</tr>
<tr>
<td>2</td>
<td>418</td>
<td>418</td>
</tr>
<tr>
<td>3</td>
<td>403</td>
<td>403</td>
</tr>
<tr>
<td>4</td>
<td>394</td>
<td>394</td>
</tr>
<tr>
<td>5</td>
<td>429</td>
<td>429</td>
</tr>
<tr>
<td>6</td>
<td>369</td>
<td>369</td>
</tr>
<tr>
<td>7</td>
<td>413</td>
<td>413</td>
</tr>
<tr>
<td>8</td>
<td>417</td>
<td>417</td>
</tr>
<tr>
<td>9</td>
<td>400</td>
<td>400</td>
</tr>
<tr>
<td>10</td>
<td>383</td>
<td>383</td>
</tr>
</tbody>
</table>

See Also

Functions:
CALL RANCAU Routine

Returns a random variate from a Cauchy distribution.

**Category:** Random Number

**Restriction:** This function is not valid on the CAS server.

**Note:** Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

**Syntax**

CALL RANCAU(seed, x);

**Required Arguments**

*seed* is the seed value. A new value for *seed* is returned each time CALL RANCAU is executed.

<table>
<thead>
<tr>
<th>Range</th>
<th>$seed &lt; 2^{31} - 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Note</strong></td>
<td>If $seed \leq 0$, the time of day is used to initialize the seed stream.</td>
</tr>
</tbody>
</table>

*x* is a numeric SAS variable. A new value for the random variate *x* is returned each time CALL RANCAU is executed.

**Details**

The CALL RANCAU routine updates *seed* and returns a variate *x* that is generated from a Cauchy distribution that has a location parameter of 0 and scale parameter of 1.

By adjusting the seeds, you can force streams of variates to agree or disagree for some or all of the observations in the same, or in subsequent, DATA steps.

An acceptance-rejection procedure applied to RANUNI uniform variates is used. If $u$ and $v$ are independent uniform $(-1/2, 1/2)$ variables and $u^2 + v^2 \leq 1/4$, then $u/v$ is a Cauchy variate.

For a discussion and example of an effective use of the random number CALL routines, see Starting, Stopping, and Restarting a Stream on page 26.

**Comparisons**

The CALL RANCAU routine gives greater control of the seed and random number streams than does the RANCAU function.
Example

This example uses the CALL RANCAU routine.

data case;
  retain Seed_1 Seed_2 Seed_3 45;
  do i=1 to 10;
    call rancau(Seed_1, X1);
    call rancau(Seed_2, X2);
    X3=rancau(Seed_3);
    if i=5 then
      do;
        Seed_2=18;
        Seed_3=18;
      end;
  output;
end;
run;

proc print;
  id i;
  var Seed_1-Seed_3 X1-X3;
run;

Figure 3.3  Output from the CALL RANCAU Routine

Here is another example of the CALL RANCAU routine:

data u1(keep=x);
  seed=104;
  do i=1 to 5;
    call rancau(seed, X);
    output;
  end;
  call symputx('seed', seed);

run;

data u2(keep=x);
    seed=&seed;
    do i=1 to 5;
        call rancau(seed, X);
        output;
    end;
run;

data all;
    set u1 u2;
    z=rancau(104);
run;

proc print label;
    label x='Separate Streams' z='Single Stream';
run;

Figure 3.4  Output from the CALL RANCAU Routine

<table>
<thead>
<tr>
<th>Obs</th>
<th>Separate Streams</th>
<th>Single Stream</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-0.6780</td>
<td>-0.6780</td>
</tr>
<tr>
<td>2</td>
<td>0.1712</td>
<td>0.1712</td>
</tr>
<tr>
<td>3</td>
<td>1.1372</td>
<td>1.1372</td>
</tr>
<tr>
<td>4</td>
<td>0.1478</td>
<td>0.1478</td>
</tr>
<tr>
<td>5</td>
<td>16.6536</td>
<td>16.6536</td>
</tr>
<tr>
<td>6</td>
<td>0.0747</td>
<td>0.0747</td>
</tr>
<tr>
<td>7</td>
<td>-0.5872</td>
<td>-0.5872</td>
</tr>
<tr>
<td>8</td>
<td>1.4713</td>
<td>1.4713</td>
</tr>
<tr>
<td>9</td>
<td>0.1792</td>
<td>0.1792</td>
</tr>
<tr>
<td>10</td>
<td>-0.0473</td>
<td>-0.0473</td>
</tr>
</tbody>
</table>

See Also

Functions:
- “RANCAU Function” on page 1031
- “RAND Function” on page 555
CALL RANEXP Routine

Returns a random variate from an exponential distribution.

**Category:** Random Number

**Restriction:** This function is not valid on the CAS server.

**Note:** Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

**Syntax**

CALL RANEXP(seed, x);

**Required Arguments**

**seed**

is the seed value. A new value for *seed* is returned each time CALL RANEXP is executed.

- **Range**: *seed* < $2^{31} - 1$
- **Note**: If *seed* $\leq 0$, the time of day is used to initialize the seed stream.

**x**

is a numeric variable. A new value for the random variate *x* is returned each time CALL RANEXP is executed.

**Details**

The CALL RANEXP routine updates *seed* and returns a variate *x* that is generated from an exponential distribution that has a parameter of 1.

By adjusting the seeds, you can force streams of variates to agree or disagree for some or all of the observations in the same, or in subsequent, DATA steps.

The CALL RANEXP routine uses an inverse transform method applied to a RANUNI uniform variate.

For a discussion and example of an effective use of the random number CALL routines, see Starting, Stopping, and Restarting a Stream on page 26.

**Comparisons**

The CALL RANEXP routine gives greater control of the seed and random number streams than does the RANEXP function.

**Example**

This example uses the CALL RANEXP routine:
data u1(keep=x);
   seed=104;
   do i=1 to 5;
      call ranexp(seed, x);
      output;
   end;
   call symputx('seed', seed);
run;

data u2(keep=x);
   seed=&seed;
   do i=1 to 5;
      call ranexp(seed, x);
      output;
   end;
run;

data all;
   set u1 u2;
   z=ranexp(104);
run;

proc print label;
   label x='Separate Streams' z='Single Stream';
run;

Figure 3.5  Output from the CALL RANEXP Routine

<table>
<thead>
<tr>
<th>Obs</th>
<th>Separate Streams</th>
<th>Single Stream</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.44347</td>
<td>1.44347</td>
</tr>
<tr>
<td>2</td>
<td>0.11740</td>
<td>0.11740</td>
</tr>
<tr>
<td>3</td>
<td>0.54175</td>
<td>0.54175</td>
</tr>
<tr>
<td>4</td>
<td>0.02280</td>
<td>0.02280</td>
</tr>
<tr>
<td>5</td>
<td>0.16645</td>
<td>0.16645</td>
</tr>
</tbody>
</table>

See Also

Functions:

- “RAND Function” on page 555
- “RANEXP Function” on page 1032
CALL RANGAM Routine

Returns a random variate from a gamma distribution.

**Category:** Random Number

**Restriction:** This function is not valid on the CAS server.

**Note:** Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

---

**Syntax**

CALL RANGAM(seed, a, x);

**Required Arguments**

**seed**

is the seed value. A new value for seed is returned each time CALL RANGAM is executed.

Range $seed < 2^{31} - 1$

Note If $seed \leq 0$, the time of day is used to initialize the seed stream.

See Seed Values on page 11 and Comparison of Seed Values in Random-Number Functions and CALL Routines on page 15 for more information about seed values

**a**

is a numeric shape parameter.

Range $a > 0$

**x**

is a numeric variable. A new value for the random variate $x$ is returned each time CALL RANGAM is executed.

---

**Details**

The CALL RANGAM routine updates seed and returns a variate $x$ that is generated from a gamma distribution with parameter $a$.

By adjusting the seeds, you can force streams of variates to agree or disagree for some or all of the observations in the same, or in subsequent, DATA steps.

For $a > 1$, an acceptance-rejection method by Cheng is used (Cheng, 1977). For $a \leq 1$, an acceptance-rejection method by Fishman is used (Fishman, 1978). For more information see “References” on page 1111.

For a discussion and example of an effective use of the random number CALL routines, see Starting, Stopping, and Restarting a Stream on page 26.
Comparisons

The CALL RANGAM routine gives greater control of the seed and random number streams than does the RANGAM function.

Examples

Example 1

This example uses the CALL RANGAM routine:

```sas
data u1(keep=x);
  seed=104;
  do i=1 to 5;
    call rangam(seed, 1, x);
    output;
  end;
  call symputx('seed', seed);
run;

data u2(keep=x);
  seed=&seed
  do i=1 to 5;
    call rangam(seed, 1, x);
    output;
  end;
run;

data all;
  set u1 u2;
  z=rangam(104, 1);
run;

proc print label;
  label x='Separate Streams' z='Single Stream';
run;
```

*Figure 3.6 Output from the CALL RANGAM Routine*

<table>
<thead>
<tr>
<th>Obs</th>
<th>Separate Streams</th>
<th>Single Stream</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.44347</td>
<td>1.44347</td>
</tr>
<tr>
<td>2</td>
<td>0.11740</td>
<td>0.11740</td>
</tr>
<tr>
<td>3</td>
<td>0.54175</td>
<td>0.54175</td>
</tr>
<tr>
<td>4</td>
<td>0.02280</td>
<td>0.02280</td>
</tr>
<tr>
<td>5</td>
<td>0.16645</td>
<td>0.16645</td>
</tr>
</tbody>
</table>
Example 2
This is another example that uses the CALL RANGAM routine:

```sas
data case;
  retain Seed_1 Seed_2 Seed_3 45;
  a=2;
  do i=1 to 10;
    call rangam(Seed_1, a, X1);
    call rangam(Seed_2, a, X2);
    X3=rangam(Seed_3, a);
    if i=5 then
      do;
        Seed_2=18;
        Seed_3=18;
      end;
    output;
  end;
run;

proc print;
  id i;
  var Seed_1-Seed_3 X1-X3;
run;
```

Figure 3.7 Output from the CALL RANGAM Routine

<table>
<thead>
<tr>
<th>i</th>
<th>Seed_1</th>
<th>Seed_2</th>
<th>Seed_3</th>
<th>X1</th>
<th>X2</th>
<th>X3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1404437564</td>
<td>1404437564</td>
<td>45</td>
<td>1.30569</td>
<td>1.30569</td>
<td>1.30569</td>
</tr>
<tr>
<td>2</td>
<td>1326029789</td>
<td>1326029789</td>
<td>45</td>
<td>1.87514</td>
<td>1.87514</td>
<td>1.87514</td>
</tr>
<tr>
<td>3</td>
<td>1988843719</td>
<td>1988843719</td>
<td>45</td>
<td>1.71597</td>
<td>1.71597</td>
<td>1.71597</td>
</tr>
<tr>
<td>4</td>
<td>50049159</td>
<td>50049159</td>
<td>45</td>
<td>1.59304</td>
<td>1.59304</td>
<td>1.59304</td>
</tr>
<tr>
<td>5</td>
<td>802575599</td>
<td>18</td>
<td>18</td>
<td>0.43342</td>
<td>0.43342</td>
<td>0.43342</td>
</tr>
<tr>
<td>6</td>
<td>100573943</td>
<td>991271755</td>
<td>18</td>
<td>1.11812</td>
<td>1.32646</td>
<td>1.11812</td>
</tr>
<tr>
<td>7</td>
<td>1986749826</td>
<td>1437043694</td>
<td>18</td>
<td>0.68415</td>
<td>0.68806</td>
<td>0.68415</td>
</tr>
<tr>
<td>8</td>
<td>52428589</td>
<td>959908645</td>
<td>18</td>
<td>1.62296</td>
<td>2.46091</td>
<td>1.62296</td>
</tr>
<tr>
<td>9</td>
<td>1216356463</td>
<td>1225034217</td>
<td>18</td>
<td>2.26435</td>
<td>4.06596</td>
<td>2.26435</td>
</tr>
<tr>
<td>10</td>
<td>805366679</td>
<td>425626811</td>
<td>18</td>
<td>2.16723</td>
<td>6.94703</td>
<td>2.16723</td>
</tr>
</tbody>
</table>

Changing Seed_2 for the CALL RANGAM statement, when i=5, forces the stream of the variates for X2 to deviate from the stream of the variates for X1. Changing Seed_3 on the RANGAM function, however, has no effect.

See Also

Functions:
CALL RANNOR Routine

Returns a random variate from a normal distribution.

**Category:** Random Number

**Restriction:** This function is not valid on the CAS server.

**Note:** Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

**Syntax**

CALL RANNOR(seed, x);

**Required Arguments**

*seed*  
is the seed value. A new value for *seed* is returned each time CALL RANNOR is executed.

<table>
<thead>
<tr>
<th>Range</th>
<th>seed &lt; $2^{31} - 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Note</td>
<td>If <em>seed</em> $\leq 0$, the time of day is used to initialize the seed stream.</td>
</tr>
</tbody>
</table>

**X**  
is a numeric variable. A new value for the random variate *x* is returned each time CALL RANNOR is executed.

**Details**

The CALL RANNOR routine updates *seed* and returns a variate *x* that is generated from a normal distribution, with mean 0 and variance 1.

By adjusting the seeds, you can force streams of variates to agree or disagree for some or all of the observations in the same, or in subsequent, DATA steps.

The CALL RANNOR routine uses the Box-Muller transformation of RANUNI uniform variates.

For a discussion and example of an effective use of the random number CALL routines, see Starting, Stopping, and Restarting a Stream on page 26.

**Comparisons**

The CALL RANNOR routine gives greater control of the seed and random number streams than does the RANNOR function.
**Example**

This example uses the CALL RANNOR routine:

```sas
data u1(keep=x);
  seed=104;
  do i=1 to 5;
    call rannor(seed, X);
    output;
  end;
  call symputx('seed', seed);
run;

data u2(keep=x);
  seed=&seed;
  do i=1 to 5;
    call rannor(seed, X);
    output;
  end;
run;

data all;
  set u1 u2;
  z=rannor(104);
run;

proc print label;
  label x='Separate Streams' z='Single Stream';
run;
```

*Figure 3.8*  
Output from the CALL RANNOR Routine

---

**The SAS System**

<table>
<thead>
<tr>
<th>Obs</th>
<th>Separate Streams</th>
<th>Single Stream</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.30390</td>
<td>1.30390</td>
</tr>
<tr>
<td>2</td>
<td>1.03049</td>
<td>1.03049</td>
</tr>
<tr>
<td>3</td>
<td>0.19491</td>
<td>0.19491</td>
</tr>
<tr>
<td>4</td>
<td>-0.34987</td>
<td>-0.34987</td>
</tr>
<tr>
<td>5</td>
<td>1.64273</td>
<td>1.64273</td>
</tr>
<tr>
<td>6</td>
<td>-1.75842</td>
<td>-1.75842</td>
</tr>
<tr>
<td>7</td>
<td>0.75080</td>
<td>0.75080</td>
</tr>
<tr>
<td>8</td>
<td>0.94375</td>
<td>0.94375</td>
</tr>
<tr>
<td>9</td>
<td>0.02436</td>
<td>0.02436</td>
</tr>
<tr>
<td>10</td>
<td>-0.97256</td>
<td>-0.97256</td>
</tr>
</tbody>
</table>
See Also

Functions:
- “RAND Function” on page 555
- “RANNOR Function” on page 1035

CALL RANPOI Routine

Returns a random variate from a Poisson distribution.

**Category:** Random Number

**Restriction:** This function is not valid on the CAS server.

**Note:** Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

**Syntax**

CALL RANPOI(\(seed, m, x\));

**Required Arguments**

\(seed\)

is the seed value. A new value for \(seed\) is returned each time CALL RANPOI is executed.

**Range** \(seed < 2^{31} - 1\)

**Note** If \(seed \leq 0\), the time of day is used to initialize the seed stream.

**See** “Seed Values” on page 11 and “Comparison of Seed Values in Random-Number Functions and CALL Routines” on page 15

\(m\)

is a numeric mean parameter.

**Range** \(m \geq 0\)

\(x\)

is a numeric variable. A new value for the random variate \(x\) is returned each time CALL RANPOI is executed.

**Details**

The CALL RANPOI routine updates \(seed\) and returns a variate \(x\) that is generated from a Poisson distribution, with mean \(m\).

By adjusting the seeds, you can force streams of variates to agree or disagree for some or all of the observations in the same, or in subsequent, DATA steps.

For \(m < 85\), an inverse transform method applied to a RANUNI uniform variate is used (Fishman, 1976; see in “References” on page 1111.) For \(m \geq 85\), the normal
approximation of a Poisson random variable is used. To expedite execution, internal variables are calculated only on initial calls (that is, with each new \( m \)).

For a discussion and example of an effective use of the random number CALL routines, see “Starting, Stopping, and Restarting a Stream” on page 26.

**Comparisons**

The CALL RANPOI routine gives greater control of the seed and random number streams than does the RANPOI function.

**Example**

This example uses the CALL RANPOI routine:

```sas
data u1(keep=x);
  seed=104;
  do i=1 to 5;
    call ranpoi(seed, 2000, x);
    output;
  end;
  call symputx('seed', seed);
run;
data u2(keep=x);
  seed=&seed;
  do i=1 to 5;
    call ranpoi(seed, 2000, x);
    output;
  end;
run;
data all;
  set u1 u2;
  z=ranpoi(104, 2000);
run;
proc print label;
  label x='Separate Streams' z = 'Single Stream';
run;
```
CALL RANTBL Routine

Returns a random variate from a tabled probability distribution.

**Category:** Random Number

**Restriction:** This function is not valid on the CAS server.

**Note:** Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

### Syntax

```
CALL RANTBL(seed, p1, ..., pi, ..., pn, x);
```
### Required Arguments

#### seed

is the seed value. A new value for seed is returned each time CALL RANTBL is executed.

**Range** \( seed < 2^{31} - 1 \)

**Note** If \( seed \leq 0 \), the time of day is used to initialize the seed stream.

**See** “Seed Values” on page 11 and “Comparison of Seed Values in Random-Number Functions and CALL Routines” on page 15

#### \( p_i \)

is a numeric SAS value.

**Range** \( 0 \leq p_i \leq 1 \) for \( 0 < i \leq n \)

#### \( x \)

is a numeric SAS variable. A new value for the random variate \( x \) is returned each time CALL RANTBL is executed.

### Details

The CALL RANTBL routine updates seed and returns a variate \( x \) generated from the probability mass function defined by \( p_1 \) through \( p_n \).

By adjusting the seeds, you can force streams of variates to agree or disagree for some or all of the observations in the same, or in subsequent, DATA steps.

An inverse transform method applied to a RANUNI uniform variate is used. The CALL RANTBL routine returns these data:

1. with probability \( p_1 \)
2. with probability \( p_2 \)
3. 
4. 
5. \( n \) with probability \( p_n \)
6. \( n + 1 \) with probability \( 1 - \sum_{i=1}^{n} p_i \) if \( \sum_{i=1}^{n} p_i \leq 1 \)

If, for some index \( j < n \),

\[
\sum_{i=1}^{j} p_i \geq 1
\]

RANTBL returns only the indices 1 through \( j \), with the probability of occurrence of the index \( j \) equal to

\[
1 - \sum_{i=1}^{j-1} p_i
\]

For a discussion and example of an effective use of the random number CALL routines, see “Starting, Stopping, and Restarting a Stream” on page 26.
Comparisons

The CALL RANTBL routine gives greater control of the seed and random number streams than does the RANTBL function.

Example

This example uses the CALL RANTBL routine:

```sas
data u1(keep=x);
   seed=104;
   do i=1 to 5;
      call rantbl(seed, .02, x);
      output;
   end;
   call symputx('seed', seed);
run;

data u2(keep=x);
   seed=&seed;
   do i=1 to 5;
      call rantbl(seed, .02, x);
      output;
   end;
run;
data all;
   set u1 u2;
   z=rantbl(104, .02);
run;
proc print label;
   label x='Separate Streams' z='Single Stream';
run;
```
CALL RANTRI Routine

Returns a random variate from a triangular distribution.

| Category:  | Random Number                                      |
| Restriction: | This function is not valid on the CAS server. |
| Note:  | Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log. |
Syntax

CALL RANTRI(seed, h, x);

Required Arguments

seed

is the seed value. A new value for seed is returned each time CALL RANTRI is executed.

Range  seed < 2^{31} - 1

Note  If seed ≤ 0, the time of day is used to initialize the seed stream.

See  “Seed Values” on page 11 and “Comparison of Seed Values in Random-Number Functions and CALL Routines” on page 15

h

is a numeric SAS value.

Range  0 < h < 1

x

is a numeric SAS variable. A new value for the random variate x is returned each time CALL RANTRI is executed.

Details

The CALL RANTRI routine updates seed and returns a variate x generated from a triangular distribution on the interval (0,1) with parameter h, which is the modal value of the distribution.

By adjusting the seeds, you can force streams of variates to agree or disagree for some or all of the observations in the same, or in subsequent, DATA steps.

The CALL RANTRI routine uses an inverse transform method applied to a RANUNI uniform variate.

For a discussion and example of an effective use of the random number CALL routines, see “Starting, Stopping, and Restarting a Stream” on page 26.

Comparisons

The CALL RANTRI routine gives greater control of the seed and random number streams than does the RANTRI function.

Example

This example uses the CALL RANTRI routine:

data u1(keep=x);
  seed=104;
  do i=1 to 5;
    call rantri(seed, .5, x);
    output;
  end;
  call symputx('seed', seed);
run;
data u2(keep=x);
seed=&seed;
do i=1 to 5;
   call rantri(seed, .5, x);
   output;
end;
run;
data all;
   set u1 u2;
   z=rantri(104, .5);
run;
proc print label;
   label x='Separate Streams' z='Single Stream';
run;

**Figure 3.11** Output from the CALL RANTRI Routine

<table>
<thead>
<tr>
<th>Obs</th>
<th>Separate Streams</th>
<th>Single Stream</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.34359</td>
<td>0.34359</td>
</tr>
<tr>
<td>2</td>
<td>0.76466</td>
<td>0.76466</td>
</tr>
<tr>
<td>3</td>
<td>0.54269</td>
<td>0.54269</td>
</tr>
<tr>
<td>4</td>
<td>0.89384</td>
<td>0.89384</td>
</tr>
<tr>
<td>5</td>
<td>0.72311</td>
<td>0.72311</td>
</tr>
<tr>
<td>6</td>
<td>0.68763</td>
<td>0.68763</td>
</tr>
<tr>
<td>7</td>
<td>0.48468</td>
<td>0.48468</td>
</tr>
<tr>
<td>8</td>
<td>0.38467</td>
<td>0.38467</td>
</tr>
<tr>
<td>9</td>
<td>0.29881</td>
<td>0.29881</td>
</tr>
<tr>
<td>10</td>
<td>0.80369</td>
<td>0.80369</td>
</tr>
</tbody>
</table>

**See Also**

**Functions:**
- “RAND Function” on page 555
- “RANTRI Function” on page 1038

**CALL RANUNI Routine**

Returns a random variate from a uniform distribution.

**Category:** Random Number

**Restriction:** This function is not valid on the CAS server.
Note: Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

Syntax

CALL RANUNI(seed, x);

Required Arguments

*seed* is the seed value. A new value for seed is returned each time CALL RANUNI is executed.

<table>
<thead>
<tr>
<th>Range</th>
<th>seed &lt; 2³¹ - 1</th>
</tr>
</thead>
</table>

Tip If seed ≤ 0, the time of day is used to initialize the seed stream.

See “Seed Values” on page 11 and “Comparison of Seed Values in Random-Number Functions and CALL Routines” on page 15

*x* is a numeric variable. A new value for the random variate x is returned each time CALL RANUNI is executed.

Details

The CALL RANUNI routine updates seed and returns a variate x that is generated from the uniform distribution on the interval (0,1), using a prime modulus multiplicative generator with modulus 2³¹−1 and multiplier 397204094 (Fishman and Moore 1982). See “References” on page 1111.

By adjusting the seeds, you can force streams of variates to agree or disagree for some or all of the observations in the same, or in subsequent, DATA steps.

For a discussion and example of an effective use of the random number CALL routines, see “Starting, Stopping, and Restarting a Stream” on page 26.

Comparisons

The CALL RANUNI routine gives greater control of the seed and random number streams than does the RANUNI function.

Example: Using the CALL RANUNI Routine

This example uses the CALL RANUNI routine:

```plaintext
data u1(keep=x);
  seed=104;
  do i=1 to 5;
    call ranuni(seed, x);
    output;
  end;
  call symputx('seed', seed);
run;
data u2(keep=x);
  seed=&seed;
```
do i=1 to 5;
    call ranuni(seed, x);
    output;
end;
run;

data all;
    set u1 u2;
    z=ranuni(104);
run;
proc print label;
    label x='Separate Streams' z='Single Stream';
run;

Figure 3.12  Output from the CALL RANUNI Routine

The SAS System

<table>
<thead>
<tr>
<th>Obs</th>
<th>Separate Streams</th>
<th>Single Stream</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.23611</td>
<td>0.23611</td>
</tr>
<tr>
<td>2</td>
<td>0.88923</td>
<td>0.88923</td>
</tr>
<tr>
<td>3</td>
<td>0.58173</td>
<td>0.58173</td>
</tr>
<tr>
<td>4</td>
<td>0.97746</td>
<td>0.97746</td>
</tr>
<tr>
<td>5</td>
<td>0.84667</td>
<td>0.84667</td>
</tr>
<tr>
<td>6</td>
<td>0.80484</td>
<td>0.80484</td>
</tr>
<tr>
<td>7</td>
<td>0.46983</td>
<td>0.46983</td>
</tr>
<tr>
<td>8</td>
<td>0.29594</td>
<td>0.29594</td>
</tr>
<tr>
<td>9</td>
<td>0.17858</td>
<td>0.17858</td>
</tr>
<tr>
<td>10</td>
<td>0.92292</td>
<td>0.92292</td>
</tr>
</tbody>
</table>

See Also

Functions:
- “RAND Function” on page 555
- “RANUNI Function” on page 1039

CALL SCAN Routine

Returns the position and length of the $n$th word from a character string.

Category: Character
Interaction: When invoked by the %SYSCALL macro statement, CALL SCAN removes the quotation marks from its arguments. For more information, see “Using CALL Routines and the %SYSCALL Macro Statement” on page 9.

Note: Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

Syntax

CALL SCAN(<string>, count, position, length <, <character-list> <, <modifier(s)>>);

Required Arguments

**count**

is a nonzero numeric constant, variable, or expression that has an integer value that specifies the number of the word in the character string that you want the CALL SCAN routine to select. For example, a value of 1 indicates the first word, a value of 2 indicates the second word, and so on. The following rules apply:

- If `count` is positive, then CALL SCAN counts words from left to right in the character string.
- If `count` is negative, then CALL SCAN counts words from right to left in the character string.

**position**

specifies a numeric variable in which the position of the word is returned. If `count` exceeds the number of words in the string, then the value that is returned in `position` is zero. If `count` is zero or missing, then the value that is returned in `position` is missing.

**length**

specifies a numeric variable in which the length of the word is returned. If `count` exceeds the number of words in the string, then the value that is returned in `length` is zero. If `count` is zero or missing, then the value that is returned in `length` is missing.

Optional Arguments

**string**

specifies a character constant, variable, or expression.

**character-list**

specifies an optional character constant, variable, or expression that initializes a list of characters. This list determines which characters are used as the delimiters that separate words. The following rules apply:

- By default, all characters in `character-list` are used as delimiters.
- If you specify the K modifier in the `modifier` argument, then all characters that are not in `character-list` are used as delimiters.

Tip You can add more characters to `character-list` by using other modifiers.

**modifier**

specifies a character constant, variable, or expression in which each non-blank character modifies the action of the CALL SCAN routine. Blanks are ignored. You can use the following characters as modifiers:
a or A adds alphabetic characters to the list of characters.
b or B scans backward, from right to left instead of from left to right, regardless of the sign of the count argument.
c or C adds control characters to the list of characters.
d or D adds digits to the list of characters.
f or F adds an underscore and English letters (that is, valid first characters in a SAS variable name using VALIDVARNAME=V7) to the list of characters.
g or G adds graphic characters to the list of characters. Graphic characters are those that, when printed, produce an image on paper.
h or H adds a horizontal tab to the list of characters.
i or I ignores the case of the characters.
k or K causes all characters that are not in the list of characters to be treated as delimiters. That is, if K is specified, then characters that are in the list of characters are kept in the returned value rather than being omitted because they are delimiters. If K is not specified, then all characters that are in the list of characters are treated as delimiters.
l or L adds lowercase letters to the list of characters.
m or M specifies that multiple consecutive delimiters, and delimiters at the beginning or end of the string argument, refer to words that have a length of zero. If the M modifier is not specified, then multiple consecutive delimiters are treated as one delimiter, and delimiters at the beginning or end of the string argument are ignored.
n or N adds digits, an underscore, and English letters (that is, the characters that can appear in a SAS variable name using VALIDVARNAME=V7) to the list of characters.
o or O processes the character-list and modifier arguments only once, rather than every time the CALL SCAN routine is called. Using the O modifier in the DATA step can make CALL SCAN run faster when you call it in a loop where the character-list and modifier arguments do not change. The O modifier applies separately to each instance of the CALL SCAN routine in your SAS code, and does not cause all instances of the CALL SCAN routine to use the same delimiters and modifiers.
p or P adds punctuation marks to the list of characters.
q or Q ignores delimiters that are inside substrings that are enclosed in quotation marks. If the value of the string argument contains unmatched quotation marks, then scanning from left to right produces different words than scanning from right to left.
s or S adds space characters to the list of characters (blank, horizontal tab, vertical tab, carriage return, line feed, and form feed).
t or T trims trailing blanks from the string and character-list arguments. If you want to remove trailing blanks from just one character argument instead of both character arguments, then use the TRIM function instead of the CALL SCAN routine with the T modifier.
u or U adds uppercase letters to the list of characters.
w or W    adds printable (writable) characters to the list of characters.

x or X    adds hexadecimal characters to the list of characters.

Tip    If the modifier argument is a character constant, then enclose it in quotation marks. Specify multiple modifiers in a single set of quotation marks. A modifier argument can also be expressed as a character variable or expression.

Details

Some language elements are supported in SAS but not in CAS. An example is the DATALINES statement. Some examples in this document use DATALINES in SAS to create a data set using the CAS engine. For more information, see “Functions in SAS and CAS” on page 54.

Definition of “Delimiter” and “Word”

A delimiter is any of several characters that are used to separate words. You can specify the delimiters in the character-list and modifier arguments.

If you specify the Q modifier, then delimiters inside substrings that are enclosed in quotation marks are ignored.

In the CALL SCAN routine, "word" refers to a substring that has all of the following characteristics:

• is bounded on the left by a delimiter or the beginning of the string
• is bounded on the right by a delimiter or the end of the string
• contains no delimiters

A word can have a length of zero if there are delimiters at the beginning or end of the string, or if the string contains two or more consecutive delimiters. However, the CALL SCAN routine ignores words that have a length of zero unless you specify the M modifier.

Using Default Delimiters in ASCII and EBCDIC Environments

If you use the CALL SCAN routine with only four arguments, then the default delimiters depend on whether your computer uses ASCII or EBCDIC characters.

• If your computer uses ASCII characters, then the default delimiters are as follows:
  blank ! $ % & ( ) * + , - . / ; < ^ ~
  In ASCII environments that do not contain the ^ character, the CALL SCAN routine uses the ~ character instead.

• If your computer uses EBCDIC characters, then the default delimiters are as follows:
  blank ! $ % & ( ) * + , - . / ; < ~ | ¢ £

If you use the modifier argument without specifying any characters as delimiters, then the only delimiters used are those that are defined by the modifier argument. In this case, the lists of default delimiters for ASCII and EBCDIC environments are not used. In other words, modifiers add to the list of delimiters that are explicitly specified by the character-list argument. Modifiers do not add to the list of default modifiers.
Using the CALL SCAN Routine with the M Modifier
If you specify the M modifier, then the number of words in a string is defined as one plus the number of delimiters in the string. However, if you specify the Q modifier, delimiters that are inside quotation marks are ignored.

If you specify the M modifier, the CALL SCAN routine returns a positive position and a length of zero if one of the following conditions is true:

- The string begins with a delimiter and you request the first word.
- The string ends with a delimiter and you request the last word.
- The string contains two consecutive delimiters and you request the word that is between the two delimiters.

If you specify a count that is greater in absolute value than the number of words in the string, then the CALL SCAN routine returns a position and length of zero.

Using the CALL SCAN Routine without the M Modifier
If you do not specify the M modifier, then the number of words in a string is defined as the number of maximal substrings of consecutive non-delimiters. However, if you specify the Q modifier, delimiters that are inside quotation marks are ignored.

If you do not specify the M modifier, then the CALL SCAN routine does the following:

- ignores delimiters at the beginning or end of the string
- treats two or more consecutive delimiters as if they were a single delimiter

If the string contains no characters other than delimiters, or if you specify a count that is greater in absolute value than the number of words in the string, then the CALL SCAN routine returns a position and length of zero.

Finding the Word as a Character String
To find the designated word as a character string after calling the CALL SCAN routine, use the SUBSTRN function with the string, position, and length arguments:

```plaintext
substrn(string, position, length);
```

Because CALL SCAN can return a length of zero, using the SUBSTR function can cause an error.

Using Null Arguments
The CALL SCAN routine allows character arguments to be null. Null arguments are treated as character strings with a length of zero. Numeric arguments cannot be null.

Examples

Example 1: Scanning for a Word in a String
The following example shows how you can use the CALL SCAN routine to find the position and length of a word in a string.

```plaintext
data artists;
  input string $60.;
  drop string;
  do i=1 to 99;
    call scan(string, i, position, length);
    if not position then leave;
    Name=substrn(string, position, length);
  end;
run;
```
output;
end;
datalines;
Picasso Toulouse-Lautrec Turner "Van Gogh" Velazquez
;
proc print data=artists;
run;

Figure 3.13  Output from Scanning for a Word in a String

<table>
<thead>
<tr>
<th>Obs</th>
<th>i</th>
<th>position</th>
<th>length</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>7</td>
<td>Picasso</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>9</td>
<td>8</td>
<td>Toulouse</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>18</td>
<td>7</td>
<td>Lautrec</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>26</td>
<td>6</td>
<td>Turner</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>33</td>
<td>4</td>
<td>&quot;Van</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>38</td>
<td>5</td>
<td>Gogh&quot;</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>44</td>
<td>9</td>
<td>Velazquez</td>
</tr>
</tbody>
</table>

You can transform the data set, artists, to a CAS table.

libname mycas cas;
data mycas.artists;
   set artists;
run;
proc print data=mycas.artists;
run;
Example 2: Finding the First and Last Words in a String

The following example scans a string for the first and last words. Note the following:

- A negative count instructs the CALL SCAN routine to scan from right to left.
- Leading and trailing delimiters are ignored because the M modifier is not used.
- In the last observation, all characters in the string are delimiters, so no words are found.

```sas
data firstlast;
  input String $60.;
  call scan(string, 1, First_Pos, First_Length);
  First_Word=substrn(string, First_Pos, First_Length);
  call scan(string, -1, Last_Pos, Last_Length);
  Last_Word=substrn(string, Last_Pos, Last_Length);
  datalines4;
  Jack and Jill
  & Bob & Carol & Ted & Alice & Leonardo
  ! $ % & ( ) * + - ./ ;
  ;;;
  proc print data=firstlast;
    var First: Last:;
  run;
```

Figure 3.14  Output from CALL SCAN Function in a CAS Table

<table>
<thead>
<tr>
<th>Obs</th>
<th>i</th>
<th>position</th>
<th>length</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>7</td>
<td>Picasso</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>9</td>
<td>8</td>
<td>Toulouse</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>18</td>
<td>7</td>
<td>Lautrec</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>26</td>
<td>6</td>
<td>Turner</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>33</td>
<td>4</td>
<td>&quot;Van</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>38</td>
<td>5</td>
<td>Gogh</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>44</td>
<td>9</td>
<td>Velazquez</td>
</tr>
</tbody>
</table>
Example 3: Finding All Words in a String without Using the M Modifier

The following example scans a string from left to right until no more words are found. Because the M modifier is not used, the CALL SCAN routine does not return any words that have a length of zero. Because blanks are included among the default delimiters, the CALL SCAN routine returns a position or length of zero only when the count exceeds the number of words in the string. The loop can be stopped when the returned position is less than or equal to zero. It is safer to use an inequality comparison to end the loop, rather than to use a strict equality comparison with zero, in case an error causes the position to be missing. (In SAS, a missing value is considered to have a lesser value than any nonmissing value.)

```sas
data all;
  length word $20;
  drop string;
  string=' The quick brown fox jumps over the lazy dog.   ';
  do until(position <= 0);
    count+1;
    call scan(string, count, position, length);
    word=substrn(string, position, length);
    output;
  end;
run;
proc print data=all noobs;
  var count position length word;
run;
```
The following example shows the results of using the M modifier with a comma as a delimiter. With the M modifier, leading, trailing, and multiple consecutive delimiters cause the CALL SCAN routine to return words that have a length of zero.

The O modifier is used for efficiency because the delimiters and modifiers are the same in every call to the CALL SCAN routine.

data comma;
  length word $30;
  string=',leading, trailing,and multiple,,delimiters, ,';
  do until(position <= 0);
    count + 1;
    call scan(string, count, position, length, ',', 'mo');
    word=substrn(string, position, length);
    output;
  end;
run;
proc print data=comma noobs;
  var count position length word;
run;
**Example 5: Using Comma-Separated Values, Substrings in Quotation Marks, and the O Modifier**

The following example uses the CALL SCAN routine with the O modifier and a comma as a delimiter.

The O modifier is used for efficiency because in each call of the CALL SCAN routine, the delimiters, and modifiers do not change.

```sas
data test;
  length word word_r $30;
  string='He said, "She said, "No!"", not "Yes!"';
  do until(position <= 0);
    count + 1;
    call scan(string, count, position, length, ',', 'oq');
    word=substrn(string, position, length);
    output;
  end;
run;
proc print data=test noobs;
  var count position length word;
run;
```
Example 6: Finding Substrings of Digits By Using the D and K Modifiers

The following example finds substrings of digits. The character-list argument is null, and consequently the list of characters is initially empty. The D modifier adds digits to the list of characters. The K modifier treats all characters that are not in the list as delimiters. Therefore, all characters except digits are delimiters.

data digits;
  length digits $20;
  string='Call (800) 555–1234 now!';
  do until(position <= 0);
    count+1;
    call scan(string, count, position, length, , 'dko');
    digits=substrn(string, position, length);
    output;
  end;
run;
proc print data=digits noobs;
  var count position length digits;
run;

Figure 3.19  Output from Finding Substrings of Digits By Using the D and K Modifiers
See Also

Functions:
- “COUNTW Function” on page 209
- “FINDW Function” on page 295
- “SCAN Function” on page 586

CALL RANCOMB Routine

Permutations the values of the arguments, and returns a random combination of \( k \) out of \( n \) values.

Category: Combinatorial
Restriction: This function is not valid on the CAS server.
Note: Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

Syntax

CALL RANCOMB\((seed, k, variable-1 <, variable–2, ...>)\);

Required Arguments

seed
is a numeric variable that contains the random number seed. For more information about seeds, see “Seed Values” on page 11.

\( k \)
is the number of values that you want to have in the random combination.

variable
specifies all numeric variables, or all character variables that have the same length. \( K \) values of these variables are randomly permuted.

Details

The Basics
If there are \( n \) variables, CALL RANCOMB permutes the values of the variables in such a way that the first \( k \) values are sorted in ascending order and form a random combination of \( k \) out of the \( n \) values. That is, all \( n!/(k!(n-k)!) \) combinations of \( k \) out of the \( n \) values are equally likely to be returned as the first \( k \) values.

If an error occurs during the execution of the CALL RANCOMB routine, then both of the following values are set:
- \&SYSERR is assigned a value that is greater than 4.
- \&SYSINFO is assigned a value that is less than -100.

If there are no errors, then \&SYSERR and \&SYSINFO are set to zero.
Using CALL RANCOMB with Macros

You can call the CALL RANCOMB routine when you use the %SYSCALL macro. In this case, the variable arguments are not required to be the same type or length. However, if the first k values that are returned include both character and numeric values, then those values are not sorted. If %SYSCALL identifies an argument as numeric, then %SYSCALL reformats the returned value.

Examples

Example 1: Using CALL RANCOMB in a DATA Step

The following example shows how to generate random combinations of given values by using the CALL RANCOMB routine in a DATA step.

```sas
data _null_
  array x x1-x5 (1 2 3 4 5)
  seed=1234567890123;
  do n=1 to 10;
    call rancomb(seed, 3, of x1-x5);
    put seed= @20 ' x= ' x1-x3;
  end;
run;
```

SAS writes the following output to the log:

```
seed=1332351321     x= 2 4 5
seed=829042065      x= 1 3 4
seed=767738639      x= 2 3 5
seed=1280236105     x= 2 4 5
seed=670350431      x= 1 2 5
seed=1956939964     x= 2 3 4
seed=353939815      x= 1 3 4
seed=1996660805     x= 1 2 5
seed=1835940555     x= 2 4 5
seed=910897519      x= 2 3 4
```

Example 2: Using CALL RANCOMB with a Macro

Here is an example of the CALL RANCOMB routine that is used with macros.

```sas
%macro test;
  %let x1=ant;
  %let x2=-.1234;
  %let x3=1e10;
  %let x4=hippopotamus;
  %let x5=zebra;
  %let k=3;
  %let seed=12345;
  %do j=1 %to 10;
    %syscall rancomb(seed, k, x1, x2, x3, x4, x5);
    %put j=&j   &x1 &x2 &x3;
  %end;
%mend;
%test;
```
SAS writes the following output to the log:

<table>
<thead>
<tr>
<th>j</th>
<th>1</th>
<th>-0.1234 hippocotamus zebra</th>
</tr>
</thead>
<tbody>
<tr>
<td>j</td>
<td>2</td>
<td>hippopotamus -0.1234 10000000000</td>
</tr>
<tr>
<td>j</td>
<td>3</td>
<td>hippopotamus ant zebra</td>
</tr>
<tr>
<td>j</td>
<td>4</td>
<td>-0.1234 zebra ant</td>
</tr>
<tr>
<td>j</td>
<td>5</td>
<td>-0.1234 ant hippopotamus</td>
</tr>
<tr>
<td>j</td>
<td>6</td>
<td>10000000000 hippopotamus ant</td>
</tr>
<tr>
<td>j</td>
<td>7</td>
<td>10000000000 hippopotamus ant</td>
</tr>
<tr>
<td>j</td>
<td>8</td>
<td>ant 10000000000 -0.1234</td>
</tr>
<tr>
<td>j</td>
<td>9</td>
<td>zebra -0.1234 10000000000</td>
</tr>
<tr>
<td>j</td>
<td>10</td>
<td>zebra hippopotamus 10000000000</td>
</tr>
</tbody>
</table>

See Also

Functions:

- “RAND Function” on page 555

CALL Routines:

- “CALL ALLPERM Routine” on page 733
- “CALL RANPERK Routine” on page 829
- “CALL RANPERM Routine” on page 831

CALL RANPERK Routine

Permutes the values of the arguments, and returns a random permutation of \( k \) out of \( n \) values.

<table>
<thead>
<tr>
<th>Category:</th>
<th>Combinatorial</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restriction:</td>
<td>This function is not valid on the CAS server.</td>
</tr>
<tr>
<td>Note:</td>
<td>Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.</td>
</tr>
</tbody>
</table>

Syntax

\[
\text{CALL RANPERK}(seed, k, variable-1 <, variable-2, \ldots>);
\]

Required Arguments

- **seed**
  
is a numeric variable that contains the random number seed. For more information about seeds, see “Seed Values” on page 11.

- **\( k \)**
  
is the number of values that you want to have in the random permutation.

- **variable**
  
  specifies all numeric variables, or all character variables that have the same length. \( K \) values of these variables are randomly permuted.
Details

**Using CALL RANPERK with Macros**

You can call the RANPERK routine when you use the %SYSCALL macro. In this case, the *variable* arguments are not required to be the same type or length. If %SYSCALL identifies an argument as numeric, then %SYSCALL reformats the returned value.

If an error occurs during the execution of the CALL RANPERK routine, then both of the following values are set:

- &SYSERR is assigned a value that is greater than 4.
- &SYSINFO is assigned a value that is less than -100.

If there are no errors, then &SYSERR and &SYSINFO are set to zero.

**Examples**

**Example 1: Using CALL RANPERK in a DATA Step**

The following example shows how to generate random permutations of given values by using the CALL RANPERK routine.

```sas
data _null_;  
array x x1-x5 (1 2 3 4 5);  
seed=1234567890123;  
do n=1 to 10;  
call ranperk(seed, 3, of x1-x5);  
put seed= &20 ' x= ' x1-x3;  
end;  
run;
```

SAS writes the following output to the log:

```
seed=1332351321     x= 5 4 2  
seed=829042065      x= 4 1 3  
seed=767738639      x= 5 1 2  
seed=1280236105     x= 3 2 5  
seed=670350431      x= 4 3 5  
seed=1956939964      x= 3 1 2  
seed=353939815       x= 4 2 1  
seed=1996660805     x= 3 4 5  
seed=1835940555      x= 5 1 4  
seed=910897519       x= 5 1 2
```

**Example 2: Using CALL RANPERK with a Macro**

Here is an example of the CALL RANPERK routine that is used with macros.

```sas
%macro test;  
%let x1=ant;  
%let x2=-.1234;  
%let x3=1e10;  
%let x4=hippopotamus;  
%let x5=zebra;  
%let k=3;  
%let seed=12345;  
%do j=1 %to 10;  
  %syscall ranperk(seed, k, x1, x2, x3, x4, x5);  
  %put j=&j   &x1 &x2 &x3;  
%end;  
%mend;
```
SAS writes the following output to the log:

| j=1  | -0.1234 hippopotamus zebra |
| j=2  | hippopotamus -0.1234 10000000000 |
| j=3  | hippopotamus ant zebra |
| j=4  | -0.1234 zebra ant |
| j=5  | -0.1234 ant hippopotamus |
| j=6  | 100000000000 hippopotamus ant |
| j=7  | 100000000000 hippopotamus ant |
| j=8  | ant 10000000000 -0.1234 |
| j=9  | zebra -0.1234 10000000000 |
| j=10 | zebra hippopotamus 10000000000 |

**See Also**

**Functions:**
- “RAND Function” on page 555

**Call Routines:**
- “CALL ALLPERM Routine” on page 733
- “CALL RANPERM Routine” on page 831

---

**CALL RANPERM Routine**

Randomly permutes the values of the arguments.

**Category:** Combinatorial

**Restriction:** This function is not valid on the CAS server.

**Note:** Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

**Syntax**

CALL RANPERM(seed, variable-1 <, variable-2, ...>);

**Required Arguments**

- **seed**
  - is a numeric variable that contains the random number seed. For more information about seeds, see “Seed Values” on page 11.

- **variable**
  - specifies all numeric variables or all character variables that have the same length. The values of these variables are randomly permuted.
Details

Using CALL RANPERM with Macros
You can call the RANPERM routine when you use the %SYSCALL macro. In this case, the variable arguments are not required to be the same type or length. If %SYSCALL identifies an argument as numeric, then %SYSCALL reformats the returned value.

If an error occurs during the execution of the CALL RANPERM routine, then both of the following values are set:

- &SYSERR is assigned a value that is greater than 4.
- &SYSINFO is assigned a value that is less than -100.

If there are no errors, then &SYSERR and &SYSINFO are set to zero.

Examples

Example 1: Using CALL RANPERM in a DATA Step
The following example generates random permutations of given values by using the CALL RANPERM routine.

data _null_;  
array x x1-x4 (1 2 3 4);  
seed=1234567890123;  
do n=1 to 10;  
call ranperm(seed, of x1-x4);  
put seed=@20 ' x= ' x1-x4;  
end;  
run;

SAS writes the following output to the log:

seed=1332351321     x= 1 3 2 4  
seed=829042065      x= 3 4 2 1  
seed=767738639      x= 4 2 3 1  
seed=1280236105     x= 1 2 4 3  
seed=670350431      x= 2 1 4 3  
seed=1956939964      x= 2 4 3 1  
seed=353939815       x= 4 1 2 3  
seed=1996660805      x= 4 3 1 2  
seed=1835940555      x= 4 3 2 1  
seed=910897519       x= 3 2 1 4  

Example 2: Using CALL RANPERM with a Macro
Here is an example of the CALL RANPERM routine that is used with the %SYSCALL macro.

%macro test;  
  %let x1=ant;  
  %let x2=-.1234;  
  %let x3=1e10;  
  %let x4=hippopotamus;  
  %let x5=zebra;  
  %let seed=12345;  
  %do j=1 %to 10;  
    %syscall ranperm(seed, x1, x2, x3, x4, x5);  
    %put j=&j   &x1 &x2 &x3;  
  %end;  
%mend;
SAS writes the following output to the log:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>j=1</td>
<td>zebra</td>
<td>ant</td>
<td>hippopotamus</td>
</tr>
<tr>
<td>j=2</td>
<td>10000000000</td>
<td>ant</td>
<td>-0.1234</td>
</tr>
<tr>
<td>j=3</td>
<td>-0.1234</td>
<td>10000000000</td>
<td>ant</td>
</tr>
<tr>
<td>j=4</td>
<td>hippopotamus</td>
<td>ant</td>
<td>zebra</td>
</tr>
<tr>
<td>j=5</td>
<td>-0.1234</td>
<td>zebra</td>
<td>10000000000</td>
</tr>
<tr>
<td>j=6</td>
<td>-0.1234</td>
<td>hippopotamus</td>
<td>ant</td>
</tr>
<tr>
<td>j=7</td>
<td>zebra</td>
<td>ant</td>
<td>-0.1234</td>
</tr>
<tr>
<td>j=8</td>
<td>-0.1234</td>
<td>hippopotamus</td>
<td>ant</td>
</tr>
<tr>
<td>j=9</td>
<td>ant</td>
<td>-0.1234</td>
<td>hippopotamus</td>
</tr>
<tr>
<td>j=10</td>
<td>-0.1234</td>
<td>zebra</td>
<td>10000000000</td>
</tr>
</tbody>
</table>

See Also

Functions:

- “RAND Function” on page 555

CALL Routines:

- “CALL ALLPERM Routine” on page 733
- “CALL RANPERK Routine” on page 829

CALL SET Routine

Links SAS data set variables to DATA step or macro variables that have the same name and data type.

**Category:** Variable Control

**Restriction:** This function is not valid on the CAS server.

**Note:** Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

**Syntax**

\[
\text{CALL SET}(\text{data-set-id});
\]

**Required Argument**

\textit{data-set-id}

is the identifier that is assigned by the OPEN function when the data set is opened.

**Details**

Using SET can significantly reduce the coding that is required for accessing variable values for modification or verification when you use functions to read or to manipulate a SAS file. After a CALL SET, whenever a read is performed from the SAS data set, the values of the corresponding macro or DATA step variables are set to the values of the
matching SAS data set variables. If the variable lengths do not match, the values are truncated or padded according to need. If you do not use SET, then you must use the GETVARC and GETVARN functions to move values explicitly between data set variables and macro or DATA step variables.

As a general rule, use CALL SET immediately following OPEN if you want to link the data set and the macro and DATA step variables.

**Example: Using the CALL SET Routine**

This example uses the CALL SET routine:

- The following statements automatically set the values of the macro variables PRICE and STYLE when an observation is fetched:

```sas
%macro setvar;
   %let dsid=%sysfunc(open(sasuser.houses, i));
   /* No leading ampersand with %SYSCALL */
   %syscall set(dsid);
   %let rc=%sysfunc(fetchobs(&dsid, 10));
   %let rc=%sysfunc(close(&dsid));
%mend setvar;
%global price style;
%setvar
%put _global_
```

- The %PUT statement writes these lines to the SAS log:

```
GLOBAL PRICE 127150
GLOBAL STYLE CONDO
```

- The following statements obtain the values for the first 10 observations in Sasuser.Houses and store them in MYDATA:

```sas
data mydata;
   /* create variables for assignment */
   /* by CALL SET */
   length style $8 sqfeet bedrooms baths 8
   street $16 price 8;
   drop rc dsid;
   dsid=open("sasuser.houses", "i");
   call set (dsid);
   do i=1 to 10;
      rc=fetchobs(dsid, i);
      output;
   end;
run;
```

**See Also**

**Functions:**

- “FETCH Function” on page 903
- “FETCHOBS Function” on page 905
- “GETVARC Function” on page 944
- “GETVARN Function” on page 945
**CALL SOFTMAX Routine**

Returns the softmax value.

**Category:** Mathematical

**Restriction:** This function is not valid on the CAS server.

**Note:** Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

---

### Syntax

CALL SOFTMAX(argument <,argument,…>);

### Required Argument

**argument**

is numeric.

**Restriction** The CALL SOFTMAX routine accepts only variables as valid arguments. Do not use a constant or a SAS expression because the CALL routine is unable to update these arguments.

---

### Details

The CALL SOFTMAX routine replaces each argument with the softmax value of that argument. For example, $x_j$ is replaced by

$$
\frac{e^{x_j}}{\sum_{i=1}^{n} e^{x_i}}
$$

If any argument contains a missing value, then CALL SOFTMAX returns missing values for all the arguments. Upon a successful return, the sum of all the values is equal to 1.

---

### Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=0.5;</td>
<td>x=0.3314989604</td>
</tr>
<tr>
<td>y=-0.5;</td>
<td>y=0.1219516523</td>
</tr>
<tr>
<td>z=1;</td>
<td>z=0.5465493873</td>
</tr>
<tr>
<td>call softmax(x, y, z);</td>
<td></td>
</tr>
<tr>
<td>put x= y= z=;</td>
<td></td>
</tr>
</tbody>
</table>

---
CALL SORTC Routine

Sorts the values of character arguments.

**Category:** Sort

**Restriction:** This function is not valid on the CAS server.

**Interaction:** When invoked by the %SYSCALL macro statement, CALL SORTC removes the quotation marks from its arguments. For more information, see "Using CALL Routines and the %SYSCALL Macro Statement" on page 9.

**Note:** Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

**Syntax**

```plaintext
CALL SORTC(variable-1 <, ..., variable-n>);
```

**Required Argument**

`variable`

specifies a character variable.

**Details**

The values of `variable` are sorted in ascending order by the CALL SORTC routine.

**Comparisons**

The CALL SORTC routine is used with character variables, and the CALL SORTN routine is used with numeric variables.

**Example**

The following example sorts the character variables in the array in ascending order.

```plaintext
data _null_;  
array x(8) $10  
     ('tweedledum' 'tweedledee' 'baboon' 'baby'  
     'humpty' 'dumpty' 'banana' 'babylon');  
call sortc(of x(*));  
put +3 x(*);  
run;  
```

SAS writes the following output to the log:

```
baboon baby babylon banana dumpty humpty tweedledee tweedledum
```

**See Also**

**CALL Routines:**

- “CALL SORTN Routine” on page 837
CALL SORTN Routine

Sorts the values of numeric arguments.

**Category:** Sort

**Restriction:** This function is not valid on the CAS server.

**Note:** Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

**Syntax**

CALL SORTN(variable-1 <, ..., variable-n>);

**Required Argument**

`variable`

specifies a numeric variable.

**Details**

The values of `variable` are sorted in ascending order by the CALL SORTN routine.

**Comparisons**

The CALL SORTN routine is used with numeric variables, and the CALL SORTC routine is used with character variables.

**Example**

The following example sorts the numeric variables in the array in ascending order.

```sas
data _null_
array x(10) (0, ., .a, 1e-12, -1e-8, .z, -37, 123456789, 1e20, 42)
call sortn(of x(*));
put +3 x(*);
run;
```

SAS writes the following output to the log:

```
. . A Z -37 -1E-8 0 1E-12 42 123456789 1E20
```

**See Also**

CALL Routines:

- “CALL SORTC Routine” on page 836
CALL STDIZE Routine

Standardizes the values of one or more variables.

Category: Mathematical
Restriction: This function is not valid on the CAS server.
Interaction: When invoked by the %SYSCALL macro statement, CALL STDIZE removes the quotation marks from its arguments. For more information, see “Using CALL Routines and the %SYSCALL Macro Statement” on page 9.

Note: Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

Syntax

CALL STDIZE(<option-1, option-2, ...,> variable-1 <,variable-2, ...>);

Required Argument

variable

is numeric. These values will be standardized according to the method that you use.

Optional Arguments

option

specifies a character expression whose values can be uppercase, lowercase, or mixed case letters. Leading and trailing blanks are ignored. option includes the following three categories:

- standardization-options
- VARDEF-options
- miscellaneous-options

Restriction Use a separate argument for each option because you cannot specify more than one option in a single argument.

Tip Character expressions can end with an equal sign that is followed by another argument that is a numeric constant, variable, or expression.

See PROC STDIZE in SAS/STAT 9.3 User’s Guide for information about formulas and other details. The options that are used in CALL STDIZE are the same as those used in PROC STDIZE.

standardization-options

specify how to compute the location and scale measures that are used to standardize the variables. The following standardization options are available:

ABW= must be followed by an argument that is a numeric expression specifying the tuning constant.
AGK= must be followed by an argument that is a numeric expression that specifies the proportion of pairs to be included in the estimation of the within-cluster variances.

AHUBER= must be followed by an argument that is a numeric expression specifying the tuning constant.

AWAVE= must be followed by an argument that is a numeric expression specifying the tuning constant.

EUCLEN specifies the Euclidean length.

IQR specifies the interquartile range.

L= must be followed by an argument that is a numeric expression with a value greater than or equal to 1 specifying the power to which differences are to be raised in computing an $L(p)$ or Minkowski metric.

MAD specifies the median absolute deviation from the median.

MAXABS specifies the maximum absolute values.

MEAN specifies the arithmetic mean (average).

MEDIAN specifies the middle number in a set of data that is ordered according to rank.

MIDRANGE specifies the midpoint of the range.

RANGE specifies a range of values.

SPACING= must be followed by an argument that is a numeric expression that specifies the proportion of data to be contained in the spacing.

STD specifies the standard deviation.

SUM specifies the result that you obtain when you add numbers.

USTD specifies the standard deviation about the origin, based on the uncorrected sum of squares.

**VARDEF-options**

specify the divisor to be used in the calculation of variances. VARDEF options can have the following values:

- **DF** specifies degrees of freedom.
- **N** specifies the number of observations. The default is DF.

**miscellaneous-options**

Miscellaneous options can have the following values:

- **ADD=** is followed by a numeric argument that specifies a number to add to each value after standardizing and multiplying by the value from the MULT= option. The default value is 0.
- **FUZZ=** is followed by a numeric argument that specifies the relative fuzz factor.
- **MISSING=** is followed by a numeric argument that specifies a value to be assigned to variables that have a missing value.
MULT= is followed by a numeric argument that specifies a number by which to multiply each value after standardizing. The default value is 1.

NORM normalizes the scale estimator to be consistent for the standard deviation of a normal distribution. This option affects only the methods AGK=, IQR, MAD, and SPACING=.

PSTAT writes the values of the location and scale measures in the log.

REPLACE replaces missing values with the value 0 in the standardized data (this value corresponds to the location measure before standardizing). To replace missing values by other values, see the MISSING= option.

SNORM normalizes the scale estimator to have an expectation of approximately 1 for a standard normal distribution. This option affects only the SPACING= method.

Details

The CALL STDIZE routine transforms one or more arguments that are numeric variables by subtracting a location measure and dividing by a scale measure. You can use a variety of location and scale measures. The default location option is MEAN, and the default scale option is STD.

In addition, you can multiply each standardized value by a constant, and you can add a constant. The final output value would be

\[ \text{result} = \text{add} + \text{mult} \times \left( \frac{\text{original} - \text{location}}{\text{scale}} \right). \]

These are the descriptions of the variables:

\text{result} specifies the final value that is returned for each variable.

\text{add} specifies the constant to add (ADD= option).

\text{mult} specifies the constant to multiply by (MULT= option).

\text{original} specifies the original input value.

\text{location} specifies the location measure.

\text{scale} specifies the scale measure.

You can replace missing values by any constant. If you do not specify the MISSING= or the REPLACE option, variables that have missing values are not altered. The initial estimation method for the ABW=, AHUBER=, and AWA= methods is MAD. Percentiles are computed using definition 5.

Comparisons

The CALL STDIZE routine is similar to the STDIZE procedure in the SAS/STAT product. However, the CALL STDIZE routine is primarily useful for standardizing the rows of a SAS data set, whereas the STDIZE procedure can standardize only the
columns of a SAS data set. For more information, see PROC STDIZE in SAS/STAT User's Guide.

Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>retain x 1 y 2 z 3;</td>
<td></td>
</tr>
<tr>
<td>call stdize(x, y ,z);</td>
<td>x=-1 y=0 z=1</td>
</tr>
<tr>
<td>put x= y= z=;</td>
<td></td>
</tr>
<tr>
<td>retain w 10 x 11 y 12 z 13;</td>
<td></td>
</tr>
<tr>
<td>call stdize('iqr', w, x, y, z);</td>
<td>w=-0.75 x=-0.25 y=0.25 z=0.75</td>
</tr>
<tr>
<td>put w= x= y= z=;</td>
<td></td>
</tr>
<tr>
<td>retain w . x 1 y 2 z 3;</td>
<td></td>
</tr>
<tr>
<td>call stdize('range', w, x, y, z);</td>
<td>w=. x=0.5 y= z=1</td>
</tr>
<tr>
<td>put w= x= y= z=;</td>
<td></td>
</tr>
<tr>
<td>retain w . x 1 y 2 z 3;</td>
<td></td>
</tr>
<tr>
<td>call stdize('mult=',10,'missing=', -1,'range', w, x, y, z);</td>
<td>w=-1 x=0 y=5 z=10</td>
</tr>
<tr>
<td>put w= x= y= z=;</td>
<td></td>
</tr>
</tbody>
</table>

CALL SYMPUT Routine

Assigns DATA step information to a macro variable.

**Category:** Macro  
**Restriction:** This function is not valid on the CAS server.  
**Note:** Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

**Syntax**

CALL SYMPUT(argument-1, argument-2);

**Required Arguments**

- **argument-1**
  - specifies a character expression that identifies the macro variable that is assigned a value. If the macro variable does not exist, the routine creates it.

- **argument-2**
  - specifies a character constant, variable, or expression that contains the value that is assigned.
Details

The CALL SYMPUT routine either creates a macro variable whose value is information from the DATA step or assigns a DATA step value to an existing macro variable. CALL SYMPUT is fully documented in “SYMPUT Routine” in SAS Viya Macro Language: Reference.

See Also

Functions:
• “SYMGET Function” on page 1070

CALL SYMPUTX Routine

Assigns a value to a macro variable, and removes both leading and trailing blanks.

Category: Macro
Restriction: This function is not valid on the CAS server.
Note: Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

Syntax

CALL SYMPUTX(macro-variable, value <, symbol-table>);

Required Arguments

macro-variable

can be one of the following:
• a character string that is a SAS name, enclosed in quotation marks.
• the name of a character variable whose values are SAS names.
• a character expression that produces a macro variable name. This form is useful for creating a series of macro variables.

a character constant, variable, or expression. Leading and trailing blanks are removed from the value of name, and the result is then used as the name of the macro variable.

value

specifies a character or numeric constant, variable, or expression. If value is numeric, SAS converts the value to a character string using the BEST. format and does not issue a note to the SAS log. Leading and trailing blanks are removed, and the resulting character string is assigned to the macro variable.

Optional Argument

symbol-table

specifies a character constant, variable, or expression. The value of symbol-table is not case sensitive. The first non-blank character in symbol-table specifies the symbol
table in which to store the macro variable. The following values are valid as the first non-blank character in \textit{symbol-table}:

- \textbf{G} specifies that the macro variable is stored in the global symbol table, even if the local symbol table exists.
- \textbf{L} specifies that the macro variable is stored in the most local symbol table that exists, which will be the global symbol table, if used outside a macro.
- \textbf{F} specifies that if the macro variable exists in any symbol table, CALL SYMPUTX uses the version in the most local symbol table in which it exists. If the macro variable does not exist, CALL SYMPUTX stores the variable in the most local symbol table.

\textit{Note}: If you omit \textit{symbol-table} or if \textit{symbol-table} is blank, CALL SYMPUTX stores the macro variable in the same symbol table as does the CALL SYMPUT routine.

**Details**

CALL SYMPUTX is similar to CALL SYMPUT except that

- CALL SYMPUTX does not write a note to the SAS log when the second argument is numeric. CALL SYMPUT, however, writes a note to the log stating that numeric values were converted to character values.
- CALL SYMPUTX uses a field width of up to 32 characters when it converts a numeric second argument to a character value. CALL SYMPUT uses a field width of up to 12 characters.
- CALL SYMPUTX left-justifies both arguments and trims trailing blanks. CALL SYMPUT does not left-justify the arguments, and trims trailing blanks from the first argument only. Leading blanks in the value of \textit{name} cause an error.
- CALL SYMPUTX enables you to specify the symbol table in which to store the macro variable, whereas CALL SYMPUT does not.

**Example: Using CALL SYMPUTX**

The following example shows the results of using CALL SYMPUTX.

```sas
data _null_; 
  call symputx('  items  ', '   leading and trailing blanks removed   ', 'lplace');
  call symputx('  x   ', 123.456); 
run;
%put items=!&items!;
%put x=!&x!;
```

SAS writes the following output to the log:

```
items='leading and trailing blanks removed'
x=123.456!
```

**See Also**

**Functions:**

- “SYMGET Function” on page 1070
CALL Routines:

• “CALL SYMPUT Routine” on page 841

CALL SYSTEM Routine

Submits an operating environment command for execution.

Category: Special

Restriction: This function is not valid on the CAS server.

Interaction: When invoked by the %SYSCALL macro statement, CALL SYSTEM removes quotation marks from its arguments. For more information, see “Using CALL Routines and the %SYSCALL Macro Statement” on page 9.

UNIX specifics: command must evaluate to a valid UNIX command

Note: Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

Syntax

CALL SYSTEM(command);

Required Argument

command

specifies any of the following: a system command that is enclosed in quotation marks (character string), an expression whose value is a system command, or the name of a character variable whose value is a system command that is executed.

Restriction The length of the command cannot be greater than 1024 characters, including trailing blanks.

Details

General Information

The behavior of the CALL SYSTEM routine is similar to that of the X command, the X statement, and the SYSTEM function. It is useful in certain situations because it can be conditionally executed, it accepts an expression as an argument, and it is executed at run time.

UNIX Specific Information

Under UNIX, the output of the command appears in the window from which you invoked SAS.

The value of the XSYNC system option affects how the CALL SYSTEM routine works.

Note: The CALL SYSTEM routine can be executed within a DATA step. However, neither the X statement nor the %SYSEXEC macro program statement is intended for use during the execution of a DATA step.
Example: Using the CALL SYSTEM Routine to Send a Message under UNIX

In the following example, for each record in answer.week, if the resp variable is y, the CALL SYSTEM routine mails a message:

```sas
data _null_;  
set answer.week;  
if resp='y' then do;  
    call system('mail mgr < $HOME/msg');  
end;  
run;
```

CALL TANH Routine

Returns the hyperbolic tangent.

**Category:** Mathematical

**Restriction:** This function is not valid on the CAS server.

**Note:** Argument types for arguments that are updated must match in CALL routines. All argument types must be CHAR, VARCHAR, or NUMERIC. If the argument types do not match, a warning is issued to the log.

### Syntax

CALL TANH(argument <, argument, ...>);

### Required Argument

**argument**

- is numeric.

**Restriction** The CALL TANH routine accepts only variables as valid arguments. Do not use a constant or a SAS expression, because the CALL routine is unable to update these arguments.

### Details

The subroutine TANH replaces each argument by the tanh of that argument. For example, \(x_j \) is replaced by

\[
tanh(x_j) = \frac{e^{x_j} - e^{-x_j}}{e^{x_j} + e^{-x_j}}
\]

If any argument contains a missing value, then CALL TANH returns missing values for all the arguments.

### Example

The following SAS statements produce these results:
**SAS Statement**

\[
\begin{align*}
x &= 0.5; \\
y &= -0.5; \\
call tanh(x, y); \\
put x = y =;
\end{align*}
\]

**Result**

\[
\begin{align*}
x &= 0.4621171573 \\
y &= -0.462117157
\end{align*}
\]

---

**See Also**

**Functions:**

- “TANH Function” on page 1079

---

**CATQ Function**

Concatenates character or numeric values by using a delimiter to separate items and by adding quotation marks to strings that contain the delimiter.

**Category:** Character

**Restrictions:**

This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility.

This function is not valid on the CAS server.

---

**Syntax**

\[
\text{CATQ}(\text{modifiers} <, \text{delimiter}>, \text{item-1} <, \ldots, \text{item-n}>)
\]

**Required Arguments**

**modifier**

specifies a character constant, variable, or expression in which each non-blank character modifies the action of the CATQ function. Blanks are ignored. You can use the following characters as modifiers:

- 1 or ' uses single quotation marks when CATQ adds quotation marks to a string.
- 2 or " uses double quotation marks when CATQ adds quotation marks to a string.
- a or A adds quotation marks to all of the item arguments.
- b or B adds quotation marks to item arguments that have leading or trailing blanks that are not removed by the S or T modifiers.
- c or C uses a comma as a delimiter.
- d or D indicates that you have specified the delimiter argument.
h or H
uses a horizontal tab as the delimiter.

m or M
inserts a delimiter for every item argument after the first. If you do not use the M modifier, then CATQ does not insert delimiters for item arguments that have a length of zero after processing that is specified by other modifiers. The M modifier can cause delimiters to appear at the beginning or end of the result and can cause multiple consecutive delimiters to appear in the result.

n or N
converts item arguments to name literals when the value does not conform to the usual syntactic conventions for a SAS name. A name literal is a string in quotation marks that is followed by the letter “n” without any intervening blanks. To use name literals in SAS statements, you must specify the SAS option, VALIDVARNAME=ANY.

q or Q
adds quotation marks to item arguments that already contain quotation marks.

s or S
strips leading and trailing blanks from subsequently processed arguments:
• To strip leading and trailing blanks from the delimiter argument, specify the S modifier before the D modifier.
• To strip leading and trailing blanks from the item arguments but not from the delimiter argument, specify the S modifier after the D modifier.

t or T
trims trailing blanks from subsequently processed arguments:
• To trim trailing blanks from the delimiter argument, specify the T modifier before the D modifier.
• To trim trailing blanks from the item arguments but not from the delimiter argument, specify the T modifier after the D modifier.

x or X
converts item arguments to hexadecimal literals when the value contains nonprintable characters.

Tips If modifier is a constant, enclose it in quotation marks. You can also express modifier as a variable name or an expression.

The A, B, N, Q, S, T, and X modifiers operate internally to the CATQ function. If an item argument is a variable, then the value of that variable is not changed by CATQ unless the result is assigned to that variable.

item specifies a constant, variable, or expression, either character or numeric. If item is numeric, then its value is converted to a character string by using the BESTw. format. In this case, leading blanks are removed and SAS does not write a note to the log.

Optional Argument
delimiter specifies a character constant, variable, or expression that is used as a delimiter between concatenated strings. If you specify this argument, then you must also specify the D modifier.
Details

Length of Returned Variable
The CATQ function returns a value to a variable or if CATQ is called inside an expression, CATQ returns a value to a temporary buffer. The value that is returned has the following length:

- up to 200 characters in WHERE clauses and in PROC SQL
- up to 32767 characters in the DATA step except in WHERE clauses
- up to 65534 characters when CATQ is called from the macro processor

If the length of the variable or the buffer is not large enough to contain the result of the concatenation, then SAS does the following steps:

- changes the result to a blank value in the DATA step and in PROC SQL
- writes a warning message to the log stating that the result was either truncated or set to a blank value, depending on the calling environment
- writes a note to the log that shows the location of the function call and lists the argument that caused the truncation
- sets _ERROR_ to 1 in the DATA step

If CATQ returns a value in a temporary buffer, then the length of the buffer depends on the calling environment, and the value in the buffer can be truncated after CATQ finishes processing. In this case, SAS does not write a message about the truncation to the log.

The Basics
If you do not use the C, D, or H modifiers, then CATQ uses a blank as a delimiter.

If you specify neither a quotation mark in modifier nor the 1 or 2 modifiers, then CATQ decides independently for each item argument which type of quotation mark to use, if quotation marks are required. The following rules apply:

- CATQ uses single quotation marks for strings that contain an ampersand (&) or percent (%) sign, or that contain more double quotation marks than single quotation marks.
- CATQ uses double quotation marks for all other strings.

The CATQ function initializes the result to a length of zero and then performs the following actions for each item argument:

1. If item is not a character string, then CATQ converts item to a character string by using the BESTw. format and removes leading blanks.
2. If you used the S modifier, then CATQ removes leading blanks from the string.
3. If you used the S or T modifiers, then CATQ removes trailing blanks from the string.
4. CATQ determines whether to add quotation marks based on the following conditions:
   - If you use the X modifier and the string contains control characters, then the string is converted to a hexadecimal literal.
   - If you use the N modifier, then the string is converted to a name literal if either of the following conditions is true:
     - The first character in the string is not an underscore or an English letter.
The string contains any character that is not a digit, underscore, or English letter.

If you did not use the X or the N modifiers, then CATQ adds quotation marks to the string if any of the following conditions is true:

- You used the A modifier.
- You used the B modifier and the string contains leading or trailing blanks that were not removed by the S or T modifiers.
- You used the Q modifier and the string contains quotation marks.
- The string contains a substring that equals the delimiter with leading and trailing blanks omitted.

5. For the second and subsequent item arguments, CATQ appends the delimiter to the result if either of the following conditions is true:

- You used the M modifier.
- The string has a length greater than zero after it has been processed by the preceding steps.

6. CATQ appends the string to the result.

**Comparisons**

The CATX function is similar to the CATQ function except that CATX does not add quotation marks.

**Example: Concatenating Strings with the CATQ Function**

The following example shows how the CATQ function concatenates strings.

```sas
options ls=110;
data _null_;  
result1=CATQ(' ', 'noblank', 'one blank', 12345, ' lots of blanks ');  
result2=CATQ('CS', 'Period (.) ', 'Ampersand (&)', 'Comma (,)', 'Double quotation marks (") ', 'Leading Blanks');  
result3=CATQ('BCQT', 'Period (.) ', 'Ampersand (&)', 'Comma (,)', 'Double quotation marks (") ', 'Leading Blanks');  
result4=CATQ('ADT', '#=#', 'Period (.) ', 'Ampersand (&)', 'Comma (,)', 'Double quotation marks (") ');  
```

### CATQ Function

849
' Leading Blanks');
result5=CATQ('N',
    'ABC_123 ',
    '123 ',
    'ABC 123');
    put (result1-result5) (=/);
run;

SAS writes the following output to the log:

result1=noblanks "one blank" 12345 " lots of blanks  
result2=Period (.),Ampersand (&),"Comma (,)",Double quotation marks ("),Leading
    Blanks
result3=Period (.),Ampersand (&),"Comma (,)",'Double quotation marks ("),"Leading Blanks"
result4="Period (.)"##'Ampersand (&)"##'Comma (,)"##'Double quotation marks
    (")'##' Leading Blanks"
result5=ABC_123 "123"n "ABC 123"n

See Also

Functions:
- “CAT Function” on page 143
- “CATS Function” on page 145
- “CATT Function” on page 147
- “CATX Function” on page 149

CALL Routines:
- “CALL CATS Routine” on page 736
- “CALL CATT Routine” on page 738
- “CALL CATX Routine” on page 740

CEXIST Function

Verifies the existence of a SAS catalog or SAS catalog entry.

Category: SAS File I/O

Restriction: This function is not valid on the CAS server.

Syntax

CEXIST(entry<,'U'>)

Required Argument

entry

is a character constant, variable, or expression that specifies a SAS catalog, or the
name of an entry in a catalog. If the entry value is a one- or two-level name, then it is
assumed to be the name of a catalog. Use a three- or four-level name to test for the
existence of an entry within a catalog.
**Optional Argument**

'*U'*

tests whether the catalog can be opened for updating.

**Details**

CEXIST returns 1 if the SAS catalog or catalog entry exists, or 0 if the SAS catalog or catalog entry does not exist.

**Examples**

**Example 1: Verifying the Existence of an Entry in a Catalog**

This example verifies the existence of the entry X.PROGRAM in LIB.CAT1:

```sas
data _null_
  if cexist("lib.cat1.x.program") then
    put "Entry X.PROGRAM exists";
run;
```

**Example 2: Determining If a Catalog Can Be Opened for Update**

This example tests whether the catalog LIB.CAT1 exists and can be opened for update. If the catalog does not exist, a message is written to the SAS log. Note that in a macro statement that you do not enclose character strings in quotation marks.

```sas
%if %sysfunc(cexist(lib.cat1, u)) %then
  %put The catalog LIB.CAT1 exists and can be opened for update.;
%else
  %put %sysfunc(sysmsg());
```

**See Also**

**Functions:**

- “EXIST Function” on page 892

---

**CHAR Function**

Returns a single character from a specified position in a character string.

**Category:** Character

**Restrictions:**

This function is not valid on the CAS server. This function is assigned an I18N Level 0 status, and is designed for SBCS data. Do not use this function to process DBCS or MBCS data. For more information, see Internationalization Compatibility.

**Syntax**

`CHAR(string, position)`
Required Arguments

string
   specifies a character constant, variable, or expression.

position
   is an integer that specifies the position of the character to be returned.

Details

In a DATA step, the default length of the target variable for the CHAR function is 1.
If position has a missing value, then CHAR returns a string with a length of 0.
Otherwise, CHAR returns a string with a length of 1.
If position is less than or equal to 0, or greater than the length of the string, then CHAR returns a blank. Otherwise, CHAR returns the character at the specified position in the string.

Comparisons

The CHAR function returns the same result as SUBPAD(string, position, 1). The results are the same, but the default length of the target variable is different.

Example

The following example shows the results of using the CHAR function.

```sas
data test;
   retain string "abc";
   do position = -1 to 4;
      result=char(string, position);
      output;
   end;
run;

proc print noobs data=test;
run;
```
CINV Function

Returns a quantile from the chi-square distribution.

**Category:** Quantile

**Restriction:** This function is not valid on the CAS server.

### Syntax

\[
\text{CINV}(p, df) \quad \text{or} \quad \text{CINV}(p, df^<, nc>)
\]

### Required Arguments

**\( p \)**

is a numeric probability.

Range \( 0 \leq p < 1 \)

**\( df \)**

is a numeric degrees of freedom parameter.

Range \( df > 0 \)
Optional Argument

\( nc \)

is a numeric noncentrality parameter.

Range \( nc \geq 0 \)

Details

The CINV function returns the \( p \)th quantile from the chi-square distribution with degrees of freedom \( df \) and a noncentrality parameter \( nc \). The probability that an observation from a chi-square distribution is less than or equal to the returned quantile is \( p \). This function accepts a noninteger degrees of freedom parameter \( df \).

If the optional parameter \( nc \) is not specified or has the value 0, the quantile from the central chi-square distribution is returned. The noncentrality parameter \( nc \) is defined such that if \( X \) is a normal random variable with mean \( \mu \) and variance 1, \( X^2 \) has a noncentral chi-square distribution with \( df=1 \) and \( nc = \mu^2 \).

CAUTION:

For large values of \( nc \), the algorithm could fail. In that case, a missing value is returned.

Note: CINV is the inverse of the PROBCHI function.

Example

The first statement following shows how to find the 95th percentile from a central chi-square distribution with 3 degrees of freedom. The second statement shows how to find the 95th percentile from a noncentral chi-square distribution with 3.5 degrees of freedom and a noncentrality parameter equal to 4.5.

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>q1=cinv(.95, 3);</td>
<td>7.8147279033</td>
</tr>
<tr>
<td>a2=cinv(.95, 3.5, 4.5);</td>
<td>7.504582117</td>
</tr>
</tbody>
</table>

See Also

Functions:

- “QUANTILE Function” on page 549

CLOSE Function

Closes a SAS data set.

Category: SAS File I/O

Restriction: This function is not valid on the CAS server.
Syntax

CLOSE(data-set-id)

Required Argument

data-set-id

is a numeric variable that specifies the data set identifier that the OPEN function returns.

Details

CLOSE returns zero if the operation was successful, or returns a nonzero value if it was not successful. Close all SAS data sets as soon as they are no longer needed by the application.

Note: All data sets opened within a DATA step are closed automatically at the end of the DATA step.

Example

This example uses OPEN to open the SAS data set PAYROLL. If the data set opens successfully, indicated by a positive value for the variable PAYID, the example uses CLOSE to close the data set.

%let payid=%sysfunc(open(payroll, is));
macro statements
%if &payid > 0 %then
  %let rc=%sysfunc(close(&payid));

See Also

Functions:

• “OPEN Function” on page 1010

CMISS Function

Counts the number of missing arguments.

Category: Descriptive Statistics
Restriction: This function is not valid on the CAS server.

Syntax

CMISS(argument-1 <, argument-2,…>)

Required Argument

argument

specifies a constant, variable, or expression. Argument can be either a character value or a numeric value.
Details

A character expression is counted as missing if it evaluates to a string that contains all blanks or has a length of zero, except when you use the CMISS function in macro processing. A numeric expression is counted as missing if it evaluates to a numeric missing value: ., ._, .A, … , .Z.

When you use the CMISS function in macro processing, use a period (.) to represent both a character and a numeric missing value. If you use a blank or null value for a character argument, SAS returns an error. Here are three examples that result in an error:

%let macvar=%sysfunc(cmiss(A,%str( )));
%let macvar=%sysfunc(cmiss(A, ));
%let macvar=%sysfunc(cmiss(A,));

Here is the example to use to avoid the error condition:

%let macvar=%sysfunc(cmiss(A,.));

Comparisons

The CMISS function does not convert any argument. The NMISS function converts all arguments to numeric values.

See Also

Functions:
- “MISSING Function” on page 436
- “NMISS Function” on page 446

COMPGED Function

Returns the generalized edit distance between two strings.

Category: Character

Restrictions: This function is assigned an I18N Level 0 status, and is designed for SBCS data. Do not use this function to process DBCS or MBCS data. For more information, see Internationalization Compatibility.

This function is not valid on the CAS server.

Syntax

COMPGED(string-1, string-2 <, cutoff > <, modifier(s)>)

Required Arguments

string-1
- specifies a character constant, variable, or expression.

string-2
- specifies a character constant, variable, or expression.
Optional Arguments

cutoff
is a numeric constant, variable, or expression. If the actual generalized edit distance is greater than the value of cutoff, the value that is returned is equal to the value of cutoff.

TIP Using a small value of cutoff improves the efficiency of COMPGED if the values of string-1 and string-2 are long.

modifier
specifies a character string that can modify the action of the COMPGED function. You can use one or more of the following characters as a valid modifier:

i or I ignores the case in string-1 and string-2.

l or L removes leading blanks in string-1 and string-2 before comparing the values.

n or N removes quotation marks from any argument that is an n-literal and ignores the case of string-1 and string-2.

: (colon) truncates the longer of string-1 or string-2 to the length of the shorter string, or to one, whichever is greater.

TIP COMPGED ignores blanks that are used as modifiers.

Details

The Order in Which Modifiers Appear
The order in which the modifiers appear in the COMPGED function is relevant.

- "LN" first removes leading blanks from each string and then removes quotation marks from n-literals.
- "NL" first removes quotation marks from n-literals and then removes leading blanks from each string.

Definition of Generalized Edit Distance
Generalized edit distance is a generalization of Levenshtein edit distance, which is a measure of dissimilarity between two strings. The Levenshtein edit distance is the number of deletions, insertions, or replacements of single characters that are required to transform string-1 into string-2.

Computing the Generalized Edit Distance
The COMPGED function returns the generalized edit distance between string-1 and string-2. The generalized edit distance is the minimum-cost sequence of operations for constructing string-1 from string-2.

The algorithm for computing the sum of the costs involves a pointer that points to a character in string-2 (the input string). An output string is constructed by a sequence of operations that might advance the pointer, add one or more characters to the output string, or both. Initially, the pointer points to the first character in the input string, and the output string is empty.

The operations and their costs are described in the following table.
<table>
<thead>
<tr>
<th>Operation</th>
<th>Default Cost in Units</th>
<th>Description of Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPEND</td>
<td>50</td>
<td>When the output string is longer than the input string, add any one character to the end of the output string without moving the pointer.</td>
</tr>
</tbody>
</table>
| BLANK     | 10                    | Do any of the following:  
- Add one space character to the end of the output string without moving the pointer.  
- When the character at the pointer is a space character, advance the pointer by one position without changing the output string.  
- When the character at the pointer is a space character, add one space character to the end of the output string, and advance the pointer by one position. If the cost for BLANK is set to zero by the COMPCOST function, the COMPGED function removes all space characters from both strings before doing the comparison. |
<p>| DELETE    | 100                   | Advance the pointer by one position without changing the output string. |
| DOUBLE    | 20                    | Add the character at the pointer to the end of the output string without moving the pointer. |
| FDELETE   | 200                   | When the output string is empty, advance the pointer by one position without changing the output string. |
| FINSERT   | 200                   | When the pointer is in position one, add any one character to the end of the output string without moving the pointer. |
| FREPLACE  | 200                   | When the pointer is in position one and the output string is empty, add any one character to the end of the output string, and advance the pointer by one position. |
| INSERT    | 100                   | Add any one character to the end of the output string without moving the pointer. |</p>
<table>
<thead>
<tr>
<th>Operation</th>
<th>Default Cost in Units</th>
<th>Description of Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>MATCH</td>
<td>0</td>
<td>Copy the character at the pointer from the input string to the end of the output string, and advance the pointer by one position.</td>
</tr>
<tr>
<td>PUNCTUATION</td>
<td>30</td>
<td>Do any of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Add one punctuation character to the end of the output string without moving the pointer.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• When the character at the pointer is a punctuation character, advance the pointer by one position without changing the output string.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• When the character at the pointer is a punctuation character, add one punctuation character to the end of the output string, and advance the pointer by one position.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If the cost for PUNCTUATION is set to zero by the COMPCOST function, the COMPGED function removes all punctuation characters from both strings before doing the comparison.</td>
</tr>
<tr>
<td>REPLACE</td>
<td>100</td>
<td>Add any one character to the end of the output string, and advance the pointer by one position.</td>
</tr>
<tr>
<td>SINGLE</td>
<td>20</td>
<td>When the character at the pointer is the same as the character that follows in the input string, advance the pointer by one position without changing the output string.</td>
</tr>
<tr>
<td>SWAP</td>
<td>20</td>
<td>Copy the character that follows the pointer from the input string to the output string. Then copy the character at the pointer from the input string to the output string. Advance the pointer two positions.</td>
</tr>
<tr>
<td>TRUNCATE</td>
<td>10</td>
<td>When the output string is shorter than the input string, advance the pointer by one position without changing the output string.</td>
</tr>
</tbody>
</table>

To set the cost of the string operations, you can use the CALL COMPCOST routine or use default costs. If you use the default costs, the values that are returned by COMPGED are approximately 100 times greater than the values that are returned by COMPLEV.
Examples of Errors
The rationale for determining the generalized edit distance is based on the number and types of typographical errors that can occur. COMPGED assigns a cost to each error and determines the minimum sum of these costs that could be incurred. Some types of errors can be more serious than others. For example, inserting an extra letter at the beginning of a string might be more serious than omitting a letter from the end of a string. For another example, if you enter a word or phrase that exists in \textit{string-2} and introduce a typographical error, you might produce \textit{string-1} instead of \textit{string-2}.

Making the Generalized Edit Distance Symmetric
Generalized edit distance is not necessarily symmetric. That is, the value that is returned by \texttt{COMPGED(string1, string2)} is not always equal to the value that is returned by \texttt{COMPGED(string2, string1)}. To make the generalized edit distance symmetric, use the \texttt{CALL COMPCOST} routine to assign equal costs to the operations within each of the following pairs:

- INSERT, DELETE
- FINSERT, FDELETE
- APPEND, TRUNCATE
- DOUBLE, SINGLE

Comparisons
You can compute the Levenshtein edit distance by using the \texttt{COMPLEV} function. You can compute the generalized edit distance by using the \texttt{CALL COMPCOST} routine and the \texttt{COMPGED} function. Computing generalized edit distance requires considerably more computer time than does computing Levenshtein edit distance. But generalized edit distance usually provides a more useful measure than Levenshtein edit distance for applications such as fuzzy file merging and text mining.

Example
The following example uses the default costs to calculate the generalized edit distance.

```sas
data test;
  infile datalines missover;
  input String1 $char8. +1 String2 $char8. +1 Operation $40.;
  Ged=compged(string1, string2);
  datalines;
baboon baboon match
baXboon baboon insert
baoon baboon delete
baKoon baboon replace
baboonX baboon append
baboo baboon truncate
babboon baboon double
babon baboon single
baobon baboon swap
bab oon baboon blank
bab,oon baboon punctuation
bXaoon baboon insert+delete
bXaYoon baboon insert+replace
bXoOn baboon delete+replace
```

Xbaboon baboon  finsert
aboon baboon  trick question: swap+delete
Xaboon baboon  freplace
axoon baboon  fdelete+replace
axoo baboon  fdelete+replace+truncate
axon baboon  fdelete+replace+single
baby baboon  replace+truncate*2
balloon baboon  replace+insert

;  

proc print data=test label;
  label GED='Generalized Edit Distance';
  var String1 String2 GED Operation;
run;

*Figure 3.21  Generalized Edit Distance Based on Operation*

<table>
<thead>
<tr>
<th>Obs</th>
<th>String1</th>
<th>String2</th>
<th>Generalized Edit Distance</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>baboon</td>
<td>baboon</td>
<td>0</td>
<td>match</td>
</tr>
<tr>
<td>2</td>
<td>babXboon</td>
<td>baboon</td>
<td>100</td>
<td>insert</td>
</tr>
<tr>
<td>3</td>
<td>baboon</td>
<td>baboon</td>
<td>100</td>
<td>delete</td>
</tr>
<tr>
<td>4</td>
<td>babXoon</td>
<td>baboon</td>
<td>100</td>
<td>replace</td>
</tr>
<tr>
<td>5</td>
<td>baboonX</td>
<td>baboon</td>
<td>50</td>
<td>append</td>
</tr>
<tr>
<td>6</td>
<td>baboo</td>
<td>baboon</td>
<td>10</td>
<td>truncate</td>
</tr>
<tr>
<td>7</td>
<td>baboon</td>
<td>baboon</td>
<td>20</td>
<td>double</td>
</tr>
<tr>
<td>8</td>
<td>babon</td>
<td>baboon</td>
<td>20</td>
<td>single</td>
</tr>
<tr>
<td>9</td>
<td>baboon</td>
<td>baboon</td>
<td>20</td>
<td>swap</td>
</tr>
<tr>
<td>10</td>
<td>baboon</td>
<td>baboon</td>
<td>10</td>
<td>blank</td>
</tr>
<tr>
<td>11</td>
<td>baboon</td>
<td>baboon</td>
<td>30</td>
<td>punctuation</td>
</tr>
<tr>
<td>12</td>
<td>babXoon</td>
<td>baboon</td>
<td>200</td>
<td>insert+delete</td>
</tr>
<tr>
<td>13</td>
<td>babXaoon</td>
<td>baboon</td>
<td>200</td>
<td>insert+replace</td>
</tr>
<tr>
<td>14</td>
<td>babXoon</td>
<td>baboon</td>
<td>200</td>
<td>delete+replace</td>
</tr>
<tr>
<td>15</td>
<td>babboon</td>
<td>baboon</td>
<td>200</td>
<td>finsert</td>
</tr>
<tr>
<td>16</td>
<td>aboon</td>
<td>baboon</td>
<td>120</td>
<td>trick question: swap+delete</td>
</tr>
<tr>
<td>17</td>
<td>baboon</td>
<td>baboon</td>
<td>200</td>
<td>fdelete+replace</td>
</tr>
<tr>
<td>18</td>
<td>axoon</td>
<td>baboon</td>
<td>300</td>
<td>fdelete+replace</td>
</tr>
<tr>
<td>19</td>
<td>axoo</td>
<td>baboon</td>
<td>310</td>
<td>fdelete+replace+truncate</td>
</tr>
<tr>
<td>20</td>
<td>axon</td>
<td>baboon</td>
<td>320</td>
<td>fdelete+replace+single</td>
</tr>
<tr>
<td>21</td>
<td>baby</td>
<td>baboon</td>
<td>120</td>
<td>replace+truncate*2</td>
</tr>
<tr>
<td>22</td>
<td>balloon</td>
<td>baboon</td>
<td>200</td>
<td>replace+insert</td>
</tr>
</tbody>
</table>
See Also

Functions:
- “COMPARE Function” on page 184
- “COMPLEV Function” on page 862

CALL Routines:
- “CALL COMPCOST Routine” on page 742

COMPLEV Function

Returns the Levenshtein edit distance between two strings.

**Category:** Character

**Restrictions:** This function is assigned an I18N Level 0 status, and is designed for SBCS data. Do not use this function to process DBCS or MBCS data. For more information, see [Internationalization Compatibility](#). This function is not valid on the CAS server.

**Syntax**

```
COMPLEV(string-1, string-2 <, cutoff> <, modifier(s)>)
```

**Required Arguments**

- `string-1` specifies a character constant, variable, or expression.
- `string-2` specifies a character constant, variable, or expression.

**Optional Arguments**

- `cutoff` specifies a numeric constant, variable, or expression. If the actual Levenshtein edit distance is greater than the value of `cutoff`, the value that is returned is equal to the value of `cutoff`.
  
  **TIP** Using a small value of `cutoff` improves the efficiency of COMPLEV if the values of `string-1` and `string-2` are long.

- `modifier` specifies a character string that can modify the action of the COMPLEV function. You can use one or more of the following characters as a valid modifier:
  
  - `i` or `I` ignores the case in `string-1` and `string-2`.
  - `l` or `L` removes leading blanks in `string-1` and `string-2` before comparing the values.
  - `n` or `N` removes quotation marks from any argument that is an n-literal and ignores the case of `string-1` and `string-2`. 
: (colon) truncates the longer of string-1 or string-2 to the length of the shorter string, or to one, whichever is greater.

**TIP** COMPLEV ignores blanks that are used as modifiers.

## Details

The order in which the modifiers appear in the COMPLEV function is relevant.

- "LN" first removes leading blanks from each string and then removes quotation marks from n-literals.
- "NL" first removes quotation marks from n-literals and then removes leading blanks from each string.

The COMPLEV function ignores trailing blanks.

COMPLEV returns the Levenshtein edit distance between string-1 and string-2. Levenshtein edit distance is the number of insertions, deletions, or replacements of single characters that are required to convert one string to the other. Levenshtein edit distance is symmetric. That is, \( \text{COMPLEV}(\text{string-1, string-2}) \) is the same as \( \text{COMPLEV}(\text{string-2, string-1}) \).

## Comparisons

The Levenshtein edit distance that is computed by COMPLEV is a special case of the generalized edit distance that is computed by COMPGED.

COMPLEV executes much more quickly than COMPGED.

## Example

The following example compares two strings by computing the Levenshtein edit distance.

```sas
data test;
   infile datalines missover;
   input string1 $char8. string2 $char8. modifiers $char8.;
   result=complev(string1, string2, modifiers);
   datalines;
   1234567812345678
   abc     abxc
   ac      abc
   aXc     abc
   aXbZc    abc
   aXYZc    abc
   WaXbYcZ  abc
   XYZ     abcdef
   aBc     abc
   aBc     AbC     i
   abc     abc
   abc     abc     l
   aXc     'abc'n
   aXc     'abc'n   n
;
   proc print data=test;
   run;
```
The SAS System

<table>
<thead>
<tr>
<th>Obs</th>
<th>string1</th>
<th>string2</th>
<th>modifiers</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12345678</td>
<td>12345678</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>abc</td>
<td>abxc</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>ac</td>
<td>abc</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>aXc</td>
<td>abc</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>aXbZc</td>
<td>abc</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>aXYZc</td>
<td>abc</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>WaXbYcZ</td>
<td>abc</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>XYZ</td>
<td>abcdef</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>9</td>
<td>aBe</td>
<td>abc</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>aBe</td>
<td>AbC</td>
<td>i</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>abc</td>
<td>abc</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>abc</td>
<td>abc</td>
<td>l</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>AxC</td>
<td>'abc'n</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>14</td>
<td>AxC</td>
<td>'abc'n</td>
<td>n</td>
<td>1</td>
</tr>
</tbody>
</table>

See Also

Functions:
- “COMPARE Function” on page 184
- “COMPGED Function” on page 856

CALL Routines:
- “CALL COMPCOST Routine” on page 742

CUROBS Function

Returns the observation number of the current observation.

Category: SAS File I/O

Restriction: This function is not valid on the CAS server.
Syntax

CUROBS(*data-set-id*)

**Required Argument**

*data-set-id*

is a numeric value that specifies the data set identifier that the OPEN function returns.

**Details**

If the engine being used does not support observation numbers, the function returns a missing value.

With a SAS view, the function returns the relative observation number, that is, the number of the observation within the SAS view (as opposed to the number of the observation within any related SAS data set).

**Example**

This example uses the FETCHOBS function to fetch the tenth observation in the data set MYDATA. The value of OBSNUM returned by CUROBS is 10.

```sas
%let dsid=%sysfunc(open(mydata, i));
%let rc=%sysfunc(fetchobs(&dsid, 10));
%let obsnum=%sysfunc(curobs(&dsid));
```

**See Also**

Functions:

- “FETCHOBS Function” on page 905
- “OPEN Function” on page 1010

---

**DACCDB Function**

Returns the accumulated declining balance depreciation.

**Category:** Financial

**Restriction:** This function is not valid on the CAS server.

**Syntax**

DACCDB(*p, v, y, r*)

**Required Arguments**

*p*

is numeric, the period for which the calculation is to be done. For noninteger *p* arguments, the depreciation is prorated between the two consecutive time periods that precede and follow the fractional period.
The DACCDB function returns the accumulated depreciation by using a declining balance method. The formula is

\[
DACCDB(p, v, y, r) = \begin{cases} 
0 & p \leq 0 \\
 v \left(1 - \left(1 - \frac{r}{y}\right)^{\text{int}(p)}\right) \left(1 - (p - \text{int}(p))\frac{r}{y}\right) & p > 0 
\end{cases}
\]

Note that \( \text{int}(p) \) is the integer part of \( p \). The \( p \) and \( y \) arguments must be expressed by using the same units of time. A double-declining balance is obtained by setting \( r \) equal to 2.

**Example**

An asset has a depreciable initial value of $1000 and a fifteen-year lifetime. Using a 200% declining balance, the depreciation throughout the first 10 years can be expressed as

\[
a = \text{daccdb}(10, 1000, 15, 2);
\]

The value that is returned is 760.93. The first and third arguments are expressed in years.

---

### DACCDBSL Function

Returns the accumulated declining balance with conversion to a straight-line depreciation.

**Category:** Financial

**Restriction:** This function is not valid on the CAS server.

**Syntax**

\[
\text{DACCDBSL}(p, v, y, r)
\]

**Required Arguments**

\( p \)

is numeric, the period for which the calculation is to be done.
\( v \) is numeric, the depreciable initial value of the asset.

\( y \) is an integer, the lifetime of the asset.

Range \( y > 0 \)

\( r \) is numeric, the rate of depreciation that is expressed as a fraction.

Range \( r > 0 \)

**Details**

The DACCDBSL function returns the accumulated depreciation by using a declining balance method, with conversion to a straight-line depreciation function that is defined by

\[
DACCDBSL(p, v, y, r) = \sum_{i=1}^{p} DEPDBSL(i, v, y, r)
\]

The declining balance with conversion to a straight-line depreciation chooses for each time period the method of depreciation (declining balance or straight-line on the remaining balance) that gives the larger depreciation. The \( p \) and \( y \) arguments must be expressed by using the same units of time.

**Example**

An asset has a depreciable initial value of \$1,000 and a ten-year lifetime. Using a declining balance rate of 150%, the accumulated depreciation of that asset in its fifth year can be expressed as

\[
y_5 = DACCDBSL(5, 1000, 10, 1.5);
\]

The value that is returned is 564.99. The first and the third arguments are expressed in years.

---

**DACCSL Function**

Returns the accumulated straight-line depreciation.

**Category:** Financial

**Restriction:** This function is not valid on the CAS server.

**Syntax**

\[
DACCSL(p, v, y)
\]
Required Arguments

\( p \)

is numeric, the period for which the calculation is to be done. For fractional \( p \), the depreciation is prorated between the two consecutive time periods that precede and follow the fractional period.

\( v \)

is numeric, the depreciable initial value of the asset.

\( y \)

is numeric, the lifetime of the asset.

Range \( y > 0 \)

Details

The DACCSL function returns the accumulated depreciation by using the straight-line method, which is given by

\[
DACCSL(p, v, y) = \begin{cases} 
0 & p < 0 \\
v \left(\frac{p}{y}\right) & 0 \leq p \leq y \\
v & p > y 
\end{cases}
\]

The \( p \) and \( y \) arguments must be expressed by using the same units of time.

Example

An asset, acquired on 01APR86, has a depreciable initial value of $1000 and a ten-year lifetime. The accumulated depreciation in the value of the asset through 31DEC87 can be expressed as

\[
a = \text{daccsl}(1.75, 1000, 10);
\]

The value that is returned is 175.00. The first and the third arguments are expressed in years.

DACCSYD Function

Returns the accumulated sum-of-years-digits depreciation.

Category: Financial

Restriction: This function is not valid on the CAS server.

Syntax

\[
\text{DACCSYD}(p, v, y)
\]
Required Arguments

$p$

is numeric, the period for which the calculation is to be done. For noninteger $p$
arguments, the depreciation is prorated between the two consecutive time periods
that precede and follow the fractional period.

$v$

is numeric, the depreciable initial value of the asset.

$y$

is numeric, the lifetime of the asset.

Range  $y > 0$

Details

The DACCSYD function returns the accumulated depreciation by using the sum-of-
years-digits method. The formula is

$$
DACCSYD(p, v, y) = \begin{cases} 
0 & p < 0 \\
\frac{\text{int}(p)(y - \frac{\text{int}(p)-1}{2}) + (p - \text{int}(p))(y - \text{int}(p))}{\text{int}(y)(y - \frac{\text{int}(y)-1}{2}) + (y - \text{int}(y))^2} & 0 \leq p \leq y \\
v & p > y
\end{cases}
$$

Note that int($y$) indicates the integer part of $y$. The $p$ and $y$ arguments must be expressed
by using the same units of time.

Example

An asset, acquired on 01OCT86, has a depreciable initial value of $1,000 and a five-year
lifetime. The accumulated depreciation of the asset throughout 01JAN88 can be
expressed as

$$
y_2=\text{daccsyd}(15/12, 1000, 5);
$$

The value that is returned is 400.00. The first and the third arguments are expressed in
years.

DACCTAB Function

Returns the accumulated depreciation from specified tables.

Category:  Financial

Restriction:  This function is not valid on the CAS server.

Syntax

$$
\text{DACCTAB}(p, v, t_1, ..., t_n)
$$
Required Arguments

\( p \)

is numeric, the period for which the calculation is to be done. For noninteger \( p \) arguments, the depreciation is prorated between the two consecutive time periods that precede and follow the fractional period.

\( v \)

is numeric, the depreciable initial value of the asset.

\( t_1, \ldots, t_n \)

are numeric, and are the fractions of depreciation for each time period with

\[ t_1 + t_2 + \ldots + t_n \leq 1. \]

Details

The DACCTAB function returns the accumulated depreciation by using user-specified tables. The formula for this function is

\[
DACCTAB(p, v, t_1, t_2, \ldots, t_n) = \begin{cases} 
0 & p \leq 0 \\
v(t_1 + t_2 + \ldots + t_{\text{int}(p)} + (p - \text{int}(p))t_{\text{int}(p)+1}) & 0 < p < n \\
v & p \geq n
\end{cases}
\]

For a given \( p \), only the arguments \( t_1, t_2, \ldots, t_k \) need to be specified with \( k = \text{ceil}(p) \).

Example

An asset has a depreciable initial value of $1000 and a five-year lifetime. Using a table of the annual depreciation rates of .15, .22, .21, .21, and .21 during the first, second, third, fourth, and fifth years, respectively, the accumulated depreciation throughout the third year can be expressed as

\[
y_3 = \text{dacctab}(3, 1000, .15, .22, .21, .21, .21);
\]

The value that is returned is 580.00. The fourth rate, .21, and the fifth rate, .21, can be omitted because they are not needed in the calculation.

DCCLOSE Function

Closes a directory that was opened by the DOPEN function.

Category: External Files

Restriction: This function is not valid on the CAS server.

Syntax

\text{DCCLOSE}(directory-id)

Required Argument

\text{directory-id}

is a numeric variable that specifies the identifier that was assigned when the directory was opened by the DOPEN function.
Details

DCLOSE returns 0 if the operation was successful, ≠0 if it was not successful. The DCLOSE function closes a directory that was previously opened by the DOPEN function. DCLOSE also closes any open members.

*Note:* All directories or members opened within a DATA step are closed automatically when the DATA step ends.

Examples

**Example 1: Using DCLOSE to Close a Directory**

This example opens the directory to which the fileref MYDIR has previously been assigned, returns the number of members, and then closes the directory:

```%macro memnum(filrf, path);
   %let rc=%sysfunc(filename(filrf, &path));
   %if %sysfunc(fileref(&filrf))=0 %then
     %do;
     /* Open the directory. */
     %let did=%sysfunc(dopen(&filrf));
     %put did=&did;
     /* Get the member count. */
     %let memcount=%sysfunc(dnum(&did));
     %put &memcount members in &filrf.;
     /* Close the directory. */
     %let rc=%sysfunc(dclose(&did));
     %end;
   %else %put Invalid FILEREF;
   %mend;
%memnum(MYDIR, physical-filename)```

**Example 2: Using DCLOSE within a DATA Step**

This example uses the DCLOSE function within a DATA step:

```%let filrf=MYDIR;
 data _null_;
  rc=filename("&filrf", "physical-filename");
  if fileref("&filrf")=0 then
    do;
    /* Open the directory. */
    did=dopen("&filrf");
    /* Get the member count. */
    memcount=dnum(did);
    put memcount "members in &filrf";
    /* Close the directory. */
    rc=dclose(did);
  end;
  else put "Invalid FILEREF";
run;```

See Also

Functions:
DCREATE Function

Returns the complete pathname of a new, external directory.

**Category:** External Files

**Restrictions:** This function is not valid on the CAS server.

If the SAS session in which you are specifying the FILEEXIST function is in a locked-down state, and the pathname specified in the function has not been added to the lockdown path list, then the function will fail and a file access error related to the locked-down data will not be generated in the SAS log unless you specify the SYSMSG function.

**Syntax**

\[
\text{DCREATE}(\text{directory-name} \text{, parent-directory})
\]

**Required Argument**

*directory-name*

is a character constant, variable, or expression that specifies the name of the directory to create. This value cannot include a pathname.

**Optional Argument**

*parent-directory*

is a character constant, variable, or expression that contains the complete pathname of the directory in which to create the new directory. If you do not supply a value for *parent-directory*, then the current directory is the parent directory.

**Details**

The DCREATE function enables you to create a directory in your operating environment. If the directory cannot be created, then DCREATE returns an empty string.

**Example**

To create a new directory in the UNIX operating environment, using the name that is stored in the variable DirectoryName, follow this form:

\[
\text{NewDirectory}=\text{dcreate(DirectoryName, '/local/u/abcdef/')};
\]

To create a new directory in the Windows operating environment, using the name that is stored in the variable DirectoryName, follow this form:

\[
\text{NewDirectory}=\text{dcreate(DirectoryName, 'd:\testdir')};
\]
DEPDB Function

Returns the declining balance depreciation.

Category:  Financial
Restriction:  This function is not valid on the CAS server.

Syntax

\[
\text{DEPDB}(p, v, y, r)
\]

Required Arguments

\(p\)

- is numeric, the period for which the calculation is to be done. For noninteger \(p\) arguments, the depreciation is prorated between the two consecutive time periods that precede and follow the fractional period.

\(v\)

- is numeric, the depreciable initial value of the asset.

\(y\)

- is numeric, the lifetime of the asset.

Range  \(y > 0\)

\(r\)

- is numeric, the rate of depreciation that is expressed as a fraction.

Range  \(r \geq 0\)

Details

The \text{DEPDB} function returns the depreciation by using the declining balance method, which is given by

\[
\text{DEPDB}(p, v, y, r) = \text{DACCDB}(p, v, y, r) - \text{DACCDB}(p - 1, v, y, r)
\]

The \(p\) and \(y\) arguments must be expressed by using the same units of time. A double-declining balance is obtained by setting \(r\) equal to 2.

Example

An asset has an initial value of $1,000 and a fifteen-year lifetime. Using a declining balance rate of 200\%, the depreciation of the value of the asset for the tenth year can be expressed as

\[
y_{10} = \text{DEPDB}(10, 1000, 15, 2);
\]

The value that is returned is 36.78. The first and the third arguments are expressed in years.
**DEPDBSL Function**

Returns the declining balance with conversion to a straight-line depreciation.

**Category:** Financial

**Restriction:** This function is not valid on the CAS server.

**Syntax**

\[
DEPDBSL(p, v, y, r)
\]

**Required Arguments**

- \(p\) is an integer, the period for which the calculation is to be done.
- \(v\) is numeric, the depreciable initial value of the asset.
- \(y\) is an integer, the lifetime of the asset.
  
  **Range** \(y > 0\)

- \(r\) is numeric, the rate of depreciation that is expressed as a fraction.
  
  **Range** \(r \geq 0\)

**Details**

The DEPDBSL function returns the depreciation by using the declining balance method with conversion to a straight-line depreciation, which is given by the following equation:

\[
DEPDBSL(p, v, y, r) = \begin{cases} 
0 & p \leq 0 \\
\frac{v - \frac{r}{y} \left(1 - \frac{r}{y}\right)^{p-1}\frac{r}{y}}{y} & 0 < p \leq t \\
\frac{v - \frac{r}{y} \left(1 - \frac{r}{y}\right)^{t}}{(y - t)} & t < p \leq y \\
0 & p > y 
\end{cases}
\]

The following relationship applies to the preceding equation:

\[
t = \text{int}\left(y - \frac{v}{r} + 1\right)
\]

\(\text{int}(\ )\) denotes the integer part of a numeric argument.

The \(p\) and \(y\) arguments must be expressed by using the same units of time. The declining balance that changes to a straight-line depreciation chooses for each time period the
method of depreciation (declining balance or straight-line on the remaining balance) that
gives the larger depreciation.

Example
An asset has a depreciable initial value of $1,000 and a ten-year lifetime. Using a
depreciating balance rate of 150%, the depreciation of the value of the asset in the fifth year
can be expressed as
\[ y_5 = \text{depdbsl}(5, 1000, 10, 1.5); \]
The value that is returned is 87.001041667. The first and the third arguments are
expressed in years.

DEPSL Function
Returns the straight-line depreciation.

Category: Financial

Restriction: This function is not valid on the CAS server.

Syntax
\[ \text{DEPSL}(p, v, y) \]

Required Arguments
\( p \)
is numeric, the period for which the calculation is to be done. For fractional \( p \), the
depreciation is prorated between the two consecutive time periods that precede and
follow the fractional period.

\( v \)
is numeric, the depreciable initial value of the asset.

\( y \)
is numeric, the lifetime of the asset.

Range \( y > 0 \)

Details
The DEPSL function returns the straight-line depreciation, which is given by
\[ \text{DEPSL}(p, v, y) = \text{DACCSL}(p, v, y) - \text{DACCSL}(p - 1, v, y) \]
The \( p \) and \( y \) arguments must be expressed by using the same units of time.

Example
An asset, acquired on 01APR86, has a depreciable initial value of $1,000 and a ten-year
lifetime. The depreciation in the value of the asset for the year 1986 can be expressed as
\[ d = \text{depsl}(9/12, 1000, 10); \]
The value that is returned is 75.00. The first and the third arguments are expressed in years.

**DEPSYD Function**

Returns the sum-of-years-digits depreciation.

**Category:** Financial

**Restriction:** This function is not valid on the CAS server.

**Syntax**

\[
\text{DEPSYD}(p, v, y)
\]

**Required Arguments**

- **\( p \)** is numeric, the period for which the calculation is to be done. For noninteger \( p \) arguments, the depreciation is prorated between the two consecutive time periods that precede and follow the fractional period.
- **\( v \)** is numeric, the depreciable initial value of the asset.
- **\( y \)** is numeric, the lifetime of the asset in number of depreciation periods.

**Range**  \( y > 0 \)

**Details**

The DEPSYD function returns the sum-of-years-digits depreciation, which is given by

\[
\text{DEPSYD}(p, v, y) = \text{DACCSYD}(p, v, y) - \text{DACCSYD}(p - 1, v, y)
\]

The \( p \) and \( y \) arguments must be expressed by using the same units of time.

**Example**

An asset, acquired on 01OCT86, has a depreciable initial value of $1,000 and a five-year lifetime. The depreciations in the value of the asset for the years 1986 and 1987 can be expressed as

\[
\begin{align*}
y_1 &= \text{depsyd}(3/12, 1000, 5); \\
y_2 &= \text{depsyd}(15/12, 1000, 5);
\end{align*}
\]

The values that are returned are 83.33 and 316.67, respectively. The first and the third arguments are expressed in years.

**DEPTAB Function**

Returns the depreciation from specified tables.
**Syntax**

\[ \text{DEPTAB}(p, v, t_1, ..., t_n) \]

**Required Arguments**

- \( p \) is numeric, the period for which the calculation is to be done. For noninteger \( p \) arguments, the depreciation is prorated between the two consecutive time periods that precede and follow the fractional period.

- \( v \) is numeric, the depreciable initial value of the asset.

- \( t_1, t_2, ..., t_n \) are numeric, the fractions of depreciation for each time period with \( t_1 + t_2 + ... + t_n \leq 1 \).

**Details**

The DEPTAB function returns the depreciation by using specified tables. The formula is

\[
\text{DEPTAB}(p, v, t_1, t_2, ..., t_n) = \text{DACCTAB}(p, v, t_1, t_2, ..., t_n) - \text{DACCTAB}(p - 1, v, t_1, t_2, ..., t_n)
\]

For a given \( p \), only the arguments \( t_1, t_2, ..., t_k \) need to be specified with \( k = \text{ceil}(p) \).

**Example**

An asset has a depreciable initial value of $1,000 and a five-year lifetime. Using a table of the annual depreciation rates of .15, .22, .21, .21, and .21 during the first, second, third, fourth, and fifth years, respectively, the depreciation in the third year can be expressed as

\[ y_3 = \text{deptab}(3, 1000, .15, .22, .21, .21, .21) \]

The value that is returned is 210.00. The fourth rate, .21, and the fifth rate, .21, can be omitted because they are not needed in the calculation.

---

**DINFO Function**

Returns information about a directory.

**Category:** External Files

**Restriction:** This function is not valid on the CAS server.

**UNIX specifics:** Directory pathname, owner, group, permissions, and time last modified information items are available.
Syntax

\texttt{DINFO(directory-id, information-item)}

\textbf{Required Arguments}

\textit{directory-id}

is a numeric variable that specifies the identifier that was assigned when the directory was opened by the DOPEN function.

\textit{information-item}

is a character constant, variable, or expression that specifies the information item to be retrieved. DINFO returns a blank if the value of the \textit{information-item} argument is invalid. The information available varies according to the operating environment.

\textbf{Details}

\textbf{Overview}

Directories that are opened with the DOPEN function are identified by a \textit{directory-id} value. Use the DOPTNAME function to determine the names of the available system-dependent directory information items. Use the DOPTNUM function to determine the number of directory information items that are available.

If \textit{directory-id} points to a list of concatenated directories, then the directory is the list of concatenated directory names.

\textbf{UNIX Specifics}

The following information items are available: directory pathname, which is the pathname of \textit{directory-id}, owner, group, permissions, and the time last modified.

\textbf{Examples}

\textbf{Example 1: Using DINFO to Return Information about a Directory}

This example opens the directory MYDIR, determines the number of directory information items available, and retrieves the value of the last one:

\begin{verbatim}
%let filrf=MYDIR;
%let rc=%sysfunc(filename(filrf, "physical-name"));
%let did=%sysfunc(dopen(&filrf));
%let numopts=%sysfunc(doptnum(&did));
%let foption=%sysfunc(doptname(&did, &numopts));
%let charval=%sysfunc(dinfo(&did, &foption));
%let rc=%sysfunc(dclose(&did));
\end{verbatim}

\textbf{Example 2: Windows and UNIX: Using DINFO within a DATA Step}

This example creates a data set that contains the name and value of each directory information item:

\begin{verbatim}
data diropts;
   length optname $ 32 optval $ 40;
   rc=filename("mydir", "physical-name");
   put "rc = 0 if the directory exists: " rc=;
   did=dopen("mydir");
   numopts=doptnum(did);
   do i=1 to numopts;
      optname=doptname(did, i);
      optval=dinfo(did, optname);
      put optname " : " optval;
   end;
\end{verbatim}
put i= optname=;
optval=dinfo(did, foption);
put optvals;
end;
run;

Output 3.1  Sample SAS Log for Windows

Output 3.2  Sample SAS Log for UNIX
**DNUM Function**

Returns the number of members in a directory.

**Category:** External Files

**Restriction:** This function is not valid on the CAS server.

**Syntax**

\[ \text{DNUM(directory-id)} \]

**Required Argument**

`directory-id` is a numeric variable that specifies the identifier that was assigned when the directory was opened by the DOPEN function.

**Details**

You can use DNUM to determine the highest possible member number that can be passed to DREAD.

**Examples**

**Example 1: Using DNUM to Return the Number of Members**

This example opens the directory MYDIR, determines the number of members, and closes the directory:

```
%let filrf=MYDIR;
%let rc=%sysfunc(filename(filrf, physical-name));
%let did=%sysfunc(dopen(&filrf));
%let memcount=%sysfunc(dnum(&did));
%let rc=%sysfunc(dclose(&did));
```

**Example 2: Using DNUM within a DATA Step**

This example creates a DATA step that returns the number of members in a directory called MYDIR:

```
data _null_;
```
rc=filename("mydir", *physical-name*);
did=dopen("mydir");
memcount=dnum(did);
rc=dclose(did);
run;

See Also

Functions:
- “DOPEN Function” on page 881
- “DREAD Function” on page 887

DOPEN Function

Opens a directory, and returns a directory identifier value.

**Category:** External Files

**Restrictions:** This function is not valid on the CAS server.

You must associate a fileref with the directory before calling DOPEN.

**See:** “FILENAME Function” on page 910

**Syntax**

DOPEN(*fileref*)

**Required Argument**

*fileref*

is a character constant, variable, or expression that specifies the fileref assigned to the directory. In a DATA step, *fileref* can be a character expression, a character string, or a DATA step variable whose value contains the fileref. If the function is used in a DATA Step, then *fileref* must be enclosed in quotation marks. If the function is used in macro code, then *fileref* must not be enclosed in quotation marks. In macro code, *fileref* can be any expression.

**Details**

DOPEN opens a directory and returns a directory identifier value (a number greater than 0) that is used to identify the open directory in other SAS external file access functions. If the directory cannot be opened, DOPEN returns 0, and you can obtain the error message by calling the SYSMSG function. The directory to be opened must be identified by a fileref. You can assign filerefs using the FILENAME statement or the FILENAME external file access function. Under some operating environments, you can also assign filerefs using system commands.

If you call the DOPEN function from a macro, then the result of the call is valid only when the result is passed to functions in a macro. If you call the DOPEN function from the DATA step, then the result is valid only when the result is passed to functions in the same DATA step.

**Operating Environment Information**
The term *directory* that is used in the description of this function and related SAS external file access functions refers to an aggregate grouping of files managed by the operating environment. Different operating environments identify such groupings with different names, such as directory, subdirectory, folder, MACLIB, or partitioned data set.

**Examples**

**Example 1: Using DOPEN to Open a Directory**
This example assigns the fileref MYDIR to a directory. It uses DOPEN to open the directory. DOPTNUM determines the number of system-dependent directory information items available, and DCLOSE closes the directory:

```sas
%let filrf=MYDIR;
%let rc=%sysfunc(filename(filrf, physical-name));
%let did=%sysfunc(dopen(&filrf));
%let infocnt=%sysfunc(doptnum(&did));
%let rc=%sysfunc(dclose(&did));
```

**Example 2: Using DOPEN within a DATA Step**
This example opens a directory for processing within a DATA step.

```sas
data _null_
drop rc did;
rc=filename("mydir", "physical-name");
did=dopen("mydir");
if did > 0 then do;
  ...more SAS statements...
end;
else do;
  msg=sysmsg();
  put msg;
end;
run;
```

**See Also**

Functions:
- “DCLOSE Function” on page 870
- “DOPTNUM Function” on page 884
- “FOPEN Function” on page 925
- “MOPEN Function” on page 996
- “SYSMSG Function” on page 1074

**DOPTNAME Function**

Returns directory attribute information.

**Category:** External Files

**Restriction:** This function is not valid on the CAS server.
UNIX specifics: Directory pathname, owner, group, permissions, and time last modified information items are available.

Syntax

**DOPTNAME**(directory-id, nval)

**Required Arguments**

- **directory-id**
  - is a numeric variable that specifies the identifier that was assigned when the directory was opened by the DOPEN function.

- **nval**
  - is a numeric constant, variable, or expression that specifies the sequence number of the information item.

Details

The **DOPTNAME** function returns the name of the specified information item number for a directory that was previously opened with the DOPEN function. If `directory-id` points to a list of concatenated directories, then **Directory** is the list of concatenated directory names.

**UNIX Specifics**

The following information items are available: directory pathname, which is the pathname of `directory-id`, owner, group, permissions, and time last modified.

Examples

**Example 1: Using DOPTNAME to Retrieve Directory Attribute Information**

This example opens the directory with the fileref MYDIR, retrieves all system-dependent directory information items, writes them to the SAS log, and closes the directory:

```sas
%let filrf=mydir;
%let rc=%sysfunc(filename(filrf, physical-name));
%let did=%sysfunc(dopen(&filrf));
%let infocnt=%sysfunc(doptnum(&did));
%do j=1 %to &infocnt;
  %let opt=%sysfunc(doptname(&did, &j));
  %put Directory information=&opt;
%end;
%let rc=%sysfunc(dclose(&did));
```

**Example 2: Using DOPTNAME within a DATA Step**

This example creates a data set that contains the name and value of each directory information item:

```sas
data diropts;
  length optname $ 12 optval $ 40;
  keep optname optval;
  rc=%sysfunc(filename("mydir", "physical-name"));
  did=dopen("mydir");
```
numopts=doptnum(did);
do i=1 to numopts;
    optname=doptname(did, i);
    optval=dinfo(did, optname);
    output;
end;
run;

See Also

Functions:
• “DINFO Function” on page 877
• “DOPEN Function” on page 881
• “DOPTNUM Function” on page 884

DOPTNUM Function
Returns the number of information items that are available for a directory.

Category: External Files
Restriction: This function is not valid on the CAS server.
UNIX specifics: Directory pathname, owner, group, permissions, and time last modified information items are available

Syntax
DOPTNUM(directory-id)

Required Argument
directory-id
is a numeric variable that specifies the identifier that was assigned when the directory was opened by the DOPEN function.

Details
UNIX Specifics
The following information items are available for a directory: directory pathname, which is the pathname of directory-id, owner, group, permissions, and time last modified. Therefore, this functions returns a value of 5.

Examples

Example 1: Retrieving the Number of Information Items
This example retrieves the number of system-dependent directory information items that are available for the directory MYDIR and closes the directory:

%let filrf=mydir;
%let rc=%sysfunc(filename(filrf, physical-name));
%let did=%sysfunc(dopen(&filrf));

```sas
numopts=doptnum(did);
do i=1 to numopts;
    optname=doptname(did, i);
    optval=dinfo(did, optname);
    output;
end;
run;
```
%let infocnt=%sysfunc(doptnum(&did));
%let rc=%sysfunc(dclose(&did));

Example 2: Using DOPTNUM within a DATA Step
This example creates a data set that retrieves the number of system-dependent information items that are available for the MYDIR directory:

data _null_;  
  rc=filename("mydir", "physical-name");  
  did=dopen("mydir");  
  infocnt=doptnum(did);  
  rc=dclose(did);  
run;

See Also

Functions:
• “DINFO Function” on page 877
• “DOPEN Function” on page 881
• “DOPTNAME Function” on page 882

DOSUBL Function
Imports macro variables from the calling environment, and exports macro variables back to the calling environment.

Category:  Macro
Restriction:  This function is not valid on the CAS server.

Syntax

DOSUBL(x)

Required Argument

x
  specifies a text string.

Details

The DOSUBL function enables the immediate execution of SAS code after a text string is passed. Macro variables that are created or updated during the execution of the submitted code are exported back to the calling environment.

DOSUBL returns a value of zero if SAS code was able to execute, and returns a nonzero value if SAS code was not able to execute.

Comparisons

When you use a macro to set a global macro variable, and then you invoke the macro with the DOSUBL function, the macro is executed immediately. However, when you use a DATA step to set the value of a macro variable, and then you use the CALL
EXECUTE routine to call the macro, the DATA step code executes after the current DATA step completes.

Example: Comparing DOSUBL with the CALL EXECUTE Routine

```sas
/* The %TRYIT macro is created. %TRYIT sets the global macro variable MYMAC from the macro code. */
%macro tryit(x);
%global mymac;
%if &x=1 %then %let mymac=2;
%else %let mymac=3;
%mend;

/* MYMAC is defined as a global variable, and its value is set to 4. */
%global mymac;
%let mymac=4;

/* The %TRYIT macro is invoked by the CALL EXECUTE routine. The macro code is executed immediately, and MYMAC has a value of 2. The value of the MYMAC variable is retrieved by using the SYMGET function. */
data _null_;  
call execute('%tryit(1)');  
value=symget('mymac');  
put value= '(should be 2)';  
runcell ;

/* The value of MYMAC is set to 4. */
%let mymac=4;

/* This DATA step uses the DOSUBL function to achieve the same result. */
data _null_;  
rc=dosubl('%tryit(1)');  
value=symget('mymac');  
put value= '(should be 2)';  
runcell ;

/* The definition of %TRYIT is changed so that a DATA step is invoked to set the value of MYMAC. */
%macro tryit(x);
%global mymac;
data _null_;  
if &x=1 then mymac=2;  
else mymac=3;  
call symputx('mymac', mymac);  
runcell ;
%mend;

/* The value of MYMAC is set to 4 by the %LET macro. CALL EXECUTE calls the %TRYIT macro to be invoked immediately, but the macro contains a DATA step. The DATA step code executes after the current DATA step completes. Therefore, the value of MYMAC remains 4. The call to the SYMGET function does not retrieve the proper */
```
/* value of MYMAC, which is 2. */
%let mymac=4;
data _null_
   call execute('%tryit(1)');
   value=symget('mymac');
   put value= '(should be 2)';
run;

/* If you use the DOSUBL function, the code in the %TRYIT macro is */
/* executed immediately, and the SYMGET function returns a value of 2. */
%let mymac=4;
data _null_
   rc=dosubl('%tryit(1)');
   value=symget('mymac');
   put value= '(should be 2)';
run;

See Also

CALL Routines:
- “CALL EXECUTE Routine” on page 745

DREAD Function

Returns the name of a directory member.

Category: External Files
Restriction: This function is not valid on the CAS server.

Syntax

DREAD(directory-id, nval)

Required Arguments

directory-id
  is a numeric value that specifies the identifier that was assigned when
  the directory was opened by the DOPEN function.

nval
  is a numeric constant, variable, or expression that specifies the sequence number
  of the member within the directory.

Details

DREAD returns a blank if an error occurs (such as when nval is out-of-range). Use
DNUM to determine the highest possible member number that can be passed to
DREAD.
Example

This example opens the directory identified by the fileref MYDIR, retrieves the number of members, and places the number in the variable MEMCOUNT. It then retrieves the name of the last member, places the name in the variable LSTNAME, and closes the directory:

```
%let filrf=mydir;
%let rc=%sysfunc(filename(filrf, physical-name));
%let did=%sysfunc(dopen(&filrf));
%let lstname=;
%let memcount=%sysfunc(dnum(&did));
%if &memcount > 0 %then
  %let lstname=%sysfunc(dread(&did, &memcount));
%let rc=%sysfunc(dclose(&did));
```

See Also

Functions:
- “DNUM Function” on page 880
- “DOPEN Function” on page 881

DROPNOTE Function

Deletes a note marker from a SAS data set or an external file.

**Categories:** SAS File I/O, External Files

**Restriction:** This function is not valid on the CAS server.

**Syntax**

```
DROPNOTE(data-set-id | file-id, note-id)
```

**Required Arguments**

- `data-set-id | file-id`
  - is a numeric variable that specifies the identifier that was assigned when the data set or external file was opened, generally by the OPEN function or the FOPEN function.

- `note-id`
  - is a numeric value that specifies the identifier that was assigned by the NOTE or FNOTE function.

**Details**

DROPNOTE deletes a marker set by NOTE or FNOTE. It returns a 0 if successful and ≠0 if not successful.
Example

This example opens the SAS data set MYDATA, fetches the first observation, and sets a note ID at the beginning of the data set. It uses POINT to return to the first observation, and then uses DROPNOTE to delete the note ID:

```sas
%let dsid=%sysfunc(open(mydata, i));
%let rc=%sysfunc(fetch(&dsid));
%let noteid=%sysfunc(note(&dsid));
   more macro statements
%let rc=%sysfunc(point(&dsid, &noteid));
%let rc=%sysfunc(fetch(&dsid));
%let rc=%sysfunc(dropnote(&dsid, &noteid));
```

See Also

Functions:
- “FETCH Function” on page 903
- “FNOTE Function” on page 924
- “FOPEN Function” on page 925
- “FPOINT Function” on page 932
- “NOTE Function” on page 1006
- “OPEN Function” on page 1010
- “POINT Function” on page 1019

---

**DSNAME Function**

Returns the SAS data set name that is associated with a data set identifier.

**Category:** SAS File I/O

**Restriction:** This function is not valid on the CAS server.

**Syntax**

`DSNAME(data-set-id)`

**Required Argument**

`data-set-id`

is a numeric variable that specifies the data set identifier that is returned by the OPEN function.

**Details**

DSNAME returns the data set name that is associated with a data set identifier, or a blank if the data set identifier is not valid.
Example
This example determines the name of the SAS data set that is associated with the variable DSID and displays the name in the SAS log.

```sas
%let dsid=%sysfunc(open(sasuser.houses,i));
%put The current open data set is %sysfunc(dsname(&dsid)).;
```

See Also
Functions:
- “OPEN Function” on page 1010

**ENVLEN Function**

Returns the length of an environment variable.

**Syntax**

`ENVLEN(argument)`

**Required Argument**

`argument`

specifies a character variable that is the name of an operating system environment variable. Enclose `argument` in quotation marks.

**Details**

The ENVLEN function returns the length of the value of an operating system environment variable. If the environment variable does not exist, SAS returns –1.

**Operating Environment Information**

The value of `argument` is specific to your operating environment.

**Example**

The following examples are for illustration purposes only. The actual value that is returned depends on where SAS is installed on your computer.

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>/* Windows operating environment */ 309</td>
<td>x=envlen(&quot;PATH&quot;); put x;</td>
</tr>
</tbody>
</table>
SAS Statement | Result
---|---
/* UNIX operating environment */
y=envlen("PATH");
put y; | 365

z=envlen("THIS IS NOT DEFINED");
put z; | -1

---

**EUCLID Function**

Returns the Euclidean norm of the nonmissing arguments.

**Category:** Descriptive Statistics  
**Restriction:** This function is not valid on the CAS server.

**Syntax**

\[ \text{EUCLID}(\text{value-1} <, \text{value-2} ...>) \]

**Required Argument**

\( \text{value} \)

specifies a numeric constant, variable, or expression.

**Details**

If all arguments have missing values, then the result is a missing value. Otherwise, the result is the Euclidean norm of the nonmissing values.

In the following example, \( x_1, x_2, ..., x_n \) are the values of the nonmissing arguments.

\[ \text{EUCLID}(x_1, x_2, ..., x_n) = \sqrt{x_1^2 + x_2^2 + ... + x_n^2} \]

**Examples**

**Example 1: Calculating the Euclidean Norm of Nonmissing Arguments**

The following example returns the Euclidean norm of the nonmissing arguments.

```sas
data _null_;  
  x=euclid(., 3, 0, .q, -4);  
  put x=;  
run;
```

SAS writes the following output to the log:

\[ x=5 \]
**Example 2: Calculating the Euclidean Norm When You Use a Variable List**

The following example uses a variable list to calculate the Euclidean norm.

```sas
data _null_;
x1=1;
x2=3;
x3=4;
x4=3;
x5=1;
x=euclid(of x1-x5);
put x=;
run;
```

SAS writes the following output to the log:

```
x=6
```

**See Also**

**Functions:**
- “LPNORM Function” on page 990
- “RMS Function” on page 572

**EXIST Function**

Verifies the existence of a SAS library member.

**Category:** SAS File I/O  
**Restriction:** This function is not valid on the CAS server.

**Syntax**

```sas
EXIST(member-name <, member-type <, generation>>)
```

**Required Argument**

`member-name`

is a character constant, variable, or expression that specifies the SAS library member. If `member-name` is blank or a null string, then `EXIST` uses the value of the `_LAST_` system variable as the member name.

**Optional Arguments**

`member-type`

is a character constant, variable, or expression that specifies the type of SAS library member. A few common member types include ACCESS, CATALOG, DATA, and VIEW. If you do not specify a `member-type`, then the member type DATA is assumed.
is a numeric constant, variable, or expression that specifies the generation number of the SAS data set whose existence you are checking. If `member-type` is not DATA, `generation` is ignored.

Positive numbers are absolute references to a historical version by its generation number. Negative numbers are relative references to a historical version in relation to the base version, from the youngest predecessor to the oldest. For example, –1 refers to the youngest version or, one version back from the base version. Zero is treated as a relative generation number.

Details

If you use a sequential library, then the results of the EXIST function are undefined. If you do not use a sequential library, then EXIST returns 1 if the library member exists, or 0 if `member-name` does not exist or `member-type` is invalid.

Use the CEXIST function to verify the existence of an entry in a catalog.

Examples

**Example 1: Verifying the Existence of a Data Set**

This example verifies the existence of a data set. If the data set does not exist, then the example displays a message in the log:

```sas
%let dsname=sasuser.houses;
%macro opends(name);
  %if %sysfunc(exist(&name)) %then
    %let dsid=%sysfunc(open(&name, i));
  %else %put Data set &name does not exist.;
%mend opends;
%opends(&dsname);
```

**Example 2: Verifying the Existence of a Data View**

This example verifies the existence of the SAS view TEST.MYVIEW. If the view does not exist, then the example displays a message in the log:

```sas
data _null_;
dsnname="test.myview";
  if (exist(dsnname, "VIEW")) then
dsid=opend(dsnname, "i");
  else put dsnname 'does not exist.';
run;
```

**Example 3: Determining If a Generation Data Set Exists**

This example verifies the existence of a generation data set by using positive generation numbers (absolute reference):

```sas
data new(genmax=3);
x=1;
run;
data new;
x=99;
run;
data new;
x=100;
```
run;
data new;
x=101;
run;
data _null_;  
    test=exist('new', 'DATA', 4);
    put test=;  
    test=exist('new', 'DATA', 3);
    put test=;  
    test=exist('new', 'DATA', 2);
    put test=;  
    test=exist('new', 'DATA', 1);
    put test=;
run;

SAS writes the following output to the log:

test=1
  test=1
  test=1
  test=0

You can change this example to verify the existence of the generation data set by using negative numbers (relative reference):

data new2(genmax=3);
    x=1;
run;
data new2;
    x=99;
run;
data new2;
    x=100;
run;
data new2;
    x=101;
run;
data _null_;  
    test=exist('new2', 'DATA', 0);
    put test=;  
    test=exist('new2', 'DATA', -1);
    put test=;  
    test=exist('new2', 'DATA', -2);
    put test=;  
    test=exist('new2', 'DATA', -3);
    put test=;  
    test=exist('new2', 'DATA', -4);
    put test=;
run;

SAS writes the following output to the log:

test=1
  test=1
  test=1
  test=0
  test=0
See Also

Functions:

- “CEXIST Function” on page 850
- “FEXIST Function” on page 906
- “FILEEXIST Function” on page 909

FAPPEND Function

Appends the current record to the end of an external file.

**Syntax**

\[
\text{FAPPEND}(\text{file-id} <, \text{cc}>)
\]

**Required Argument**

\textit{file-id} is a numeric variable that specifies the identifier that was assigned when the file was opened, generally by the FOPEN function.

**Optional Argument**

\textit{cc} is a character constant, variable, or expression that specifies a carriage-control character:

- \textit{blank} indicates that the record starts a new line.
- \text{0} skips one blank line before this new line.
- \text{-} skips two blank lines before this new line.
- \text{1} specifies that the line starts a new page.
- \text{+} specifies that the line overstrikes a previous line.
- \text{P} specifies that the line is a computer prompt.
- \text{=} specifies that the line contains carriage control information.
- \text{all else} specifies that the line record starts a new line.

**Details**

FAPPEND adds the record that is currently contained in the File Data Buffer (FDB) to the end of an external file. FAPPEND returns a 0 if the operation was successful and \#0 if it was not successful.
Example

This example assigns the fileref MYFILE to an external file and attempts to open the file. If the file is opened successfully, it moves data into the File Data Buffer, appends a record, and then closes the file. Note that in a macro statement that you do not enclose character strings in quotation marks.

```sas
%let filrf=myfile;
%let rc=%sysfunc(filename(filrf, physical-filename));
%let fid=%sysfunc(fopen(&filrf, a));
%if &fid > 0 %then
  %do;
    %let rc=%sysfunc(fput(&fid, Data for the new record));
    %let rc=%sysfunc(fappend(&fid));
    %let rc=%sysfunc(fclose(&fid));
  %end;
%else
  %do;
    /* unsuccessful open processing */
  %end;
```

See Also

Functions:

- “DOPEN Function” on page 881
- “FCLOSE Function” on page 896
- “FGET Function” on page 907
- “FOPEN Function” on page 925
- “FPUT Function” on page 936
- “FWRITE Function” on page 942
- “MOPEN Function” on page 996

FCLOSE Function

Closes an external file, directory, or directory member.

**Category:** External Files

**Restriction:** This function is not valid on the CAS server.

**Syntax**

`FCLOSE(file-id)`

**Required Argument**

`file-id`

is a numeric variable that specifies the identifier that was assigned when the file was opened, generally by the FOPEN function.
Details

FCLOSE returns a 0 if the operation was successful and ≠0 if it was not successful. If you open a file within a DATA step, it is closed automatically when the DATA step ends.

Operating Environment Information

Under UNIX, FCLOSE is not required at the end of the DATA step.

Example

This example assigns the fileref MYFILE to an external file, and attempts to open the file. If the file is opened successfully, indicated by a positive value in the variable FID, the program reads the first record, closes the file, and deassigns the fileref:

```sas
%let filrf=myfile;
%let rc=%sysfunc(filename(filrf, physical-filename));
%let fid=%sysfunc(fopen(&filrf));
%if &fid > 0 %then
  %do;
    %let rc=%sysfunc(fread(&fid));
    %let rc=%sysfunc(fclose(&fid));
  %end;
%else
  %do;
    %put %sysfunc(sysmsg());
  %end;
%let rc=%sysfunc(filename(filrf));
```

See Also

Functions:

- “DCLOSE Function” on page 870
- “DOPEN Function” on page 881
- “FOPEN Function” on page 925
- “FREAD Function” on page 937
- “MOPEN Function” on page 996

---

**FCOL Function**

Returns the current column position in the File Data Buffer (FDB).

**Category:** External Files

**Restriction:** This function is not valid on the CAS server.

### Syntax

```
FCOL(file-id)
```
**Required Argument**

`file-id`  
is a numeric variable that specifies the identifier that was assigned when the file was opened, generally by the FOPEN function.

**Details**

Use FCOL combined with FPOS to manipulate data in the File Data Buffer (FDB).

**Example**

This example assigns the fileref MYFILE to an external file and attempts to open the file. If the file is successfully opened, indicated by a positive value in the variable FID, it puts more data into the FDB relative to position POS, writes the record, and closes the file:

```sas
%let filrf=myfile;
%let rc=%sysfunc(filename(filrf, physical-filename));
%let fid=%sysfunc(fopen(&filrf, o));
%if (&fid > 0) %then
  %do;
  %let record=This is data for the record.;
  %let rc=%sysfunc(fput(&fid, &record));
  %let pos=%sysfunc(fcol(&fid));
  %let rc=%sysfunc(fpos(&fid, %eval(&pos+1)));  
  %let rc=%sysfunc(fput(&fid, more data));
  %let rc=%sysfunc(fwrite(&fid));
  %let rc=%sysfunc(fclose(&fid));
  %end;
%let rc=%sysfunc(filename(filrf));
```

SAS writes the following new record to the external file:

```
This is data for the record.
```

**See Also**

**Functions:**

- “FCLOSE Function” on page 896
- “FOPEN Function” on page 925
- “FPOS Function” on page 934
- “FPUT Function” on page 936
- “FWRITE Function” on page 942
- “MOPEN Function” on page 996

---

**FCOPY Function**

Copies records from one fileref to another fileref, and returns a value that indicates whether the records were successfully copied.

**Category:** SAS File I/O
Restriction: This function is not valid on the CAS server.

Syntax

FCOPY('fileref-1', 'fileref-2')

Required Arguments

'fileref-1'
specifies an existing fileref from which records are to be copied.

'fileref-2'
specifies an existing fileref to which records are to be copied.

Details

Values That Are Returned by the FCOPY Function

FCOPY returns these values:

- a value of 0 if records were copied without errors or warnings
- a positive value if an error occurred
- a negative value if a warning was issued

You can use the SYMSMG function to retrieve error or warning messages, and you can use the SYSRC function to retrieve the return code.

Using Macro Variables with the FCOPY Function

The following macro variables provide information for the FCOPY function:

- The &SYSCC and &SYSERR macro variables are set if FCOPY writes an error or warning message to the log.
- The &SYSCC and &SYSERR macro variables are not set if FCOPY returns a warning return code and there is no log output from FCOPY.
- The &SYSERRORTEXT macro variable is set if FCOPY writes an error message to the log.
- The &SYSWARNINGTEXT macro variable is set if FCOPY writes a warning message to the log.

Setting a Logical Record Length for a Text File

The default logical record length for reading from or writing to external files is 32,767 bytes. The maximum logical record length is 1 gigabyte. Text files have a data stream that consists of an unstructured sequence of bytes. A delimiter, such as a carriage-control character, controls the length of the data stream, and divides the information in the data stream into records. If the length of a record is greater than 32,767 bytes, you must define the logical record length of your records so that your data is not truncated when FCOPY copies the text file. To set the logical record length to a larger value, use the LRECL= system option in an OPTIONS statement, or the LRECL= option in a FILENAME statement.

For an example that shows how to set an LRECL value, see “Example 3: Diagnostic Messages” on page 901.

Tip: Selecting an arbitrarily large value for the LRECL= option can result in excessive use of memory, which can degrade performance.
Examples

Example 1: Copying a Text File
Setting the MSGLEVEL= system option to I causes informational messages from FCOPY to be written to the log.

```sas
/* Set MSGLEVEL to I to write messages from FCOPY to the log. */
options msglevel=i;

filename src 'source.txt';
filename dest 'destination.txt';

/* Create an example file to copy. */
data _null_; file src;
do i=1, 2105, 300312, 400501;
   put i:words256.;
end;
run;

/* Copy the records of SRC to DEST. */
data _null_; length msg $ 384;
   rc=fcopy('src', 'dest');
   if rc=0 then
      put 'Copied SRC to DEST.,'
         msg='Copied SRC to DEST.';
   else do;
      msg=sysmsg();
      put rc= msg=
         msg='Copied SRC to DEST.';
   end;
run;
```

SAS writes the following output to the log:

```
INFO: The source fileref SRC for the FCOPY function is:
   Filename=your-source-file,
   RECFM=V,LRECL=32767,File Size (bytes)=121,
   Last Modified=15Aug2012:11:21:39,
   Create Time=15Aug2012:09:13:38
INFO: The destination fileref DEST for the FCOPY function is:
   Filename=your-destination-file,
   RECFM=V,LRECL=32767,File Size (bytes)=0,
   Last Modified=15Aug2012:11:21:39,
   Create Time=15Aug2012:09:13:38
Copied SRC to DEST.
```

Example 2: Copying a Binary File
This example copies a binary file from one directory to another. Setting the MSGLEVEL= system option to I causes informational messages from FCOPY to be written to the log.

```sas
/* Set MSGLEVEL to I to write messages from FCOPY to the log. */
options msglevel=i;
```

filename src 'raises.xlsx' recfm=n;
filename dest 'raises-2012.xlsx' recfm=n;

/* Create an example file to copy. */
data _null_;  
  file src;  
  do i=1, 2105, 300312, 400501;  
    put i:words256.;es  
  end;  
run;

data _null_;  
  length msg $ 384;  
  rc=fcopy('src', 'dest');  
  if rc=0 then  
    put 'Copied SRC to DEST.';  
  else do;  
    msg=sysmsg();  
    put rc= msg=;  
  end;  
run;

SAS writes the following output to the log:

INFO: The source fileref SRC for the FCOPY function is:  
Filename=your-source-file,  
RECFM=N,LRECL=256,File Size (bytes)=117,  
Last Modified=15Aug2012:12:49:18,  
Create Time=15Aug2012:12:42:00

INFO: The destination fileref DEST for the FCOPY function is:  
Filename=your-destination-file,  
RECFM=N,LRECL=256,File Size (bytes)=0,  
Last Modified=15Aug2012:12:49:18,  
Create Time=15Aug2012:12:42:01

Copied SRC to DEST.

Example 3: Diagnostic Messages
This example shows diagnostic messages that result from the FCOPY function when the MSGLEVEL= system option is set to 1. The file to be copied from has a record length of 256 bytes, and the file to be copied to has a record length of 5 bytes. Warning messages identify that the file was truncated.

filename src 'source.txt' lrecl=256;

/* Create example file to copy. */
data _null_;  
  file src;  
  do i=1, 2105, 300312, 400501;  
    put i:words256.;  
  end;  
run;

/* Make LRECL for DEST short, to force output truncation. */
filename dest 'destination.txt' lrecl=5;
/* Set MSGLEVEL to I to write messages from FCOPY to the log. */
options msglevel=i;
data _null_;  
rc=fcopy('src', 'dest');
run;

SAS writes the following output to the log:

```
INFO: The source fileref SRC for the FCOPY function is:
Filename=your-source-file,
RECFM=V,LRECL=256,File Size (bytes)=121,
Last Modified=15Aug2012:15:14:38,
Create Time=15Aug2012:09:13:38

INFO: The destination fileref DEST for the FCOPY function is:
Filename=your-destination-file,
RECFM=V,LRECL=5,File Size (bytes)=0,
Last Modified=15Aug2012:15:14:38,

WARNING: 3 records were truncated when the FCOPY function wrote to fileref DEST.
WARNING: To prevent the truncation of records in future operations, you can
increase amount of space needed to accomodate the records by using
the LRECL= system option or the LRECL= option in the FILENAME statement.
```

---

### FDELETE Function

Deletes an external file or an empty directory.

**Category:** External Files  
**Restriction:** This function is not valid on the CAS server.  
**See:** "FILENAME Function" on page 910

#### Syntax

```
FDELETE(fileref | directory)
```

#### Required Arguments

- `fileref`
  - is a character constant, variable, or expression that specifies the `fileref` that you assign to the external file or directory. You can assign `filerefs` by using the `FILENAME` statement, the `FILENAME` external file access function, or the `FILENAME` statement, FTP, Catalog, Hadoop, WebDAV, and ZIP access methods.

  **Restriction**  
  - The `fileref` that you use with FDELETE cannot be a concatenation.

- `directory`
  - is a character constant, variable, or expression that specifies an empty directory that you want to delete.
**Restriction**

You must have authorization to delete the directory.

**Details**

FDELETE returns 0 if the operation was successful or a nonzero number if it was not successful.

**Examples**

**Example 1: Deleting an External File**

This example generates a fileref for an external file in the variable FNAME. Then it calls FDELETE to delete the file and calls the FILENAME function again to deassign the fileref.

```sas
data _null_
  fname="tempfile"
  rc=filename(fname, "physical-filename");
  if rc = 0 and fexist(fname) then
    rc=fdelete(fname);
  rc=filename(fname);
run;
```

**Example 2: Deleting a Directory**

This example uses FDELETE to delete an empty directory to which you have Write access. If the directory is not empty, the optional SYSMSG function returns an error message stating that SAS is unable to delete the file.

```sas
filename testdir 'physical-filename';
data _null_
  rc=fdelete('testdir');
  put rc=;
  msg=sysmsg();
  put msg=;
run;
```

**See Also**

**Functions:**

- “FEXIST Function” on page 906

**Statements:**

- “FILENAME Statement” in *SAS Viya Statements: Reference*

---

**FETCH Function**

Reads the next non-deleted observation from a SAS data set into the Data Set Data Vector (DDV).

**Category:** SAS File I/O

**Restriction:** This function is not valid on the CAS server.
Syntax

FETCH(data-set-id <, 'NOSET'>)

Required Argument

data-set-id

is a numeric variable that specifies the data set identifier that is returned by the
OPEN function.

Optional Argument

'NOSET'

prevents the automatic passing of SAS data set variable values to macro or DATA
step variables even if the SET routine has been called.

Details

FETCH returns a 0 if the operation is successful, ≠0 if it is not successful, and −1 if the
end of the data set is reached. FETCH skips observations marked for deletion.

If the SET routine has been called previously, the values for any data set variables are
automatically passed from the DDV to the corresponding DATA step or macro variables.
To override this behavior temporarily so that fetched values are not automatically copied
to the DATA step or macro variables, use the NOSET option.

Example

This example fetches the next observation from the SAS data set MYDATA. If the end
of the data set is reached or if an error occurs, SYSMSG retrieves the appropriate
message and writes it to the SAS log. Note that in a macro statement that you do not
enclose character strings in quotation marks.

%let dsid=%sysfunc(open(mydata, i));
%let rc=%sysfunc(fetch(&dsid));
%if &rc ne 0 %then
  %put %sysfunc(sysmsg());
%else
  %do;
    ... more macro statements ...
  %end;
%let rc=%sysfunc(close(&dsid));

See Also

Functions:

• “FETCHOBS Function” on page 905
• “GETVARC Function” on page 944
• “GETVARN Function” on page 945

CALL Routines:

• “CALL SET Routine” on page 833
FETCHOBS Function

Reads a specified observation from a SAS data set into the Data Set Data Vector (DDV).

**Category:** SAS File I/O

**Restriction:** This function is not valid on the CAS server.

**Syntax**

\[
\text{FETCHOBS}(\text{data-set-id, observation-number, options})
\]

**Required Arguments**

- **data-set-id** is a numeric variable that specifies the data set identifier that is returned by the OPEN function.

- **observation-number** is a numeric constant, variable, or expression that specifies the number of the observation to read. FETCHOBS treats the observation value as a relative observation number unless you specify the ABS option. The relative observation number might not coincide with the physical observation number on disk, because the function skips observations marked for deletion. When a WHERE clause is active, the function counts only observations that meet the WHERE condition.

**Default** FETCHOBS skips deleted observations.

**Optional Argument**

- **options** is a character constant, variable, or expression that names one or more options, separated by blanks:

  - **ABS** specifies that the value of **observation-number** is absolute. That is, deleted observations are counted.

  - **NOSET** prevents the automatic passing of SAS data set variable values to DATA step or macro variables even if the SET routine has been called.

**Details**

FETCHOBS returns 0 if the operation was successful, ≠0 if it was not successful, and −1 if the end of the data set is reached. To retrieve the error message that is associated with a nonzero return code, use the SYSMSG function. If the SET routine has been called previously, the values for any data set variables are automatically passed from the DDV to the corresponding DATA step or macro variables. To override this behavior temporarily, use the NOSET option.

If **observation-number** is less than 1, the function returns an error condition. If **observation-number** is greater than the number of observations in the SAS data set, the function returns an end-of-file condition.
Example

This example fetches the tenth observation from the SAS data set MYDATA. If an error occurs, the SYSMSG function retrieves the error message and writes it to the SAS log. Note that in a macro statement that you do not enclose character strings in quotation marks.

```sas
%let rc = %sysfunc(fetchobs(&mydataid,10));
%if &rc = -1 %then
   %put End of data set has been reached.;
%if &rc > 0 %then %put %sysfunc(sysmsg());
```

See Also

Functions:
- “FETCH Function” on page 903
- “GETV ARC Function” on page 944
- “GETVARN Function” on page 945

CALL Routines:
- “CALL SET Routine” on page 833

FEXIST Function

Verifies the existence of an external file that is associated with a fileref.

Category: External Files

Restriction: This function is not valid on the CAS server.

See: “FILENAME Function” on page 910

Syntax

FEXIST(fileref)

Required Argument

- **fileref**
  - is a character constant, variable, or expression that specifies the fileref that is assigned to an external file.

Restriction: Fileref must have been previously assigned.

Operating environment: Under UNIX if the function is used in a DATA step, fileref or the environment variable that you specify must be enclosed in quotation marks. If the function is used in macro code, fileref must not be enclosed in quotation marks.
Details

FEXIST returns 1 if the external file that is associated with fileref exists, and 0 if the file does not exist. You can assign filerefs by using the FILENAME statement or the FILENAME external file access function. In some operating environments, you can also assign filerefs by using system commands.

Note: If you are using the FILENAME statement, FTP access method to reference the file used in the FEXIST function and the function reports a data set not found but the data set is cataloged, use the FTP access method’s WAIT_MILLISECONDS option to increase the response time.

Comparisons

FILEEXIST verifies the existence of a file based on its physical name.

Example

This example verifies the existence of an external file and writes the result to the SAS log:

```sas
%if %sysfunc(fexist(&fref)) %then
   %put The file identified by the fileref &fref exists.;
%else
   %put %sysfunc(sysmsg());
```

See Also

Functions:

- “EXIST Function” on page 892
- “FILEEXIST Function” on page 909
- “FILEREF Function” on page 913

Statements:

- “FILENAME Statement” in SAS Viya Statements: Reference

---

**FGET Function**

Copies data from the File Data Buffer (FDB) into a variable.

**Category:** External Files

**Restriction:** This function is not valid on the CAS server.

**Syntax**

```
FGET(file-id, variable <, length>)
```
Required Arguments

file-id

is a numeric variable that specifies the identifier that was assigned when the file was
opened, generally by the FOPEN function.

variable

in a DATA step, specifies a character variable to hold the data. In a macro, specifies a
macro variable to hold the data. If variable is a macro variable and it does not exist,
it is created.

Optional Argument

length

specifies the number of characters to retrieve from the FDB. If length is specified,
only the specified number of characters is retrieved (or the number of characters
remaining in the buffer if that number is less than length). If length is omitted, all
characters in the FDB from the current column position to the next delimiter are
returned. The default delimiter is a blank. The delimiter is not retrieved.

See “FSEP Function” on page 941 for more information about delimiters.

Details

FGET returns 0 if the operation was successful, or −1 if the end of the FDB was reached
or no more tokens were available.

After FGET is executed, the column pointer moves to the next read position in the FDB.

Example

This example assigns the fileref MYFILE to an external file and attempts to open the
file. If the file is opened successfully, it reads the first record into the File Data Buffer,
retrieves the first token of the record and stores it in the variable MYSTRING, and then
closes the file. Note that in a macro statement that you do not enclose character strings in
quotation marks.

%let filrf=myfile;
%let rc=%sysfunc(filename(filrf, physical-filename));
%let fid=%sysfunc(fopen(&filrf));
%if &fid > 0 %then
  %do;
    %let rc=%sysfunc(fread(&fid));
    %let rc=%sysfunc(fget(&fid, mystring));
    %put &mystring;
    %let rc=%sysfunc(fclose(&fid));
  %end;
%let rc=%sysfunc(filename(filrf));

See Also

Functions:

• “FCLOSE Function” on page 896
• “FILENAME Function” on page 910
• “FOPEN Function” on page 925
FILEEXIST Function

Verifies the existence of an external file by its physical name.

**Category:** External Files

**Restrictions:** This function is not valid on the CAS server.

If the SAS session in which you are specifying the FILEEXIST function is in a locked-down state, and the pathname specified in the function has not been added to the lockdown path list, then the function will fail and a file access error related to the locked-down data will not be generated in the SAS log unless you specify the SYSMSG function.

**See:** “FILENAME Function” on page 910

**Syntax**

FILEEXIST(*file-name*)

**Required Argument**

*filename*

is a character constant, variable, or expression that specifies a fully qualified physical filename of the external file in the operating environment.

**Details**

FILEEXIST returns 1 if the external file exists and 0 if the external file does not exist. The specification of the physical name for *filename* varies according to the operating environment.

Although your operating environment utilities might recognize partial physical filenames, you must always use fully qualified physical filenames with FILEEXIST.

FILEEXIST can also verify a directory’s existence.

**Example**

This example verifies the existence of an external file. If the file exists, FILEEXIST opens the file. If the file does not exist, FILEEXIST displays a message in the SAS log. Note that in a macro statement that you do not enclose character strings in quotation marks.

```sas
%if %sysfunc(fileexist(&myfilerf)) %then
  %let fid=%sysfunc(fopen(&myfilerf));
%else
  %put The external file &myfilerf does not exist.;
```
FILENAME Function

Assigns or deassigns a fileref to an external file, directory, or output device.

**Category:** External Files

**Restrictions:** If the SAS session in which you are specifying the FILENAME function is in a locked-down state, and the pathname specified in the function has not been added to the lockdown path list, then the function fails and a file access error related to the locked-down data is not generated in the SAS log unless you specify the SYSMSG function.

This function is not valid on the CAS server.

**Note:** You can assign a fileref by using a File Shortcut in the Navigation pane, the FILENAME statement, the FILENAME function, or you can use a Windows environment variable to point to the file.

**Syntax**

FILENAME(fileref, filename <, device-type> <, 'host-options'> <, directory-reference>);

**Required Arguments**

*fileref*

specifies the fileref to assign to the external file. In a DATA step, *fileref* can be a character expression, a string enclosed in single quotation marks that specifies the fileref, or a DATA step variable whose value contains the fileref. In a macro (for example, in the %SYSFUNC function), *fileref* is the name of a macro variable (without an ampersand) whose value contains the fileref to assign to the external file. If the function is used within a DATA step, the fileref must be enclosed in single quotation marks. If the function is used in macro code, the fileref must not be enclosed in quotation marks.

**Requirement**

If *fileref* is a DATA step variable, its length must be no longer than eight characters.

**Tip**

If a fileref is a DATA step character variable with a blank value and a maximum length of eight characters, or if a macro variable named in *fileref* has a null value, then a fileref is generated and assigned to the character variable or macro variable, respectively.

*filename*

is a character constant, variable, or expression that specifies the external file.

---

**See Also**

**Functions:**

- “EXIST Function” on page 892
- “FEXIST Function” on page 906
- “FILEREF Function” on page 913
- “FOPEN Function” on page 925
Tip You can deassign a fileref that was previously assigned by passing a blank value for the `filename` argument.

**Optional Arguments**

`device-type` is a character constant, variable, or expression that specifies the type of device or the access method that is used if the fileref points to an input or output device or location that is not a physical file:

- **DISK**
  - Specifies that the device is a disk drive.
  - **Alias**: `BASE`

  Tip When you assign a fileref to a file on disk, you are not required to specify DISK.

- **DUMMY**
  - Specifies that the output to the file is discarded.
  - **Tip**: Specifying DUMMY can be useful for testing.

- **GTERM**
  - Indicates that the output device type is a graphics device that is receiving graphics data.

- **PIPE**
  - Specifies an unnamed pipe.
  - **Note**: Some operating environments do not support pipes.

- **PLOTTER**
  - Specifies an unbuffered graphics output device.

- **PRINTER**
  - Specifies a printer or printer spool file.

- **TAPE**
  - Specifies a tape drive.

- **TEMP**
  - Creates a temporary file that exists only as long as the filename is assigned. The temporary file can be accessed only through the logical name and is available only while the logical name exists.

  **Restriction**: Do not specify a physical pathname. If you do, SAS returns an error.

  **Tip**: Files that are manipulated by the TEMP device can have the same attributes and behave identically to DISK files.

- **TERMINAL**
  - Specifies the user's personal computer.

- **UPRINTER**
  - Specifies a Universal Printing printer definition name.

Operating Environment Information
The FILENAME function also supports operating-environment specific devices. For more information, see the SAS documentation for your operating environment.

*host-options*

specifies host-specific details such as file attributes and processing attributes.

directory-reference

specifies the fileref that was assigned to the directory or partitioned data set in which the external file resides.

Details

You can create a fileref with the FILENAME function and FILENAME statement.

FILENAME returns 0 if the operation was successful; ≠0 if it was not successful. The name that is associated with the file or device is called a fileref (file reference name). Other system functions that manipulate external files and directories require that the files be identified by fileref rather than by physical filename. The association between a fileref and a physical file lasts only for the duration of the current SAS session or until you change or discontinue the association by using FILENAME.

You can deassign filerefs by specifying one argument in the FILENAME function, or by passing a blank value for the filename argument.

Operating Environment Information

The term directory in this description refers to an aggregate grouping of files that are managed by the operating environment. Different operating environments identify these groupings with different names, such as directory, subdirectory, folder, MACLIB, or partitioned data set. For details, see the SAS documentation for your operating environment.

Examples

Example 1: Assigning a Fileref to an External File

This example assigns the fileref MYFILE to an external file. Next, it deassigns the fileref. Note that in a macro statement that you do not enclose character strings in quotation marks.

```sas
%let filrf=myfile;
%let rc=%sysfunc(filename(filrf, physical-filename));
%if &rc ne 0 %then
   %put %sysfunc(sysmsg());
%let rc=%sysfunc(filename(filrf));
```

Example 2: Assigning a System-Generated Fileref

This example assigns a system-generated fileref to an external file. The fileref is stored in the variable FNAME. Note that in a macro statement that you do not enclose character strings in quotation marks.

```sas
%let rc=%sysfunc(filename(fname, physical-filename));
%if &rc %then
   %put %sysfunc(sysmsg());
%else
   %do;
      ... more macro statements ...
   %end;
```
Example 3: Assigning a Fileref to a Pipe File
This example assigns the fileref MYPIPE to a pipe file with the output from the UNIX command LS, which lists the files in the directory /u/myid. Note that in a macro statement that you do not enclose character strings in quotation marks.

```sas
%let filrf=mypipe;
%let rc=%sysfunc(filename(filrf, %str(ls /u/myid), pipe));
```

See Also

Functions:
- “FEXIST Function” on page 906
- “FILEEXIST Function” on page 909
- “FILEREF Function” on page 913
- “SYSMSG Function” on page 1074

FILEREF Function
Verifies whether a fileref has been assigned for the current SAS session.

**Category:** External Files

**Restriction:** This function is not valid on the CAS server.

**Syntax**

FILEREF(fileref)

**Required Argument**

fileref

is a character constant, variable, or expression that specifies the fileref to be validated.

**UNIX Specifics**

In a DATA step, the fileref that you specify must be enclosed in double quotation marks. In macro code, fileref must not be enclosed in double quotation marks. You can assign filerefs by using the FILENAME statement or the FILENAME external file access function. Under UNIX, filerefs can also be a UNIX environment variable.

**Range**

1 to 8 characters

**Details**

A negative return code indicates that the fileref exists but the physical file associated with the fileref does not exist. A positive value indicates that the fileref is not assigned. A value of zero indicates that the fileref and external file both exist.

A fileref can be assigned to an external file by using the FILENAME statement or the FILENAME function.
Examples

**Example 1: Verifying That a Fileref Is Assigned**
This example tests whether the fileref MYFILE is currently assigned to an external file. A system error message is issued if the fileref is not currently assigned:

```sas
%if %sysfunc(fileref(myfile))>0 %then
  %put MYFILE is not assigned;
```

**Example 2: Verifying That Both a Fileref and a File Exist**
This example tests for a zero value to determine whether both the fileref and the file exist:

```sas
%if %sysfunc(fileref(myfile)) ne 0 %then
  %put %sysfunc(sysmsg());
```

See Also

Functions:
- “FEXIST Function” on page 906
- “FILEEXIST Function” on page 909
- “FILENAME Function” on page 910
- “SYMSM Function” on page 1074

Statements:
- “FILENAME Statement” in SAS Viya Statements: Reference

---

**FINFO Function**

Returns the value of a file information item.

**Category:** External Files

**Restriction:** This function is not valid on the CAS server.

**Note:** FINFO returns the value of a system-dependent information item for an external file. FINFO returns a blank if the value given for information-item is invalid.

**Syntax**

```
FINFO(file-id, information-item)
```

**Required Arguments**

- **file-id**
  - is a numeric constant, variable, or expression that specifies the identifier that was assigned when the file was opened, generally by the FOPEN function.

- **information-item**
  - is a character constant, variable, or expression that specifies the name of the file information item to be retrieved.
• Create Time: \textit{ddmmyyyy:hh:mm:ss}

\textit{Note:} The Create Time date/time information item is localized to the site's locale. The date/time format might appear slightly different in the locale.

• Last Modified: \textit{ddmmyyyy:hh:mm:ss}

• Filename

• File size (bytes)

• RECFM

• LRECL

\textit{Information-item} for pipe files can be one of these file information items:

• Unnamed pipe access device

• PROCESS

• RECFM

• LRECL

\textit{UNIX Specifics}

\textit{Information-item} for disk files can have one of the following values:

• Filename

• Owner Name

• Group Name

• Access Permission

• Last Modified

• File Size (bytes)

If you concatenate filenames, then an additional information-item, File List, is available.

If you are using pipe files, then the only valid value for information-item is PIPE Command.

\textbf{Comparisons}

• The FOPTNAME function determines the names of the available file information items.

• The FOPTNUM function determines the number of system-dependent information items that are available.

\textbf{Example}

This example stores information items about an external file in a SAS data set:

```sas
data info;
  length infoname infoval $60;
  drop rc fid infonum i close;
  rc=filename('abc', 'physical-filename');
  fid=fopen('abc');
  infonum=foptnum(fid);
  do i=1 to infonum;
    infoname=foptname(fid, i);    
    infoval=finfo(fid, infoname);
  end;
```

\textbf{FINFO Function}
output;
end;
close=fclose(fid);
run;

See Also

Functions:
- “FCLOSE Function” on page 896
- “FOPTNUM Function” on page 931
- “MOPEN Function” on page 996

FINV Function

Returns a quantile from the $F$ distribution.

<table>
<thead>
<tr>
<th>Category:</th>
<th>Quantile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restriction:</td>
<td>This function is not valid on the CAS server.</td>
</tr>
</tbody>
</table>

Syntax

**FINV** ($p$, $ndf$, $ddf <, nc> )$

**Required Arguments**

$p$

is a numeric probability.

Range  $0 \leq p < 1$

$ndf$

is a numeric numerator degrees of freedom parameter.

Range  $ndf > 0$

$ddf$

is a numeric denominator degrees of freedom parameter.

Range  $ddf > 0$

**Optional Argument**

$nc$

is an optional numeric noncentrality parameter.

Range  $nc \geq 0$

**Details**

The FINV function returns the $p$th quantile from the $F$ distribution with numerator degrees of freedom $ndf$, denominator degrees of freedom $ddf$, and noncentrality
parameter $nc$. The probability that an observation from the $F$ distribution is less than the quantile is $p$. This function accepts noninteger degrees of freedom parameters $ndf$ and $ddf$.

If the optional parameter $nc$ is not specified or has the value 0, the quantile from the central $F$ distribution is returned. The noncentrality parameter $nc$ is defined such that if $X$ and $Y$ are normal random variables with means $\mu$ and 0, respectively, and variance 1, then $\frac{X^2}{Y^2}$ has a noncentral $F$ distribution with $nc = \mu^2$.

**CAUTION:**
For large values of $nc$, the algorithm could fail. In that case, a missing value is returned.

*Note:* FINV is the inverse of the PROBF function.

### Example

These statements compute the 95th quantile value of a central $F$ distribution with 2 and 10 degrees of freedom and a noncentral $F$ distribution with 2 and 10.3 degrees of freedom and a noncentrality parameter equal to 2:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_1 = \text{finv}(.95, 2, 10)$;</td>
<td>4.1028210151</td>
</tr>
<tr>
<td>$q_2 = \text{finv}(.95, 2, 10.3, 2)$;</td>
<td>7.583766024</td>
</tr>
</tbody>
</table>

### See Also

**Functions:**
- “QUANTILE Function” on page 549

---

**FIPNAME Function**

Converts two-digit FIPS codes to uppercase state names.

- **Category:** State and ZIP code
- **Restriction:** This function is not valid on the CAS server.

### Syntax

```
FIPNAME(expression)
```

### Required Argument

**expression**

specifies a numeric constant, variable, or expression that represents a U.S. FIPS code.
Details

If the FIPNAME function returns a value to a variable that has not yet been assigned a length, by default the variable is assigned a length of 20.

The FIPNAME function converts a U.S. Federal Information Processing Standards (FIPS) code to the corresponding state or U.S. territory name in uppercase, returning a value of up to 20 characters.

Comparisons

The FIPNAME, FIPNAMEL, and FIPSTATE functions take the same argument but return different values. FIPNAME returns uppercase state names. FIPNAMEL returns mixed case state names. FIPSTATE returns a two-character state postal code (or worldwide GSA geographic code for U.S. territories) in uppercase.

Example

The examples show the differences when using FIPNAME, FIPNAMEL, and FIPSTATE:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=fipname(37); put x;</td>
<td>NORTH CAROLINA</td>
</tr>
<tr>
<td>x=fipnamel(37); put x;</td>
<td>North Carolina</td>
</tr>
<tr>
<td>x=fipstate(37); put x;</td>
<td>NC</td>
</tr>
</tbody>
</table>

See Also

Functions:

- “FIPNAMEL Function” on page 918
- “FIPSTATE Function” on page 920
- “STFIPS Function” on page 1064
- “STNAME Function” on page 1066
- “STNAMEL Function” on page 1067

FIPNAMEL Function

Converts two-digit FIPS codes to mixed case state names.

- **Category:** State and ZIP code
- **Restriction:** This function is not valid on the CAS server.
Syntax

FIPNAMEL(expression)

Required Argument

expression
specifies a numeric constant, variable, or expression that represents a U.S. FIPS code.

Details

If the FIPNAMEL function returns a value to a variable that has not yet been assigned a length, by default the variable is assigned a length of 20.

The FIPNAMEL function converts a U.S. Federal Information Processing Standards (FIPS) code to the corresponding state or U.S. territory name in mixed case, returning a value of up to 20 characters.

Comparisons

The FIPNAME, FIPNAMEL, and FIPSTATE functions take the same argument but return different values. FIPNAME returns uppercase state names. FIPNAMEL returns mixed case state names. FIPSTATE returns a two-character state postal code (or worldwide GSA geographic code for U.S. territories) in uppercase.

Example

The examples show the differences when using FIPNAME, FIPNAMEL, and FIPSTATE:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=fipname(37); put x;</td>
<td>NORTH CAROLINA</td>
</tr>
<tr>
<td>x=fipnamel(37); put x;</td>
<td>North Carolina</td>
</tr>
<tr>
<td>x=fipstate(37); put x;</td>
<td>NC</td>
</tr>
</tbody>
</table>

See Also

Functions:
- “FIPNAME Function” on page 917
- “FIPSTATE Function” on page 920
- “STFIPS Function” on page 1064
- “STNAME Function” on page 1066
- “STNAMEL Function” on page 1067
FIPSTATE Function

Converts two-digit FIPS codes to two-character state postal codes.

**Category:** State and ZIP code

**Restriction:** This function is not valid on the CAS server.

**Syntax**

```
FIPSTATE(expression)
```

**Required Argument**

`expression` specifies a numeric constant, variable, or expression that represents a U.S. FIPS code.

**Details**

If the FIPSTATE function returns a value to a variable that has not yet been assigned a length, by default the variable is assigned a length of 20.

The FIPSTATE function converts a U.S. Federal Information Processing Standards (FIPS) code to a two-character state postal code (or worldwide GSA geographic code for U.S. territories) in uppercase.

**Comparisons**

The FIPNAME, FIPNAMEL, and FIPSTATE functions take the same argument but return different values. FIPNAME returns uppercase state names. FIPNAMEL returns mixed case state names. FIPSTATE returns a two-character state postal code (or worldwide GSA geographic code for U.S. territories) in uppercase.

**Example**

The examples show the differences when using FIPNAME, FIPNAMEL, and FIPSTATE:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=fipname(37); put x;</td>
<td>NORTH CAROLINA</td>
</tr>
<tr>
<td>x=fipnamel(37); put x;</td>
<td>North Carolina</td>
</tr>
<tr>
<td>x=fipstate(37); put x;</td>
<td>NC</td>
</tr>
</tbody>
</table>
FIRST Function

Returns the first character in a character string.

Category:   Character

Restrictions:   This function is assigned an I18N Level 0 status, and is designed for SBCS data. Do not use this function to process DBCS or MBCS data. For more information, see Internationalization Compatibility.

This function is not valid on the CAS server.

Syntax

FIRST(string)

Required Argument

string

specifies a character string.

Details

In a DATA step, the default length of the target variable for the FIRST function is 1. The FIRST function returns a string with a length of 1. If string has a length of 0, then the FIRST function returns a single blank.

Comparisons

The FIRST function returns the same result as CHAR(string, 1) and SUBPAD(string, 1, 1). Although the results are the same, the default length of the target variable is different.

Example

The following example shows the results of using the FIRST function.

data test;
    string1="abc";
    result1=first(string1);
    string2="";
    result2=first(string2);
run;
proc print noobs data=test;
See Also

Functions:
• “CHAR Function” on page 851

FMTINFO Function
Retrieves information about a format or informat.

<table>
<thead>
<tr>
<th>Category:</th>
<th>Special</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restriction:</td>
<td>This function is not valid on the CAS server.</td>
</tr>
</tbody>
</table>

Syntax

FMTINFO(fmtname, info)

Required Arguments

fmtname
  specifies the format or informat name.

info
  specifies the category, type, or description of the format or informat.

Note: You can specify only one info argument in your code.

CAT
  specifies the category:
  • BIDI Text Handling
  • Character
  • Currency Conversion
  • Date and Time
  • DBCS
  • Hebrew Text Handling
  • ISO 8601
Numeric

TYPE
specifies the type of language element:
• informat
• format
• both

DESC
specifies a short description of the format or informat.

MIND
specifies the minimum decimal value of the format or informat.

MAXD
specifies the maximum decimal value of the format or informat.

DEFD
specifies the default decimal value of the format or informat.

MINW
specifies the minimum width value of the format or informat.

MAXW
specifies the maximum width value of the format or informat.

DEFW
specifies the default width value of the format or informat.

Details
The FMTINFO function retrieves information about a format or informat. You can retrieve information about a format or informat’s category, the type of language element (format, informat, or both), a description of the language element, and the minimum, maximum, and default decimal and width values.

You cannot specify multiple arguments with the FMTINFO function. You can specify only one of these arguments: CAT, TYPE, DESC, MIND, MAXD, DEFD, MINW, MAXW, and DEFW. For example,

```
a=fmtinfo('best','cat');
```

is valid because you specify only one argument, the category argument `cat`.

You must enclose the arguments with apostrophes unless you are passing a character variable name.

The FMTINFO function returns a character string for all data values, including these numeric value arguments: MIND, MAXD, DEFD, MINW, MAXW, and DEFW.

Example
The following examples retrieve information about the BEST format.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>data; a=num</td>
<td></td>
</tr>
<tr>
<td>Statement</td>
<td>Result</td>
</tr>
<tr>
<td>-----------</td>
<td>--------</td>
</tr>
<tr>
<td>data; a=fmtinfo('best','type'); put a=; run;</td>
<td>a=both</td>
</tr>
<tr>
<td>data; a=fmtinfo('best','desc'); put a=; run;</td>
<td>a=SAS System chooses best notation</td>
</tr>
</tbody>
</table>

**FNOTE Function**

Identifies the last record that was read, and returns a value that the FPOINT function can use.

**Category:** External Files

**Restriction:** This function is not valid on the CAS server.

**Syntax**

FNOTE(file-id)

**Required Argument**

file-id

is a numeric constant, variable, or expression that specifies the identifier that was assigned when the file was opened, generally by the FOPEN function.

**Details**

You can use FNOTE like a bookmark, marking the position in the file so that your application can later return to that position using FPOINT. The value that is returned by FNOTE is required by the FPOINT function to reposition the file pointer on a specific record.

To free the memory associated with each FNOTE identifier, use DROPNOTE.

**Note:** You cannot write a new record in place of the current record if the new record has a length that is greater than the current record.

**Example**

This example assigns the fileref MYFILE to an external file and attempts to open the file. If the file is opened successfully, indicated by a positive value in the variable FID, then it reads the records, stores in the variable NOTE 3 the position of the third record read, and then later uses FPOINT to point back to NOTE3 to update the file. After updating the record, it closes the file:

```
%let fref=MYFILE;
%let rc=%sysfunc(filename(fref, physical-filename));
%let fid=%sysfunc(fopen(&fref, u));
```
%if &fid > 0 %then
%do;
    %let rc=%sysfunc(fread(&fid));
    /* Read second record. */
    %let rc=%sysfunc(fread(&fid));
    /* Read third record. */
    %let rc=%sysfunc(fread(&fid));
    /* Note position of third record. */
    %let note3=%sysfunc(fnote(&fid));
    /* Read fourth record. */
    %let rc=%sysfunc(fread(&fid));
    /* Read fifth record. */
    %let rc=%sysfunc(fread(&fid));
    /* Point to third record. */
    %let rc=%sysfunc(fpoint(&fid,&note3));
    /* Read third record. */
    %let rc=%sysfunc(fread(&fid));
    /* Copy new text to FDB. */
    %let rc=%sysfunc(fput(&fid, New text));
    /* Update third record */
    /* with data in FDB. */
    %let rc=%sysfunc(fwrite(&fid));
    /* Close file. */
    %let rc=%sysfunc(fclose(&fid));
%end;
%let rc=%sysfunc(filename(fref));

See Also

Functions:

- “DROPNOTE Function” on page 888
- “FCLOSE Function” on page 896
- “FILENAME Function” on page 910
- “FOPEN Function” on page 925
- “FPOINT Function” on page 932
- “FPUT Function” on page 936
- “FREAD Function” on page 937
- “FREIND Function” on page 938
- “FWRITE Function” on page 942
- “MOPEN Function” on page 996

FOPEN Function

Opens an external file and returns a file identifier value.

Category: External Files

Restriction: This function is not valid on the CAS server.
Syntax

FOPEN(fileref <, open-mode <, record-length <, record-format>>>)

Required Argument

fileref

is a character constant, variable, or expression that specifies the fileref assigned to the external file.

Note: You cannot create a fileref that is greater than eight characters in SAS, although an environment variable that is used as a fileref can be greater than eight characters.

Optional Arguments

open-mode

is a character constant, variable, or expression that specifies the type of access to the file:

A  APPEND mode allows writing new records after the current end of the file.
I  INPUT mode allows reading only (default).
O  OUTPUT mode defaults to the OPEN mode specified in the operating environment option in the FILENAME statement or function. If no operating environment option is specified, it allows writing new records at the beginning of the file.
S  Sequential input mode is used for pipes and other sequential devices such as hardware ports.
U  UPDATE mode allows both reading and writing.

Default I

record-length

is a numeric constant, variable, or expression that specifies the logical record length of the file. To use the existing record length for the file, specify a length of 0, or do not provide a value here.

record-format

is a character constant, variable, or expression that specifies the record format of the file. To use the existing record format, do not specify a value here. Valid values are:

B  data are to be interpreted as binary data.
D  use default record format.
E  use editable record format.
F  file contains fixed length records.
P  file contains printer carriage control in operating environment-dependent record format.
V  file contains variable length records.

Note: If an argument is invalid, FOPEN returns 0, and you can obtain the text of the corresponding error message from the SYMSG function. Invalid arguments do
not produce a message in the SAS log and do not set the _ERROR_ automatic variable.

Details

General Information

CAUTION:

Use OUTPUT mode with care. Opening an existing file for output overwrites the current contents of the file without warning.

The FOPEN function opens an external file for reading or updating and returns a file identifier value that is used to identify the open file to other functions. You must associate a fileref with the external file before calling the FOPEN function. FOPEN returns a 0 if the file could not be opened. You can assign filerefs by using the FILENAME statement or the FILENAME external file access function. Under some operating environments, you can also assign filerefs by using system commands.

If you call the FOPEN function from a macro, the result of the call is valid only when it is passed to functions in a macro. If you call the FOPEN function from the DATA step, the result is valid only when it is passed to functions in the same DATA step.

Operating Environment Information

It is good practice to use the FCLOSE function at the end of a DATA step if you used FOPEN to open the file, even though using FCLOSE might not be required in your operating environment. For more information about FOPEN, see the SAS documentation for your operating environment.

Examples

Example 1: Opening a File Using Defaults

This example assigns the fileref MYFILE to an external file and attempts to open the file for input using all defaults. Note that in a macro statement that you do not enclose character strings in quotation marks.

```sas
%let filrf=myfile;
%let rc=%sysfunc(filename(filrf, physical-filename));
%let fid=%sysfunc(fopen(&filrf));
```

Example 2: Opening a File without Using Defaults

This example attempts to open a file for input without using defaults. Note that in a macro statement that you do not enclose character strings in quotation marks.

```sas
%let fid=%sysfunc(fopen(file2, o, 132, e));
```

Example 3: Handling Errors

This example shows how to check for errors and write an error message from the SYSMSG function.

```sas
data _null_
   f=fopen('bad', '?');
   if not f then do;
      m=sysmsg();
      put m;
      abort;
   end;
```
See Also

Functions:
- “DOPEN Function” on page 881
- “FCLOSE Function” on page 896
- “FILENAME Function” on page 910
- “FILEREF Function” on page 913
- “MOPEN Function” on page 996
- “SYMS Function” on page 1074

Statements:
- “FILENAME Statement” in SAS Viya Statements: Reference

FOPTNAME Function

Returns the name of an item of information about an external file.

**Category:** External Files

**Restriction:** This function is not valid on the CAS server.

**Syntax**

FOPTNAME(file-id, nval)

**Required Arguments**

*file-id*  
A numeric variable that specifies the identifier that was assigned when the file was opened, generally by the FOPEN function.

*nval*  
A numeric constant, variable, or expression that specifies the number of the information item.

**Details**

**General Information**

FOPTNAME returns a missing or null value if an invalid argument to FOPTNAME is used.
UNIX: Available Information Items
The following table shows the information items that correspond to the nval values for single and concatenated files under UNIX operating environments.

Table 3.1  Information Items for Files under UNIX

<table>
<thead>
<tr>
<th>nval</th>
<th>Single File</th>
<th>Pipe Files</th>
<th>Concatenated Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Filename</td>
<td>PIPE command</td>
<td>Filename</td>
</tr>
<tr>
<td>2</td>
<td>Owner name</td>
<td>File list</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Group name</td>
<td>Owner name</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Access permission</td>
<td>Group name</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>File size (bytes)</td>
<td>Access permissions</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>File size (bytes)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Examples

Example 1: Retrieving File Information Items and Writing Them to the Log
This example retrieves the system-dependent file information items that are available and writes them to the log:

```plaintext
%let filrf=myfile;
%let rc=%sysfunc(filename(filrf, physical-filename));
%let fid=%sysfunc(fopen(&filrf));
%let infonum=%sysfunc(foptnum(&fid));
%do j=1 %to &infonum;
   %let name=%sysfunc(foptname(&fid, &j));
   %let value=%sysfunc(finfo(&fid, &name));
   %put File attribute &name equals &value;
%end;
%let rc=%sysfunc(fclose(&fid));
%let rc=%sysfunc(filename(filrf));
```

Example 2: Creating a Data Set with Names and Values of File Attributes
This example creates a data set that contains the name and value of the available file attributes:

```plaintext
data fileatt;
  length name $ 20 value $ 40;
drop rc fid j infonum;
rc=filename("myfile", "physical-filename");
fid=fopen("myfile");
infonum=foptnum(fid);
do j=1 to infonum;
   name=foptname(fid, j);
Example 3: File Attributes When Using the Pipe Device Type under UNIX

The following example creates a data set that contains the NAME and VALUE attributes returned by the FOPTNAME function when you are using pipes under UNIX:

```sas
data fileatt;
  length name $ 20 value $ 40;
  drop fid j infonum;
  filename mypipe pipe 'UNIX-command';
  fid=fopen("mypipe", "s");
  infonum=foptnum(fid);
  do j=1 to infonum;
    name=foptname(fid, j);
    value=finfo(fid, name);
    put 'File attribute' name 'has a value of ' value;
    output;
  end;
  rc=filename("myfile");
run;
```

The following statement appears in the SAS log.

**Log 3.1  SAS Log**

```
File attribute Pipe Command has a value of UNIX-command
```

*Unix-command* is the UNIX command or program where you are piping your output or where you are reading your input. This command or program must be either fully qualified or defined in your PATH environment variable.

**See Also**

**Functions:**

- “DINFO Function” on page 877
- “DOPTNAME Function” on page 882
- “DOPTNUM Function” on page 884
- “FCLOSE Function” on page 896
- “FILENAME Function” on page 910
- “FINFO Function” on page 914
- “FOPEN Function” on page 925
- “FOPTNUM Function” on page 931
- “MOPEN Function” on page 996
FOPTNUM Function

Returns the number of information items, such as filename or record length, that are available for an external file.

**Category:** External Files

**Restriction:** This function is not valid on the CAS server.

**Syntax**

FOPTNUM(file-id)

**Required Argument**

file-id

is the identifier that was assigned when the file was opened, generally by the FOPEN function.

**Details**

**General Information**

The number of information items that are available for a file depend on the operating environment and the type of file.

**FOPTNUM Specifics under UNIX**

Under UNIX, five information items are available for all types of files:

- Filename
- Owner Name
- Group Name
- Access Permission
- File Size (bytes)

If you concatenate filenames, then an additional information item–File List–is available. If you are using piped files, then the only information item that is available is the PIPE Command.

The open-mode specified in the FOPEN function determines the value that FOPTNUM returns.

**Table 3.2 Open Mode and FOPTNUM Values**

<table>
<thead>
<tr>
<th>Open Mode</th>
<th>FOPTNUM Value</th>
<th>Information Items Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>Append</td>
<td>6 for concatenated</td>
<td>All information items available.</td>
</tr>
<tr>
<td></td>
<td>files</td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>5 for single files</td>
<td></td>
</tr>
<tr>
<td>Update</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Open Mode

<table>
<thead>
<tr>
<th>Open Mode</th>
<th>FOPTNUM Value</th>
<th>Information Items Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>5 for concatenated files</td>
<td>Because the file is open for output, the File Size information type is unavailable.</td>
</tr>
<tr>
<td></td>
<td>4 for single files</td>
<td></td>
</tr>
<tr>
<td>Sequential</td>
<td>1</td>
<td>The only information item available is PIPE Command.</td>
</tr>
<tr>
<td>(using Pipe Device Type)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For an example of how to use the FOPTNUM function, see “Example 3: File Attributes When Using the Pipe Device Type under UNIX” on page 930.

### Comparisons

- Use FOPTNAME to determine the names of the items that are available for a particular operating environment.
- Use FINFO to retrieve the value of a particular information item.

### Example

This example opens the external file with the fileref MYFILE and determines the number of system-dependent file information items available:

```sas
%let fid=%sysfunc(fopen(myfile));
%let infonum=%sysfunc(foptnum(&fid));
```

### See Also

**Functions:**
- “DINFO Function” on page 877
- “DOPTNAME Function” on page 882
- “DOPTNUM Function” on page 884
- “FINFO Function” on page 914
- “FOPEN Function” on page 925
- “FOPTNAME Function” on page 928
- “MOPEN Function” on page 996

---

### FPOINT Function

Positions the read pointer on the next record to be read.

**Category:** External Files

**Restriction:** This function is not valid on the CAS server.
Syntax

FPOINT(file-id, note-id)

Required Arguments

file-id

is a numeric variable that specifies the identifier that was assigned when the file was opened, generally by the FOPEN function.

note-id

specifies the identifier that was assigned by the FNOTE function.

Details

FPOINT returns 0 if the operation was successful, or ≠0 if it was not successful. FPOINT determines only the record to read next. It has no impact on which record is written next. When you open the file for update, FWRITE writes to the most recently read record.

Note: You cannot write a new record in place of the current record if the new record has a length that is greater than the current record.

Example

This example assigns the fileref MYFILE to an external file and attempts to open the file. If the file is opened successfully, it reads the records and uses NOTE3 to store the position of the third record read. Later, it points back to NOTE3 to update the file, and closes the file afterward:

```
%let filrf=myfile;
%let rc=%sysfunc(filename(filrf, physical-filename));
%let fid=%sysfunc(fopen(&filrf, u));
%if &fid > 0 %then
  %do;
    /* Read first record. */
    %let rc=%sysfunc(fread(&fid));
    /* Read second record. */
    %let rc=%sysfunc(fread(&fid));
    /* Read third record. */
    %let rc=%sysfunc(fread(&fid));
    /* Note position of third record. */
    %let note3=%sysfunc(fnote(&fid));
    /* Read fourth record. */
    %let rc=%sysfunc(fread(&fid));
    /* Read fifth record. */
    %let rc=%sysfunc(fread(&fid));
    /* Point to third record. */
    %let rc=%sysfunc(fpoint(&fid, &note3));
    /* Read third record. */
    %let rc=%sysfunc(fread(&fid));
    /* Copy new text to FDB. */
    %let rc=%sysfunc(fput(&fid, New text));
    /* Update third record */
    /* with data in FDB. */
    %let rc=%sysfunc(fwrite(&fid));
  %end;
%end;
```
%let rc=%sysfunc(fclose(&fid));
%end;
%let rc=%sysfunc(filename(filrf));

See Also

Functions:

• “DROPNOTE Function” on page 888
• “FCLOSE Function” on page 896
• “FILENAME Function” on page 910
• “FNOTE Function” on page 924
• “FOPEN Function” on page 925
• “FPUT Function” on page 936
• “FREAD Function” on page 937
• “FREWIND Function” on page 938
• “FWRITE Function” on page 942
• “MOPEN Function” on page 996

FPOS Function

Sets the position of the column pointer in the File Data Buffer (FDB).

**Category:** External Files

**Restriction:** This function is not valid on the CAS server.

**Syntax**

FPOS(file-id, nval)

**Required Arguments**

- **file-id**
  
  is a numeric variable that specifies the identifier that was assigned when the file was opened, generally by the FOPEN function.

- **nval**
  
  is a numeric constant, variable, or expression that specifies the column at which to set the pointer.

**Details**

FPOS returns 0 if the operation was successful, ≠0 if it was not successful. If you open a file in output mode and the specified position is past the end of the current record, the size of the record is increased appropriately. However, in a fixed block or VBA file, if you specify a column position beyond the end of the record, the record size does not change and the text string is not written to the file.

If you open a file in Update mode and the specified position is not past the end of the current record, then SAS writes the record to the file. If the specified position is past the
end of the current record, then SAS returns an error message and does not write the new record:

ERROR: Cannot increase record length in update mode.

Note: If you use the Update mode with the FOPEN function, then you must execute FREAD before you execute FWRITE functions.

Example

This example assigns the fileref MYFILE to an external file and opens the file in Update mode. If the file is opened successfully, indicated by a positive value in the variable FID, SAS reads a record and places data into the file's buffer at column 12. If the resulting record length is less than or equal to the original record length, then SAS writes the record and closes the file. If the resulting record length is greater than the original record length, then SAS writes an error message to the log.

```sas
%macro ptest;
%let filrf=myfile;
%let rc=%sysfunc(filename(filrf, test.txt));
%let fid=%sysfunc(fopen(&filrf, o));
%let rc=%sysfunc(fread(&fid));
%put &fid;
%if (&fid > 0) %then
  %do;
    %let dataline=This is some data.; /* Position at column 12 in the FDB. */
    %let rc=%sysfunc(fpos(&fid, 12));
    %put &rc one;
    /* Put the data in the FDB. */
    %let rc=%sysfunc(fput(&fid, &dataline));
    %put &rc two;
    %if (&rc ne 0) %then
      %do;
        %put %sysfunc(sysmsg());
      %end;
    %else %do;
      /* Write the record. */
      %let rc=%sysfunc(fwrite(&fid));
      %if (&rc ne 0) %then
        %do;
          %put write fails &rc;
        %end;
      %end;
      /* Close the file. */
      %let rc=%sysfunc(fclose(&fid));
    %end;
%end;
%let rc=%sysfunc(filename(filrf));
%mend;
%ptest;
```

SAS writes the following output to the log:

```
1
0 one
0 two
```
FPUT Function

Moves data to the File Data Buffer (FDB) of an external file, starting at the FDB’s current column position.

**Category:** External Files

**Restriction:** This function is not valid on the CAS server.

### Syntax

\[
\text{FPUT}(\text{file-id, cval})
\]

### Required Arguments

- **file-id**
  - is a numeric variable that specifies the identifier that was assigned when the file was opened, generally by the FOPEN function.

- **cval**
  - is a character constant, variable, or expression that specifies the file data.

### Details

FPUT returns 0 if the operation was successful, ≠0 if it was not successful. The number of bytes moved to the FDB is determined by the length of the variable. The value of the column pointer is then increased to one position past the end of the new text.

**Note:** You cannot write a new record in place of the current record if the new record has a length that is greater than the current record.

### Example

This example assigns the fileref MYFILE to an external file and attempts to open the file in APPEND mode. If the file is opened successfully, indicated by a positive value in the variable FID, it moves data to the FDB using FPUT, appends a record using FWRITE, and then closes the file. Note that in a macro statement that you do not enclose character strings in quotation marks.

```
%macro ptest;
  %let filref=myfile;
```

See Also

**Functions:**

- “FCLOSE Function” on page 896
- “FCOL Function” on page 897
- “FILENAME Function” on page 910
- “FOPEN Function” on page 925
- “FPUT Function” on page 936
- “FWRITE Function” on page 942
- “MOPEN Function” on page 996
%let rc=%sysfunc(filename(filrf, test.txt));
%let fid=%sysfunc(fopen(&filrf, a));
%if &fid > 0 %then
  %do;
    %let rc=%sysfunc(fread(&fid));
    %let mystring=This is some data.;
    %let rc=%sysfunc(fput(&fid, &mystring));
    %let rc=%sysfunc(fwrite(&fid));
    %let rc=%sysfunc(fclose(&fid));
  %end;
%else
  %put %sysfunc(sysmsg());
%let rc=%sysfunc(filename(filrf));
%put return code=&rc;
%mend;
ptest;

SAS writes the following output to the log:

| return code = 0 |

See Also

Functions:
- “FCLOSE Function” on page 896
- “FILENAME Function” on page 910
- “FNOTE Function” on page 924
- “FOPEN Function” on page 925
- “FPOINT Function” on page 932
- “FPOS Function” on page 934
- “FWRITE Function” on page 942
- “MOPEN Function” on page 996
- “SYSMSG Function” on page 1074

---

**FREAD Function**

Reads a record from an external file into the File Data Buffer (FDB).

**Category:** External Files

**Restriction:** This function is not valid on the CAS server.

**Syntax**

\[
\text{FREAD}(\text{file-}id)
\]
**Required Argument**

*file-id*

is a numeric variable that specifies the identifier that was assigned when the file was opened, generally by the FOPEN function.

**Details**

FREAD returns 0 if the operation was successful, ≠0 if it was not successful. The position of the file pointer is updated automatically after the Read operation so that successive FREAD functions read successive file records.

To position the file pointer explicitly, use FNOTE, FPOINT, and FREWIND.

**Example**

This example assigns the fileref MYFILE to an external file and attempts to open the file. If the file opens successfully, it lists all of the file's records in the log:

```sas
%let filrf=myfile;
%let rc=%sysfunc(filename(filrf, physical-filename));
%let fid=%sysfunc(fopen(&filrf));
%if &fid > 0 %then
   %do %while(%sysfunc(fread(&fid))=0);
      %let rc=%sysfunc(fget(&fid, c, 200));
      %put &c;
   %end;
%let rc=%sysfunc(fclose(&fid));
%let rc=%sysfunc(filename(filrf));
```

**See Also**

**Functions:**

- “FCLOSE Function” on page 896
- “FGET Function” on page 907
- “FILENAME Function” on page 910
- “FNOTE Function” on page 924
- “FOPEN Function” on page 925
- “FPOINT Function” on page 932
- “FREWIND Function” on page 938
- “MOPEN Function” on page 996

---

**FREWIND Function**

Positions the file pointer to the start of the file.

- **Category:** External Files
- **Restriction:** This function is not valid on the CAS server.
Syntax
FREWIND(file-id)

Required Argument
file-id

is a numeric variable that specifies the identifier that was assigned when the file was opened, generally by the FOPEN function.

Details
FREWIND returns 0 if the operation was successful, ≠0 if it was not successful. FREWIND has no effect on a file opened with sequential access.

Example
This example assigns the fileref MYFILE to an external file. Then it opens the file and reads the records until the end of the file is reached. The FREWIND function then repositions the pointer to the beginning of the file. The first record is read again and stored in the File Data Buffer (FDB). The first token is retrieved and stored in the macro variable VAL:

```
%let filrf=myfile;
%let rc=%sysfunc(filename(filrf, physical-filename));
%let fid=%sysfunc(fopen(&filrf));
%let rc=0;
%do %while (&rc ne -1);
  /* Read a record. */
  %let rc=%sysfunc(fread(&fid));
%end;
  /* Reposition pointer to beginning of file. */
%if &rc = -1 %then
  %do;
    %let rc=%sysfunc(frewind(&fid));
    /* Read first record. */
    %let rc=%sysfunc(fread(&fid));
    /* Read first token */
    /* into macro variable VAL. */
    %let rc=%sysfunc(fget(&fid, val));
    %put val=&val;
  %end;
%else
  %put Error on fread=%sysfunc(sysmsg());
%let rc=%sysfunc(fclose(&fid));
%let rc=%sysfunc(filename(filrf));
```

See Also
Functions:
- “FCLOSE Function” on page 896
- “FGET Function” on page 907
- “FILENAME Function” on page 910
- “FOPEN Function” on page 925
FRLEN Function

Returns the size of the last record that was read, or, if the file is opened for output, returns the current record size.

**Category:** External Files

**Restriction:** This function is not valid on the CAS server.

### Syntax

FRLEN(file-id)

### Required Argument

**file-id**

is a numeric variable that specifies the identifier that was assigned when the file was opened, generally by the FOPEN function.

### Example

This example opens the file that is identified by the fileref MYFILE. It determines the minimum and maximum length of records in the external file and writes the results to the log:

```sas
%let fid=%sysfunc(fopen(myfile));
%let min=0;
%let max=0;
%if (%sysfunc(fread(&fid))=0) %then %do;
  %let min=%sysfunc(frlen(&fid));
  %let max=%min;
%do %while(%sysfunc(fread(&fid))=0);
  %let reclen=%sysfunc(frlen(&fid));
  %if (&reclen > &max) %then
    %let max=&reclen;
  %if (&reclen < &min) %then
    %let min=&reclen;
%end;
%end;
%let rc=%sysfunc(fclose(&fid));
%put max=&max min=&min;
```

### See Also

**Functions:**

- “FCLOSE Function” on page 896
FSEP Function
Sets the token delimiters for the FGET function.

**Category:** External Files

**Restriction:** This function is not valid on the CAS server.

### Syntax

\[
\text{FSEP}(file-id, \text{character(s)} < ',x' | 'X'>)
\]

### Required Arguments

- **file-id**
  - is a numeric variable that specifies the identifier that was assigned when the file was opened, generally by the FOPEN function.

- **character**
  - is a character constant, variable, or expression that specifies one or more delimiters that separate items in the File Data Buffer (FDB). Each character listed is a delimiter. That is, if `character` is `#@`, either `#` or `@` can separate items. Multiple consecutive delimiters, such as `@#@`, are treated as a single delimiter.

  **Default** blank

### Optional Argument

- `'x' | 'X'`
  - specifies that the character delimiter is a hexadecimal value.

  **Restrictions**
  - `'x' and 'X' are the only valid values for this argument. All other values will cause an error to occur.

  If you pass `'x' or 'X' as the third argument, a valid hexadecimal string must be passed as the second argument, `character`. Otherwise, the function will fail. A valid hexadecimal string is an even number of 0–9 and A–F characters.

  **Tip**
  - If you use a macro statement, then quotation marks enclosing x or X are not required.

### Details

FSEP returns 0 if the operation was successful, ≠0 if it was not successful.

### Example

An external file has data in this form:
John J. Doe, Male, 25, Weight Lifter
Pat O’Neal, Female, 22, Gymnast

Note that each field is separated by a comma.

This example reads the file that is identified by the fileref MYFILE, using the comma as a separator, and writes the values for NAME, GENDER, AGE, and WORK to the SAS log. Note that in a macro statement that you do not enclose character strings in quotation marks, but a literal comma in a function argument must be enclosed in a macro quoting function such as %STR.

```sas
%let fid=%sysfunc(fopen(myfile));
%let rc=%sysfunc(fsep(&fid,%str(,)));       %do %while(%sysfunc(fread(&fid))=0);
%let rc=%sysfunc(fget(&fid, name));        %let rc=%sysfunc(fget(&fid, gender));
%let rc=%sysfunc(fget(&fid, age));         %let rc=%sysfunc(fget(&fid, work));
    %put name=%bquote(&name) gender=&gender
        age=&age work=&work;
%end;
%let rc=%sysfunc(fclose(&fid));
```

See Also

Functions:
- “FCLOSE Function” on page 896
- “FGET Function” on page 907
- “FOPEN Function” on page 925
- “FREAD Function” on page 937
- “MOPEN Function” on page 996

FWRITE Function

Writes a record to an external file.

**Category:** External Files

**Restriction:** This function is not valid on the CAS server.

**Syntax**

`FWRITE(file-id <, cc>)`

**Required Argument**

`file-id`

is a numeric variable that specifies the identifier that was assigned when the file was opened, generally by the FOPEN function.
**Optional Argument**

*cc*

is a character constant, variable, or expression that specifies a carriage-control character:

- **blank** starts the record on a new line.
- **0** skips one blank line before a new line.
- **-** skips two blank lines before a new line.
- **1** starts the line on a new page.
- **+** overstrikes the line on a previous line.
- **P** interprets the line as a computer prompt.
- **=** interprets the line as carriage control information.
- **all else** starts the line record on a new line.

**Details**

`FWRITE` returns 0 if the operation was successful, ≠0 if it was not successful. `FWRITE` moves text from the File Data Buffer (FDB) to the external file. In order to use the carriage-control characters, you must open the file with a record format of `P` (print format) in `FOPEN`.

*Note:* When you use the Update mode, you must execute `FREAD` before you execute `FWRITE`. You cannot write a new record in place of the current record if the new record has a length that is greater than the current record.

**Example**

This example assigns the fileref MYFILE to an external file and attempts to open the file. If the file is opened successfully, it writes the numbers 1 to 50 to the external file, skipping two blank lines. Note that in a macro statement that you do not enclose character strings in quotation marks.

```plaintext
%let filref=myfile;
%let rc=%sysfunc(filename(filref, physical-filename));
%let fid=%sysfunc(fopen(&filref, o, 0, P));
%do i=1 %to 50;
  %let rc=%sysfunc(fput(&fid,
    %sysfunc(putn(&i,2.))));
  %if (%sysfunc(fwrite(&fid,-)) ne 0) %then
    %put %sysfunc(sysmsg());
%end;
%let rc=%sysfunc(fclose(&fid));
```

**See Also**

**Functions:**
- “FAPPEND Function” on page 895
- “FCLOSE Function” on page 896
- “FGET Function” on page 907
- “FILENAME Function” on page 910
GETVARC Function

Returns the value of a SAS data set character variable.

Category: SAS File I/O
Restriction: This function is not valid on the CAS server.

Syntax

GETVARC(data-set-id, variable-number)

Required Arguments

data-set-id
is a numeric constant, variable, or expression that specifies the data set identifier that
the OPEN function returns.

variable-number
is a numeric constant, variable, or expression that specifies the number of the
variable in the Data Set Data Vector (DDV).

Tips
You can obtain this value by using the VARNUM function.

This value is listed next to the variable when you use the CONTENTS
procedure.

Details

Use VARNUM to obtain the number of a variable in a SAS data set. VARNUM can be
nested or it can be assigned to a variable that can then be passed as the second argument,
as shown in the following examples. GETVARC reads the value of a character variable
from the current observation in the Data Set Data Vector (DDV) into a macro or DATA
step variable.

Example

This example opens the Sasuser.Houses data set and gets the entire tenth observation.
The data set identifier value for the open data set is stored in the macro variable
MYDATAID. This example nests VARNUM to return the position of the variable in
the DDV, and reads in the value of the character variable STYLE.

```sas
%let mydataid=%sysfunc(open(sasuser.houses, i));
%let rc=%sysfunc(fetchobs(&mydataid, 10));
%let style=%sysfunc(getvarc(&mydataid, &mydataid, &mydataid, style));
%let rc=%sysfunc(close(&mydataid));
```
This example assigns VARNUM to a variable that can then be passed as the second argument. This example fetches data from observation 10.

```sas
%let namenum=%sysfunc(varnum(&mydataid, NAME));
%let rc=%sysfunc(fetchobs(&mydataid, 10));
%let user=%sysfunc(getvarc(&mydataid, &namenum));
```

See Also

Functions:
- “FETCH Function” on page 903
- “FETCHOBS Function” on page 905
- “GETVARN Function” on page 945
- “VARNUM Function” on page 1095

---

**GETVARN Function**

Returns the value of a SAS data set numeric variable.

<table>
<thead>
<tr>
<th>Category:</th>
<th>SAS File I/O</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Restriction:</strong></td>
<td>This function is not valid on the CAS server.</td>
</tr>
</tbody>
</table>

**Syntax**

`GETVARN(data-set-id, variable-number)`

**Required Arguments**

- **data-set-id**
  is a numeric constant, variable, or expression that specifies the data set identifier that the OPEN function returns.

- **variable-number**
  is a numeric constant, variable, or expression that specifies the number of the variable in the Data Set Data Vector (DDV).

**Tips**

You can obtain this value by using the VARNUM function.

This value is listed next to the variable when you use the CONTENTS procedure.

**Details**

Use VARNUM to obtain the number of a variable in a SAS data set. You can nest VARNUM or you can assign it to a variable that can then be passed as the second argument, as shown in the "Examples" section. GETVARN reads the value of a numeric variable from the current observation in the Data Set Data Vector (DDV) into a macro variable or DATA step variable.
Example

- This example obtains the entire tenth observation from a SAS data set. The data set must have been previously opened using OPEN. The data set identifier value for the open data set is stored in the variable MYDATAID. This example nests VARNUM, and reads in the value of the numeric variable PRICE from the tenth observation of an open SAS data set.

```sas
%let rc=%sysfunc(fetchobs(&mydataid, 10));
%let price=%sysfunc(getvarn(&mydataid, %sysfunc(varnum(&mydataid, price))));
```

- This example assigns VARNUM to a variable that can then be passed as the second argument. This example fetches data from observation 10.

```sas
%let pricenum=%sysfunc(varnum(&mydataid, price));
%let rc=%sysfunc(fetchobs(&mydataid, 10));
%let price=%sysfunc(getvarn(&mydataid, &pricenum));
```

See Also

Functions:
- “FETCH Function” on page 903
- “FETCHOBS Function” on page 905
- “GETVARC Function” on page 944
- “VARNUM Function” on page 1095

GRAYCODE Function

Generates all subsets of $n$ items in a minimal change order.

**Category:** Combinatorial

**Restrictions:** This function is not valid on the CAS server. The GRAYCODE function cannot be executed when you use the %SYSFUNC macro.

**Syntax**

```
GRAYCODE($k$, numeric-variable-1, ..., numeric-variable-$n$)
GRAYCODE($k$, character-variable $<, n <$, in-out$>\rangle$)
```

**Required Arguments**

$k$

specifies a numeric variable. Initialize $k$ to either of the following values before executing the GRAYCODE function:

- a negative number to cause GRAYCODE to initialize the subset to be empty
• the number of items in the initial set indicated by numeric-variable-1 through numeric-variable-n, or character-variable, which must be an integer value between 0 and n inclusive.

The value of k is updated when GRAYCODE is executed. The value that is returned is the number of items in the subset.

**numeric-variable**

specifies numeric variables that have values of 0 or 1 which are updated when GRAYCODE is executed. A value of 1 for numeric-variable-j indicates that the jth item is in the subset. A value of 0 for numeric-variable-j indicates that the jth item is not in the subset.

If you assign a negative value to k before you execute GRAYCODE, then you do not need to initialize numeric-variable-1 through numeric-variable-n before executing GRAYCODE unless you want to suppress the note about uninitialized variables.

If you assign a value between 0 and n inclusive to k before you execute GRAYCODE, then you must initialize numeric-variable-1 through numeric-variable-n to k values of 1 and n-k values of 0.

**character-variable**

specifies a character variable that has a length of at least n characters. The first n characters indicate which items are in the subset. By default, an "I" in the jth position indicates that the jth item is in the subset, and an "O" in the jth position indicates that the jth item is out of the subset. You can change the two characters by specifying the in-out argument.

If you assign a negative value to k before you execute GRAYCODE, then you do not need to initialize character-variable before executing GRAYCODE unless you want to suppress the note about an uninitialized variable.

If you assign a value between 0 and n inclusive to k before you execute GRAYCODE, then you must initialize character-variable to k characters that indicate an item is in the subset, and n-k characters that indicate an item is out of the subset.

### Optional Arguments

**n**

specifies a numeric constant, variable, or expression. By default, n is the length of character-variable.

**in-out**

specifies a character constant, variable, or expression. The default value is "IO." The first character is used to indicate that an item is in the subset. The second character is used to indicate that an item is out of the subset.

### Details

When you execute GRAYCODE with a negative value of k, the subset is initialized to be empty. The GRAYCODE function returns zero.

When you execute GRAYCODE with an integer value of k between 0 and n inclusive, one item is either added to the subset or removed from the subset, and the value of k is updated to equal the number of items in the subset. If the jth item is added to the subset or removed from the subset, the GRAYCODE function returns j.

To generate all subsets of n items, you can initialize k to a negative value and execute GRAYCODE in a loop that iterates 2**n times. If you want to start with a non-empty subset, then initialize k to be the number of items in the subset, initialize the other
arguments to specify the desired initial subset, and execute GRAYCODE in a loop that
iterates $2^{n}-1$ times. The sequence of subsets that are generated by GRAYCODE is
cyclical, so you can begin with any subset that you want.

Examples

Example 1: Using $n=4$ Numeric Variables and Negative Initial $k$
The following program uses numeric variables to generate subsets in a minimal change
order.

```sas
data _null_;  
array x[4];  
n=dim(x);  
k=-1;  
nsubs=2**n;  
do i=1 to nsubs;  
   rc=graycode(k, of x[*]);  
   put i 5. +3 k= ' x=' x[*] +3 rc=;  
end;  
run;
```

SAS writes the following output to the log:

```
1 k=0 x=0 0 0 0 rc=0
2 k=1 x=1 0 0 0 rc=1
3 k=2 x=1 1 0 0 rc=2
4 k=1 x=0 1 0 0 rc=1
5 k=2 x=0 1 1 0 rc=3
6 k=3 x=1 1 1 0 rc=1
7 k=2 x=1 0 1 0 rc=2
8 k=1 x=0 0 1 0 rc=1
9 k=2 x=0 0 1 1 rc=4
10 k=3 x=1 0 1 1 rc=1
11 k=4 x=1 1 1 1 rc=2
12 k=3 x=0 1 1 1 rc=1
13 k=2 x=0 1 0 1 rc=3
14 k=3 x=1 1 0 1 rc=1
15 k=2 x=1 0 0 1 rc=2
16 k=1 x=0 0 0 1 rc=1
```

Example 2: Using a Character Variable and Positive Initial $k$
The following example uses a character variable to generate subsets in a minimal change
order.

```sas
data _null_;  
x='++++';  
n=length(x);  
k=countc(x, '+');  
put ' 1' +3 k= ' x=' x +3 x=;  
nsubs=2**n;  
do i=2 to nsubs;  
   rc=graycode(k, x, n, '+-');  
   put i 5. +3 k= ' x=' x +3 rc=;  
end;  
run;
```

948 Chapter 3 • Dictionary of Functions and CALL Routines for SAS Only
SAS writes the following output to the log:

<table>
<thead>
<tr>
<th></th>
<th>k=</th>
<th>x=</th>
<th>rc=</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>++++</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>++++</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>++++</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>++++</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>++++</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>++++</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>++++</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>++++</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>++++</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>++++</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>++++</td>
<td>3</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>++++</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>++++</td>
<td>2</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>++++</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>++++</td>
<td>4</td>
</tr>
<tr>
<td>16</td>
<td>3</td>
<td>++++</td>
<td>1</td>
</tr>
</tbody>
</table>

See Also

CALL Routines:
- “CALL GRAYCODE Routine” on page 746

HOLIDAYCK Function

Returns the number of occurrences of the holiday value between date1 and date2.

**Category:** Date and Time

**Restriction:** This function is not valid on the CAS server.

**Syntax**

HOLIDAYCK(holiday, date1, date2 [,locale])

**Required Arguments**

'holiday'

is a character constant, variable, or expression that specifies one of the values that are listed in the following table. The holiday variable can also be defined using the DATEKEYS procedure and is made available to the function by using the EVENTDS= system option.

**Table 3.3 Holiday Values and Their Descriptions**

<table>
<thead>
<tr>
<th>Holiday Value</th>
<th>Description</th>
<th>Date Celebrated</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOXING</td>
<td>Boxing Day</td>
<td>December 26</td>
</tr>
<tr>
<td>CANADA</td>
<td>Canada Day</td>
<td>July 1</td>
</tr>
<tr>
<td>CANADAOBSERVED</td>
<td>Canada Day observed</td>
<td>July 1, or July 2 if July 1 is a Sunday</td>
</tr>
<tr>
<td>Holiday Value</td>
<td>Description</td>
<td>Date Celebrated</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------------------------------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>CHRISTMAS</td>
<td>Christmas</td>
<td>December 25</td>
</tr>
<tr>
<td>COLUMBUS</td>
<td>Columbus Day</td>
<td>2nd Monday in October</td>
</tr>
<tr>
<td>EASTER</td>
<td>Easter Sunday</td>
<td>date varies</td>
</tr>
<tr>
<td>FATHERS</td>
<td>Father's Day</td>
<td>3rd Sunday in June</td>
</tr>
<tr>
<td>HALLOWEEN</td>
<td>Halloween</td>
<td>October 31</td>
</tr>
<tr>
<td>LABOR</td>
<td>Labor Day</td>
<td>1st Monday in September</td>
</tr>
<tr>
<td>MLK</td>
<td>Martin Luther King, Jr.'s birthday</td>
<td>3rd Monday in January beginning in 1986</td>
</tr>
<tr>
<td>MEMORIAL</td>
<td>Memorial Day</td>
<td>last Monday in May (since 1971)</td>
</tr>
<tr>
<td>MOTHERS</td>
<td>Mother's Day</td>
<td>2nd Sunday in May</td>
</tr>
<tr>
<td>NEWYEAR</td>
<td>New Year's Day</td>
<td>January 1</td>
</tr>
<tr>
<td>THANKSGIVING</td>
<td>U.S. Thanksgiving Day</td>
<td>4th Thursday in November</td>
</tr>
<tr>
<td>THANKSGIVINGCANADA</td>
<td>Canadian Thanksgiving Day</td>
<td>2nd Monday in October</td>
</tr>
<tr>
<td>USINDEPENDENCE</td>
<td>U.S. Independence Day</td>
<td>July 4</td>
</tr>
<tr>
<td>USPRESIDENTS</td>
<td>Abraham Lincoln's and George Washington's birthdays observed</td>
<td>3rd Monday in February (since 1971)</td>
</tr>
<tr>
<td>VALENTINES</td>
<td>Valentine's Day</td>
<td>February 14</td>
</tr>
<tr>
<td>VETERANS</td>
<td>Veterans Day</td>
<td>November 11</td>
</tr>
<tr>
<td>VETERANSUSG</td>
<td>Veterans Day - U.S. government-observed</td>
<td>U.S. government-observed date for Monday–Friday schedule</td>
</tr>
<tr>
<td>VETERANSUSPS</td>
<td>Veterans Day - U.S. post office observed</td>
<td>U.S. government-observed date for Monday–Saturday schedule (U.S. Post Office)</td>
</tr>
<tr>
<td>VICTORIA</td>
<td>Victoria Day</td>
<td>Monday on or preceding May 24</td>
</tr>
</tbody>
</table>
**HOLIDAYCK Function**

---

**date1**

specifies the beginning date value in the form ‘ddmonyy’d or ‘ddmonyyyy’d.

- **dd** is a two-digit integer that represents the day of the month.
- **mon** is a three-character string that represents the month: JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC.
- **yy** or **yyyy** is a two- or four-digit integer that represents the year. If only two digits are used, the YEARCUTOFF= option is valid.

**date2**

specifies the end date value in the form ‘ddmonyy’d or ‘ddmonyyyy’d.

- **dd** is a two-digit integer that represents the day of the month.
- **mon** is a three-character string that represents the month: JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC.
- **yy** or **yyyy** is a two- or four-digit integer that represents the year. If only two digits are used, the YEARCUTOFF= option is valid.

**Optional Argument**


**Details**

The EVENTDS= option makes the date keys available. You can define the date keys with the DATEKEYS procedure. Also, you can assign a locale to a date key with the DATEKEYS procedure. If you do not assign the locale to the date keys, then the predefined date keys do not have locales. If the locale argument is specified, then the date key must be defined using the DATEKEYS procedure, or a predefined date key needs to have a locale set that uses the DATEKEYS procedure. Then the new definition needs to be made available with the EVENTDS= system option.

**Example**

The following example specifies the Easter holiday when it occurs in March and April:

```sas
data;
  do year = 2000 to 2010;
    EasterInMarch=HOLIDAYCK('EASTER',MDY(3,1,year),mdy(3,31,year));
    EasterInApril=HOLIDAYCK('EASTER',MDY(4,1,year),mdy(4,30,year));
    output;
  end;
run;
```
Output 3.3  Easter Holiday in March and April

<table>
<thead>
<tr>
<th>EasterInMarch</th>
<th>EasterInApril</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

HOLIDAYCOUNT Function

Returns the number of holidays defined for a SAS date value.

Category: Date and Time
Restriction: This function is not valid on the CAS server.

Syntax

HOLIDAYCOUNT(date<,'locale'>)

Required Argument
date
specifies a SAS date value 'ddmmyy'd or 'ddmmyyyy'd.

<table>
<thead>
<tr>
<th>dd</th>
<th>mon</th>
<th>yy</th>
<th>yyyy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Optional Argument

'locale'
Details

The EVENTDS= option makes the date keys available. You can define the date keys with the DATEKEYS procedure. Also, you assign a locale to a date key with the DATEKEYS procedure. If you do not assign the locale to the date keys, then the predefined date keys do not have locales. If the locale argument is specified, then the date key must be defined using the DATEKEYS procedure, or a predefined date key needs to have a locale set that uses the DATEKEYS procedure. Then the new definition needs to be made available with the EVENTDS= system option.

Example

The following example specifies the number of holidays:

```plaintext
data a;
  length Holiday $32;
  format date weekdatx17.;
  do year=2000 to 2010;
    date=mdy(11,11,year);
    ObsVetNov11=holidaycount(date);
    do index=1 to ObsVetNov11;
      Holiday=holidayname(date,index);
      output;
    end;
  end;
run;
proc print data=a;
run;
```
Output 3.4  The Number of Veterans Day Holidays

### The SAS System

<table>
<thead>
<tr>
<th>Obs</th>
<th>Holiday</th>
<th>date</th>
<th>year</th>
<th>ObsVetNov11</th>
<th>index</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>VETERANS</td>
<td>Sat, 11 Nov 2000</td>
<td>2000</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>VETERANSUSPS</td>
<td>Sat, 11 Nov 2000</td>
<td>2000</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>VETERANS</td>
<td>Sun, 11 Nov 2001</td>
<td>2001</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>VETERANS</td>
<td>Mon, 11 Nov 2002</td>
<td>2002</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>VETERANSUSG</td>
<td>Mon, 11 Nov 2002</td>
<td>2002</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>VETERANSUSPS</td>
<td>Mon, 11 Nov 2002</td>
<td>2002</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>VETERANS</td>
<td>Tue, 11 Nov 2003</td>
<td>2003</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>VETERANSUSG</td>
<td>Tue, 11 Nov 2003</td>
<td>2003</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>VETERANSUSPS</td>
<td>Tue, 11 Nov 2003</td>
<td>2003</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>VETERANS</td>
<td>Thu, 11 Nov 2004</td>
<td>2004</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>VETERANSUSG</td>
<td>Thu, 11 Nov 2004</td>
<td>2004</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>VETERANSUSPS</td>
<td>Thu, 11 Nov 2004</td>
<td>2004</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>13</td>
<td>VETERANS</td>
<td>Fri, 11 Nov 2005</td>
<td>2005</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>VETERANSUSC</td>
<td>Fri, 11 Nov 2005</td>
<td>2005</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>15</td>
<td>VETERANSUSPS</td>
<td>Fri, 11 Nov 2005</td>
<td>2005</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>16</td>
<td>VETERANS</td>
<td>Sat, 11 Nov 2006</td>
<td>2006</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>VETERANSUSPS</td>
<td>Sat, 11 Nov 2006</td>
<td>2006</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>18</td>
<td>VETERANS</td>
<td>Sun, 11 Nov 2007</td>
<td>2007</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>VETERANS</td>
<td>Tue, 11 Nov 2008</td>
<td>2008</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>VETERANSUSG</td>
<td>Tue, 11 Nov 2008</td>
<td>2008</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>21</td>
<td>VETERANSUSPS</td>
<td>Tue, 11 Nov 2008</td>
<td>2008</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>22</td>
<td>VETERANS</td>
<td>Wed, 11 Nov 2009</td>
<td>2009</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>23</td>
<td>VETERANSUSG</td>
<td>Wed, 11 Nov 2009</td>
<td>2009</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>24</td>
<td>VETERANSUSPS</td>
<td>Wed, 11 Nov 2009</td>
<td>2009</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>25</td>
<td>VETERANS</td>
<td>Thu, 11 Nov 2010</td>
<td>2010</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>26</td>
<td>VETERANSUSG</td>
<td>Thu, 11 Nov 2010</td>
<td>2010</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>27</td>
<td>VETERANSUSPS</td>
<td>Thu, 11 Nov 2010</td>
<td>2010</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
HOLIDAYNAME Function

Returns the name of the holiday that corresponds to the SAS date or a blank string if a holiday is not defined for the SAS date.

Category: Date and Time
Restriction: This function is not valid on the CAS server.

Syntax

HOLIDAYNAME(date<,n<,'locale'>>)

Required Argument

date

specifies a SAS date value in the form 'ddmonyy'd or 'ddmonyyyy'd.

dd

is a two-digit integer that represents the day of the month.

mon

is a three-character string that represents the month: JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC.

yy or yyyy

is a two- or four-digit integer that represents the year. If only two digits are used, the YEARCUTOFF= option is valid.

Optional Arguments

n

specifies the index.

'locale'


Details

The EVENTDS= option makes the date keys available. You can define the date keys with the DATEKEYS procedure. Also, you assign a locale to a date key with the DATEKEYS procedure. If you do not assign the locale to the date keys, then the predefined date keys do not have locales. If the locale argument is specified, then the date key must be defined using the DATEKEYS procedure, or a predefined date key needs to have a locale set that uses the DATEKEYS procedure. Then the new definition needs to be made available with the EVENTDS= system option.

Example

The following example specifies the name of the holiday:

data a;
length Holiday $32;
format date weekdatx17.;
do year=2000 to 2010;
date=mdy(11,11,year);
ObsVetNov11=holidaycount(date);
do index=1 to ObsVetNov11;
   Holiday=holidayname(date,index);
   output;
   end;
end;
run;
proc print data=a;
run;
### The Name of the Holiday That Corresponds to the SAS Date

<table>
<thead>
<tr>
<th>Obs</th>
<th>Holiday</th>
<th>date</th>
<th>year</th>
<th>ObsVetNov11</th>
<th>index</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>VETERANS</td>
<td>Sat, 11 Nov 2000</td>
<td>2000</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>VETERANSUSPS</td>
<td>Sat, 11 Nov 2000</td>
<td>2000</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>VETERANS</td>
<td>Sun, 11 Nov 2001</td>
<td>2001</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>VETERANS</td>
<td>Mon, 11 Nov 2002</td>
<td>2002</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>VETERANSUSG</td>
<td>Mon, 11 Nov 2002</td>
<td>2002</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>VETERANSUSPS</td>
<td>Mon, 11 Nov 2002</td>
<td>2002</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>VETERANS</td>
<td>Tue, 11 Nov 2003</td>
<td>2003</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>VETERANSUSG</td>
<td>Tue, 11 Nov 2003</td>
<td>2003</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>VETERANSUSPS</td>
<td>Tue, 11 Nov 2003</td>
<td>2003</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>VETERANS</td>
<td>Thu, 11 Nov 2004</td>
<td>2004</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>VETERANSUSG</td>
<td>Thu, 11 Nov 2004</td>
<td>2004</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>VETERANSUSPS</td>
<td>Thu, 11 Nov 2004</td>
<td>2004</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>13</td>
<td>VETERANS</td>
<td>Fri, 11 Nov 2005</td>
<td>2005</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>VETERANSUSG</td>
<td>Fri, 11 Nov 2005</td>
<td>2005</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>15</td>
<td>VETERANSUSPS</td>
<td>Fri, 11 Nov 2005</td>
<td>2005</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>16</td>
<td>VETERANS</td>
<td>Sat, 11 Nov 2006</td>
<td>2006</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>VETERANSUSPS</td>
<td>Sat, 11 Nov 2006</td>
<td>2006</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>18</td>
<td>VETERANS</td>
<td>Sun, 11 Nov 2007</td>
<td>2007</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>VETERANS</td>
<td>Tue, 11 Nov 2008</td>
<td>2008</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>VETERANSUSG</td>
<td>Tue, 11 Nov 2008</td>
<td>2008</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>21</td>
<td>VETERANSUSPS</td>
<td>Tue, 11 Nov 2008</td>
<td>2008</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>22</td>
<td>VETERANS</td>
<td>Wed, 11 Nov 2009</td>
<td>2009</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>23</td>
<td>VETERANSUSG</td>
<td>Wed, 11 Nov 2009</td>
<td>2009</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>24</td>
<td>VETERANSUSPS</td>
<td>Wed, 11 Nov 2009</td>
<td>2009</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>25</td>
<td>VETERANS</td>
<td>Thu, 11 Nov 2010</td>
<td>2010</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>26</td>
<td>VETERANSUSG</td>
<td>Thu, 11 Nov 2010</td>
<td>2010</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>27</td>
<td>VETERANSUSPS</td>
<td>Thu, 11 Nov 2010</td>
<td>2010</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
HOLIDAYNX Function

Returns the \( n \)th occurrence of the holiday relative to the date argument.

**Category:** Date and Time

**Restriction:** This function is not valid on the CAS server.

**Syntax**

\[
\text{HOLIDAYNX('holiday',date,<n>,locale>)}
\]

**Required Arguments**

'holiday'

is a character constant, variable, or expression that specifies one of the values that are listed in the following table. The holiday variable can also be defined using the DATEKEYS procedure and is made available to the function by using the EVENTDS= system option.

**Table 3.4 Holiday Values and Their Descriptions**

<table>
<thead>
<tr>
<th>Holiday Value</th>
<th>Description</th>
<th>Date Celebrated</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOXING</td>
<td>Boxing Day</td>
<td>December 26</td>
</tr>
<tr>
<td>CANADA</td>
<td>Canada Day</td>
<td>July 1</td>
</tr>
<tr>
<td>CANADAOBSERVED</td>
<td>Canada Day observed</td>
<td>July 1, or July 2 if July 1 is a Sunday</td>
</tr>
<tr>
<td>CHRISTMAS</td>
<td>Christmas</td>
<td>December 25</td>
</tr>
<tr>
<td>COLUMBUS</td>
<td>Columbus Day</td>
<td>2nd Monday in October</td>
</tr>
<tr>
<td>EASTER</td>
<td>Easter Sunday</td>
<td>date varies</td>
</tr>
<tr>
<td>FATHERS</td>
<td>Father's Day</td>
<td>3rd Sunday in June</td>
</tr>
<tr>
<td>HALLOWEEN</td>
<td>Halloween</td>
<td>October 31</td>
</tr>
<tr>
<td>LABOR</td>
<td>Labor Day</td>
<td>1st Monday in September</td>
</tr>
<tr>
<td>MLK</td>
<td>Martin Luther King, Jr.'s birthday</td>
<td>3rd Monday in January beginning in 1986</td>
</tr>
<tr>
<td>MEMORIAL</td>
<td>Memorial Day</td>
<td>last Monday in May (since 1971)</td>
</tr>
<tr>
<td>MOTHERS</td>
<td>Mother's Day</td>
<td>2nd Sunday in May</td>
</tr>
<tr>
<td>NEWYEAR</td>
<td>New Year's Day</td>
<td>January 1</td>
</tr>
<tr>
<td>Holiday Value</td>
<td>Description</td>
<td>Date Celebrated</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------------------------------------------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>THANKSGIVING</td>
<td>U.S. Thanksgiving Day</td>
<td>4th Thursday in November</td>
</tr>
<tr>
<td>THANKSGIVINGCANADA</td>
<td>Canadian Thanksgiving Day</td>
<td>2nd Monday in October</td>
</tr>
<tr>
<td>USINDEPENDENCE</td>
<td>U.S. Independence Day</td>
<td>July 4</td>
</tr>
<tr>
<td>USPRESIDENTS</td>
<td>Abraham Lincoln's and George Washington's birthdays observed</td>
<td>3rd Monday in February (since 1971)</td>
</tr>
<tr>
<td>VALENTINES</td>
<td>Valentine's Day</td>
<td>February 14</td>
</tr>
<tr>
<td>VETERANS</td>
<td>Veterans Day</td>
<td>November 11</td>
</tr>
<tr>
<td>VETERANSUSG</td>
<td>Veterans Day - U.S. observed</td>
<td>U.S. government-observed date for Monday–Friday schedule</td>
</tr>
<tr>
<td>VETERANSUSPS</td>
<td>Veterans Day - U.S. post office observed</td>
<td>U.S. government-observed date for Monday–Saturday schedule (U.S. Post Office)</td>
</tr>
<tr>
<td>VICTORIA</td>
<td>Victoria Day</td>
<td>Monday on or preceding May 24</td>
</tr>
</tbody>
</table>

**date**

specifies the beginning date value in the form ‘ddmonyy’D or ‘ddmonyyyy’D.

**dd**
is a two-digit integer that represents the day of the month.

**mon**
is a three-character string that represents the month: JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC.

**yy or yyyy**
is a two- or four-digit integer that represents the year. If only two digits are used, the YEARCUTOFF= option is valid.

**Optional Arguments**

**n**
specifies the number of occurrences before or after the date. Specifying 0 for n indicates the occurrence on or immediately preceding the date. Specifying 1 for n indicates the next occurrence immediately after the date.

**'locale'**
Details

The `locale` argument is optional and should be the POSIX locale string. When `locale` is specified, the holiday date key should be defined using the DATEKEYS procedure. If the locale is specified, then only date keys with the specified locale or no locale defined that uses the EVENTDS= option are available.

Example

The following example specifies the nth occurrence of the holiday:

```sas
data holidays;
input holiday $char18.;
datalines;
Easter
Christmas
;
run;
proc print;run;
data WhenIs;
set holidays;
Today='25DEC2013'D;
put Today=;
put holiday=;
NextHoliday = HOLIDAYNX(holiday,today,1);
PreviousHoliday = HOLIDAYNX(holiday,NextHoliday,-1);
if ( PreviousHoliday = today ) then do;
    ThisDayIs=holiday;
p
    put ThisDayIs=;
    PreviousHoliday = HOLIDAYNX(holiday,today,-1);
end;
put PreviousHoliday=;
put NextHoliday=;
format Today NextHoliday PreviousHoliday DATE.;
run;
```

```
Today=25DEC13
holiday=Easter
PreviousHoliday=31MAR13
NextHoliday=20APR14
Today=25DEC13
holiday=Christmas
ThisDayIs=Christmas
PreviousHoliday=25DEC12
NextHoliday=25DEC14
```

**HOLIDAYNY Function**

Returns the nth occurrence of the holiday for the year.

- **Category:** Date and Time
- **Restriction:** This function is not valid on the CAS server.
Syntax

\[ \text{HOLIDAYNY}(\text{holiday}, \text{year}<,\text{locale}>) \]

**Required Arguments**

\textit{holiday}

is a character constant, variable, or expression that specifies one of the values that are listed in the following table. The \textit{holiday} variable can also be defined using the DATEKEYS procedure and is made available to the function by using the EVENTDS= system option.

**Table 3.5 Holiday Values and Their Descriptions**

<table>
<thead>
<tr>
<th>Holiday Value</th>
<th>Description</th>
<th>Date Celebrated</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOXING</td>
<td>Boxing Day</td>
<td>December 26</td>
</tr>
<tr>
<td>CANADA</td>
<td>Canada Day</td>
<td>July 1</td>
</tr>
<tr>
<td>CANADAOBSERVED</td>
<td>Canada Day observed</td>
<td>July 1, or July 2 if July 1 is a Sunday</td>
</tr>
<tr>
<td>CHRISTMAS</td>
<td>Christmas</td>
<td>December 25</td>
</tr>
<tr>
<td>COLUMBUS</td>
<td>Columbus Day</td>
<td>2nd Monday in October</td>
</tr>
<tr>
<td>EASTER</td>
<td>Easter Sunday</td>
<td>date varies</td>
</tr>
<tr>
<td>FATHERS</td>
<td>Father's Day</td>
<td>3rd Sunday in June</td>
</tr>
<tr>
<td>HALLOWEEN</td>
<td>Halloween</td>
<td>October 31</td>
</tr>
<tr>
<td>LABOR</td>
<td>Labor Day</td>
<td>1st Monday in September</td>
</tr>
<tr>
<td>MLK</td>
<td>Martin Luther King, Jr.'s birthday</td>
<td>3rd Monday in January beginning in 1986</td>
</tr>
<tr>
<td>MEMORIAL</td>
<td>Memorial Day</td>
<td>last Monday in May (since 1971)</td>
</tr>
<tr>
<td>MOTHERS</td>
<td>Mother's Day</td>
<td>2nd Sunday in May</td>
</tr>
<tr>
<td>NEWYEAR</td>
<td>New Year's Day</td>
<td>January 1</td>
</tr>
<tr>
<td>THANKSGIVING</td>
<td>U.S. Thanksgiving Day</td>
<td>4th Thursday in November</td>
</tr>
<tr>
<td>THANKSGIVINGCANADA</td>
<td>Canadian Thanksgiving Day</td>
<td>2nd Monday in October</td>
</tr>
<tr>
<td>USINDEPENDENCE</td>
<td>U.S. Independence Day</td>
<td>July 4</td>
</tr>
<tr>
<td>Holiday Value</td>
<td>Description</td>
<td>Date Celebrated</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------------------------------------------</td>
</tr>
<tr>
<td>USPRESIDENTS</td>
<td>Abraham Lincoln's and George Washington's birthdays observed</td>
<td>3rd Monday in February (since 1971)</td>
</tr>
<tr>
<td>VALENTINES</td>
<td>Valentine's Day</td>
<td>February 14</td>
</tr>
<tr>
<td>VETERANS</td>
<td>Veterans Day</td>
<td>November 11</td>
</tr>
<tr>
<td>VETERANSUSG</td>
<td>Veterans Day - U.S. government-observed</td>
<td>U.S. government-observed date for Monday–Friday schedule</td>
</tr>
<tr>
<td>VETERANSUSPS</td>
<td>Veterans Day - U.S. post office observed</td>
<td>U.S. government-observed date for Monday–Saturday schedule (U.S. Post Office)</td>
</tr>
<tr>
<td>VICTORIA</td>
<td>Victoria Day</td>
<td>Monday on or preceding May 24</td>
</tr>
</tbody>
</table>

*year*

is a numeric constant, variable, or expression that specifies a four-digit year. If you use a two-digit year, you must specify the YEARCUTOFF= system option.

**Optional Arguments**

*n*

specifies the occurrence of the holiday within the year.

`locale`


**Details**

The EVENTDS= option makes the date keys available. You can define the date keys with the DATEKEYS procedure. Also, you can assign a locale to a date key with the DATEKEYS procedure. If you do not assign the locale to the date keys, then the predefined date keys do not have locales. If the locale argument is specified, then the date key must be defined using the DATEKEYS procedure, or a predefined date key needs to have a locale set that uses the DATEKEYS procedure. Then the new definition needs to be made available with the EVENTDS= system option.

**Example**

The following example specifies the *n*th occurrence of the holiday for the year.

```sas
proc datekeys;
   datekeydef Sabbath=Saturday;
   datekeydata out=Sabbath;
run;
```
option eventds=(Sabbath);

data a;
  count = holidayck('Sabbath','01JAN2013'd,'31DEC2013'd);
  do i=1 to count;
    Sabbath=holidayny('Sabbath',2013,i);
    put Sabbath=;
  end;
  format Sabbath weekdatx17.;
run;
## Output 3.6  Dates When the Sabbath Holiday Occurs in 2013

<table>
<thead>
<tr>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sabbath=Sat, 5 Jan 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 12 Jan 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 19 Jan 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 26 Jan 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 2 Feb 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 9 Feb 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 16 Feb 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 23 Feb 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 2 Mar 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 9 Mar 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 16 Mar 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 23 Mar 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 30 Mar 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 6 Apr 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 13 Apr 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 20 Apr 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 27 Apr 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 4 May 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 11 May 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 18 May 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 25 May 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 1 Jun 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 8 Jun 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 15 Jun 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 22 Jun 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 29 Jun 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 6 Jul 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 13 Jul 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 20 Jul 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 27 Jul 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 3 Aug 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 10 Aug 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 17 Aug 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 24 Aug 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 31 Aug 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 7 Sep 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 14 Sep 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 21 Sep 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 28 Sep 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 5 Oct 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 12 Oct 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 19 Oct 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 26 Oct 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 2 Nov 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 9 Nov 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 16 Nov 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 23 Nov 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 30 Nov 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 7 Dec 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 14 Dec 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 21 Dec 2013</td>
</tr>
<tr>
<td>Sabbath=Sat, 28 Dec 2013</td>
</tr>
</tbody>
</table>

**HOLIDAYTEST Function**

Returns 1 if the holiday occurs on the SAS date value.

**Category:** Date and Time  
**Restriction:** This function is not valid on the CAS server.
Syntax

HOLIDAYTEST('holiday', date<',locale'>)

Required Arguments

'holiday'

is a character constant, variable, or expression that specifies one of the values that are listed in the following table. The holiday variable can also be defined using the DATEKEYS procedure and is made available to the function that uses the EVENTDS= system option.

Table 3.6  Holiday Values and Their Descriptions

<table>
<thead>
<tr>
<th>Holiday Value</th>
<th>Description</th>
<th>Date Celebrated</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOXING</td>
<td>Boxing Day</td>
<td>December 26</td>
</tr>
<tr>
<td>CANADA</td>
<td>Canada Day</td>
<td>July 1</td>
</tr>
<tr>
<td>CANADAOBSERVED</td>
<td>Canada Day observed</td>
<td>July 1, or July 2 if July 1 is a Sunday</td>
</tr>
<tr>
<td>CHRISTMAS</td>
<td>Christmas</td>
<td>December 25</td>
</tr>
<tr>
<td>COLUMBUS</td>
<td>Columbus Day</td>
<td>2nd Monday in October</td>
</tr>
<tr>
<td>EASTER</td>
<td>Easter Sunday</td>
<td>date varies</td>
</tr>
<tr>
<td>FATHERS</td>
<td>Father's Day</td>
<td>3rd Sunday in June</td>
</tr>
<tr>
<td>HALLOWEEN</td>
<td>Halloween</td>
<td>October 31</td>
</tr>
<tr>
<td>LABOR</td>
<td>Labor Day</td>
<td>1st Monday in September</td>
</tr>
<tr>
<td>MLK</td>
<td>Martin Luther King, Jr.'s birthday</td>
<td>3rd Monday in January beginning in 1986</td>
</tr>
<tr>
<td>MEMORIAL</td>
<td>Memorial Day</td>
<td>last Monday in May (since 1971)</td>
</tr>
<tr>
<td>MOTHERS</td>
<td>Mother's Day</td>
<td>2nd Sunday in May</td>
</tr>
<tr>
<td>NEWYEAR</td>
<td>New Year's Day</td>
<td>January 1</td>
</tr>
<tr>
<td>THANKSGIVING</td>
<td>U.S. Thanksgiving Day</td>
<td>4th Thursday in November</td>
</tr>
<tr>
<td>THANKSGIVINGCANADA</td>
<td>Canadian Thanksgiving Day</td>
<td>2nd Monday in October</td>
</tr>
<tr>
<td>USINDEPENDENCE</td>
<td>U.S. Independence Day</td>
<td>July 4</td>
</tr>
<tr>
<td>Holiday Value</td>
<td>Description</td>
<td>Date Celebrated</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------------------------------------------</td>
<td>------------------------------------------------------</td>
</tr>
<tr>
<td>USPRESIDENTS</td>
<td>Abraham Lincoln's and George Washington’s birthdays observed</td>
<td>3rd Monday in February (since 1971)</td>
</tr>
<tr>
<td>VALENTINES</td>
<td>Valentine's Day</td>
<td>February 14</td>
</tr>
<tr>
<td>VETERANS</td>
<td>Veterans Day</td>
<td>November 11</td>
</tr>
<tr>
<td>VETERANSUSG</td>
<td>Veterans Day - U.S. government-observed</td>
<td>U.S. government-observed date for Monday–Friday schedule</td>
</tr>
<tr>
<td>VETERANSUSPS</td>
<td>Veterans Day - U.S. post office observed</td>
<td>U.S. government-observed date for Monday–Saturday schedule (U.S. Post Office)</td>
</tr>
<tr>
<td>VICTORIA</td>
<td>Victoria Day</td>
<td>Monday on or preceding May 24</td>
</tr>
</tbody>
</table>

**date**

Specifies the beginning date value in the form ‘ddmonyy’ or ‘ddmonyyyy’.

- **dd** is a two-digit integer that represents the day of the month.
- **mon** is a three-character string that represents the month: JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC.
- **yy** or **yyyy** is a two- or four-digit integer that represents the year. If only two digits are used, the YEARCUTOFF= option is valid.

**Optional Argument**

**'locale'**


**Details**

The EVENTDS= option makes the date keys available. You can define the date keys with the DATEKEYS procedure. Also, you can assign a locale to a date key with the DATEKEYS procedure. If you do not assign the locale to the date keys, then the predefined date keys do not have locales. If the locale argument is specified, then the date key must be defined using the DATEKEYS procedure, or a predefined date key needs to have a locale set that uses the DATEKEYS procedure. Then the new definition needs to be made available with the EVENTDS= system option.

**Example**

The following example specifies whether the holiday occurs on the SAS date:

```sas
data a;
```
length is $9;
holiday='Christmas';
test=holidaytest(holiday,today());
if (test=1) then is='today';
else is='not today';
put holiday= is=;
run;

holiday=Christmas is=not today

**HTMLDECODE Function**

Decodes a string that contains HTML numeric character references or HTML character entity references, and returns the decoded string.

**Category:** Web Tools

**Restrictions:**
This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see [Internationalization Compatibility](#).

This function is not valid on the CAS server.

**Syntax**

```
HTMLDECODE(expression)
```

**Required Argument**

`expression`

- specifies a character constant, variable, or expression.

**Details**

The HTMLDECODE function recognizes the following character entity references:

<table>
<thead>
<tr>
<th>Character Entity Reference</th>
<th>Decoded Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>&amp;</td>
</tr>
<tr>
<td>&lt;</td>
<td>&lt;</td>
</tr>
<tr>
<td>&gt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td>'</td>
<td>’</td>
</tr>
<tr>
<td>&amp;copy</td>
<td>©</td>
</tr>
<tr>
<td>&amp;reg</td>
<td>®</td>
</tr>
</tbody>
</table>
Unrecognized entities (&<name>;+) are left unmodified in the output string.

The HTMLDECODE function recognizes numeric entity references that are of the form

&###nnn;
where nnn specifies a decimal number that contains one or more digits.

&amp;###Xnnn;
where nnn specifies a hexadecimal number that contains one or more digits.

Note: Numeric character references that cannot be represented in the current SAS session encoding will not be decoded. The reference will be copied unchanged to the output string.

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1=htmldecode('not a &lt;tag&gt;');</td>
<td>not a &lt;tag&gt;</td>
</tr>
<tr>
<td>x2=htmldecode('&amp;');</td>
<td>&amp;</td>
</tr>
<tr>
<td>x3=htmldecode ('©®™');</td>
<td>© ® ™</td>
</tr>
</tbody>
</table>

**See Also**

Functions:

- “HTMLENCODE Function” on page 968

**HTMLENCODE Function**

Encodes characters using HTML character entity references, and returns the encoded string.

- **Category:** Web Tools
- **Restrictions:**
  This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility.
  This function is not valid on the CAS server.

- **Syntax**

  HTMLENCODE(expression, <option(s)>)

<table>
<thead>
<tr>
<th>Character Entity Reference</th>
<th>Decoded Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;trade</td>
<td>™</td>
</tr>
</tbody>
</table>
**Required Argument**

`expression` specifies a character constant, variable, or expression. By default, any greater-than (`>`), less-than (`<`), and ampersand (`&`) characters are encoded as `&gt;`, `&lt;`, and `&amp;`, respectively. In SAS 9 only, this behavior can be modified with the `option` argument.

*Note:* The encoded string can be longer than the output string. You should take the additional length into consideration when you define your output variable. If the encoded string exceeds the maximum length that is defined, the output string might be truncated.

**Optional Argument**

`option` is a character constant, variable, or expression that specifies the type of characters to encode. If you use more than one option, separate the options by spaces. The following options are available:

<table>
<thead>
<tr>
<th>Option</th>
<th>Character</th>
<th>Character Entity Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>amp</td>
<td>&amp;</td>
<td>&amp;</td>
<td>The HTMLENCODE function encodes these characters by default. If you need to encode these characters only, then you do not need to specify the options argument. However, if you specify any value for the options argument, then the defaults are overridden, and you must explicitly specify the options for all of the characters that you want to encode.</td>
</tr>
<tr>
<td>gt</td>
<td>&gt;</td>
<td>&gt;</td>
<td></td>
</tr>
<tr>
<td>lt</td>
<td>&lt;</td>
<td>&lt;</td>
<td></td>
</tr>
<tr>
<td>apos</td>
<td>'</td>
<td>'</td>
<td>Use this option to encode the apostrophe (') character in text that is used in an HTML or XML tag attribute.</td>
</tr>
<tr>
<td>quot</td>
<td>&quot;</td>
<td>&quot;</td>
<td>Use this option to encode the double quotation mark (&quot;) character in text that is used in an HTML or XML tag attribute.</td>
</tr>
<tr>
<td>7bit</td>
<td>any character that is not represented in 7-bit ASCII encoding</td>
<td>&amp;#xnnn; (Unicode)</td>
<td><code>nnn</code> is a one or more digit hexadecimal number. Encode these characters to create HTML or XML that is easily transferred through communication paths that might support only 7-bit ASCII encodings (for example, ftp or email).</td>
</tr>
<tr>
<td>Option</td>
<td>Character</td>
<td>Character Entity Reference</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-----------</td>
<td>----------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>copy</td>
<td>©</td>
<td>&amp;copy</td>
<td>Use this option in the HTML ENCODE function to turn on the encoding for this character. The value of val is &amp;copy after the following code is processed: <code>val = htmlencode('©', 'copy');</code></td>
</tr>
<tr>
<td>reg</td>
<td>®</td>
<td>&amp;reg</td>
<td>Use this option in the HTML ENCODE function to turn on the encoding for this character. The value of val is &amp;reg after the following code is processed: <code>val = htmlencode('®', 'reg');</code></td>
</tr>
<tr>
<td>trade</td>
<td>™</td>
<td>&amp;trade</td>
<td>Use this option in the HTML ENCODE function to turn on the encoding for this character. The value of val is &amp;trade after the following code is processed: <code>val = htmlencode('™', 'trade');</code></td>
</tr>
</tbody>
</table>

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>htmlencode(&quot;John's test &lt;tag&gt;&quot;)</td>
<td>John's test &lt;tag&gt;</td>
</tr>
<tr>
<td>htmlencode(&quot;John's test &lt;tag&gt;&quot;, 'apos')</td>
<td>John&amp;apos;s test &lt;tag&gt;</td>
</tr>
<tr>
<td>htmlencode(&quot;John &quot;Jon&quot; Smith &lt;tag&gt;&quot;, 'quot')</td>
<td>John &quot;Jon&quot; Smith &lt;tag&gt;</td>
</tr>
<tr>
<td>htmlencode(&quot;A&amp;B&amp;C&quot;, 'amp lt gt apos')</td>
<td>'A&amp;&amp;B&amp;&amp;C'</td>
</tr>
<tr>
<td>htmlencode('80'x, '7bit') (80'x is the euro symbol in Western European locales.)</td>
<td>&amp;#x20AC; (20AC is the Unicode code point for the euro symbol.)</td>
</tr>
<tr>
<td>copy = htmlencode('©', 'copy');</td>
<td>©</td>
</tr>
<tr>
<td>reg = htmlencode('®', 'reg');</td>
<td>®</td>
</tr>
<tr>
<td>trade = htmlencode('™', 'trade');</td>
<td>™</td>
</tr>
</tbody>
</table>
See Also

Functions:
- “HTMLDECODE Function” on page 967

### IFC Function

Returns a character value based on whether an expression is true, false, or missing.

**Category:** Character  

**Restrictions:** This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see [Internationalization Compatibility](#).

This function is not valid on the CAS server.

### Syntax

```
IFC(logical-expression, value-returned-when-true, value-returned-when-false <, value-returned-when-missing>)
```

### Required Arguments

- `logical-expression` specifies a numeric constant, variable, or expression.
- `value-returned-when-true` specifies a character constant, variable, or expression that is returned when the value of `logical-expression` is true.
- `value-returned-when-false` specifies a character constant, variable, or expression that is returned when the value of `logical-expression` is false.

### Optional Argument

- `value-returned-when-missing` specifies a character constant, variable, or expression that is returned when the value of `logical-expression` is missing.

### Details

**Length of Returned Variable**

In a DATA step, if the IFC function returns a value to a variable that has not previously been assigned a length, then that variable is given a length of 200 bytes.

**The Basics**

The IFC function uses conditional logic that enables you to select among several values based on the value of a logical expression.

IFC evaluates the first argument, `logical-expression`. If `logical-expression` is true (that is, not zero and not missing), then IFC returns the value in the second argument. If `logical-expression` is a missing value, and you have a fourth argument, then IFC returns the
value in the fourth argument. Otherwise, if `logical-expression` is false, IFC returns the value in the third argument.

The IFC function is useful in DATA step expressions, and even more useful in WHERE clauses and other expressions where it is not convenient or possible to use an IF/THEN/ELSE construct.

**Comparisons**

The IFC function is similar to the IFN function except that IFC returns a character value while IFN returns a numeric value.

**Examples**

**Example 1**

In the following example, IFC evaluates the expression `grade>80` to implement the logic that determines the performance of several members on a team.

```sas
data _null_
  input name $ grade;
  performance = ifc(grade>80, 'Pass             ', 'Needs Improvement');
  put name= performance=
  datalines;
John 74
Kareem 89
Kati 100
Maria 92
;
```

SAS writes the following output to the log:

```
name=John performance=Needs Improvement
name=Kareem performance=Pass
name=Kati performance=Pass
name=Maria performance=Pass
```

**Example 2**

This example uses an IF/THEN/ELSE construct to generate the same output that is generated by the IFC function.

```sas
data _null_
  input name $ grade;
  if grade>80 then performance='Pass             ';
    else performance = 'Needs Improvement';
  put name= performance=
  datalines;
John 74
Sam 89
Kati 100
Maria 92
;
```
IFN Function

Returns a numeric value based on whether an expression is true, false, or missing.

**Category:** Numeric

**Restrictions:** This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility.

This function is not valid on the CAS server.

**Syntax**

```plaintext
IFN(logical-expression, value-returned-when-true, value-returned-when-false, value-returned-when-missing)
```

**Required Arguments**

- `logical-expression` specifies a numeric constant, variable, or expression.
- `value-returned-when-true` specifies a numeric constant, variable, or expression that is returned when the value of `logical-expression` is true.
- `value-returned-when-false` specifies a numeric constant, variable, or expression that is returned when the value of `logical-expression` is false.

**Optional Argument**

- `value-returned-when-missing` specifies a numeric constant, variable or expression that is returned when the value of `logical-expression` is missing.

**Details**

The IFN function uses conditional logic that enables you to select among several values based on the value of a logical expression.

IFN evaluates the first argument, then `logical-expression`. If `logical-expression` is true (that is, not zero and not missing), then IFN returns the value in the second argument. If

---

SAS writes the following output to the log:

<table>
<thead>
<tr>
<th>name</th>
<th>performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Needs Improvement</td>
</tr>
<tr>
<td>Sam</td>
<td>Pass</td>
</tr>
<tr>
<td>Kati</td>
<td>Pass</td>
</tr>
<tr>
<td>Maria</td>
<td>Pass</td>
</tr>
</tbody>
</table>

See Also

Functions:

- “IFN Function” on page 973
**logical-expression** is a missing value, and you have a fourth argument, then IFN returns the value in the fourth argument. Otherwise, if **logical-expression** is false, IFN returns the value in the third argument.

The IFN function, an IF/THEN/ELSE construct, or a WHERE statement can produce the same results. (See examples.) However, the IFN function is useful in DATA step expressions when it is not convenient or possible to use an IF/THEN/ELSE construct or a WHERE statement.

**Comparisons**

The IFN function is similar to the IFC function, except that IFN returns a numeric value whereas IFC returns a character value.

**Examples**

**Example 1: Calculating Commission Using the IFN Function**

In the following example, IFN evaluates the expression **TotalSales > 10000**. If total sales exceeds $10,000, then the sales commission is 5% of the total sales. If total sales is less than $10,000, then the sales commission is 2% of the total sales.

```sas
data _null_;  
  input TotalSales;  
  commission=ifn(TotalSales > 10000, TotalSales*.05, TotalSales*.02);  
  put commission=;  
  datalines;  
  25000  
  10000  
  500  
  10300  
;  
SAS writes the following output to the log:
```

```
commission=1250  
commission=200  
commission=10  
commission=515
```

**Example 2: Calculating Commission Using an IF/THEN/ELSE Construct**

In the following example, an IF/THEN/ELSE construct evaluates the expression **TotalSales > 10000**. If total sales exceeds $10,000, then the sales commission is 5% of the total sales. If total sales is less than $10,000, then the sales commission is 2% of the total sales.

```sas
data _null_;  
  input TotalSales;  
  if TotalSales > 10000 then commission = .05 * TotalSales;  
  else commission = .02 * TotalSales;  
  put commission=;  
  datalines;  
  25000  
  10000  
  500  
  10300  
;  
SAS writes the following output to the log:
```

```
commission=1250  
commission=200  
commission=10  
commission=515
```
SAS writes the following output to the log:

```
commission=1250
commission=200
commission=10
commission=515
```

**Example 3: Calculating Commission Using a WHERE Statement**

In the following example, a WHERE statement evaluates the expression $\text{TotalSales} > 10000$. If total sales exceeds $10,000$, then the sales commission is 5% of the total sales. If total sales is less than $10,000$, then the sales commission is 2% of the total sales. The output shows only those salespeople whose total sales exceed $10,000$.

```sas
data sales;
    input SalesPerson $ TotalSales;
datalines;
    Michaels 25000
    Janowski 10000
    Chen 500
    Gupta 10300
    ;
data commission;
    set sales;
    where TotalSales > 10000;
    commission = TotalSales * .05;
run;
proc print data=commission;
    title 'Commission for Total Sales > 1000';
run;
```

**Figure 3.24** Output from a WHERE Statement

**Commission for Total Sales > 1000**

<table>
<thead>
<tr>
<th>Obs</th>
<th>SalesPerson</th>
<th>TotalSales</th>
<th>commission</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Michaels</td>
<td>25000</td>
<td>1250</td>
</tr>
<tr>
<td>2</td>
<td>Gupta</td>
<td>10300</td>
<td>515</td>
</tr>
</tbody>
</table>

**See Also**

**Functions:**
- “IFC Function” on page 971

**IORCMSG Function**

Returns a formatted error message for _IORC_.

**Category:** SAS File I/O
Restriction: This function is not valid on the CAS server.

Syntax

IORCMSG()

Details

If the IORCMSG function returns a value to a variable that has not yet been assigned a length, then by default the variable is assigned a length of 200.

The IORCMSG function returns the formatted error message that is associated with the current value of the automatic variable _IORC_. The _IORC_ variable is created when you use the MODIFY statement, or when you use the SET statement with the KEY= option. The value of the _IORC_ variable is internal and is meant to be read in conjunction with the SYSRC autocall macro. If you try to set _IORC_ to a specific value, you might get unexpected results.

Example

In the following program, observations are either rewritten or added to the updated master file that contains bank accounts and current bank balance. The program queries the _IORC_ variable and returns a formatted error message if the _IORC_ value is unexpected.

```sas
libname bank 'SAS-library';
data bank.master(index=(AccountNum));
   infile 'external-file-1';
   format balance dollar8.;
   input @ 1 AccountNum $ 1-3 @ 5 balance 5-9;
run;
data bank.trans(index=(AccountNum));
   infile 'external-file-2';
   format deposit dollar8.;
   input @ 1 AccountNum $ 1-3 @ 5 deposit 5-9;
run;
data bank.master;
   set bank.trans;
   modify bank.master key=AccountNum;
   if (_IORC_ EQ %sysrc(_SOK)) then
     do;
        balance=balance+deposit;
        replace;
     end;
   else
     if (_IORC_ = %sysrc(_DSENOM)) then
     do;
        balance=deposit;
        output;
     _error_=0;
     end;
   else
     do;
        errmsg=IORCMSG();
        put 'Unknown error condition:' errmsg;
```

```
LEXCOMB Function

Generates all distinct combinations of the nonmissing values of \( n \) variables taken \( k \) at a time in lexicographic order.

**Category:** Combinatorial

**Restrictions:**
- This function is not valid on the CAS server.
- The LEXCOMB function cannot be executed when you use the \%SYSFUNC macro.

**Syntax**

\[
\text{LEXCOMB}(\text{count}, \, k, \, \text{variable-1}, \, \ldots, \, \text{variable-n})
\]

**Required Arguments**

- \(\text{count}\)
  - specifies an integer variable that is assigned values from 1 to the number of combinations in a loop.

- \(k\)
  - is a constant, variable, or expression between 1 and \( n \), inclusive, that specifies the number of items in each combination.

- \(\text{variable}\)
  - specifies either all numeric variables, or all character variables that have the same length. The values of these variables are permuted.

**Requirement**

Initialize these variables before you execute the LEXCOMB function.

**Tip**

After executing the LEXCOMB function, the first \( k \) variables contain the values in one combination.

**Details**

**The Basics**

Use the LEXCOMB function in a loop where the first argument to LEXCOMB takes each integral value from 1 to the number of distinct combinations of the nonmissing values of the variables. In each execution of LEXCOMB within this loop, \( k \) should have the same value.

**Number of Combinations**

When all of the variables have nonmissing, unequal values, then the number of combinations is \( \text{COMB}(n,k) \). If the number of variables that have missing values is \( m \), and all the nonmissing values are unequal, then LEXCOMB produces \( \text{COMB}(n-m,k) \) combinations because the missing values are omitted from the combinations.

When some of the variables have equal values, the exact number of combinations is difficult to compute, but \( \text{COMB}(n,k) \) provides an upper bound. You do not need to
compute the exact number of combinations, provided that your program leaves the loop when LEXCOMB returns a value that is less than zero.

**LEXCOMB Processing**

On the first execution of the LEXCOMB function, the following actions occur:

- The argument types and lengths are checked for consistency.
- The m missing values are assigned to the last m arguments.
- The \( n-m \) nonmissing values are assigned in ascending order to the first \( n-m \) arguments following \( \text{count} \).
- LEXCOMB returns 1.

On subsequent executions, up to and including the last combination, the following actions occur:

- The next distinct combination of the nonmissing values is generated in lexicographic order.
- If \( \text{variable-}i \) through \( \text{variable-}i \) did not change, but \( \text{variable-}j \) did change, where \( j=i+1 \), then LEXCOMB returns \( j \).

If you execute the LEXCOMB function after generating all the distinct combinations, then LEXCOMB returns \(-1\).

If you execute the LEXCOMB function with the first argument out of sequence, then the results are not useful. In particular, if you initialize the variables and then immediately execute the LEXCOMB function with a first argument of \( j \), you do not get the \( j \)th combination (except when \( j \) is 1). To get the \( j \)th combination, you must execute the LEXCOMB function \( j \) times. The first argument takes values from 1 through \( j \) in that exact order.

**Comparisons**

The LEXCOMB function generates all distinct combinations of the nonmissing values of \( n \) variables taken \( k \) at a time in lexicographic order. The ALLCOMB function generates all combinations of the values of \( k \) variables taken \( k \) at a time in a minimum change order.

**Examples**

**Example 1: Generating Distinct Combinations in Lexicographic Order**

The following example uses the LEXCOMB function to generate distinct combinations in lexicographic order.

```sas
data _null_;
  array x[5] $3 ('ant' 'bee' 'cat' 'dog' 'ewe');
  n=dim(x);
  k=3;
  ncomb=comb(n, k);
  do j=1 to ncomb+1;
    rc=lexcomb[j, k, of x[*]];
    put j 5. +3 x1-x3 +3 rc=;
    if rc<0 then leave;
  end;
run;
```

978 Chapter 3 • Dictionary of Functions and CALL Routines for SAS Only
SAS writes the following output to the log:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>rc=1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ant</td>
<td>bee</td>
<td>cat</td>
</tr>
<tr>
<td>2</td>
<td>ant</td>
<td>bee</td>
<td>dog</td>
</tr>
<tr>
<td>3</td>
<td>ant</td>
<td>bee</td>
<td>ewe</td>
</tr>
<tr>
<td>4</td>
<td>ant</td>
<td>cat</td>
<td>dog</td>
</tr>
<tr>
<td>5</td>
<td>ant</td>
<td>cat</td>
<td>ewe</td>
</tr>
<tr>
<td>6</td>
<td>ant</td>
<td>dog</td>
<td>ewe</td>
</tr>
<tr>
<td>7</td>
<td>bee</td>
<td>cat</td>
<td>dog</td>
</tr>
<tr>
<td>8</td>
<td>bee</td>
<td>cat</td>
<td>ewe</td>
</tr>
<tr>
<td>9</td>
<td>bee</td>
<td>dog</td>
<td>ewe</td>
</tr>
<tr>
<td>10</td>
<td>cat</td>
<td>dog</td>
<td>ewe</td>
</tr>
<tr>
<td>11</td>
<td>cat</td>
<td>dog</td>
<td>ewe</td>
</tr>
</tbody>
</table>

**Example 2: Generating Distinct Combinations in Lexicographic Order: Another Example**

Here is another example of using the LEXCOMB function.

```sas
data _null_;
  array x[5] $3 ('X' 'Y' 'Z' 'Z' 'Y');
  n=dim(x);
  k=3;
  ncomb=comb(n, k);
  do j=1 to ncomb+1;
    rc=lexcomb(j, k, of x[*]);
    put j 5. +3 x1-x3 +3 rc=;
    if rc<0 then leave;
  end;
run;
```

SAS writes the following output to the log:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>rc=1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>2</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
</tr>
<tr>
<td>3</td>
<td>X</td>
<td>Z</td>
<td>Z</td>
</tr>
<tr>
<td>4</td>
<td>Y</td>
<td>Y</td>
<td>Z</td>
</tr>
<tr>
<td>5</td>
<td>Y</td>
<td>Z</td>
<td>Z</td>
</tr>
<tr>
<td>6</td>
<td>Y</td>
<td>Z</td>
<td>Z</td>
</tr>
</tbody>
</table>

**See Also**

Functions:
- “ALLCOMB Function” on page 716

CALL Routines:
- “CALL LEXCOMB Routine” on page 760

---

### LEXCOMBI Function

Generates all combinations of the indices of \( n \) objects taken \( k \) at a time in lexicographic order.

**Category:** Combinatorial

**Restrictions:**
- This function is not valid on the CAS server.
- The LEXCOMBI function cannot be executed when you use the %SYSFUNC macro.
Syntax

LEXCOMBI\( (n, k, index-1, \ldots, k) \)

**Required Arguments**

\( n \)

is a numeric constant, variable, or expression that specifies the total number of objects.

\( K \)

is a numeric constant, variable, or expression that specifies the number of objects in each combination.

\( index \)

is a numeric variable that contains indices of the objects in the combination that is returned. Indices are integers between 1 and \( n \) inclusive.

*Tip* If \( index-1 \) is missing or zero, then the LEXCOMBI function initializes the indices to \( index-1 = 1 \) through \( index-k = k \). Otherwise, LEXCOMBI creates a new combination by removing one index from the combination and adding another index.

**Details**

Before the first execution of the LEXCOMBI function, complete one of the following tasks:

- Set \( index-1 \) equal to zero or to a missing value.
- Initialize \( index-1 \) through \( index-k \) to distinct integers between 1 and \( n \) inclusive.

The number of combinations of \( n \) objects taken \( k \) at a time can be computed as \( \text{COMB}(n, k) \). To generate all combinations of \( n \) objects taken \( k \) at a time, execute the LEXCOMBI function in a loop that executes \( \text{COMB}(n, k) \) times.

In the LEXCOMBI function, the returned value indicates which, if any, indices changed. If \( index-1 \) through \( index-i \) did not change, but \( index-j \) did change, where \( j = i+1 \), then LEXCOMBI returns \( i \). If LEXCOMBI is called after the last combinations in lexicographic order have been generated, then LEXCOMBI returns –1.

**Comparisons**

The LEXCOMBI function generates all combinations of the indices of \( n \) objects taken \( k \) at a time in lexicographic order. The ALLCOMBI function generates all combinations of the indices of \( n \) objects taken \( k \) at a time in a minimum change order.

**Example**

The following example uses the LEXCOMBI function to generate combinations of indices in lexicographic order.

```sas
data _null_;
array x[5] $3 ('ant' 'bee' 'cat' 'dog' 'ewe');
array c[3] $3;
array i[3];
n=dims(x);
```
k=dim(i);
i[1]=0;
ncomb=comb(n, k);
do j=1 to ncomb+1;
   rc=lexcombi(n, k, of i[*]);
do h=1 to k;
   c[h]=x[i[h]];
end;
put @4 j= @10 'i= ' i[*] +3 'c= ' c[*] +3 rc=;
end;
run;

SAS writes the following output to the log:

```
j=1   i= 1 2 3    c= ant bee cat    rc=1
j=2   i= 1 2 4    c= ant bee dog    rc=3
j=3   i= 1 2 5    c= ant bee ewe    rc=3
j=4   i= 1 3 4    c= ant cat dog    rc=2
j=5   i= 1 3 5    c= ant cat ewe    rc=3
j=6   i= 1 4 5    c= ant dog ewe    rc=2
j=7   i= 2 3 4    c= bee cat dog    rc=1
j=8   i= 2 3 5    c= bee cat ewe    rc=3
j=9   i= 2 4 5    c= bee dog ewe    rc=2
j=10  i= 3 4 5    c= cat dog ewe    rc=1
j=11  i= 3 4 5    c= cat dog ewe    rc=-1
```

See Also

CALL Routines:
- “CALL ALLCOMBI Routine” on page 730
- “CALL LEXCOMBI Routine” on page 764

LEXPERK Function

Generates all distinct permutations of the nonmissing values of \( n \) variables taken \( k \) at a time in lexicographic order.

**Category:** Combinatorial

**Restrictions:**
This function is not valid on the CAS server.
The LEXPERK function cannot be executed when you use the %SYSFUNC macro.

**Syntax**

```
LEXPERK(count, k, variable-1, ..., variable-n)
```

**Required Arguments**

**count**
- specifies an integer variable that ranges from 1 to the number of permutations.

**k**
- is a numeric constant, variable, or expression with an integer value between 1 and \( n \) inclusive.
variable

specifies either all numeric variables, or all character variables that have the same
length. The values of these variables are permuted.

Requirement

Initialize these variables before you execute the LEXPERK function.

Tip

After executing LEXPERK, the first \( k \) variables contain the values in
one permutation.

Details

The Basics

Use the LEXPERK function in a loop where the first argument to LEXPERK takes each
integral value from 1 to the number of distinct permutations of \( k \) nonmissing values of
the variables. In each execution of LEXPERK within this loop, \( k \) should have the same
value.

Number of Permutations

When all of the variables have nonmissing, unequal values, the number of permutations
is \( \text{PERM}(n,k) \). If the number of variables that have missing values is \( m \), and all the
nonmissing values are unequal, the LEXPERK function produces \( \text{PERM}(n-m,k) \)
permutations because the missing values are omitted from the permutations. When some
of the variables have equal values, the exact number of permutations is difficult to
compute, but \( \text{PERM}(n,k) \) provides an upper bound. You do not need to compute the
exact number of permutations, provided you exit the loop when the LEXPERK function
returns a value that is less than zero.

LEXPERK Processing

On the first execution of the LEXPERK function, the following actions occur:

- The argument types and lengths are checked for consistency.
- The \( m \) missing values are assigned to the last \( m \) arguments.
- The \( n-m \) nonmissing values are assigned in ascending order to the first \( n-m \)
  arguments following count.
- LEXPERK returns 1.

On subsequent executions, up to and including the last permutation, the following
actions occur:

- The next distinct permutation of \( k \) nonmissing values is generated in lexicographic
  order.
- If \( \text{variable-1} \) through \( \text{variable-i} \) did not change, but \( \text{variable-i} \) did change, where \( j=i+1 \), then LEXPERK returns \( j \).

If you execute the LEXPERK function after generating all the distinct permutations, then
LEXPERK returns \(-1\).

If you execute the LEXPERK function with the first argument out of sequence, then the
results are not useful. In particular, if you initialize the variables and then immediately
execute the LEXPERK function with a first argument of \( j \), you do not get the \( j^{th} \)
permutation (except when \( j \) is 1). To get the \( j^{th} \) permutation, you must execute the
LEXPERK function \( j \) times. The first argument takes values from 1 through \( j \) in that
exact order.
Comparisons

The LEXPERK function generates all distinct permutations of the nonmissing values of \( n \) variables taken \( k \) at a time in lexicographic order. The LEXPERM function generates all distinct permutations of the nonmissing values of \( n \) variables in lexicographic order. The ALLPERM function generates all permutations of the values of several variables in a minimal change order.

Example

Here is an example of the LEXPERK function.

```sas
data _null_;
  array x[5] $3 {'X' 'Y' 'Z' 'Z' 'Y'};
  n=dim(x);
  k=3;
  nperm=perm(n, k);
  do j=1 to nperm+1;
    rc=lexperk(j, k, of x[*]);
    put j 5. +3 x1-x3 +3 rc=;
    if rc<0 then leave;
  end;
run;
```

SAS writes the following output to the log:

```
1   X Y Y   rc=1
2   X Y Z   rc=3
3   X Z Y   rc=2
4   X Z Z   rc=3
5   Y X Y   rc=1
6   Y X Z   rc=3
7   Y Y X   rc=2
8   Y Y Z   rc=3
9   Y Z X   rc=2
10  Y Z Y   rc=3
11  Y Z Z   rc=3
12  Z X Y   rc=1
13  Z X Z   rc=3
14  Z Y X   rc=2
15  Z Y Y   rc=3
16  Z Y Z   rc=3
17  Z Z X   rc=2
18  Z Z Y   rc=3
19  Z Z Y   rc=-1
```

See Also

Functions:
- “ALLPERM Function” on page 718
- “LEXPERM Function” on page 984

CALL Routines:
- “CALL RANPERK Routine” on page 829
- “CALL RANPERM Routine” on page 831
LEXPERM Function

Generates all distinct permutations of the nonmissing values of several variables in lexicographic order.

**Category:** Combinatorial

**Restriction:** This function is not valid on the CAS server.

**Syntax**

```sas
LEXPERM(count, variable-1 <, ..., variable-N>)
```

**Required Arguments**

- `count` specifies an integer variable that ranges from 1 to the number of permutations.
- `variable` specifies either all numeric variables, or all character variables that have the same length. The values of these variables are permuted by LEXPERM.

**Requirement** Initialize these variables before you execute the LEXPERM function.

**Details**

**Determine the Number of Distinct Permutations**

The following variables are defined for use in the equation that follows:

- `N` specifies the number of variables that are being permuted, that is, the number of arguments minus one.
- `M` specifies the number of missing values among the variables that are being permuted.
- `d` specifies the number of distinct nonmissing values among the arguments.
- `N_i` for i=1, through i=d, `N_i` specifies the number of instances of the `i`th distinct value.

The number of distinct permutations of nonmissing values of the arguments is expressed as follows:

\[
P = \frac{(N_1 + N_2 + \ldots + N_d)!}{N_1!N_2!\ldots N_d!} \leq N!
\]

**Note:** The LEXPERM function cannot be executed with the %SYSFUNC macro.

**LEXPERM Processing**

Use the LEXPERM function in a loop where the argument `count` takes each integral value from 1 to P. You do not need to compute P provided you exit the loop when LEXPERM returns a value that is less than zero.

For 1=\(count\)<P, the following actions occur:
• The argument types and lengths are checked for consistency.
• The M missing values are assigned to the last M arguments.
• The N-M nonmissing values are assigned in ascending order to the first N-M arguments following count.
• LEXPERM returns 1.

For 1<\text{count}<P, the following actions occur:
• The next distinct permutation of the nonmissing values is generated in lexicographic order.
• If \text{variable-1} through \text{variable-I} did not change, but \text{variable-J} did change, where J=I +1, then LEXPERM returns J.

For \text{count}>P, LEXPERM returns –1.

If the LEXPERM function is executed with the first argument out of sequence, the results might not be useful. In particular, if you initialize the variables and then immediately execute LEXPERM with a first argument of K, you do not get the K\text{th} permutation (except when K is 1). To get the K\text{th} permutation, you must execute LEXPERM K times. The first argument takes values from 1 through K in that exact order.

**Comparisons**

SAS provides three functions or CALL routines for generating all permutations:
• ALLPERM generates all possible permutations of the values, missing or nonmissing, of several variables. Each permutation is formed from the previous permutation by interchanging two consecutive values.
• LEXPERM generates all distinct permutations of the nonmissing values of several variables. The permutations are generated in lexicographic order.
• LEXPERK generates all distinct permutations of K of the nonmissing values of N variables. The permutations are generated in lexicographic order.

ALLPERM is the fastest of these functions and CALL routines. LEXPERK is the slowest.

**Example**

Here is an example of the LEXPERM function.

```sas
data _null_;
array x[6] $1 ('X' 'Y' 'Z' ' ' 'Z' 'Y');
nfact=fact(dim(x));
put +3 nfact=;
do i=1 to nfact;
   rc=lexperm(i, of x[*]);
   put i 5. +2 rc= +2 x[*];
   if rc<0 then leave;
end;
run;
```
SAS writes the following output to the log:

```
nfact=720
  1 rc=1   X Y Y Z Z
  2 rc=3   X Y Z Y Z
  3 rc=4   X Y Z Z Y
  4 rc=2   X Z Y Y Z
  5 rc=4   X Z Y Z Y
  6 rc=3   X Z Z Y Y
  7 rc=1   Y X Y Z Z
  8 rc=3   Y X Z Y Z
  9 rc=4   Y X Z Z Y
 10 rc=2   Y Y X Z Z
 11 rc=3   Y Y Z X Z
 12 rc=4   Y Y Z X Z
 13 rc=2   Y Z X Y Z
 14 rc=4   Y Z X Z Y
 15 rc=3   Y Z Y X Z
 16 rc=4   Y Z Y Z X
 17 rc=3   Y Z Z X Y
 18 rc=4   Y Z Z Y X
 19 rc=1   Z X Y Y Z
 20 rc=4   Z X Y Z Y
 21 rc=3   Z X Z Y Y
 22 rc=2   Z Y X Y Z
 23 rc=4   Z Y X Z Y
 24 rc=3   Z Y Y X Z
 25 rc=4   Z Y Y Z X
 26 rc=3   Z Y Z X Y
 27 rc=4   Z Y Z Y X
 28 rc=2   Z Z X Y Y
 29 rc=3   Z Z Y X Y
 30 rc=4   Z Z Y Y X
 31 rc=-1   Z Z Y Y X
```

See Also

Functions:
- “ALLPERM Function” on page 718

CALL Routines:
- “CALL ALLPERM Routine” on page 733
- “CALL RANPERK Routine” on page 829
- “CALL RANPERM Routine” on page 831

LIBNAME Function

Assigns or clears a libref for a SAS library.

Category: SAS File I/O

Restriction: This function is not valid on the CAS server.

Syntax

```
LIBNAME(libref <, SAS-library <, engine <, options>>>)
```
**Required Argument**

.libref

specifies the libref that is assigned to a SAS library.

**Tip**  
The maximum length of.libref is eight characters.

---

**Optional Arguments**

**SAS-library**

specifies the physical name of the SAS library that is associated with the libref.  
Specify this name as required by the host operating environment. This argument can be null.

**engine**

specifies the engine that is used to access SAS files opened in the data library. If you are specifying a SAS/SHARE server, then the value of engine should be REMOTE. This argument can be null.

**options**

specifies one or more valid options for the specified engine, delimited with blanks. This argument can be null.

---

**Details**

**Basic Information about Return Codes**

The LIBNAME function assigns or clears a libref from a SAS library. When you use the LIBNAME function with two or more arguments, SAS attempts to assign the libref. When you use one argument, SAS attempts to clear the libref. Return codes are generated depending on the value of the arguments that are used in the LIBNAME function and whether the libref is assigned.

When assigning a libref, the return code is 0 if the libref is successfully assigned. If the return code is nonzero and the SYSMSG function returns a warning message or a note, then the assignment was successful. If the SYSMSG function returns an error, then the assignment was unsuccessful.

If a library is already assigned, and you attempt to assign a different name to the library, the libref is assigned, the LIBNAME function returns a nonzero return code, and the SYSMSG function returns a note.

---

**When LIBNAME Has One Argument**

When LIBNAME has one argument, the following rules apply:

- If the libref is not assigned, a nonzero return code is returned and the SYSMSG function returns a warning message.
- If the libref is successfully assigned, a 0 return code is returned and the SYSMSG function returns a blank value.

---

**When LIBNAME Has Two Arguments**

When LIBNAME has two arguments, the following rules apply:

- If the second argument is null, all blanks, or zero length, SAS attempts to deassign the libref.
- If the second argument is not null, not all blanks, and not zero length, SAS attempts to assign the specified path (the second argument) to the libref.
• If the libref is not assigned, a nonzero return code is returned and the SYSMSG function returns an error message.

• If the libref is successfully assigned, a 0 return code is returned and the SYSMSG function returns a blank value.

**When LIBNAME Has Three or Four Arguments**

When LIBNAME has three or four arguments, the following rules apply:

• If the second argument is null, all blanks, or zero length, the results depend on your operating environment.

• If the second argument is null and the libref is not already assigned, then a nonzero return code is returned and the SYSMSG function returns an error message.

• If the second argument is null and the libref has already been assigned, then LIBNAME returns a value of 0 and the SYSMSG function returns a blank value.

• If at least one of the previous conditions is not met, then SAS attempts to assign the specified path (second argument) to the libref.

**Note:** In the DATA step, a character constant that consists of two consecutive quotation marks without any intervening spaces is interpreted as a single space, not as a string with a length of 0. To specify a string with a length of 0, use the [TRIM Function on page 640](#).

**Operating Environment Information**

Some systems allow a *SAS-library* value of " " (a space between the single quotation marks) to assign a libref to the current directory. Other systems clear the libref from the SAS library when the *SAS-library* value contains only blanks. The behavior of LIBNAME when a single space is specified for *SAS-library* is dependent on your operating environment. Under some operating environments, you can assign librefs by using system commands that are outside the SAS session.

**Windows Specifics**

Under Windows, if you do not specify a *SAS-library*, or if you specify a *SAS-library* as " " (a space between single quotation marks) or " " (no space between single quotation marks), SAS deassigns the libref.

**Examples**

**Example 1: Assigning a Libref**

This example attempts to assign the libref NEW to the SAS library MYLIB. If an error or warning occurs, the message is written to the SAS log. Note that in a macro statement, you do not enclose character strings in quotation marks.

```sas
%let mylib=c:\projects\May2015;
%if %sysfunc(libname(new,&mylib)) %then
  %put %sysfunc(sysmsg());
```

**Example 2: Deassigning a Libref**

This example deassigns the libref NEW that was previously associated with the data library MYLIB in the preceding example. If an error or warning occurs, the message is written to the SAS log. In a macro statement, you do not enclose character strings in quotation marks.

```sas
%if (%sysfunc(libname(new))) %then
  %put %sysfunc(sysmsg());
```
Example 3: Compressing a Library
This example assigns the libref NEW to the MYLIB data library and uses the COMPRESS option to compress the library. This example uses the default SAS engine. In a DATA step, you enclose character strings in quotation marks.

```sas
data test;
    rc=libname('new', 'MYLIB', , 'compress=yes');
run;
```

See Also

Functions:
- “LIBREF Function” on page 989
- “SYSMSG Function” on page 1074

LIBREF Function
Verifies that a libref has been assigned.

**Category:** SAS File I/O

**Restriction:** This function is not valid on the CAS server.

**Syntax**

```
LIBREF(libref)
```

**Required Argument**

`libref`

specifies the libref to be verified. In a DATA step, `libref` can be a character expression, a string enclosed in quotation marks, or a DATA step variable whose value contains the libref. In a macro, `libref` can be any expression.

**Range** 1 to 8 characters

**Details**

The LIBREF function returns 0 if the libref has been assigned, or returns a nonzero value if the libref has not been assigned.

**Example**

This example verifies a libref. If an error or warning occurs, the message is written to the log. Under some operating environments, the user can assign librefs by using system commands outside the SAS session.

```sas
%if (%sysfunc(libref(sashelp))) %then
    %put %sysfunc(sysmsg());
```
See Also

Functions:

• “LIBNAME Function” on page 986

LOGISTIC Function

Returns the logistic transformation of the argument.

Category: Mathematical

Restriction: This function is not valid on the CAS server.

Syntax

LOGISTIC(argument)

Required Argument

argument

is a numeric variable, constant, or expression that specifies the value of a numeric random variable. When argument is missing, the LOGISTIC function returns a missing value.

Details

The LOGISTIC function returns the logistic transformation of an argument. It is typically used to convert a log odds value to a value on the probability scale. The function is mathematically expressed by the following equation:

\[
\text{logistic} = \frac{e^x}{1 + e^x}
\]

If the argument contains a missing value, then the LOGISTIC function returns a missing value.

Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=0.5;</td>
<td></td>
</tr>
<tr>
<td>z=logistic(x);</td>
<td>z=0.6224593312</td>
</tr>
<tr>
<td>put z=;</td>
<td></td>
</tr>
</tbody>
</table>
Syntax

LPNORM(p, value-1 <, value-2 …>)

Required Arguments

p
specifies a numeric constant, variable, or expression that is greater than or equal to 1, which is used as the power for computing the \( L_p \) norm.

value
specifies a numeric constant, variable, or expression.

Details

If all arguments have missing values, then the result is a missing value. Otherwise, the result is the \( L_p \) norm of the nonmissing values of the second and subsequent arguments.

In the following example, \( p \) is the value of the first argument, and \( x_1, x_2, \ldots, x_n \) are the values of the other nonmissing arguments.

LPNORM(\( p \), \( x_1, x_2, \ldots, x_n \)) = (abs(\( x_1 \))^{\( p \)} + abs(\( x_2 \))^{\( p \}) + \ldots + abs(\( x_n \))^{\( p \})^{1/\( p \)}

Examples

Example 1: Calculating the \( L_p \) Norm

The following example returns the \( L_p \) norm of the second and subsequent nonmissing arguments.

```
data _null_;   
x1=lpnorm(1, ., 3, 0, .q, -4);   
x2=lpnorm(2, ., 3, 0, .q, -4);   
x3=lpnorm(3, ., 3, 0, .q, -4);   
x999=lpnorm(999, ., 3, 0, .q, -4);   
put x1= / x2= / x3= / x999=;   
run;   
```

SAS writes the following output to the log:

```
x1=7
x2=5
x3=4.4979414453
x999=4
```

Example 2: Calculating the \( L_p \) Norm When You Use a Variable List

The following example uses a variable list and returns the \( L_p \) norm.

```
data _null_;   
x1=1;   
x2=3;   
x3=4;   
```
\texttt{x4=3;}
\texttt{x5=1;}
\texttt{x=lpnorm(of x1-x5);}
\texttt{put x=;}
\texttt{run;}

SAS writes the following output to the log:

\begin{verbatim}
x=11
\end{verbatim}

**See Also**

**Functions:**
- “EUCLID Function” on page 891 (L2 norm)
- “MAX Function” on page 429 (Linfinity norm)
- “SUMABS Function” on page 623 (L1 norm)

---

### MODEEXIST Function

Determines whether a software image exists in the version of SAS that you have installed.

**Category:** Numeric

**Restriction:** This function is not valid on the CAS server.

**Operating environment:** This function is supported under UNIX and Windows.

**Syntax**

Windows and UNIX:

\texttt{MODEEXIST('product-name' | 'pathname')}

**Required Arguments**

- `'product-name'`
  - specifies a character constant, variable, or expression that is the name of the product image that you are checking.

- `'pathname'`
  - specifies the pathname for the product image that you are checking.

**Details**

When you supply a product name, the MODEEXIST function determines whether a software image exists in the version of SAS that you have installed. If an image exists, then MODEEXIST returns a value of 1. If an image does not exist, then MODEEXIST returns a value of 0.

When you supply a pathname, the MODEEXIST function searches the directories that are listed in the `pathname` argument for an executable module. The name of the executable module is passed to MODEEXIST. MODEEXIST returns 1 if the module is found, and 0 if the module is not found.
Comparisons

The MODEXIST function determines whether a software image exists in the version of SAS that you have installed. The SYSPROD function determines whether a product is licensed.

Example

This example determines whether a product is licensed and the image is installed. The example returns a value of 1 if a SAS/GRAPH image is installed in your version of SAS, and returns a value of 0 if the image is not installed. The SYSPROD function determines whether the product is licensed.

```
data _null_;  
  rc1=sysprod('graph');  
  rc2=modexist('sasgplot');  
  put rc1= rc2=;  
run;
```

SAS writes the following output to the log:

```
rc1=1 rc2=1
```

MODULE Function

Calls a specific routine or module that resides in an external dynamic link library (DLL).

**Category:** External Routines

**Restriction:** This function is not valid on the CAS server.

**CAUTION:** Be sure to use the correct arguments and attributes. Using incorrect arguments or attributes for a DLL function can cause SAS, and possibly your operating system, to fail.

**Syntax**

- `CALL MODULE(<control string>, module, argument-1, ..., argument-n);`
- `num=MODULEN(<control string>, module, argument-1, ..., argument-n);`
- `char=MODULEC(<control string>, module, argument-1, ..., argument-n);`
- `CALL MODULEI(<control string>, module, argument-1, ..., argument-n);`
- `num=MODULEIN(<control string>, module, argument-1, ..., argument-n);`
- `char=MODULEIC(<control string>, module, argument-1, ..., argument-n);`

**Required Arguments**

- `module` is the name of the external module to use, specified as a DLL name and the routine name or ordinal value, separated by a comma. The module must reside in a dynamic link library (DLL) and it must be externally callable. For example, the value `'KERNEL32,GetProfileString'` specifies to load KERNEL32.DLL and to invoke the GetProfileString routine. Note that while the DLL name is not case
sensitive, the routine name is based on the restraints of the routine's implementation language, so the routine name is case sensitive.

If the DLL supports ordinal-value naming, you can provide the DLL name followed by a decimal number, such as `'XYZ, 30'`.

You do not need to specify the DLL name if you specified the MODULE attribute for the routine in the SASCBTBL attribute table, as long as the routine name is unique (that is, no other routines have the same name in the attribute file).

You can specify `module` as a SAS character expression instead of as a constant; most often, though, you pass it as a constant.

**argument**

are the arguments to pass to the requested routine. Use the proper attributes for the arguments (numeric arguments for numeric attributes and character arguments for character attributes).

**Optional Argument**

**control-string**

is an optional control string whose first character must be an asterisk (*), followed by any combination of the following characters:

I   prints the hexadecimal representations of all arguments to the MODULE function and to the requested DLL routine before and after the DLL routine is called. You can use this option to help diagnose problems that are caused by incorrect arguments or attribute tables. If you specify the I option, the E option is implied.

E   prints detailed error messages. Without the E option (or the I option, which supersedes it), the only error message that the MODULE function generates is "Invalid argument to function," which is usually not enough information to determine the cause of the error.

Sx  uses x as a separator character to separate field definitions. You can then specify x in the argument list as its own character argument to serve as a delimiter for a list of arguments that you want to group together as a single structure. Use this option only if you do not supply an entry in the SASCBTBL attribute table. If you do supply an entry for this module in the SASCBTBL attribute table, you should use the FDSTART option in the ARG statement in the table to separate structures.

H   provides brief help information about the syntax of the MODULE routines, the attribute file format, and the suggested SAS formats and informats.

For example, the control string `'*IS/'` specifies that parameter lists be printed and that the string `/` is to be treated as a separator character in the argument list.

**Details**

The MODULE functions execute a routine `module` that resides in an external (outside SAS) dynamic link library with the specified arguments `arg-1` through `arg-n`.

The MODULE call routine does not return a value. The MODULEN and MODULEC functions return a number `num` or a character `char`, respectively. Which routine you use depends on the expected return value of the DLL function that you want to execute.

MODULEI, MODULEIC, and MODULEIN are special versions of the MODULE functions that permit vector and matrix arguments. Their return values are still scalar. You can invoke these functions only from PROC IML.
Other than this name difference, the syntax for all six routines is the same.

The MODULE function builds a parameter list by using the information in arg-1 to arg-n and by using a routine description and argument attribute table that you define in a separate file. Before you invoke the MODULE routine, you must define the fileref of SASCBTBL to point to this external file. You can name the file whatever you want when you create it.

You can use SAS variables and formats as arguments to the MODULE function and ensure that these arguments are properly converted before being passed to the DLL routine.

CALL MODULEI, MODULEIN, and MODULEIC permit vector and matrix arguments; you can use them within the IML procedure. For more information, see the SAS/IML Studio: User's Guide.

**MODULEC Function**

Calls an external routine and returns a character value.

**Category:**External Routines

**Restriction:**This function is not valid on the CAS server.

**See:**“CALL MODULE Routine” on page 775

**Syntax**

MODULEC(<cntl-string, > module-name <, argument-1, ..., argument-n>)

**Details**

For details about the MODULEC function, see “CALL MODULE Routine” on page 775.

**See Also**

**Functions:**

- “MODULEN Function” on page 995

**CALL Routines:**

- “CALL MODULE Routine” on page 775

**MODULEN Function**

Calls an external routine and returns a numeric value.

**Category:**External Routines

**Restriction:**This function is not valid on the CAS server.

**See:**“CALL MODULE Routine” on page 775
Syntax

MODULEN(<cntl-string, > module-name <, argument-1, ..., argument-n>)

Details

For details about the MODULEN function, see “CALL MODULE Routine” on page 775.

See Also

Functions:
- “MODULEC Function” on page 995

CALL Routines:
- “CALL MODULE Routine” on page 775

MOPEN Function

Opens a file by directory ID and member name, and returns either the file identifier or a 0.

Category: External Files
Restriction: This function is not valid on the CAS server.
UNIX specifics: Open-modes

Syntax

MOPEN(directory-id, member-name <, open-mode <, record-length <, record-format>>>)

Required Arguments

directory-id
is a numeric variable that specifies the identifier that was assigned when the directory was opened, generally by the DOPEN function.

member-name
is a character constant, variable, or expression that specifies the member name in the directory.

Optional Arguments

open-mode
is a character constant, variable, or expression that specifies the type of access to the file:

A  APPEND mode allows writing new records after the current end of the file.
I  INPUT mode allows reading only (default).
O  OUTPUT mode defaults to the OPEN mode specified in the operating environment option in the FILENAME statement or function. If no
operating environment option is specified, it allows writing new records at
the beginning of the file.

S  Sequential input mode is used for pipes and other sequential devices such
as hardware ports.

U  UPDATE mode allows both reading and writing.

W  Sequential Update mode is used for pipes and other sequential devices such
as ports.

Default  I

record-length

is a numeric variable, constant, or expression that specifies a new logical record
length for the file. To use the existing record length for the file, specify a length of 0,
or do not provide a value here.

record-format

is a character constant, variable, or expression that specifies a new record format for
the file. To use the existing record format, do not specify a value here. The following
values are valid:

B  specifies that data is to be interpreted as binary data.

D  specifies the default record format.

E  specifies the record format that you can edit.

F  specifies that the file contains fixed-length records.

P  specifies that the file contains printer carriage control in operating
environment-dependent record format.

V  specifies that the file contains variable-length records.

Note: If an argument is invalid, then MOPEN returns 0. You can obtain the text of
the corresponding error message from the SYSMSG function. Invalid arguments
do not produce a message in the SAS log and do not set the _ERROR_ automatic
variable.

Details

MOPEN returns the identifier for the file, or 0 if the file could not be opened. You can
use a file-id that is returned by the MOPEN function as you would use a file-id returned
by the FOPEN function.

CAUTION: Use OUTPUT mode with care. Opening an existing file for output might overwrite
the current contents of the file without warning.

The member is identified by directory-id and member-name instead of by a fileref. You
can also open a directory member by using FILENAME to assign a fileref to the
member, followed by a call to FOPEN. However, when you use MOPEN, you do not
have to use a separate fileref for each member.

If the file already exists, the output and Update modes default to the operating
environment option (append or replace) specified with the FILENAME statement or
function. For example,

%let rc=%sysfunc(filename(file, physical-name, , mod));
%let did=%sysfunc(dopen(&file));
%let fid=%sysfunc(mopen(&did, member-name, o, 0, d));
If "file" already exists, FWRITE appends the new record instead of writing it at the beginning of the file. However, if no operating environment option is specified with the FILENAME function, the output mode implies that the record be replaced.

If the open fails, use SYSMSG to retrieve the message text.

**Operating Environment Information**

The term *directory* in this description refers to an aggregate grouping of files that are managed by the operating environment. Different host operating environments identify such groupings with different names, such as directory, subdirectory, folder, MACLIB, or partitioned data set. For details, see the SAS documentation for your operating environment. Opening a directory member for output or append is not possible in some operating environments.

**Example**

This example assigns the fileref MYDIR to a directory. Then it opens the directory, determines the number of members, retrieves the name of the first member, and opens that member. The last three arguments to MOPEN are the defaults. Note that in a macro statement, you do not enclose character strings in quotation marks.

```sas
%let filrf=mydir;
%let rc=%sysfunc(filename(filrf, physical-name));
%let did=%sysfunc(dopen(&filrf));
%let firstname=' '; %let memcount=%sysfunc(dnum(&did));
%if (&memcount > 0) %then
  %do;
  %let firstname=%sysfunc(dread(&did, 1));
  %let fid=%sysfunc(mopen(&did, &firstname, i, 0, d));
  macro statements to process the member
  %let rc=%sysfunc(fclose(&fid));
  %end;
%selse
  %put %sysfunc(sysmsg());
%let rc=%sysfunc(dclose(&did));
```

**See Also**

**Functions:**
- “DCLOSE Function” on page 870
- “DNUM Function” on page 880
- “DOPEN Function” on page 881
- “DREAD Function” on page 887
- “FCLOSE Function” on page 896
- “FILENAME Function” on page 910
- “FOPEN Function” on page 925
MSPLINT Function

Returns the ordinate of a monotonicity-preserving interpolating spline.

**Category:** Mathematical

**Restriction:** This function is not valid on the CAS server.

**Syntax**

MSPLINT$(X, n, X_1 <, X_2, ..., X_n>, Y_1 <, Y_2, ..., Y_n>, D_1, D_n)$

**Required Arguments**

- $X$ is a numeric constant, variable, or expression that specifies the abscissa for which the ordinate of the spline is to be computed.
- $n$ is a numeric constant, variable, or expression that specifies the number of knots. $N$ must be a positive integer.
- $X_1, ..., X_n$ are numeric constants, variables, or expressions that specify the abscissas of the knots. These values must be nonmissing and listed in nondecreasing order. Otherwise, the result is undefined. MSPLINT does not check the order of the $X_1$ through $X_n$ arguments.
- $Y_1, ..., Y_n$ are numeric constants, variables, or expressions that specify the ordinates of the knots. The number of $Y_1$ through $Y_n$ arguments must be the same as the number of $X_1$ through $X_n$ arguments.

**Optional Argument**

- $D_1, D_n$ are optional numeric constants, variables, or expressions that specify the derivatives of the spline at $X_1$ and $X_n$. These derivatives affect only abscissas that are less than $X_2$ or greater than $X_{n-1}$.

**Details**

The MSPLINT function returns the ordinate of a monotonicity-preserving cubic interpolating spline for a single abscissa, $X$.

An interpolating spline is a function that passes through each point that is specified by the ordered pairs $(X_1, Y_1), (X_2, Y_2), ..., (X_n, Y_n)$. These points are called knots.

A spline preserves monotonicity if both of the following conditions are true:
• For any two or more consecutive knots with nondecreasing ordinates, all interpolated values within that interval are also nondecreasing.

• For any two or more consecutive knots with nonincreasing ordinates, all interpolated values within that interval are also nonincreasing.

However, if you specify values of $D_1$ or $D_n$ with the wrong sign, monotonicity is not preserved for values that are less than $X_i$ or greater than $X_{n-1}$.

If the arguments $D_1$ and $D_n$ are omitted or missing, then the following actions occur:

• For $n=1$, MSPLINT returns $Y_i$.

• For $n=2$, MSPLINT uses linear interpolation or extrapolation.

If the arguments $D_1$ and $D_n$ have nonmissing values, or if $n \geq 3$, then the following actions occur:

• If $X < X_i$ or $X > X_n$, MSPLINT uses linear extrapolation.

• If $X_i \leq X \leq X_n$, MSPLINT uses cubic spline interpolation.

If two knots have equal abscissas but different ordinates, then the spline is discontinuous at that abscissa. If two knots have equal abscissas and equal ordinates, then the spline is continuous at that abscissa, but the first derivative is usually discontinuous at that abscissa. Otherwise, the spline is continuous and has a continuous first derivative.

If $X$ is missing, or if any other arguments required to compute the result are missing, then MSPLINT returns a missing value. MSPLINT does not check all of the arguments for missing values. Because the arguments $D_1$ and $D_n$ are optional, and they are not required to compute the result, if one or both are missing and no errors occur, then MSPLINT returns a nonmissing result.

Example

Here is an example of the MSPLINT function.

```sas
data msplint;
  do x=0 to 100 by .1;
    msplint=msplint(x, 9,
      10, 20, 25, 50, 55, 70, 70, 80, 90,
      20, 30, 30, 40, 70, 60, 50, 40, 40);
    output;
  end;
run;
data knots;
  input x y;
datalines;
10 20
20 30
25 30
50 40
55 70
70 60
70 50
80 40
90 40
;
data plot;
  merge knots msplint;
```
by x;
run;
title "Comparison of Splines";
title2 "Non-monotonicity-preserving and Monotonicity-preserving Splines";
legend1 value=('Non-monotonicity-preserving spline'
'Monotonicity-preserving spline') label=none;
symbol1 value=dot interpol=spline color=black width=5;
symbol2 value=none interpol=join color=red;
proc gplot data=plot;
  plot y*x=1 msplint*x=2/overlay legend=legend1;
run;
quit;

Comparison of Splines
Non-monotonicity-preserving and Monotonicity-preserving Splines

References

MVALID Function
Checks the validity of a character string for use as a SAS member name.

**Category:** Character

**Restrictions:** This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see *Internationalization Compatibility*.

This function is not valid on the CAS server.
Syntax

**MVAlID**(*libname, string, member-type <, valid-member-name>*)

**Required Arguments**

*libname*

specifies a character constant, variable, or expression that associates a SAS library with a libref. Leading and trailing blanks are ignored.

*string*

specifies a character constant, variable, or expression that is checked to determine whether its value can be used as a SAS member name. Leading and trailing blanks are ignored.

*member-type*

specifies a character constant, variable, or expression that is the member type of the member name that you are using. Leading and trailing blanks are ignored. The value of *member-type* is not validated. The following member types are available:

- **ACCESS** specifies access descriptor files that are created by SAS/ACCESS.
- **CATALOG** specifies SAS catalogs.
- **DATA** specifies SAS data files.
- **FDB** specifies a financial database.
- **ITEMSTOR** specifies a SAS data set that consists of pieces of information that can be accessed independently. The SAS Registry is an example of an item store.
- **MDDB** specifies a multidimensional database.
- **PROGRAM** specifies stored compiled SAS programs.
- **VIEW** specifies SAS views.

**Optional Argument**

*valid-member-name*

specifies a character constant, variable, or expression. The values for *valid-member-name* can be uppercase or lowercase. Leading and trailing blanks are ignored. The following list contains the values that you can use with *valid-member-name*:

- **COMPAT**
- **COMPATIBLE** determines that *string* is a valid SAS member name when all three of the following conditions are true:
  - The *string* argument begins with an English letter or an underscore.
  - All subsequent characters are English letters, underscores, or digits.
  - The length of *string* is 32 or fewer alphanumeric characters.

- **EXTEND** determines that *string* is a valid SAS member name when all of the following conditions are true:
  - The length of *string* is 32 or fewer bytes.
  - The *string* argument does not contain the characters / \ * ? " < > | : –
Note: The SPD Engine also does not allow '" as the first character. It also
does not allow a period (.) in the member name.

• The string argument does not contain null bytes.
• The string argument does not begin with a blank or period (.).
• The string argument contains at least one character. A name that consists of
all blanks is not valid.

Default VALIDMEMNAME= is set to COMPAT.

Note If no value is specified, the MVALUE function determines that string is a
valid SAS member name based on the value of the VALIDMEMNAME=
system option.

Details

The Basics
The MVALID function checks the value of string to determine whether it can be used as
a SAS member name.

The MVALID function returns a value of 1 if string can be used as a SAS member name,
and a value of 0 if string cannot be used as a SAS member name.

MVALID returns a missing value if one of the following conditions is true:

• The libname argument is not an assigned libref.
• The member-type argument is longer than nine characters.
• The valid-member-name argument does not have one of the following values:
  COMPATIBLE, COMPAT, or EXTEND, regardless of whether the value is
  uppercase or lowercase.

Requirements for Validation of a SAS Member Name
The string argument is evaluated to determine whether it is a valid SAS member name.
An engine name with its associated library, as well as member type, affect the validation
of string. Of the member types, only DATA, ITEMSTOR, and VIEW allow names with
extended characters. When string is evaluated, the EXTEND value of the optional
valid-member-name argument is taken into account. Not all engines support
valid-member-name processing. For the engines that do not, string is validated based on
the rules for that engine.

The following example shows you how to use the MVALID function to determine
whether string is a valid SAS member name, based on engine name, DATA member
type, and the EXTEND value for valid-member-name:

```
libname V9eng V9 'mypath';
data _null_;  
   rc=MVALID('V9eng', 'my name', 'data', 'extend');
   put rc=;
run;
```

The following items apply to the preceding example:

• The example returns a value of 1, indicating that 'my name' is a valid member name
  for the V9 engine when member type equals DATA and valid-member-name equals
  EXTEND.
If you use the V6 engine in the example, the program returns a value of 0, indicating that 'my name' is not valid when member type equals DATA and valid-member-name equals EXTEND. The V6 engine does not support valid-member-name processing.

In the following example, CATALOG is used instead of DATA for member type:

```sas
libname V9eng V9 'mypath';
data _null_;    
rc=MVALID('V9eng', 'my name', 'catalog', 'extend');
put rc=;
runch;
```

The following items apply to the preceding example:

- If you use CATALOG in the example instead of DATA, the program returns a value of 0, indicating that 'my name' is not valid when member type equals CATALOG and valid-member-name equals EXTEND. The member type CATALOG does not support extended names, and therefore the EXTEND value for valid-member-name is not valid.
- If you use COMPAT in the example instead of EXTEND, the program returns a value of 0, indicating that 'my name' is not valid when member type equals CATALOG and valid-member-name equals COMPAT. The COMPAT value of valid-member-name does not allow spaces in member names.

---

**NLITERAL Function**

Converts a character string that you specify to a SAS name literal.

**Category:** Character

**Restrictions:**

This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see [Internationalization Compatibility](#).

This function is not valid on the CAS server.

**Syntax**

```sas
NLITERAL(string)
```

**Required Argument**

`string`

specifies a character constant, variable, or expression that is to be converted to a SAS name literal.

**Restriction**

If the string is a valid SAS variable name, it is not changed.

**Tip**

Enclose a literal string of characters in quotation marks.

**Details**

**Length of Returned Variable**

In a DATA step, if the NLITERAL function returns a value to a variable that has not previously been assigned a length, then the variable is given a length of 200 bytes.
The Basics

String is converted to a name literal, unless it qualifies under the default rules for a SAS variable name. These default rules are in effect when the SAS system option VALIDVARNAME=V7:

- It begins with an English letter or an underscore.
- All subsequent characters are English letters, underscores, or digits.
- The length is 32 or fewer alphanumeric characters.

String qualifies as a SAS variable name, when all of these rules are true.

The NLITERAL function encloses the value of string in single or double quotation marks, based on the contents of string.

<table>
<thead>
<tr>
<th>Value in string</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>an ampersand (&amp;)</td>
<td>enclosed in single quotation marks</td>
</tr>
<tr>
<td>a percent sign (%)</td>
<td>enclosed in single quotation marks</td>
</tr>
<tr>
<td>more double quotation marks than single quotation marks</td>
<td>enclosed in quotation marks</td>
</tr>
<tr>
<td>none of the above</td>
<td>enclosed in double quotation marks</td>
</tr>
</tbody>
</table>

If insufficient space is available for the resulting n-literal, NLITERAL returns a blank string, writes an error message, and sets _ERROR_ to 1.

Example

This example demonstrates multiple uses of NLITERAL.

```plaintext
data test;
  input string $32.;
  length result $ 67;
  result=nliteral(string);
datalines;
abc_123
This and That
cats & dogs
Company's profits (%)
"Double Quotes"
'Single Quotes'
;
proc print;
title 'Strings Converted to N-Literals or Returned Unchanged';
run;
```
Figure 3.25  Output from Converting Strings to Name Literals with NLITERAL

<table>
<thead>
<tr>
<th>Obs</th>
<th>string</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>abc_123</td>
<td>abc_123</td>
</tr>
<tr>
<td>2</td>
<td>This and That</td>
<td>&quot;This and That&quot;N</td>
</tr>
<tr>
<td>3</td>
<td>cats &amp; dogs</td>
<td>'cats &amp; dogs'N</td>
</tr>
<tr>
<td>4</td>
<td>Company's profits (%)</td>
<td>'Company''s profits (%)'N</td>
</tr>
<tr>
<td>5</td>
<td>&quot;Double Quotes&quot;</td>
<td>&quot;Double Quotes&quot;N</td>
</tr>
<tr>
<td>6</td>
<td>'Single Quotes'</td>
<td>&quot;Single Quotes&quot;N</td>
</tr>
</tbody>
</table>

See Also

Functions:
- “COMPARE Function” on page 184
- “DEQUOTE Function” on page 224
- “NVALID Function” on page 1008

System Options:
- “VALIDVARNAME= System Option” in SAS Viya System Options: Reference

NORMAL Function

Returns a random variate from a normal, or Gaussian, distribution.

Category:  Random Number
Alias:     RANNOR
Restriction:  This function is not valid on the CAS server.
See:       “RANNOR Function” on page 1035

NOTE Function

Returns an observation ID for the current observation of a SAS data set.

Category:  SAS File I/O
Restriction:  This function is not valid on the CAS server.
Syntax

\texttt{NOTE(data-set-id)}

\textbf{Required Argument}

\textit{data-set-id} is a numeric variable that specifies the data set identifier that the OPEN function returns.

\textbf{Details}

You can use the observation ID value to return to the current observation by using \texttt{POINT}. Observations can be marked by using \texttt{NOTE} and then returned to later by using \texttt{POINT}. Each observation ID is a unique numeric value.

To free the memory that is associated with an observation ID, use \texttt{DROPNOTE}.

\textbf{Example}

This example calls \texttt{CUROBS} to display the observation number, calls \texttt{NOTE} to mark the observation, and calls \texttt{POINT} to point to the observation that corresponds to \texttt{NOTEID}.

```
%let dsid=%sysfunc(open(sasuser.fitness,i));
/* Go to observation 10 in data set */
%let rc=%sysfunc(fetchobs(&dsid, 10));
%if %sysfunc(abs(&rc)) %then
  %put FETCHOBS FAILED;
%else
  %do;
    /* Display observation number */
    /* in the Log */
    %let cur=%sysfunc(curobs(&dsid));
    %put CUROBS=&cur;
    /* Mark observation 10 using NOTE */
    %let noteid=%sysfunc(note(&dsid));
    /* Rewind pointer to beginning */
    /* of data */
    /* set using REWIND */
    %let rc=%sysfunc(rewind(&dsid));
    /* FETCH first observation into DDV */
    %let rc=%sysfunc(fetch(&dsid));
    /* Display first observation number */
    %let cur=%sysfunc(curobs(&dsid));
    %put CUROBS=&cur;
    /* POINT to observation 10 marked */
    /* earlier by NOTE */
    %let rc=%sysfunc(point(&dsid, &noteid));
    /* FETCH observation into DDV */
    %let rc=%sysfunc(fetch(&dsid));
    /* Display observation number */
    /* marked by NOTE */
    %let cur=%sysfunc(curobs(&dsid));
    %put CUROBS=&cur;
  %end;
%if (&dsid > 0) %then
  %let rc=%sysfunc(close(&dsid));
```
SAS writes the following output to the log:

```
CUROBS=10
CUROBS=1
CUROBS=10
```

See Also

Functions:
- “DROPNOTE Function” on page 888
- “OPEN Function” on page 1010
- “POINT Function” on page 1019
- “REWIND Function” on page 1043

NVALID Function

Checks the validity of a character string for use as a SAS variable name.

- **Category**: Character
- **Restrictions**: This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see *Internationalization Compatibility*. This function is not valid on the CAS server.

**Syntax**

```
NVALID(string <, validvarname>)
```

**Required Argument**

- `string`
  - specifies a character constant, variable, or expression that is checked to determine whether its value can be used as a SAS variable name.
  - *Note*: Trailing blanks are ignored.
  - *Tip*: Enclose a literal string of characters in quotation marks.

**Optional Argument**

- `validvarname`
  - is a character constant, variable, or expression that specifies one of the following values:
    - `V7`
      - determines that `string` is a valid SAS variable name when all three of the following are true:
        - `string` begins with an English letter or an underscore
        - All subsequent characters are English letters, underscores, or digits
• The length is 32 or fewer alphanumeric characters

ANY
determines that string is a valid SAS variable name if it contains 32 or fewer characters of any type, including blanks.

NLITERAL
determines that string is a valid SAS variable name if it is in the form of a SAS name literal (‘name’N) or if it is a valid SAS variable name when VALIDVARNAME=V7.

See V7 above in this same list.

Default If no value is specified, the NVALID function determines that string is a valid SAS variable name based on the value of the SAS system option VALIDVARNAME=.

Details

The NVALID function checks the value of string to determine whether it can be used as a SAS variable name.

The NVALID function returns a value of 1 or 0.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Returned Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>string can be used as a SAS variable name</td>
<td>1</td>
</tr>
<tr>
<td>string cannot be used as a SAS variable name</td>
<td>0</td>
</tr>
</tbody>
</table>

Example

This example determines the validity of specified strings as SAS variable names. The value that is returned by the NVALID function varies with the validvarname argument. The value of 1 is returned when the string is determined to be a valid SAS variable name under the rules for the specified validvarname argument. Otherwise, the value of 0 is returned.

options validvarname=v7 ls=64;
data string;
  input string $char40.;
v7=nvalid(string, 'v7');
any=nvalid(string, 'any');
 nliteral=nvalid(string, 'nliteral');
default=nvalid(string);
datelines;
Tooooooooooooooooooooooooooo Long
OK
Very_Long_But_Still_OK_for_V7
1st_char_is_a_digit
Embedded blank
!@#$%^&*
"Very Loooong N-Literal with ""N
'No closing quotation mark
;
proc print noobs;
title1 'NLITERAL and Validvarname Arguments Determine';
title2 'Invalid (0) and Valid (1) SAS Variable Names';
run;

Figure 3.26  Output from Determining the Validity of SAS Variable Names with NLITERAL

### NLITERAL and Validvarname Arguments Determine
Invalid (0) and Valid (1) SAS Variable Names

<table>
<thead>
<tr>
<th>string</th>
<th>v7</th>
<th>any</th>
<th>nliteral</th>
<th>default</th>
</tr>
</thead>
<tbody>
<tr>
<td>T000000000000000000000000000000000000000000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>OK</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Very_Long_But_Still_OK_for_V7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1st_char_is_a_digit</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Embedded blank</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>!@#$%^&amp;*</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>&quot;Very Looooong N-Literal with ####N</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>'No closing quotation mark</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

See Also

**Functions:**

- “COMPARE Function” on page 184
- “NLITERAL Function” on page 1004

**System Options:**

- “VALIDVARNAME= System Option” in *SAS Viya System Options: Reference*

---

OPEN Function

Opens a SAS data set.

- **Category:** SAS File I/O
- **Restriction:** This function is not valid on the CAS server.

**Syntax**

```
OPEN(<data-set-name <, mode <, generation-number <, type>>>)
```
**Optional Arguments**

`data-set-name`  
is a character constant, variable, or expression that specifies the name of the SAS data set or SAS SQL view to be opened. The value of this character string should be of the form  

`<libref.> member-name<(data-set-options)>`  

**Default**  
The default value for `data-set-name` is `_LAST_`.  

**Restriction**  
If you specify the FIRSTOBS= and OBS= data set options, they are ignored. All other data set options are valid.

`mode`  
is a character constant, variable, or expression that specifies the type of access to the data set:

- `I` opens the data set in INPUT mode (default). Values can be read but not modified. `'I'` uses the strongest access mode available in the engine. That is, if the engine supports random access, OPEN defaults to random access. Otherwise, the file is opened in `'IN'` mode automatically. Files are opened with sequential access and a system-level warning is set.
- `IN` opens the data set in INPUT mode. Observations are read sequentially, and you are allowed to revisit an observation.
- `IS` opens the data set in INPUT mode. Observations are read sequentially, but you are not allowed to revisit an observation.

**Default**  
`I`

`generation-number`  
specifies a consistently increasing number that identifies one of the historical versions in a generation group.

**Tip**  
The `generation-number` argument is ignored if `type=F`.

`type`  
is a character constant and can be one of the following values:

- `D` specifies that the first argument, `data-set-name`, is a one-level or two-level data set name. The following example shows how the D `type` value can be used:

  ```
  rc=open('lib.mydata', , , 'D');
  ```

  **Tip**  
  D is the default if there is no fourth argument.

- `F` specifies that the first argument, `data-set-name`, is a filename, a physical path to a file. The following examples show how the F `type` value can be used:

  ```
  rc=open('c:\data\mydata.sas7bdat', , , 'F');
  rc=open('c:\data\mydata', , , 'F');
  ```

  **Tip**  
  If you use the F `value`, then the third argument, `generation-number`, is ignored.
Note  If an argument is invalid, OPEN returns 0. You can obtain the text of the corresponding error message from the SYSMSG function. Invalid arguments do not produce a message in the SAS log and do not set the _ERROR_ automatic variable.

Details
The OPEN function opens a SAS data set, DATA step, or a SAS SQL view and returns a unique numeric data set identifier, which is used in most other data set access functions. OPEN returns 0 if the data set could not be opened.

If you call the OPEN function from a macro, then the result of the call is valid only when the result is passed to functions in a macro. If you call the OPEN function from the DATA step, then the result is valid only when the result is passed to functions in the same DATA step.

By default, a SAS data set is opened with a control level of RECORD. For more information, see “CNTLLEV= Data Set Option” in SAS Viya Data Set Options: Reference. An open SAS data set should be closed when it is no longer needed. If you open a data set within a DATA step, it closes automatically when the DATA step ends.

OPEN defaults to the strongest access mode available in the engine. That is, if the engine supports random access, OPEN defaults to random access. Otherwise, data sets are opened with sequential access, and a system-level warning is set.

Example
• This example opens the data set PRICES in the library MASTER using INPUT mode. Note that in a macro statement, you do not enclose character strings in quotation marks.

```
%let dsid=%sysfunc(open(master.prices, i));
%if (&dsid=0) %then
  %put %sysfunc(sysmsg());
%else
  %put PRICES data set has been opened;
```

• This example passes values from macro or DATA step variables to be used on data set options. It opens the data set Sasuser.Houses, and uses the WHERE= data set option to apply a permanent WHERE clause. Note that in a macro statement, you do not enclose character strings in quotation marks.

```
%let choice = style="RANCH";
%let dsid=%sysfunc(open(sasuser.houses(where=(&choice)), i));
```

• This example shows how to check the returned value for errors and to write an error message from the SYSMSG function.

```
data _null_
  d=open('bad', '?');
  if not d then do;
    m=sysmsg();
    put m;
    abort;
  end;
  ... more SAS statements ...
run;
```
PATHNAME Function

Returns the physical name of an external file or a SAS library, or returns a blank.

**Categories:** SAS File I/O
External Files

**Restriction:** This function is not valid on the CAS server.

**UNIX specifics:** `fileref` or `libref` argument can also specify a UNIX environment variable.

### Syntax

```
PATHNAME((fileref | libref) <, search-ref>)
```

### Required Arguments

- **`fileref`**
  - is a character constant, variable, or expression that specifies the fileref that is assigned to an external file.

  **Operating Environment Information**
  - In a DATA step, `fileref` can be a character expression, a string enclosed in quotation marks, or a DATA step variable whose value contains the fileref. In macro code, `fileref` can be any expression that resolves to a macro variable. The value of `fileref` can be a WINDOWS or UNIX environment variable.

- **`libref`**
  - is a character constant, variable, or expression that specifies the libref that is assigned to a SAS library.

  **Operating Environment Information**
  - In a DATA step, `libref` can be a character expression, a string enclosed in quotation marks, or a DATA step variable whose value contains the libref. In macro code, `libref` can be any expression. The value of `libref` can be a WINDOWS or UNIX environment variable.

### Optional Argument

- **`search-ref`**
  - is a character constant, variable, or expression that specifies whether to search for a fileref or a libref.

  - F specifies a search for a `fileref`.
  - L specifies a search for a `libref`.

### See Also

**Functions:**
- “CLOSE Function” on page 854
- “SYMSMSG Function” on page 1074
Details

PATHNAME returns the physical name of an external file or SAS library, or blank if fileref or libref is invalid.

If the name of a fileref is identical to the name of a libref, you can use the search-ref argument to choose which reference you want to search. If you specify a value of F, SAS searches for a fileref. If you specify a value of L, SAS searches for a libref.

If you do not specify a search-ref argument, and the name of a fileref is identical to the name of a libref, PATHNAME searches first for a libref. If a libref does not exist, PATHNAME then searches for a fileref.

The default length of the target variable in the DATA step is 200 characters.

You can assign a fileref to an external file by using the FILENAME statement or the FILENAME function.

You can assign a libref to a SAS library using the LIBNAME statement or the LIBNAME function. Some operating environments enable you to assign a libref using system commands.

Example

This example uses the FILEREF function to verify that the fileref MYFILE is associated with an external file. Then it uses PATHNAME to retrieve the actual name of the external file:

```sas
data _null_
  length fname $ 100;
  rc=fileref('myfile');
  if (rc=0) then do;
    fname=pathname('myfile');
    put fname=;
  end;
run;
```

See Also

Functions:

- “FEXIST Function” on page 906
- “FILEEXIST Function” on page 909
- “FILENAME Function” on page 910
- “FILEREF Function” on page 913

Statements:

- “FILENAME Statement” in SAS Viya Statements: Reference
- “LIBNAME Statement” in SAS Viya Statements: Reference

PEEK Function

Stores the contents of a memory address in a numeric variable on a 32-bit platform.
Category: Special
Restrictions: Use on 32-bit platforms only.
This function is not valid on the CAS server.
Interaction: When a SAS server is in a locked-down state, the PEEK function does not execute.

Syntax

PEEK(address <, length>)

Required Argument

address

is a numeric constant, variable, or expression that specifies the memory address.

Optional Argument

length

is a numeric constant, variable, or expression that specifies the data length.

Default  a 4-byte address pointer

Range 2–8

Details

If you do not have access to the memory storage location that you are requesting, the PEEK function returns an invalid argument error.

If you attempt to use the PEEK function on 64-bit platforms, SAS writes a message to the log stating that this restriction applies. If you have legacy applications that use PEEK, change the applications and use PEEKLONG instead. You can use PEEKLONG on 32-bit and 64-bit platforms.

Comparisons

The PEEK function stores the contents of a memory address into a numeric variable. The PEEKC function stores the contents of a memory address into a character variable.

Note: As a best practice, use PEEKLONG instead of PEEK because PEEKLONG can be used on 32-bit and 64-bit platforms.

See Also

Functions:

- “ADDR Function” on page 714
- “PEEKC Function” on page 1016

CALL Routines:

- “CALL POKE Routine” on page 780
PEEKC Function
Stores the contents of a memory address in a character variable on a 32-bit platform.

**Category:** Special

**Restrictions:**
- Use on 32-bit platforms only.
- This function is not valid on the CAS server.

**Interaction:**
- When a SAS server is in a locked-down state, the PEEKC function does not execute.

**Syntax**

```
PEEKC(address <, length>)
```

**Required Argument**

*address*

is a numeric constant, variable, or expression that specifies the memory address.

**Optional Argument**

*length*

is a numeric constant, variable, or expression that specifies the data length.

- **Default**
  - 8, unless the variable length has already been set (for example, by the LENGTH statement)

- **Range**
  - 1–32,767

**Details**

If you do not have access to the memory storage location that you are requesting, the PEEKC function returns an *Invalid argument* error.

If you attempt to use the PEEKC function on 64-bit platforms, SAS writes a message to the log stating that this restriction applies. If you have legacy applications that use PEEKC, change the applications and use PEEKCLONG instead. You can use PEEKCLONG on 32-bit and 64-bit platforms.

**Comparisons**

The PEEKC function stores the contents of a memory address into a *character* variable. The PEEK function stores the contents of a memory address into a *numeric* variable.

*Note:* As a best practice, use PEEKCLONG instead of PEEKC because PEEKCLONG can be used on 32-bit and 64-bit platforms.

**See Also**

**Functions:**
- “ADDR Function” on page 714
- “PEEK Function” on page 1014
CALL Routines:
- “CALL POKE Routine” on page 780

PEEKCLONG Function
Stores the contents of a memory address in a character variable on 32-bit and 64-bit platforms.

- **Category:** Special
- **Restriction:** This function is not valid on the CAS server.
- **Interaction:** When a SAS server is in a locked-down state, the PEEKCLONG function does not execute.

**Syntax**

```
PEEKCLONG(address <, length>)
```

**Required Argument**

- `address`
  - Specifies a character constant, variable, or expression that contains the binary pointer address.

**Optional Argument**

- `length`
  - Is a numeric constant, variable, or expression that specifies the length of the character data.
    - **Default:** 8
    - **Range:** 1–32,767

**Details**

If you do not have access to the memory storage location that you are requesting, the PEEKCLONG function returns an `Invalid argument` error.

**Comparisons**

The PEEKCLONG function stores the contents of a memory address in a `character` variable.

The PEEKLONG function stores the contents of a memory address in a `numeric` variable. The function assumes that the input address refers to an integer in memory.

**Example: Example for a 32-bit Platform**

This example returns the pointer address for the character variable Z.

```
data _null_;  
x='ABCDE';  
y=addrlong(x);  
z=peekclong(y, 2);```

put z=;
run;

SAS writes the following results to the log:

\[
\text{z=AB}
\]

See Also

Functions:
- “PEEKLONG Function” on page 1018

PEEKLONG Function

Stores the contents of a memory address in a numeric variable on 32-bit and 64-bit platforms.

**Category:** Special  
**Restriction:** This function is not valid on the CAS server.  
**Interaction:** When a SAS server is in a locked-down state, the PEEKLONG function does not execute.  
**See:** “PEEKCLONG Function” on page 1017  
**CAUTION:** The PEEKLONG functions can directly access memory addresses. Improper use of the PEEKLONG functions can cause SAS, and your operating system, to fail. Use the PEEKLONG functions only to access information that is returned by one of the MODULE functions.

**Syntax**

**PEEKLONG**\((address <, \text{length}>)\)

**Required Argument**

\(address\)

specifies a character constant, variable, or expression that contains the binary pointer address.

**Optional Argument**

\(length\)

is a numeric constant, variable, or expression that specifies the length of the character data.

<table>
<thead>
<tr>
<th>Default</th>
<th>4 on 32-bit computers; 8 on 64-bit computers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>1–4 on 32-bit computers; 1–8 on 64-bit computers</td>
</tr>
</tbody>
</table>

**Details**

If you do not have access to the memory storage location that you are requesting, the PEEKLONG function returns an Invalid argument error.
Comparisons

The PEEKLONG function stores the contents of a memory address in a numeric variable. The function assumes that the input address refers to an integer in memory.

The PEEKCLONG function stores the contents of a memory address in a character variable. The function assumes that the input address refers to character data.

Usually, when you need to use one of the PEEKLONG functions, you use PEEKCLONG to access a character string.

Example: Example for a 32-bit Platform

This example returns the pointer address for the numeric variable Z.

```sas
data _null_
  length y $4;
  y=put(1, IB4.);
  addry=addrlong(y);
  z=peeklong(addry, 4);
  put z=;
run;
```

SAS writes the following results to the log:

```
  z=1
```

See Also

Functions:
- “PEEKCLONG Function” on page 1017

POINT Function

Locates an observation that is identified by the NOTE function.

- **Category:** SAS File I/O
- **Restriction:** This function is not valid on the CAS server.

**Syntax**

```
POINT(data-set-id, note-id)
```

**Required Arguments**

- **data-set-id**
  - is a numeric variable that specifies the data set identifier that the OPEN function returns.

- **note-id**
  - is a numeric variable that specifies the identifier assigned to the observation by the NOTE function.
**Details**

POINT returns 0 if the operation was successful, ≠0 if it was not successful. POINT prepares the program to read from the SAS data set. The Data Set Data Vector is not updated until a read is done using FETCH or FETCHOBS.

**Example**

This example calls NOTE to obtain an observation ID for the most recently read observation of the SAS data set MYDATA. It calls POINT to point to that observation, and calls FETCH to return the observation marked by the pointer.

```sas
%let dsid=%sysfunc(open(mydata, i));
%let rc=%sysfunc(fetch(&dsid));
%let noteid=%sysfunc(note(&dsid));
...more macro statements...
%let rc=%sysfunc(point(&dsid, &noteid));
%let rc=%sysfunc(fetch(&dsid));
...more macro statements...
%let rc=%sysfunc(close(&dsid));
```

**See Also**

Functions:
- “DROPNOTE Function” on page 888
- “NOTE Function” on page 1006
- “OPEN Function” on page 1010

---

**PROPCase Function**

Converts all words in an argument to proper case.

<table>
<thead>
<tr>
<th>Category:</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restrictions:</td>
<td>This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility.</td>
</tr>
<tr>
<td></td>
<td>This function is not valid on the CAS server.</td>
</tr>
</tbody>
</table>

**Syntax**

PROPCase(argument <, delimiters>)

**Required Argument**

argument

specifies a character constant, variable, or expression.
Optional Argument

delimiter

delimiter specifies one or more delimiters that are enclosed in quotation marks. The default delimiters are blank, forward slash, hyphen, open parenthesis, period, and tab.

Tip If you use this argument, then the default delimiters, including the blank, are no longer in effect.

Details

Length of Returned Variable

In a DATA step, if the PROPCASE function returns a value to a variable that has not previously been assigned a length, then that variable is given the length of the first argument that is passed to PROPCASE.

The Basics

The PROPCASE function copies a character argument and converts all uppercase letters to lowercase letters. It then converts to uppercase the first character of a word that is preceded by a blank, forward slash, hyphen, open parenthesis, period, or tab. PROPCASE returns the value that is altered.

If you use the second argument, then the default delimiters are no longer in effect.

Examples

Example 1: Changing the Case of Words

The following example shows how PROPCASE handles the case of words:

```sas
data _null_;  
  input place $ 1-40;  
  name=propcase(place);  
  put name;  
  datalines;  
  INTRODUCTION TO THE SCIENCE OF ASTRONOMY  
  VIRGIN ISLANDS (U.S.)  
  SAINT KITTS/NEVIS  
  WINSTON-SALEM, N.C.  
  ;  
run;
```

SAS writes the following output to the log:

```
Introduction To The Science Of Astronomy  
Virgin Islands (U.S.)  
Saint Kitts/Nevis  
Winston-Salem, N.C.
```

Example 2: Using a Second Argument with PROPCASE

The following example uses a blank, a hyphen, and a single quotation mark as the second argument so that names such as O’Keeffe and Burne-Jones are written correctly.

```sas
data names;  
  infile datalines dlm='#';  
  input CommonName : $20. CapsName : $20.;
```
PropcaseName=propcase(capsname, " -'");

datalines;
Delacroix, Eugene# EUGENE DELACROIX
O’Keeffe, Georgia# GEORGIA O’KEEFFE
Rockwell, Norman# NORMAN ROCKWELL
Burne-Jones, Edward# EDWARD BURNE-JONES
;
proc print data=names noobs;
  title 'Names of Artists';
run;

Figure 3.27  Output Showing the Results of Using PROPCASE with a Second Argument

<table>
<thead>
<tr>
<th>CommonName</th>
<th>CapsName</th>
<th>PropcaseName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delacroix, Eugene</td>
<td>EUGENE DELACROIX</td>
<td>Eugene Delacroix</td>
</tr>
<tr>
<td>O’Keeffe, Georgia</td>
<td>GEORGIA O’KEEFFE</td>
<td>Georgia O’Keeffe</td>
</tr>
<tr>
<td>Rockwell, Norman</td>
<td>NORMAN ROCKWELL</td>
<td>Norman Rockwell</td>
</tr>
<tr>
<td>Burne-Jones, Edward</td>
<td>EDWARD BURNE-JONES</td>
<td>Edward Burne-Jones</td>
</tr>
</tbody>
</table>

See Also

Functions:
- “LOWCASE Function” on page 423
- “UPCASE Function” on page 644

PRXPAREN Function

Returns the last bracket match for which there is a match in a pattern.

Category:  Character String Matching

Restrictions:  This function is assigned an I18N Level 0 status, and is designed for SBCS data. Do not use this function to process DBCS or MBCS data. For more information, see Internationalization Compatibility.

This function is not valid on the CAS server.

Use with the PRXPARSE function.

Do not use this function to process DBCS and MBCS data, because this routine requires the PRXPARSE function, which is not DBCS compatible.

Syntax

PRXPAREN(regular-expression-id)
**Required Argument**

`regular-expression-id`

specifies a numeric variable with a value that is an identification number that is returned by the PRXPARSE function.

**Details**

The PRXPAREN function is useful in finding the largest capture-buffer number that can be passed to the CALL PRXPOSN routine, or in identifying which part of a pattern matched.

For more information about pattern matching, see “Pattern Matching Using Perl Regular Expressions (PRX)” on page 42.

**Comparisons**

The Perl regular expression (PRX) functions and CALL routines work together to manipulate strings that match patterns. To see a list and short description of these functions and CALL routines, see the Character String Matching category in “SAS Functions and CALL Routines by Category” on page 701.

**Example**

The following example uses Perl regular expressions and writes the results to the SAS log.

```sas
data _null_;  
  ExpressionID=prxparse('/(magazine)|(book)|(newspaper)/');  
  position=prxmatch(ExpressionID, 'find book here');  
  if position then paren=prxparen(ExpressionID);  
  put 'Matched paren ' paren;  
  position=prxmatch(ExpressionID, 'find magazine here');  
  if position then paren=prxparen(ExpressionID);  
  put 'Matched paren ' paren;  
  position=prxmatch(ExpressionID, 'find newspaper here');  
  if position then paren=prxparen(ExpressionID);  
  put 'Matched paren ' paren;  
run;
```

SAS writes the following output to the log:

```
Matched paren 2  
Matched paren 1  
Matched paren 3
```

**See Also**

**Functions:**

- “PRXCHANGE Function” on page 527
- “PRXMATCH Function” on page 533
- “PRXPARSE Function” on page 538
- “PRXPOSN Function” on page 541

**CALL Routines:**
**PTRLONGADD Function**

Returns the pointer address as a character variable on 32-bit and 64-bit platforms.

**Category:** Special

**Restriction:** This function is not valid on the CAS server.

**Syntax**

\[ \text{PTRLONGADD}(\text{pointer}<, \text{amount}>) \]

**Required Arguments**

- **pointer** is a character constant, variable, or expression that specifies the pointer address.

- **amount** is a numeric constant, variable, or expression that specifies the amount to add to the address.

  **Tip**  \( \text{amount} \) can be a negative number.

**Details**

The PTRLONGADD function performs pointer arithmetic and returns a pointer address as a character string.

**Example**

The following example returns the pointer address for the variable \( Z \).

```sas
data _null_;
  x='ABCDE';
  y=ptrlongadd(addrlong(x), 2);
  z=peekclong(y, 1);
  put z=;
run;
```

SAS writes the following output to the log:

```
z=C
```
PUTC Function

Enables you to specify a character format at run time.

**Category:** Special

**Restrictions:** This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see [Internationalization Compatibility](#).

This function is not valid on the CAS server.

**Syntax**

```plaintext
PUTC(value, format-specification <, w>)
```

**Required Arguments**

- `value` specifies a character value to be formatted.
- `format-specification` is a character format that you want to apply to `value`.

Here are valid format forms:

- `format-name`
- `format-name.
- `format-namew`.

Except for `format-name`, you can use –L, –R, and –C in `format-specification` to left-align, right-align, and center your output. For example, you can use `upcase.-c` as the value for the second argument, `format-specification`.

**Optional Argument**

- `w` is a numeric constant, variable, or expression that specifies a width to apply to the format.

**Interaction**

If you specify a width here, it overrides any width specification in the format.

**Details**

If the PUTC function returns a value to a variable that has not yet been assigned a length, then the variable length is determined by the length of the first argument.

**Comparisons**

The PUTN function enables you to specify a numeric format at run time.

The PUT function is faster than PUTC because PUT enables you to specify a format at compile time rather than at run time.
Examples

**Example 1: Working with the PUTC Function**
The PROC FORMAT step creates a format, TYPEFMT., that formats the variable values 1, 2, and 3 with the name of one of the three other formats that this step creates. These three formats output responses of "positive," "negative," and "neutral" as different words, depending on the type of question. After PROC FORMAT creates the formats, the DATA step creates a SAS data set from raw data that consists of a number identifying the type of question and a response. After reading a record, the DATA step uses the value of TYPE to create a variable, RESPFMT. This variable contains the value of the appropriate format for the current type of question. The DATA step also creates another variable, WORD, whose value is the appropriate word for a response. The PUTC function assigns the value of WORD based on the type of question and the appropriate format.

```sas
proc format;
  value typefmt 1='$groupx'
    2='$groupy'
    3='$groupz';
  value $groupx 'positive'='agree'
    'negative'='disagree'
    'neutral'='notsure';
  value $groupy 'positive'='accept'
    'negative'='reject'
    'neutral'='possible';
  value $groupz 'positive'='pass'
    'negative'='fail'
    'neutral'='retest';
run;

data answers;
  length word $ 8;
  input type response $;
  respfmt=put(type, typefmt.);
  word=putc(response, respfmt);
  datalines;
1 positive
1 negative
1 neutral
2 positive
2 negative
2 neutral
3 positive
3 negative
3 neutral
;

proc print data=answers;
  title 'Using the Third Argument as a Number or Cycle';
run;
```
Figure 3.28 Output from the PUTC Function

The value of the variable WORD is agree for the first observation. The value of the variable WORD is retest for the last observation.

Example 2: Aligning Character Values
This example shows how to use a format and an alignment character to align the value of A.

```plaintext
data _null_;  
length a $20;  
a='experiment';  
y=putc(a,'upcase.-r',20);  
put '*' y $char20. '*';  
put '*' a $upcase20. '*';  
run;
```

* * EXPERIMENT* 
* EXPERIMENT *

See Also

Functions:
- “INPUT Function” on page 334
- “INPUTC Function” on page 337
- “INPUTN Function” on page 339
- “PUT Function” on page 545
- “PUTN Function” on page 1028
PUTN Function

Enables you to specify a numeric format at run time.

Category: Special
Restrictions: This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility.
This function is not valid on the CAS server.

Syntax

PUTN(value, format-specification <, w <, d>>)

Required Arguments

value
specifies a numeric value to be formatted.

format-specification
is the numeric format that you want to apply to value.
Here are valid format forms:
• format-name
• format-name.
• format-namew.
• format-namew.d
Except for format-name, you can use –L, –R, and –C in format-specification to left-align, right-align, and center your output. For example, you can use ‘weekdate.-c’ as the value for the second argument, format-specification.

Optional Arguments

w
is a numeric constant, variable, or expression that specifies a width to apply to the format.

Interaction If you specify a width here, it overrides any width specification in the format.

d
is a numeric constant, variable, or expression that specifies the number of decimal places to use.

Interaction If you specify a number here, it overrides any decimal-place specification in the format.

Details

If the PUTN function returns a value to a variable that has not yet been assigned a length, by default the variable is assigned a length of 200 bytes.
Comparisons

The PUTC function enables you to specify a character format at run time.

The PUT function is faster than PUTN because PUT enables you to specify a format at compile time rather than at run time.

Examples

Example 1: Working with Dates and Formats in the PUTN Function

The PROC FORMAT step creates a format, WRITFMT., that formats the variable values 1 and 2 with the name of a SAS date format. The DATA step creates a SAS data set from raw data that consists of a number and a key. After reading a record, the DATA step uses the value of KEY to create a variable, DATEFMT, that contains the value of the appropriate date format. The DATA step also creates a new variable, DATE, whose value is the formatted value of the date. PUTN assigns the value of DATE based on the value of NUMBER and the appropriate format.

```sas
proc format;
  value writfmt 1='date9.'
    2='mmdyy10.';
run;
data dates;
  input number key;
  datefmt=put(key,writfmt.);
  date=putn(number,datefmt);
datalines;
15756 1
14552 2
;
proc print data=dates;
  title 'Working with Dates and Formats';
run;
```

Figure 3.29  Output from Using the PUTN Function

<table>
<thead>
<tr>
<th>Obs</th>
<th>number</th>
<th>key</th>
<th>datefmt</th>
<th>date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15756</td>
<td>1</td>
<td>date9.</td>
<td>20FEB2003</td>
</tr>
<tr>
<td>2</td>
<td>14552</td>
<td>2</td>
<td>mmdyy10.</td>
<td>11/04/1999</td>
</tr>
</tbody>
</table>

Example 2: Aligning Output from the PUTN Function

This example shows how to use a format and an alignment character to left-align a value.

```sas
data _null_;
  length y $30;
  y=putn(today(),'weekdate.-l',30);
  put '*' y $char30. '*';
```
run;

Log 3.2  Alignment Results

*Monday, November 19, 2012  *

See Also

Functions:
• “INPUT Function” on page 334
• “INPUTC Function” on page 337
• “INPUTN Function” on page 339
• “PUT Function” on page 545
• “PUTC Function” on page 1025

RANBIN Function

Returns a random variate from a binomial distribution.

Category: Random Number

Restriction: This function is not valid on the CAS server.

Tip: If you want to change the seed value during execution, you must use the CALL RANBIN routine instead of the RANBIN function.

Syntax

RANBIN(seed, n, p)

Required Arguments

seed

is a numeric constant, variable, or expression with an integer value. If seed ≤ 0, the time of day is used to initialize the seed stream.

Range  seed < 2^{31}−1

See “Seed Values” on page 11 for more information about seed values

n

is a numeric constant, variable, or expression with an integer value that specifies the number of independent Bernoulli trials parameter.

Range  n > 0

p

is a numeric constant, variable, or expression that specifies the probability of success.
Details

The RANBIN function returns a variate that is generated from a binomial distribution with mean $np$ and variance $np(1-p)$. If $n \leq 50$, $np \leq 5$, or $n(1-p) \leq 5$, an inverse transform method applied to a RANUNI uniform variate is used. If $n > 50$, $np > 5$, and $n(1-p) > 5$, the normal approximation to the binomial distribution is used. In that case, the Box-Muller transformation of RANUNI uniform variates is used.

For a discussion about seeds and streams of data, as well as examples of using the random-number functions, see “Generating Multiple Variables from One Seed in Random-Number Functions” on page 22.

Comparisons

The CALL RANBIN routine, an alternative to the RANBIN function, gives greater control of the seed and random number streams.

See Also

Functions:

- “RAND Function” on page 555

CALL Routines:

- “CALL RANBIN Routine” on page 794

RANCAU Function

Returns a random variate from a Cauchy distribution.

Category: Random Number

Restriction: This function is not valid on the CAS server.

Tip: If you want to change the seed value during execution, you must use the CALL RANCAU routine instead of the RANCAU function.

Syntax

RANCAU(seed)

Required Argument

seed

is a numeric constant, variable, or expression with an integer value. If $seed \leq 0$, the time of day is used to initialize the seed stream.

Range $seed < 2^{31} - 1$

See “Seed Values” on page 11 for more information about seed values.
The RANCAU function returns a variate that is generated from a Cauchy distribution with location parameter 0 and scale parameter 1. An acceptance-rejection procedure applied to RANUNI uniform variates is used. If $u$ and $v$ are independent uniform $(-1/2, 1/2)$ variables and $u^2 + v^2 \leq 1/4$, then $u/v$ is a Cauchy variate. A Cauchy variate $X$ with location parameter $\text{ALPHA}$ and scale parameter $\text{BETA}$ can be generated:

\[
x = \text{ALPHA} + \text{BETA} \times \text{RANCAU(seed)};
\]

For a discussion about seeds and streams of data, as well as examples of using the random-number functions, see “Generating Multiple Variables from One Seed in Random-Number Functions” on page 22.

**Comparisons**

The CALL RANCAU routine, an alternative to the RANCAU function, gives greater control of the seed and random number streams.

**See Also**

* Functions:
  - “RAND Function” on page 555

* CALL Routines:
  - “CALL RANCAU Routine” on page 797

---

### RANEXP Function

Returns a random variate from an exponential distribution.

**Category:** Random Number

**Restriction:** This function is not valid on the CAS server.

**Tip:** If you want to change the seed value during execution, you must use the CALL RANEXP routine instead of the RANEXP function.

**Syntax**

\[
\text{RANEXP}(\text{seed})
\]

**Required Argument**

\textit{seed}

is a numeric constant, variable, or expression with an integer value. If $\text{seed} \geq 0$, the time of day is used to initialize the seed stream.

**Range** $\text{seed} < 2^{31}-1$

**See** “Seed Values” on page 11 for more information about seed values.
Details

The RANEXP function returns a variate that is generated from an exponential distribution with parameter 1. An inverse transform method applied to a RANUNI uniform variate is used.

An exponential variate X with parameter LAMBDA can be generated:

\[ x = \frac{\text{ranexp(seed)}}{\lambda} \]

An extreme value variate X with location parameter ALPHA and scale parameter BETA can be generated:

\[ x = \alpha - \beta \log(\text{ranexp(seed)}) \]

A geometric variate X with parameter P can be generated as follows:

\[ x = \text{floor}\left(-\frac{\text{ranexp(seed)}}{\log(1-p)}\right) \]

For a discussion about seeds and streams of data, as well as examples of using the random-number functions, see “Generating Multiple Variables from One Seed in Random-Number Functions” on page 22.

Comparisons

The CALL RANEXP routine, an alternative to the RANEXP function, gives greater control of the seed and random number streams.

See Also

Functions:

• “RAND Function” on page 555

CALL Routines:

• “CALL RANEXP Routine” on page 800

RANGAM Function

Returns a random variate from a gamma distribution.

<table>
<thead>
<tr>
<th>Category</th>
<th>Random Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restriction</td>
<td>This function is not valid on the CAS server.</td>
</tr>
<tr>
<td>Tip</td>
<td>If you want to change the seed value during execution, you must use the CALL RANGAM routine instead of the RANGAM function.</td>
</tr>
</tbody>
</table>

Syntax

\[ \text{RANGAM}(\text{seed}, a) \]

Required Arguments

seed

is a numeric constant, variable, or expression with an integer value. If \( seed \leq 0 \), the time of day is used to initialize the seed stream.
Range  seed < $2^{31}$−1

See  “Seed Values” on page 11 for more information about seed values

\[ a \]

is a numeric constant, variable, or expression that specifies the shape parameter.

Range  \[ a > 0 \]

Details

The RANGAM function returns a variate that is generated from a gamma distribution with parameter \( a \). For \( a > 1 \), an acceptance-rejection method due to Cheng (1977) is used. (See “References” on page 1111). For \( a \leq 1 \), an acceptance-rejection method due to Fishman is used (1978, Algorithm G2) (See “References” on page 1111).

A gamma variate \( X \) with shape parameter \( \text{ALPHA} \) and scale \( \text{BETA} \) can be generated:

\[ x = \text{beta} \times \text{rangam} \left( \text{seed}, \alpha \right); \]

If \( 2 \times \text{ALPHA} \) is an integer, a chi-square variate \( X \) with \( 2 \times \text{ALPHA} \) degrees of freedom can be generated:

\[ x = 2 \times \text{rangam} \left( \text{seed}, \alpha \right); \]

If \( N \) is a positive integer, an Erlang variate \( X \) can be generated:

\[ x = \text{beta} \times \text{rangam} \left( \text{seed}, N \right); \]

It has the distribution of the sum of \( N \) independent exponential variates whose means are \( \text{BETA} \).

And finally, a beta variate \( X \) with parameters \( \text{ALPHA} \) and \( \text{BETA} \) can be generated:

\[
\begin{align*}
  y_1 &= \text{rangam} \left( \text{seed}, \alpha \right); \\
  y_2 &= \text{rangam} \left( \text{seed}, \beta \right); \\
  x &= y_1 / (y_1 + y_2);
\end{align*}
\]

For a discussion about seeds and streams of data, as well as examples of using the random-number functions, see “Generating Multiple Variables from One Seed in Random-Number Functions” on page 22.

Comparisons

The CALL RANGAM routine, an alternative to the RANGAM function, gives greater control of the seed and random number streams.

See Also

Functions:

•  “RAND Function” on page 555

CALL Routines:

•  “CALL RANGAM Routine” on page 802
RANNOR Function

Returns a random variate from a normal distribution.

**Category:** Random Number

**Restriction:** This function is not valid on the CAS server.

**Tip:** If you want to change the seed value during execution, you must use the CALL RANNOR routine instead of the RANNOR function.

**Syntax**

\[
\text{RANNOR}(\text{seed})
\]

**Required Argument**

\(\text{seed}\)

is a numeric constant, variable, or expression with an integer value. If \(\text{seed} \leq 0\), the time of day is used to initialize the seed stream.

**Range** \(\text{seed} < 2^{31} - 1\)

**See** “Seed Values” on page 11 for more information about seed values.

**Details**

The RANNOR function returns a variate that is generated from a normal distribution with mean 0 and variance 1. The Box-Muller transformation of RANUNI uniform variates is used.

A normal variate \(X\) with mean \(\mu\) and variance \(\sigma^2\) can be generated with this code:

\[
x = \mu + \sqrt{\sigma^2} \times \text{rannor}(\text{seed});
\]

A lognormal variate \(X\) with mean \(\exp(\mu + \sigma^2/2)\) and variance \(\exp(2\mu + 2\sigma^2) - \exp(2\mu + \sigma^2)\) can be generated with this code:

\[
x = \exp(\mu + \sqrt{\sigma^2} \times \text{rannor}(\text{seed}));
\]

For a discussion about seeds and streams of data, as well as examples of using the random-number functions, see “Generating Multiple Variables from One Seed in Random-Number Functions” on page 22.

**Comparisons**

The CALL RANNOR routine, an alternative to the RANNOR function, gives greater control of the seed and random number streams.

**See Also**

**Functions:**

- “RAND Function” on page 555

**CALL Routines:**
RANPOI Function

Returns a random variate from a Poisson distribution.

**Category:** Random Number

**Restriction:** This function is not valid on the CAS server.

**Tip:** If you want to change the seed value during execution, you must use the CALL RANPOI routine instead of the RANPOI function.

**Syntax**

RANPOI(seed, m)

**Required Arguments**

*seed*

is a numeric constant, variable, or expression with an integer value. If $seed \leq 0$, the time of day is used to initialize the seed stream.

Range $seed < 2^{31} - 1$

See “Seed Values” on page 11 for more information about seed values

*m*

is a numeric constant, variable, or expression that specifies the mean of the distribution.

Range $m \geq 0$

**Details**

The RANPOI function returns a variate that is generated from a Poisson distribution with mean $m$. For $m < 85$, an inverse transform method applied to a RANUNI uniform variate is used (Fishman 1976). (See “References” on page 1111.) For $m \geq 85$, the normal approximation of a Poisson random variable is used. To expedite execution, internal variables are calculated only on initial calls (that is, with each new $m$).

For a discussion about seeds and streams of data, as well as examples of using the random-number functions, see “Generating Multiple Variables from One Seed in Random-Number Functions” on page 22.

**Comparisons**

The CALL RANPOI routine, an alternative to the RANPOI function, gives greater control of the seed and random number streams.

**See Also**

**Functions:**

- “RAND Function” on page 555
CALL Routines:

- “CALL RANPOI Routine” on page 807

RANTBL Function

Returns a random variate from a tabled probability distribution.

<table>
<thead>
<tr>
<th>Category:</th>
<th>Random Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restriction:</td>
<td>This function is not valid on the CAS server.</td>
</tr>
<tr>
<td>Tip:</td>
<td>If you want to change the seed value during execution, you must use the CALL RANTBL routine instead of the RANTBL function.</td>
</tr>
</tbody>
</table>

Syntax

RANTBL(seed, p1, ..., pn)

Required Arguments

seed

is a numeric constant, variable, or expression with an integer value. If seed ≤ 0, the time of day is used to initialize the seed stream.

Range  seed < 2^{31}−1

See “Seed Values” on page 11 for more information about seed values

pi

is a numeric constant, variable, or expression.

Range  0 ≤ pi ≤ 1 for 0 < i ≤ n

Details

The RANTBL function returns a variate that is generated from the probability mass function defined by p1 through pn. An inverse transform method applied to a RANUNI uniform variate is used. RANTBL returns

1 with probability p1

2 with probability p2

. .

n with probability pn

n + 1 with probability \(1 - \sum_{i=1}^{n} p_i\) if \(\sum_{i=1}^{n} p_i \leq 1\)
If, for some index \( j < n \), \( \sum_{i=1}^{j} p_i \geq 1 \), RANTBL returns only the indices 1 through \( j \) with the probability of occurrence of the index \( j \) equal to \( 1 - \sum_{i=1}^{j-1} p_i \).

Let \( n = 3 \) and \( P1, P2, \) and \( P3 \) be three probabilities with \( P1+P2+P3=1 \), and \( M1, M2, \) and \( M3 \) be three variables. The variable \( X \) in these statements

```plaintext
array m{3} m1-m3;
x=rantbl(seed, of p1-p3);
```

will be assigned one of the values of \( M1, M2, \) or \( M3 \) with probabilities of occurrence \( P1, P2, \) and \( P3 \), respectively.

For a discussion and example of an effective use of the random number CALL routines, see “Starting, Stopping, and Restarting a Stream” on page 26.

**Comparisons**

The CALL RANTBL routine, an alternative to the RANTBL function, gives greater control of the seed and random number streams.

**See Also**

**Functions:**
- “RAND Function” on page 555

**CALL Routines:**
- “CALL RANTBL Routine” on page 809

---

### RANTRI Function

Returns a random variate from a triangular distribution.

**Category:** Random Number

**Restriction:** This function is not valid on the CAS server.

**Tip:** If you want to change the seed value during execution, you must use the CALL RANTRI routine instead of the RANTRI function.

---

### Syntax

**RANTRI**(*seed, h*)

**Required Arguments**

- **seed**
  
  is a numeric constant, variable, or expression with an integer value. If \( seed \leq 0 \), the time of day is used to initialize the seed stream.

  **Range** \( seed < 2^{31}-1 \)
See “Seed Values” on page 11 for more information about seed values.

\( h \)

is a numeric constant, variable, or expression that specifies the mode of the distribution.

Range \( 0 < h < 1 \)

**Details**

The RANTRI function returns a variate that is generated from the triangular distribution on the interval \((0,1)\) with parameter \(h\), which is the modal value of the distribution. An inverse transform method applied to a RANUNI uniform variate is used.

A triangular distribution \(X\) on the interval \((A,B)\) with mode \(C\), where \(A \leq C \leq B\), can be generated:

\[
x = (b-a)*\text{rantri}(\text{seed}, (c-a)/(b-a))+a;
\]

For a discussion about seeds and streams of data, as well as examples of using the random-number functions, see “Generating Multiple Variables from One Seed in Random-Number Functions” on page 22.

**Comparisons**

The CALL RANTRI routine, an alternative to the RANTRI function, gives greater control of the seed and random number streams.

**See Also**

**Functions:**
- “RAND Function” on page 555

**CALL Routines:**
- “CALL RANTRI Routine” on page 812

---

**RANUNI Function**

Returns a random variate from a uniform distribution.

- **Category:** Random Number
- **Restriction:** This function is not valid on the CAS server.
- **Tip:** If you want to change the seed value during execution, you must use the CALL RANUNI routine instead of the RANUNI function.

**Syntax**

\[
\text{RANUNI}(\text{seed})
\]
**Required Argument**

*seed*

A numeric constant, variable, or expression with an integer value. If \( seed \leq 0 \), the time of day is used to initialize the seed stream.

Range \( seed < 2^{31} - 1 \)

See “Seed Values” on page 11 for more information about seed values.

**Details**

The RANUNI function returns a number that is generated from the uniform distribution on the interval \((0,1)\) using a prime modulus multiplicative generator with modulus \(2^{31} - 1\) and multiplier 397204094 (Fishman and Moore 1982). (See “References” on page 1111.)

You can use a multiplier to change the length of the interval and an added constant to move the interval. For example,

\[
\text{random_variate} = a \cdot \text{ranuni(seed)} + b;
\]

returns a number that is generated from the uniform distribution on the interval \((b, a+b)\).

**Comparisons**

The CALL RANUNI routine, an alternative to the RANUNI function, gives greater control of the seed and random number streams.

**See Also**

Functions:
- “RAND Function” on page 555

CALL Routines:
- “CALL RANUNI Routine” on page 814

---

**RENAME Function**

Renames a member of a SAS library, an entry in a SAS catalog, an external file, or a directory.

**Categories:**
- External Files
- SAS File I/O

**Restrictions:**
- If the SAS session in which you are specifying the FILEEXIST function is in a locked-down state, and the pathname specified in the function has not been added to the lockdown path list, then the function will fail and a file access error related to the locked-down data will not be generated in the SAS log unless you specify the SYSMSG function.
- This function is not valid on the CAS server.

**Syntax**

\[
\text{RENAME}(old-name, new-name <, type <, description <, password <, generation>>>>)
\]
Required Arguments

old-name
specifies a character constant, variable, or expression that is the current name of a member of a SAS library, an entry in a SAS catalog, an external file, or an external directory.

For a data set, old-name can be a one-level or two-level name. For a catalog entry, old-name can be a one-level, two-level, or four-level name. For an external file or directory, old-name must be the full pathname of the file or the directory. If the value for old-name is not specified, then SAS uses the current directory.

new-name
specifies a character constant, variable, or expression that is the new one-level name for the library member, catalog entry, external file, or directory.

Optional Arguments

type
is a character constant, variable, or expression that specifies the type of element to rename. Type can be a null argument, or one of the following values:

ACCESS specifies a SAS/ACCESS descriptor that was created using SAS/ACCESS software.
CATALOG specifies a SAS catalog or catalog entry.
DATA specifies a SAS data set.
VIEW specifies a SAS data set view.
FILE specifies an external file or directory.

Default 'DATA'

description
specifies a character constant, variable, or expression that is the description of a catalog entry. You can specify description only when the value of type is CATALOG. Description can be a null argument.

password
is a character constant, variable, or expression that specifies the password for the data set that is being renamed. Password can be a null argument.

generation
is a numeric constant, variable, or expression that specifies the generation number of the data set that is being renamed. Generation can be a null argument.

Details

You can use the RENAME function to rename members of a SAS library or entries in a SAS catalog. SAS returns 0 if the operation was successful, and a value other than 0 if the operation was not successful.

To rename an entry in a catalog, specify the four-level name for old-name and a one-level name for new-name. You must specify CATALOG for type when you rename an entry in a catalog.

Operating Environment Information
Use RENAME in directory-based operating environments only. If you use RENAME in a mainframe operating environment, SAS generates an error.
Examples

Example 1: Renaming Data Sets and Catalog Entries
The following examples rename a SAS data set from Data1 to Data2, and also rename a catalog entry from A.SCL to B.SCL.

rc1=rename('mylib.data1', 'data2');
rc2=rename('mylib.mycat.a.scl', 'b', 'catalog');

Example 2: Renaming an External File
The following examples rename external files.

/* Rename a file that is located in another directory. */
rc=rename('/local/u/testdir/first',
      '/local/u/second', 'file');
/* Rename a PC file. */
rc=rename('d:\temp', 'd:\testfile', 'file');

Example 3: Renaming a Directory
The following example renames a directory in the UNIX operating environment.

rc=rename('/local/u/testdir/', '/local/u/oldtestdir', 'file');

Example 4: Renaming a Generation Data Set
The following example renames the generation data set Work.One to Work.Two, where the password for Work.One#003 is my-password.

rc=rename('work.one', 'two', 3, 'my-password');

See Also

Functions:
• “EXIST Function” on page 892
• “FDELETE Function” on page 902
• “FILEEXIST Function” on page 909

RESOLVE Function
Returns the resolved value of the argument after the argument has been processed by the macro facility.

Category:    Macro
Restriction: This function is not valid on the CAS server.
Note:        The RESOLVE function cannot reference secure macros.

Syntax
RESOLVE(argument)
**Required Argument**

*argument* is a character constant, variable, or expression with a value that is a macro expression.

**Details**

If the RESOLVE function returns a value to a variable that has not yet been assigned a length, by default the variable is assigned a length of 200 bytes.

RESOLVE is fully documented in *SAS Viya Macro Language: Reference*.

**See Also**

Functions:
- “SYMGET Function” on page 1070

---

### REWIND Function

Positions the data set pointer at the beginning of a SAS data set.

**Category:** SAS File I/O  
**Restriction:** This function is not valid on the CAS server.

**Syntax**

`REWIND(data-set-id)`

**Required Argument**

*data-set-id* is a numeric variable that specifies the data set identifier that the OPEN function returns.

**Restriction**  
The data set cannot be opened in IS mode.

**Details**

REWIND returns 0 if the operation was successful, ≠0 if it was not successful. After a call to REWIND, a call to FETCH reads the first observation in the data set.

If there is an active WHERE clause, REWIND moves the data set pointer to the first observation that satisfies the WHERE condition.

**Example**

This example calls FETCHOBS to fetch the tenth observation in the data set MYDATA. Next, the example calls REWIND to return to the first observation and fetch the first observation.

```%let dsid=%sysfunc(open(mydata, i));%let rc=%sysfunc(fetchobs(&dsid, 10));```
### SAVING Function

Returns the future value of a periodic saving.

**Category:** Financial  
**Restriction:** This function is not valid on the CAS server.

**Syntax**

```
SAVING(f, p, r, n)
```

**Required Arguments**

- **f**  
  - is numeric, the future amount (at the end of $n$ periods).  
  - **Range** $f \geq 0$

- **p**  
  - is numeric, the fixed periodic payment.  
  - **Range** $p \geq 0$

- **r**  
  - is numeric, the periodic interest rate expressed as a decimal.  
  - **Range** $r \geq 0$

- **n**  
  - is an integer, the number of compounding periods.  
  - **Range** $n \geq 0$

**Details**

The SAVING function returns the missing argument in the list of four arguments from a periodic saving. The arguments are related by
\[ f = \frac{p(1 + r)((1 + r)^n - 1)}{r} \]

One missing argument must be provided. It is then calculated from the remaining three. No adjustment is made to convert the results to round numbers.

**Example**

A savings account pays a 5% nominal annual interest rate, compounded monthly. For a monthly deposit of $100, the number of payments that are needed to accumulate at least $12,000, can be expressed as

\[
\text{number} = \text{saving}(12000, 100, .05/12, .); 
\]

The value returned is 97.18 months. The fourth argument is set to missing, which indicates that the number of payments is to be calculated. The 5% nominal annual rate is converted to a monthly rate of 0.05/12. The rate is the fractional (not the percentage) interest rate per compounding period.

---

**SHA256 Function**

Returns the result of the message digest of a specified string.

- **Category:** Character
- **Restrictions:** This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see [Internationalization Compatibility](#). This function is not valid on the CAS server.

**Syntax**

\[
\text{SHA256}(string) 
\]

- **Required Argument**
  - \(string\) specifies a character constant, variable, or expression.

**Details**

**The Basics**

The SHA256 function converts a string, based on the SHA256 algorithm, to a 256-bit hash value.

The SHA256 function does not format its own output. Use the $BINARYw$ or $HEXw$ formats to view readable results.

**Using the SHA256 Function**

You can use the SHA256 function to track changes in your data sets. The SHA256 function can generate a digest of a set of column values in a table record. This digest could be treated as the signature of the record, and be used to track changes that are made to the record. If the digest from the new record matches the existing digest of a
table record, then the two records are the same. If the digest is different, then a column value in the record has changed. The new changed record could then be added to the table along with a new surrogate key because it represents a change to an existing keyed value.

The SHA256 function can be useful when you are developing shell scripts or Perl programs for software installation, file comparison, and detection of file corruption and tampering.

Example: Generating Results with the SHA256 Function

This example generates results that are returned by the SHA256 function.

```sas
data _null_; y=sha256('abc'); z=sha256('access method'); put y=$hex64.; put z=$hex64.; run;
```

The following output is displayed for ASCII systems:

```
y=BA7816BF8F01CFEA414140DE5DAE2223B00361A396177A9CB410FF61F20015AD
z=F2758E91725621F59F2F80D15DE8824560EDC471EBB40A83BA6D1259B1605915
```
SOAPACTION
specifies a character value that is a SOAPAction element to invoke on the web service.

WEBUSERNAME
specifies a character value that is a user name for basic web authentication.

WEBPASSWORD
specifies a character value that is a password for basic web authentication.
Encodings that are produced by PROC PWENCODE are supported.

WEBDOMAIN
specifies a character value that is the domain or realm for the user name and password.

MUSTUNDERSTAND
specifies a numeric value that is the setting for the mustUnderstand attribute in the SOAP header.

PROXYPORT
specifies a numeric value that is an HTTP proxy server port.

PROXYHOST
specifies a character value that is an HTTP proxy server host.

PROXYUSERNAME
specifies a character value that is an HTTP proxy server user name.

PROXYPASSWORD
specifies a character value that is an HTTP proxy server password. Encodings that are produced by PROC PWENCODE are supported.

CONFIGFILE
specifies a character value that is a Spring configuration file that is used primarily to set time-out values.

DEBUG
specifies a character value that is the full path to a file that is used for debugging logging output.

Example
The following example shows how to use the SOAPWEB function in a DATA step:

FILENAME request 'c:\temp\Request.xml';
FILENAME response 'c:\temp\Response.xml';

data _null_;  
  url="http://www.weather.gov/forecasts/xml/SOAP_server/ndfdXMLserver.php";
  soapaction="http://www.weather.gov/forecasts/xml/DWMLgen/wsd1/ndfdXML.wsdl#CornerPoints";
  proxyhost="someproxy.abc.xyz.com";
  proxyport=80;
  
  rc = soapweb("request", url, "response", soapaction, , , , proxyport, proxyhost);
run;

This section provides information about the SOAP request:

Request.xml:
See Also

Functions:

- “SOAPWEBMETA Function” on page 1048
- “SOAPWIPSERVICE Function” on page 1050
- “SOAPWIPSRS Function” on page 1052
- “SOAPWS Function” on page 1055
- “SOAPWSMETA Function” on page 1056

SOAPWEBMETA Function

Calls a web service by using basic web authentication; credentials for the authentication domain are retrieved from metadata.

**Category:** Web Service

**Restriction:** This function is not valid on the CAS server.

**Syntax**

```sas
SOAPWEBMETA(IN, URL <, option(s)>)
```

**Required Arguments**

- **IN**
  - specifies a character value that is the fileref. IN is used to input XML data that contains the SOAP request.

- **URL**
  - specifies a character value that is the URL of the web service endpoint.

**Optional Argument**

- **option**
  - specifies an option that you can use with the SOAPWEBMETA function. The following options are available:
OUT
specifies a character value that is the fileref where the SOAP response output XML is written.

SOAPACTION
specifies a character value that is a SOAPAction element to invoke on the web service.

WEBAUTHDOMAIN
specifies a character value that is the authentication domain from which to retrieve a user name and password from metadata for basic web authentication.

MUSTUNDERSTAND
specifies a numeric value that is the setting for the mustUnderstand attribute in the SOAP header.

PROXYPORT
specifies a numeric value that is an HTTP proxy server port.

PROXYHOST
specifies a character value that is an HTTP proxy server host.

PROXYUSERNAME
specifies a character value that is an HTTP proxy server user name.

PROXYPASSWORD
specifies a character value that is an HTTP proxy server password. Encodings that are produced by PROC PWENCODE are supported.

CONFIGFILE
specifies a character value that is a Spring configuration file that is used primarily to set time-out values.

DEBUG
specifies a character value that is the full path to a file that is used for debugging logging output.

Example
The following example shows how to use the SOAPWEBMETA function with the DATA step:

FILENAME request 'C:\temp\Request.xml';
FILENAME response 'C:\temp\Response.xml';

OPTIONS metauser="metadata-user"
    metapass="password"
    metaprotocol=bridge
    metaport=8561
    metaserver="somemachine.abc.xyz.com";

data _null_;
  url="http://somemachine/basicauth/AddService.asmx";
  soapaction="http://tempuri.org/Add";
  webauthdomain="DefaultAuth";

  rc = soapwebmeta("request", url, "response", soapaction, webauthdomain);
run;
See Also

Functions:
- “SOAPWEB Function” on page 1046
- “SOAPWIPSERVICE Function” on page 1050
- “SOAPWIPSRS Function” on page 1052
- “SOAPWS Function” on page 1055
- “SOAPWSMETA Function” on page 1056

SOAPWIPSERVICE Function

Calls a SAS registered web service by using WS-Security authentication; credentials are provided in the arguments.

**Category:** Web Service

**Restriction:** This function is not valid on the CAS server.

**Note:** This function uses the SAS environments file.

**Syntax**

```
SOAPWIPSERVICE(IN, SERVICE <, option(s)>)
```

**Required Arguments**

**IN**

specifies a character value that is the fileref. IN is used to input XML data that contains the SOAP request.

**SERVICE**

specifies the service name of the endpoint service as the service is stored in the Service Registry.

**Optional Argument**

**option**

specifies an option that you can use with the SOAPWIPSERVICE function. The following options are available:

**OUT**

specifies a character value that is the fileref where the SOAP response output XML is written.

**SOAPACTION**

specifies a character value that is a SOAPAction element to invoke on the web service.

**WSSUSERNAME**

specifies a character value that is a WS-Security user name.
WSSPASSWORD
specifies a character value that is a WS-Security password, which is the password for WSSUSERNAME. Encodings that are produced by PROC PWENCODE are supported.

ENVFILE
specifies a character value that is the location of the SAS environments file.

ENVIRONMENT
specifies a character value that is the environment defined in the SAS environments file to use.

MUSTUNDERSTAND
specifies a numeric value that is the setting for the mustUnderstand attribute in the SOAP header.

CONFIGFILE
specifies a character value that is a Spring configuration file that is used primarily to set time-out values.

DEBUG
specifies a character value that is the full path to a file that is used for debugging logging output.

Details

The SAS Environments File
The name of the service is provided in the Service Registry. The SAS environments file is used to locate the Service Registry and the destination service, as well as the Security Token Service, which generates a security token with the provided credentials.

Example
The following example shows how to use the SOAPWIPSERVICE function in a DATA step:

FILENAME request 'c:\temp\Request.xml';
FILENAME response 'c:\temp\Response.xml';

data _null;
  service="ReportRepositoryService";
  envfile="http://somemachine.abc.xyz.com/schemas/test-environment.xml";
  environment="test";
  wssusername="user-name";
  wsspassword="password";
  
  rc=soapwipservice("REQUEST", service, "RESPONSE", soapaction, wssusername, wsspassword, envfile, environment);
run;

This section gives you information about the SOAP request:

Request.xml:
See Also

Functions:

- “SOAPWEB Function” on page 1046
- “SOAPWEBMETA Function” on page 1048
- “SOAPWIPSRS Function” on page 1052
- “SOAPWS Function” on page 1055
- “SOAPWSMETA Function” on page 1056

SOAPWIPSRS Function

Calls a SAS registered web service by using WS-Security authentication; credentials are provided in the arguments.
Category: Web Service

Restriction: This function is not valid on the CAS server.

Notes: The credentials that are provided are used to generate a security token to call the destination service. The URL of the destination service is provided. The Registry Service is called directly to determine how to locate the Security Token Service.

Syntax

SOAPWIPSRS(IN, URL, SRSURL <, option(s)>)

Required Arguments

IN
specifies a character value that is the fileref. IN is used to input XML data that contains the SOAP request.

URL
specifies a character value that is the URL of the web service endpoint.

SRSURL
specifies a character value that is the URL of the System Registry Service.

Optional Argument

option
specifies an option that you can use with the SOAPWIPSRS function. The following options are available:

OUT
specifies a character value that is the fileref where the SOAP response output XML is written.

SOAPACTION
specifies a character value that is a SOAPAction element to invoke on the web service.

WSSUSERNAME
specifies a character value that is a WS-Security user name.

WSSPASSWORD
specifies a character value that is a WS-Security password, which is the password for WSSUSERNAME. Encodings that are produced by PROC PWENCODE are supported.

MUSTUNDERSTAND
specifies a numeric value that is the setting for the mustUnderstand attribute in the SOAP header.

CONFIGFILE
specifies a character value that is a Spring configuration file that is used primarily to set time-out values.

DEBUG
specifies a character value that is the full path to a file that is used for debugging logging output.
Example

The following example shows how to use the SOAPWIPSRS function in a DATA step:

```sas
FILENAME request 'c:\temp\Request.xml';
FILENAME response 'c:\temp\Response.xml';

data _null_;
  url="http://somemachine.abc.xyz.com:8080/TESTWIPSoapServices/services/ReportRepositoryService";
  srsurl="http://somemachine.abc.xyz.com:8080/TESTWIPSoapServices/services/ServiceRegistry";
  WSSUSERNAME="user-name";
  WSSPASSWORD="password";
  rc = soapwipsrs("request", url, srsurl, "response", soapaction, wssusername, wsspassword);
run;
```

This section provides information about the SOAP request:

Request.xml:

```xml
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header>
  </soapenv:Header>
  <soapenv:Body>
    <rep:isDirectoryDirectoryServiceInterfaceRequest>
      <rep:dirPathUrl>SBIP://Foundation/Users/someuser/My Folder</rep:dirPathUrl>
    </rep:isDirectoryDirectoryServiceInterfaceRequest>
  </soapenv:Body>
</soapenv:Envelope>
```

See Also

Functions:

- “SOAPWEB Function” on page 1046
- “SOAPWEBMETA Function” on page 1048
- “SOAPWIPSERVICE Function” on page 1050
- “SOAPWS Function” on page 1055
- “SOAPWSMETA Function” on page 1056
SOAPWS Function

Calls a web service by using WS-Security authentication; credentials are provided in the arguments.

Category: Web Service
Restriction: This function is not valid on the CAS server.

Syntax

SOAPWS(IN, URL <, option(s)>)

Required Arguments

IN
specifies a character value that is the fileref. IN is used to input XML data that contains the SOAP request.

URL
specifies a character value that is the URL of the web service endpoint.

Optional Argument

option
specifies an option that you can use with the SOAPWS function. The following options are available:

OUT
specifies a character value that is the fileref where the SOAP response output XML is written.

SOAPACTION
specifies a character value that is a SOAPAction element to invoke on the web service.

WSSUSERNAME
specifies a character value that is a WS-Security user name.

WSSPASSWORD
specifies a character value that is a WS-Security password, which is the password for WSSUSERNAME. Encodings that are produced by PROC PWENCODE are supported.

MUSTUNDERSTAND
specifies a numeric value that is the setting for the mustUnderstand attribute in the SOAP header.

PROXYPORT
specifies a numeric value that is an HTTP proxy server port.

PROXYHOST
specifies a character value that is an HTTP proxy server host.

PROXYUSERNAME
specifies a character value that is an HTTP proxy server user name.

PROXYPASSWORD
specifies a character value that is an HTTP proxy server password. Encodings that are produced by PROC PWENCODE are supported.
CONFIGFILE
specifies a character value that is a Spring configuration file that is used primarily to set time-out values.

DEBUG
specifies a character value that is the full path to a file that is used for debugging logging output.

Example
The following example shows how to use the SOAPWS function in a DATA step:

```sas
FILENAME request 'C:\temp\Request.xml';
FILENAME response 'C:\temp\Response.xml';

data _null_;
  url="http://somemachine.na.abc.com/SASBIWS/ProcSoapServices.asmx";
  soapaction="http://tempuri.org/ProcSoapServices/copyintoout_xml_att";
  WSSUSERNAME="sasuser";
  WSSPASSWORD="password";

  rc = soapws("request", url, "response", soapaction, wssusername, wsspassword);
run;
```

See Also

Functions:

- “SOAPWEB Function” on page 1046
- “SOAPWEBMETA Function” on page 1048
- “SOAPWIPSERVICE Function” on page 1050
- “SOAPWIPSRS Function” on page 1052
- “SOAPWSMETA Function” on page 1056

SOAPWSMETA Function
Calls a web service by using WS-Security authentication; credentials for the provided authentication domain are retrieved from metadata.

**Category:** Web Service

**Restriction:** This function is not valid on the CAS server.

**Syntax**

```sas
SOAPWSMETA(IN, URL <, option(s)>)
```
**Required Arguments**

**IN**
specifies a character value that is the fileref. IN is used to input XML data that contains the SOAP request.

**URL**
specifies a character value that is the URL of the web service endpoint.

**Optional Argument**

**option**
specifies an option that you can use with the SOAPWSMETA function. The following options are available:

**OUT**
specifies a character value that is the fileref where the SOAP response output XML is written.

**SOAPACTION**
specifies a character value that is a SOAPAction element to invoke on the web service.

**WSSAUTHDOMAIN**
specifies a character value that is the authentication domain for which to retrieve credentials to be used for WS-Security authentication.

**MUSTUNDERSTAND**
specifies a numeric value that is the setting for the mustUnderstand attribute in the SOAP header.

**PROXYPORT**
specifies a numeric value that is an HTTP proxy server port.

**PROXYHOST**
specifies a character value that is an HTTP proxy server host.

**PROXYUSERNAME**
specifies a character value that is an HTTP proxy server user name.

**PROXYPASSWORD**
specifies a character value that is an HTTP proxy server password. Encodings that are produced by PROC PWENCODE are supported.

**CONFIGFILE**
specifies a character value that is a Spring configuration file that is used primarily to set time-out values.

**DEBUG**
specifies a character value that is the full path to a file that is used for debugging logging output.

**See Also**

**Functions:**

- “SOAPWEB Function” on page 1046
- “SOAPWEBMETA Function” on page 1048
- “SOAPWIPSERVICE Function” on page 1050
- “SOAPWIPSRS Function” on page 1052
**SOUNDEX Function**

Encodes a string to facilitate searching.

- **Category:** Character
- **Restrictions:** This function is not valid on the CAS server. SOUNDEX algorithm is English-biased. This function is assigned an I18N Level 0 status, and is designed for SBCS data. Do not use this function to process DBCS or MBCS data. For more information, see Internationalization Compatibility.

**Syntax**

```
SOUNDEX(argument)
```

**Required Argument**

`argument`

specifies a character constant, variable, or expression.

**Details**

**Length of Returned Variable**

In a DATA step, if the SOUNDEX function returns a value to a variable that has not previously been assigned a length, then that variable is given a length of 200 bytes.

**The Basics**

The SOUNDEX function encodes a character string according to an algorithm that was originally developed by Margaret K. Odell and Robert C. Russel (US Patents 1261167 (1918) and 1435663 (1922)). The algorithm is described in Knuth, *The Art of Computer Programming, Volume 3*. (See “References” on page 1111.) Note that the SOUNDEX algorithm is English-biased and is less useful for languages other than English.

The SOUNDEX function returns a copy of the `argument` that is encoded by using the following steps:

1. Retain the first letter in the `argument` and discard the following letters:

   A E H I O U W Y

2. Assign the following numbers to these classes of letters:

   - 1: B F P V
   - 2: C G J K Q S X Z
   - 3: D T
   - 4: L
   - 5: M N
   - 6: R
3. If two or more adjacent letters have the same classification from Step 2, then discard all but the first. (Adjacent refers to the position in the word before discarding letters.)

The algorithm that is described in Knuth adds trailing zeros and truncates the result to the length of 4. You can perform these operations with other SAS functions.

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=soundex('Paul'); put x;</td>
<td>P4</td>
</tr>
<tr>
<td>word='amnesty'; x=soundex(word); put x;</td>
<td>A523</td>
</tr>
</tbody>
</table>

**SPEDIS Function**

Determines the likelihood of two words matching, expressed as the asymmetric spelling distance between the two words.

- **Category:** Character
- **Restrictions:**
  - This function is assigned an I18N Level 0 status, and is designed for SBCS data. Do not use this function to process DBCS or MBCS data. For more information, see [Internationalization Compatibility](#).
  - This function is not valid on the CAS server.

**Syntax**

\[
\text{SPEDIS}(\text{query, keyword})
\]

**Required Arguments**

- **query**
  - identifies the word to query for the likelihood of a match. SPEDIS removes trailing blanks before comparing the value.

- **keyword**
  - specifies a target word for the query. SPEDIS removes trailing blanks before comparing the value.

**Details**

**Length of Returned Variable**

In a DATA step, if the SPEDIS function returns a value to a variable that has not previously been assigned a length, then that variable is given a length of 200 bytes.
The Basics
SPEDIS returns the distance between the query and a keyword, a nonnegative value that is usually less than 100 but never greater than 200 with the default costs.

SPEDIS computes an asymmetric spelling distance between two words as the normalized cost for converting the keyword to the query word by using a sequence of operations. SPEDIS(QUERY, KEYWORD) is not the same as SPEDIS(KEYWORD, QUERY).

Costs for each operation that is required to convert the keyword to the query are listed in the following table:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cost</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>match</td>
<td>0</td>
<td>no change</td>
</tr>
<tr>
<td>singlet</td>
<td>25</td>
<td>delete one of a double letter</td>
</tr>
<tr>
<td>doublet</td>
<td>50</td>
<td>double a letter</td>
</tr>
<tr>
<td>swap</td>
<td>50</td>
<td>reverse the order of two consecutive letters</td>
</tr>
<tr>
<td>truncate</td>
<td>50</td>
<td>delete a letter from the end</td>
</tr>
<tr>
<td>append</td>
<td>35</td>
<td>add a letter to the end</td>
</tr>
<tr>
<td>delete</td>
<td>50</td>
<td>delete a letter from the middle</td>
</tr>
<tr>
<td>insert</td>
<td>100</td>
<td>insert a letter in the middle</td>
</tr>
<tr>
<td>replace</td>
<td>100</td>
<td>replace a letter in the middle</td>
</tr>
<tr>
<td>firstdel</td>
<td>100</td>
<td>delete the first letter</td>
</tr>
<tr>
<td>firstins</td>
<td>200</td>
<td>insert a letter at the beginning</td>
</tr>
<tr>
<td>firstrep</td>
<td>200</td>
<td>replace the first letter</td>
</tr>
</tbody>
</table>

The distance is the sum of the costs divided by the length of the query. If this ratio is greater than one, the result is rounded down to the nearest whole number.

Comparisons
The SPEDIS function is similar to the COMPLEV and COMPGED functions, but COMPLEV and COMPGED are much faster, especially for long strings.

Example
Here is an example of using the SPEDIS function.

```sas
data words;
  input Operation $ Query $ Keyword $;
  Distance=spedis(query, keyword);
  Cost=distance * length(query);
```

datalines;
mach fuzzy fuzzy
singlet fuzzy fuzzy
doublet fuzzzy fuzzy
swap fuzzzy fuzzy
truncat fuzz fuzzy
append fuzzys fuzzy
delete fzy fuzzy
insert fluzzy fuzzy
replace fizzy fuzzy
firstdel uzzy fuzzy
firstins pfuzzy fuzzy
firstrep wuzzy fuzzy
several floozy fuzzy
;
proc print data=words;
run;

Figure 3.30  Costs for SPEDIS Operations

<table>
<thead>
<tr>
<th>Obs</th>
<th>Operation</th>
<th>Query</th>
<th>Keyword</th>
<th>Distance</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>match</td>
<td>fuzzy</td>
<td>fuzzy</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>singlet</td>
<td>fuzzy</td>
<td>fuzzy</td>
<td>6</td>
<td>24</td>
</tr>
<tr>
<td>3</td>
<td>doublet</td>
<td>fuzzzy</td>
<td>fuzzy</td>
<td>8</td>
<td>48</td>
</tr>
<tr>
<td>4</td>
<td>swap</td>
<td>fuzzzy</td>
<td>fuzzy</td>
<td>10</td>
<td>50</td>
</tr>
<tr>
<td>5</td>
<td>truncate</td>
<td>fuzz</td>
<td>fuzzy</td>
<td>12</td>
<td>48</td>
</tr>
<tr>
<td>6</td>
<td>append</td>
<td>fuzzys</td>
<td>fuzzy</td>
<td>5</td>
<td>30</td>
</tr>
<tr>
<td>7</td>
<td>delete</td>
<td>fzy</td>
<td>fuzzy</td>
<td>12</td>
<td>48</td>
</tr>
<tr>
<td>8</td>
<td>insert</td>
<td>fluzzy</td>
<td>fuzzy</td>
<td>16</td>
<td>96</td>
</tr>
<tr>
<td>9</td>
<td>replace</td>
<td>fizzy</td>
<td>fuzzy</td>
<td>20</td>
<td>100</td>
</tr>
<tr>
<td>10</td>
<td>firstdel</td>
<td>uzzy</td>
<td>fuzzy</td>
<td>25</td>
<td>100</td>
</tr>
<tr>
<td>11</td>
<td>firstins</td>
<td>pfuzzy</td>
<td>fuzzy</td>
<td>33</td>
<td>198</td>
</tr>
<tr>
<td>12</td>
<td>firstrep</td>
<td>wuzzy</td>
<td>fuzzy</td>
<td>40</td>
<td>200</td>
</tr>
<tr>
<td>13</td>
<td>several</td>
<td>floozy</td>
<td>fuzzy</td>
<td>50</td>
<td>300</td>
</tr>
</tbody>
</table>

See Also

Functions:
- “COMPGED Function” on page 856
- “COMPLEV Function” on page 862
## SQUANTILE Function

Returns the quantile from a distribution when you specify the right probability (SDF).

**Category:** Quantile  
**Restriction:** This function is not valid on the CAS server.  
**See:** “SDF Function” on page 596

### Syntax

```plaintext
SQUANTILE(distribution, probability, parameter-1, ..., parameter-k)
```

### Required Arguments

**distribution**  
is a character constant, variable, or expression that identifies the distribution. Valid distributions are as follows:

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bernoulli</td>
<td>BERNULLI</td>
</tr>
<tr>
<td>Beta</td>
<td>BETA</td>
</tr>
<tr>
<td>Binomial</td>
<td>BINOMIAL</td>
</tr>
<tr>
<td>Cauchy</td>
<td>CAUCHY</td>
</tr>
<tr>
<td>Chi-Square</td>
<td>CHISQUARE</td>
</tr>
<tr>
<td>Conway-Maxwell-Poisson</td>
<td>CONMAXPOI</td>
</tr>
<tr>
<td>Exponential</td>
<td>EXPONENTIAL</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>Gamma</td>
<td>GAMMA</td>
</tr>
<tr>
<td>Generalized Poisson</td>
<td>GENPOISSON</td>
</tr>
<tr>
<td>Geometric</td>
<td>GEOMETRIC</td>
</tr>
<tr>
<td>Hypergeometric</td>
<td>HYPERGEOMETRIC</td>
</tr>
<tr>
<td>Laplace</td>
<td>LAPLACE</td>
</tr>
<tr>
<td>Logistic</td>
<td>LOGISTIC</td>
</tr>
<tr>
<td>Lognormal</td>
<td>LOGNORMAL</td>
</tr>
<tr>
<td>Distribution</td>
<td>Argument</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Negative binomial</td>
<td>NEGBINOMIAL</td>
</tr>
<tr>
<td>Normal</td>
<td>NORMAL</td>
</tr>
<tr>
<td>Normal mixture</td>
<td>NORMALMIX</td>
</tr>
<tr>
<td>Pareto</td>
<td>PARETO</td>
</tr>
<tr>
<td>Poisson</td>
<td>POISSON</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>Tweedie</td>
<td>TWEEDIE</td>
</tr>
<tr>
<td>Uniform</td>
<td>UNIFORM</td>
</tr>
<tr>
<td>Wald (inverse Gaussian)</td>
<td>WALD</td>
</tr>
<tr>
<td>Weibull</td>
<td>WEIBULL</td>
</tr>
</tbody>
</table>

Note: Except for T, F, and NORMALMIX, you can minimally identify any distribution by its first four characters.

probability is a numeric constant, variable, or expression that specifies the value of a random variable.

parameter-1, ..., parameter-k are optional shape, location, or scale parameters that are appropriate for the specific distribution.

Details

The SQUANTILE function computes the quantile from the specified continuous or discrete distribution, based on the probability value that is provided. For more information, see “Details” on page 154 in the CDF function.

The Conway-Maxwell-Poisson distribution of the SQUANTILE function returns the counts value \( y \) that is the smallest integer whose SDF value is less than \( p \). The syntax for the Conway-Maxwell-Poisson distribution in the SQUANTILE function has the following form:

\[
\text{SQUANTILE('CONMAXPOI',} p, \lambda, \nu)\]

\( p \) is a real number between 0 and 1, inclusively.

\( \lambda \) is similar to the mean, as in the Poisson distribution.

\( \nu \) is a dispersion parameter.

For more information, see “Conway-Maxwell-Poisson” distribution in the PDF function on page 482.
For more information about the distributions that are listed in the table, see “PDF Function” on page 477.

Examples

Example 1: Using the LOGISTIC Distribution

data;
  dist='logistic';
  sdf=squantile(dist, 1.e-20);
  put sdf=;
  p=sdf(dist, sdf);
  put p=/* p will be 1.e-20 */;
run;

SAS writes the following results to the log:

```
sdf=46.05170186
p=1E-20
```

Example 2: Using the Conway-Maxwell-Poisson Distribution

data _null_
  y=squantile('conmaxpoi',.2,2.3,.4);
  put y=;
run;

SAS writes the following results to the log:

```
y=12
```

See Also

Functions:

- “CDF Function” on page 153
- “LOGCDF Function” on page 416
- “LOGPDF Function” on page 418
- “LOGSDF Function” on page 420
- “PDF Function” on page 477
- “QUANTILE Function” on page 549
- “SDF Function” on page 596

STFIPS Function

Converts state postal codes to FIPS state codes.

<table>
<thead>
<tr>
<th>Category:</th>
<th>State and ZIP code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restriction:</td>
<td>This function is not valid on the CAS server.</td>
</tr>
</tbody>
</table>
Syntax

STFIPS(*postal-code*)

Required Argument

*postal-code*

specifies a character expression that contains the two-character standard state postal code. Characters can be mixed case. The function ignores trailing blanks, but generates an error if the expression contains leading blanks.

Details

The STFIPS function converts a two-character state postal code (or worldwide GSA geographic code for U.S. territories) to the corresponding numeric U.S. Federal Information Processing Standards (FIPS) code.

Comparisons

The STFIPS, STNAME, and STNAMEL functions take the same argument but return different values. STFIPS returns a numeric U.S. Federal Information Processing Standards (FIPS) code. STNAME returns an uppercase state name. STNAMEL returns a mixed case state name.

Example

The following examples show the differences when using STFIPS, STNAME, and STNAMEL:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>fips=stfips('NC'); put fips;</td>
<td>37</td>
</tr>
<tr>
<td>state=stname('NC'); put state;</td>
<td>NORTH CAROLINA</td>
</tr>
<tr>
<td>state=stnamel('NC'); put state;</td>
<td>North Carolina</td>
</tr>
</tbody>
</table>

See Also

Functions:
- “FIPNAME Function” on page 917
- “FIPNAMEL Function” on page 918
- “FIPSTATE Function” on page 920
- “STNAME Function” on page 1066
- “STNAMEL Function” on page 1067
STNAME Function

Converts state postal codes to uppercase state names.

<table>
<thead>
<tr>
<th>Category:</th>
<th>State and ZIP code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restriction:</td>
<td>This function is not valid on the CAS server.</td>
</tr>
</tbody>
</table>

Syntax

\[ \text{STNAME}(\text{postal-code}) \]

Required Argument

\( \text{postal-code} \)

specifies a character expression that contains the two-character standard state postal code. Characters can be mixed case. The function ignores trailing blanks, but generates an error if the expression contains leading blanks.

Details

The STNAME function converts a two-character state postal code (or worldwide GSA geographic code for U.S. territories) to the corresponding state name in uppercase.

Note: For Version 6, the maximum length of the value that is returned is 200 characters. For Version 7 and beyond, the maximum length is 20 characters.

Comparisons

The STFIPS, STNAME, and STNAMEL functions take the same argument but return different values. STFIPS returns a numeric U.S. Federal Information Processing Standards (FIPS) code. STNAME returns an uppercase state name. STNAMEL returns a mixed case state name.

Example

The following examples show the differences when using STFIPS, STNAME, and STNAMEL:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>fips=stfips('NC'); put fips;</td>
<td>37</td>
</tr>
<tr>
<td>state=stname('NC'); put state;</td>
<td>NORTH CAROLINA</td>
</tr>
<tr>
<td>state=stnamel('NC'); put state;</td>
<td>North Carolina</td>
</tr>
</tbody>
</table>
STNAMEL Function
Converts state postal codes to mixed case state names.

Category: State and ZIP code
Restriction: This function is not valid on the CAS server.

Syntax
STNAMEL(postal-code)

Required Argument
postal-code
specifies a character expression that contains the two-character standard state postal
code. Characters can be mixed case. The function ignores trailing blanks, but
generates an error if the expression contains leading blanks.

Details
If the STNAMEL function returns a value to a variable that has not yet been assigned a
length, by default the variable is assigned a length of 20.

The STNAMEL function converts a two-character state postal code (or worldwide GSA
geographic code for U.S. territories) to the corresponding state name in mixed case.

Note: For Version 6, the maximum length of the value that is returned is 200 characters.
       For Version 7 and beyond, the maximum length is 20 characters.

Comparisons
The STFIPS, STNAME, and STNAMEL functions take the same argument but return
different values. STFIPS returns a numeric U.S. Federal Information Processing
Standards (FIPS) code. STNAME returns an uppercase state name. STNAMEL returns a
mixed case state name.
Example

The following examples show the differences when using STFIPS, STNAME, and STNAMEL:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>fips=stfips('NC'); put fips;</td>
<td>37</td>
</tr>
<tr>
<td>state=stname('NC'); put state;</td>
<td>NORTH CAROLINA</td>
</tr>
<tr>
<td>state=stnamel('NC'); put state;</td>
<td>North Carolina</td>
</tr>
</tbody>
</table>

See Also

Functions:
- “FIPNAME Function” on page 917
- “FIPNAMEL Function” on page 918
- “FIPSTATE Function” on page 920
- “STFIPS Function” on page 1064
- “STNAME Function” on page 1066

SUBPAD Function

Returns a substring that has a length that you specify, using blank padding if necessary.

**Category:** Character

**Restrictions:**
- This function is assigned an I18N Level 0 status, and is designed for SBCS data. Do not use this function to process DBCS or MBCS data. For more information, see Internationalization Compatibility.
- This function is not valid on the CAS server.

**Syntax**

```
SUBPAD(string, position <, length>)
```

**Required Arguments**

- **string** specifies a character constant, variable, or expression.
- **position** is a positive integer that specifies the position of the first character in the substring.
Optional Argument

length

is a nonnegative integer that specifies the length of the substring. If you do not specify length, the SUBPAD function returns the substring that extends from the position that you specify to the end of the string.

Details

In a DATA step, if the SUBPAD function returns a value to a variable that has not previously been assigned a length, then that variable is given a length of 200 bytes. If the substring that you specify extends beyond the length of the string, the result is padded with blanks.

Comparisons

The SUBPAD function is similar to the SUBSTR function except for the following differences:

• If the value of length in SUBPAD is zero, SUBPAD returns a zero-length string. If the value of length in SUBSTR is zero, SUBSTR
  • writes a note to the log stating that the third argument is invalid
  • sets _ERROR_=1
  • returns the substring that extends from the position that you specified to the end of the string.

• If the substring that you specify extends past the end of the string, SUBPAD pads the result with blanks to yield the length that you requested. If the substring that you specify extends past the end of the string, SUBSTR
  • writes a note to the log stating that the third argument is invalid
  • sets _ERROR_=1
  • returns the substring that extends from the position that you specified to the end of the string.

See Also

Functions:

• “SUBSTRN Function” on page 618

SYMEXIST Function

Returns an indication of the existence of a macro variable.

Category:     Macro
Restriction:  This function is not valid on the CAS server.
See:          “SYMEXIST Function” in SAS Viya Macro Language: Reference

Syntax

SYMEXIST (argument)
**Required Argument**

*argument*

- the name of a macro variable within double quotation marks but without an ampersand
- the name of a DATA step character variable, specified with no quotation marks, which contains a macro variable name
- a character expression that constructs a macro variable name

**Details**

The SYMEXIST function searches any enclosing local symbol tables and then the global symbol table for the indicated macro variable and returns 1 if the macro variable is found or 0 if the macro variable is not found.

For more information, see the “SYMEXIST Function” in *SAS Viya Macro Language: Reference*.

**SYMGET Function**

Returns the value of a macro variable during DATA step execution.

- **Category:** Macro
- **Restriction:** This function is not valid on the CAS server.

**Syntax**

`SYMGET(argument)`

**Required Argument**

*argument*

- the name of a macro variable within double quotation marks but without an ampersand
- the name of a DATA step character variable, specified with no quotation marks, which contains a macro variable name
- a character expression that constructs a macro variable name

**Details**

If the SYMGET function returns a value to a variable that has not yet been assigned a length, by default the variable is assigned a length of 200.

The SYMGET function returns the value of a macro variable during DATA step execution. For more information, see the “SYMGET Function” in *SAS Viya Macro Language: Reference*.

**See Also**

- *SAS Viya Macro Language: Reference*
CALL Routines:

- “CALL SYMPUT Routine” on page 841

SYMGLOBL Function

Returns an indication of whether a macro variable is in global scope to the DATA step during DATA step execution.

<table>
<thead>
<tr>
<th>Category:</th>
<th>Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restriction:</td>
<td>This function is not valid on the CAS server.</td>
</tr>
<tr>
<td>See:</td>
<td>“SYMGLOBL Function” in SAS Viya Macro Language: Reference</td>
</tr>
</tbody>
</table>

Syntax

SYMGLOBL (argument)

Required Argument

argument can be one of the following items:

- the name of a macro variable within double quotation marks but without an ampersand.
- the name of a DATA step character variable, specified with no quotation marks, which contains a macro variable name.
- a character expression that constructs a macro variable name.

Details

The SYMGLOBL function searches only the global symbol table for the indicated macro variable and returns 1 if the macro variable is found or 0 if the macro variable is not found.

For more information, see “SYMGLOBL Function” in SAS Viya Macro Language: Reference.

SYMLOCAL Function

Returns an indication of whether a macro variable is in local scope to the DATA step during DATA step execution.

<table>
<thead>
<tr>
<th>Category:</th>
<th>Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restriction:</td>
<td>This function is not valid on the CAS server.</td>
</tr>
<tr>
<td>See:</td>
<td>“SYMLOCAL Function” in SAS Viya Macro Language: Reference</td>
</tr>
</tbody>
</table>

Syntax

SYMLOCAL (argument)
**Required Argument**

*argument*

- can be one of the following items:
  - the name of a macro variable within double quotation marks but without an ampersand.
  - the name of a DATA step character variable, specified with no quotation marks, which contains a macro variable name.
  - a character expression that constructs a macro variable name.

**Details**

The SYMLOCAL function searches the enclosing local symbol tables for the indicated macro variable and returns 1 if the macro variable is found or 0 if the macro variable is not found.

For more information, see “SYMLOCAL Function” in *SAS Viya Macro Language: Reference*.

---

**SYSEXIST Function**

Returns a value that indicates whether an operating-environment variable exists in your environment.

**Categories:** SAS File I/O

**Special:**

**Restriction:** This function is not valid on the CAS server.

**Syntax**

SYSEXIST (*argument*)

**Required Argument**

*argument*

- specifies a character variable that is the name of an operating-environment variable that you want to test.

**Details**

The SYSEXIST function searches for the existence of an operating-environment variable and returns 1 if the variable is found or 0 if the variable is not found.

**Comparisons**

The SYSEXIST function tests for the existence of an operating-environment variable. The SYSGET function retrieves the value of an operating-environment variable.

**Example**

The following example assumes that HOME is a valid operating-environment variable in your environment, and that TEST is not valid. SYSEXIST tests both values:

```sas
data _null_;```
SAS writes the following output to the log:

<table>
<thead>
<tr>
<th>rc=1</th>
</tr>
</thead>
<tbody>
<tr>
<td>rc=0</td>
</tr>
</tbody>
</table>

If SYSEXIST returns a value of 1, then the variable that is being tested is an operating-environment variable. If SYSEXIST returns a value of 0, then the variable that is being tested is not an operating-environment variable in your environment.

See Also

Functions:
- “SYSGET Function” on page 1073

---

**SYSGET Function**

Returns the value of the specified operating environment variable.

**Category:** Special

**Restriction:** This function is not valid on the CAS server.

**Syntax**

Windows and UNIX:

SYSGET('environment-variable')

**Required Argument**

*environment-variable*

is a character constant, variable, or expression with a value that is the name of an environment variable under Windows and UNIX. This argument must be enclosed in single quotation marks.

**Details**

**General Information**

The SYSGET function returns the value of an environment variable as a character string. For example, this statement returns the value of the HOME environment variable under UNIX:

```
here=sysget('HOME');
```

Since trailing blanks are treated as significant digits by the SYSGET function, environment variable names that are passed to these functions should be trimmed of trailing blanks. Use the TRIM function to remove trailing blanks. See “Example: Obtain...”
Environment Variable Values under UNIX on page 1074 for an example of using the TRIM function with the SYSGET function to trim variable names.

If the SYSGET function returns a value to a variable that has not yet been assigned a length, by default the variable is assigned a length of 200.

If the value of the operating environment variable is truncated or the variable is not defined in the operating environment, SYSGET displays a warning message in the SAS log.

SYSGET Specifics under UNIX
The case of the value that you supply in the environment-variable argument must agree with the case of the variable that is stored in the UNIX operating environment.

Example: Obtain Environment Variable Values under UNIX
This example obtains the value of two environment variables in the UNIX environment:

```sas
data _null_;  
  length result $200;  
  input env_var $;  
  result=sysget(trim(env_var));  
  put env_var= result=;  
  datalines;  
  USER  
  PATH  
;  
Executing this DATA step for user ABCDEF displays these lines:
```

```
ENV_VAR=USER RESULT=abcdef
ENV_VAR=PATH RESULT=path-for-abcdef
```

See Also

Functions:

- “ENVLEN Function” on page 890

SYSMSG Function
Returns error or warning message text from processing the last data set or external file function.

**Categories:** SAS File I/O

**External Files**

**Restriction:** This function is not valid on the CAS server.

**Syntax**

SYSMSG()
Details
SYSMSG returns the text of error messages or warning messages that are produced when a data set or external file access function encounters an error condition. If no error message is available, the returned value is blank. The internally stored error message is reset to blank after a call to SYSMSG, so subsequent calls to SYSMSG before another error condition occurs return blank values.

Example
This example uses SYSMSG to write to the SAS log the error message generated if FETCH cannot copy the next observation into the Data Set Data Vector. The return code is 0 only when a record is fetched successfully:

```sas
%let rc=%sysfunc(fetch(&dsid));
%if &rc ne 0 %then
   %put %sysfunc(sysmsg());
```

See Also

Functions:
- “FETCH Function” on page 903
- “SYSRC Function” on page 1079

SYSPARM Function

Returns the system parameter string.

- **Category:** Special
- **Restriction:** This function is not valid on the CAS server.

Syntax

SYSPARM()

Details
If the SYSPARM function returns a value to a variable that has not yet been assigned a length, by default the variable is assigned a length of 200.

SYSPARM enables you to access a character string specified with the SYSPARM= system option at SAS invocation or in an OPTIONS statement.

Note: If the SYSPARM= system option is not specified, the SYSPARM function returns a string with a length of zero.

Example

This example shows the SYSPARM= system option and the SYSPARM function.

```sas
options sysparm='yes';
data a;
   If sysparm()='yes' then
```
do;
  ...more SAS Statements...
end;
run;

See Also

System Options:
- “SYSPARM= System Option” in SAS Viya Macro Language: Reference

SYSPROCESSID Function

Returns the process ID of the current process.

Category: Special
Restriction: This function is not valid on the CAS server.

Syntax
SYSPROCESSID()

Details
The SYSPROCESSID function returns the 32-character hexadecimal ID of the current process. This ID can be passed to the SYSPROCESSNAME function to obtain the name of the current process.

Examples

Example 1: Using a DATA Step
The following DATA step writes the current process ID to the SAS log:

data _null_
  id=sysprocessid();
  put id;
run;

Example 2: Using SAS Macro Language
The following SAS Macro Language code writes the current process ID to the SAS log:

%let id=%sysfunc(sysprocessid());
%put &id;

See Also

Functions:
- “SYSPROCESSNAME Function” on page 1077
SYSPROCESSNAME Function

Returns the process name that is associated with a given process ID, or returns the name of the current process.

**Category:** Special

**Restriction:** This function is not valid on the CAS server.

**Syntax**

SYSPROCESSNAME(<process_id>)

**Required Argument**

*process_id*

specifies a 32-character hexadecimal process ID.

**Details**

The SYSPROCESSNAME function returns the process name associated with the process ID that you supply as an argument. You can use the value returned from the SYSPROCESSID function as the argument to SYSPROCESSNAME. If you omit the argument, then SYSPROCESSNAME returns the name of the current process.

You can also use the values stored in the automatic macro variables SYSPROCESSID and SYSSTARTID as arguments to SYSPROCESSNAME.

**Examples**

**Example 1: Using SYSPROCESSNAME without an Argument in a DATA Step**

The following DATA step writes the current process name to the SAS log:

```sas
data _null_;
    name=sysprocessname();
    put name;
run;
```

**Example 2: Using SYSPROCESSNAME with an Argument in SAS Macro Language**

The following SAS Macro Language code writes the process name associated with the given process ID to the SAS log:

```sas
%let id=&sysprocessid;
%let name=%sysfunc(sysprocessname(&id));
%put &name;
```

**See Also**

Functions:

- “SYSPROCESSID Function” on page 1076
SYSPROD Function

Determines whether a product is licensed.

**Category:** Special

**Restriction:** This function is not valid on the CAS server.

**Syntax**

SYSPROD(product-name)

**Required Argument**

product-name

specifies a character constant, variable, or expression with a value that is the name of a SAS product.

**Requirement**

Product-name must be the correct official name of the product or solution.

**Details**

The SYSPROD function returns 1 if a specific SAS software product is licensed, 0 if it is a SAS software product but not licensed for your system, and -1 if the product name is not recognized. If SYSPROD indicates that a product is licensed, it means that the final license expiration date has not passed.

It is possible for a SAS software product to exist on your system even though the product is no longer licensed. In this case, SAS cannot access this product. Similarly, it is possible for a product to be licensed, but not installed.

Use SYSPROD in the DATA step, in an IML step, or in an SCL program.

**Example**

These examples determine whether a specified product is licensed.

- \( x=\text{sysprod('graph')}; \)
  
  If SAS/GRAPH software is currently licensed, then SYSPROD returns a value of 1. If SAS/GRAPH software is not currently licensed, then SYSPROD returns a value of 0.

- \( x=\text{sysprod('abc')}; \)
  
  SYSPROD returns a value of -1 because ABC is not a valid product name.

- \( x=\text{sysprod('base')}; \)
  
  or
  
  \( x=\text{sysprod('base sas')}; \)

  SYSPROD always returns a value of 1 because the Base product must be licensed for the SYSPROD function to run successfully.
SYSRC Function

Returns a system error number.

**Categories:** SAS File I/O

**Restriction:** This function is not valid on the CAS server.

### Syntax

SYSRC()

### Details

SYSRC returns the error number for the last system error encountered by a call to one of the data set functions or external file functions.

### Example

This example determines the error message if FILeref does not exist:

```
%if %sysfunc(fileref(myfile)) ne 0 %then
    %put %sysfunc(sysrc()) - %sysfunc(sysmsg());
```

### See Also

**Functions:**

- “FILEREF Function” on page 913
- “SYMSLG Function” on page 1074

TANH Function

Returns the hyperbolic tangent.

**Category:** Hyperbolic

**Restriction:** This function is not valid on the CAS server.

### Syntax

TANH(argument)

**Required Argument**

argument

specifies a numeric constant, variable, or expression.
Details

The TANH function returns the hyperbolic tangent of the argument, which is given by
\[
\frac{e^{\text{argument}} - e^{-\text{argument}}}{e^{\text{argument}} + e^{-\text{argument}}}
\]

Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=tanh(0);</td>
<td>0</td>
</tr>
<tr>
<td>x=tanh(0.5);</td>
<td>0.4621171573</td>
</tr>
<tr>
<td>x=tanh(-0.5);</td>
<td>-0.462117157</td>
</tr>
</tbody>
</table>

TYPEOF Function

Returns a value that indicates whether the argument is character or numeric.

**Category:** Character

**Restrictions:** The TYPEOF function is used exclusively with the Graph Template Language (GTL) and in WHERE clauses, but not in DATA steps. This function is not valid on the CAS server.

**Syntax**

\[\text{TYPEOF}(\text{column})\]

**Required Argument**

\(\text{column}\)

Can have one of the following values:

- **C** indicates that the argument is a character value.
- **N** indicates that the argument is a numeric value.

TZONEID Function

Returns the current time zone ID.

**Category:** Date and Time

**Alias:** TZID
Restrictions: This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility. This function is not valid in the CAS server.

Syntax

TZONEID<time-zone-id>

Optional Argument

time-zone-id

specifies a region or area value that is defined by SAS. When you specify a zone ID, the time zone that SAS uses is determined by the time zone name and daylight savings time rules.

Details

The TZONEID function returns a blank value if the TIMEZONE= option is blank or a user-defined time zone is specified.

The TZONEID function validates the timezone ID. If you specify the timezone ID, the function returns the timezone ID if it is valid or returns a blank value if the ID is invalid.

Example

In the first example, the TIMEZONE option is set to JST. In the second example, TIMEZONE is set to a blank value. In the third example TIMEZONE is set to user-specified time zone. In the fourth example a valid timezoneid and an invalid timezoneid is displayed.

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>options timezone=jst;</td>
<td>tزيد=ASIA/TOKYO</td>
</tr>
<tr>
<td>data <em>null</em> ;</td>
<td></td>
</tr>
<tr>
<td>tzid=tzoneid() ;</td>
<td></td>
</tr>
<tr>
<td>put tzid=;</td>
<td></td>
</tr>
<tr>
<td>run;</td>
<td></td>
</tr>
</tbody>
</table>

| options timezone='';                            | tزيد=           |
| data _null_ ;                                   |                  |
|     tzid=tzoneid() ;                           |                  |
|     put tzid=;                                  |                  |
| run;                                           |                  |

| options timezone='xxx-12';                      | tزيد=           |
| /* user defined timezone */                     |                  |
| data _null_ ;                                   |                  |
|     tzid=tzoneid() ;                           |                  |
|     put tzid=;                                  |                  |
| run;                                           |                  |
### TZONENAME Function

Returns the current standard or daylight savings time, time zone name.

**Category:** Date and Time

**Restrictions:** This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility. This function is not valid in the CAS server.

#### Syntax

TZONENAME()<time-zone-id,datetime>

**Optional Arguments**

- **time-zone-id**
  
  Specifies a region/area value that is defined by SAS. When you specify a zone ID, the time zone that SAS uses is determined by time zone name and daylight savings time rules.

  **See** For a list of the time zone IDs, see “Time Zone IDs and Time Zone Names” in SAS Viya National Language Support (NLS): Reference Guide.

- **datetime**
  
  Specifies a SAS datetime value.

#### Details

The TZONENAME function returns a blank value if the TIMEZONE= option is blank.

The TZONENAME function returns the timezone name based on the specified timezone and datetime. If the SAS datetime is not specified, then the current date is used. If TZID is not specified, then the timezone ID that is specified with the TIMEZONE= option is used.

#### Example

In the first example, the TIMEZONE option is set to a blank value. In the second example, TIMEZONE is set to timezone name, JST. In the third example, TIMEZONE
is set to a user-specified time zone. In the fourth example, TIMEZONE is set to a time zone ID.

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>options tz=''; data <em>null</em>; tzname=tzonename(); put tzname =; run;</td>
<td>tzname=</td>
</tr>
<tr>
<td>options tz='jst'; data <em>null</em>; tzname=tzonename(); put tzname =; run;</td>
<td>tzname=JST</td>
</tr>
<tr>
<td>options tz='xxx-12'; data <em>null</em>; tzname=tzonename(); put tzname =; run;</td>
<td>tzname=XXX</td>
</tr>
<tr>
<td>options tz='American/Chicago'; data_null_; tzname=tzonename('01SEP2014:01:01:01'dt); put tzname =; run;</td>
<td>tzname=CDT</td>
</tr>
</tbody>
</table>

**TZONEOFF Function**

Returns the user time zone offset.

**Category:** Date and Time

**Restrictions:**

This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility.

This function is not valid in the CAS server.

**Syntax**

TZONEOFF()

TZONEOFF(<time-zone-id, datetime>)

**Optional Arguments**

**time-zone-id**

specifies a region/area value that is defined by SAS. When you specify a time zone ID, the time zone that SAS uses is determined by time zone name and daylight savings time rules.
See For a list of time zone IDs, see “Time Zone IDs and Time Zone Names” in SAS Viya National Language Support (NLS): Reference Guide.

datetime
   specifies a SAS datetime value.

Details
If no arguments are specified, the TZONEOFF function returns the time zone offset for the specified TIMEZONE option. The TZONEOFF (time-zone-id) function with the time zone ID argument returns the time zone offset for the specified time zone ID. The TZONEOFF function with the time zone ID argument returns the time zone offset for the specified time zone name. If the time zone name is not valid for the current locale, you receive an error. We recommend that you use the time zone ID, since it is not locale dependent.

If SASDTM is not provided, TZONEOFF returns the current timezone offset. If SASDTM is provided, it returns the offset to get the local time for specified time value.

Example
The first example has no argument, so the TZONEOFF function returns an offset for the current SAS session. The second example returns an offset based on a specific time zone ID. The third example returns an offset based on a specific time zone ID and a specific date and time. The fourth example returns an offset based on the Time Zone option and a specific date.

If the SAS datetime is not specified, then the TZONEOFF function returns the current timezone offset. If the SAS datetime is specified, then the function returns the offset to provide the local time for the specified time value.

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>option TIMEZONE='AUSTRALIA/MELBOURNE';</td>
<td>39600</td>
</tr>
<tr>
<td>%PUT %SYSFUNC(TZONEOFF());</td>
<td></td>
</tr>
<tr>
<td>option TIMEZONE='AUSTRALIA/MELBOURNE';</td>
<td>3600</td>
</tr>
<tr>
<td>%PUT %SYSFUNC(TZONEOFF('EUROPE/ROME'));</td>
<td></td>
</tr>
<tr>
<td>data <em>null</em>;</td>
<td></td>
</tr>
<tr>
<td>dt1='05DEC2012:08:17:52'dt  ;</td>
<td></td>
</tr>
<tr>
<td>dt2='05JUN2012:08:17:52'dt  ;</td>
<td></td>
</tr>
<tr>
<td>offset1= TZONEOFF('EUROPE/MOSCOW', dt1) ;</td>
<td>offset1=10800</td>
</tr>
<tr>
<td>offset2= TZONEOFF('EUROPE/MOSCOW', dt2) ;</td>
<td>offset2=14400</td>
</tr>
<tr>
<td>put offset1= / offset2= ;</td>
<td></td>
</tr>
<tr>
<td>run ;</td>
<td></td>
</tr>
<tr>
<td>option TIMEZONE='EUROPE/MOSCOW' ;</td>
<td>offset1=10800</td>
</tr>
<tr>
<td>data <em>null</em>;</td>
<td>offset2=14400</td>
</tr>
<tr>
<td>dt1='05DEC2012:08:17:52'dt  ;</td>
<td></td>
</tr>
<tr>
<td>dt2='05JUN2012:08:17:52'dt  ;</td>
<td></td>
</tr>
<tr>
<td>offset1= TZONEOFF(dt1) ;</td>
<td></td>
</tr>
<tr>
<td>offset2= TZONEOFF(dt2) ;</td>
<td></td>
</tr>
<tr>
<td>put offset1= / offset2= ;</td>
<td></td>
</tr>
<tr>
<td>run ;</td>
<td></td>
</tr>
</tbody>
</table>
TZONES2U Function

Converts a SAS date time value to a UTC date time value.

**Category:** Date and Time

**Restrictions:** This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility. This function is not valid in the CAS server.

**Syntax**

`TZONES2U <datetime, time-zone-id>`

**Optional Arguments**

- **datetime**
  - Specifies a SAS datetime value.

- **time-zone-id**
  - Specifies a region or area value that is defined by SAS. When you specify a time zone ID, the time zone that SAS uses is determined by time zone name and daylight savings time rules.

**See** For a list of time zone IDs, see “Time Zone IDs and Time Zone Names” in SAS Viya National Language Support (NLS): Reference Guide

**Details**

The `TZONES2U()` function returns UTC-based time for the specified TIMEZONE. The `TZONES2U`(time-zone-id) function with the time zone ID argument returns UTC-based time for the specified time zone ID. If the time zone name is not valid for the current locale, you receive an error.

**Example**

The following example converts a SAS date time into UTC time.

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>option locale=ja_JP TZ='JST' ;</td>
<td>dt=1667722672</td>
</tr>
<tr>
<td>data <em>null</em>;</td>
<td></td>
</tr>
<tr>
<td>dt='05Nov2012:08:17:52'dt ;</td>
<td>utc1=2012-11-04T23:17:52+00:00</td>
</tr>
<tr>
<td>utc1 = tzones2u(dt) ;</td>
<td></td>
</tr>
<tr>
<td>utc2 = tzones2u(dt,'ASIA/TOKYO') ;</td>
<td>dt=1667722672</td>
</tr>
<tr>
<td>utc3 = tzones2u(dt,'JST') ;</td>
<td>utc2=2012-11-04T23:17:52+00:00</td>
</tr>
<tr>
<td>put dt= /utc1= is8601dz. //;</td>
<td></td>
</tr>
<tr>
<td>put dt= /utc2= is8601dz. //;</td>
<td></td>
</tr>
<tr>
<td>put dt= /utc3= is8601dz. //;</td>
<td></td>
</tr>
<tr>
<td>run ;</td>
<td></td>
</tr>
</tbody>
</table>
TZONEU2S Function

Converts a UTC date time value to a SAS date time value.

**Category:** Date and Time

**Restrictions:** This function is assigned an I18N Level 2 status, and is designed for use with SBCS, DBCS, and MBCS (UTF8). For more information, see Internationalization Compatibility.

This function is not valid in the CAS server.

**Syntax**

TZONEU2S <UTC date time value, time-zone id>

**Optional Arguments**

- **UTC date time value**
  - specifies a Coordinated Universal Time (UTC) datetime value

- **time-zone id**
  - specifies a region or area value that is defined by SAS. When you specify a zone ID, the time zone that SAS uses is determined by the time zone name and daylight savings time rules.

**Details**

The TZONEU2S(datetime) function returns the SAS datetime for a UTC time for the specified TIMEZONE option. The TZONEU2S(datetime, time-zone-id) function with the time zone ID argument, returns the SAS datetime for the UTC time for the specified time zone ID. If the time zone name is not valid for the current locale, you receive an error.

**Example**

The following example converts a UTC date time to three specific SAS date time values.

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>option locale=fr_FR TZ='AMERICA/DENVER';</td>
<td>sdt1=01SEP12:20:34:56</td>
</tr>
<tr>
<td>data <em>null</em>;</td>
<td>sdt2=02SEP12:03:34:56</td>
</tr>
<tr>
<td>utc_date = '2012-09-02T02:34:56+00:00';</td>
<td>sdt3=02SEP12:03:34:56</td>
</tr>
<tr>
<td>udt = input(utc_date,is8601dz.);</td>
<td></td>
</tr>
<tr>
<td>sdt1 = tzoneu2s(udt);</td>
<td></td>
</tr>
<tr>
<td>sdt2 = tzoneu2s(udt,'EUROPE/AMSTERDAM');</td>
<td></td>
</tr>
<tr>
<td>sdt3 = tzoneu2s(udt,'CET');</td>
<td></td>
</tr>
<tr>
<td>put sdt1= datetime. / sdt2= datetime. / sdt3= datetime.;</td>
<td></td>
</tr>
<tr>
<td>run;</td>
<td></td>
</tr>
</tbody>
</table>
URLDECODE Function

Returns a string that was decoded using the URL escape syntax.

Category: Web Tools

Restrictions: This function is assigned an I18N Level 1 status. If possible, avoid I18N Level 1 functions if you are using a non-English language. Under certain circumstances, the I18N Level 1 functions might not work correctly with Double-Byte Character Set (DBCS) or Multi-Byte Character Set (MBCS) encodings. For more information, see Internationalization Compatibility.

This function is not valid on the CAS server.

Syntax

URLDECODE(argument)

Required Argument

argument specifies a character constant, variable, or expression.

Details

Length of Returned Variable in a DATA Step

If the URLDECODE function returns a value to a variable that has not previously been assigned a length, then that variable is given the length of the argument.

The URLDECODE and URLENCODE functions do not verify that the bytes produced by the escape sequences are valid characters based on the encoding.

The Basics

The URL escape syntax is used to hide characters that might otherwise be significant when used in a URL.

A URL escape sequence can be one of the following:

• a plus sign, which is replaced by a blank

• a sequence of three characters beginning with a percent sign and followed by two hexadecimal characters, which is replaced by a single character that has the specified hexadecimal value.

Argument can be decoded using either SAS session encoding or UTF-8 encoding. To decode argument by using the SAS session encoding, set the system option URLENCODING=SESSION. To decode argument by using UTF-8 encoding, set the system option URLENCODING=UTF8.
Example

The following SAS statements produce these results using SAS session decoding:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1=urldecode('abc+def'); put x1;</td>
<td>abc def</td>
</tr>
<tr>
<td>x2=urldecode('why%3F'); put x2;</td>
<td>why?</td>
</tr>
<tr>
<td>x3=urldecode('%41%42%43%23%31'); put x3;</td>
<td>ABC#1</td>
</tr>
</tbody>
</table>

See Also

Functions:

• “URLENCODE Function” on page 1088

URLENCODE Function

Returns a string that was encoded using the URL escape syntax.

Category: Web Tools

Restrictions: This function is assigned an I18N Level 1 status. If possible, avoid I18N Level 1 functions if you are using a non-English language. Under certain circumstances, the I18N Level 1 functions might not work correctly with Double-Byte Character Set (DBCS) or Multi-Byte Character Set (MBCS) encodings. For more information, see Internationalization Compatibility.

This function is not valid on the CAS server.

Syntax

URLENCODE(argument)

Required Argument

argument

specifies a character constant, variable, or expression.

Details

Length of Returned Variable in a DATA Step

If the URLENCODE function returns a value to a variable that has not previously been assigned a length, then that variable is given a length of 200 bytes.

The URLDECODE and URLENCODE functions do not verify that the bytes produced by the escape sequences are valid characters based on the encoding.
The Basics

_Argument_ can be encoded using either SAS session encoding or UTF-8 encoding. To encode _argument_ by using the SAS session encoding, set the system option URLENCODING=SESSION. To encode _argument_ by using UTF-8 encoding, set the system option URLENCODING=UTF8.

The URLENCODE function encodes characters that might otherwise be significant when used in a URL. This function encodes all characters except for the following:

- all alphanumeric characters
- dollar sign ($)
- hyphen (-)
- underscore (_)
- at sign (@)
- period (.)
- exclamation point (!)
- asterisk (*)
- open parenthesis ( ( ) and close parenthesis ( )
- comma (,).

*Note:* The encoded string might be longer than the original string. Ensure that you consider the additional length when you use this function.

Example

The following SAS statements produce these results using SAS session encoding:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1=urlencode('abc def');</td>
<td>abc%20def</td>
</tr>
<tr>
<td>put x1;</td>
<td></td>
</tr>
<tr>
<td>x2=urlencode('why?');</td>
<td>why%3F</td>
</tr>
<tr>
<td>put x2;</td>
<td></td>
</tr>
<tr>
<td>x3=urlencode('ABC%1');</td>
<td>ABC%231</td>
</tr>
<tr>
<td>put x3;</td>
<td></td>
</tr>
</tbody>
</table>

See Also

Functions:

- “URLDECODE Function” on page 1087
Restriction: This function is not valid on the CAS server.

Syntax

VARFMT(data-set-id, variable-number)

Required Arguments

data-set-id
specifies the data set identifier that the OPEN function returns.

variable-number
specifies the number of the variable's position in the SAS data set.

Tips
This number is next to the variable in the list that is produced by the CONTENTS procedure.
The VARNUM function returns this number.

Details
If no format has been assigned to the variable, a blank string is returned.

Examples

Example 1: Using VARFMT to Obtain the Format of the Variable NAME
This example obtains the format of the variable NAME in the SAS data set MYDATA.

```sas
%let dsid=%sysfunc(open(mydata, i));
%if &dsid %then
%do;
  %let fmt=%sysfunc(varfmt(&dsid,
   %sysfunc(varnum
   (&dsid, NAME))));
  %let rc=%sysfunc(close(&dsid));
%end;
```

Example 2: Using VARFMT to Obtain the Format of All the Numeric Variables in a Data Set
This example creates a data set that contains the name and formatted content of each numeric variable in the SAS data set MYDATA.

```sas
data vars;
  length name $ 8 content $ 12;
  drop dsid i num rc fmt;
  dsid=open("mydata", "i");
  num=attrn(dsid, "nvars");
  do while (fetch(dsid)=0);
    do i=1 to num;
      name=varname(dsid, i);
      if (vartype(dsid, i)='N') then do;
        fmt=varfmt(dsid, i);
        if fmt='' then fmt="BEST12.");
      ```
content=putc(putn(getvarn(dsid,i),fmt),"$char12.");
output;
end;
end;
rc=close(dsid);
run;

See Also

Functions:

• “VARINFMT Function” on page 1091
• “VARNUM Function” on page 1095

VARINFMT Function

Returns the informat that is assigned to a SAS data set variable.

**Category:** SAS File I/O

**Restriction:** This function is not valid on the CAS server.

**Syntax**

VARINFMT(data-set-id, variable-number)

**Required Arguments**

*data-set-id*

specifies the data set identifier that the OPEN function returns.

*variable-number*

specifies the number of the variable's position in the SAS data set.

**Tips**

This number is next to the variable in the list that is produced by the CONTENTS procedure.

The VARNUM function returns this number.

**Details**

If no informat has been assigned to the variable, a blank string is returned.

**Examples**

**Example 1: Using VARINFMT to Obtain the Informat of the Variable NAME**

This example obtains the informat of the variable NAME in the SAS data set MYDATA.

```sas
%let dsid=%sysfunc(open(mydata, i));
%if &dsid %then
```
Example 2: Using VARINFMT to Obtain the Informat of All the Variables in a Data Set

This example creates a data set that contains the name and informat of the variables in MYDATA.

```sas
data vars;
  length name $ 8 informat $ 10;
  drop dsid i num rc;
  dsid=open("mydata", "i");
  num=attrn(dsid, "nvars");
  do i=1 to num;
    name=varname(dsid, i);
    informat=varinfmt(dsid, i);
    output;
  end;
  rc=close(dsid);
run;
```

See Also

Functions:
- “OPEN Function” on page 1010
- “VARFMT Function” on page 1089
- “VARNUM Function” on page 1095

VARLABEL Function

Returns the label that is assigned to a SAS data set variable.

**Category:** SAS File I/O

**Restriction:** This function is not valid on the CAS server.

**Syntax**

```
VARLABEL(data-set-id, variable-number)
```

**Required Arguments**

- `data-set-id` specifies the data set identifier that the OPEN function returns.
- `variable-number` specifies the number of the variable's position in the SAS data set.
**Tips**

This number is next to the variable in the list that is produced by the CONTENTS procedure.

The VARNUM function returns this number.

**Details**

If no label has been assigned to the variable, a blank string is returned.

**Comparisons**

VLABEL returns the label that is associated with the given variable.

**Example**

This example obtains the label of the variable NAME in the SAS data set MYDATA.

**Example Code 3.1  Obtaining the Label of the Variable NAME**

```
%let dsid=%sysfunc(open(mydata, i));
%if &dsid %then
  %do;
    %let fmt=%sysfunc(varlabel(&dsid,
         %sysfunc(varnum
         (&dsid, NAME))));
    %let rc=%sysfunc(close(&dsid));
  %end;
```

**See Also**

Functions:

- “OPEN Function” on page 1010
- “VARNUM Function” on page 1095

---

**VARLEN Function**

Returns the length of a SAS data set variable.

**Category:** SAS File I/O

**Restriction:** This function is not valid on the CAS server.

**Syntax**

```
VARLEN(data-set-id, variable-number)
```

**Required Arguments**

- `data-set-id`
  - specifies the data set identifier that the OPEN function returns.

- `variable-number`
  - specifies the number of the variable's position in the SAS data set.
Tips  This number is next to the variable in the list that is produced by the CONTENTS procedure.

The VARNUM function returns this number.

Details

VLENGTH returns the compile-time (allocated) size of the given variable.

Example

- This example obtains the length of the variable ADDRESS in the SAS data set MYDATA.

```sas
%let dsid=%sysfunc(open(mydata, i));
%if &dsid %then
  %do;
    %let len=%sysfunc(varlen(&dsid, %sysfunc(varnum (&dsid,ADDRESS))));
    %let rc=%sysfunc(close(&dsid));
  %end;
```

- This example creates a data set that contains the name, type, and length of the variables in MYDATA.

```sas
data vars;
  length name $ 8 type $ 1;
  drop dsid i num rc;
  dsid=open("mydata", "i");
  num=attrn(dsid, "nvars");
  do i=1 to num;
    name=varname(dsid, i);
    type=vartype(dsid, i);
    length=varlen(dsid, i);
    output;
  end;
  rc=close(dsid);
run;
```

See Also

Functions:

- “OPEN Function” on page 1010
- “VARNUM Function” on page 1095

VARNAME Function

Returns the name of a SAS data set variable.

Category:  SAS File I/O

Restriction:  This function is not valid on the CAS server.
Syntax

VARNAME(data-set-id, variable-number)

Required Arguments

data-set-id
  specifies the data set identifier that the OPEN function returns.

variable-number
  specifies the number of the variable's position in the SAS data set.

Tips
  This number is next to the variable in the list that is produced by the CONTENTS procedure.

  The VARNUM function returns this number.

Example

This example copies the names of the first five variables in the SAS data set CITY (or all of the variables if there are fewer than five) into a macro variable.

%macro names;
  %let dsid=%sysfunc(open(city, i));
  %let varlist=
  %do i=1 %to %sysfunc(min(5,%sysfunc(attrn(&dsid, nvars))));
    %let varlist=&varlist %sysfunc(varname(&dsid, &i));
  %end;
  %put varlist=&varlist;
%mend names;
%names

See Also

Functions:
  • “OPEN Function” on page 1010
  • “VARNUM Function” on page 1095

VARNUM Function

Returns the number of a variable's position in a SAS data set.

  Category: SAS File I/O

  Restriction: This function is not valid on the CAS server.
Required Arguments

data-set-id
  specifies the data set identifier that the OPEN function returns.

variable-name
  specifies the variable's name.

Details

VARNUM returns the number of a variable's position in a SAS data set, or 0 if the variable is not in the SAS data set. This is the same variable number that is next to the variable in the output from PROC CONTENTS.

Example

• This example obtains the number of a variable's position in the SAS data set CITY, given the name of the variable.

  %let dsid=%sysfunc(open(city, i));
  %let citynum=%sysfunc(varnum(&dsid, CITYNAME));
  %let rc=%sysfunc(fetch(&dsid));
  %let cityname=%sysfunc(getvarc
      (&dsid, &citynum));

• This example creates a data set that contains the name, type, format, informat, label, length, and position of the variables in Sasuser.Houses.

  data vars;
    length name $ 8 type $ 1
        format informat $ 10 label $ 40;
    drop dsid i num rc;
    dsid=open("sasuser.houses", "i");
    num=attrn(dsid, "nvars");
    do i=1 to num;
      name=varname(dsid, i);
      type=vartype(dsid, i);
      format=varfmt(dsid, i);
      informat=varinfmt(dsid, i);
      label=varlabel(dsid, i);
      length=varlen(dsid, i);
      position=varnum(dsid, name);
      output;
    end;
    rc=close(dsid);
  run;

See Also

Functions:

• “OPEN Function” on page 1010
• “VARNAME Function” on page 1094
VARTYPE Function

Returns the data type of a SAS data set variable.

**Category:** SAS File I/O

**Restriction:** This function is not valid on the CAS server.

## Syntax

\[ \text{VARTYPE}( \text{data-set-id, variable-number} ) \]

### Required Arguments

- **data-set-id**
  - specifies the data set identifier that the OPEN function returns.

- **variable-number**
  - specifies the number of the variable's position in the SAS data set.

### Tips

This number is next to the variable in the list that is produced by the CONTENTS procedure.

The VARNUM function returns this number.

## Details

VARTYPE returns C for a character variable or N for a numeric variable.

## Examples

### Example 1: Using VARTYPE to Determine Which Variables Are Numeric

This example places the names of all the numeric variables of the SAS data set MYDATA into a macro variable.

```sas
%let dsid=%sysfunc(open(mydata, i));
%let varlist=;
%do i=1 %to %sysfunc(attrn(&dsid, nvars));
  %if (%sysfunc(vartype(&dsid, &i))=N) %then
    %let varlist=&varlist %sysfunc(varname (&dsid, &i));
%end;
%let rc=%sysfunc(close(&dsid));
```

### Example 2: Using VARTYPE to Determine Which Variables Are Character

This example creates a data set that contains the name and formatted contents of each character variable in the SAS data set MYDATA.

```sas
data vars;
  length name $ 8 content $ 20;
  drop dsid i num fmt rc;
```
dsid=open("mydata", "i");
num=attrn(dsid, "nvars");
do while (fetch(dsid)=0);
  do i=1 to num;
    name=varname(dsid, i);
    fmt=varfmt(dsid, i);
    if (vartype(dsid, i)='C') then do;
      content=getvarc(dsid, i);
      if (fmt ne '') then
        content=left(putc(content, fmt));
      output;
    end;
  end;
end;
rc=close(dsid);
run;

See Also

Functions:

• “VARNUM Function” on page 1095

ZIPCITY Function

Returns a city name and the two-character postal code that corresponds to a ZIP code.

Category: State and ZIP code

Restriction: This function is not valid on the CAS server.

Syntax

ZIPCITY(ZIP-code)

Required Argument

ZIP-code

specifies a numeric or character expression that contains a five-digit ZIP code.

Tip If the value of ZIP-code begins with leading zeros, you can enter the value without the leading zeros. For example, if you enter 1040, ZIPCITY assumes that the value is 01040.

Details

The Basics

If the ZIPCITY function returns a value to a variable that has not yet been assigned a length, by default the variable is assigned a length of 20.

ZIPCITY returns a city name and the two-character postal code that corresponds to its five-digit ZIP code argument. ZIPCITY returns the character values in mixed-case. If the ZIP code is unknown, ZIPCITY returns a blank value.
Note: The Sashelp.Zipcode data set must be present when you use this function. If you remove the data set, ZIPCITY returns unexpected results.

How the ZIP Code Is Translated to the State Postal Code
To determine which state corresponds to a particular ZIP code, this function uses a zone table that consists of the start and end ZIP code values for each state. It then finds the corresponding state for that range of ZIP codes. The zone table consists of start and end ZIP code values for each state to allow for exceptions, and does not validate ZIP code values.

With very few exceptions, a zone does not span multiple states. The exceptions are included in the zone table. It is possible for new zones or new exceptions to be added by the U.S. Postal Service at any time. However, SAS software is updated only with each new release of the product.

Determining When the State Postal Code Table Was Last Updated
The Sashelp.Zipcode data set contains postal code information that is used with ZIPCITY and other ZIP code functions. To determine when this data set was last updated, execute PROC CONTENTS:

```
proc contents data=sashelp.zipcode;
run;
```

Output from the CONTENTS procedure provides the date of the last update, along with the contents of the Sashelp.Zipcode data set.

Note: You can download the latest version of the Sashelp.Zipcode file from the Technical Support Web site. Select Zipcode Data Set from the Name column to begin the download process. You must execute the CIMPORT procedure after you download and unzip the data set.

If you use an invalid ZIP code (one that is not found in the Sashelp.Zipcode data set), then SAS returns a message that indicates there is an invalid argument in the ZIPCITY function.

Comparisons
The ZIPCITY, ZIPNAME, ZIPNAMEL, and ZIPSTATE functions accept the same argument but return different values:

- ZIPCITY returns the name of the city in mixed-case and the two-character postal code that corresponds to its five-digit ZIP code argument.
- ZIPNAME returns the uppercase name of the state or U.S. territory that corresponds to its five-digit ZIP code argument.
- ZIPNAMEL returns the mixed case name of the state or U.S. territory that corresponds to its five-digit ZIP code argument.
- ZIPSTATE returns the uppercase two-character state postal code (or worldwide GSA geographic code for U.S. territories) that corresponds to its five-digit ZIP code argument.

Example
The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>city1=zipcity(27511); put city1;</td>
<td>Cary, NC</td>
</tr>
<tr>
<td>length zip $10.; zip='90049-1392'; zip=substr(zip, 1, 5); city2=zipcity(zip); put city2;</td>
<td>Los Angeles, CA</td>
</tr>
<tr>
<td>city3=zipcity(4338); put city3;</td>
<td>Augusta, ME</td>
</tr>
<tr>
<td>city4=zipcity(01040); put city4;</td>
<td>Holyoke, MA</td>
</tr>
</tbody>
</table>

See Also

Functions:
- “ZIPFIPS Function” on page 1101
- “ZIPNAME Function” on page 1103
- “ZIPNAMEL Function” on page 1105
- “ZIPSTATE Function” on page 1106

ZIPCITYDISTANCE Function

Returns the geodetic distance between two ZIP code locations.

Categories: Distance State and ZIP code

Alias: ZIPCITYDIST

Restriction: This function is not valid on the CAS server.

Syntax

ZIPCITYDISTANCE(ZIP-code-1, ZIP-code-2)

Required Argument

ZIP-code

specifies a numeric or character expression that contains the ZIP code of a location in the United States of America.
Details

The ZIPCITYDISTANCE function returns the geodetic distance in miles between two ZIP code locations. The centroid of each ZIP code is used in the calculation.

The Sashelp.Zipcode data set must be present when you use this function. If you remove the data set, then ZIPCITYDISTANCE returns unexpected results.

The Sashelp.Zipcode data set contains postal code information that is used with ZIPCITYDISTANCE and other ZIP code functions. To determine when this data set was last updated, execute PROC CONTENTS:

```sas
proc contents data=sashelp.zipcode;
run;
```

Output from the CONTENTS procedure provides the date of the last update, along with the contents of the SASHELP.ZIPCODE data set.

Note: You can download the latest version of the Sashelp.Zipcode file from the SAS external website. The file is located at the Technical Support Web site. Select Zipcode Data Set from the Name column to begin the download process. You must execute the CIMPORT procedure after you download and unzip the data set.

If you use an invalid ZIP code (one that is not found in the Sashelp.Zipcode data set), then SAS returns a message that indicates there is an invalid argument in the ZIPCITYDISTANCE function.

Example

In the following example, the first ZIP code identifies a location in San Francisco, CA, and the second ZIP code identifies a location in Bangor, ME. ZIPCITYDISTANCE returns the distance in miles between these two locations.

```sas
data _null_;
    distance=zipcitydistance('94103', '04401');
    put 'Distance from San Francisco, CA, to Bangor, ME: ' distance 4. ' miles';
run;
```

SAS writes the following output to the log:

```
Distance from San Francisco, CA, to Bangor, ME: 2782 miles
```

See Also

Functions:

- “ZIPCITY Function” on page 1098
Syntax

ZIPFIPS(ZIP-code)

Required Argument

ZIP-code
specifies a numeric or character expression that contains a five-digit ZIP code.

Tip
If the value of ZIP-code begins with leading zeros, you can enter the value without the leading zeros. For example, if you enter 1040, ZIPFIPS assumes that the value is 01040.

Details

The Basics
The ZIPFIPS function returns the two-digit numeric U.S. Federal Information Processing Standards (FIPS) code that corresponds to its five-digit ZIP code argument.

How the ZIP Code Is Translated to the State Postal Code
To determine which state corresponds to a particular ZIP code, this function uses a zone table that consists of the start and end ZIP code values for each state. It then finds the corresponding state for that range of ZIP codes. The zone table consists of start and end ZIP code values for each state to allow for exceptions, and does not validate ZIP code values.

With very few exceptions, a zone does not span multiple states. The exceptions are included in the zone table. It is possible for new zones or new exceptions to be added by the U.S. Postal Service at any time. However, SAS software is updated only with each new release of the product.

Example
The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>fips1=zipfips('27511');</td>
<td></td>
</tr>
<tr>
<td>put fips1;</td>
<td>37</td>
</tr>
<tr>
<td>fips2=zipfips('01040');</td>
<td></td>
</tr>
<tr>
<td>put fips2;</td>
<td>25</td>
</tr>
<tr>
<td>fips3=zipfips(1040);</td>
<td></td>
</tr>
<tr>
<td>put fips3;</td>
<td>25</td>
</tr>
<tr>
<td>fips4=zipfips(59017);</td>
<td></td>
</tr>
<tr>
<td>put fips4;</td>
<td>30</td>
</tr>
<tr>
<td>fips5=zipfips(24862);</td>
<td></td>
</tr>
<tr>
<td>put fips5;</td>
<td>54</td>
</tr>
</tbody>
</table>
See Also

Functions:
- “ZIPCITY Function” on page 1098
- “ZIPNAME Function” on page 1103
- “ZIPNAMEL Function” on page 1105
- “ZIPSTATE Function” on page 1106

ZIPNAME Function

Converts ZIP codes to uppercase state names.

**Category:** State and ZIP code

**Restriction:** This function is not valid on the CAS server.

**Syntax**

ZIPNAME(ZIP-code)

**Required Argument**

ZIP-code specifies a numeric or character expression that contains a five-digit ZIP code.

**Tip** If the value of ZIP-code begins with leading zeros, you can enter the value without the leading zeros. For example, if you enter 1040, ZIPNAME assumes that the value is 01040.

**Details**

**The Basics**

If the ZIPNAME function returns a value to a variable that has not yet been assigned a length, by default the variable is assigned a length of 20.

ZIPNAME returns the name of the state or U.S. territory that corresponds to its five-digit ZIP code argument. ZIPNAME returns character values up to 20 characters long, all in uppercase.

**How the ZIP Code Is Translated to the State Postal Code**

To determine which state corresponds to a particular ZIP code, this function uses a zone table that consists of the start and end ZIP code values for each state. It then finds the corresponding state for that range of ZIP codes. The zone table consists of start and end ZIP code values for each state to allow for exceptions, and does not validate ZIP code values.

With very few exceptions, a zone does not span multiple states. The exceptions are included in the zone table. It is possible for new zones or new exceptions to be added by the U.S. Postal Service at any time. However, SAS software is updated only with each new release of the product.
Comparisons

The ZIPCITY, ZIPNAME, ZIPNAMEL, and ZIPSTATE functions accept the same argument but return different values:

- ZIPCITY returns the mixed-case name of the city and the two-character postal code that corresponds to its five-digit ZIP code argument.
- ZIPNAME returns the uppercase name of the state or U.S. territory that corresponds to its five-digit ZIP code argument.
- ZIPNAMEL returns the mixed-case name of the state or U.S. territory that corresponds to its five-digit ZIP code argument.
- ZIPSTATE returns the uppercase two-character state postal code (or worldwide GSA geographic code for U.S. territories) that corresponds to its five-digit ZIP code argument.

Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>state1=zipname('27511'); put state1;</td>
<td>NORTH CAROLINA</td>
</tr>
<tr>
<td>state2=zipname('01040'); put state2;</td>
<td>MASSACHUSETTS</td>
</tr>
<tr>
<td>state3=zipname(1040); put state3;</td>
<td>MASSACHUSETTS</td>
</tr>
<tr>
<td>state4=zipname('59017'); put state4;</td>
<td>MONTANA</td>
</tr>
<tr>
<td>length zip $10.; zip='90049-1392'; zip=substr(zip, 1, 5); state5=zipname(zip); put state5;</td>
<td>CALIFORNIA</td>
</tr>
</tbody>
</table>

See Also

Functions:

- “ZIPCITY Function” on page 1098
- “ZIPFIPS Function” on page 1101
- “ZIPNAMEL Function” on page 1105
- “ZIPSTATE Function” on page 1106
ZIPNAMEL Function

Converts ZIP codes to mixed case state names.

- **Category:** State and ZIP code
- **Restriction:** This function is not valid on the CAS server.

**Syntax**

```
ZIPNAMEL(ZIP-code)
```

**Required Argument**

- **ZIP-code** specifies a numeric or character expression that contains a five-digit ZIP code.

**Tip** If the value of ZIP-code begins with leading zeros, you can enter the value without the leading zeros. For example, if you enter 1040, ZIPNAMEL assumes that the value is 01040.

**Details**

**The Basics**

If the ZIPNAMEL function returns a value to a variable that has not yet been assigned a length, by default the variable is assigned a length of 20.

ZIPNAMEL returns the name of the state or U.S. territory that corresponds to its five-digit ZIP code argument. ZIPNAMEL returns mixed-case character values up to 20 characters long.

**How the ZIP Code Is Translated to the State Postal Code**

To determine which state corresponds to a particular ZIP code, this function uses a zone table that consists of the start and end ZIP code values for each state. It then finds the corresponding state for that range of ZIP codes. The zone table consists of start and end ZIP code values for each state to allow for exceptions, and does not validate ZIP code values.

With very few exceptions, a zone does not span multiple states. The exceptions are included in the zone table. It is possible for new zones or new exceptions to be added by the U.S. Postal Service at any time. However, SAS software is updated only with each new release of the product.

**Comparisons**

The ZIPCITY, ZIPNAME, ZIPNAMEL, and ZIPSTATE functions accept the same argument but return different values:

- ZIPCITY returns the name of the city in mixed-case and the two-character postal code that corresponds to its five-digit ZIP code argument.
- ZIPNAME returns the uppercase name of the state or U.S. territory that corresponds to its five-digit ZIP code argument.
ZIPNAMEL returns the mixed-case name of the state or U.S. territory that corresponds to its five-digit ZIP code argument.

ZIPSTATE returns the uppercase two-character state postal code (or worldwide GSA geographic code for U.S. territories) that corresponds to its five-digit ZIP code argument.

**Example**

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>state1=zipnamel('27511'); put state1;</td>
<td>North Carolina</td>
</tr>
<tr>
<td>state2=zipnamel('01040'); put state2;</td>
<td>Massachusetts</td>
</tr>
<tr>
<td>state3=zipnamel(1040); put state3;</td>
<td>Massachusetts</td>
</tr>
<tr>
<td>state4=zipnamel(59017); put state4;</td>
<td>Montana</td>
</tr>
<tr>
<td>length zip $10.; zip='90049-1392'; substr(zip, 1, 5); state5=zipnamel(zip);</td>
<td>California</td>
</tr>
<tr>
<td>put state5;</td>
<td></td>
</tr>
</tbody>
</table>

**See Also**

**Functions:**
- “ZIPCITY Function” on page 1098
- “ZIPFIPS Function” on page 1101
- “ZIPNAME Function” on page 1103
- “ZIPSTATE Function” on page 1106

**ZIPSTATE Function**

Converts ZIP codes to two-character state postal codes.

**Category:** State and ZIP code

**Restriction:** This function is not valid on the CAS server.

**Syntax**

```
ZIPSTATE(ZIP-code)
```
Required Argument

ZIP-code
specifies a numeric or character expression that contains a valid five-digit ZIP code.

Tip If the value of ZIP-code begins with leading zeros, you can enter the value without the leading zeros. For example, if you enter 1040, ZIPSTATE assumes that the value is 01040.

Details

The Basics
If the ZIPSTATE function returns a value to a variable that has not yet been assigned a length, by default the variable is assigned a length of 20.

ZIPSTATE returns the two-character state postal code (or worldwide GSA geographic code for U.S. territories) that corresponds to its five-digit ZIP code argument. ZIPSTATE returns character values in uppercase.

Note: ZIPSTATE does not validate the ZIP code.

How the ZIP Code Is Translated to the State Postal Code
To determine which state corresponds to a particular ZIP code, this function uses a zone table that consists of the start and end ZIP code values for each state. It then finds the corresponding state for that range of ZIP codes. The zone table consists of start and end ZIP code values for each state to allow for exceptions, and does not validate ZIP code values.

With very few exceptions, a zone does not span multiple states. The exceptions are included in the zone table. It is possible for new zones or new exceptions to be added by the U.S. Postal Service at any time. However, SAS software is updated only with each new release of the product.

Army Post Office (APO) and Fleet Post Office (FPO) Postal Codes
The ZIPSTATE function recognizes APO and FPO ZIP codes. These military ZIP codes correspond to their exit bases in the United States. The ZIP codes are contained in the Sashelp.Zipmil data set. To determine when this data set was last updated, execute PROC CONTENTS:

```sas
proc contents data=sashelp.zipmil;
run;
```

Output from the CONTENTS procedure provides the date of the last update, along with the contents of the Sashelp.Zipmil data set.

Note: You can download the latest version of the Sashelp.Zipmil data set from the Technical Support Web site. Select Zipcode Data Set from the Name column to begin the download process. You must execute the CIMPORT procedure after you download and unzip the data set.

Determining When the State Postal Code Table Was Last Updated
Except for APO and FPO addresses, the Sashelp.Zipcode data set contains postal code information that is used with ZIPSTATE and other ZIP code functions. To determine when this data set was last updated, execute PROC CONTENTS:

```sas
proc contents data=sashelp.zipcode;
run;
```
Output from the CONTENTS procedure provides the date of the last update, along with the contents of the Sashelp.Zipcode data set.

Note: You can download the latest version of the Sashelp.Zipcode data set from the Technical Support Web site. Select Zipcode Data Set from the Name column to begin the download process. You must execute the CIMPORT procedure after you download and unzip the data set.

Comparisons

The ZIPCITY, ZIPNAME, ZIPNAMEL, and ZIPSTATE functions accept the same argument but return different values:

- ZIPCITY returns the mixed-case name of the city and the two-character postal code that corresponds to its five-digit ZIP code argument.
- ZIPNAME returns the uppercase name of the state or U.S. territory that corresponds to its five-digit ZIP code argument.
- ZIPNAMEL returns the mixed-case name of the state or U.S. territory that corresponds to its five-digit ZIP code argument.
- ZIPSTATE returns the uppercase two-character state postal code (or worldwide GSA geographic code for U.S. territories) that corresponds to its five-digit ZIP code argument.

Example

The following SAS statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>state1=zipstate('27511');</td>
<td></td>
</tr>
<tr>
<td>put state1;</td>
<td>NC</td>
</tr>
<tr>
<td>state2=zipstate('01040');</td>
<td></td>
</tr>
<tr>
<td>put state2;</td>
<td>MA</td>
</tr>
<tr>
<td>state3=zipstate(1040);</td>
<td></td>
</tr>
<tr>
<td>put state3;</td>
<td>NA</td>
</tr>
<tr>
<td>state4=zipstate(59017);</td>
<td></td>
</tr>
<tr>
<td>put state4;</td>
<td>MT</td>
</tr>
<tr>
<td>length zip $10.;</td>
<td></td>
</tr>
<tr>
<td>zip='90049-1392';</td>
<td></td>
</tr>
<tr>
<td>zip=substr(zip, 1, 5);</td>
<td></td>
</tr>
<tr>
<td>state5=zipstate(zip);</td>
<td></td>
</tr>
<tr>
<td>put state5;</td>
<td>CA</td>
</tr>
</tbody>
</table>

See Also

Functions:
- “ZIPCITY Function” on page 1098
- “ZIPFIPS Function” on page 1101
• “ZIPNAME Function” on page 1103
• “ZIPNAMEL Function” on page 1105
Chapter 4

References


Amos, D. E., S. L. Daniel, and K. Weston. 1977. “CDC 6600 Subroutines IBESS and JBESS for Bessel Functions I(ν,x) and J(ν,x), x ≥ 0, ν ≥ 0.” ACM Transactions on Mathematical Software 3: 76–95.


Appendix 1
Tables of Perl Regular Expression (PRX) Metacharacters

General Constructs

Table A1.1  General Constructs

<table>
<thead>
<tr>
<th>Metacharacter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( )</td>
<td>indicates grouping.</td>
</tr>
<tr>
<td>non-metacharacter</td>
<td>matches a character.</td>
</tr>
<tr>
<td>{ ] [ ) ^ $ . * + ? \</td>
<td>to match these characters, override (escape) with .</td>
</tr>
<tr>
<td>\</td>
<td>overrides the next metacharacter.</td>
</tr>
<tr>
<td>\n</td>
<td>matches capture buffer n.</td>
</tr>
<tr>
<td>(?:...)</td>
<td>specifies a non-capturing group.</td>
</tr>
</tbody>
</table>

Basic Perl Metacharacters

The following table lists the metacharacters that you can use to match patterns in Perl regular expressions.

Table A1.2  Basic Perl Metacharacters and Their Descriptions

<table>
<thead>
<tr>
<th>Metacharacter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\a</td>
<td>matches an alarm (bell) character.</td>
</tr>
<tr>
<td>\A</td>
<td>matches a character only at the beginning of a string.</td>
</tr>
</tbody>
</table>
| \b            | matches a word boundary (the position between a word and a space):
|               | • "er\b" matches the "er" in "never"
<p>|               | • &quot;er\b&quot; does not match the &quot;er&quot; in &quot;verb&quot;        |</p>
<table>
<thead>
<tr>
<th>Metacharacter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\B</td>
<td>matches a non-word boundary:</td>
</tr>
<tr>
<td></td>
<td>• &quot;er\B&quot; matches the &quot;er&quot; in &quot;verb&quot;</td>
</tr>
<tr>
<td></td>
<td>• &quot;er\B&quot; does not match the &quot;er&quot; in &quot;never&quot;</td>
</tr>
<tr>
<td>\cA-\cZ</td>
<td>matches a control character. For example, \cX matches the control character control-X.</td>
</tr>
<tr>
<td>\C</td>
<td>matches a single byte.</td>
</tr>
<tr>
<td>\d</td>
<td>matches a digit character that is equivalent to [0−9].</td>
</tr>
<tr>
<td>\D</td>
<td>matches a non-digit character that is equivalent to [^0−9].</td>
</tr>
<tr>
<td>\e</td>
<td>matches an escape character.</td>
</tr>
<tr>
<td>\E</td>
<td>specifies the end of case modification.</td>
</tr>
<tr>
<td>\f</td>
<td>matches a form feed character.</td>
</tr>
<tr>
<td>\l</td>
<td>specifies that the next character is lowercase.</td>
</tr>
<tr>
<td>\L</td>
<td>specifies that the next string of characters, up to the \E metacharacter, is lowercase.</td>
</tr>
<tr>
<td>\n</td>
<td>matches a newline character.</td>
</tr>
<tr>
<td>\num \Snum</td>
<td>matches capture buffer num, where num is a positive integer. Perl variable syntax ($num) is valid when referring to capture buffers, but not in other cases.</td>
</tr>
<tr>
<td>\Q</td>
<td>escapes (places a backslash before) all non-word characters.</td>
</tr>
<tr>
<td>\r</td>
<td>matches a return character.</td>
</tr>
<tr>
<td>\s</td>
<td>matches any whitespace character, including space, tab, form feed, and so on, and is equivalent to [\f\n\r\t\v].</td>
</tr>
<tr>
<td>\S</td>
<td>matches any character that is not a whitespace character and is equivalent to [^\f\n\r\t\v].</td>
</tr>
<tr>
<td>\t</td>
<td>matches a tab character.</td>
</tr>
<tr>
<td>\u</td>
<td>specifies that the next character is uppercase.</td>
</tr>
<tr>
<td>\U</td>
<td>specifies that the next string of characters, up to the \E metacharacter, is uppercase.</td>
</tr>
<tr>
<td>\w</td>
<td>vertical white space.</td>
</tr>
</tbody>
</table>
Metacharacter and Replacement Strings

You can use the following metacharacters in both a regular expression and in replacement text, when you use a substitution regular expression:

- \l
- \u
- \L
- \E
- \U
- \Q

These metacharacters are useful in replacement text for controlling the case of capture buffers that are used within replacement text. For an example of how these metacharacters can be used, see “Replacing Text” on page 45

For a description of these metacharacters, see Table A1.2 on page 1113.

Other Quantifiers

The following table lists other qualifiers that you can use in Perl regular expressions. The descriptions of the metacharacters in the table include examples of how the metacharacters can be used.
Table A1.3  Other Quantifiers

<table>
<thead>
<tr>
<th>Metacharacter</th>
<th>Description</th>
</tr>
</thead>
</table>
| \ | marks the next character as either a special character, a literal, a back reference, or an octal escape:  
  • “\n” matches a newline character  
  • “\" matches “\"  
  • “\\” matches \"  
| | specifies the or condition when you compare alphanumeric strings. For example, the construct x/y matches either x or y:  
  • “z|food” matches either “z” or “food”  
  • “(z|f)ood” matches “zood” or “food” |
| ^ | matches the position at the beginning of the input string. |
| $ | matches the position at the end of the input string. |
| period (.) | matches any single character except newline. To match any character including newline, use a pattern such as “[.:n]”. |
| (pattern) | specifies grouping. Matches a pattern and creates a capture buffer for the match. To retrieve the position and length of the match that is captured, use CALL PRXPOSN. To retrieve the value of the capture buffer, use the PRXPOSN function. To match parentheses characters, use “\(“ or “\)”. |

Greedy and Lazy Repetition Factors

Perl regular expressions support “greedy” repetition factors and “lazy” repetition factors. A repetition factor is considered greedy when the repetition factor matches a string as many times as it can when using a specific starting location. A repetition factor is considered lazy when it matches a string the minimum number of times that is needed to satisfy the match. To designate a repetition factor as lazy, add a ? to the end of the repetition factor. By default, repetition factors are considered greedy.

The following table lists the greedy repetition factors. The descriptions of the repetition factors in the table include examples of how they can be used.

Table A1.4  Greedy Repetition Factors

<table>
<thead>
<tr>
<th>Metacharacter</th>
<th>Description</th>
</tr>
</thead>
</table>
| * | matches the preceding subexpression zero or more times:  
  • zo* matches “z” and “zoo”  
  • * is equivalent to {0,} |
Metacharacter | Description
---|---
+ | matches the preceding subexpression one or more times:
  - "zo+" matches "zo" and "zoo"
  - "zo+" does not match "z"
  - + is equivalent to \{1,\}
?
| matches the preceding subexpression zero or one time:
  - "do(es)?" matches the "do" in "do" or "does"
  - ? is equivalent to \{0,1\}
{n} | matches at least \(n\) times.
{n,} | matches a pattern at least \(n\) times.
{n,m} | \(m\) and \(n\) are nonnegative integers, where \(n\leq m\). They match at least \(n\) and at most \(m\) times:
  - "o\{1,3\}" matches the first three o's in "fooooood"
  - "o\{0,1\}" is equivalent to "o?"
You cannot put a space between the comma and the numbers.

The following table lists the lazy repetition metacharacters.

**Table A1.5  Lazy Repetition Factors**

| Metacharacter | Description |
---|---|
*? | matches a pattern zero or more times. |
+? | matches a pattern one or more times. |
?? | matches a pattern zero or one time. |
{n}? | matches exactly \(n\) times. |
{n,}? | matches a pattern at least \(n\) times. |
{n,m}? | matches a pattern at least \(n\) times but not more than \(m\) times. |

**Class Groupings**

The following table lists character class groupings. You specify these classes by enclosing characters inside brackets. These metacharacters share a set of common properties. To be successful, the character class must always match a character. The negated character class must always match a character that is not in the list of characters that are designated inside the brackets. The descriptions of the metacharacters in the table include examples of how the metacharacters can be used.
<table>
<thead>
<tr>
<th>Metacharacter</th>
<th>Description</th>
</tr>
</thead>
</table>
| [...]        | specifies a character set that matches any one of the enclosed characters:  
  • “[abc]” matches the “a” in “plain” |
| [^...]       | specifies a negative character set that matches any character that is not enclosed:  
  • “[^abc]” matches the “p” in “plain” |
| [a-z]        | specifies a range of characters that matches any character in the range:  
  • “[a-z]” matches any lowercase alphabetic character in the range “a” through “z” |
| [^a-z]       | specifies a range of characters that does not match any character in the range:  
  • “[^a-z]” matches any character that is not in the range “a” through “z” |
| [:alpha:]    | matches an alphabetic character. |
| [:^alpha:]   | matches a nonalphabetic character. |
| [:alnum:]    | matches an alphanumeric character. |
| [:^alnum:]   | matches a non-alphanumeric character. |
| [:ascii:]    | matches an ASCII character. Equivalent to [\0–\177]. |
| [:^ascii:]   | matches a non-ASCII character. Equivalent to [^\0–\177]. |
| [:blank:]    | matches a blank character. |
| [:^blank:]   | matches a non-blank character. |
| [:ctrl:]     | matches a control character. |
| [:^ctrl:]    | matches a character that is not a control character. |
| [:digit:]    | matches a digit. Equivalent to \d. |
| [:^digit:]   | matches a non-digit character. Equivalent to \D. |
| [:graph:]    | is a visible character, excluding the space character. Equivalent to [:alnum:][:punct:]. |
| [:^graph:]   | is not a visible character. Equivalent to [^[:alnum:][:punct:]]. |
| [:lower:]    | matches lowercase characters. |
| [:^lower:]   | does not match lowercase characters. |
| [:print:]    | prints a string of characters. |
| [:^print:]   | does not print a string of characters. |
### Tables of Perl Regular Expression (PRX) Metacharacters

<table>
<thead>
<tr>
<th>Metacharacter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[:punct:]</td>
<td>matches a punctuation character or a visible character that is not a space or alphanumeric.</td>
</tr>
<tr>
<td>[:^punct:]</td>
<td>does not match a punctuation character or a visible character that is not a space or alphanumeric.</td>
</tr>
<tr>
<td>[:space:]</td>
<td>matches a space. Equivalent to \s.</td>
</tr>
<tr>
<td>[:^space:]</td>
<td>does not match a space. Equivalent to \S.</td>
</tr>
<tr>
<td>[:upper:]</td>
<td>matches uppercase characters.</td>
</tr>
<tr>
<td>[:^upper:]</td>
<td>does not match uppercase characters.</td>
</tr>
<tr>
<td>[:word:]</td>
<td>matches a word. Equivalent to \w.</td>
</tr>
<tr>
<td>[:^word:]</td>
<td>does not match a word. Equivalent to \W.</td>
</tr>
<tr>
<td>[:xdigit:]</td>
<td>matches a hexadecimal character.</td>
</tr>
<tr>
<td>[:^xdigit:]</td>
<td>does not match a hexadecimal character.</td>
</tr>
</tbody>
</table>

### Look-Ahead and Look-Behind Behavior

Look-ahead and look-behind are ways to look ahead or behind a match to see whether a particular text occurs. The text that is found with look-ahead or look-behind is not included in the match that is found. For example, if you want to find names that end with “Jr.”, but you do not want “Jr.” to be part of the match, you could use the regular expression `/.*(?=Jr\./)`. For the value "John Wainright Jr.", the regular expression finds "John Wainright" as a match because it is followed by "Jr."

#### Table A1.7  Look-Ahead and Look-Behind Behavior

<table>
<thead>
<tr>
<th>Metacharacter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(?=...)</td>
<td>specifies a zero-width, positive, look-ahead assertion. For example, in the expression <code>regex1 (?=regex2)</code>, a match is found if both <code>regex1</code> and <code>regex2</code> match. <code>regex2</code> is not included in the final match.</td>
</tr>
<tr>
<td>(!=...)</td>
<td>specifies a zero-width, negative, look-ahead assertion. For example, in the expression <code>regex1 (?!regex2)</code>, a match is found if <code>regex1</code> matches and <code>regex2</code> does not match. <code>regex2</code> is not included in the final match.</td>
</tr>
<tr>
<td>(&lt;=...)</td>
<td>specifies a zero-width, positive, look-behind assertion. For example, in the expression <code>(?&lt;=regex1) regex2</code>, a match is found if both <code>regex1</code> and <code>regex2</code> match. <code>regex1</code> is not included in the final match. Works with fixed-width look-behind only.</td>
</tr>
<tr>
<td>(!&lt;...)</td>
<td>specifies a zero-width, negative, look-behind assertion. Works with fixed-width look-behind only.</td>
</tr>
</tbody>
</table>
Comments and Inline Modifiers

The metacharacters in this table contain a question mark as the first element inside the parentheses. The characters after the question mark indicate the extension.

Table A1.8 Comments and Inline Modifiers

<table>
<thead>
<tr>
<th>Metacharacter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(?,text)</td>
<td>specifies a comment in which the text is ignored.</td>
</tr>
<tr>
<td>(?imsx)</td>
<td>specifies one or more embedded pattern-matching modifiers. If the pattern is case insensitive, you can use (?i) at the front of the pattern. An example is $pattern=&quot;(?i)foo\bar&quot;; Letters that appear after a hyphen (-) turn the modifiers off.</td>
</tr>
</tbody>
</table>

Selecting the Best Condition By Using Combining Operators

The elementary regular expressions (for example, \a and \w) that are described in the preceding tables can match at most one substring at the given position in the input string. However, operators that perform combining in typical regular expressions combine elementary metacharacters to create more complex patterns. In an ambiguous situation, these operators can determine the best match or the worst match. The match that is the best is always chosen.

Table A1.9 Best Match Using Combining Operators

<table>
<thead>
<tr>
<th>Metacharacter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST</td>
<td>in the following example, specifies that AB and A'B', and A and A' are substrings that can be matched by S, and that B and B' are substrings that can be matched by T:</td>
</tr>
<tr>
<td></td>
<td>• If A is a better match for S than A', then AB is a better match than A'B'.</td>
</tr>
<tr>
<td></td>
<td>• If A and A' coincide, then AB is a better match than AB' if B is a better match for T than B'.</td>
</tr>
<tr>
<td>S</td>
<td>T</td>
</tr>
<tr>
<td>S{repeat-count}</td>
<td>matches as SSS . . . S (repeated as many times as necessary).</td>
</tr>
<tr>
<td>S{min,max}</td>
<td>matches as S{max}</td>
</tr>
<tr>
<td>S{min,max}?</td>
<td>matches as S{min}</td>
</tr>
<tr>
<td>S?, S*, S+</td>
<td>same as S{0,1}, S{0, big-number}, S{1,big-number}, respectively.</td>
</tr>
<tr>
<td>Metacharacter</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>S??, S*?, S+</td>
<td>same as S{0,1}? , S{0, big-number}? , S{1,big-number}? , respectively.</td>
</tr>
<tr>
<td>(?=S), (?&lt;=S)</td>
<td>considers the best match for S. (This is important only if S has capturing parentheses, and back references are used elsewhere in the whole regular expression.)</td>
</tr>
<tr>
<td>(?!S), (?)&lt;S)</td>
<td>unnecessary to describe the ordering for this grouping operator because only whether S can match is important.</td>
</tr>
</tbody>
</table>
Here is the recommended reading list for this title:

- Base SAS Glossary
- SAS DS2 Language Reference
- SAS Scalable Performance Data Engine: Reference
- SAS FedSQL Language: Reference
- Batch and Line Mode Processing in SAS Viya
- SAS Viya Data Set Options: Reference
- SAS Viya Formats and Informats: Reference
- SAS Viya Statements: Reference
- SAS Viya System Options: Reference
- SAS Language Reference: Concepts
- SAS Metadata LIBNAME Engine: User’s Guide
- SAS Viya Utility Procedures Guide

The recommended reading list from SAS Press includes the following titles:

- An Array of Challenges - Test Your SAS Skills
- Cody’s Data Cleaning Techniques Using SAS
- Combining and Modifying SAS Data Sets: Examples
- SAS Functions by Example
- SAS Guide to Report Writing: Examples
- The Little SAS Book: A Primer
- Output Delivery System: The Basics and Beyond
- The SAS Workbook
- Learning SAS by Example: A Programmer’s Guide
Recommended Reading

For a complete list of SAS publications, go to sas.com/store/books. If you have questions about which titles you need, please contact a SAS Representative:

SAS Books
SAS Campus Drive
Cary, NC 27513-2414
Phone: 1-800-727-0025
Fax: 1-919-677-4444
Email: sasbook@sas.com
Web address: sas.com/store/books
### Special Characters
- _IORC_ variable
  - formatted error messages for 975
- %SYSCALL macro
  - CALL GRAYCODE routine with 749
- %SYSFUNC macro
  - generating random number streams with function calls 14

### Numbers
- 32-bit platforms
  - memory address of character variables 715
  - memory address of numeric variables 714
- 64-bit platforms
  - memory address of character variables 715

### A
- ABS function 79
  - absolute value 79
  - sum of, for non-missing arguments 623
  - accrued interest
    - securities paying interest at maturity 248, 271
    - securities paying periodic interest 247, 271
- ADDR function 714
- ADDRLONG function 715
- AIRY function 79
- ALLCOMB function 716
- ALLPERM function 718
  - alphabetic characters
    - searching character string for 82
  - alphanumeric characters
    - searching character string for 80
  - annuities
    - interest rate per period 265, 282
    - periodic payment 262, 280
- ANYALNUM function 80
- ANYALPHA function 82
  - ANYCNTRL function 84
  - ANYDIGIT function 86
  - ANYFIRST function 87
  - ANYGRAPH function 89
  - ANYLOWER function 91
  - ANYNAME function 93
  - ANYPRINT function 95
  - ANYPUNCT function 97
  - ANYSPACE function 99
  - ANYUPPER function 101
  - ANYXDIGIT function 102
  - arc tangent 108
    - of two numeric variables 109
  - arccosine 104
  - ARCOS function 104
  - ARCOSH function 105
  - arcsine 106
    - arguments 231, 745
      - converting to lowercase 423
      - converting words to proper case 1020
      - counting missing arguments 855
      - difference between nthlag 231
      - extracting substrings 617
      - format decimal values, returning 653
      - format names, returning 655
      - format width, returning 657
      - informat decimal values, returning 664
      - informat names, returning 666
      - informat width, returning 668
      - resolving 745
      - returning length of 406
      - searching for character values, equal to first argument 688
      - searching for numeric values, equal to first argument 689
      - size, returning 674
      - arithmetic mean 433
      - arrays 234
        - finding contents 660
        - finding dimensions 234
        - finding values in 648
        - identifying 647
        - lower bounds 402
upper bounds of
ARSIN function 106
ARSINH function 106
ARTANH function 107
ASCII characters, returning 123, 179
by number 123
number of 568
asymmetric spelling differences 1059
ATAN function 108
ATAN2 function 109
ATTRC function 720
ATTRN function 723
average 433

B
BAND function 110
base interval
shift interval corresponding to 382
Bernoulli distributions 154, 227
cumulative distribution functions 154
probability density functions 479
bessel function, returning value of 326, 390
beta distribution
returning a quantile from 112
beta distributions
cumulative distribution functions 155
probabilities from 501
probability density functions 479
BETA function 111
BETAINV function 112
binomial distributions 156, 228, 794
cumulative distribution functions 156
probabilities from 502
probability density functions 480
random numbers 794, 1030
bitwise logical operations
AND 110
EXCLUSIVE OR 123
NOT 121
OR 121
shift left 120
shift right 122
bivariate normal distribution
probability computed from 504
Black model
call prices for European options on
futures 113
put prices for European options on
futures 115
Black-Scholes model
call prices for European options on
stocks 116
BLACKCLPRC function 113
BLACKPTPRC function 115
blanks 187
compressing 187, 194
removing from search string 634
searching character string for 99
trimming trailing 640, 642
BLKSHCLPRC function 116
BLKSHPTPRC function 118
BLSHIFT function 120
BNOT function 121
bond-equivalent yield 267, 283
bookmarks 924
finding 1019
setting 924
BOR function 121
BRSHIFT function 122
BXOR function 123
BYTE function 123

C
CALL ALLCOMB routine 727
in DATA step 729
with macros 728, 729
CALL ALLCOMBI routine 730
in DATA step 732
with macros 731, 732
CALL ALLPERM routine 733
CALL CATS routine 736
CALL CATT routine 738
CALL CATX routine 740
CALL COMPCOST routine 742
CALL EXECUTE routine 745
CALL GRAYCODE routine 746
%SYSCALL macro with 749
in DATA step 747
with macros 747
CALL IS8601_CONVERT routine 750
CALL LABEL routine 124
CALL LEXCOMB routine 760
in DATA step 762
with macros 762, 763
CALL LEXCOMBI routine 764
with DATA step 765
with macros 764, 766
CALL LEXPERK routine 766
in DATA step 768
with macros 768, 770
CALL LEXPERM routine 770
in DATA step 773
with macros 772, 773
CALL LOGISTIC routine 774
CALL MISSING routine 126
comparison 126
details 126
CALL MODULE routine 775
arguments 775
comparisons 778
details 777
examples 779
MODULEIN function and 779
MODULEN function and 779
CALL POKE routine 780
CALL POKELONG routine 781
call prices
European options on futures, Black model 113
European options on stocks, Black-Scholes model 116
for European options, based on Margrabe model 425
CALL PRXCHANGE routine 782
CALL PRXDEBUG routine 785
CALL PRXFREE routine 127
CALL PRXNEXT routine 787
CALL PRXPOSN routine 789
CALL PRXSUBSTR routine 792
CALL RANBIN routine 794
CALL RANCAU routine 797
CALL RANCOMB routine 827
CALL RANEXP routine 800
CALL RANGAM routine 802
CALL RANNOR routine 805
CALL RANPERK routine 829
CALL RANPERM routine 831
CALL RANPOI routine 807
CALL RANTBL routine 809
CALL RANTRI routine 812
CALL RANUNI routine 814
CALL routines 2
Perl regular expression (PRX) CALL routines 43
random-number CALL routines 11
syntax 4
CALL SCAN routine 128, 816
CALL SET routine 833
CALL SLEEP routine 138
CALL SOFTMAX routine 835
CALL SORTC routine 836
CALL SORTN routine 837
CALL STREAMINIT routine 139
CALL SYMPUT routine 841
CALL SYMPUTX routine 842
CALL SYSTEM routine 844
CALL TANH routine 845
CALL VNAME routine 141
capture buffers 541
carriage returns
searching character string for 99
case
converting argument words to proper case 1020
cashflow stream, periodic
convexity for 200
modified duration for 238
present value for 547
cashflow, enumerated
convexity for 199
modified duration for 237
CAT function 143
catalogs
renaming entries 1040
CATQ function 846
CATS function 145
CATT function 147
CATX function 149
Cauchy distributions 156, 797
  cumulative distribution functions 156
  probability density functions 481
  random numbers 797, 1031
CDF 153
CDF function 153
CEIL function 171
ceiling values 171
CEILZ function 172
CEXIST function 850
CHAR function 851
character arguments
converting words to proper case 1020
returning value of 178
character attributes
returning the value of 720
character expressions 327
converting to uppercase 644
encoding for searching 1058
first unique character 649
left aligning 405
missing values, returning a result for 436
repeating 569
replacing characters in 632
replacing words in 635
reversing 570
right aligning 571
searching by index 327
searching for specific characters 329
searching for words 330
selecting a word from 586
character string, validity 1001
character strings
character position of a word in 295
compressing specified characters 191
counting words in 209
first character in 921
number of a word in 295
returning single character from specified position 851
searching 295
searching for a character in a variable name 93
searching for alphabetic characters in 82
searching for alphanumeric characters in 80
searching for control characters in 84
searching for digits in 86
searching for first character in a variable name 87
searching for graphical characters in 89
searching for hexadecimal character in 102
searching for lowercase letter in 91
searching for printable character in 95
searching for punctuation character in 97
searching for uppercase letter in 101
searching for white-space character in 99
character values
based on true, false, or missing expressions 971
choice from a list of arguments 173
replacing contents of 615
searching for, equal to first argument 688
character variables
memory address of 715
sorting argument values 836
chi-squared distributions 157, 176, 853
cumulative distribution functions 157
probabilities 505
probability density functions 481
CHOOSEC function 173
CHOOSEN function 175
CINV function 853
quantiles 853
CLOSE function 854
CMISS function 855
CNONCT function 176
COALESCE function 177
COALESCEC function 178
coefficient of variation 216
COLLATE function 179
COMB function 183
logarithm of 404
combinatorial CALL routines
all combinations 727
distinct non-missing, in lexicographic order 760, 766
indices 730
indices, in lexicographic order 764
subsetting 746
combinatorial functions
all combinations 716
all permutations 718
distinct non-missing, in lexicographic order 981
indices, in lexicographic order 979
non-missing distinct, in lexicographic order 977
non-missing values, in lexicographic order 984
subsetting 946
combinatorial routines
non-missing values, in lexicographic order 770
COMPARE function 184
COMPBL function 187
COMPFUZZ function 188
COMPGED function 856
complementary error function 241
COMPLEV function 862
COMPOUND function 190
compound interest 190
COMPRESS function 187, 191
arguments 191
compared to COMPBL function 187
compressing blanks 194
compressing lowercase letters 194
compressing tab characters 194
details 193
examples 194
keeping characters in the list 194
compressing 187
blanks 187
compressing character strings 191
blanks 194
keeping characters in the list 194
lowercase letters 194
tab characters 194
concatenation
with delimiter and quotation marks 846
confidence intervals, computing 521
CONSTANT function 195
constants, calculating
double-precision numbers, largest 197
double-precision numbers, smallest 198
Euler constant 196
exact integer 197
machine precision 198
natural base 195
overview 195
control characters
searching character string for 84
converting ISO 8601 intervals 750
convexity, for enumerated cashflow 199
convexity, for periodic cashflow stream 200
CONVX function 199
CONVXP function 200
Conway-Maxwell-Poisson distribution 482
corrected sum of squares 213
COS function 201
cosecant 212
COSH function 202
cosine 201
inverse hyperbolic 105
COT function 203
cotangent 203
COUNT function 204
COUNTC function 206
counting
missing arguments 855
words in a character string 209
COUNTW function 209
coupon period
coupons payable between settlement and maturity dates 251, 273
days from beginning to settlement date 249, 272
days from settlement date to next coupon date 250, 272
next coupon date after settlement date 251, 273
number of days 250, 272
pervious coupon date before settlement date 251, 273
CSC function 212
CSS function 213
CUMIPMT function 214
CUMPRINC function 215
cumulative distribution functions 153
Bernoulli distribution 154
beta distribution 155
binomial distribution 156
Cauchy distribution 156
chi-squared distribution 157
exponential distribution 158
F distribution 158
gamma distribution 159
generalized Poisson distributions 160, 486
geometric distribution 160
hypergeometric distribution 161
Laplace distribution 162
logistic distribution 162
lognormal distribution 163
negative binomial distribution 163
normal distribution 164
Pareto distribution 165
Poisson distribution 166
T distribution 166
tweedie distribution 167
uniform distribution 168
Wald (Inverse Gaussian) distribution 168
Weibull distribution 169
cumulative interest 252, 274
in CUMIPMT function 214
cumulative principal 252, 274
CUROBS function 864
custom time intervals 34
reasons for using 34
CV function 216
cycle index 341

D
DACCDB function 865
DACCDBSL function 866
DACCSCL function 867
DACCCSYD function 868
DACCTAB function 869
DAIRY function 216
data libraries
verifying existence of members 892
Data Set Data Vector (DDV), reading
observations into 903, 905
data set names, returning 889
data set pointer, positioning at start of data set 1043
data sets
character attributes, returning value of 720
numeric attributes, returning value of 723
renaming 1042
verifying existence of 893
DATA step 833, 841
assigning data to macro variables 841
CALL ALLCOMB routine in 729
CALL ALLCOMBI routine in 732
CALL GRAYCODE routine in 747
CALL LEXCOMB routine in 762
CALL LEXCOMBI routine with 765
CALL LEXPENK routine in 768
CALL LEXPERM routine in 773
generating random number streams with function calls 11
linking SAS data set variables 833, 841
Perl regular expressions (PRX) in 43
DATA step functions
within macro functions 8
data type, returning 678
data validation 45
data views
verifying existence of 893
DATDIF function 217
date and time intervals 31
commonly used time intervals 32
definition 31
incrementing dates and times 31
interval names and SAS dates 31
date calculations
  years between dates 692
DATE function 220
date intervals
  cycle index 341
  recommended format for 356
  seasonal cycle 350, 378
  seasonal index 361
date values
  aligning output 373
  holidays 323
  incrementing 370
date/time functions 390
  date values, returning 432
  dates, extracting from datetime value 221
  dates, returning current 220, 222
  datetime value, creating 230
  day of the month, returning 223
  day of week, returning 687
  hour value, extracting 325
  Julian dates, converting to SAS values 221
  minute values, returning 435
  month values, returning 442
  seconds value, returning 600
  time intervals, extracting integer values of 344
  time values, creating 322
  time, extracting from datetime values 626
  time, returning current 222
  year quarter, returning 548
  year quarter, returning date value from 695
  year value, returning 690
DATEJUL function 221
DATEPART function 221
dates
  time intervals aligned between two 353
  time intervals based on three values 359
  weekdays 473
dates, Julian 390
DATETIME function 222
datetime intervals
  cycle index 341
  recommended format for 356
  seasonal cycle 350, 378
  seasonal index 361
datetime values
  converting to/from ISO 8601 intervals 750
incrementing 370
time intervals based on three values 359
DAY function 223
DCLOSE function 870
DCREATE function 872
DDV (Data Set Data Vector), reading observations into 905
DDV (Data Set Data Vector), reading observations into 903
debugging
  writing Perl debug output to log 52
declining balance method 268, 284
degrees
  geodetic distance input in 314
delimiters
  concatenation and 846
DEPDB function 873
DEPDBSL function 874
depreciation 865
  accumulated declining balance 865, 866
  accumulated from tables 869
  accumulated straight-line 867
  accumulated straight-line, converting from declining balance 866
  accumulated sum-of-years 868
  declining balance 873
  declining balance method 268, 284
  depreciation coefficient 248, 271
  double-declining balance method 253, 274
  fixed-declining balance method 253, 274
  for each accounting period 249, 272
  from tables 876
  straight-line 266, 282, 867, 875
  straight-line, converting from declining balance 874
  sum-of-years digits 267, 283
  sum-of-years-digits 876
DEPSL function 875
DEPSYD function 876
DEPTAB function 876
DEQUOTE function 224
descriptive statistic functions 6
DEVIANCE function 226
deviance, computing
  Bernoulli distribution 227
  binomial distribution 228
  Gamma distribution 228
  inverse Gaussian (Wald) distribution 229
  normal distribution 229
  overview 226
  Poisson distribution 229
DHMS function 230
DIF function 231
difference between nthlag 231
DIGAMMA function 233
digits
  searching character string for 86
DIM function 234, 321
  compared to HBOUND function 321
DINFO function 877
directories 870
  assigning/deassigning filerefs 910
  closing 870, 896
  creating 872
  opening 881
  renaming 1040
directories, returning
  attribute information 882
  information about 877
  number of information items 884
  number of members in 880
directory members 887
  closing 896
  name of, returning 887
discount rate 254, 275
DIVIDE function 235
division
  ODS missing values and 235
DLLs
calling 993
  MODULE functions 993
DNUM function 880
dollar price
  converting from decimal number to fraction 255, 275
  converting from fraction to decimal number 254, 275
DOPEN function 881
DOPTNAME function 882
DOPTNUM function 884
DOSUBL function 885
double-declining balance method 253, 274
double-precision number constants 197, 198
DREAD function 887
DROPNOTE function 888
DSNAME function 889
Dunnett's one-sided test 514
Dunnett's two-sided test 515
DUR function 237
duration
  Macauley modified 258, 277
  securities with periodic interest payments 255, 275
duration values
  converting to/from ISO 8601 intervals 750
DURP function 238

E

EBCDIC characters 123
  getting by number 123
  returning a string of 179
  returning numeric value of 568
effective annual interest rate 239, 255, 276
EFFRATE function 239
encoding strings 1058
enumerated cashflow
  convexity for 199
  modified duration for 237
environment variables
  length of 890
ENVLEN function 890
ERF function 240
ERFC function 241
error function 240
error function, complementary 241
error messages 1074
  for _IORC_ variable 975
  returning 1074
EUCLID function 891
Euclidean norm
  calculating with variable list 892
  of non-missing arguments 891
Euler constants 196
European options on futures
  call prices, based on Black model 113
  put prices, based on Black model 115
European options on stocks
  call prices, based on Black-Scholes model 116
  call prices, based on Margrabe model 425
  put prices based on Margrabe model 427
exact integer constants 197
EXECUTE CALL routine 745
EXIST function 892
existence of software image 992
EXP function 242
exponential distribution 800
exponential distributions 158
  cumulative distribution functions 158
  probability density functions 484
  random numbers 800, 1032
exponential functions 242
expressions
  character values based on 971
  numeric values based on 973
external files 888
appending records to 895
assigning filerefs 912
closing 896
deassigning filerefs 910
deleting 902
getting information about 931
names of information items 928
note markers, returning 888
number of information items 931
opening 925
opening by directory id 996
opening by member name 996
pathnames, returning 1013
pointer to next record 932
reading 937
renaming 1040
size of current record 940
size of last record read 940
verifying existence 906, 909
writing 942
external files, reading 937
to File Data Buffer (FDB) 937
external routines
calling, without return code 775
extracting strings from substrings 48

F
F distributions 158
  cumulative distribution functions 158
  noncentrality parameter 303
  probabilities from 506
  probability density functions 484
  quantiles 916
FACT function 242
  logarithm of 411
false expressions 971, 973
FAPPEND function 895
FCLOSE function 896
FCOL function 897
FCOPY function 898
FDELETE function 902
FETCH function 903
FETCHOBS function 905
FEXIST function 906
FGET function 907
  setting token delimiters for 941
File Data Buffer (FDB) 897
  column pointer, setting 934
  copying data from 907
  current column position 897
  moving data to 936
  reading external files to 937
  file information items, value of 914
  file manipulation
functions for 9
file pointer, setting to start of file 938
FILEEXIST function 909
FILENAME function 910
  arguments 910
  details 912
  examples 912
  filerefs for external files 912
  filerefs for pipe files 913
  system-generated filerefs 912
FILEREF function 913
  filerefs
    assigning to directories 910
    assigning to external files 912
    assigning to output devices 910
    assigning to pipe files 913
    deassigning 910
    FILENAME function 910
    system-generated 912
    verifying 913
FINANCE function 243
  financial calculations 243
financial functions
  pricing functions 8
FIND function 285
FINDC function 288
FINDW function 295
FINFO function 914
  compared to FOPTNUM function 932
FINV function 916
FIPNAME function 917
FIPNAMEL function 918
FIPS codes
  converting to mixed case state names
  converting to postal codes 920
  converting to uppercase state names
  converting ZIP codes to 1101
FIPSTATE function 920
FIRST function 921
  fixed-declining balance method 253, 274
FLOOR function 301
  floor values 301
FLOORZ function 302
FMTINFO function 922
FNONCT function 303
FNOTE function 924
FOPEN function 925
  compared to FINO function 915
  compared to FOPTNUM function 932
FOPTNAME function 915, 928
  compared to FINFO function 915
  compared to FOPTNUM function 932
FOPTNUM function 915, 931
  compared to FINFO function 915
form feeds
  searching character string for 99
format decimal values, returning 652
  arguments 653
  variables 652
format names, returning 654
  arguments 655
  variables 654
format width, returning 654
  arguments 657
  variables 654, 656
formats applying 545
  character, specifying at run time 1025
  numeric, specifying at run time 1028
  recommended for date, time, or datetime intervals 356
  returning 651, 658, 1089
FPOINT function 932
FPOS function 934
FPUT function 936
FREAD function 937
FREWIND function 938
FRLEN function 940
FSEP function 941
functions 2
  CONSTANT 195
  DATA step functions within macro functions 8
  descriptive statistic functions 6
  file manipulation with 9
  for Web applications 54
  Perl regular expression (PRX) functions 43
  PERM 497
  pricing functions 8
  PROBMC 511
  random-number functions 11
  restrictions on arguments 4
  syntax 3
  target variables 5
YRDIF 692
future value
  of an investment 256, 276
  of initial principal 256, 276
future value of periodic savings 1044
futures
  call prices for European options on, Black model 113
  put prices for European options on, Black model 115
FUZZ function 305
FWRITE function 942
G
GAMINV function 306
gamma distributions 159, 228, 802
cumulative distribution functions 159
  probabilities from 507
  probability density functions 485
  quantiles 306
  random numbers 802, 1033
GAMMA function 306
  returning value of 306
GARKHCLPRC function 307
GARKHPTPRC function 309
Garman-Kohlhagen model
call prices for European options on stocks 307
GCD function 311
generation data sets
  renaming 1042
  verifying existence of 893
geodetic distance 312
  between two ZIP codes 1100
  in kilometers 313
  in miles 313
  input measured in degrees 314
  input measured in radians 314
GEODIST function 312
GEOMEAN function 315
GEOMEANZ function 316
geometric distributions 160
  cumulative distribution functions 160
  probability density functions 486
  geometric mean 315
  zero fuzzing 316
GETV ARC function 944
GETV ARN function 945
graphical characters
  searching character string for 89
GRAYCODE function 946
greatest common divisor 311
H
HARMEAN function 318
HARMEANZ function 319
harmonic mean 318
  zero fuzzing 319
HBOUND function 234, 320
  compared to DIM function 234
hexadecimal characters
  searching character string for 102
HMS function 322
HOLIDAY function 323
HOLIDAYCK function 949
HOLIDAYCOUNT function 952
HOLIDAYNAME function 955
HOLIDAYNX function 958
HOLIDAYNY function 960
holidays
  date value for 323
HOLIDAYTEST function 964
horizontal tabs
  searching character string for 99
HOUR function 325
HTML
decoding 967
encoding 968
HTMLDECODE function 967
HTMLENCODE function 968
hyperbolic cosine 202
  inverse 105
hyperbolic sine 606
  inverse 106
hyperbolic tangent 845
  inverse 107
hyperbolic tangents 1079
hypergeometric distributions 161
  cumulative distribution functions 161
  probabilities from 508
  probability density functions 487

I
IBESSEL function 326
IFC function 971
IFN function 973
IML procedure
  MODULEIN function in 779
incrementing values 370
INDEX function 327
  compared to INDEXC function 329
INDEXC function 329
indexes
  cycle index 341
  seasonal 361
INDEXW function 330
indices
  CALL ALLCOMBI routine and 730
  CALL LEXCOMBI routine and 764
  LEXCOMBI function and 979
informat decimal values, returning 663
  arguments 664
  variables 663
informat names, returning 665
  arguments 666
  variables 665
informat width, returning 667
  arguments 668
  variables 667
informats
  reading results of expressions 334
  returning 662, 669, 1091
  specifying at run time 334, 339
INPUT function 334
INPUT statement 335
  compared to INPUT function 335
INPUTC function 337
  compared to INPUTN function 339
INPUTN function 337, 339
  compared to INPUTC function 337
INT function 340
INTCIINDEX function 341
INTCK function 344
INTCYCLE function 350
integers
  greatest common divisor for 311
  interest
    accrued 247, 248
    cumulative 252
    payment for a given period 257, 277
  interest paid
    investment 277
  interest rate
    effective annual 255, 276
    fully invested securities 256, 276
    nominal 259, 278
    per period of an annuity 265, 282
  internal rate of return 257, 258, 268, 377
    as fraction 377
    as percentage 389
    examples 277, 278, 284
  interpolating spline
    monotonicity-preserving 999
  interval names 31
  INTFIT function 353
  INTFMT function 356
  INTGET function 359
  INTINDEX function 361
  INTNINDEX function 367
  INTNX 370
  INTNINDEX function 370
    aligning date output 373
    examples 375
  INTRR function 377
    compared to IRR function 389
  INTSEAS function 378
  INTSHIFT function 382
  INTTEST function 384
  INTZ function 385
  inverse Gaussian (Wald) distributions 229
  inverse hyperbolic cosine 105
  inverse hyperbolic sine 106
  inverse hyperbolic tangent 107
  IORCMSG function 975
  IPMT function 387
  IQR function 388
  IRR function 389
  ISO 8601 intervals
    converting 750
  ISPMT 258
J
JBESSEL function 390
JULDATE function 390
JULDATE7 function 391
julian date 391
Julian dates 390

K
kilometers
  geodetic distance in 313
kurtosis 392
KURTOSIS function 392

L
LAG function 393
Laplace distributions 162
  cumulative distribution functions 162
  probability density functions 488
LARGEST function 400
latitude
  geodetic distance between latitude and longitude coordinates 312
LBOUND function 402
LCM function 403
LCOMB function 404
least common multiple 403
LEFT function 405
length
  of environment variables 890
LENGTH function 406
  compared to VLENGTH function 673
LENGTHC function 407
LENGTHM function 408
LENGTHN function 410
LEXCOMB function 977
LEXCOMBI function 979
lexicographic order 760, 764, 766, 770, 977, 979, 981, 984
LEXPERK function 981
LEXPERM function 984
LFACT function 411
LGAMMA function 412
natural logarithm of 412
LIBNAME function 986
libraries
  renaming members 1040
LIBREF function 989
librefs 989
  assigning/deassigning 986
SAS libraries 989
  verifying 989
license verification 1078
licensing 992
line feeds
  searching character string for 99
log
  of 1 plus the argument 413
  writing Perl debug output to 52
LOG function 413
LOG10 function 414
LOG1PX function 413
LOG2 function 415
logarithms 412
  base 10 414
  base 2 415
  natural logarithms 413
  of COMB function 404
  of FACT function 411
  of LGAMMA function 412
  of PERM function 423
  of probability functions 418
  of survival functions 420
LOGBETA function 415
LOGCDF function 416
logistic distributions 162
  cumulative distribution functions 162
  probability density functions 488
LOGISTIC function 990
logistic values 774, 990
lognormal distributions 163
  cumulative distribution functions 163
  probability density functions 489
LOGPDF function 418
LOGSDF function 420
longitude
  geodetic distance between latitude and longitude coordinates 312
LOWCASE function 423
lowercase
  searching character string for 91
lowercase letters
  compressing 194
lowercase, converting arguments to 423
Lp norm 990
LPERM function 423
LPNORM function 990

M
Macauley modified duration 258, 277
machine precision constants 198
macro functions
  within DATA step functions 8
  macro variables 833, 841
  assigning DATA step data 841
  linking SAS data set variables 833, 841
  returning during DATA step 1070
  macros 1042
  CALL ALLCOMB routine with 728, 729
CALL ALLCOMBI routine with 731, 732
CALL GRAYCODE routine with 747
CALL LEXCOMB routine with 762, 763
CALL LEXCOMBI routine with 764, 766
CALL LEXPERK routine with 768, 770
CALL LEXPERM routine with 772, 773
returning values from 1042
MAD function 424
many-one t-statistics, Dunnett's one-sided test 514
many-one t-statistics, Dunnett's two-sided test 515
Margrabe model
call prices for European options on stocks 425
put prices for European options on stocks 427
MARGRCLPRC function 425
MARGRPTPRC function 427
matching words 1059
maturation
amount received at maturity 266, 282
MAX function 429
maximum values, returning 429
MD5 function 430
MDY function 432
MEAN function 433
means
multiple comparisons of 519
MEDIAN function 434
memory address
character variables 715
numeric variables 714
memory addresses, storing contents of 1014
as character variables 1016
as numeric variables 1014
message digest 430
metacharacters, PRX 1113
miles
geodetic distance in 313
MIN function 434
minimum values, returning 434
MINUTE function 435
missing arguments
counting 855
missing expressions 971, 973
MISSING function 436
missing values 446
assigning to specified variables 126
number of 446
ODS and 235
returning a value for 436
MOD function 438
MODEXIST function 992
MODULE functions 993
MODULEC function 995
MODULEIN function
CALL MODULE routine and 779
MODULEN function 995
CALL MODULE routine and 779
modulus 438
MODZ function 440
monotonicity-preserving interpolating spline 999
MONTH function 442
MOPEN function 996
MORT function 443
MSPLINT function 999
MV ALID function 1001

N
N function 444
natural logarithms 413
negative binomial distributions 163
cumulative distribution functions 163
probabilities from 524
probability density functions 489
net present value 260, 269, 445, 472
as fraction 445
as percentage 472
examples 279, 284
NETPV function 445
NLITERAL function 1004
NMISS function 446
nominal interest rate 259, 278
NOMRATE function 447
noncentrality parameters 176
chi-squared distribution 176
F distribution 303
student's t distribution 629
nonmissing values 444
normal distributions 164, 805
cumulative distribution functions 164
deviance from 229
probability density functions 490
random numbers 805, 1035
NORMAL function 1006
NOTALNUM function 448
NOTALPHA function 450
NOTCNTRL function 452
NOTDIGIT function 453
NOTE function 1006
NOTFIRST function 455
NOTGRAPH function 457
NOTLOWER function 459
NOTNAME function 461
NOTPRINT function 462
NOTPUNCT function 464
NOTSPACE function 466
NOTUPPER function 468
NOTXDIGIT function 470
NPV function 472
numeric arguments
  returning value of 177
numeric attributes
  returning the value of 723
numeric data 340
truncating 340, 643
numeric expressions
  missing values, returning a result for 436
numeric values
  based on true, false, or missing expressions 973
  choice from a list of arguments 175
  searching for, equal to first argument 689
numeric variables
  memory address of 714
  sorting argument values 837
NVALID function 1008
NWKDOM function 473

O
observations 864
  bookmarks, finding 1019
  bookmarks, setting 924
  number of current 864
  observation ID, returning 1006
  reading 903, 905
odd first period
  price per $100 face value 260, 279
  yield 260, 279
odd last period
  price per $100 face value 261, 280
  yield 262, 280
ODS output
  division and 235
  missing values for 235
OPEN function 1010
operating system commands 844
  executing 844
operating system variable, existing 1072
operating system variables, returning 1073
options on futures
  call prices for European, based on Black model 113
  put prices for European, based on Black model 115
options on stocks
  call prices for European, based on Black-Scholes model 116
ORDINAL function 475
output devices
  assigning/deassigning filerefs 910

P
parameters
  returning system parameter string 1075
Pareto distributions 165
  cumulative distribution functions 165
  probability density functions 491
PATHNAME function 1013
pattern matching 42, 533
  definition 42
  Perl regular expression (PRX) functions and CALL routines 43
  Perl regular expressions (PRX) in DATA step 43
  replacement 527
  writing Perl debug output to log 52
payment on principal 263
PCTL function 476
PDF function 477
PEEK function 1014
  compared to PEEKC function 1016
PEEK function 1016
  compared to PEEK function 1015
PEEKCLONG function 1017
PEEKLONG function 1018
periodic cashflow stream
  convexity for 200
  modified duration for 238
  present value for 547
  periodic payment of annuity 262
  periods for an investment 259, 278
Perl
  compiling regular expressions 534
  Perl regular expression (PRX) functions and CALL routines 43
  Perl regular expressions (PRX)
    benefits of using in DATA step 43
    extracting substring from a string 48
    pattern matching with 42
    Perl Artistic License compliance 53
    syntax 43
    validating data 45
    writing Perl debug output to log 52
PERM function 497
  logarithm of 423
  permuting values 827
pipe files
  assigning/deassigning filerefs 913
PMT function 498
POINT function 1019
Poisson distributions 166, 229, 807
cumulative distribution functions 166
probabilities from 499
probability density functions 492
random numbers 807, 1036
POISSON function 499
POKE CALL routine 780
POKELONG CALL routine 781
population size, returning 444
postal codes 1066
converting FIPS codes to 920
converting to FIPS codes 1066
converting to state names 1066, 1067
converting ZIP codes to 1106
PPMT function 500
present value 265, 282
price
discounted security 264, 281
security paying interest at maturity 264, 281
security paying periodic interest 263, 281
treasury bills 267, 283
pricing functions 8
principal
cumulative 252, 274
future value of 256, 276
payment on 263, 280
printable characters
searching character string for 95
probabilities 499
beta distributions 501
binomial distributions 502
chi-squared distributions 505
F distribution 506
gamma distribution 507
hypergeometric distributions 508
negative binomial distributions 524
Poisson distributions 499
standard normal distributions 525
student's t distribution 526
probabilities, computing
examples 519
for multiple comparisons of means, example 519
many-one t-statistics, Dunnett's one-sided test 514
many-one t-statistics, Dunnett's two-sided test 515
studentized maximum modulus 516
studentized range 516
Williams' test 517
Williams' test, example 522
probability 504
probability density functions 477
Bernoulli distributions 479
beta distributions 479
binomial distributions 480
Cauchy distributions 481
chi-squared distributions 481
exponential distributions 484
F distributions 484
gamma distributions 485
geometric distributions 486
hypergeometric distributions 487
Laplace distributions 488
logistic distributions 488
lognormal distributions 489
negative binomial distributions 489
normal distributions 490
Pareto distributions 491
Poisson distributions 492
uniform distributions 494
Wald distributions 494
Weibull distributions 495
probability functions 418
logarithms of 418
PROBBETA function 501
PROBBNML function 502
PROBBNRM function 504
PROBCHI function 505
PROBF function 506
PROBGAM function 507
PROBHYP function 508
PROBIT function 510
PROBM function 511
PROBN function 524
PROBNRM function 525
PROBT function 526
product license verification 1078
product licensing 992
PROCAS function 1020
PRX metacharacters 1113
PRXCHANGE function 527
PRXPOSN function 1024
PRODUCT function 1025
put prices
European options on futures, Black model 115
for European options, based on Margrabe model 427
PUT function 545
put prices
European options on futures, Black model 115
for European options, based on Margrabe model 427
PUT statement 546
PUTC function 1025
compared to PUTN function 1029
PUTN function 1025, 1028
compared to PUTC function 1025
PVP function 547

Q
QTR function 548
QUANTILE function 549
quantiles
chi-squared distribution 853
F distribution 916
from standard normal distribution 510
from student's t distribution 628
gamma distribution 306
returning from beta distribution 112
specifying the left probability (CDF) 549
specifying the right probability (SDF) 1062
question mark (?) format modifier 334
INPUT function 334
question marks (??) format modifier 334
INPUT function 334
queues, returning values from 393
quotation marks 224
adding 553
concatenation and 846
removing 224
QUOTE function 553

R
radians
geodetic distance input in 314
RANB滨 CALL routine 794
RANB滨 function 1030
RANCAU CALL routine 797
RANCAU function 1031
RANCOMB 827
RAND function 555
random numbers 794, 797, 800, 802, 805,
807, 809, 812, 814, 1006
binomial distribution 794, 1030
Cauchy distribution 797, 1031
exponential distribution 800, 1032
gamma distribution 802, 1033
normal distribution 805, 1006, 1035
Poisson distribution 807, 1036
tabled probability distribution 809,
1037
triangular distribution 812, 1038
uniform distribution 814, 1039
random-number functions and CALL
routines 11
comparison of 15
seed values 11
RANEXP CALL routine 800
RANEXP function 1032
RANGAM CALL routine 802
RANGAM function 1033
RANGE function 568
ranges of values, returning 568
RANK function 568
RANNOR CALL routine
compared to RANNOR function 1035
RANNOR function 1035
RANPERK 829
RANPERM 831
RANPOI CALL routine 1036
compared to RANPOI function 1036
RANPOI function 1036
RANTBL CALL routine 1038
compared to RANTBL function 1038
RANTBL function 1037
RANTRI CALL routine
compared to RANTRI function 1039
RANTRI function 1038
RANUNI CALL routine 1040
compared to RANUNI function 1040
RANUNI function 1039
remainder values 438
RENAME function 1040
REPEAT function 569
RESOLVE function 1042
resolving arguments 745
retail calendar intervals 363
REVERSE function 570
REWIND function 1043
RIGHT function 571
RMS function 572
root mean square 572
ROUND function 573
ROUNDDE function 580
rounding 573
ROUNDZ function 582

S
SAS catalog entries, verifying existence 850
SAS catalogs 850
verifying existence 850
SAS data sets
character variables, returning values of
944
closing 854
note markers, returning 888
numeric variables, returning values of
945
opening 1010
setting data set pointer to start of 1043
variable data type, returning 1097
variable labels, returning 1091
variable length, returning 1093
variable names, returning 1094
variable position, returning 1095
SAS dates 31
SAS functions
  See functions
SAS libraries
  pathnames, returning 1013
SA VING function 1044
SA VINGS function 585
SCAN function 586
SDF function 596
searching
  character strings 295
  encoding strings for 1058
  for character value, equal to first argument 688
  for numeric value, equal to first argument 689
seasonal cycle 378
seasonal cycles 350
seasonal indexes 361
SEC function 599
secant 599
SECOND function 600
seed values 11
SHA256 function 1045
SHA256HEX function 602
SHA256HMACHEX function 604
shift interval
  corresponding to base interval 382
SIGN function 601
signs, returning 601
SIN function 605
sine 605
  inverse hyperbolic 106
SINH function 606
skewness 607
SKEWNESS function 607
SLEEP function 608
SMALLEST function 609
SOAPWEB function 1046
SOAPWEBMETA function 1048
SOAPWIPSERVICE function 1050
SOAPWIPSRS function 1052
SOAPWS function 1055
SOAPWSMETA function 1056
softmax value 835
software images
  existence of 992
sorting
  character argument values 836
  numeric argument values 837
SOUNDEX function 1058
SPEDIS function 1059
spline
  monotonicity-preserving interpolating 999
SQR T function 611
SQUANTILE function 1062
square roots 611
standard deviations 611
standard error of means 612
standard normal distributions 510
  probabilities from 525
quantiles 510
state names
  converting FIPS codes to, mixed case 918
  converting FIPS codes to, uppercase 917
  converting ZIP codes to, mixed case 1105
  converting ZIP codes to, uppercase 1103
STD function 611
STDERR function 612
STFIPS function 1064
  compared to STNAME function 1066
  compared to STNAMEL function 1067
STNAME function 1065, 1066
  compared to STFIPS function 1065
  compared to STNAME function 1067
STNAMEL function 1065, 1067
  compared to STFIPS function 1065
  compared to STNAME function 1066
stocks
  call prices for European options on,
    Black-Scholes model 116
  call prices for European options on,
    Garman-Kohlhagen model 307
  call prices for European options, based
    on Margrabe model 425
  put prices for European options on,
    Garman-Kohlhagen model 309
  put prices for European options, based
    on Margrabe model 427
straight-line depreciation 266, 282
strings
  extracting substrings from 48
  message digest of 430
  removing blanks 634
  replacing or removing substrings 633
STRIP function 613
student's t distributions
  noncentrality parameter 629
  probabilities from 526
quantiles 628
studentized maximum modulus 516
studentized range 516
SUBPAD function 1068
subsetting 746, 946
SUBSTR (left of =) function 615
SUBSTR (right of =) function 617
substrings
extracting from arguments 617
extracting strings from 48
replacing or removing 633
SUBSTRN function 618
sum
of absolute values, for non-missing arguments 623
SUM function 622
sum-of-years digits depreciation 267, 283
SUMABS function 623
survival functions 420
computing 596
logarithms of 420
SYMEXIST function 1069
SYMGET function 1070
SYMGLOBL function 1071
SYMLOCAL function 1071
SYMPUT CALL routine 841
SYSEXIST function 1072
SYSGET function 1073
SYSMSG function 1074
SYSPARM function 1075
SYSPROCESSID function 1076
SYSPROCESSNAME function 1077
SYSPROD function 1078
SYSRANDOM macro variable
random number streams 28
SYSRANEND macro variable
random number streams 28
SYSRC function 1079
system error numbers, returning 1079
system parameter string, returning 1075
system-generated filerefs 912

T
distributions 166
cumulative distribution functions 166
probability density functions 492
tab characters
compressing 194
tabled probability distribution, random numbers 809
tabs
searching character string for 99
TAN function 624
tangent
inverse hyperbolic 107
tangents 624
TANH function 1079
target variables 5
TIME function 625
time intervals
See also date and time intervals
aligned between two dates 353
based on three date or datetime values 359
cycle index 341
recommended format for 356
seasonal cycle 350, 378
seasonal index 361
validity checking 384
time values
incrementing 370
time/date functions
time, returning current 625
TIMEPART function 626
TIMEVALUE function 626
TINV function 628
TNONCT function 629
TODAY function 630
trailing blanks, trimming 640
TRANSLATE function 632
compared to TRANWRD function 636
TRANSTRN function 633
TRANWRD function 632, 635
compared to TRANSLATE function 632
treasury bills
bond-equivalent yield 267, 283
price per $100 face value 267, 283
yield computation 268, 283
triangular distributions, random numbers 812, 1038
TRIGAMMA function 639
returning value of 639
TRIM function 640
compared to TRIMN function 642
trimming trailing blanks 640
TRIMN function 641, 642
compared to TRIM function 641
ttrue expressions 971, 973
TRUNC function 643
TWEEDIE distribution 493
TYPEOF function 1080
TZID
TZONENAME function 1082
TZONEDOFF function 1083
TZONES2U function 1085
TZONEU2S function 1086
uncorrected sum of squares 645
uniform distributions 168, 814
cumulative distribution functions 168
probability density functions 494
random numbers 814, 1039
UNIFORM function 644
Universal Unique Identifier (UUID) 646
UPCASE function 644
uppercase 644
converting character expressions to 644
searching character string for 101
UPCASE function 644
URLDECODE function 1087
URL ENCODE function 1088
URLs
decoding 1087
encoding 1088
escape syntax 1087, 1088
UUID function 645
UUID (Universal Unique Identifier) 646
UUIDGEN function 646
V
validating data 45
VALIDVARNAM E= system option
ANYFIRST function and 87
ANYNAME function and 93
values
signs, returning 601
VAR function 646
VARIANT function 1089
variable lists
Euclidean norm and 892
Lp norm and 991
variable names
searching character string for first character of 87
searching character string for valid character in 93
variables 124, 141
counter, returning values of 944
data type, returning 1097
format decimal values, returning 652
format names, returning 654
format width, returning 656
informat decimal values, returning 663
informat names, returning 665
informat width, returning 667
labels, assigning 124
labels, returning 670, 1092
length, returning 1093
names, assigning 141
names, returning 675, 1094
numeric, returning values of 945
operating system, returning 1073
position, returning 1095
size, returning 672
target variables 5
type, returning 677
values, returning 676
v ariance 646
VARI FMT function 1091
VARNAM E function 1092
VARED function 1093
VARNAM E function 1094
VARNUM function 1095
VARRAY function 647
compared to VARRAYX function 648
VARRAYX function 647, 648
compared to VARRAY function 647
VARTYPE function 1097
VERIFY function 649
vertical tabs
searching character string for 99
VFOMAT function 651
compared to VFOMATX function 659
VFOMATD function 652
compared to VFOMATDX function 653
VFOMATDX function 652, 653
compared to VFOMATD function 652
VFOMATN function 654
compared to VFOMATNX function 656
VFOMATNX function 654, 655
compared to VFOMATN function 654
VFOMATW function 656
compared to VFOMATWX function 658
VFOMATRX function 657
VFOMATX function 651, 658
compared to VFOMAT function 651
VINARRAY function 659
compared to VINARRAYX function 661
VINARRAYX function 660
compared to VINARRAY function 660
VINFORMAT function 662
compared to VINFORMATX function 670
VINFORMAT function 663
compared to VINFORMATDX function 664
VINFORMATDX function 664
compared to VINFORMATD function 663
VINFORMATN function 665
compared to VINFORMATNX function 666
VINFORMATNX function 666
compared to VINFORMATN function 665
VINFORMATW function 667
compared to VINFORMATWX function 670
VINFORMATWX function 668
compared to VINFORMATW function 668
VINFORMATX function 662, 669
compared to VINFORMAT function 662
VLABEL function 670, 1093
compared to VARLABEL function 1093
compared to VLABELX function 672
VLABELX function 671
compared to VLABEL function 671
VLENGTH function 672, 1094
compared to VARLEN function 1094
compared to VLENGTH function 674
VLENGTHX function 674
VNAME function 675
VNAMEX function 676
compared to VNAME function 675
VTYPE function 677
compared to VTYPEX function 679
VTYPEX function 678
compared to VTYPE function 678
VVALUE function 679
VVALUEX function 681

WHICH function 688
WHICHN function 689
white-space characters
searching character string for 99
Williams' test 522
words
character position in a string 295
converting, in character strings 209
number of a word in a string 295
replacing all occurrences of 634
searching character expressions for 330
writing values to memory 780

Y
YEAR function 690
yield
bond-equivalent 267, 283
discounted security 270, 285
odd first period 260, 279
odd last period 262, 280
security paying interest at maturity 270, 285
security paying periodic interest 269, 284
treasury bills 268, 283
YIELD function 691
YRDIF function 692
YYQ function 695

Wald distributions 168
cumulative distribution functions 168
probability density functions 494
Web applications
functions for 54
Web service
basic Web authentication 1046, 1048
WS-Security authentication 1050, 1052, 1055, 1056
WEEK function 682
WEEKDAY function 687
weekdays
dates of 473
Weibull distributions 169
cumulative distribution functions 169
probability density functions 495

Z
ZIP codes
city name and postal code for 1098
converting to FIPS codes 1101
converting to mixed case state names 1105
converting to postal codes 1106
converting to uppercase state names 1103
geodetic distance between two 1100
ZIPCITY function 1098
ZIPCITYDISTANCE function 1100
ZIPFIPS function 1101
ZIPNAME function 1103
ZIPNAMEL function 1106
ZIPSTATE function 1105
Gain Greater Insight into Your SAS® Software with SAS Books.

Discover all that you need on your journey to knowledge and empowerment.

support.sas.com/bookstore for additional books and resources.