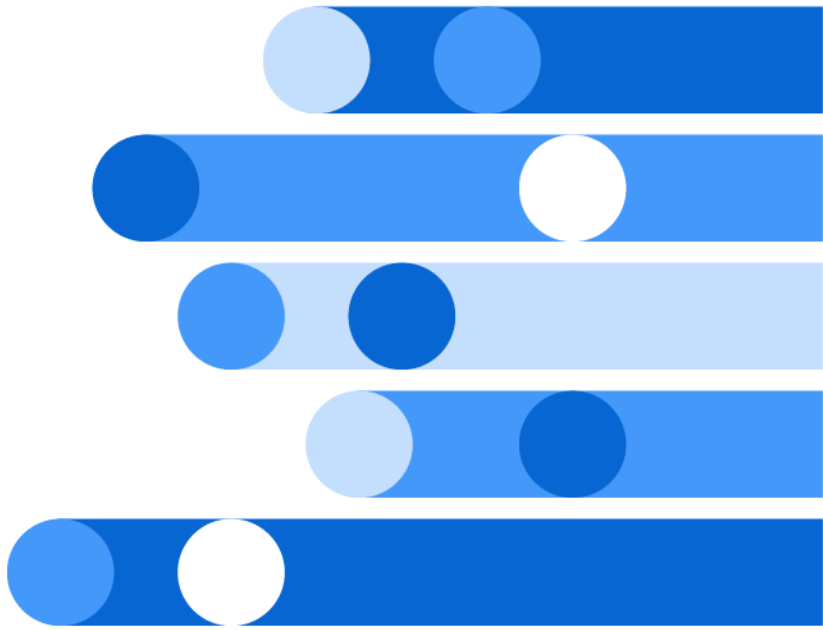
SAS® DS2 言語リファレンス

2020.1 - 2026.03*



* このドキュメントは、ソフトウェアの追加バージョンに適用される場合があります。このドキュメントを SAS Help Center で開き、バナーのバージョンをクリックすると、使用できるすべてのバージョンが表示されます。

SAS® ドキュメント
2026 年 3 月 26 日

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2020. *SAS® DS2 言語リファレンス*. Cary, NC: SAS Institute Inc.

SAS® DS2 言語リファレンス

Copyright © 2020, SAS Institute Inc., Cary, NC, USA

All Rights Reserved. Produced in the United States of America.

For a hard copy book: No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

For a web download or e-book: Your use of this publication shall be governed by the terms established by the vendor at the time you acquire this publication.

The scanning, uploading, and distribution of this book via the Internet or any other means without the permission of the publisher is illegal and punishable by law. Please purchase only authorized electronic editions and do not participate in or encourage electronic piracy of copyrighted materials. Your support of others' rights is appreciated.

U.S. Government License Rights; Restricted Rights: The Software and its documentation is commercial computer software developed at private expense and is provided with RESTRICTED RIGHTS to the United States Government. Use, duplication, or disclosure of the Software by the United States Government is subject to the license terms of this Agreement pursuant to, as applicable, FAR 12.212, DFAR 227.7202-1(a), DFAR 227.7202-3(a), and DFAR 227.7202-4, and, to the extent required under U.S. federal law, the minimum restricted rights as set out in FAR 52.227-19 (DEC 2007). If FAR 52.227-19 is applicable, this provision serves as notice under clause (c) thereof and no other notice is required to be affixed to the Software or documentation. The Government’s rights in Software and documentation shall be only those set forth in this Agreement.

SAS Institute Inc., SAS Campus Drive, Cary, NC 27513-2414

March 2026

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.

v_001-P1:ds2ref

目次

1 部 概要 1

1 章 / DS2 言語について **3**
DS2 言語について 3
DS2 プログラムの実行 4
サポートされているデータソース 5
対象とするユーザー 6
DS2 を使用する場合 7
DATA ステッププログラムから DS2 プログラムへの変換 7
DS2 言語の構文規則 7

2 部 入門ガイド 11

2 章 / DS2 と DATA ステップ **13**
DS2 について 13
DS2 と DATA ステップの類似点 13
DS2 と DATA ステップの相違点 14

3 章 / 例による学習: サンプルプログラムの使用 **23**
入門ガイドのサンプルプログラムについて 23
最初のサンプルプログラム 25
関連項目 36

4 章 / DS2 プログラムの構成ブロック **37**
DS2 言語の基本概念 37
DS2 プログラムについて 44
例: ブロックスコープ 45
例: 単純なスレッドプログラム 47

5 章 / DS2 メソッドとパッケージの理解 **51**
DS2 でのモジュール化、カプセル化、抽象化 51
DS2 メソッドについて 52
DS2 パッケージについて 55
例: DS2 メソッドの概要 58
例: DS2 パッケージの概要 59

3 部 DS2 言語リファレンス 63

6 章 / DS2 出力形式 65
出力形式の概要 67
一般的な出力形式構文 67
DS2 での出力形式の使用 68
DS2 出力形式の検証 70
DS2 出力形式の例 70
出力形式カテゴリ 70
ディクショナリ 77

7 章 / DS2 関数 235
DS2 関数の概要 244
一般的な関数構文 244
関数の使用 246
DS2 関数の例 250
関数カテゴリ 250
ディクショナリ 273

8 章 / DS2 入力形式 1285
入力形式の概要 1285
一般的な入力形式構文 1285
DS2 での入力形式の使用方法 1286
DS2 で入力形式を指定する方法 1287
DS2 入力形式の検証 1287
DS2 入力形式の例 1287

9 章 / DS2 演算子 1289
ディクショナリ 1289

10 章 / DS2 ステートメント 1293
ステートメントの概要 1294
ブロックステートメント 1295
グローバル宣言ステートメント 1297
ローカルステートメント 1298
DS2 プログラムにコメントを含める 1299
ディクショナリ 1300

11 章 / DS2 システムメソッド 1443
システムメソッドの概要 1443
ディクショナリ 1445

12 章 / DS2 システムオプション 1453
システムオプションの概要 1453
ディクショナリ 1454

13 章 / DS2 テーブルオプション 1457
テーブルオプションの概要 1458
DS2 でのテーブルオプションの使用 1458
DS2 でテーブルオプションを指定する方法 1460
DS2 テーブルオプションの例 1460

DS2 では無効な SAS データセットオプション 1461
データソース別の DS2 ステートメントテーブルオプション 1461
ディクショナリ 1469

14 章 / DS2 データグリッドパッケージのメソッド、演算子、およびステートメント 1531
ディクショナリ 1533

15 章 / DS2 FCMP パッケージのメソッド、演算子、およびステートメント 1637
ディクショナリ 1637

16 章 / DS2 ハッシュおよびハッシュ反復子パッケージの属性、メソッド、演算子、ステートメント 1645
ディクショナリ 1646

17 章 / DS2 HTTP パッケージのメソッド、演算子、およびステートメント 1729
メソッド命名規則 1730
SAS LOCKDOWN サポート 1730
ディクショナリ 1730

18 章 / DS2 JSON パッケージのメソッド、演算子、およびステートメント 1771
ディクショナリ 1772

19 章 / DS2 ロガーパッケージのメソッド、演算子、およびステートメント 1807
ディクショナリ 1807

20 章 / DS2 行列パッケージのメソッド、演算子、およびステートメント 1817
ディクショナリ 1818

21 章 / DS2 PCRXFIND パッケージのメソッド、演算子、およびステートメント 1893
ディクショナリ 1893

22 章 / DS2 PCRXREPLACE パッケージのメソッド、演算子、およびステートメント 1911
ディクショナリ 1911

23 章 / DS2 RedisCli パッケージのメソッド、演算子、およびステートメント 1919
概要 1919
接続操作の順序 1920
ディクショナリ 1922

24 章 / DS2 SQLSTMT パッケージのメソッド、演算子、およびステートメント 1943
ディクショナリ 1944

25 章 / DS2 TZ パッケージのメソッド、演算子、およびステートメント 2021
ディクショナリ 2021

4 部 付録 2037

付録 1 / データ型リファレンス 2039
SAS データセットのデータ型 2040
SPD Engine データセットのデータ型 2041
SPD Server テーブルのデータ型 2042

CAS のデータ型 2044
Amazon Redshift のデータ型 2046
UNIX および PC ホストでの DB2 のデータ型 2048
Google BigQuery のデータ型 2049
Greenplum のデータ型 2051
Hive のデータ型 2052
Impala のデータ型 2054
JDBC のデータ型 2056
MDS のデータ型 2058
Microsoft SQL Server のデータ型 2060
MongoDB のデータ型 2062
MySQL のデータ型 2064
Netezza のデータ型 2066
ODBC のデータ型 2067
Oracle のデータ型 2069
PostgreSQL のデータ型 2070
Salesforce から読み取るためのデータ型 2073
Salesforce に書き込むためのデータ型 2075
SAP のデータ型 2076
SAP HANA のデータ型 2078
SAP IQ のデータ型 2080
SingleStore のデータ型 2081
Snowflake のデータ型 2084
Spark のデータ型 2086
Teradata のデータ型 2088
Vertica のデータ型 2090
Yellowbrick のデータ型 2092

付録 2 / DS2 式 2095
式とは 2095
式のタイプ 2096
式の演算子 2120

付録 3 / DS2 サンプルプログラム 2127
例: 最小値を見つける 2129
例: DS2 プログラムの SQL 2131
例: 条件に基づいて 2 つの新しいテーブルを作成する 2134
例: テキスト出力の大文字/小文字の変更 2136
例: スコープ 2137
例: 関数 2139
例: 配列 2140
例: SELECT ステートメント 2145
例: ラベル付きの GOTO および LEAVE ステートメント 2146
例: オーバーロードされたメソッド 2149
例: 複数のスレッドを使用してテーブルを分析する 2151
例: 4 つのスレッドを使用して要約統計量を計算する 2153
例: データクリーニング 2155
例: SUBSTR 関数 2157
例: PUT で SAS 出力形式を使用 2158
例: テーブルデータから統計量を生成する 2159
例: 行列と非線形方程式 2161
例: DS2 行列と回帰分析 2165

例: バルク挿入を使用してテーブルを追加する 2168
例: 別のパッケージで SQLSTMT パッケージを使用する 2170
例: 2 つのデータベースを使用してテーブルの値を更新する 2171
例: FedSQL ステートメントをハッシュパッケージに保存する 2172
例: MySQL 8 CHAR()データから末尾のスペースをフィルタリングする 2178
例: HTTP および JSON パッケージを使用して REST 形式のファイルを抽出および解析する 2180
例: HTTP から JSON テキストを読み取り、SAS データセットを作成する 2184

1 部

概要

1 章
DS2 言語について 3

1

DS2 言語について

DS2 言語について . . . 3
DS2 プログラムの実行 . . . 4
サポートされているデータソース . . . 5
対象とするユーザー . . . 6
DS2 を使用する場合 . . . 7
DATA ステッププログラムから DS2 プログラムへの変換 . . . 7
DS2 言語の構文規則 . . . 7
文字体裁の規則 . . . 7
構文規則 . . . 8

DS2 言語について

DS2 は、高度なデータ操作に適した SAS 独自のプログラミング言語です。DS2 は SAS Viya プラットフォームに含まれており、SAS DATA ステップと交差します。また、ANSI SQL データ型を含む追加のデータ型、追加のプログラミング構造要素、およびユーザー定義のメソッドとパッケージも含まれています。

いくつかの DS2 言語要素は、埋め込まれた FedSQL 構文を受け入れ、ランタイムで生成されたクエリは、DS2 とサポートされているデータベース間でインタラクティブにデータを交換できます。これにより、入力テーブルの SQL 前処理が可能になり、2 つの言語の能力が効果的に組み合わされます。

SAS Viya プラットフォームでは、DS2 プログラムは次の環境で実行できます。

- SAS Cloud Analytic Services (CAS) Server
- SAS Compute Server
- SAS Event Stream Processing (ESP) Server
- SAS Federation Server
- SAS Micro Analytic Service

SAS Federation Server は、マルチユーザーの標準ベースのフェデレーションデータアクセスを提供する、個別にライセンスされたサーバーです。SAS Micro Analytic Service は、インストリームデータ分析とディシジョンの自動化をサポートする、メモリに常駐する高性能のプログラム実行プラットフォームです。マイクロサービスとして配置することも、ESP、CAS、計算サーバーなどのサーバープロセス内で実行することもできます。このサービスは、SAS Event Stream Processing ソリューションに加えて、SAS Intelligent Decision および SAS Model Manager で使用できます。

DS2 プロシジャを使用すると、SAS Studio から DS2 言語ステートメントをサブミットできます。PROC DS2 の詳細については、Base SAS プロシジャガイドを参照してください。

注: DS2 言語は多くのデータソースで使用できるため、データ要素を説明するために行、列、およびテーブルという用語が使用されます。これを SAS DATA ステップの用語と比較すると、行はオブザベーションに対応し、列は変数に対応し、テーブルはデータセットに対応します。

DS2 プログラムの実行

DS2 プログラムは、次のいずれかの方法でサブミットできます。

- DS2 プロシジャを使用して SAS Studio を介する方法。DS2 プロシジャは、SAS Compute Server または CAS サーバーで DS2 コードを実行するために使用できます。1 つの PROC DS2 ステップに、複数の DS2 プログラムを含めることができます。

  詳細については、*"DS2 プロシジャ" (Base SAS プロシジャガイド)*を参照してください。

- SAS Studio で、CAS サーバーで runDS2 アクションを使用します。runDS2 アクションは、CAS プロシジャと組み合わせて使用されます。

  注: Python または Lua を使用している場合を除き、PROC DS2 を使用して DS2 コードを CAS サーバーにサブミットすることをお勧めします。

  詳細については、*"DS2 アクションセット" (SAS Viya プラットフォーム: システムプログラミングガイド)*および*"CAS の DS2: 概念" (SAS DS2 プログラマガイド)*を参照してください。

- SAS Federation Server で SAS Federation Server 用の SAS LIBNAME エンジンを使用します。詳細については、*SAS LIBNAME Engine for SAS Federation Server: ユーザーガイド*を参照してください。

- SAS Event Stream Processing、SAS Intelligent Decisions、および SAS Model Manager で、SAS Micro Analytic Service を使用します。詳細については、*SAS Micro Analytic Service: プログラミングと管理ガイド*の About Using SAS Micro

Analytic Service および DS2 Programming for SAS Micro Analytic Service を参照してください。

サポートされているデータソース

DS2 は、次のデータソースにアクセスできます。

- Amazon Redshift
- CAS テーブル
- UNIX 動作環境用の DB2
- Google BigQuery
- Greenplum
- Hadoop (Hive)
- Impala
- JDBC に準拠しているデータベース(PostgreSQL など)
- メモリデータストア(MDS)
- Microsoft SQL Server
- MongoDB *
- MySQL (MySQL 8 サーバーを含む)
- Netezza
- ODBC に準拠しているデータベース(Microsoft SQL Server など)
- Oracle
- PostgreSQL
- Salesforce
- SAP (読み取り専用)
- SAP HANA
- SAP IQ
- SAS データセット
- SAS Scalable Performance Data Engine (SPD Engine)データセット
- UNIX 動作環境の SAS Scalable Performance Data Server (SPD Server)テーブル
- SingleStore
- Snowflake
- Spark
- UNIX 動作環境用の Teradata

- Vertica
- Yellowbrick

注: 次のデータソースはサポートされていません。

- Informix
- OLEDB
- SAP ASE

DS2 は、SAS プログラミングランタイム環境または CAS サーバー上の MDS および SPD Server を除くすべてのデータソースで 動作できます。

*MongoDB データソースには、テーブルを作成するための特別な要件があります。データに応じて、ルートテーブル、親テーブル、または子テーブルを作成することをお勧めします。DS2 ではルートテーブルのみを作成できます。親テーブルと子テーブルを作成するには、FedSQL を使用する必要があります。*SAS/ACCESS for Relational Databases: リファレンス*の MongoDB に関する情報を参照してくださ。

DS2 プロシジャと SAS Federation Server は、異なるデータソースをサポートします。それぞれがサポートするデータソースについては、*Base SAS プロシジャガイド* および *SAS Federation Server: 管理者ガイド*を参照してください。

データソースへのアクセスについては、必要に応じて *SAS/ACCESS for Relational Databases: リファレンス*および *SAS/ACCESS for Nonrelational Databases: リファレンス*を参照してください。

対象とするユーザー

このドキュメントの情報は、これらの役割を実行する次のユーザーを対象としています。

- クライアントアプリケーションを作成する**アプリケーション開発者**。テーブルの作成、テーブルのバルクロード、テーブルの操作、データのクエリを行うアプリケーションを作成します。
- クライアント/サーバー環境を設計および実装する**データベース管理者**。データベースを設計し、データソースのメタデータを設定することでデータを管理します。つまり、データベース管理者はデータモデルを構築します。
- 数値精度の向上、CPU バインドプロセスの並列計算、DATA ステッププロセスへの直接入力としての明示的なパススルー SQL クエリの使用、ビッグデータ環境での in-databas 処理など、DS2 言語の高度な機能を利用したいか、または利用する必要がある **SAS プログラマ**。

DS2 を使用する場合

通常、DS2 プログラムは、次のアクションを実行するアプリケーション用に作成されています。

- サポートされている新しいデータ型を使用した結果の精度を必要とする
- 変数スコープ、ユーザー定義のメソッドとパッケージ、スレッド処理のサポートなど、DS2 言語の新しいプログラミング要素を活用する
- DS2 プログラム内から SAS FedSQL を実行する必要がある
- SAS セッションの外部で実行する必要がある(たとえば、SAS Federation Server 上)

DATA ステッププログラムから DS2 プログラムへの変換

DSTODS2 プロシジャを使用して、DATA ステップコードを DS2 に変換することができます。すべての DATA ステップコードの変換がサポートされているわけではなく、手動での変換が必要になる場合があります。変換できないコード行はコメントに入れられます。詳細については、"DSTODS2 プロシジャ" (*Base SAS プロシジャガイド*)を参照してください。

DATA ステップコードを DS2 に変換し、プログラムが構文的に完全であることを確認したら、DS2 プロシジャを使用して、SAS Viya プラットフォームでプログラムを実行できます。

DS2 言語の構文規則

文字体裁の規則

DS2 言語構文のドキュメントで使用される書体には、特別な意味があります。

UPPERCASE BOLD
ステートメントや関数の名前などの DS2 キーワードを識別します(例: PUT)。

UPPERCASE ROMAN (大文字のローマン体)
リテラルの引数と値を識別します(例: FROM)。

italic
指定する引数または値を識別します。斜体の項目は、次のいずれかのユーザー指定の値を表すことができます。

- 引数に割り当てられた非リテラル値(例: ALTER=*alter-password*)。
- 非リテラル引数(例: KEEP=(*column-list*)。

斜体の項目を複数使用できる場合、その項目は *item* [, ...*item*]として表されます。

monospace
SAS コードの例を識別します。

構文規則

このドキュメントでは、バッカスナウア記法(BNF)を使用しています。具体的には、Jim Melton が *SQL:1999 Understanding Relational Language Components* で使用しているのと同じ構文表記法です。

従来の SAS 構文と DS2 言語リファレンスドキュメントで使用されている構文の主な違いは、オプションの構文引数の表示方法にあります。従来の SAS 構文では、オプションの構文を示すために山かっこ(< >)が使用されます。DS2 言語の構文では、角かっこ([ ])がオプションの構文を示すために使用され、山かっこは非終端の構成要素を示すために使用されます。

DS2 言語の構文では、次の記号が使用されます。

::=
この記号は、"で構成される"または"として定義されている"と解釈できます。

<>
山かっこは、非終端の構成要素(つまり、下位レベルの構文文法にさらに分解できる構文の構成要素を示します。

[ ]
角かっこはオプション引数を示します。角かっこで囲まれていない引数はすべて必須引数です。角かっこは、その前にリテラルであることを示すバックスラッシュ(\)が付いていない限り、入力しないでください。

{ }
中かっこは、必要な複数単語の引数を区別する方法を提供します。中かっこは、その前にリテラルであることを示すバックスラッシュ(\)が付いていない限り、入力しないでください。

|
縦棒は、グループの中から 1 つの値を選択できることを示しています。棒で区切られた値は相互に排他的です。

...
省略記号は、省略記号に続く引数または引数のグループが何度でも繰り返されることを示しています。省略記号と次の引数が角かっこで囲まれている場合はオプションです。

\
バックスラッシュは、次の文字がリテラルであることを示します。

次の例は、このセクションで説明されている構文規則を示しています。これらの例には、完全な構文ではなく、選択された構文要素が含まれています。

```
1 SET 2 <table-reference> [... [<table-reference>] [INDSNAME=variable];
  [ 3 BY [DESCENDING] 4 column [ 5 ...[DESCENDING]column];
  6 <table-reference>::=
  {table (table-options)} 7 | 8 \{sql-text 8 \}
```

1 SET はステートメントの名前であるため、大文字の太字で示されています。

2 <table-reference>は非終端引数であり、さらに下位レベルの構文文法に分解されるため、山かっこで囲まれています。少なくとも 1 つの<table-reference>を指定する必要があります。

3 BY と DESCENDING はリテラル引数であるため、大文字のローマン体で表記されます。DESCENDING はオプションの引数であるため、角かっこで囲まれています。

4 *column* は、指定できる引数であるため、斜体で示しています。

5 *column* の 2 番目のインスタンスを囲む角かっこと省略記号は、引数がカンマで区切られている限り、この引数を何度でも繰り返すことができることを示しています。

6 <table-reference>::=非終端引数の構文は次のように読み取られます: <table-reference>はテーブル名とテーブルオプション、または埋め込み SQL テキストで構成されます。

7 縦棒(|)は、*table* [*table-options*]または *sql-text* のいずれかを指定できますが、両方は指定できないことを示します。

8 *sql-text* を囲む中かっこの前のバックスラッシュ(\)は、それらの中かっこがリテラルであり、入力する必要があることを示します。

2 部

入門ガイド

2 章
DS2 と DATA ステップ 13

3 章
例による学習: サンプルプログラムの使用 23

4 章
DS2 プログラムの構成ブロック 37

5 章
DS2 メソッドとパッケージの理解 51

2

DS2 と DATA ステップ

DS2 について 13
DS2 と DATA ステップの類似点 13
DS2 と DATA ステップの相違点 14

DS2 について

DS2 は、データ操作アプリケーションやデータモデリングアプリケーションに使用される SAS Proprietary プログラミング言語です。DS2 言語は、コアとなる機能を DATA ステップと共有します。しかし、DS2 には、DATA ステップでは提供されない機能があります。

DS2 は、変数とスコープ、メソッド、パッケージ、コントロールフローステートメント、テーブル I/O ステートメント、並列プログラミングステートメントを備えた手続き型言語です。メソッドとパッケージによって、DS2 のモジュール性とデータのカプセル化が提供されます。DS2 を使用すると、SQL を SET ステートメントに直接挿入できるため、2 つの強力なデータ操作言語の力を融合させることができます。

DS2 と DATA ステップの類似点

DS2 と DATA ステップは多くの言語要素を共有しており、これらの要素はほとんど同じように動作します。

- SAS 出力形式。
- ほとんどの SAS 関数。
- DATA、SET、KEEP、DROP、RUN、BY、RETAIN、PUT、OUTPUT、DO、IF-THEN/ELSE、Sum などの SAS ステートメント。
- DATA ステップキーワードは、DS2 キーワードのリストに含まれています。

DS2 を使用して、多くの DATA ステップタスクを実行できます。

- 変数配列、多次元配列、ハッシュテーブルの処理
- データ型の変換
- 式の操作
- 日時値の計算
- 欠損値の処理

注: このドキュメントでは、サポートされているすべての DS2 構文について説明します。他の SAS ドキュメントに記載されている構文は、ここにも記載されていない限り、サポートされている DS2 構文の一部ではありません。

DS2 と DATA ステップの相違点

表 2.1 トピック別比較

トピック	DATA ステップ	DS2
実行可能なステートメント	DATA ステップのすべての実行可能なステートメントは、単一のコードブロックに存在します。	DS2 プログラムの実行可能なステートメントは、メソッドコードブロックに存在します。DS2 プログラムは、1 つ以上のメソッドコードブロックから構成されます。
スコープ	スコープの概念はありません。すべての変数は、グローバルです。	メソッドで宣言される変数のスコープは、ローカルです。他のすべての識別子のスコープは、グローバルです。 グローバルスコープには、次の 3 つの種類があります。 ■ データプログラム ■ スレッドプログラム ■ パッケージ
変数の宣言	変数を明示的に宣言する必要はありません。変数は割り当てによって作成できます。変数のデータ型は、それが最初に使用される方法のコンテキストによって決定されます。すべての変数のスコープは、グローバルです。	変数は、DECLARE ステートメントを使用して宣言されます。このステートメントは、変数のデータ型とスコープ属性も決定します。 変数は割り当てによって宣言できますが、ベストプラクティス

トピック	DATA ステップ	DS2
	変数は、SET ステートメントが使用されるときにも定義されます。	は、システムオプション DS2SCOND=ERROR または PROC DS2 オプション SCOND=ERROR を設定して、変数宣言の strict モードを適用することです。宣言されていない変数に対して警告メッセージが発行されます。 グローバル変数は、MERGE、SET、または SETFROM ステートメントが使用される場合にも定義されます。
変数の属性	変数の属性を設定するには、LENGTH、FORMAT、INFORMAT、LABEL、ATTRIB ステートメントを使用する必要があります。	変数の属性を設定するには、DECLARE ステートメントとその HAVING 句のみが必要です。
データ型	数値と文字の 2 つのデータ型がサポートされています。数値データは、3-8 バイトの浮動小数点です。文字データは、固定長または可変長です。	ほとんどの ANSISQL データ型がサポートされます。数値データ型のサイズと精度は、さまざまです。文字データ型は、固定長と可変長が可能です。DS2 は、ANSI の日付、時間、タイムスタンプのデータ型をサポートしますが、変換関数を使用して SAS 日付値、時間値、日時値を処理することもできます。
キーワードと予約語	予約キーワードはありません。	キーワードは予約語です。予約語のリストについては、“DS2 言語の予約語” (*SAS DS2 プログラマガイド*)を参照してください。予約語は、二重引用符で囲むことにより、識別子として使用できます。
引用符	単一引用符または二重引用符は文字定数を指定します。同値である 2 つの例を次に示します。 "Tom" 'Tom' 名前に特殊文字が含まれるテーブル名と列名(区切られた識別子)は、名前リテラル(引用符の後に文字 n が続く)として指定されま	ANSI SQL の引用符標準は次のとおりです。 ■ 単一引用符は、文字定数を指定します。 ■ 二重引用符は、区切られた識別子を指定します。 たとえば、"Tom"は区切られた識別子であり、'Tom'は文字定数です。

トピック	DATA ステップ	DS2
	す。たとえば、"Tom"n は SAS 名前リテラルです。	
マクロ変数参照から解決されるテキスト	マクロ変数の解決された値を参照するには、二重引用符が必要です。次に例を示します。 my_host = "&syshostname";	ANSI SQL の引用符標準に準拠しているため、二重引用符は区切られた識別子を示します。リテラル文字列でマクロ変数を参照するには、%TSLIT マクロ関数を使用します。次に例を示します。 my_host = %tslit(&syshostname);
PUT ステートメント	DATA ステップと DS2 の両方に、SAS ログに行を書き込むための PUT ステートメントがあります。DATA ステップと DS2 PUT ステートメントはどちらも、フォーマットされた変数値と名前付き変数値の書き込みをサポートしています。 DATA ステップでは、指定された列への書き込みと変数リストの使用もサポートします。	DS2 では、指定された列への書き込みと変数リストの使用はサポートしません。
欠損値と null 値	欠損値のみをサポートします。null 値の概念はありません。	欠損値と null 値の両方をサポートします。データベースからの null は、ANSI モードまたは SAS モードで処理できます。
データ型の自動変換	SAS では、文字データ型と数値データ型の間での変換が、一方のデータ型がもう一方のデータ型の変数に割り当てられるときに試みられます。	より多くのデータ型があるため、より多くのルールがあります。ほとんどのデータ型は強制的です。DATE、TIME、TIMESTAMP、BINARY、VARBINARY のデータ型は強制的ではありません。
SQL 言語ステートメント	PROC SQL で使用でき、DATA ステップでは使用できません。	DS2 SET ステートメント、SQLSTMT パッケージ、SQLEXEC 関数は、FedSQL ステートメントを受け入れます。詳細については、“DS2 と FedSQL の使用” (*SAS DS2 プログラマガイド*)を参照してください。
SAS マクロ	Base SAS の DATA ステップでは、実行時にマクロの DATA ステップ呼び出しルーチンを使用して、マクロと対話できます。	SAS マクロ処理は、PROC DS2 を使用してサブミットされる DS2 プログラムで使用できます。マクロは、DS プログラムがローカルで実行されるか、CAS サーバーや SAS Federation Server でリ

トピック	DATA ステップ	DS2
	SAS マクロ機能は、Cloud Analytic Services (CAS)内では実行されません。CAS サーバーにサブミットされる DATA ステップで指定されたマクロは、DATA ステップが CAS サーバーに送信される前に、マクロ機能によって前処理されます。	モートで実行されるかに関係なく、DS2 プログラムが実行される前に SAS マクロ機能によって前処理されます。 DS2 は、マクロの DATA ステップ呼び出しルーチンをサポートしていません。DS2 が SAS Embedded Process を使用した in-database 処理などのグリッドコンピューティング環境で実行される場合、SAS マクロのサポートは利用できません。

データセットの上書き	DATA ステップは、SAS プロシジャと同様に、既存のデータセットを上書きします。
	`data one;`
	`  set two;`
	`run;`
	`data one;`
	`  set three;`
	`run;`
		SQL やデータベースの標準に準拠して、DS2 は既存のデータセットを自動的に上書きしません。**overwrite** テーブルオプションを使用する必要があります。次に例を示します。
		`proc ds2;`
		`data one;`
		`  method run();`
		`  set two;`
		`  end;`
		`enddata;`
		`run;`
		`data one / overwrite=yes;`
		`  method run();`
		`  set three;`
		`  end;`
		`enddata;`
		`run;`
		`quit;`

同じデータセットからの読み取りと同じデータセットへの書き込み	DATA ステップまたは PROC ステップでは、同じデータセット名からの読み取りと書き込みを許可します。
	`data one;`
	`  set one;`
	`  s = s + 1;`
	`run;`
		DS2 では、単一のデータプログラムで同じテーブルからの読み取りと同じテーブルへの書き込みはできません。データベースでの要領で、ユーザーは一時テーブルを作成し、元のテーブルを削除してから、一時テーブルの名前を変更する必要があります。この例では、DATA ステップや PROC ステップが正常に終了したときに、同じデータセットからの読み取りと同じデータセットへの書き込みの機能を実装するために SAS で行われることと同じことが行われます。
		`proc ds2;`
		`data temp001;`

トピック	DATA ステップ	DS2

		method run();
		set one;
		s = s * 1;
		end;
		enddata;
		run;
		quit;
		
		proc fedsql;
		drop table one;
		alter table temp001 rename to one;
		run;
		quit;

SAS データセットオプション	DATA ステップでは、特定のデータセットに適用されるアクションを指定できるようにする SAS データセットオプションが用意されています。SAS データセットオプションは、SAS データセット名の後にかっこで囲んで指定します。データセットオプションは、データアクセス、データセットコントロール、オブザベーションコントロール、SAS インデックス使用のユーザーコントロール、変数コントロールなど、複数のカテゴリの機能を提供します。	DS2 には、変数コントロールのためのデータセットオプションに類似したテーブルオプションがあります。たとえば、DROP=、KEEP=、IN=、RENAME=のデータセットオプションは、DS2 のテーブルオプションとしてサポートされています。DS2 には、オブザベーションコントロールのためのテーブルオプションはありません。かわりに、埋め込み SQL ステートメントを使用して、オブザベーション(行)コントロールを実行できます。データアクセスとテーブルコントロールのテーブルオプションは制限されていて、データソースによって 異なります。“データソース別の DS2 ステートメントテーブルオプション” (1461 ページ)を参照してください。 テーブルオプションは、使用されているステートメントに応じて、かっこで囲むか、スラッシュ(/)を前に付けます。スラッシュを前に付けたテーブルオプションは、ステートメントの最後に配置する必要があります。
データセットから返される行のフィルタリング	DATA ステップでは、行の選択基準は WHERE ステートメントまたは WHERE=データセットオプションで指定されます。WHERE データセットオプションの例を次に示します。 data ds1;	DS2 では、埋め込み FedSQL を使用して、行の選択基準を指定します。FedSQL ステートメントの使用は、SET ステートメントの **sql-text** 引数でサポートされています。また、SQLSTMT 事前定義パッケージのコンストラクターとしてサポートされています。

トピック	DATA ステップ	DS2

	set test(where=(x=2));
	put x=;
	run;
	x=2
		DS2 SET ステートメントで指定された FedSQL SELECT ステートメントの例を次に示します。
		
		proc ds2;
		data ds2;
		method run();
		set {select x from test where x=2};
		put x=;
		end;
		enddata;
		run;
		quit;
		x=2
		
		注: SET ステートメントで使用する場合、埋め込み SQL ステートメントは中かっこ({})の中に指定されます。

処理を連続行のセットに限定	DATA ステップでは、FIRSTOBS=データセットオプションと OBS=データセットオプションを使用して、処理される最初と最後のオブザベーションを指定します。
	
	data ds1;
	set study(firstobs=5 obs=10);
	run;
	
	proc print data=ds1;
	run;
		DS2 では、埋め込まれた FedSQLSELECT ステートメントで OFFSET 句と LIMIT 句を指定します。OFFSET は、SELECT ステートメントが行を返し始める前にスキップする行数を指定します。LIMIT は、SELECT ステートメントが返すことができる行数を指定します。
		
		proc ds2;
		data ds2;
		method run();
		set {select x from test offset 5 limit 6};
		end;
		enddata;
		run;
		quit;
		
		proc print data=ds2;
		run;

CAS テーブルへのアクセス	PROC DS2 または ds2.runds2 アクションのいずれかによって実行される DS2 プログラムと、DATA ステップまたは datastep.runcode アクションのいずれかによって実行される DATA ステッププログラムの両方で、caslib を使用して CAS テーブルにアクセスできます。さらに、プログラムが DATA ステップを介してサブミットされる場合、	DS2 では、CAS libref をサポートしません。PROC DS2 で CAS libref を使用しようとすると、DS2 はエラーを返します。

トピック	DATA ステップ	DS2
	DATA ステップで CAS libref がサポートされます。	
CAS でリモートで実行するプログラムの指定	PROC DS2 と DATA ステップはどちらも、プログラムを SAS セッションでローカルに実行するのではなく、CAS サーバーでリモートで実行するように明示的に指定するための SESSREF オプションと SESSUUID オプションをサポートしています。また、DATA ステップには、入力(SET)テーブルと出力(DATA)テーブルが CAS サーバーにある場合に、CAS サーバーでプログラムをリモートで実行することを暗黙的に決定する機能があります。	DS2 プログラムは、明示的に CAS にサブミットされます。
マルチスレッド処理	複数のスレッドは、CAS でのみ使用できます。	DS2 が実行されるすべての環境(SAS プログラミングランタイム環境、CAS、SAS Federation Server)で複数のスレッドがサポートされます。スレッド化されたコードブロックは、THREAD ステートメントを使用して定義されます。
ユーザー定義関数	PROC FCMP を使用すると、ユーザーは DATA ステップから呼び出すことができる SAS 関数を作成し、保存できます。	ユーザー定義関数のサポートは、METHOD ステートメントを使用して DS2 言語に組み込まれています。DS2 では、複数の DS2 プログラムで関数を再利用するために、PACKAGE ステートメントを使用して関数をグループ化して保存する機能を提供します。 DS2 は、PROC FCMP によって保存された SAS 関数を呼び出すための FCMP パッケージも提供します。
FCMP 関数の呼び出し	SAS システムオプションの CMPLIB=は、すでにコンパイルされている関数とサブルーチンを検索する場所を指定します。FCMP 関数とサブルーチンの使用をサポートするすべてのプロシジャ(FCMP を含む)は、このシステムオプションを使用します。	DS2 では、CMPLIB システムオプションや CMPLIB セッションオプションをサポートしません。FCMP 関数を呼び出すには、FCMP パッケージを使用します。 DS2 では、データセットに保存されている、PROC FCMP によって定義された関数を呼び出すことができます。DS2 FCMP パッケ

トピック	DATA ステップ	DS2
	CAS では、CMPLIB はセッションオプションです。	ージにより、データセットから FCMP 関数をロードして呼び出す機能が提供されます。詳細については、“FCMP パッケージの使用” (*SAS DS2 プログラマガイド*)を参照してください。

3

例による学習: サンプルプログラムの使用

入門ガイドのサンプルプログラムについて **23**
サンプルプログラムの実行方法 23
推奨オプション 24
DS2 プロシジャの使用 24
DS2 へのアクセスの確認 25
最初のサンプルプログラム **25**
サンプルプログラムの概要 25
例 1: “Hello World!”プログラム – システムメソッドの場合 26
例 2: “Hello World!”プログラム –ユーザー定義メソッドの場合 26
例 3: “Hello World!”プログラム – ユーザー定義パッケージの場合 27
例 4: “Hello World!”プログラム – 暗黙的なループの場合 29
例 5: “Hello World!”プログラム – 複数のパッケージインスタンスの場合 31
例 6: “Hello World!”プログラム – 配列パッケージ変数の場合 32
例 7: “Hello World!”プログラム – スレッドを使用する場合 34
関連項目 **36**

入門ガイドのサンプルプログラムについて

サンプルプログラムの実行方法

DS2 プロシジャを使用して、SAS セッションからこの章にある入門ガイドのサンプルプログラムをすべて 実行できます。それぞれのサンプルプログラムは完全なプログラムであり、SAS セッションにプログラムをコピーしてサブミットするだけで実行できます。

入門ガイドのサンプルプログラムは、既存のデータソースに依存せず、データベースへの接続も必要としません。データが必要な場合、プログラムによってデータが作成されます。

推奨オプション

入門ガイドのサンプルプログラムはすべて、次の SAS システムオプショングローバルステートメントを使用することで、正しく実行されます。

```
options DS2SCOND=ERROR;
```

次の DS2 プロシジャオプションを指定することで、デフォルトの DS2SCOND オプション設定である WARNING を上書きすることもできます。

```
proc ds2 SCOND=ERROR;
```

ERROR 設定は、変数宣言の strict モードを適用します。strict モードでは、変数を明示的に宣言しないと、コンパイルエラーが発生します。

> ヒント　DS2 プログラムを作成する場合は、変数宣言の strict モードが推奨されるベストプラクティスです。

DATA ステップとは異なり、DS2 は既存のテーブルが上書きされないように保護します。しかし、テーブル、パッケージ、スレッドを開発したり変更したりする場合は、これらのテーブルを上書きする機能が必要です。

OVERWRITE=YES テーブルオプションを使用すると、テーブルを上書きできます。次に例を示します。

```
package foo /overwrite=yes;
thread work.foo(int x) /overwrite=yes;
data foo (overwrite=yes);
data my_data /overwrite=yes;
```

> ヒント　OVERWRITE=オプションには、PACKAGE ステートメントと THREAD ステートメントを含むスラッシュ(/)構文が必要です。DATA ステートメントでは、/構文またはかっこ構文のいずれかを使用できます。

デフォルトでは、OVERWRITE=テーブルオプションの値は NO です。

DS2 プロシジャの使用

DS2 プロシジャを使用してプログラムを作成する場合は、次のフレームワーク内にコードを配置します。

```
options ds2scond=error;
proc ds2;
... DS2 statements ...
```

```
run;
quit;
```

詳細については、“DS2 プロシジャ” (*Base SAS プロシジャガイド*)を参照してください。

DS2 へのアクセスの確認

SAS セッションで、次のプログラムを実行します。

```
proc ds2;
quit;
```

次のようにログに書き込まれます。

```
3317  proc ds2;
3318  quit;

NOTE: PROCEDURE DS2 used (Total process time):
      real time          0.08 seconds
      cpu time           0.04 seconds
```

ERROR: Procedure DS2 not found というエラーメッセージが表示された場合、システム管理者に連絡してください。

最初のサンプルプログラム

サンプルプログラムの概要

多くのプログラマはコードを読んで学ぶことを好むため、この章では、プログラムが使用する言語コンストラクトを説明する前に、いくつかのサンプルプログラムを紹介します。サンプルプログラムは、DS2 の構文と概念をすばやく学習して、よくある落とし穴を回避するために役立ちます。

最初のサンプルプログラムは、"Hello World!"プログラムです。その後のサンプルプログラムは、このプログラムのバリエーションです。一部のバリエーションは、不必要に複雑に見える場合があります。ここでのポイントは、DS2 言語の共通する構造プログラミング要素を示すことであり、“Hello World!”を作成するための最良の方法を示すことではありません。

注: サンプルプログラムは、より複雑なものへと構文や概念を紹介していますが、完全に理解するために特定の順序に依存するものではありません。

例 1: “Hello World!”プログラム – システムメソッドの場合

ここでは、"Hello World!"プログラムのコーディング方法の一つを紹介します。このプログラムは、INIT( )システムメソッドから SAS ログに"Hello World!"を書き込みます。

注意事項

- 変数 MESSAGE は、INITINIT( )メソッドで宣言されているため、ローカルスコープを持っています。
- INITINIT( )システムメソッドは、DS2 データプログラムで最初に自動的に実行されます。
- 単一引用符は、文字定数を区切ります。
- libs=work オプションは、SAS Work ライブラリのみを使用するように接続文字列を制限します。詳細については、“PROC DS2 ステートメント” (*Base SAS プロシジャガイド*)を参照してください。

SAS セッションで、次のプログラムをコピーまたは入力します。次に、サブミットします。

```
proc ds2 libs=work;
data _null_;

 /* init() - system method */
 method init();
   declare varchar(16) message; /* method (local) scope */
   message = 'Hello World!';
   put message;
 end;
enddata;
run;
quit;
```

The following is written to the log:

```
Hello World!
```

例 2: “Hello World!”プログラム –ユーザー定義メソッドの場合

このバリエーションのプログラムは、ユーザー定義メソッドから SAS ログに"Hello World!"に書き込みます。

注意事項

- 変数 MESSAGE はデータプログラムのグローバルスコープを持っているため、すべてのメソッドがアクセスできます。
- INIT( )システムメソッドは、ユーザー定義の GREET( )メソッドを呼び出して、メッセージをログに書き込みます。

  注: GREET( )メソッドは、GREET( )メソッドを参照するメソッドの前に定義されます。そうしない場合、コンパイルエラーが発生します。

SAS セッションで、次のプログラムをコピーまたは入力します。次に、サブミットします。

```
proc ds2 libs=work;
data _null_;
 dcl varchar(16) message; /* data program (global) scope */

 /* greet() - user-defined method */
 method greet();
   put message;
 end;

 /* init() - automatically runs first in the data program.*/
 method init();
   message = 'Hello World';
   message = cat(message, '!');
   greet();
 end;
enddata;
run;
quit;
```

The following is written to the log:

```
Hello World!
```

例 3: “Hello World!”プログラム – ユーザー定義パッケージの場合

このバリエーションのプログラムは、ユーザー定義パッケージを介して、"Hello World!"と他のメッセージを SAS ログに書き込みます。

注意事項

- PACKAGE ステートメントは OVERWRITE=YES テーブルオプションを使用するため、エラーなしでプログラムを複数回実行できます。デフォルトでは、DS2 は既存のパッケージが上書きされないように保護します。
- 変数 MESSAGE は、データプログラムではなく、パッケージで宣言されます。
- パッケージには、挨拶文の文字列を操作するためのコンストラクターと 2 つのパッケージメソッドが含まれています。

  FORWARD ステートメントを使用すると、SETMESSAGE( )メソッドを、SETMESSAGE( )メソッドを参照するメソッドの後に定義できます。そうしない場合、コンパイルエラーが発生します。

  SETMESSAGE( )メソッドは、THIS 演算子を使用して、グローバル変数 MESSAGE を MESSAGE という名前のパラメーターと区別します。
- データプログラムでは、DECLARE PACKAGE ステートメントはパッケージ変数を宣言すると同時に、パッケージコンストラクターを使用してパッケージのインスタンスを作成します。
- ドット表記は、データプログラムからパッケージメソッドへのアクセスを提供します。

SAS セッションで、次のプログラムをコピーまたは入力します。次に、サブミットします。

```
proc ds2 libs=work;
/* GREETING - User-defined package that writes a message to the SAS log */
package greeting /overwrite=yes;
 dcl varchar(100) message;    /* package (global) scope */
 FORWARD setMessage;

 /* greeting(MESSAGE) - constructor */
 method greeting(varchar(100) message);
  setMessage(message);
 end;

 method greet();
  put message;
 end;

 method setMessage(varchar(100) message);
  /* Must use THIS. to distinguish global */
  /* variable MESSAGE from parameter named MESSAGE. */
  this.message = message;
 end;
endpackage;
run;

/* data program */
data _null_;
 /* declares and instantiates an instance of the GREETING package */
 dcl package greeting g('Hello World!');  /* data program (global) scope */

 /* init() - automatically runs first in the data program.*/
```

```
 method init();
   g.greet();
   g.setMessage('What''s new?'); /* change greeting */
   g.greet();
end;
enddata;
run;
quit;
```

The following is written to the log:

```
Hello World!
What's new?
```

例 4: “Hello World!”プログラム – 暗黙的なループの場合

このバリエーションには、2 つのデータプログラムが含まれています。1 つは挨拶文のテーブルを作成するためのもので、もう 1 つはそのテーブルを処理するためのものです。

注意事項

- 最初のデータプログラムでは、DATA ステートメントは OVERWRITE=YES テーブルオプションを使用するため、エラーなしでプログラムを複数回実行できます。デフォルトでは、DS2 は既存のパッケージが上書きされないように保護します。
- GREETING パッケージには、デフォルトのコンストラクターと引数を受け入れるコンストラクターの 2 つのコンストラクターがあります。
- 2 番目のデータプログラムは、パッケージをインスタンス化するために 2 ステップのメソッドを使用します。

  1.DECLARE PACKAGE ステートメントは、グローバルパッケージ変数を宣言します。

  2.INIT( )メソッドは、_NEW_演算子を使用してパッケージインスタンスを作成します。
- 2 番目のデータプログラムでは、RUN( )メソッドは SET ステートメントの暗黙的なループを使用して、テーブルの各行から MESSAGE 変数を読み取って処理します。

SAS セッションで、次のプログラムをコピーまたは入力します。次に、サブミットします。

```
proc ds2 libs=work;
/* data program # 1 - Creates a table of greetings */
data work.greetings /overwrite=yes;
 dcl char(100) message; /* data program (global) scope */

 method init();
```

```
    message = 'Hello World!'; output;
    message = 'What''s new?'; output;
    message = 'Good-bye World!'; output;
  end;
enddata;
run;
quit;

proc ds2;
/* GREETING - User-defined package that writes a message to the SAS log */
package greeting /overwrite=yes;
  dcl varchar(100) message;    /* package (global) scope */
  forward setMessage;

  /* greeting() - default constructor */
  method greeting();
    setMessage('This is the default greeting.');
  end;

  /* greeting(MESSAGE) - constructor */
  method greeting(varchar(100) message);
    setMessage(message);
  end;

  method greet();
    put message;
  end;

  method setMessage(varchar(100) message);
    /* Must use THIS. to distinguish global */
    /* variable MESSAGE from parameter named MESSAGE. */
    this.message = message;
  end;
endpackage;
run;

/* data program #2 */
data _null_;
  dcl package greeting g; /* package (global) scope */

  /* init() - automatically runs first in the data program. */
  method init();
    g = _NEW_ greeting();
    g.greet();
  end;

  /* run() - automatically runs after INIT() completes. */
  method run();
    /* Implicit loop reads each row from the table */
    set work.greetings;
    g.setMessage(message); /* MESSAGE is read from row by SET statement */
    g.greet();
  end;
enddata;
run;
quit;
```

The following is written to the log:

```
This is the default greeting.
Hello World!

What's new?

Good-bye World!
```

例 5: “Hello World!”プログラム – 複数のパッケージインスタンスの場合

このバージョンのプログラムは、パッケージの複数のインスタンスを使用して、テーブルを作成せずに例 4 と同じ結果を取得します。

注意事項

- GREETING パッケージは、前の例の GREETING パッケージと同一です。パッケージはすでに存在するため、パッケージブロックはこのプログラムから省略されている可能性があります。
- 同じ DECLARE PACKAGE ステートメントで異なるコンストラクターを使用できます。

SAS セッションで、次のプログラムをコピーまたは入力します。次に、サブミットします。

```
proc ds2 libs=work;
/* GREETING - User-defined package that writes a message to the SAS log */
package greeting /overwrite=yes;
 dcl varchar(100) message;    /* package (global) scope */
 FORWARD setMessage;

 /* greeting() - default constructor */
 method greeting();
  setMessage('This is the default greeting.');
 end;

 /* greeting(MESSAGE) - constructor */
 method greeting(varchar(100) message);
  setMessage(message);
 end;

 method greet();
  put message;
 end;

 method setMessage(varchar(100) message);
  /* Must use THIS. to distinguish global */
```

```
   /* variable MESSAGE from parameter named MESSAGE. */
   this.message = message;
 end;
endpackage;
run;

/* data program */
data _null_;
 dcl package greeting
 g0() g1('Hello World!') g2('What''s new?') g3('Good-bye World!');

 /* init() - automatically runs first in the data program.*/
 method init();
   g0.greet();
   g1.greet();
   g2.greet();
   g3.greet();
 end;
enddata;
run;
quit;
```

The following is written to the log:

```
This is the default greeting.
Hello World!
What's new?
Good-bye World!
```

例 6: “Hello World!”プログラム – 配列パッケージ変数の場合

このバージョンのプログラムでは、配列パッケージ変数を使用して例 4 および 5 と同じ結果を取得します。

注意事項

- GREETING パッケージは、前の例の GREETING パッケージと同一です。
- DECLARE PACKAGE ステートメントは、配列パッケージ変数を宣言します。配列パッケージ変数を使用すると、複数のパッケージインスタンスを 1 つの変数にグループ化できます。
- パッケージインスタンスが作成され、コンストラクターを使用して DECLARE PACKAGE ステートメント内の配列パッケージ変数の各要素に割り当てられます。この例では、メッセージは、デフォルトのメッセージを保存する最初の呼び出しを除き、各コンストラクター呼び出しで指定されます。
- DO ループは、DIM 関数を使用して配列パッケージ変数の各要素を反復処理します。パッケージ GREETINGS の GREET メソッドは、ドット表記を使用して各パッケージインスタンスで呼び出され、メッセージをログに出力します。

SAS セッションで、次のプログラムをコピーまたは入力します。次に、サブミットします。

```
proc ds2;
/* GREETING - User-defined package that writes a message to the SAS log */
package greeting /overwrite=yes;
 dcl varchar(100) message;    /* package (global) scope */
 FORWARD setMessage;

 /* greeting() - default constructor */
 method greeting();
  setMessage('This is the default greeting.');
 end;

 /* greeting(MESSAGE) - constructor */
 method greeting(varchar(100) message);
  setMessage(message);
 end;

 method greet();
  put message;
 end;

 method setMessage(varchar(100) message);
  /* Must use THIS. to distinguish global */
  /* variable MESSAGE from parameter named MESSAGE. */
  this.message = message;
 end;
endpackage;
run;

data _null_;
/* init() - automatically runs first in the data program.*/
 method init();
/* declare array package variable with constructor arguments */
  dcl package greeting garray[4] := [(), ('Hello World!'),
('What''s new?'), ('Good-bye World!')];
  dcl int i;

  do i = 1 to dim(garray);
   garray[i].greet();
  end;
 end;
enddata;
run;
quit;
```

The following is written to the log:

```
This is the default greeting.
Hello World!
What's new?
Good-bye World!
```

例 7: “Hello World!”プログラム – スレッドを使用する場合

このバージョンのプログラムは、スレッドを使用してテーブルを 読み取り、変数をデータプログラムに渡します。

注意事項

- この単一の DS2 プログラムには、パッケージとスレッドプログラムの 2 つのデータプログラムが含まれています。
- データプログラムは、SET FROM ステートメントで 2 つのスレッドを指定します。
- スレッドプログラムでは、RUN( )メソッドは SET ステートメントの暗黙的なループを使用して、テーブルから各 MESSAGE 変数を読み取って処理します。
- データプログラムでは、RUN( )メソッドは SET FROM ステートメントの暗黙的なループを使用して、スレッドプログラムから各 MESSAGE 変数を読み取って処理します。
- ログのスレッドプログラムの TERM( )出力は、スレッドプログラムが、スレッドごとに 1 回ずつ、2 回実行されたことを示します。さらに、_N_の値は、各スレッドに代わって RUN( )メソッドが実行された回数を示します。

SAS セッションで、次のプログラムをコピーまたは入力します。次に、サブミットします。

```
proc ds2 libs=work;
/* data program #1 - Creates a table of greetings */
data work.greetings /overwrite=yes;
 dcl char(100) message; /* data program (global) scope */

 method init();
   message = 'Hello World!'; output;
   message = 'What''s new?'; output;
   message = 'Good-bye World!'; output;
 end;
enddata;
run;

/* GREETING - User-defined package that writes a message to the SAS log */
package greeting /overwrite=yes;
 dcl varchar(100) message;    /* package (global) scope */
 forward setMessage;

 /* greeting() - default constructor */
 method greeting();
   setMessage('This is the default greeting.');
 end;
```

```
 /* greeting(MESSAGE) - constructor */
 method greeting(varchar(100) message);
  setMessage(message);
 end;

 method greet();
  put message;
 end;

 method setMessage(varchar(100) message);
  /* Must use THIS. to distinguish global */
  /* variable MESSAGE from parameter named MESSAGE. */
  this.message = message;
 end;
endpackage;
run;

/* thread program - Read the table */
thread work.t /overwrite=yes;

 /* run() - system method */
 method run();
  set work.greetings;
  output; /* output variables to calling program */
 end;

 /* term() - system method */
 method term();
  put _all_;
 end;
endthread;
run;

/* data program #2 */
data _null_;
  dcl package greeting g; /* data program (global) scope */
  dcl thread work.t t; /* data program (global) scope */

 /* init() - automatically runs first in the data program. */
  method init();
    g = _NEW_ greeting();
    g.greet();
  end;

 /* run() - automatically runs after INIT() completes. */
 method run();
    /* Implicit loop reads each row from the table */
    set from t threads=2;
    g.setMessage(message); /* MESSAGE read from row by SET FROM stmt */
    g.greet();
  end;
enddata;
run;
quit;
```

The following is written to the log:

```
This is the default greeting.
_N_=4 message=Good-bye World!

_N_=1 message=

Hello World!

What's new?

Good-bye World!
```

関連項目

- DS2 の概念の概要については、4 章, “DS2 プログラムの構成ブロック” (37 ページ)を参照してください。
- DS2 メソッドとパッケージの詳細については、5 章, “DS2 メソッドとパッケージの理解” (51 ページ)を参照してください。
- DS2 プログラムの高度な実例を見るには、付録 3, “DS2 サンプルプログラム” (2127 ページ)を参照してください。

4

DS2 プログラムの構成ブロック

DS2 言語の基本概念 37
DS2 データ型の紹介 37
データ型の自動変換 39
DS2 プログラミングブロックとスコープ 40
DS2 での変数宣言 42
DS2 メソッドとパッケージ 43
DS2 での並列処理 43
DS2 プログラムについて 44
例: ブロックスコープ 45
例: 単純なスレッドプログラム 47

DS2 言語の基本概念

DS2 データ型の紹介

SAS DATA ステップ言語とは異なり、DS2 は、SAS がサポートするデータソースにネイティブな ANSISQL データ型の多くをサポートします。したがって、データソースに保存されているデータにアクセスするためにデータ型の変換を必要としない DS2 変数を宣言できます。データ型の変換を回避する機能により、データベースまたは他のデータソースとの間でデータを効率的に移動できます。

注: 保存できるデータの型は、データソースがサポートするネイティブの型によって異なります。

詳細については、“DS2 データ型” (*SAS DS2 プログラマガイド*)を参照してください。

次の表は、DS2 データ型を選択する際に考慮すべき事項をまとめたものです。

表 4.1 DS2 データ型: クイックリファレンス

DS2 データ型カテゴリ	考慮事項
文字	CHAR(n)と VARCHAR(n)は、1 文字に付き 1 バイトを使用します。 Unicode 各国語文字セットを処理する NCHAR(n)と NVARCHAR(n)は、1 マルチバイト文字に付き 2 バイトまたは 4 バイトを使用します。 変数の長さ *n* を指定すると、その長さによって固定長変数のサイズまたは可変長変数の最大サイズが決まります。 **注:** 固定長 CHAR(n)は、DATA ステップ文字変数に相当します。ここでは、*n* は文字数です。また、宣言されていない DS2 文字変数のデフォルト型でもあります。
小数	DECIMAL(*p*,*s*) (別名: NUMERIC)は正確な精度を持ちます。 その他の小数値型には、DOUBLE、FLOAT (*p*)、REAL (単精度浮動小数点)があり、近似値と見なされます。 **注:** DOUBLE は、DATA ステップの数値変数に相当します。また、宣言されていない DS2 数値変数のデフォルト型でもあります。
整数	符号付きの正確な整数で、次のようにストレージサイズが異なります。 TINYINT (-128 から 127 まで) – 1 バイト SMALLINT (-32,768 から 32,767 まで) – 2 バイト INTEGER (-2,147,483,648 から 2,147,483,647 まで) – 4 バイト BIGINT (-9,223,372,036,854,775,808 から 9,223,372,036,854,775,807 まで) – 8 バイト
バイナリ	BINARY (*n*)は固定長です。 VARBINARY(*n*)は可変長です。
日時	DS2 は、ANSI の日付、時間、タイムスタンプのデータ型をサポートしますが、変換関数を使用して SAS 日付値、時間値、日時値を処理することもできます。 ■ TO_DATE 数値の SAS 日付を DS2 DATE に型変換します。 ■ TO_TIME 数値の SAS 時間を DS2 TIME に型変換します。 ■ TO_TIMESTAMP 数値の SAS 日時を DS2 TIMESTAMP に型変換します。 ■ TO_DOUBLE DS2 DATE、TIME、TIMESTAMP を数値の SAS 日付、時間、日時に型変換します。

データ型の自動変換

自動変換について

意図しない結果を回避するには、自動データ型変換の DS2 ルールを理解する必要があります。

注意
型変換は、データまたは精度、あるいはその両方の損失につながる可能性があります。 SAS データセットは 2 つのデータ型のみをサポートするため、データ型の変換は、DS2 データ型を SAS データセットに保存する場合に特に重要になります。つまり、DS2 変数は、固定長文字または倍精度数値のいずれかに自動的に変換される可能性があります。

自動型変換は、次の場合に発生します。

- 文字型が数値式で使用されている。
- 数値型が文字式で使用されている。
- メソッドの呼び出しから、メソッドのシグネチャと正確に一致しない引数値が提供されている。
- オペランドの型が、論理式、算術式、リレーショナル式、連結式で異なっている。
- DS2 データ型が、その型をサポートしていないデータソースに保存されている。

強制型変換と優先順位

DS2 では、強制型変換と優先順位のルールを使用して、変換後のデータ型を決定します。

- 強制型変換可能なデータ型は、複数のデータ型に自動的に変換できます。
- 強制型変換できないデータ型は、文字データ型のみに自動的に変換されます。
- 式に複数のデータ型が含まれている場合、優先順位によって変換の型が決まります。

詳細については、“DS2 型変換” (*SAS DS2 プログラマガイド*)を参照してください。

null 値と欠損値の変換

null 値と欠損値を処理するために使用するモードは、データに影響を与える可能性があります。null 値と欠損値を処理するためのデフォルトのモード(SAS モードまたは ANSI モード)は、DS2 プログラムをサブミットする環境と選択したオプションによって異なります。たとえば、デフォルトでは、DS2 プロシジャは SAS モードでデータを処理します。SAS Federation Server は、ANSI モードでデータを処理します。

注意
複数の変換中に、データの本来の意味を失う可能性があります。 特に、SAS 特殊欠損値(._, .A-.Z)の値で顕著です。

詳細については、"DS2 による null および SAS 欠損値の処理方法" (*SAS DS2 プログラマガイド*)を参照してください。

DS2 プログラミングブロックとスコープ

プログラミングブロックは、変数とコードをカプセル化する DS2 プログラムのセクションを定義します。プログラミングブロックは、モジュール式の再利用可能なコードの作成を促進します。さらに、プログラミングブロックは、そのブロック内の識別子のスコープを定義します。DS2 では、スコープが異なる限り、変数が同じ名前とデータ型を持つことが可能です。

次の表は、DS2 プログラミングブロックとスコープの特性をまとめたものです。

表 4.2 DS2 プログラミングブロックとスコープ: クイックリファレンス

ブロック	区切り文字	スコープ
データプログラム	DATA...ENDDATA	このプログラミングブロックの先頭で宣言されている変数は、データプログラム内でグローバルスコープを持っています。さらに、SET ステートメントが参照する変数はグローバルスコープを持っています。 明示的に削除されない限り、データプログラムのグローバル変数はプログラムデータベクトル(PDV)に含まれます。 **注:** グローバル変数は、データプログラムの期間中、存在します。
パッケージ	PACKAGE...ENDPACKAGE	このプログラミングブロックの先頭で宣言されている変数は、パッケージ内でグローバルスコープを持っています。パッケージスコープ変数は、パッケージのインスタンスを使用しているデータプログラムの PDV には含まれていません。

ブロック	区切り文字	スコープ
		注: パッケージスコープのグローバル変数は、パッケージインスタンスの期間中、存在します。
スレッドプログラム	THREAD...ENDTHREAD	このプログラミングブロックの先頭で宣言されている変数は、スレッドプログラム内でグローバルスコープを持っています。さらに、SET ステートメントが参照する変数はグローバルスコープを持っています。 明示的に削除されない限り、スレッドプログラムのグローバル変数はスレッド出力セットに含まれます。 **注:** スレッドスコープのグローバル変数は、スレッドプログラムインスタンスの期間中存在しますが、データプログラムの SET FROM ステートメントに渡すことができます。
メソッド	METHOD...END	メソッドは、データプログラム、パッケージ、またはスレッドプログラムのサブブロックです。メソッド名は、メソッドを含むブロック内でグローバルスコープを持っています。 このプログラミングブロックの先頭で宣言された変数は、ローカルスコープを持っています。ローカル変数は、PDV に含まれません。 **注:** ローカル変数は、メソッド呼び出しの期間中、存在します。
DO ループ	DO...END	適用できません。

ヒント　メソッド名はグローバルスコープを持っていますが、メソッドを前方参照するには、外側のプログラミングブロックの上部で FORWARD ステートメントを使用します。詳細については、“FORWARD ステートメント” (1358 ページ)を参照してください。

前の表に示すように、グローバルスコープには次の 3 種類があります。

- データプログラム
- パッケージ
- スレッドプログラム

DS2 での変数宣言

変数の暗黙の宣言

SAS DATA ステッププログラミングと同様に、DS2 を使用すると、割り当てによってグローバル変数を暗黙的に作成できます。ただし、次の理由から、これはお勧めしません。

- わかりにくいエラーが発生する可能性があります(たとえば、変数名のつづりが間違っている場合など)。
- このような変数のデータ型は、DOUBLE と CHAR(*n*)に限定されます。
- 宣言されていない変数は、他の人があなたのプログラムを読んで理解することを難しくするかもしれません。

DECLARE ステートメントを使用した変数宣言

SAS DATA ステッププログラミングとは異なり、DS2 では、DECLARE ステートメントを使用して変数を明示的に宣言できます。単一の DECLARE ステートメントで変数のすべての属性を制御できます。

DECLARE ステートメントは次の形式を取ります。

DECLARE [PRIVATE] data-type *variable-list* [HAVING *having-clause*];

注: DECLARE ステートメントは、使用されるプログラミングブロックの先頭でのみ許可されます。そうしない場合、コンパイルエラーが発生します。

詳細については、“DECLARE ステートメント” (1313 ページ)を参照してください。

変数宣言の strict モード

ベストプラクティスは、常に変数宣言の strict モードを使用して、DS2 プログラムを実行することです。これにより、すべてのプログラム変数の明示的な宣言が強制されます。

変数宣言の strict モードの制御の詳細については、“DS2SCOND= システムオプション” (1454 ページ)を参照してください。

DECLARE ステートメントとスコープ

DECLARE ステートメントを使用して変数を定義すると、変数に、変数が宣言されているプログラミングブロックのスコープが設定されます。メソッドは、別のプログラミングブロックのサブブロックです。したがって、メソッドで宣言された変数はローカルスコープを持ち、メソッドの実行時にのみ存在します。メソッドの外部で宣言された変数は、そのプログラミングブロック内にグローバルスコープを持っています。

スコープを示すサンプルプログラムについては、“例: ブロックスコープ” (45 ページ)を参照してください。

パッケージインスタンスの宣言

パッケージインスタンスは変数の一種ですが、2 つの要素から構成され、説明が必要な特殊なものです。

パッケージ変数
: パッケージの種類の参照をデータ型とする変数。

パッケージインスタンス
: パッケージの種類のインスタンス。理想的には、パッケージインスタンスは常に少なくとも 1 つのパッケージ変数によって参照されます。

> ヒント　パッケージ変数のスコープは、宣言されているプログラミングブロックによって決定されます。パッケージインスタンスには関連付けられたスコープがなく、その存続期間は参照するパッケージ変数に依存します。

詳細については、“パッケージ、スコープ、および有効期間” (*SAS DS2 プログラマガイド*)を参照してください。

DS2 メソッドとパッケージ

メソッドとパッケージについては、5 章, “DS2 メソッドとパッケージの理解” (51 ページ)を参照してください。

DS2 での並列処理

DS2 は、テーブルのさまざまな部分を操作できる単一のプログラムの並列実行をサポートします。この種類の並列処理は、SPMD (Single Program, Multiple Data)並列

処理として分類されます。DS2 では、プログラマによって並行操作可能なログラムステートメントが識別されます。

> ヒント　CPU バウンドプログラムの場合、対称型マルチプロセッシング(SMP)ハードウェアでスレッドプログラムを使用すると、パフォーマンスを向上させることができます。CPU または I/O バウンドのプログラムの場合、超並列処理(MPP)ハードウェアによってパフォーマンスを向上させることができます。

詳細については、“スレッド処理” (*SAS DS2 プログラマガイド*)を参照してください。

スレッドの簡単な使用法を示すサンプルプログラムについては、“例: 単純なスレッドプログラム” (47 ページ)を参照してください。

DS2 プログラムについて

DS2 の入門ガイドの用途に合わせて、DS2 プログラムは、DS2 プロシジャで実行される DS2 ステートメントから構成されています。入門ガイドのサンプルプログラムは、DS2 プログラムが次の用途に使用できることを示していますが、これらの用途に限定されるものではありません。

- 1 つ以上のパッケージまたはスレッドを定義し、永続的または一時的な場所に保存します。
- 永続的または一時的な場所に 1 つ以上のデータセットまたはテーブルを作成します。
- 前述の構成要素のいずれか、すべて、またはいずれも使用せずに、1 つ以上のデータプログラムを実行します。
- 前述の任意の組み合わせ。つまり、データ、パッケージ、スレッドを作成し、さらに単一の DS2 プログラム内で 1 つ以上のデータプログラムを実行できます。

プログラムがコンパイルされ、プログラムを実行するのに十分な数の RUN ステートメントが含まれている限り、DS2 プログラムのプログラミングブロックの順序と数は重要ではありません。

> ヒント　常にログでコンパイルエラーを確認してください。

さらに、SAS セッションでは、DS2 プロシジャと SAS DATA ステップを交互に実行して、コードをテストし、プログラミングの目標を達成することができます。

例: ブロックスコープ

次のプログラムは、スコープがプログラム識別子の表示を決定する方法を示しています。

注意事項

- このプログラムには、i という名前の 6 つの INTEGER 変数があります。
- このプログラムには、SHOWME( )という名前の 3 つのユーザー定義メソッドがあります。

```
options ds2scond=error;
proc ds2;
/* INNERPKG */
package innerPkg /overwrite=yes;
 dcl int i;   /* i is global in this package */
 dcl varchar(100) str;

 /* init() - initializes package variables */
 method init();
  i = 5;     /* global i */
  str = 'I am INNERPKG!';
 end;

 /* showMe() - displays values that INNERPKG can "see" */
 method showMe() returns int;
  dcl int i; /* local i */
  i = 10;    /* local i */
  put str;
  put 'Local i=' i;
  put 'Global i=' this.i;
  return 1;
 end;
endpackage;
run;

/* OUTERPKG */
package outerPkg /overwrite=yes;
 dcl int i; /* i is global in this package */
 dcl package innerPkg ip();
 dcl varchar(100) str;

 /* init() - initializes package variables */
 method init();
  i = 15;     /* global i */
  str = 'I am OUTERPKG!';
  ip.init();    /* tell INNERPKG to initialize itself */
```

```
 end;

 /* showMe() - displays values that OUTERPKG can "see" */
 method showMe();
  dcl int i; /* local i */
  i = 20; /* local i */
  put str;
  put 'Local i=' i; /* local i */
  put 'Global i=' this.i; /* global i */
  ip.showMe(); /* tell INNERPKG to show what it can "see" */
 end;
endpackage;
run;

/* data program */
data _null_;
 dcl int i; /* i is global in this data program */
 dcl package outerPkg op(); /* instantiate with default constructor */
 dcl varchar(100) str;
 FORWARD showMe;

 /* init() - automatically runs as the first method in the program */
 method init();
  dcl int rc;
  i = 25;       /* global i */
  str = 'I am the data program!';
  op.init();    /* tell OUTERPKG to initialize itself */
  showMe();     /* show what the data program can "see" */
 end;

 /* showMe() - displays values that the data program can "see" */
 method showMe();
  dcl int i;    /* local i */
  i = 30;       /* local i */
  put str;
  put 'Local i=' i;      /* local i */
  put 'Global i=' this.i; /* global i */
  op.showMe();  /* tell OUTERPKG instance to show what it can "see" */
 end;
enddata;
run;
quit;
```

次のようにログに書き込まれます。

```
I am the data program!
Local i= 30
Global i= 25
I am OUTERPKG!
Local i= 20
Global i= 15
I am INNERPKG!
Local i= 10
Global i= 5
```

例: 単純なスレッドプログラム

次のプログラムは、スレッドがデータを作成し、変数をデータプログラムに渡す方法を示しています。

注意事項

- パラメーター化されたスレッドは、データプログラムが SETPARMS( )システムメソッドを使用して初期化する必要がある値を受け入れます。
- OVERWRITE=YES テーブルオプションを使用すると、スレッドプログラムを上書きできます。

  注: THREAD ステートメントの構文には、'/' (スラッシュ文字)構文が必要です。
- データプログラムは 2 つのスレッドを指定しているため、スレッドプログラムは単一のプロセスで 2 つのスレッドで実行されます。

  注: このスレッドプログラムは、スレッドごとに 1 セットの出力変数を生成します。
- スレッドは非同期で実行されるため、ログに示されているように、処理の順序は予測できません。
- データプログラムでは、グローバルアキュムレータ変数 TOTAL は、total+answer; Sum ステートメントの構文により、暗黙的に保持されます。

```
options DS2SCOND=ERROR;
proc ds2;
/* thread program - Creates data in a loop */
thread work.t (double d) /overwrite=yes;
 dcl int x;
 dcl double y;

 method init();
  dcl int i; /* local - not included in the output table */

  do i = 1 to 9;
   x = i;
   y = i * 2.5 + d;
   put 'THREAD: i=' i ' x= ' x ' y= ' y;
   output; /* output variables include X and Y */
  end;
 end;

 method term();
  put 'THREAD TERM (_ALL_):';
  put _all_;
 end;
endthread;
```

```
run;

/* data program - Reads data from a thread program */
data;
 dcl thread work.t t;
 dcl double answer total;

 method init();
  t.setparms(1.25); /* initialize parameter of thread */
  put 'INIT (_ALL_):';
  put _all_;
 end;

 method run();
  set from t threads=2; /* input variables include X and Y */
  answer = x + y;
  total+answer; /* Sum statement syntax implicitly retains TOTAL */
  put ' x= ' x ' y= ' y ' answer= ' answer ' total= ' total;
 end;

 method term();
  put 'TERM: (_ALL_)';
  put _all_;
 end;
enddata;
run;
quit;
```

次のようにログに書き込まれます。

```
INIT (_ALL_):
total=0 answer=. x= y=. _N_=1
THREAD: i= 1  x=  1  y=  3.75
THREAD: i= 1  x=  1  y=  3.75
THREAD: i= 2  x=  2  y=  6.25
THREAD: i= 3  x=  3  y=  8.75
THREAD: i= 4  x=  4  y=  11.25
THREAD: i= 5  x=  5  y=  13.75
THREAD: i= 6  x=  6  y=  16.25
THREAD: i= 7  x=  7  y=  18.75
THREAD: i= 8  x=  8  y=  21.25
THREAD: i= 2  x=  2  y=  6.25
THREAD: i= 9  x=  9  y=  23.75
THREAD: i= 3  x=  3  y=  8.75
THREAD TERM (_ALL_):
THREAD: i= 4  x=  4  y=  11.25
d=1.25 x= y=. _N_=1
THREAD: i= 5  x=  5  y=  13.75
THREAD: i= 6  x=  6  y=  16.25
 x=  1  y=  3.75  answer=  4.75  total=  4.75
THREAD: i= 7  x=  7  y=  18.75
THREAD: i= 8  x=  8  y=  21.25
THREAD: i= 9  x=  9  y=  23.75
THREAD TERM (_ALL_):
d=1.25 x= y=. _N_=1
 x=  2  y=  6.25  answer=  8.25  total=  13
 x=  3  y=  8.75  answer=  11.75  total=  24.75
 x=  4  y=  11.25  answer=  15.25  total=  40
 x=  5  y=  13.75  answer=  18.75  total=  58.75
 x=  6  y=  16.25  answer=  22.25  total=  81
 x=  7  y=  18.75  answer=  25.75  total=  106.75
 x=  8  y=  21.25  answer=  29.25  total=  136
 x=  9  y=  23.75  answer=  32.75  total=  168.75
 x=  1  y=  3.75  answer=  4.75  total=  173.5
 x=  2  y=  6.25  answer=  8.25  total=  181.75
 x=  3  y=  8.75  answer=  11.75  total=  193.5
 x=  4  y=  11.25  answer=  15.25  total=  208.75
 x=  5  y=  13.75  answer=  18.75  total=  227.5
 x=  6  y=  16.25  answer=  22.25  total=  249.75
 x=  7  y=  18.75  answer=  25.75  total=  275.5
 x=  8  y=  21.25  answer=  29.25  total=  304.75
 x=  9  y=  23.75  answer=  32.75  total=  337.5
TERM: (_ALL_)
total=337.5 answer=. x=9 y=23.75 _N_=19
```

5

DS2 メソッドとパッケージの理解

DS2 でのモジュール化、カプセル化、抽象化 **51**

DS2 メソッドについて **52**
DS2 メソッドについて 52
DS2 システムメソッドについて 52
DS2 ユーザー定義メソッドについて 54

DS2 パッケージについて **55**
DS2 パッケージについて 55
DS2 事前定義パッケージについて 58

例: DS2 メソッドの概要 **58**

例: DS2 パッケージの概要 **59**

DS2 でのモジュール化、カプセル化、抽象化

DS2 のメソッドとパッケージを使用する場合、DATA ステップでプログラミングする場合とは異なる方法で SAS プログラムを作成します。これらの DS2 言語コンストラクトは、より構造化されたプログラミングとオブジェクト指向のアプローチに従っています。

一般に、DS2 メソッドは、関数、プロシジャ、サブルーチン、Java などのオブジェクト指向言語のメソッドに似ています。*メソッド*は、特定のタスクを実行するための一連の命令を含むモジュールと考えることができます。DS2 メソッドは、データプログラム、スレッドプログラム、またはパッケージの中にのみ存在できます。

このように、メソッドを使用すると、複雑な問題をより小さなモジュールに分割できます。このようなモジュールは、設計、実装、テストが簡単です。コードの再利用は、開発時間を短縮し、頻繁に繰り返されるプログラミングタスクやビジネス固有のプログラミングタスクを標準化するのに役立ちます。また、モジュール化されたプログラミングは、テスターや他のプログラマによる読みやすさや理解しやすさを向上させます。

DS2 パッケージは、動作のカプセル化と抽象化を可能にします。DS2 パッケージは、オブジェクト指向言語のクラスに似ています。しかし、DS2 パッケージは、必要であれば、有用であるが関連性のないメソッドと変数のバケットとして使用することもできます。

DS2 パッケージは、データとメソッドを名前付きオブジェクトにバンドルし、他の DS2 プログラムで保存および再利用できるようにします。このようなパッケージでは、オブジェクト内で定義されたメソッドにより、パッケージデータの制御された操作が可能になります。パッケージとそのコンポーネントはデフォルトでパブリックですが、PRIVATE アクセス修飾子を定義してパッケージのすべてまたは 一部を他のプログラムから隠すことができます。

DS2 メソッドについて

DS2 メソッドについて

実行可能なコードはメソッドにのみ存在できるため、メソッドは DS2 プログラムの構造上の構成ブロックです。

メソッドには、次の 2 つの種類があります。

システムメソッド
: 事前定義メソッドとも呼ばれるこれらのメソッドは、プログラムを実行するための構造的で機能的なフレームワークを提供します。

ユーザー定義のメソッド
: 他の言語の関数、プロシジャ、サブルーチンと同様に、自分が作成したメソッド、または他の誰かが再利用のために作成したメソッドです。

すべてのメソッドは、他のプログラミングブロックのサブブロックです。各メソッドは、ローカル変数が定義されるメソッドスコープを作成します。

次の例に示すように、メソッドプログラミングブロックは METHOD キーワードで始まり、END キーワードで終わります。

```
method foo();
...DS2 variables and statements...
end;
```

詳細については、“メソッド” (*SAS DS2 プログラマガイド*)を参照してください。

DS2 システムメソッドについて

システムメソッドは、コードを実行する構造的なフレームワークを提供します。つまり、プログラムを作成するには、INIT( )、RUN( )、TERM( )のシステムメソッドの 1 つ以上に DS2 ステートメントを追加するだけです。

システムメソッドは、暗黙的なループ処理など、SAS プログラマがデータやスレッドプログラムに必要とする特別な機能も提供します。さらに、システムメソッドにより、コードのセマンティックグループ化が可能になり、DS2 プログラムの編成、理解、保守が容易になります。

システムメソッドに関する基本的な事項は次のとおりです。

- すべての DS2 プログラムには、3 つの主要なシステムメソッドすべてが含まれ、実行されます。システムメソッドを明示的にコーディングしない場合、デフォルトバージョンが実行されます。
- システムメソッドのいずれか、またはすべてを明示的にコーディングできます。あるいは、すべてを明示的にコーディングしないこともできます。

  注: システムメソッドを含まないプログラムは構文的には正しいですが、有用な機能を実行しません。

- システムメソッドのシグネチャを変更したり、システムメソッドをオーバーロードしたりすることはできません。
- スレッドプログラムにのみ適用される SETPARMS( )を除いて、システムメソッドを直接呼び出すことはできません。

詳細については、“システムメソッドの概要” (1443 ページ)を参照してください。

次の表は、各 DS2 システムメソッドの実行の詳細をまとめたものです。

表 5.1 DS2 システムメソッド: 実行の詳細

システムメソッド	実行の詳細
INIT( )	プログラムの最初のメソッドとして、1 回自動的に実行されます。 詳細については、“INIT メソッド” (1445 ページ)を参照してください。
RUN( )	INIT( )の完了後に自動的に実行されます。RUN( )メソッドは DATA ステップと機能的に同等であり、メソッドに SET または SET FROM ステートメントが含まれている場合は暗黙的なループとして実行されます。 詳細については、“RUN メソッド” (1446 ページ)を参照してください。
TERM( )	プログラムの最後のメソッドとして、1 回自動的に実行されます。 詳細については、“TERM メソッド” (1450 ページ)を参照してください。
SETPARMS( )	データプログラムから呼び出されたときに 1 回実行されて、パラメーター化されたスレッドの値を初期化します。 詳細については、“SETPARMS メソッド” (1448 ページ)を参照してください。

DS2 ユーザー定義メソッドについて

他の言語の関数、プロシジャ、サブルーチンと同様に、自分が作成したメソッド、または他の誰かが再利用のために作成したメソッドです。メソッドは他のプログラミングブロックのサブブロックであるため、ユーザー定義メソッドはデータプログラム、パッケージ、スレッドプログラムに存在できます。

次の表は、ユーザー定義メソッドの基本概念をまとめたものです。

表 5.2　DS2 ユーザー定義メソッド: 基本概念

ユーザー定義メソッドのトピック	説明
スコープ	ユーザー定義メソッドの名前は、定義されているプログラミングブロック内でグローバルスコープを保有します。さらに、各メソッドは、ローカル変数が定義されるメソッドスコープを作成します。 **注:** ユーザー定義メソッドは、そのメソッドがスコープ内にあればどこからでも呼び出すことができ、何度でも呼び出すことができます。
メソッドパラメーター	ユーザー定義メソッドは、次の方法で引数を受け入れることができます。 ■ 値を使用します。引数値がメソッドにコピーされます。 ■ 参照(IN_OUT パラメーター)を使用します。このメソッドは、引数変数の値を変更します。 **ヒント**　DS2 メソッドでは、数千もの引数をサポートできます。DS2 メソッドでサポートされる引数の数は、引数のデータ型と DS2 プログラムが実行されるシステムによって異なります。サポートされている引数の数を超える DS2 メソッドは、コンパイルエラーを生成します。
戻り値	関数と同様に、ユーザー定義メソッドは値を返すことができます。

メソッドのオーバーロード	同じ名前を持つユーザー定義メソッドは、引数のシグネチャが重複しない場合、同じスコープに存在できます。つまり、戻り型のみが異なる場合、オーバーロードを識別できません。
	次の例は、3 つの重複しない引数のシグネチャを使用してオーバーロードされたメソッドを示しています。
	`method squareIt(int value) returns int;`
	`  return value**2;`
	`end;`
	
	`method squareIt(decimal(6,2) value) returns decimal(8,4);`
	`  return value**2;`

ユーザー定義メソッドのトピック	説明

	end;
	
	method squareIt(int value, IN_OUT int square);
	  square = value**2;
	end;

呼び出し	自動的に実行されるシステムメソッドとは異なり、ユーザー定義メソッドを呼び出す必要があります。ユーザー定義メソッドは、必要に応じて何度でも呼び出すことができます。

詳細については、“METHOD ステートメント” (1376 ページ)を参照してください。

DS2 パッケージについて

DS2 パッケージについて

DS2 パッケージは言語コンストラクトであり、変数とメソッドを名前付きオブジェクトにバンドルし、他の DS2 プログラムで保存および再利用できるようにします。パッケージの主な利点は、一連の有用なメソッドの再利用性にあります。しかし、DS2 パッケージはそれ以上の機能を備えています。

DS2 パッケージは、オブジェクト指向言語のクラスに似ています。したがって、オブジェクト指向クラスのカプセル化と抽象化に類似したパッケージを作成できます。

注: DS2 パッケージはプログラムではありません。DS2 パッケージは、プログラムで使用できるオブジェクトをインスタンス化するためのテンプレートです。

パッケージには、次の 2 種類があります。

事前定義パッケージ
: これらのパッケージは、ハッシュや行列データ構造の操作など、多くの顧客ソリューションに役立つ共通機能をカプセル化します。事前定義パッケージは、DS2 の一部です。

ユーザー定義パッケージ
: 自分が作ったパッケージや、他の誰かが再利用のために作成したパッケージです。

パッケージは、パッケージプログラミングブロックによって定義されます。次の例に示すように、パッケージは PACKAGE キーワードで始まり、ENDPACKAGE キーワードで終わります。

```
package foo;
  ...DS2 variables, statements, and methods...
endpackage;
```

次の表は、ユーザー定義パッケージと事前定義パッケージの両方の基本概念をまとめたものです。

表 5.3 DS2 パッケージ: 基本概念

パッケージのトピック	説明
スコープ	すべてのパッケージは独自のスコープを定義します。パッケージの本体内で宣言されたメソッドとグローバル変数の名前は、パッケージのスコープ内にあります。パッケージのメソッドと属性には、同じパッケージスコープ内の他のメソッドから他の修飾を必要とせずに名前によってアクセスできます。 パッケージのスコープ外のメソッドでは、ドット表記を使用してのみパッケージのメソッドと属性にアクセスできます。

パッケージメソッド	パッケージメソッドを実行するには、プログラムはパッケージインスタンスを使用して呼び出す必要があります。パッケージインスタンスは、ドット表記を使用して明示的に指定することも、前のパッケージメソッド呼び出しの暗黙的な THIS 引数にすることもできます。次に例を示します。
	`proc ds2;`
	`package mypackage / inline;`
	`  method a(int x);`
	`   put 'method called with' x=;`
	`  end;`
	
	`  method b(int x);`
	`   a(x);  /* 1 */`
	`  end;`
	`endpackage;`
	
	`data _null_;`
	`  method init();`
	`   declare package mypackage p();`
	`   p.b(20);  /* 2 */`
	`  end;`
	`enddata;`
	`run;`
	`quit;`
	1 メソッド b は、メソッド b が呼び出されたときと同じ暗黙的な THIS 引数を使用してメソッド a を呼び出します。
	2 メソッド INIT は、パッケージ変数 p によって参照されるパッケージインスタンスを使用してメソッド b を呼び出します。
	ネストされたドット表記がサポートされているため、パッケージ属性にアクセスし、パッケージの階層内からパッケージメソッドを呼び出すことができます。

使用するパッケージの準備	パッケージは、プログラムで使用する前に定義する必要があります。DS2 プロシジャを使用して、ユーザー定義パッケージを定義して保存できます。 詳細については、“PACKAGE ステートメント” (1392 ページ)を参照してください。

パッケージのトピック	説明

パッケージの上書き	DATA ステップとは異なり、DS2 は既存のテーブルが上書きされないように保護します。しかし、パッケージを開発したり変更したりする場合は、パッケージを上書きする機能が必要です。次の例に示すように、既存のパッケージを上書きするには、OVERWRITE=YES テーブルオプションを使用します。
	`package greeting /overwrite=yes;`
	`  ...DS2 variables, statements, and methods...`
	`endpackage;`
	`run;`

パッケージのインスタンス化	パッケージを使用するには、パッケージのインスタンス(パッケージインスタンス)と、インスタンスを参照する変数(パッケージ変数)を作成します。パッケージは、次の 2 つの方法でインスタンス化できます。
	■ DECLARE PACKAGE ステートメントは、1 つ以上のパッケージ変数を同時に宣言できます。宣言されたパッケージ変数ごとに、オプションで、DECLARE PACKAGE ステートメントではパッケージインスタンスを作成できます。次に例を示します。
	`dcl package matrix m0 m1(3, 3) m2(5, 4);`
	前述のステートメントは、3 つの行列パッケージ変数(m0、m1、m2)を宣言し、2 つの行列パッケージインスタンスを作成します(そのうちの 1 つは m1 に割り当てられ、もう 1 つは m2 に割り当てられます)。
	■ DECLARE PACKAGE ステートメントを使用してパッケージ変数を宣言し(コンストラクター引数を省略)、_NEW_演算子を使用してパッケージインスタンスを割り当てることができます。
	`data _null_;`
	`  dcl package matrix m1 m2;`
	`  method init();`
	`  dcl package matrix m3;`
	`  m1 = _NEW_ matrix(3, 2);`
	`  m3 = _NEW_ matrix(5, 4);`
	`  m2 = _NEW_ matrix();`
	`  end;`
	`  ...DS2 methods...`
	`enddata;`
	`run;`
	これらのステートメントは、パッケージ行列の 3 つのパッケージ変数も作成し、インスタンス化します。ただし、_NEW_演算子はパッケージインスタンスを構築し、パッケージ変数に割り当てます。
	注: パッケージ変数の宣言時に同時にパッケージのインスタンスを宣言しない場合、パッケージ変数の値は NULL になります。
	注: DS2 は配列パッケージ変数をサポートしています。配列パッケージ変数を使用すると、単一の変数から複数のパッケージインスタンスを参照できます。
	詳細については、DECLARE PACKAGE ステートメントを参照してください。

詳細については、“DS2 パッケージ” (*SAS DS2 プログラマガイド*)を参照してください。

DS2 事前定義パッケージについて

事前定義パッケージは、多くの顧客ソリューションに役立つ共通機能をカプセル化します。

DS2 で使用できる事前定義パッケージのリストについては、“事前定義された DS2 パッケージ” (*SAS DS2 プログラマガイド*)を参照してください。

付録 3, “DS2 サンプルプログラム” (2127 ページ)には、事前定義パッケージを使用する多くのサンプルプログラムが記載されています。

例: DS2 メソッドの概要

次のプログラムには、システムとユーザー定義の DS2 メソッドの両方の多くの特性が使用されています。

注意事項

- このプログラムには、SQUAREIT( )という名前の 3 つのオーバーロードされたユーザー定義メソッドがあります。
- このプログラムには、3 つのシステムメソッドが含まれています。

```
options DS2SCOND=ERROR;
proc ds2;
/* data program */
data _null_;
 dcl int root1 result1;
 dcl decimal(6,2) root2;
 dcl decimal(8,4) result2;

 /* overloaded user-defined method */
 method squareIt(int value) returns int;
  return value**2;
 end;

 /* overloaded user-defined method */
 method squareIt(decimal(6,2) value) returns decimal(8,4);
  return value**2;
 end;

 /* overloaded user-defined method with IN_OUT parameter */
 method squareIt(int value, IN_OUT int square);
```

```
  square = value**2;
 end;

 /* system method */
 method init();
  root1 = 3.01;
  root2 = 3.01;
 end;

 /* system method */
 method run();
  result1 = squareIt(root1);
  put 'The square of INTEGER ' root1 ' is ' result1;

  result2 = squareIt(root2);
  put 'The square of DECIMAL ' root2 ' is ' result2;

  root1 = 4.99;
  squareIt(root1, result1);
  put 'The square of INTEGER ' root1 ' is ' result1;
 end;

 /* system method */
 method term();
  put _all_; /* final values of global variables */
 end;
enddata;
run; quit;
```

次のようにログに書き込まれます。

```
The square of INTEGER  3  is  9
The square of DECIMAL  3.01  is  9.0601
The square of INTEGER  4  is  16
root1= root2= result1= result2= _N_=1
```

例: DS2 パッケージの概要

次のプログラムは、パッケージの基本概念を示しています。これは、"Hello World!" プログラムの 1 つのバージョンで、SAS ログにメッセージを書き込むパッケージを使用しています。

注意事項

- PACKAGE ステートメントは OVERWRITE=YES テーブルオプションを使用するため、エラーなしでプログラムを複数回実行できます。デフォルトでは、DS2 は既存のパッケージが上書きされないように保護します。

注意事項

- データプログラムでは、DECLARE PACKAGE ステートメントがパッケージ変数を宣言すると同時に、パッケージコンストラクターを使用してパッケージの 2 つの異なるインスタンスを作成します。
- ドット表記は、データプログラムからパッケージメソッドとパッケージグローバル変数へのアクセスを提供します。

  注: このサンプルプログラムは、パッケージグローバル変数に直接アクセスできることを示していますが、正当な理由がない限り、この方法を避けることをお勧めします。このベストプラクティスは、オブジェクト指向プログラミングで一般的なプログラミング手法である、データのカプセル化です。

```
options DS2SCOND=ERROR;
proc ds2;
/* GREETING - User-defined package that writes a message to the SAS log */
package greeting /overwrite=yes;
 dcl varchar(100) message;    /* Global scope package attribute */

 /* greeting(MESSAGE) - constructor that accepts an argument */
 method greeting(varchar(100) message);
  THIS.message = message;
 end;

 /* greeting() - default constructor */
 method greeting();
 end;

 /* package method */
 method greet();
  if not missing(message) then
    put message;
  else put 'Message is null or missing.';
 end;
endpackage;
run;

/* data program */
data _null_;
 /* declares and instantiates two instances of the GREETING package */
 dcl package greeting g1('Hello World!') g2();  /* data program (global) scope */

 /* init() - system method */
 method init();
  g1.greet();
  g2.greet();
  g2.message = 'Good-bye World!';
  g2.greet();
 end;
enddata;
run;
quit;
```

次のようにログに書き込まれます。

```
Hello World!
Message is null or missing.
Good-bye World!
```

3 部

DS2 言語リファレンス

6 章
DS2 出力形式 65

7 章
DS2 関数 235

8 章
DS2 入力形式 1285

9 章
DS2 演算子 1289

10 章
DS2 ステートメント 1293

11 章
DS2 システムメソッド 1443

12 章
DS2 システムオプション 1453

13 章
DS2 テーブルオプション 1457

14 章
DS2 データグリッドパッケージのメソッド、演算子、およびステートメント 1531

15 章
DS2 FCMP パッケージのメソッド、演算子、およびステートメント 1637

16 章
DS2 ハッシュおよびハッシュ反復子パッケージの属性、メソッド、演算子、ステートメント 1645

17 章
DS2 HTTP パッケージのメソッド、演算子、およびステートメント 1729

18 章
DS2 JSON パッケージのメソッド、演算子、およびステートメント 1771

19 章
DS2 ロガーパッケージのメソッド、演算子、およびステートメント 1807

20 章
DS2 行列パッケージのメソッド、演算子、およびステートメント 1817

21 章
DS2 PCRXFIND パッケージのメソッド、演算子、およびステートメント . 1893

22 章
DS2 PCRXREPLACE パッケージのメソッド、演算子、およびステートメント 1911

23 章
DS2 RedisCli パッケージのメソッド、演算子、およびステートメント ... 1919

24 章
DS2 SQLSTMT パッケージのメソッド、演算子、およびステートメント .. 1943

25 章
DS2 TZ パッケージのメソッド、演算子、およびステートメント 2021

6

DS2 出力形式

出力形式の概要 **67**
一般的な出力形式構文 **67**
DS2 での出力形式の使用 **68**
出力形式の指定方法 68
DS2 でフォーマットされたデータを書き込む 68
DS2 出力形式の検証 **70**
DS2 出力形式の例 **70**
出力形式カテゴリ **70**
ディクショナリ **77**
$BASE64Xw. 出力形式 77
$BINARYw. 出力形式 78
$CHARw. 出力形式 80
$HEXw. 出力形式 81
$OCTALw. 出力形式 82
$QUOTEw. 出力形式 83
$REVERJw. 出力形式 85
$REVERSw. 出力形式 86
$UPCASEw. 出力形式 87
$UUIDw. 出力形式 88
$w. 出力形式 89
B8601DAw. 出力形式 91
B8601DNw. 出力形式 92
B8601DTw.d 出力形式 94
B8601DZw. 出力形式 95
B8601LZw. 出力形式 98
B8601TMw.d 出力形式 99
B8601TZw. 出力形式 101
BESTw. 出力形式 103
BESTDOTXw. 出力形式 105
BESTDw.p 出力形式 107
BINARYw. 出力形式 109
COMMAw.d 出力形式 110
COMMAXw.d 出力形式 112
Dw.p 出力形式 113

DATEw. 出力形式 115
DATEAMPMw.d 出力形式 117
DATETIMEw.d 出力形式 119
DAYw. 出力形式 121
DDMMYYw. 出力形式 123
DDMMYYxw. 出力形式 124
DOLLARw.d 出力形式 127
DOLLARXw.d 出力形式 128
DOWNAMEw. 出力形式 130
DTDATEw. 出力形式 131
DTMONYYw. 出力形式 133
DTWKDATXw. 出力形式 135
DTYEARw. 出力形式 137
DTYYQCw. 出力形式 138
E8601DAw. 出力形式 139
E8601DNw. 出力形式 141
E8601DTw.d 出力形式 142
E8601DZw. 出力形式 144
E8601LZw. 出力形式 146
E8601TMw.d 出力形式 149
E8601TZw.d 出力形式 150
Ew. 出力形式 153
EUROw.d 出力形式 154
EUROXw.d 出力形式 156
FLOATw.d 出力形式 157
FRACTw. 出力形式 158
HEXw. 出力形式 160
HHMMw.d 出力形式 161
HOURw.d 出力形式 164
IEEEw.d 出力形式 165
JULIANw. 出力形式 167
MDYAMPMw.d 出力形式 169
MMDDYYw. 出力形式 171
MMDDYYxw. 出力形式 173
MMSSw.d 出力形式 175
MMYYw. 出力形式 176
MMYYxw. 出力形式 178
MONNAMEw. 出力形式 180
MONTHw. 出力形式 181
MONYYw. 出力形式 182
NEGPARENw.d 出力形式 184
NENGOw. 出力形式 186
OCTALw. 出力形式 187
PERCENTw.d 出力形式 188
PERCENTNw.d 出力形式 190
QTRw. 出力形式 192
QTRRw. 出力形式 193
ROMANw. 出力形式 194
TIMEw.d 出力形式 195
TIMEAMPMw.d 出力形式 198
TODw.d 出力形式 200
VAXRBw.d 出力形式 202
w.d 出力形式 203
WEEKDATEw. 出力形式 205

WEEKDATXw. 出力形式 . . . 207
WEEKDAYw. 出力形式 . . . 209
YEARw. 出力形式 . . . 210
YENw.d 出力形式 . . . 212
YYMMw. 出力形式 . . . 213
YYMMxw. 出力形式 . . . 215
YYMMDDw. 出力形式 . . . 217
YYMMDDxw. 出力形式 . . . 219
YYMONw. 出力形式 . . . 221
YYQw. 出力形式 . . . 223
YYQxw. 出力形式 . . . 225
YYQRw. 出力形式 . . . 227
YYQRxw. 出力形式 . . . 229
YYQZw. 出力形式 . . . 231
Zw.d 出力形式 . . . 232

出力形式の概要

出力形式は、DS2 がデータ値を書き込むために使用する命令です。出力形式は、書き込まれるデータ値の表示を制御するために使用されます。たとえば、数値をローマ数字に変換する ROMAN*w*.出力形式は、数値 2006 を **MMVI** として書き込みます。出力形式を使用して、分析のためにデータ値をグループ化することもできます。

一般的な出力形式構文

DS2 出力形式は次のような形式になります。

[$] *format* [*w*] . [*d*]

構文の説明は次のとおりです。

$
文字出力形式であることを表します。数値出力形式には使用されません。

format
出力形式名です。出力形式には、DS2 出力形式、または PROC FORMAT の INVALUE ステートメントを使用して事前に定義されたユーザー定義の出力形式があります。ユーザー定義の出力形式の詳細については、*Base SAS プロシジャガイド*.

制限事項
ユーザー定義の出力形式を作成してアクセスするには、SAS 出力形式の定義を保存する SAS カタログファイルにアクセスするために、SAS セッションが使用可能である必要があります。

w
出力形式幅を指定します。ほとんどの出力形式では、入力データの文字数です。

d
オプションの小数点以下の桁数です(数値出力形式)。DS2 は、入力データを 10 の *d* 乗で除算します。

ヒント *d* の値が 15 より大きい場合、小数点以下 15 桁以降の桁の小数値が正確でない場合があります。

出力形式は、名前の一部として必ずピリオド(.)を含みます。出力形式の *w* 値と *d* 値を省略すると、DS2 ではデフォルト値が使用されます。出力形式で指定する *d* 値は、データの小数点以下の桁数に関係なく、その小数点以下の桁数を表示するように DS2 に指示します。出力形式は、内部的に保存されているデータ値は変更または切り捨てるものではありません。

たとえば、DOLLAR10.2 では、*w* 値に 10 が指定されているため、値の最大文字数は 10 となります。*d* 値には 2 が指定されているため、10 文字のうち 2 文字が値の小数部分に使用されます。値の残りのすべての文字には 8 文字が使用されます。これには、小数点、小数以外の数値、マイナス記号(値が負の場合)、ドル記号、およびカンマが必要に応じて含まれます。

値を表すには出力形式の幅が狭すぎる場合、DS2 では、値は利用可能なスペースに合うように圧縮されます。文字出力形式では、値の右側が切り捨てられます。数値出力形式では、BEST*w.d* 出力形式が使用される場合があります。十分な幅を指定していない場合、DS2 では、アスタリスクが出力されます。次の例では、結果は x=** となります。

```
x=123;
put x= 2.;
```

数値出力形式を使用して文字値を書き出すなど、矛盾する出力形式を使用した場合、DS2 では、まず他の種類の類似した出力形式が試されます。これが実行不能な場合は、問題を説明するエラーメッセージが SAS ログに表示されます。

DS2 での出力形式の使用

出力形式の指定方法

DS2 では、DECLARE ステートメントの HAVING 句の属性として出力形式を指定します。HAVING 句は、DATA ステップの FORMAT および LABEL ステートメントと同等の機能を提供しますが、変数の宣言ステートメントで属性を指定する必要がある点が異なります。

たとえば、次のステートメントでは、*profit* と *loss* の列が EURO13.2 出力形式で宣言されています。

```
dcl double profit loss having format euro13.2;
```

注: DS2 では、HAVING 句で列に定義されている出力形式を変更したり削除したりすることはできません。ただし、PUT 関数の引数として出力形式を使用して、別の出力形式でデータを書き込むことができます。詳細については、“DS2 でフォーマットされたデータを書き込む” (69 ページ)を参照してください。

DS2 でフォーマットされたデータを書き込む

PUT 関数の引数として出力形式を使用して、フォーマットされたデータを書き込むことができます。出力形式の幅または小数点以下の桁数の指定に値が指定されていない場合、DS2 はその出力形式のデフォルト値を使用します。$OCTAL や$HEX などの入力幅に対応した出力形式を使用する場合、幅、小数点以下の桁数の指定、またはその両方が入力フィールドの幅から計算されることに注意してください。たとえば、$HEX は幅の 2 倍を使用し、$OCTAL は幅の 3 倍を使用します。

たとえば、この場合、PUT 関数は BEST12.出力形式を使用してフォーマット値 99 を返します。

```
x=99;
y=put(x, best12.);
```

PUT 関数を出力形式なしで使用すると、すべてのデータがフォーマットなしで書き込まれます。

フォーマットされる値の型変換はすべて、出力形式名に基づいて行われます。数値出力形式で PUT 関数に渡される値はすべて、必要に応じて DOUBLE に変換されます。文字出力形式で PUT 関数に渡される値はすべて、NCHAR に変換されます。詳細については、“PUT 関数” (1038 ページ)を参照してください。

DS2 は、次のように SAS 出力形式をサポートします。

- SAS が提供する出力形式とユーザー定義の出力形式を使用できます。SAS で独自の出力形式を作成する方法については、*Base SAS プロシジャガイド*の PROC FORMAT を参照してください。

  注: ユーザー定義の出力形式を作成してアクセスするには、SAS 出力形式の定義を保存する SAS カタログファイルにアクセスするために、SAS セッションが使用可能である必要があります。

- SAS データセット、SPD Engine データセット、および SPD Server テーブルのみが、列を含む出力形式の保存と取得をサポートします。
- 出力形式はすべてのデータ型に関連付けることができますが、すべてのデータ型は CHAR または DOUBLE のいずれかに変換されます。
- DS2 DECLARE ステートメントの HAVING 句を使用して、SAS 出力形式を列に関連付けます。詳細については、“出力形式の指定方法” (68 ページ)を参照してください。

DS2 出力形式の検証

出力形式は、実行時までデータソースによって 検証されたり、列に適用されたりすることはありません。

列のメタデータが要求されると、検証なしで出力形式名が返されます。

DS2 出力形式の例

```
declare datetime shipdate having format dateampm22.2;
x = put (name, $char8.);
x = put (price, myformat. -c);
```

出力形式カテゴリ

出力形式は、操作する値の種類によって分類できます。各 DS2 出力形式は、次のいずれかのカテゴリに属します。

表 6.1 出力形式カテゴリの説明

カテゴリ	説明
文字	文字変数から文字データ値を書き出します。 出力形式のリストについては、文字を参照してください。
日付と時間	文字変数から文字データ値を書き出します。 出力形式のリストについては、日付と時間を参照してください。
数値	数値変数から数値データ値を書き出します。 出力形式のリストについては、数値を参照してください。

次の表に、DS2 出力形式の簡単な説明を示します。詳細については、各出力形式を参照してください。

カテゴリ	言語要素	説明
ISO 8601	B8601DAw. 出力形式 (p. 91)	ISO 8601 規格の基本表記 yyyymmdd を使用して日付値を書き出します。
	B8601DNw. 出力形式 (p. 92)	ISO 8601 規格の基本表記 yyyymmdd を使用して、日時値から日付を書き出します。
	B8601DTw.d 出力形式 (p. 94)	ISO 8601 規格の基本表記 yyyymmddThhmmss<ffffff>を使用して日時値を書き出します。
	B8601DZw. 出力形式 (p. 95)	ISO 8601 規格の日時とタイムゾーンの基本表記 yyyymmddThhmmss+0000 を使用して、基準子午線の協定世界時(UTC)の日時値を書き出します。
	B8601LZw. 出力形式 (p. 98)	ISO 8601 規格の基本表記 hhmmss+|–hhmm を使用して、ローカル時間と UTC 間のタイムゾーンオフセット差を追加することにより、時間値をローカル時間として書き出します。
	B8601TMw.d 出力形式 (p. 99)	ISO 8601 規格の基本表記 hhmmss<ffff>を使用して、時間値を書き出します。
	B8601TZw. 出力形式 (p. 101)	ISO 8601 規格の基本時間表記 hhmmss+|–hhmm を使用して、時間値を協定世界時(UTC)に調整し、時間値を書き出します。
	E8601DAw. 出力形式 (p. 139)	ISO 8601 規格の拡張表記 yyyy-mm-dd を使用して日付値を書き出します。
	E8601DNw. 出力形式 (p. 141)	ISO 8601 規格の拡張表記 yyyy-mm-dd を使用して、SAS 日時値から日付を書き出します。
	E8601DTw.d 出力形式 (p. 142)	ISO 8601 規格の拡張表記 yyyy-mm-ddThh:mm:ss.ffffff を使用して日時値を書き出します。
	E8601DZw. 出力形式 (p. 144)	ISO 8601 規格の日時とタイムゾーンの拡張表記 yyyy-mm-ddThh:mm:ss+00:00 を使用して、基準子午線の協定世界時(UTC)の日時値を書き出します。
	E8601LZw. 出力形式 (p. 146)	ISO 8601 規格の拡張時間表記 hh:mm:ss+|–hh:mm を使用して、ローカル SAS セッションに対応した協定世界時(UTC)のオフセットを追加することにより、時間値をローカル時間として書き出します。
	E8601TMw.d 出力形式 (p. 149)	ISO 8601 規格の拡張表記 hh:mm:ss.ffffff を使用して時間値を書き出します。
	E8601TZw.d 出力形式 (p. 150)	時間値を協定世界時(UTC)に調整し、時間値を ISO 8601 規格の拡張表記 hh:mm:ss.<fff>+|–hh:mm を使用して書き出します。
数値	BESTw. 出力形式 (p. 103)	SAS により最適な出力が選択されます。
	BESTDOTXw. 出力形式 (p. 105)	SAS が最適な表記を選び、小数点としてドット(.)を使うように指定します。

カテゴリ	言語要素	説明
	BESTDw.p 出力形式 (p. 107)	同じような大きさの数値に対しては小数点以下の桁数を揃えて出力します。整数は小数なしで出力します。
	BINARYw. 出力形式 (p. 109)	数値を 2 進表現に変換します。
	COMMAw.d 出力形式 (p. 110)	3 桁ごとにカンマを入れ、小数部分との区切りにはピリオドを使用して数値を出力します。
	COMMAXw.d 出力形式 (p. 112)	3 桁ごとにピリオドを入れ、小数部分との区切りにはカンマを使用して数値を出力します。
	Dw.p 出力形式 (p. 113)	値の桁数をできる限り最大にし、同じような大きさの数値に対しては小数点以下の桁数を揃えて出力します。
	DOLLARw.d 出力形式 (p. 127)	先頭にドル($)記号を付け、3 桁ごとにカンマを入れ、小数部分との区切りにはピリオドを使用して数値を出力します。
	DOLLARXw.d 出力形式 (p. 128)	先頭にドル($)記号を付け、3 桁ごとにピリオドを入れ、小数部分との区切りにはカンマを使用して数値を出力します。
	Ew. 出力形式 (p. 153)	数値を指数表記で書き出します。
	EUROw.d 出力形式 (p. 154)	先頭にユーロ記号(E)を付け、3 桁ごとにカンマを入れ、小数部分との区切りにはピリオドを使用して数値を出力します。
	EUROXw.d 出力形式 (p. 156)	先頭にユーロ記号(E)を付け、3 桁ごとにピリオドを入れ、小数部分との区切りにはカンマを使用して数値を出力します。
	FLOATw.d 出力形式 (p. 157)	数値に 10 の d 乗を掛けた、単精度のネイティブ浮動小数点値を生成します。
	FRACTw. 出力形式 (p. 158)	数値を分数に変換します。
	HEXw. 出力形式 (p. 160)	バイナリ実数(浮動小数点)値を 16 進表現に変換します。
	IEEEw.d 出力形式 (p. 165)	数値に 10 の d 乗を掛けて、IEEE 浮動小数点値を生成します。
	NEGPARENw.d 出力形式 (p. 184)	負の数値をかっこで囲んで書き出します。
	OCTALw. 出力形式 (p. 187)	数値を 8 進表現に変換します。
	PERCENTw.d 出力形式 (p. 188)	数値を百分率として書き出します。
	PERCENTNw.d 出力形式 (p. 190)	百分率を書き出します。負の値にはマイナス符号を使用します。
	ROMANw. 出力形式 (p. 194)	数値をローマ数字で書き出します。
	VAXRBw.d 出力形式 (p. 202)	VMS 形式のバイナリ実数(浮動小数点)を書き出します。
	w.d 出力形式 (p. 203)	標準数値でデータを書き出します(1 バイト 1 桁)。

カテゴリ	言語要素	説明
	YENw.d 出力形式 (p. 212)	円記号、カンマ、小数点を使用して数値を書き込みます。
	Zw.d 出力形式 (p. 232)	先頭に 0 を含む標準数値データを書き出します。
日付と時間	B8601DAw. 出力形式 (p. 91)	ISO 8601 規格の基本表記 yyyymmdd を使用して日付値を書き出します。
	B8601DNw. 出力形式 (p. 92)	ISO 8601 規格の基本表記 yyyymmdd を使用して、日時値から日付を書き出します。
	B8601DTw.d 出力形式 (p. 94)	ISO 8601 規格の基本表記 yyyymmddThhmmss<ffffff>を使用して日時値を書き出します。
	B8601DZw. 出力形式 (p. 95)	ISO 8601 規格の日時とタイムゾーンの基本表記 yyyymmddThhmmss+0000 を使用して、基準子午線の協定世界時(UTC)の日時値を書き出します。
	B8601LZw. 出力形式 (p. 98)	ISO 8601 規格の基本表記 hhmmss+|–hhmm を使用して、ローカル時間と UTC 間のタイムゾーンオフセット差を追加することにより、時間値をローカル時間として書き出します。
	B8601TMw.d 出力形式 (p. 99)	ISO 8601 規格の基本表記 hhmmss<ffff>を使用して、時間値を書き出します。
	B8601TZw. 出力形式 (p. 101)	ISO 8601 規格の基本時間表記 hhmmss+|–hhmm を使用して、時間値を協定世界時(UTC)に調整し、時間値を書き出します。
	DATEw. 出力形式 (p. 115)	SAS 日付値を ddmmmyy、ddmmmyyyy または dd-mmm-yyyy 形式で書き出します。
	DATEAMPMw.d 出力形式 (p. 117)	SAS 日時値を、AM または PM を付けて ddmmmyy:hh:mm:ss.ss 形式で書き出します。
	DATETIMEw.d 出力形式 (p. 119)	SAS 日時値を ddmmmyy:hh:mm:ss.ss 形式で書き出します。
	DAYw. 出力形式 (p. 121)	SAS 日付値の日付部分を書き出します。
	DDMMYYw. 出力形式 (p. 123)	SAS 日付値を ddmm<yy>yy または dd/mm/<yy>yy 形式で書き出します。スラッシュが区切り文字として使用され、年は 2 桁または 4 桁の数字で書き出されます。
	DDMMYYxw. 出力形式 (p. 124)	SAS 日付値を ddmm<yy>yy または dd-mm-yy<yy>形式で書き出します。出力形式名の x は、日、月、年を区切る特殊文字を表す文字で、ハイフン(-)、ピリオド(.)、空白文字、スラッシュ(/)、コロン(:)、または区切り文字なしになります。年は 2 桁または 4 桁のいずれかになります。
	DOWNAMEw. 出力形式 (p. 130)	SAS 日付値から曜日名を書き出します。
	DTDATEw. 出力形式 (p. 131)	入力として SAS 日時値が必要です。SAS 日付値を ddmmmyy または ddmmmyyyy 形式で書き出します。

カテゴリ	言語要素	説明
	DTMONYYw. 出力形式 (p. 133)	SAS 日時値の日付部分を月と年として mmmyy または mmmyyyy 形式で書き出します。
	DTWKDATXw. 出力形式 (p. 135)	SAS 日時値の日付部分を曜日と日付として day-of-week、dd month-name yy (または yyyy)形式で書き出します。
	DTYEARw. 出力形式 (p. 137)	SAS 日時値の日付部分を年として yy または yyyy 形式で書き出します。
	DTYYQCw. 出力形式 (p. 138)	SAS 日時値から、年と四半期をコロン(:)で区切って書き出します。
	E8601DAw. 出力形式 (p. 139)	ISO 8601 規格の拡張表記 yyyy-mm-dd を使用して日付値を書き出します。
	E8601DNw. 出力形式 (p. 141)	ISO 8601 規格の拡張表記 yyyy-mm-dd を使用して、SAS 日時値から日付を書き出します。
	E8601DTw.d 出力形式 (p. 142)	ISO 8601 規格の拡張表記 yyyy-mm-ddThh:mm:ss.ffffff を使用して日時値を書き出します。
	E8601DZw. 出力形式 (p. 144)	ISO 8601 規格の日時とタイムゾーンの拡張表記 yyyy-mm-ddThh:mm:ss+00:00 を使用して、基準子午線の協定世界時(UTC)の日時値を書き出します。
	E8601LZw. 出力形式 (p. 146)	ISO 8601 規格の拡張時間表記 hh:mm:ss+|–hh:mm を使用して、ローカル SAS セッションに対応した協定世界時(UTC)のオフセットを追加することにより、時間値をローカル時間として書き出します。
	E8601TMw.d 出力形式 (p. 149)	ISO 8601 規格の拡張表記 hh:mm:ss.ffffff を使用して時間値を書き出します。
	E8601TZw.d 出力形式 (p. 150)	時間値を協定世界時(UTC)に調整し、時間値を ISO 8601 規格の拡張表記 hh:mm:ss.<fff>+|–hh:mm を使用して書き出します。
	HHMMw.d 出力形式 (p. 161)	SAS 時間値を時間および分として hh:mm 形式で書き出します。
	HOURw.d 出力形式 (p. 164)	SAS 時間値を時間および時間の小数として書き出します。
	JULIANw. 出力形式 (p. 167)	SAS 日付値を yyddd または yyyyddd 形式のユリウス日付として書き出します。
	MDYAMPMw.d 出力形式 (p. 169)	SAS 日時値を mm/dd/yy<yy> hh:mm AM|PM 形式で書き出します。年は 2 桁または 4 桁のいずれかです。
	MMDDYYw. 出力形式 (p. 171)	SAS 日付値を mmdd<yy>yy または mm/dd/<yy>yy 形式で書き出します。スラッシュが区切り文字として使用され、年は 2 桁または 4 桁の数字で書き出されます。
	MMDDYYxw. 出力形式 (p. 173)	SAS 日付値を mmdd<yy>yy または mm-dd-<yy>yy 形式で書き出します。ここで、出力形式名の x は、月、日、年を区切る特殊文字を表す文字です。特殊文字は、ハイフン(-)、ピ

カテゴリ	言語要素	説明
		リオド(.)、空白文字、スラッシュ(/)、コロン(:)または区切り文字なしになります。年は 2 桁または 4 桁のいずれかになります。
	MMSSw.d 出力形式 (p. 175)	SAS 時間値を午前 0 時から経過した分数および秒数として書き出します。
	MMYYw. 出力形式 (p. 176)	SAS 日付値を mmM<yy>yy 形式で書き出します。M は区切り文字を表し、年は 2 桁または 4 桁の数字で書き出されます。
	MMYYxw. 出力形式 (p. 178)	SAS 日付値を mm<yy>yy または mm-<yy>yy 形式で書き出します。出力形式名の x は、月と年を区切る特殊文字を表す文字で、ハイフン(-)、ピリオド(.)、空白文字、スラッシュ(/)、コロン(:)、または区切り文字なしになります。年は 2 桁または 4 桁のいずれかになります。
	MONNAMEw. 出力形式 (p. 180)	SAS 日付値を月の名前として書き出します。
	MONTHw. 出力形式 (p. 181)	SAS 日付値を月として書き出します。
	MONYYw. 出力形式 (p. 182)	SAS 日付値を月および年として mmmyy または mmmyyyy 形式で書き出します。
	NENGOw. 出力形式 (p. 186)	SAS 日付値を e.yymmdd 形式で日本の日付として書き出します。
	QTRw. 出力形式 (p. 192)	SAS 日付値を年の四半期として書き出します。
	QTRRw. 出力形式 (p. 193)	SAS 日付値を年の四半期としてローマ数字で書き出します。
	TIMEw.d 出力形式 (p. 195)	SAS 時間値を時間、分、および秒として hh:mm:ss.ss 形式で書き出します。
	TIMEAMPMw.d 出力形式 (p. 198)	SAS 時間値を時間、分、および秒として hh:mm:ss.ss 形式で AM または PM を付けて書き出します。
	TODw.d 出力形式 (p. 200)	SAS 時間値と、SAS 日時値の時間部分を hh:mm:ss.ss 形式で書き出します。
	WEEKDATEw. 出力形式 (p. 205)	SAS 日付値を曜日と日付として day-of-week, month-name dd, yy (または yyyy)形式で書き出します。
	WEEKDATXw. 出力形式 (p. 207)	SAS 日付値を曜日と日付として day-of-week, dd month-name yy (または yyyy)形式で書き出します。
	WEEKDAYw. 出力形式 (p. 209)	SAS 日付値から曜日の値を書き出します。
	YEARw. 出力形式 (p. 210)	SAS 日付値から年部分を書き出します。
	YYMMw. 出力形式 (p. 213)	SAS 日付値を<yy>yyMmm 形式で書き出します。M は月番号が M の後に続くことを示す区切り文字で、年は 2 桁または 4 桁の数字で書き出されます。

カテゴリ	言語要素	説明
	YYMMxw. 出力形式 (p. 215)	SAS 日付値を<yy>yymm または<yy>yy-mm 形式で書き出します。出力形式名の x は、年と月を区切る特殊文字を表す文字で、ハイフン(-)、ピリオド(.)、空白文字、スラッシュ(/)、コロン(:)、または区切り文字なしになります。年は 2 桁または 4 桁の数字で書き出されます。
	YYMMDDw. 出力形式 (p. 217)	SAS 日付値を<yy>yymmdd または<yy>yy–mm–dd 形式で書き出します。ハイフンが区切り文字として使用され、年は 2 桁または 4 桁の数字で書き出されます。
	YYMMDDxw. 出力形式 (p. 219)	SAS 日付値を<yy>yymmdd または<yy>yy-mm-dd 形式で書き出します。出力形式名の x は、年、月、日を区切る特殊文字を表す文字です。特殊文字は、ハイフン(-)、ピリオド(.)、空白文字、スラッシュ(/)、コロン(:)または区切り文字なしになります。年は 2 桁または 4 桁のいずれかになります。
	YYMONw. 出力形式 (p. 221)	SAS 日付値を yymmm または yyyymmm 形式で書き出します。
	YYQw. 出力形式 (p. 223)	SAS 日付値を<yy>yyQq 形式で書き出します。Q は区切り文字として使用され、年は 2 桁または 4 桁の数字で書き出されます。また、q は四半期を表します。
	YYQxw. 出力形式 (p. 225)	SAS 日付値を<yy>yyq または<yy>yy-q 形式で書き出します。出力形式名の x は、年と四半期を区切る特殊文字を表す文字で、ハイフン(-)、ピリオド(.)、空白文字、スラッシュ(/)、コロン(:)、または区切り文字なしになります。年は 2 桁または 4 桁の数字で書き出されます。
	YYQRw. 出力形式 (p. 227)	SAS 日付値を<yy>yyQqr 形式で書き出します。Q は区切り文字として使用され、年は 2 桁または 4 桁の数字で書き出されます。また qr はローマ数字表現の四半期を表します。
	YYQRxw. 出力形式 (p. 229)	SAS 日付値を<yy>yyqr または<yy>yy-qr 形式で書き出します。出力形式名の x は、年と四半期を区切る特殊文字を表す文字で、ハイフン(-)、ピリオド(.)、空白文字、スラッシュ(/)、コロン(:)、または区切り文字なしになります。年は 2 桁または 4 桁の数字で書き出されます。また qr は、ローマ数字表現の四半期です。
	YYQZw. 出力形式 (p. 231)	SAS 日付値を<yy><qq>形式で書き出します。年は 2 桁または 4 桁の数字で書き出され、qq は四半期を表します。
文字	$BASE64Xw. 出力形式 (p. 77)	Base 64 エンコーディングを使用して、文字データを ASCII テキストに変換します。
	$BINARYw. 出力形式 (p. 78)	文字データを 2 進表現に変換します。
	$CHARw. 出力形式 (p. 80)	標準文字データを書き出します。
	$HEXw. 出力形式 (p. 81)	文字データを 16 進表現に変換します。
	$OCTALw. 出力形式 (p. 82)	文字データを 8 進表現に変換します。

カテゴリ	言語要素	説明
	$QUOTEw. 出力形式 (p. 83)	データ値を二重引用符で囲んで書き込みます。
	$REVERJw. 出力形式 (p. 85)	文字データの末尾から順に書き込みます。空白も保持されます。
	$REVERSw. 出力形式 (p. 86)	文字データの末尾から順に、左揃えで書き込みます。
	$UPCASEw. 出力形式 (p. 87)	文字データを大文字に変換します。
	$UUIDw. 出力形式 (p. 88)	ユニバーサル一意識別子(UUID)形式で文字データを書き込みます。
	$w. 出力形式 (p. 89)	標準文字データを書き出します。

ディクショナリ

$BASE64Xw. 出力形式

Base 64 エンコーディングを使用して、文字データを ASCII テキストに変換します。

カテゴリ: 文字

配置: 左

構文

$BASE64X*w*.

必須引数

w

出力フィールドの幅を指定します。

デフォルト 1

範囲 1-32767

注 幅なしで$BASE64X.出力形式を指定すると、デフォルトの幅 1 が使用され、出力を生成するには不十分です。結果は返されません。

詳細

Base 64 は、ポジション指定スキームに基づき、ASCII 文字のみ使用したエンコード文字を指定するエンコーディング方式です。複数の Base 64 エンコーディングスキームが、メールやコンテンツマスキングなどの特定の用途に定義されています。ポジション 0 - 61 を文字 A - Z、a - z、0 - 9 にマップします。ポジション 62 は文字+にマップし、ポジション 63 は文字/にマップします。

次に、Base 64 エンコーディングの使用例を示します。

- XML ファイルの埋め込みバイナリデータ
- パスワードのエンコード
- URL のエンコード

エンコード結果の'='文字は、結果にゼロビットが埋め込まれたことを示します。エンコード文字をデコードするには、'='をデコードする値に含める必要があります。

例

次のプログラムは、$BASE64X*w*.出力形式を示しています。

```
data _null_;
  declare varchar(30) a b c ;
  method run();
    a= put('FCA01A7993BC', $base64x64.);
    b= put('MyPassword', $base64x64.);
    c= put('www.mydomain.com/myhiddenURL', $base64x64.);
    put a= b= c=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=RkNBMDFBNzk5M0JD          b=TXlQYXNzd29yZA==          c=d3d3Lm15ZG9tYWluLmNvbS9teWhpZ
```

$BINARYw. 出力形式

文字データを 2 進表現に変換します。

カテゴリ: 文字

配置: 左

構文

$BINARY*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト	デフォルトの幅は、出力対象となる変数の長さに基づいて計算されます。
範囲	1-32767
ヒント	文字値はそれぞれ 8 つの 2 進表現の文字を生成するため、*w* の値には文字値の長さを 8 倍した値を指定します。

比較

$BINARY*w*.出力形式は、文字値を 2 進表現に変換します。$BINARY*w*.出力形式は、数値を 2 進表現に変換します。

例

次のプログラムは、$BINARY*w*.出力形式を示しています。

```
data _null_;
  declare varchar(30) a;
  method run();
    a = put ('name', $binary16.);
    put a=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=0110111001100001
```

関連項目

出力形式:

- “BINARYw. 出力形式” (109 ページ)

$CHARw. 出力形式

標準文字データを書き出します。

カテゴリ: 文字

配置: 左

別名: $*w*.と$F*w*.

構文

$CHAR*w*.

必須引数

w
出力フィールドの幅を指定します。数値または列範囲を指定できます。

デフォルト 変数の長さが未定義の場合は 8。それ以外の場合は、変数の長さ。

範囲 1-32767

比較

- $CHAR*w*.出力形式は、$*w*.および$F*w*.出力形式と同じです。
- $CHAR*w*.、$F*w*.、および$*w*.出力形式は先頭の空白を切り捨てません。先頭の空白を切り捨てるには、LEFT 関数を使用して、出力前に文字データを左揃えにします。

例

次のプログラムは、$CHAR*w*.出力形式を示しています。

```
data _null_;
  declare varchar(30) a;
  method run();
    a= put ('XYZ',$char.);
    put a=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=XYZ
```

$HEXw. 出力形式

文字データを 16 進表現に変換します。

カテゴリ: 文字

配置: 左

構文

$HEX*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト: デフォルトの幅は、出力対象となる変数の長さに基づいて計算されます。

範囲: 1-32767

注: *w* が変換したい変数の長さの 2 倍を超えている場合、$HEX*w*.により空白が埋め込まれます。

ヒント: 文字値はそれぞれ 2 つの 16 進表現の文字を生成するため、*w* の値には文字値の長さを 2 倍した値を指定します。

詳細

$HEX*w*.出力形式は、各文字を 2 つの 16 進文字に変換します。各空白は、末尾の空白を含めて 1 文字としてカウントされます。

比較

HEX*w*.出力形式は、バイナリ実数を 16 進数の表現に変換します。

例

次のプログラムは、$HEX*w*.出力形式を示しています。

```
data _null_;
  declare varchar(30) a;
  method run();
    a=put ('bank', $hex.);
    put a=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=62616E6B
```

関連項目

出力形式:

- “HEXw. 出力形式” (160 ページ)

$OCTALw. 出力形式

文字データを 8 進表現に変換します。

カテゴリ: 文字

配置: 左

構文

$OCTAL*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト: デフォルトの幅は、出力対象となる変数の長さに基づいて計算されます。

範囲: 1-32767

ヒント: 文字値はそれぞれ 3 つの 8 進表現の文字を生成するため、*w* の値には文字値の長さを 3 倍した値を指定します。

比較

$OCTALw.出力形式は、文字値をその文字コードの 8 進表現に変換します。
OCTALw.出力形式は、数値を 8 進表現に変換します。

例

次のプログラムは、$OCTALw.出力形式を示しています。

```
data _null_;
  declare varchar(30) a b c;
  method run();
    a=put('art', $octal9.);
    b=put('rice', $octal12.);
    c=put('bank', $octal12.);
    put a= b= c=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=141162164 b=162151143145 c=142141156153
```

関連項目

出力形式:

- “OCTALw. 出力形式” (187 ページ)

$QUOTEw. 出力形式

データ値を二重引用符で囲んで書き込みます。

カテゴリ: 文字

配置: 左

構文

$QUOTE*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト　変数の長さが未定義の場合は 2。それ以外の場合は、変数の長さ + 2。

範囲　2-32767

ヒント　*w* の幅を左と右の引用符を含められるよう十分な大きさにします。

詳細

次のリストは、$QUOTE*w*.出力形式を使用した場合に作成される出力です。

- データ値が引用符で囲まれていない場合、出力が二重引用符で囲まれます。
- データ値が引用符で囲まれていないが値に単一引用符が含まれている場合、次のアクションが実行されます。
  - データ値を二重引用符で囲む
  - 単一引用符を変更しない
- データ値が単一引用符で始まり終わっていて、値に二重引用符が含まれている場合、次のアクションが実行されます。
  - データ値を二重引用符で囲む
  - データ値に見つかった二重引用符を複製する
  - 単一引用符を変更しない
- データ値が単一引用符で始まり終わっていて、値に 2 つの隣接した単一引用符が含まれている場合、次のアクションが実行されます。
  - 値を二重引用符で囲む
  - 単一引用符を変更しない
- データ値が単一引用符で始まり終わっていて、二重引用符と隣接した単一引用符が含まれている場合、次のアクションが実行されます。
  - 値を二重引用符で囲む
  - データ値に見つかった二重引用符を複製する
  - 単一引用符を変更しない
- ターゲットフィールドの長さが文字列と引用符を含めるのに十分でない場合、そのフィールドの長さに合う長さの引用符付き文字列が返されます。たとえば、指定された値が ABCDE で、$QUOTE5.を指定した場合、値の DE は切り捨てられ、結果は"ABC"になります。

注: 埋め込まれた引用符は、追加の引用符で囲む必要があります。A"B の値を指定すると、$QUOTE5.は値の B を切り捨てて、"A""と書き込みます。

例

次のプログラムは、$QUOTE*w*.出力形式を示しています。

```
data _null_;
  declare varchar(30) a b c d e f;
  method run();
    a=put('SAS',$quote.);
    b=put('SAS''s',$quote.);
    c=put('ad''verb',$quote16.);
    d=put('"ad" "verb"',$quote16.);
    e=put('"ad""""verb"',$quote20.);
    f=put('deoxyribonucleotide',$quote20.);
    put a= b= c= d= e= f=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a="SAS" b="SAS's" c="ad'verb"       d="""ad"" ""verb"  e="""ad""""""verb""""
f="deoxyribonucleotid"
```

$REVERJw. 出力形式

文字データの末尾から順に書き込みます。空白も保持されます。

カテゴリ: 文字

配置: 右

構文

$REVERJ*w*.

必須引数

w

出力フィールドの幅を指定します。

デフォルト *w* が指定されていない場合は 1。

範囲 1-32767

比較

$REVERJ*w*.出力形式は、$REVERS*w*.出力形式とほぼ同じですが、$REVERS*w*.では先頭の空白がすべて削除され、結果を左揃えにします。

例

次のプログラムは、$REVERJ*w*.出力形式を示しています。

```
data _null_;
  declare varchar(30) a b;
  method run();
    a=put('ABCD###',$reverj7.);
    b=put('###ABCD',$reverj7.);
    put a= b=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=###DCBA b=DCBA###
```

関連項目

出力形式:

- “$REVERSw. 出力形式” (86 ページ)

$REVERSw. 出力形式

文字データの末尾から順に、左揃えで書き込みます。

カテゴリ: 文字

配置: 左

構文

$REVERS*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト　*w* が指定されていない場合は 1。

範囲　1-32767

比較

$REVERS*w*.出力形式は、$REVERJ*w*.出力形式とほぼ同じですが、$REVERJ*w*. では結果を左揃えにしません。

例

次のプログラムは、$REVERS*w*.出力形式を示しています。

```
data _null_;
  declare varchar(30) a b;
  method run();
    a=put('ABCD   ',$revers7.);
    b=put('   ABCD',$revers7.);
    put a= b=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=DCBA   b=DCBA
```

関連項目

出力形式:

- “$REVERJw. 出力形式” (85 ページ)

$UPCASEw. 出力形式

文字データを大文字に変換します。

カテゴリ:　文字

配置:　左

構文

$UPCASE*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト 変数の長さが未定義の場合は 8。それ以外の場合は、変数の長さ。

範囲 1-32767

詳細

ハイフンやその他の記号などの特殊文字は、変更されません。

例

次のプログラムは、$UPCASE*w*.出力形式を示しています。

```
data _null_;
  declare varchar(30) a;
  method run();
    a=put('coxe-ryan',$upcase9.);
    put a=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=COXE-RYAN
```

$UUIDw. 出力形式

ユニバーサル一意識別子(UUID)形式で文字データを書き込みます。

カテゴリ: 文字

配置: 左

構文

$UUID*w*.

必須引数

w
出力フィールドの幅を指定します。

範囲 1-200

詳細

必要な値を 16 進形式で指定する必要があります。それ以外の場合は、元の文字の 16 進表現が UUID として返されます。

注: 値を 16 進リテラルとして指定し、先頭に'x'を付ける場合、値は単一引用符でのみ囲むことができます。二重引用符は使用できません。

例

次のプログラムは、$UUID*w*.出力形式を示しています。

```
data _null_;
  declare char(36) x y;
  method run();
    x=put (x'1548611fb8cb6a47a721a6fd284e74f2', $uuid36.);
    put x=;
    x = inputc('1548611f-b8cb-6a47-a721-a6fd284e74f2','$uuid36.');
    y=put (x, $uuid36.);
    put y=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
x=1548611f-b8cb-6a47-a721-a6fd284e74f2
y=1548611f-b8cb-6a47-a721-a6fd284e74f2
```

$w. 出力形式

標準文字データを書き出します。

カテゴリ: 文字

配置: 左

別名: $F*w*.と$CHAR*w*.

構文

$*w*.

必須引数

w
出力フィールドの幅を指定します。数値または列範囲を指定できます。

デフォルト 変数の長さが未定義の場合は 1。それ以外の場合は、変数の長さ。

範囲 1-32767

比較

$*w*.、$F*w*.、および$CHAR*w*.出力形式は同じで、先頭の空白を切り捨てません。先頭の空白を切り捨てるには、LEFT 関数を使用して、出力前に文字データを左揃えにします。

例

次のプログラムは、$*w*.出力形式を示しています。

```
data _null_;
  declare char(15) a;
  method run();
    a=put(' Cary',$5.);
    put a=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a= Cary
```

関連項目

関数:

- “LEFT 関数” (806 ページ)

B8601DAw. 出力形式

ISO 8601 規格の基本表記 *yyyymmdd* を使用して日付値を書き出します。

カテゴリ: 日付と時間
ISO 8601

配置: 左

制限事項: UTC タイムゾーンオフセット値はサポートされていません。

サポート: ISO 8601 Element 5.2.1.1, complete representation

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

B8601DA*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト 10

範囲 8-10

詳細

B8601DA 出力形式は、ISO 8601 規格の基本表記 *yyyymmdd* を使用して日付値を書き出します。

yyyy
4 桁の年です。

mm
01 から 12 までの 2 桁の月です(ゼロ埋め込み)。

dd
0 から 31 までの 2 桁の日です(ゼロ埋め込み)。

例

次の例では、入力値として 20999 を使用します。20999 は、2017 年 6 月 29 日に対応する SAS 日付値です。

```
data _null_;
  declare varchar(15) a;
  method run();
    a=put(20999,b8601da.);
    put a=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=20170629
```

関連項目

"ISO 8601 規格の基本表記と拡張表記を使用した日付と時間の処理" (*SAS 出力形式と入力形式: リファレンス*)

B8601DNw. 出力形式

ISO 8601 規格の基本表記 *yyyymmdd* を使用して、日時値から日付を書き出します。

カテゴリ:	日付と時間 ISO 8601
配置:	左
制限事項:	UTC タイムゾーンオフセット値はサポートされていません。
サポート:	ISO 8601 Element 5.2.1.1, complete representation
注:	2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

B8601DN*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト 10

範囲 8-10

詳細

B8601DN 出力形式は、ISO 8601 規格の基本表記 *yyyymmdd* を使用して、日時値から日付を書き出します。

yyyy
4 桁の年です。

mm
01 から 12 までの 2 桁の月です(ゼロ埋め込み)。

dd
01 から 31 までの 2 桁の日です(ゼロ埋め込み)。

例

次の例では、入力値として 1793308532 を使用します。1793308532 は、2016 年 10 月 28 日に対応する SAS 日付値です。

```
data _null_;
  declare varchar(15) a;
  method run();
    a=put(1793308532,b8601dn.);
    put a=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a= 20161028
```

関連項目

"ISO 8601 規格の基本表記と拡張表記を使用した日付と時間の処理" (*SAS 出力形式と入力形式: リファレンス*)

B8601DTw.d 出力形式

ISO 8601 規格の基本表記 *yyyymmdd*T*hhmmss*<*fffffff*>を使用して日時値を書き出します。

カテゴリ: 日付と時間
ISO 8601

配置: 左

制限事項: UTC タイムゾーンオフセット値はサポートされていません。

サポート: ISO 8601 Element 5.4.1, complete representation

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

B8601DT*w.d*

必須引数

w
出力フィールドの幅を指定します。

デフォルト 19

範囲 15-26

d
秒の端数を表す秒値の小数点以下の桁数を指定します。この引数はオプションです。

デフォルト 0

範囲 0-6

詳細

B8601DT 出力形式は、ISO 8601 規格の基本表記 *yyyymmdd*T*hhmmss*<*fffffff*>を使用して、日時値を書き出します。

yyyy
4 桁の年です。

mm
01 から 12 までの 2 桁の月です(ゼロ埋め込み)。

dd
01 から 31 までの 2 桁の日です(ゼロ埋め込み)。

hh
00 から 23 までの 2 桁の時間です(ゼロ埋め込み)。

mm
00 から 59 までの 2 桁の分です(ゼロ埋め込み)。

ss
00 から 59 までの 2 桁の秒です(ゼロ埋め込み)。

ffffff
6 桁までの有効桁数を持つ、オプションの秒の分数です。各桁は 0 から 9 までです。

例

次の例では、入力値として 1793308532 を使用します。1793308532 は、2016 年 10 月 28 日の午後 9:15:32 に対応する SAS 日時値です。

```
data _null_;
  declare varchar(30) a;
  method run();
    a=put(1793308532, b8601dt.);
    put a=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a= 20161028T211532
```

関連項目

"ISO 8601 規格の基本表記と拡張表記を使用した日付と時間の処理" (*SAS 出力形式と入力形式: リファレンス*)

B8601DZw. 出力形式

ISO 8601 規格の日時とタイムゾーンの基本表記 *yyyymmdd*T*hhmmss*+0000 を使用して、基準子午線の協定世界時(UTC)の日時値を書き出します。

カテゴリ: 日付と時間
ISO 8601

配置: 左

サポート: ISO 8601 Element 5.4.1, complete representation

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

B8601DZ*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト 26

範囲 20-35

詳細

UTC 値は、イギリスのグリニッジの基準子午線に基づく時間とタイムゾーンを示します。B8601DZ 出力形式は、次の ISO 8601 規格の基本日時表記のいずれかを使用して、基準子午線の日時に対応した SAS 日時値を書き出します。

- *yyyymmdd*T*hhmmss*+0000

  注: この形式は、*w* の長さがこのタイムゾーン表記に十分な場合に使用します。

- *yyyymmdd*T*hhmmss*Z

  注: この形式は、*w* の長さが+0000 タイムゾーン表記に十分でない場合に使用します。

yyyy
4 桁の年です。

mm
01 から 12 までの 2 桁の月です(ゼロ埋め込み)。

dd
01 から 31 までの 2 桁の日です(ゼロ埋め込み)。

hh
00 から 23 までの 2 桁の時間です(ゼロ埋め込み)。

mm
00 から 59 までの 2 桁の分です(ゼロ埋め込み)。

ss
00 から 59 までの 2 桁の秒です(ゼロ埋め込み)。

+0000
基準子午線(イギリスのグリニッジ)の UTC 時間を示します。

タイムゾーンオフセットが指定された ISO 8601 規格の時間値または日時値は、オフセットに示された時間数と分数によって調整されます。その後、タイムゾーンオフセットが基準子午線(イギリスのグリニッジ)の時間または日時として処理されます。B8601DZ 出力形式は、常に基準子午線オフセット値の+0000 を使用して日時値を書き出します。+0000 以外のタイムゾーンオフセットを使用した日時を書き出すには、“B8601LZw. 出力形式” (98 ページ)を参照してください。

制約: 短い形式+00 はサポートされていません。

Z
時間が基準子午線(イギリスのグリニッジ)に対応した時間、つまり+0000UTC 時間であることを示します。出力形式の幅が+0000 表記をサポートしない場合に Z が使用されます。

例

次の例では、入力値として 1793308532 を使用します。1793308532 は、2016 年 10 月 28 日の午後 9:15:32 に対応する SAS 日付値です。

```
data _null_;
  declare varchar(30) a;
  method run();
    a=put(1793308532,b8601dz.);
   put a=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=    20161028T211532+0000
```

関連項目

“ISO 8601 規格の基本表記と拡張表記を使用した日付と時間の処理” (*SAS 出力形式と入力形式: リファレンス*)

B8601LZw. 出力形式

ISO 8601 規格の基本表記 *hhmmss+|–hhmm* を使用して、ローカル時間と UTC 間のタイムゾーンオフセット差を追加することにより、時間値をローカル時間として書き出します。

カテゴリ: 日付と時間
ISO 8601

配置: 左

サポート: ISO 8601 Elements 5.3.3 and 5.3.4.2

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

B8601LZ*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト 14

範囲 9-20

詳細

B8601LZ 出力形式は、ISO 8601 規格の基本表記 *hhmmss+|–hhmm* を使用して時間値を調整せずに書き出し、ローカル SAS セッションに対応した UTC タイムゾーンオフセットを追加します。

hh
00 から 23 までの 2 桁の時間です(ゼロ埋め込み)。

mm
00 から 59 までの 2 桁の分です(ゼロ埋め込み)。

ss
00 から 59 までの 2 桁の秒です(ゼロ埋め込み)。

+|–*hhmm*
基準子午線時間からの時間と分の符号付きオフセットです。オフセットは+|–*hhmm* (つまり+または–を伴う 4 文字)である必要があります。

+は基準子午線の東部タイムゾーンに、-は基準子午線の西部タイムゾーンにそれぞれ使用します。たとえば、+0200 は基準子午線の東部での 2 時間差を表し、-0600 は基準子午線の西部での 6 時間差を表します。

制約: 短い形式+|–*hh* はサポートされていません。

SAS では UTC 時間を B8601TZ 入力形式を使用して読み込む際に、オフセット調整後の時間が 24 時間を超えていたり 00 時間未満である場合、オフセット調整後の時間が 000000 と 235959 の間になるように値を調整します。B8601LZ 出力形式をこの時間範囲外の時間に適用する場合、時間はアスタリスクを使用してフォーマットされ、その値が範囲外であることが示されます。

例

次の例では、20999 を入力値として使用します。これは、ローカル時間値 054959-0000 です。

```
data _null_;
  declare varchar(30) a;
  method run();
    a=put(20999,b8601lz.);
    put a=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=  054959-0000
```

関連項目

"ISO 8601 規格の基本表記と拡張表記を使用した日付と時間の処理" (*SAS 出力形式と入力形式: リファレンス*)

B8601TMw.d 出力形式

ISO 8601 規格の基本表記 *hhmmss<ffff>*を使用して、時間値を書き出します。

カテゴリ: 日付と時間
ISO 8601

配置: 左

制限事項: UTC タイムゾーンオフセット値はサポートされていません。

サポート: ISO 8601 Element 5.3.1.1, complete representation

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

B8601TM*w.d*

必須引数

w
出力フィールドの幅を指定します。

デフォルト 8

範囲 6-15

d
秒の端数を表す秒値の小数点以下の桁数を指定します。この引数はオプションです。

デフォルト 0

範囲 0-6

詳細

B8601TM 出力形式は、ISO 8601 規格の基本時間表記 *hhmmss<fffff>*を使用して、SAS 時間値を書き出します。

hh
00 から 23 までの 2 桁の時間です(ゼロ埋め込み)。

mm
00 から 59 までの 2 桁の分です(ゼロ埋め込み)。

ss
00 から 59 までの 2 桁の秒です(ゼロ埋め込み)。

fffff
6 桁までの有効桁数を持つ、オプションの秒の分数です。各桁は 0 から 9 までです。

例

次の例では、054959 の時間値に対応する 20999 の入力値を使用しています。

```
data _null_;
  declare varchar(30) a;
  method run();
    a=put(20999,b8601tm.);
    put a=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=054959
```

関連項目

"ISO 8601 規格の基本表記と拡張表記を使用した日付と時間の処理" (*SAS 出力形式と入力形式: リファレンス*)

B8601TZw. 出力形式

ISO 8601 規格の基本時間表記 *hhmmss*+|–*hhmm* を使用して、時間値を協定世界時(UTC)に調整し、時間値を書き出します。

カテゴリ: 日付と時間
ISO 8601

配置: 左

サポート: ISO 8601 Elements 5.3.3 and 5.3.4

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

B8601TZ*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト 14

範囲 9-20

詳細

UTC 時間値は、イギリスのグリニッジの基準子午線に基づく時間とタイムゾーンを示します。B8601TZ 出力形式は、時間値を基準子午線の時間に調整し、その時間値を次の ISO 8601 規格の基本時間表記のいずれかで書き出します。

- *hhmmss*+|–*hhmm*

  注: この形式は、*w* の長さがこの時間表記に十分である場合に使用します。

- *hhmmss*Z

  注: この形式は、*w* の長さが+|–*hhmm* タイムゾーン表記に十分でない場合に使用します。

hh
00 から 23 までの 2 桁の時間です(ゼロ埋め込み)。

mm
00 から 59 までの 2 桁の分です(ゼロ埋め込み)。

ss
00 から 59 までの 2 桁の秒です(ゼロ埋め込み)。

+|–*hh:mm*
基準子午線時間からの時間と分の符号付きオフセットです。オフセットは+|–*hhmm* (つまり+または–を伴う 4 文字)である必要があります。

+は基準子午線の東部タイムゾーンに、-は基準子午線の西部タイムゾーンにそれぞれ使用します。たとえば、+0200 は基準子午線の東部での 2 時間差を表し、-0600 は基準子午線の西部での 6 時間差を表します。

制約: 短い形式+|–*hh* はサポートされていません。

Z
時間が基準子午線(イギリスのグリニッジ)に対応した時間、つまり+0000 UTC 時間であることを示します。出力形式の幅が+0000 表記をサポートしない場合に Z が使用されます。

SAS では UTC 時間を B8601TZ 入力形式を使用して読み込む際に、オフセット調整後の時間が 24 時間を超えていたり 00 時間未満である場合、オフセット調整後の時間が 000000 と 240000 の間になるように値を調整します。B8601TZ 出力形式をこの時間範囲外の時間に適用する場合、時間はアスタリスクを使用してフォーマットされ、その値が範囲外であることが示されます。

比較

- 000000 と 240000 の間の時間値に対して、B8601TZ 出力形式は時間値を基準子午線の時間に調整し、調整後の値を国際規格の拡張時間表記で書き出します。
- B8601LZ 出力形式は時間を調整せず、ローカル SAS セッションの UTC タイムゾーンオフセットを使用して時間値を国際規格の拡張時間表記で書き出します。

例

次の例では、054959+0000 の時間値に対応する 20999 の入力値を使用しています。

```
data _null_;
  declare varchar(30) a;
  method run();
    a=put(20999,b8601tz.);
    put a=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=  054959+0000
```

関連項目

“ISO 8601 規格の基本表記と拡張表記を使用した日付と時間の処理” (*SAS 出力形式と入力形式: リファレンス*)

BESTw. 出力形式

SAS により最適な出力が選択されます。

カテゴリ: 数値

配置: 右

操作: DECIMALCONV=システムオプションを STDIEEE に設定した場合、この出力形式を使用して書き出される出力が前のリリースと多少異なる可能性があります。詳細については、“DECIMALCONV= システムオプション” (*SAS システムオプション: リファレンス*)を参照してください。

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

BEST*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト 12

範囲 1-32

ヒント 0 から.01 までの数字を出力する場合、過度に丸めないように 7 以上のフィールド幅を使用します。0 から-.01 までの数字を出力する場合は、8 以上のフィールド幅を使用します。

詳細

BEST*w*.出力形式は、数値を書き込むためのデフォルトの出力形式です。出力形式の指定がない場合、SAS は、使用可能なフィールド幅に従って、値に関する最も多くの情報を提供する出力形式を選択します。BEST*w*.は値を丸め、SAS が指定された幅内で小数点以下の部分に少なくとも 1 つの有効数字を表示できる場合、BEST*w*.は結果を 10 進数で生成します。それ以外の場合は、指数表記で結果が生成されます。SAS では、表現に使用する出力形式にかかわらず、常に完全な値を保存します。

比較

- BEST*w*.出力形式は、出力フィールドに可能な限りの有効桁数を書き出しますが、数字の大きさが異なる場合に小数点以下の桁数を揃えません。整数は小数点なしで出力されます。
- BESTDOTX*w*.出力形式の場合、出力フィールド内にできるだけ多くの有効数字を収めようとします。整数は小数点付きで出力されます。
- D*w*.*p* 出力形式は数字を指定した有効桁数で書き出し、BEST*w*.出力形式よりも小数点揃えを行います。
- *w*.*d* 出力形式は小数点揃えを行いますが、すべての数字が必ずしも同じ有効桁数で表示されません。

例

次のプログラムは、BEST*w*.出力形式を示しています。

```
data _null_;
  declare varchar(30) a b;
  method run();
```

```
    a=put(1257000,best6.);
    b=put(1257000,best3.);
    put a= b=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=1.26E6 b=1E6
```

関連項目

出力形式:

- “BESTDw.p 出力形式” (107 ページ)
- “BESTDOTXw. 出力形式” (105 ページ)
- “Dw.p 出力形式” (113 ページ)
- “w.d 出力形式” (203 ページ)

BESTDOTXw. 出力形式

SAS が最適な表記を選び、小数点としてドット(.)を使うように指定します。

カテゴリ: 数値

配置: 右

操作: DECIMALCONV=システムオプションを STDIEEE に設定した場合、この出力形式を使用して書き出される出力が前のリリースと多少異なる可能性があります。詳細については、“DECIMALCONV= システムオプション” (*SAS システムオプション: リファレンス*)を参照してください。

構文

BESTDOTX.*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト 12

範囲 1-32

ヒント 0 から.01 までの数字を出力する場合、過度に丸めないように 7 以上のフィールド幅を使用します。0 から-.01 までの数字を出力する場合は、8 以上のフィールド幅を使用します。

詳細

NLDECSEPARATOR システムオプションが無効である場合、BEST*w.*と BESTDOTX*w.* はデータを同じように処理します。NLDECSEPARATOR システムオプションが有効であれば、BEST*w.*か BESTDOTX*w.*によって出力形式が異なります。違いについては次の表を参照してください。

LOCALE オプション	ロケールに応じたデフォルトの小数点文字	NLDECSEPARATOR オプション	BEST*w.*の場合の小数点文字	BESTDOTX*w.*の場合の小数点文字
en_US	ドット(.)	無効(デフォルト)	ドット(.)	ドット(.)
		有効	ドット(.)	ドット(.)
fr_FR	カンマ(,)	無効(デフォルト)	ドット(.)	ドット(.)
		有効	カンマ(,)	ドット(.)

NLDECSEPARATOR システムオプションの詳細については、*SAS 各国語サポート (NLS): リファレンスガイド*を参照してください。

比較

- BESTDOTX*w* 出力形式の場合、出力フィールド内にできるだけ多くの有効数字を収めようとします。整数は小数点付きで出力されます。
- BEST*w.*出力形式は、出力フィールドに可能な限りの有効桁数を書き出しますが、数字の大きさが異なる場合に小数点以下の桁数を揃えません。整数は小数点なしで出力されます。

関連項目

出力形式:

- “BESTw. 出力形式” (103 ページ)

BESTDw.p 出力形式

同じような大きさの数値に対しては小数点以下の桁数を揃えて出力します。整数は小数なしで出力します。

カテゴリ: 数値

配置: 右

操作: DECIMALCONV=システムオプションを STDIEEE に設定した場合、この出力形式を使用して書き出される出力が前のリリースと多少異なる可能性があります。詳細については、“DECIMALCONV= システムオプション” (*SAS システムオプション: リファレンス*)を参照してください。

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

BESTD*w*.[*p*]

必須引数

w
出力フィールドの幅を指定します。

デフォルト 12

範囲 1-32

オプション引数

p
有効桁数を指定します。

デフォルト 3

範囲 0 - *w*–1

要件 *w* より小さい数にします。

ヒント *p* が省略される、または 0 が指定されると、*p* は 3 に設定されます。

詳細

BESTD*w.p* 出力形式は、同じような大きさの値グループ内で小数点を揃えて値を書き出します。整数は小数点なしで出力されます。*p* の値が大きくなると、出力されるデータ値の有効桁数が増え、小数点揃えのシフトが増す可能性があります。*p* の値が小さくなると、出力されるデータ値の有効桁数が減り、小数点揃えの可能性が高くなります。

出力形式は、値の範囲に対して、値をより少ない小数点以下の桁数で表示できる場合でも、出力に適した小数点以下の桁数を選択します。

比較

- BEST*w*.出力形式は、出力フィールドに可能な限りの有効桁数を書き出しますが、数字の大きさが異なる場合に小数点以下の桁数を揃えません。整数は小数点なしで出力されます。
- BESTD*w.p* 出力形式は、BEST*w*.出力形式と D*w.p* 出力形式を組み合わせたもので、すべての数値データに適用されます。また、BEST*w*.出力形式よりも小数点揃えに優れています。
- D*w.p* 出力形式は数字を指定した有効桁数で書き出し、BEST*w* 出力形式よりも小数点揃えを行います。
- *w.d* 出力形式は小数点揃えを行いますが、すべての数字が必ずしも同じ有効桁数で表示されません。

例

次のプログラムは、BESTD*w.p* 出力形式を示しています。

```
data _null_;
  declare varchar(30) a b c d e;
  method run();
    a=put(12345, bestd14.);
    b=put(123.45, bestd14.);
    c=put(1.2345, bestd14.);
    d=put(.12345, bestd14.);
    e=put(1.23456789, bestd14.);
    put a= b= c= d= e=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=      12345 b=   123.4500000 c=    1.2345000 d=    0.1234500 e=    1.2345679
```

関連項目

出力形式:

- “BESTw. 出力形式” (103 ページ)
- “Dw.p 出力形式” (113 ページ)
- “w.d 出力形式” (203 ページ)

BINARYw. 出力形式

数値を 2 進表現に変換します。

カテゴリ: 数値

配置: 左

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

BINARY*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト 8

範囲 1-64

比較

BINARY*w*. は、数値を 2 進表現に変換します。$BINARY*w*.出力形式は、文字値を 2 進表現に変換します。

例

次のプログラムは、BINARY*w*.出力形式を示しています。

```
data _null_;
  declare varchar(30) a b c;
  method run();
    a = put (123.45, binary8.);
    b = put (123, binary8.);
    c = put (-123, binary8.);
    put a= b= c=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=01111011 b=01111011 c=10000101
```

関連項目

出力形式:

- “$BINARYw. 出力形式” (78 ページ)

COMMAw.d 出力形式

3 桁ごとにカンマを入れ、小数部分との区切りにはピリオドを使用して数値を出力します。

カテゴリ: 数値

配置: 右

操作: DECIMALCONV=システムオプションを STDIEEE に設定した場合、この出力形式を使用して書き出される出力が前のリリースと多少異なる可能性があります。詳細については、“DECIMALCONV= システムオプション” (*SAS システムオプション: リファレンス*)を参照してください。

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

COMMA*w*.[*d*]

必須引数

w
出力フィールドの幅を指定します。

デフォルト 6

範囲 1-32

ヒント *w* の幅に数値、カンマ、オプションの小数点の書き出しに十分な値を指定します。

オプション引数

d
数値の小数点以下の桁数を指定します。

範囲 0-31

要件 *w* より小さい数にします。

比較

- COMMA*w.d* 出力形式は COMMAX*w.d* 出力形式とほぼ同じですが、COMMAX*w.d* 出力形式では小数点とカンマの役割が逆になっています。この規則はヨーロッパの国で共通です。
- COMMA*w.d* 出力形式は DOLLAR*w.d* 出力形式とほぼ同じですが、COMMA*w.d* 出力形式は先頭のドル記号を出力しません。

例

次のプログラムは、COMMA*w*.出力形式を示しています。

```
data _null_;
  declare varchar(30) a b;
  method run();
    a=put (23451.23,comma10.2);
    b=put (123451.234,comma10.2);
    put a= b=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a= 23,451.23 b=123,451.23
```

関連項目

出力形式:

- “COMMAXw.d 出力形式” (112 ページ)
- “DOLLARw.d 出力形式” (127 ページ)

COMMAXw.d 出力形式

3 桁ごとにピリオドを入れ、小数部分との区切りにはカンマを使用して数値を出力します。

カテゴリ: 数値

配置: 右

操作: DECIMALCONV=システムオプションを STDIEEE に設定した場合、この出力形式を使用して書き出される出力が前のリリースと多少異なる可能性があります。詳細については、“DECIMALCONV= システムオプション” (*SAS システムオプション: リファレンス*)を参照してください。

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

COMMAX[*w*.*d*]

必須引数

w
出力フィールドの幅を指定します。

デフォルト 6

範囲 1-32

ヒント *w* の幅に数値、カンマ、オプションの小数点の書き出しに十分な値を指定します。

オプション引数

d
数値の小数点以下の桁数を指定します。

範囲 0-31

要件　*w* より小さい数にします。

比較

COMMA*w.d* 出力形式は COMMAX*w.d* 出力形式とほぼ同じですが、COMMAX*w.d* 出力形式では小数点とカンマの役割が逆になっています。この規則はヨーロッパの国で共通です。

例

次のプログラムは、COMMAX*w*.出力形式を示しています。

```
data _null_;
  declare varchar(15) a b;
  method run();
    a=put (23451.23,commax10.2);
    b=put (123451.234,commax10.2);
    put a= b=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a= 23.451,23 b=123.451,23
```

関連項目

出力形式:

- “COMMAw.d 出力形式” (110 ページ)

Dw.p 出力形式

値の桁数をできる限り最大にし、同じような大きさの数値に対しては小数点以下の桁数を揃えて出力します。

カテゴリ:　数値

配置:　右

操作:　DECIMALCONV=システムオプションを STDIEEE に設定した場合、この出力形式を使用して書き出される出力が前のリリースと多少異なる可能性があります。詳細については、“DECIMALCONV= システムオプション” (*SAS システムオプション: リファレンス*)を参照してください。

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

D[*w*.*p*]

必須引数

w
出力フィールドの幅を指定します。

デフォルト 12

範囲 1-32

p
有効桁数を指定します。

デフォルト 3

範囲 0-16

要件 *w* より小さい数にします。

詳細

D*w*.*p* 出力形式は、同じような大きさの値グループ内で小数点を揃えて値を書き出します。*p* の値が大きくなると、出力されるデータ値の有効桁数が増え、小数点揃えのシフトが増す可能性があります。*p* の値が小さくなると、出力されるデータ値の有効桁数が減り、小数点揃えの可能性が高くなります。

比較

- BEST*w*.出力形式は、出力フィールドに可能な限りの有効桁数を書き出しますが、数字の大きさが異なる場合に小数点以下の桁数を揃えません。
- D*w*.*p* は数字を指定した有効桁数で書き出し、BEST*w*.出力形式よりも小数点揃えを行います。

例

次のプログラムは、D*w.p* 出力形式を示しています。

```
data _null_;
  declare char(15) a b c d e f;
  method run();
    a=put (12345,d10.4);
    b=put (1234.5,d10.4);
    c=put (123.45,d10.4);
    d=put (12.345,d10.4);
    e=put (1.2345,d10.4);
    f=put (.12345,d10.4);
    put a= b= c= d= e= f=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=  12345.0    b=   1234.5    c= 123.45000
d= 12.34500    e=  1.23450    f=0.12345
```

関連項目

出力形式:

- “BESTw. 出力形式” (103 ページ)

DATEw. 出力形式

SAS 日付値を *ddmmmyy*、*ddmmmyyyy* または *dd-mmm-yyyy* 形式で書き出します。

カテゴリ: 日付と時間

配置: 右

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

DATE*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト 7

範囲 5-11

ヒント 4 桁の年を日、月、年の間を区切らずに出力する場合は、幅に 9 を指定します。4 桁の年を日、月、年の間をハイフンで区切って出力する場合は、幅に 11 を指定します。

詳細

DATE*w*.出力形式は、SAS 日付値を *ddmmmyy*、*ddmmmyyyy* または *dd-mmm-yyyy* 形式で書き出します。構文の説明は次のとおりです。

dd
月の日を表す整数です。

mmm
月名の最初の 3 文字になります。

yy または *yyyy*
年を表す 2 桁または 4 桁の整数です。

例

次の例では、入力値として 20999 を使用します。20999 は、2017 年 6 月 29 日に対応する SAS 日付値です。

```
data _null_;
  declare varchar(15) a b c d e;
  method run();
    a=put (20999,date5.);
    b=put (20999,date6.);
    c=put (20999,date7.);
    d=put (20999,date8.);
    e=put (20999,date9.);
    put a= b= c= d= e=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=29JUN b= 29JUN c=29JUN17 d= 29JUN17 e=29JUN2017
```

関連項目

概念:

- “DS2 式” (*SAS DS2 プログラマガイド*)

関数:

- “DATE 関数” (495 ページ)

DATEAMPMw.d 出力形式

SAS 日時値を、AM または PM を付けて *ddmmmyy:hh:mm:ss.ss* 形式で書き出します。

カテゴリ: 日付と時間

配置: 右

操作: DECIMALCONV=システムオプションを STDIEEE に設定した場合、この出力形式を使用して書き出される出力が前のリリースと多少異なる可能性があります。詳細については、“DECIMALCONV= システムオプション” (*SAS システムオプション: リファレンス*)を参照してください。

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

DATEAMPM*w*.[*d*]

必須引数

w
出力フィールドの幅を指定します。

デフォルト 19

範囲 7-40

ヒント AM または PM を出力するには、*w* に少なくとも 13 を指定する必要があります。幅の値が 10 と 12 の間の場合、24 時間表記が出力されます。

オプション引数

d
秒値の小数点以下の桁数を指定します。

範囲 0-39

要件 *w* より小さい数にします。

注 *w–d*< 17 の場合、SAS は小数値を切り捨てます。

詳細

DATEAMPM*w.d* 出力形式は、SAS 日時値を *ddmmmyy:hh:mm:ss.ss* 形式で書き出します。構文の説明は次のとおりです。

dd
月の日を表す整数です。

mmm
月名の最初の 3 文字になります。

yy
年を表す 2 桁の整数です。

hh
時間を表す整数です。

mm
分を表す整数です。

ss.ss
小数点以下 2 桁の秒数です。

比較

DATEAMPM*w.d* 出力形式は DATETIME*w.d* 出力形式とほぼ同じですが、DATEAMPM*w.d* は時間の最後に AM または PM を出力します。

例

例の表では 1823167525 を入力値として使用します。18123167525 は 2017 年 10 月 9 日の午前 11:25:25 に相当する SAS 日時値です。

```
data _null_;
  declare varchar(15) a b c d e;
  method run();
    a=put (1823167525,dateampm.);
    b=put (1823167525,dateampm7.);
    c=put (1823167525,dateampm10.);
```

```
    d=put (1823167525,dateampm13.);
    e=put (1823167525,dateampm22.2);
    put a= b= c= d= e=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=09OCT17:11:25:2 b=09OCT17 c=09OCT17:11 d=09OCT17:11 AM e=09OCT17:11:25:2
```

関連項目

概念:

- “DS2 式” (*SAS DS2 プログラマガイド*)

出力形式:

- “DATETIMEw.d 出力形式” (119 ページ)

DATETIMEw.d 出力形式

SAS 日時値を *ddmmmyy:hh:mm:ss.ss* 形式で書き出します。

カテゴリ: 日付と時間

配置: 右

操作: DECIMALCONV=システムオプションを STDIEEE に設定した場合、この出力形式を使用して書き出される出力が前のリリースと多少異なる可能性があります。詳細については、“DECIMALCONV= システムオプション” (*SAS システムオプション: リファレンス*)を参照してください。

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

DATETIME*w*.[*d*]

必須引数

w
出力フィールドの幅を指定します。

デフォルト 16

範囲 7-40

ヒント SAS では、日付、時間、秒を含む SAS 日時値を書き込むために、最低でも *w* 値を 16 にする必要があります。オプションの小数点以下の秒数で値を返すには、*w* を 2 桁増やし、*d* に値を追加します。

オプション引数

d
秒値の小数点以下の桁数を指定します。

範囲 0-39

要件 *w* より小さい数にします。

詳細

DATETIME*w.d* 出力形式は、SAS 日時値を *ddmmmyy:hh:mm:ss.ss* 形式で書き出します。構文の説明は次のとおりです。

dd
月の日を表す整数です。

mmm
月名の最初の 3 文字になります。

yy
年を表す 2 桁の整数です。

hh
時間を 24 時間表記で表す整数です。

mm
分を表す整数です。

ss.ss
小数点以下 2 桁の秒数です。

注: *w–d*<17 の場合、SAS は小数値を切り捨てます。

比較

DATEAMPM*w.d* 出力形式は DATETIME*w.d* 出力形式とほぼ同じですが、DATEAMPM*w.d* は時間の最後に AM または PM を出力します。

例

例の表では 1817623985 を入力値として使用します。1817623985 は 2017 年 8 月 6 日の午前 7:33:05 に相当する SAS 日時値です。

```
data _null_;
  declare varchar(15) a b c d e f g h;
  method run();
    a=put (1817623985,datetime.);
    b=put (1817623985,datetime7.);
    c=put (1817623985,datetime12.);
    d=put (1817623985,datetime18.);
    e=put (1817623985,datetime18.1);
    f=put (1817623985,datetime19.);
    g=put (1817623985,datetime20.1);
    h=put (1817623985,datetime21.2);
    put a= b= c= d= e= f= g= h=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=06AUG17:07:33:0 b=06AUG17 c=  06AUG17:07 d=  06AUG17:07:33 e=06AUG17:07:33:0 f=06AUG2017:07:3
g=06AUG2017:07:33 h=06AUG2017:07:33
```

関連項目

概念:

- “DS2 式” (*SAS DS2 プログラマガイド*)

出力形式:

- “DATEw. 出力形式” (115 ページ)
- “DATEAMPMw.d 出力形式” (117 ページ)
- “TIMEw.d 出力形式” (195 ページ)

関数:

- “DATETIME 関数” (499 ページ)

DAYw. 出力形式

SAS 日付値の日付部分を書き出します。

カテゴリ: 日付と時間

配置: 右

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

DAY*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト 2

範囲 2-32

例

次の例では、入力値として 20999 を使用します。20999 は、2017 年 6 月 28 日に対応する SAS 日付値です。

```
data _null_;
  declare varchar(15) a;
  method run();
    a=put(20999,day2.);
    put a=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=29
```

関連項目

“DS2 式” (*SAS DS2 プログラマガイド*)

DDMMYYw. 出力形式

SAS 日付値を *ddmm<yy>yy* または *dd/mm/<yy>yy* 形式で書き出します。スラッシュが区切り文字として使用され、年は 2 桁または 4 桁の数字で書き出されます。

カテゴリ: 日付と時間

配置: 右

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

DDMMYY*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト 8

範囲 2-10

操作 *w* の値が 2 から 5 までの場合、日と月が可能な範囲で表示されます。*w* が 7 の場合、年を 2 桁で表し、スラッシュを使わずに出力します。

詳細

DDMMYY*w*.出力形式は、SAS 日付値を *ddmm<yy>yy* または *dd/mm/<yy>yy* 形式で書き出します。構文の説明は次のとおりです。

dd
月の日を表す整数です。

/
区切り文字です。

mm
月を表す整数です。

<yy>yy
年を表す 2 桁または 4 桁の整数です。

例

次の例では、入力値として 20999 を使用します。20999 は、2017 年 6 月 29 日に対応する SAS 日付値です。

```
data _null_;
  declare varchar(15) a b c d e f;
  method run();
    a=put(20999,ddmmyy.);
    b=put(20999,ddmmyy5.);
    c=put(20999,ddmmyy6.);
    d=put(20999,ddmmyy7.);
    e=put(20999,ddmmyy8.);
    f=put(20999,ddmmyy10.);
    put a= b= c= d= e= f=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=29/06/17 b=29/06 c=290617 d= 290617 e=29/06/17 f=29/06/2017
```

関連項目

概念:

- “DS2 式” (*SAS DS2 プログラマガイド*)

出力形式:

- “DATEw. 出力形式” (115 ページ)
- “DDMMYYxw. 出力形式” (124 ページ)
- “MMDDYYw. 出力形式” (171 ページ)
- “YYMMDDw. 出力形式” (217 ページ)

関数:

- “MDY 関数” (847 ページ)

DDMMYYxw. 出力形式

SAS 日付値を *ddmm<yy>yy* または *dd-mm-yy<yy>*形式で書き出します。出力形式名の *x* は、日、月、年を区切る特殊文字を表す文字で、ハイフン(-)、ピリオド(.)、空白文字、スラッシュ(/)、コロン(:)、または区切り文字なしになります。年は 2 桁または 4 桁のいずれかになります。

カテゴリ: 日付と時間

配置: 右

構文

DDMMYY*xw*.

必須引数

x
区切り文字を示します。または、区切り文字を日、月、年の間に挿入しないことを示します。有効な値は次のとおりです。

B
空白で区切ります。

C
コロンで区切ります。

D
ハイフンで区切ります。

N
区切り文字なしを表します。

P
ピリオドで区切ります。

S
スラッシュで区切ります。

w
出力フィールドの幅を指定します。

デフォルト 8

範囲 2-10

操作 *w* の値が 2 から 5 までの場合、日と月が可能な範囲で表示されます。*w* が 7 の場合、日付の年は 2 桁で、区切り文字を使用せずに表示されます。

x の値が N の場合、幅の範囲は 2–8 に変化します。

詳細

DDMMYY*xw*.出力形式は、SAS 日付値を *ddmm*<*yy*>*yy* または *dd*X*mm*X<*yy*>*yy* 形式で書き出します。構文の説明は次のとおりです。

dd
月の日を表す整数です。

X
指定された区切り文字です。

mm
月を表す整数です。

<yy>yy
年を表す 2 桁または 4 桁の整数です。

例

次の例では、入力値として 20999 を使用します。20999 は、2017 年 6 月 29 日に対応する SAS 日付値です。

```
data _null_;
  declare varchar(15) a b c d;
  method run();
    a=put(20999,ddmmyyc5.);
    b=put(20999,ddmmyyd8.);
    c=put(20999,ddmmyyn8.);
    d=put(20999,ddmmyyp10.);
    put a= b= c= d=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=29:06 b=29-06-17 c=29062017 d=29.06.2017
```

関連項目

概念:

- “DS2 式” (*SAS DS2 プログラマガイド*)

出力形式:

- “DATEw. 出力形式” (115 ページ)
- “DDMMYYw. 出力形式” (123 ページ)
- “MMDDYYxw. 出力形式” (173 ページ)
- “YYMMDDxw. 出力形式” (219 ページ)

関数:

- “DAY 関数” (500 ページ)
- “MDY 関数” (847 ページ)
- “MONTH 関数” (863 ページ)
- “YEAR 関数” (1263 ページ)

DOLLARw.d 出力形式

先頭にドル($)記号を付け、3 桁ごとにカンマを入れ、小数部分との区切りにはピリオドを使用して数値を出力します。

カテゴリ: 数値

配置: 右

操作: DECIMALCONV=システムオプションを STDIEEE に設定した場合、この出力形式を使用して書き出される出力が前のリリースと多少異なる可能性があります。詳細については、“DECIMALCONV= システムオプション” (*SAS システムオプション: リファレンス*)を参照してください。

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

DOLLAR*w*.[*d*]

必須引数

w
出力フィールドの幅を指定します。

デフォルト 6

範囲 2-32

オプション引数

d
数値の小数点以下の桁数を指定します。

範囲 0-31

要件 *w* より小さい数にします。

詳細

DOLLAR*w.d* 出力形式は、先頭にドル($)記号を付け、3 桁ごとにカンマを入れ、小数部分との区切りにはピリオドを使用して数値を出力します。

比較

- DOLLAR*w.d* 出力形式は DOLLARX*w.d* 出力形式とほぼ同じですが、DOLLARX*w.d* 出力形式では小数点とカンマの役割が逆になっています。この規則はヨーロッパの国で共通です。
- DOLLAR*w.d* 出力形式は COMMA*w.d* 出力形式とほぼ同じですが、COMMA*w.d* 出力形式は先頭のドル記号を出力しません。

例

次のプログラムは、DOLLAR*w.d* 出力形式を示しています。

```
data _null_;
  declare varchar(15) a;
  method run();
    a=put(1254.71,dollar10.2);
    put a=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a= $1,254.71
```

関連項目

出力形式:

- “COMMAw.d 出力形式” (110 ページ)
- “DOLLARXw.d 出力形式” (128 ページ)

DOLLARXw.d 出力形式

先頭にドル($)記号を付け、3 桁ごとにピリオドを入れ、小数部分との区切りにはカンマを使用して数値を出力します。

カテゴリ: 数値

配置: 右

操作: DECIMALCONV=システムオプションを STDIEEE に設定した場合、この出力形式を使用して書き出される出力が前のリリースと多少異なる可能性があります。詳細については、“DECIMALCONV= システムオプション” (*SAS システムオプション: リファレンス*)を参照してください。

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

DOLLARX*w*.[*d*]

必須引数

w
出力フィールドの幅を指定します。

デフォルト 6

範囲 2-32

オプション引数

d
数値の小数点以下の桁数を指定します。

デフォルト 0

範囲 0-31

要件 *w* より小さい数にします。

詳細

DOLLARX*w.d* 出力形式は、先頭にドル($)記号を付け、3 桁ごとにカンマを入れ、小数部分との区切りにはピリオドを使用して数値を出力します。

比較

- DOLLARX*w.d* 出力形式は DOLLAR*w.d* 出力形式とほぼ同じですが、DOLLARX*w.d* 出力形式では小数点とカンマの役割が逆になっています。この規則はヨーロッパの国で共通です。
- DOLLARX*w.d* 出力形式は COMMAX*w.d* 出力形式とほぼ同じですが、COMMA*w.d* 出力形式は先頭のドル記号を出力しません。

例

次のプログラムは、DOLLARX*w.d* 出力形式を示しています。

```
data _null_;
  declare varchar(15) a;
  method run();
    a=put(1254.71,dollarx10.2);
    put a=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a= $1.254,71
```

関連項目

出力形式:

- “COMMAXw.d 出力形式” (112 ページ)
- “DOLLARw.d 出力形式” (127 ページ)

DOWNAMEw. 出力形式

SAS 日付値から曜日名を書き出します。

カテゴリ: 日付と時間

配置: 右

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

DOWNAME*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト	9
範囲	1-32
ヒント	*w* を省略すると、 曜日名全体が出力されます。

詳細

必要に応じて、SAS により出力形式の幅に合うように曜日名が切り捨てられます。たとえば、出力形式 DOWNAME2.を指定すると、曜日名の最初の 2 文字が出力されます。

例

次のプログラムは、DOWNAME*w*.出力形式を示しています。

```
data _null_;
  declare varchar(15) a;
  method run();
    a=put(20999,downame.);
    put a=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a= Thursday
```

関連項目

概念:

- “DS2 式” (*SAS DS2 プログラマガイド*)

出力形式:

- “WEEKDAYw. 出力形式” (209 ページ)

DTDATEw. 出力形式

入力として SAS 日時値が必要です。SAS 日付値を *ddmmmyy* または *ddmmmyyyy* 形式で書き出します。

カテゴリ: 日付と時間

配置: 右

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

DTDATE*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト 7

範囲 5-9

ヒント 4 桁の年を出力する場合は幅に 9 を指定します。

詳細

DTDATE*w*.出力形式は、SAS 日付値を *ddmmmyy* または *ddmmmyyyy* 形式で書き出します。構文の説明は次のとおりです。

dd
月の日を表す整数です。

mmm
月名の最初の 3 文字になります。

yy または *yyyy*
年を表す 2 桁または 4 桁の整数です。

比較

DTDATE*w*.出力形式は、DATE*w*.出力形式と同じ種類の出力を作成します。相違点は、DTDATE*w*.出力形式には SAS 日時値が必要なことです。

例

例の表では 1823167525 の入力値を使用し、DTDATE*w*.出力形式に対応した 2 桁の年と 4 桁の年を出力します。1823167525 は 2017 年 10 月 9 日の午前 11:25:25 に相当する SAS 日時値です。

```
data _null_;
  declare varchar(15) a b;
  method run();
    a=put(1823167525,dtdate.);
    b=put(1823167525,dtdate9.);
    put a= b=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=09OCT17 b=09OCT2017
```

関連項目

概念:

- “DS2 式” (*SAS DS2 プログラマガイド*)

出力形式:

- “DATEw. 出力形式” (115 ページ)

DTMONYYw. 出力形式

SAS 日時値の日付部分を月と年として *mmmyy* または *mmmyyyy* 形式で書き出します。

カテゴリ: 日付と時間

配置: 右

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

DTMONYY*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト 5

範囲 5-7

詳細

DTMONYY*w*.出力形式は、SAS 日時値を *mmmyy* または *mmmyyyy* 形式で書き出します。構文の説明は次のとおりです。

mmm
月名の最初の 3 文字になります。

yy または *yyyy*
年を表す 2 桁または 4 桁の整数です。

比較

DTMONYY*w.* 出力形式と MONYY*w*.出力形式は、どちらも日付値を書き出すという点でほぼ同じです。相違点は、DTMONYY*w*.が入力として SAS 日時値を必要とし、MONYY*w*.は SAS 日付値が必要であることです。

例

例の表では、入力として値 1823167525 を使用します。1823167525 は、2017 年 10 月 9 日の午前 11:25:25 に相当する SAS 日時値です。

```
data _null_;
  declare varchar(15) a b c d;
  method run();
    a=put(1823167525,dtmonyy.);
    b=put(1823167525,dtmonyy5.);
    c=put(1823167525,dtmonyy6.);
    d=put(1823167525,dtmonyy7.);
    put a= b= c= d=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=OCT17 b=OCT17 c= OCT17 d=OCT2017
```

関連項目

概念:

- “DS2 式” (*SAS DS2 プログラマガイド*)

出力形式:

- “DATETIMEw.d 出力形式” (119 ページ)
- “MONYYw. 出力形式” (182 ページ)

DTWKDATXw. 出力形式

SAS 日時値の日付部分を曜日と日付として *day-of-week*、*dd month-name yy* (または *yyyy*)形式で書き出します。

カテゴリ: 日付と時間

配置: 右

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

DTWKDATX*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト 29

範囲 3-37

詳細

DTWKDATX*w*.出力形式は、SAS 日付値を *day-of-week*、*dd*、*month-name*、*yy* または *yyyy* 形式で書き出します。構文の説明は次のとおりです。

day-of-week
曜日名の最初の 3 文字か、曜日名全体です。

dd
月の日を表す整数です。

month-name
月名の最初の 3 文字か、月名全体です。

yy または *yyyy*
年を表す 2 桁または 4 桁の整数です。

比較

DTWKDATX*w*. 出力形式と WEEKDATX*w*. 出力形式は、どちらも日付値を書き出すという点でほぼ同じです。相違点は、DTWKDATX*w*.が入力として SAS 日時値を必要とし、WEEKDATX*w*.は SAS 日付値が必要であることです。

例

例の表では、入力として値 1823167525 を使用します。1823167525 は、2017 年 10 月 9 日の午前 11:25:25 に相当する SAS 日時値です。

```
data _null_;
  declare varchar(30) a b c d;
  method run();
    a=put(1823167525,dtwkdatx.);
    b=put(1823167525,dtwkdatx3.);
    c=put(1823167525,dtwkdatx8.);
    d=put(1823167525,dtwkdatx25.);
    put a= b= c= d=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=     Monday, 9 October 2017 b=Mon c=    Mon d=     Monday, 9 Oct 2017
```

関連項目

概念:

- “DS2 式” (*SAS DS2 プログラマガイド*)

出力形式:

- “DATETIMEw.d 出力形式” (119 ページ)
- “WEEKDATXw. 出力形式” (207 ページ)

DTYEARw. 出力形式

SAS 日時値の日付部分を年として *yy* または *yyyy* 形式で書き出します。

カテゴリ: 日付と時間

配置: 右

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

DTYEAR*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト 4

範囲 2-4

比較

DTYEAR*w*.出力形式と YEAR*w*.出力形式は、どちらも日付値を書き出すという点でほぼ同じです。相違点は、DTYEAR*w*.が入力として SAS 日時値を必要とし、YEAR*w*.は SAS 日付値が必要であることです。

例

例の表では値 1823167525 を入力として使用します。1823167525 は 2017 年 10 月 9 日、午前 11:25:25 に相当する SAS 日時値です。

```
data _null_;
  declare varchar(15) a b c d;
  method run();
    a=put(1823167525,dtyear.);
    b=put(1823167525,dtyear2.);
    c=put(1823167525,dtyear3.);
```

```
    d=put(1823167525,dtyear4.);
    put a= b= c= d=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=2017 b=17 c= 17 d=2017
```

関連項目

概念:

- “DS2 式” (*SAS DS2 プログラマガイド*)

出力形式:

- “DATETIMEw.d 出力形式” (119 ページ)
- “YEARw. 出力形式” (210 ページ)

DTYYQCw. 出力形式

SAS 日時値から、年と四半期をコロン(:)で区切って書き出します。

カテゴリ: 日付と時間

配置: 右

注:

構文

DTYYQC*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト 4

範囲 4-6

詳細

DTYYQC*w*.出力形式は、SAS 日時値を *yy* または *yyyy* 形式の年にコロン(:)と四半期を示す数値を続けて書き出します。

例

例の表では、入力として値 1823167525 を使用します。1823167525 は、2017 年 10 月 9 日の午後 11:25:25 に相当する SAS 日時値です。

```
data _null_;
  declare varchar(15) a b c d;
  method run();
    a=put(1823167525,dtyyqc.);
    b=put(1823167525,dtyyqc4.);
    c=put(1823167525,dtyyqc5.);
    d=put(1823167525,dtyyqc6.);
    put a= b= c= d=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=17:4 b=17:4 c= 17:4 d=2017:4
```

関連項目

概念:

- “DS2 式” (*SAS DS2 プログラマガイド*)

出力形式:

- “DATETIMEw.d 出力形式” (119 ページ)

E8601DAw. 出力形式

ISO 8601 規格の拡張表記 *yyyy-mm-dd* を使用して日付値を書き出します。

カテゴリ:	日付と時間 ISO 8601
配置:	左
別名:	IS8601DA*w*.
制限事項:	UTC タイムゾーンオフセット値はサポートされていません。

サポート: ISO 8601 Element 5.2.1.1, complete representation

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

E8601DA*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト 10

要件 10 である必要があります。

詳細

E8601DA 出力形式は、ISO 8601 規格の拡張表記 *yyyy-mm-dd* を使用して日付を書き出します。

yyyy
4 桁の年です。

mm
01 から 12 までの 2 桁の月です(ゼロ埋め込み)。

dd
01 から 31 までの 2 桁の日です(ゼロ埋め込み)。

例

次の例では、入力値として 20999 を使用します。20999 は、2017 年 6 月 29 日に対応する SAS 日付値です。

```
data _null_;
  declare varchar(15) a;
  method run();
    a=put(20999,e8601da.);
    put a=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=2017-06-29
```

関連項目

"ISO 8601 規格の基本表記と拡張表記を使用した日付と時間の処理" (*SAS 出力形式と入力形式: リファレンス*)

E8601DNw. 出力形式

ISO 8601 規格の拡張表記 *yyyy-mm-dd* を使用して、SAS 日時値から日付を書き出します。

カテゴリ: 日付と時間
ISO 8601

配置: 左

別名: IS8601DN

制限事項: UTC タイムゾーンオフセット値はサポートされていません。

サポート: ISO 8601 Element 5.2.1.1, complete representation

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

E8601DN*w*.

必須引数

w
入力幅を指定します。

デフォルト 10

要件 入力幅は 10 である必要があります。

詳細

E8601DN 出力形式は、ISO 8601 規格の拡張日付表記 *yyyy-mm-dd* を使用して日付を書き出します。

yyyy
4 桁の年です。

mm
01 から 12 までの 2 桁の月です(ゼロ埋め込み)。

dd
01 から 31 までの 2 桁の日です(ゼロ埋め込み)。

例

次の例では、入力値として 1793308532 を使用します。1793308532 は、2016 年 10 月 28 日に対応する SAS 日付値です。

```
data _null_;
  declare varchar(15) a;
  method run();
    a=put(1793308532,e8601dn.);
    put a=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=2016-10-28
```

関連項目

"ISO 8601 規格の基本表記と拡張表記を使用した日付と時間の処理" (*SAS 出力形式と入力形式: リファレンス*)

E8601DTw.d 出力形式

ISO 8601 規格の拡張表記 *yyyy-mm-ddThh:mm:ss.ffffff* を使用して日時値を書き出します。

カテゴリ:	日付と時間 ISO 8601
配置:	左
別名:	IS8601DT*w.d*
制限事項:	UTC タイムゾーンオフセット値はサポートされていません。

サポート: ISO 8601 Element 5.4.1, complete representation

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

E8601DT*w.d*

必須引数

w
入力幅を指定します。

デフォルト 19

範囲 16-26

d
秒値の小数点以下の桁数を指定します。この引数はオプションです。

デフォルト 0

範囲 0-6

詳細

E8601DT 出力形式は、ISO 8601 規格の拡張日付表記 *yyyy-mm-dd*T*hh:mm:ss.ffffff* を使用して日時値を書き出します。

yyyy
4 桁の年です。

mm
01 から 12 までの 2 桁の月です(ゼロ埋め込み)。

dd
01 から 31 までの 2 桁の日です(ゼロ埋め込み)。

hh
00 から 23 までの 2 桁の時間です(ゼロ埋め込み)。

mm
00 から 59 までの 2 桁の分です(ゼロ埋め込み)。

ss
00 から 59 までの 2 桁の秒です(ゼロ埋め込み)。

ffffff
6 桁までの有効桁数を持つ、オプションの秒の分数です。各桁は 0 から 9 までです。

注: 幅を 16 に指定すると、SAS は秒の値を 0 と見なして出力から省略します。

例

次の例では、入力値として 1793308532 を使用します。1793308532 は、2016 年 10 月 28 日の午後 9:15:32 に対応する SAS 日時値です。

```
data _null_;
  declare varchar(15) a;
  method run();
    a=put(1793308532,e8601dt.);
    put a=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=2016-10-28T21:1
```

関連項目

"ISO 8601 規格の基本表記と拡張表記を使用した日付と時間の処理" (*SAS 出力形式と入力形式: リファレンス*)

E8601DZw. 出力形式

ISO 8601 規格の日時とタイムゾーンの拡張表記 *yyyy-mm-dd*T*hh:mm:ss*+00:00 を使用して、基準子午線の協定世界時(UTC)の日時値を書き出します。

カテゴリ:	日付と時間 ISO 8601
配置:	左
別名:	IS8601DZ*w.*
サポート:	ISO 8601 Element 5.4.1, complete representation
注:	2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

E8601DZ*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト 26

範囲 20-35

詳細

UTC 値は、イギリスのグリニッジの基準子午線に基づく時間とタイムゾーンを示します。E8601DZ 出力形式は、SAS 日時値を次の ISO 8601 規格の拡張日時表記のいずれかを使用して書き出します。

- *yyyy-mm-dd*T*hh:mm:ss*+00:00

  注: この形式は、*w* の長さがこのタイムゾーン表記に十分な場合に使用します。

- *yyyy-mm-dd*T*hh:mm:ss*Z

  注: この形式は、*w* の長さが+00:00 タイムゾーン表記に十分でない場合に使用します。

yyyy
4 桁の年です。

mm
01 から 12 までの 2 桁の月です(ゼロ埋め込み)。

dd
01 から 31 までの 2 桁の日です(ゼロ埋め込み)。

hh
00 から 24 までの 2 桁の時間です(ゼロ埋め込み)。

mm
00 から 59 までの 2 桁の分です(ゼロ埋め込み)。

ss
00 から 59 までの 2 桁の秒です(ゼロ埋め込み)。

+00:00
時間が基準子午線(イギリスのグリニッジ)に対応した時間であることを示します。

タイムゾーンオフセットが指定された ISO 8601 規格の時間値または日時値は、オフセットに示された時間数と分数によって調整され、基準子午線(イギリスのグ

リニッジ)の時間または日時として処理されます。E8601DZ 出力形式は、常に +00:00 の基準子午線オフセット値を使用して日時値を書き出します。+00:00 以外の UTC オフセットを使用して日時を書き出すには、“E8601LZw. 出力形式” (*SAS 出力形式と入力形式: リファレンス*)を参照してください。

制約: 短い形式+00 はサポートされていません。

Z

時間が基準子午線(イギリスのグリニッジ)に対応した時間、つまり+00:00 UTC 時間であることを示します。出力形式の幅が+00:00 表記に十分でない場合は、Z を使用します。

例

次の例では、入力値として 1793308532 を使用します。1793308532 は、2016-10-28T21:15:32+00:00 に対応する SAS 日付値です。

```
data _null_;
  declare varchar(30) a;
  method run();
    a=put(1793308532,e8601dz.);
    put a=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a= 2016-10-28T21:15:32+00:00
```

関連項目

“ISO 8601 規格の基本表記と拡張表記を使用した日付と時間の処理” (*SAS 出力形式と入力形式: リファレンス*)

E8601LZw. 出力形式

ISO 8601 規格の拡張時間表記 *hh:mm:ss+|–hh:mm* を使用して、ローカル SAS セッションに対応した協定世界時(UTC)のオフセットを追加することにより、時間値をローカル時間として書き出します。

カテゴリ:	日付と時間 ISO 8601
配置:	左
別名:	IS8601LZ*w*.
サポート:	ISO 8601 Element 5.3.1.1, complete representation

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

E8601LZ*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト 14

範囲 9-20

詳細

E8601LZ 出力形式は、次の ISO 8601 規格の拡張時間表記のいずれかを使用して時間値を調整せずに書き出し、ローカル SAS セッションに対応した UTC タイムゾーンオフセットを追加します。

- *hh:mm:ss*+|–*hh:mm*

  注: この形式は、*w* の長さがこの時間表記に十分である場合に使用します。

- *hh:mm:ss*Z

  注: この形式は、*w* の長さが+|– *hh:mm* タイムゾーン表記に十分でない場合に使用します。

hh
00 から 23 までの 2 桁の時間です(ゼロ埋め込み)。

mm
00 から 59 までの 2 桁の分です(ゼロ埋め込み)。

ss
00 から 59 までの 2 桁の秒です(ゼロ埋め込み)。

+|–*hh:mm*
基準子午線時間からの時間と分の符号付きオフセットです。オフセットは+|–*hh:mm* (つまり+または–に続く 5 文字)である必要があります。

+は基準子午線の東部タイムゾーンに、-は基準子午線の西部タイムゾーンにそれぞれ使用します。たとえば、+02:00 は基準子午線の東部での 2 時間差を表し、–06:00 は基準子午線の西部での 6 時間差を表します。

制約: 短い形式+|–*hh* はサポートされていません。

Z
基準子午線(イギリスのグリニッジ)に対応した時間、つまり+00:00 UTC 時間であることを示します。

SAS では、時間値を *hh:mm.ffffff* 形式を使用して書き出し、ローカル SAS セッションに対応した基準子午線とのタイムゾーンオフセットのタイムゾーンインジケーター+|–*hh:mm* または Z を追加します。Z タイムゾーンインジケーターは、出力形式の長さの値が 14 未満の場合に使用されます。

両方のゾーンインジケーターを使用して同じ時間を書き出す場合、UTC に基づく 2 つの異なる時間を示します。たとえば、ローカル SAS セッションでアメリカの東部標準時が使用され、その時間値が 45824 である場合、12:43:44-04:00 または 12:43:44Z が書き出されます。時間 12:43:44–04:00 は、基準子午線の時間の 16:43:44+00:00 です。Z は、その時間が基準子午線の時間である、つまり 12:43:44+00:00 であることを示します。

SAS では UTC 時間を E8601TZ 入力形式を使用して読み込む際に、オフセット調整後の時間が 24 時間を超えていたり 00 時間未満である場合、オフセット調整後の時間が 00:00:00 と 24:00:00 の間になるように値を調整します。E8601LZ 出力形式をこの時間範囲外の時間に適用する場合、時間はアスタリスクを使用してフォーマットされ、その値が範囲外であることが示されます。

例

次の例では、20999 を入力値として使用します。これは、ローカル時間値 05:49:59-04:00 です。

```
data _null_;
  declare varchar(30) a;
  method run();
    a=put(20999,e8601lz.);
    put a=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=05:49:59-04:00
```

関連項目

“ISO 8601 規格の基本表記と拡張表記を使用した日付と時間の処理” (*SAS 出力形式と入力形式: リファレンス*)

E8601TMw.d 出力形式

ISO 8601 規格の拡張表記 *hh:mm:ss.ffffff* を使用して時間値を書き出します。

カテゴリ: 日付と時間
ISO 8601

配置: 左

別名: IS8601TM*w.d*

制限事項: UTC タイムゾーンオフセット値はサポートされていません。

サポート: ISO 8601 Element 5.3.1.1, complete representation, and 5.3.1.3, representation of decimal fractions

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

E8601TM*w.d*

必須引数

w
出力フィールドの幅を指定します。

デフォルト 8

範囲 8-15

d
秒値の小数点以下の桁数を指定します。この引数はオプションです。

デフォルト 0

範囲 0-6

詳細

E8601TM 出力形式は、SAS 時間値を ISO 8601 規格の拡張時間表記 *hh:mm:ss.ffffff* を使用して書き出します。

hh
00 から 23 までの 2 桁の時間です(ゼロ埋め込み)。

mm
00 から 59 までの 2 桁の分です(ゼロ埋め込み)。

ss
00 から 59 までの 2 桁の秒です(ゼロ埋め込み)。

.ffffff
6 桁までの有効桁数を持つ、オプションの秒の分数です。各桁は 0 から 9 までです。

例

次の例では、05:49:59 の時間値に対応する 20999 の入力値を使用しています。

```
data _null_;
  declare varchar(30) a;
  method run();
    a=put(20999,e8601tm.);
    put a=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=05:49:59
```

関連項目

"ISO 8601 規格の基本表記と拡張表記を使用した日付と時間の処理" (*SAS 出力形式と入力形式: リファレンス*)

E8601TZw.d 出力形式

時間値を協定世界時(UTC)に調整し、時間値を ISO 8601 規格の拡張表記 *hh*:*mm*:*ss*.<*fff*>+|–*hh*:*mm* を使用して書き出します。

カテゴリ:	日付と時間 ISO 8601
配置:	左
別名:	IS8601TZ*w.d*
サポート:	ISO 8601 Element 5.3.1.1, complete representation

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

E8601TZ*w*.*d*

必須引数

w
出力フィールドの幅を指定します。

デフォルト 14

範囲 9-20

d
秒値の小数点以下の桁数を指定します。この引数はオプションです。

デフォルト 0

範囲 0-6

詳細

UTC 時間値は、イギリスのグリニッジの基準子午線に基づく時間とタイムゾーンを示します。E8601TZ 出力形式は、時間値を次の ISO 8601 規格の拡張時間表記のいずれかを使用して書き出します。

- *hh:mm:ss<.fff>+|–hh:mm*

  注: この形式は、*w* の長さがこのタイムゾーン表記に十分な場合に使用します。

- *hh:mm:ss*Z

  注: この形式は、*w* の長さが+|–*hh:mm* タイムゾーン表記に十分でない場合に使用します。

hh
00 から 23 までの 2 桁の時間です(ゼロ埋め込み)。

mm
00 から 59 までの 2 桁の分です(ゼロ埋め込み)。

ss
00 から 59 までの 2 桁の秒です(ゼロ埋め込み)。

fff
オプションの秒の端数です。

+|–*hh:mm*
基準子午線時間からの時間と分の符号付きオフセットです。オフセットは+|–*hh:mm* (つまり、+または–を伴う 5 文字)である必要があります。

制約: 短い形式+|–*hh* はサポートされていません。

+は基準子午線の東部タイムゾーンに、-は基準子午線の西部タイムゾーンにそれぞれ使用します。たとえば、+02:00 は基準子午線の東部での 2 時間差を表し、–06:00 は基準子午線の西部での 6 時間差を表します。

Z
基準子午線(イギリスのグリニッジ)に対応した時間、つまり+00:00 UTC 時間であることを示します。

SAS では UTC 時間を E8601TZ 入力形式を使用して読み込む際に、オフセット調整後の時間が 24 時間を超えていたり 00 時間未満である場合、オフセット調整後の時間が 00:00:00 と 24:00:00 の間になるように値を調整します。E8601TZ 出力形式をこの時間範囲外の時間に適用する場合、時間はアスタリスクを使用してフォーマットされ、その値が範囲外であることが示されます。

比較

00:00:00 と 24:00:00 の間の時間値に対して、B8601TZ 出力形式は時間値を基準子午線の時間に調整し、調整後の値を国際規格の拡張時間表記で書き出します。B8601LZ 出力形式は時間を調整せず、ローカル SAS セッションの UTC タイムゾーンオフセットを使用して時間値を国際規格の拡張時間表記で書き出します。

例

次の例では、05:49:59+00:00 の時間値に対応する 20999 の入力値を使用しています。

```
data _null_;
  declare varchar(30) a;
  method run();
    a=put(20999,e8601tz.);
    put a=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=05:49:59+00:00
```

関連項目

"ISO 8601 規格の基本表記と拡張表記を使用した日付と時間の処理" (*SAS 出力形式と入力形式: リファレンス*)

Ew. 出力形式

数値を指数表記で書き出します。

カテゴリ: 数値

配置: 右

操作: DECIMALCONV=システムオプションを STDIEEE に設定した場合、この出力形式を使用して書き出される出力が前のリリースと多少異なる可能性があります。詳細については、"DECIMALCONV= システムオプション" (*SAS システムオプション: リファレンス*)を参照してください。

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

E*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト 12

範囲 7-32

詳細

SAS は、結果の最初のカラムをマイナス記号を書き込む場所として確保します。

例

次の例では、入力値として 1257 を使用します。

```
data _null_;
  declare char(15) a b;
  method run();
    a=put(1257,e10.);
    b=put(-1257,e10.);
    put a= b=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a= 1.257E+03    b=-1.257E+03
```

EUROw.d 出力形式

先頭にユーロ記号(E)を付け、3 桁ごとにカンマを入れ、小数部分との区切りにはピリオドを使用して数値を出力します。

カテゴリ: 数値

配置: 右

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

EURO*w.d*

必須引数

w
出力フィールドの幅を指定します。

デフォルト 6

範囲 1-32

ヒント ユーロ記号を出力の一部にする場合は、必ず適切な幅を選択してください。

d
数値の小数点以下の桁数を指定します。

デフォルト 0

範囲　0-31

要件　*w* より小さい数にします。

比較

- EURO*w.d* 出力形式は EUROX*w.d* 出力形式とほぼ同じですが、EUROX*w.d* 出力形式では小数点とカンマの役割が逆になっています。この規則はヨーロッパの国で共通です。
- EURO*w.d* 出力形式は DOLLAR*w.d* 出力形式とほぼ同じですが、DOLLAR*w.d* 出力形式がユーロ記号のかわりにドル記号を先頭に書き込む点が異なります。

注: EUROX*w.d* 出力形式は、ユーロ文字(U+20AC)を使用します。DBCS バージョンの SAS と、ユーロ文字をサポートしていないエンコーディングを使用すると、エラーが発生します。このエラーを防ぐには、セッションエンコーディングをユーロ文字をサポートするエンコーディングに変更します。

例

次のプログラムは、EURO*w.d* 出力形式を示しています。

```
data _null_;
  declare char(15) a b c d;
  method run();
    a=put(1254.71,euro10.2);
    b=put(1254.71,euro5.);
    c=put(1254.71,euro9.2);
    d=put(1254.71,euro15.3);
    put a= b= c= d=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a= €1,254.71    b=€1255       c=€1,254.71    d=    €1,254.710
```

関連項目

出力形式:

- “EUROXw.d 出力形式” (156 ページ)

EUROXw.d 出力形式

先頭にユーロ記号(E)を付け、3 桁ごとにピリオドを入れ、小数部分との区切りにはカンマを使用して数値を出力します。

カテゴリ: 数値

配置: 右

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

EUROX*w.d*

必須引数

w
出力フィールドの幅を指定します。

デフォルト 6

範囲 1-32

ヒント ユーロ記号を出力の一部にする場合は、必ず適切な幅を選択してください。

d
数値の小数点以下の桁数を指定します。

デフォルト 0

範囲 0-31

要件 *w* より小さい数にします。

比較

- EUROX*w.d* 出力形式は EURO*w.d* 出力形式とほぼ同じですが、EURO*w.d* 出力形式ではカンマと小数点の役割が逆になっています。この規則は英語圏の国で共通です。

- EUROX*w.d* 出力形式は DOLLARX*w.d* 出力形式とほぼ同じですが、DOLLARX*w.d* 出力形式がユーロ記号のかわりにドル記号を先頭に書き込む点が異なります。

注: EUROX*w.d* 出力形式は、ユーロ文字(U+20AC)を使用します。DBCS バージョンの SAS と、ユーロ文字をサポートしていないエンコーディングを使用すると、エラーが発生します。このエラーを防ぐには、セッションエンコーディングをユーロ文字をサポートするエンコーディングに変更します。

例

次のプログラムは、EUROX*w.d* 出力形式を示しています。

```
data _null_;
  declare char(15) a b c d;
  method run();
    a=put(1254.71,eurox10.2);
    b=put(1254.71,eurox5.);
    c=put(1254.71,eurox9.2);
    d=put(1254.71,eurox15.3);
    put a= b= c= d=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a= €1.254,71    b=€1255       c=€1.254,71    d=    €1.254,710
```

関連項目

出力形式:

- “DOLLARXw.d 出力形式” (128 ページ)
- “EUROw.d 出力形式” (154 ページ)

FLOATw.d 出力形式

数値に 10 の *d* 乗を掛けた、単精度のネイティブ浮動小数点値を生成します。

カテゴリ: 数値

配置: 左

構文

FLOAT*w*.[*d*]

必須引数

w
出力フィールドの幅を指定します。

要件 幅は 4 である必要があります。

オプション引数

d
値を乗算する 10 のべき乗を指定します。

デフォルト 0

範囲 0-31

詳細

FLOAT4.によって書き出される値は通常、使用している動作環境で実行される、単精度値を扱う他の外部プログラムによって読み込まれるための値です。フォーマットされる値が欠損値、あるいは単精度のネイティブ浮動小数点の範囲外である場合、ゼロの単精度の値が生成されます。

例

次の例では、DECLARE ステートメントで VARBINARY データ型を使用して、バイナリ値の 16 進表現を取得します。

ステートメント	結果 [1]
a=put(1,float4.);	0000803F

1 結果は、IEEE 形式で保存したバイナリ値を 16 進表現で表したものです。

FRACTw. 出力形式

数値を分数に変換します。

カテゴリ: 数値

配置: 右

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

FRACT*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト 10

範囲 4-32

詳細

1 を 3 で割ると、値は 0.33333333 となります。この値を 1/3 として書き出すには、FRACT*w*.出力形式を使用します。FRACT*w*. は、分数を約分して書き出します。50/100 は 1/2 となります。

例

次のプログラムは、FRACT*w*.出力形式を示しています。

```
data _null_;
  declare char(15) a b;
  method run();
    a=put(0.6666666667,fract8.);
    b=put(0.2784,fract8.);
    put a= b=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=   2/3      b= 174/625
```

HEXw. 出力形式

バイナリ実数(浮動小数点)値を 16 進表現に変換します。

カテゴリ: 数値

配置: 左

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

HEX*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト 8

範囲 1-16

ヒント *w* < 16 の場合、HEX*w*.出力形式はバイナリ実数を固定小数点数に変換してから 16 進表現に変換します。また、負数を 2 つの補数にして右揃えで書き出します。*w* が 16 の場合、HEX*w*.は浮動小数点数値を 16 進表現で表示します。

詳細

どの動作環境でも、HEX*w*.によって書き出される最下位バイトは右端のバイトになります。一部の動作環境では、最下位桁が最初のバイトである整数が保存されます。HEX*w*.出力形式は、バイト単位の重要度に関係なく、どの動作環境でも一貫した結果を生成します。

注: 動作環境によって浮動小数点数の保存方法は異なります。しかし、HEX16.出力形式は、使用されている動作環境での保存方法と同じように、16 進表現の浮動小数点数を書き出します。

比較

HEX*w*. 数値出力形式と$HEX*w*. 文字出力形式は、ともに値を 16 進表現に変換します。

例

次のプログラムは、HEX*w*.出力形式を示しています。

```
data _null_;
  declare char(15) a b c d;
  method run();
    a=put(35.4, hex8.);
    b=put(88, hex8.);
    c=put(2.33, hex8.);
    d=put(-150, hex8.);
    put a= b= c= d=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=00000023      b=00000058      c=00000002      d=FFFFFF6A
```

関連項目

出力形式:

- “$HEXw. 出力形式” (81 ページ)

HHMMw.d 出力形式

SAS 時間値を時間および分として *hh:mm* 形式で書き出します。

カテゴリ: 日付と時間

配置: 右

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

HHMM*w*.[*d*]

必須引数

w
出力フィールドの幅を指定します。

デフォルト 5

範囲 2-20

オプション引数

d
分値の小数点以下の桁数を指定します。小数点以下の桁数は、分の端数を示します。

デフォルト 0

範囲 0-19

要件 *w* より小さい数にします。

詳細

HHMM*w.d* 出力形式は、SAS 日時値を *hh:mm* 形式で書き出します。構文の説明は次のとおりです。

hh
整数です。

mm
00 から 59 の範囲の時間(分)です。

時間と分は、SAS 時間値の秒値に基づき丸められます。

比較

HHMM*w.d* 出力形式は TIME*w.d* 出力形式とほぼ同じですが、HHMM*w.d* 出力形式は秒を出力しません。

HHMM*w.d* 出力形式形式と TIME*w.d* 出力形式は、1 桁の時間にはその先頭に空白を書き出します。TOD*w.d* 出力形式は、1 桁の時間にはその先頭にゼロを書き出します。

例

次の例では、入力値として 46796 を使用します。46796 は、午後 12:59:56 に相当する SAS 時間値です。

最初の例では、SAS 時間値の秒値に基づき、時間値が 4 秒切り上げられます。2 番目の例では、出力形式に小数点以下の桁数として 2 を追加指定することにより、56 秒が 1 分の 93%として表示されています。

```
data _null_;
  declare char(15) a b;
  method run();
    a=put(46796,hhmm.);
    b=put(46796,hhmm8.2);
    put a= b=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=13:00         b=12:59.93
```

関連項目

概念:

- “DS2 式” (*SAS DS2 プログラマガイド*)

出力形式:

- “HOURw.d 出力形式” (164 ページ)
- “MMSSw.d 出力形式” (175 ページ)
- “TIMEw.d 出力形式” (195 ページ)
- “TODw.d 出力形式” (200 ページ)

関数:

- “HMS 関数” (693 ページ)
- “HOUR 関数” (699 ページ)
- “MINUTE 関数” (854 ページ)
- “SECOND 関数” (1126 ページ)
- “TIME 関数” (1187 ページ)

HOURw.d 出力形式

SAS 時間値を時間および時間の小数として書き出します。

カテゴリ: 日付と時間

配置: 右

操作: DECIMALCONV=システムオプションを STDIEEE に設定した場合、この出力形式を使用して書き出される出力が前のリリースと多少異なる可能性があります。詳細については、“DECIMALCONV= システムオプション” (*SAS システムオプション: リファレンス*)を参照してください。

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

HOUR*w*.[*d*]

必須引数

w
出力フィールドの幅を指定します。

デフォルト 2

範囲 2-20

オプション引数

d
時間値の小数点以下の桁数を指定します。時間の小数が出力されます。

範囲 0-19

要件 *w* より小さい数にします。

詳細

時間は、SAS 時間値の分値に基づき丸められます。

例

次の例では、入力値として 41400 を使用します。41400 は、午前 11:30 に相当する SAS 時間値です。

```
data _null_;
  declare char(15) a;
  method run();
    a=put(41400,hour4.1);
    put a=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=11.5
```

関連項目

概念:

- “DS2 式” (*SAS DS2 プログラマガイド*)

出力形式:

- “HHMMw.d 出力形式” (161 ページ)
- “MMSSw.d 出力形式” (175 ページ)
- “TIMEw.d 出力形式” (195 ページ)
- “TODw.d 出力形式” (200 ページ)

関数:

- “HMS 関数” (693 ページ)
- “HOUR 関数” (699 ページ)
- “MINUTE 関数” (854 ページ)
- “SECOND 関数” (1126 ページ)
- “TIME 関数” (1187 ページ)

IEEEw.d 出力形式

数値に 10 の *d* 乗を掛けて、IEEE 浮動小数点値を生成します。

カテゴリ: 数値

配置: 左

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

注意: **値が大きい浮動小数点数値や有効桁数が必要な浮動小数点数値は、IEEE 出力形式を使用して IBM メインフレームに書き出し、IEEE 入力形式で読み込みなおす場合、元の SAS 値とは異なる場合があります。**

構文

IEEE*w*.[*d*]

必須引数

w
出力フィールドの幅を指定します。

デフォルト 8

範囲 3-8

ヒント *w* が 8 の場合、IEEE の倍精度浮動小数点数が書き出されます。*w* が 5、6 または 7 の場合、相当するバイト数が切り捨てられたと仮定して、IEEE の倍精度浮動小数点数が書き出されます。*w* が 4 の場合、IEEE の単精度浮動小数点数が書き出されます。*w* が 3 の場合、1 バイトが切り捨てられたと仮定して、IEEE の単精度浮動小数点数が書き出されます。

オプション引数

d
10^d で数値を乗算するよう指定します。

デフォルト 0

範囲 0-10

詳細

この出力形式は、IEEE*w.d* が使用される浮動小数点数表現である動作環境で使用されます。また、IEEE*w.d* 出力形式を使用して、IEEE 浮動小数点数表現を使用する動作環境のプログラムによって使用されるファイルを作成することもできます。

通常プログラムは、単精度(4 バイト)または倍精度(8 バイト)で IEEE 値を生成します。プログラムは、出力ファイルのスペースを削減する場合にのみ切り捨てを実行します。機械語命令では、浮動小数点数が 4 バイトまたは 8 バイトのいずれかであ

る必要があります。IEEE*w.d* 出力形式では、他の長さも処理できるため、スペースの削減のために切り捨てられたデータを含むファイルにデータを書き込めます。

例

次の例では、DECLARE ステートメントで VARBINARY データ型を使用して、IEEE 形式で保存されたバイナリ値の 16 進表現である結果を生成します。

```
data _null_;
  declare varbinary(8) a;
  declare varchar(15) b;
  method run();
    a=put(1,ieee4.);
    b= put(a,$hex.);
    put b=;
    a=put(1,ieee5.);
    b= put(a,$hex.);
    put b=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
b=3F800000
b=3FF0000000
```

JULIANw. 出力形式

SAS 日付値を *yyddd* または *yyyyddd* 形式のユリウス日付として書き出します。

カテゴリ: 日付と時間

配置: 左

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

JULIAN*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト 5

範囲 5-7

ヒント *w* が 5 の場合、JULIAN*w*.出力形式では日付の年の部分を 2 桁で書き出します。*w* が 7 の場合、JULIAN*w*.出力形式では日付の年の部分を 4 桁で書き出します。

詳細

JULIAN*w*.出力形式では、SAS 日付値を *yyddd* または *yyyyddd* 形式で書き出します。構文の説明は次のとおりです。

yy または *yyyy*
年を表す 2 桁または 4 桁の整数です。

ddd
1 から 365 まで(うるう年の場合は 1 から 366 まで)の年の日数です。

例

次の例では、入力値として 20999 を使用します。20999 は、2017 年 6 月 29 日(年の 180 日目)に相当する SAS 日付値です。

```
data _null_;
  declare char(30) a b;
  method run();
    a=put(20999,julian5.);
    b=put(20999,julian7.);
    put a= b=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=17180                b=2017180
```

関連項目

概念:

- “DS2 式” (*SAS DS2 プログラマガイド*)

関数:

- “DATEJUL 関数” (496 ページ)

MDYAMPMw.d 出力形式

SAS 日時値を *mm/dd/yy<yy> hh:mm* AM|PM 形式で書き出します。年は 2 桁または 4 桁のいずれかです。

カテゴリ: 日付と時間

配置: 右

操作: DECIMALCONV=システムオプションを STDIEEE に設定した場合、この出力形式を使用して書き出される出力が前のリリースと多少異なる可能性があります。詳細については、“DECIMALCONV= システムオプション” (*SAS システムオプション: リファレンス*)を参照してください。

注: デフォルトの期間は AM です。

2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

MDYAMPM*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト 19

範囲 8-40

詳細

MDYAMPM*w.d* 出力形式は、SAS 日時値を次の形式で書き出します。

mm/*dd*/*yy*[yy] *hh*:*mm*[AM | PM]:

mm
月を表す 1 から 12 までの整数です。

dd
月の日を表す 1 から 31 の整数です。

yy または *yyyy*
年を表す 2 桁または 4 桁の整数です。

hh
時間を表す 00 から 23 までの整数です。

mm
分を表す 00 から 59 までの整数です。

AM | PM
00:01−12:00 正午(AM)または 12:01−12:00 深夜(PM)のいずれかです。デフォルトは AM です。

日時の区切り文字
SAS で日付要素と時間要素を区切るために使用するスラッシュ(/)、コロン(:)、空白文字などの特殊文字のいずれか 1 つです。

比較

MDYAMPM*w.*出力形式は、日時値を区切り文字付きの *mm/dd/yy<yy> hh:mm* AM | PM 形式で書き出します。日付と時間の間にスペースが挿入されます。

DATETIME*w.d* 出力形式は、日時値を区切り文字付きの *ddmmmyy<yy>: hh:mm:ss.ss* 形式で書き出します。

例

この例では、入力値として 1823167525 を使用します。1823167525 は 2017 年 10 月 9 日の午前 11:25:25 に相当する SAS 日時値です。

```
data _null_;
  declare char(30) a;
  method run();
    a=put(1823167525,mdyampm18.);
    put a=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=  10/9/17 11:25 AM
```

関連項目

出力形式:

- “DATETIMEw.d 出力形式” (119 ページ)

MMDDYYw. 出力形式

SAS 日付値を *mmdd<yy>yy* または *mm/dd/<yy>yy* 形式で書き出します。スラッシュが区切り文字として使用され、年は 2 桁または 4 桁の数字で書き出されます。

カテゴリ: 日付と時間

配置: 右

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

MMDDYY*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト 8

範囲 2-10

操作 *w* の値が 2 から 5 までの場合、月と日が可能な範囲で表示されます。*w* が 7 の場合、年を 2 桁で表し、スラッシュを使わずに出力します。

詳細

MMDDYY*w*.出力形式は、SAS 日付値を *mmdd<yy>yy* または *mm/dd/<yy>yy* 形式で書き出します。構文の説明は次のとおりです。

mm
月を表す整数です。

/
区切り文字です。

dd
月の日を表す整数です。

<yy>yy
年を表す 2 桁または 4 桁の整数です。

例

次の例では、入力値として 20999 を使用します。20999 は、2017 年 6 月 29 日に対応する SAS 日付値です。

```
data _null_;
  declare char(15) a b c d;
  method run();
    a=put(20999,mmddyy5.);
    b=put(20999,mmddyy8.);
    c=put(20999,mmddyy8.);
    d=put(20999,mmddyy10.);
    put a= b= c= d=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=06/29      b=06/29/17     c=06/29/17     d=06/29/2017
```

関連項目

概念:

- “DS2 式” (*SAS DS2 プログラマガイド*)

出力形式:

- “DATEw. 出力形式” (115 ページ)
- “DDMMYYw. 出力形式” (123 ページ)
- “MMDDYYxw. 出力形式” (173 ページ)
- “YYMMDDw. 出力形式” (217 ページ)

関数:

- “DAY 関数” (500 ページ)
- “MDY 関数” (847 ページ)
- “MONTH 関数” (863 ページ)
- “YEAR 関数” (1263 ページ)

MMDDYYxw. 出力形式

SAS 日付値を *mmdd<yy>yy* または *mm-dd-<yy>yy* 形式で書き出します。ここで、出力形式名の *x* は、月、日、年を区切る特殊文字を表す文字です。特殊文字は、ハイフン(-)、ピリオド(.)、空白文字、スラッシュ(/)、コロン(:)または区切り文字なしになります。年は 2 桁または 4 桁のいずれかになります。

カテゴリ: 日付と時間

配置: 右

構文

MMDDYY*xw*.

必須引数

x
区切り文字を示します。または、区切り文字を日、月、年の間に挿入しないことを示します。有効な値は次のとおりです。

B
空白で区切ります。

C
コロンで区切ります。

D
ハイフンで区切ります。

N
区切り文字なしを表します。

P
ピリオドで区切ります。

S
スラッシュで区切ります。

w
出力フィールドの幅を指定します。

デフォルト: 8

範囲: 2-10

操作: *w* の値が 2 から 5 までの場合、月と日が可能な範囲で表示されます。*w* が 7 の場合、日付の年は 2 桁で、区切り文字を使用せずに表示されます。

x の値が N の場合、幅の範囲は 2–8 に変化します。

詳細

MMDDYY*xw*.出力形式は、SAS 日付値を *mmdd<yy>yy* または *mm*X*dd*X*<yy>yy* 形式で書き出します。構文の説明は次のとおりです。

mm
月を表す整数です。

X
指定された区切り文字です。

dd
月の日を表す整数です。

<yy>yy
年を表す 2 桁または 4 桁の整数です。

例

次の例では、入力値として 20999 を使用します。20999 は、2017 年 6 月 29 日に対応する SAS 日付値です。

```
data _null_;
  declare char(15) a b c d;
  method run();
    a=put(20999,mmddyyc5.);
    b=put(20999,mmddyyd8.);
    c=put(20999,mmddyyn8.);
    d=put(20999,mmddyyp10.);
    put a= b= c= d=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=06:29      b=06-29-17     c=06292017     d=06.29.2017
```

関連項目

概念:

- “DS2 式” (*SAS DS2 プログラマガイド*)

出力形式:

- “DATEw. 出力形式” (115 ページ)
- “DDMMYYxw. 出力形式” (124 ページ)
- “MMDDYYxw. 出力形式” (173 ページ)
- “YYMMDDxw. 出力形式” (219 ページ)

関数:

- “DAY 関数” (500 ページ)
- “MDY 関数” (847 ページ)
- “MONTH 関数” (863 ページ)
- “YEAR 関数” (1263 ページ)

MMSSw.d 出力形式

SAS 時間値を午前 0 時から経過した分数および秒数として書き出します。

カテゴリ: 日付と時間

配置: 右

操作: DECIMALCONV=システムオプションを STDIEEE に設定した場合、この出力形式を使用して書き出される出力が前のリリースと多少異なる可能性があります。詳細については、“DECIMALCONV= システムオプション” (*SAS システムオプション: リファレンス*)を参照してください。

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

MMSS*w*.[*d*]

必須引数

w
出力フィールドの幅を指定します。

デフォルト 5

範囲 2-20

ヒント 分数と秒数を表す値を書き出すには、*w* に少なくとも 5 を指定します。

オプション引数

d
秒値の小数点以下の桁数を指定します。このため、SAS 時間値には秒の端数が含まれます。

範囲	0-19
制限事項	*w* より小さい数にします。

例

次のプログラムは、MMSS*w*.出力形式を示しています。

```
data _null_;
  declare char(15) a;
  method run();
    a=put(4530,mmss.);
    put a=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=75:30
```

関連項目

概念:

- “DS2 式” (*SAS DS2 プログラマガイド*)

出力形式:

- “HHMMw.d 出力形式” (161 ページ)
- “TIMEw.d 出力形式” (195 ページ)

関数:

- “HMS 関数” (693 ページ)
- “MINUTE 関数” (854 ページ)
- “SECOND 関数” (1126 ページ)

MMYYw. 出力形式

SAS 日付値を *mm*M<*yy*>*yy* 形式で書き出します。M は区切り文字を表し、年は 2 桁または 4 桁の数字で書き出されます。

カテゴリ:	日付と時間
配置:	右

構文

MMYY*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト 7

範囲 5-32

操作 *w* の値が 5 または 6 の場合、日付の年は下 2 桁でのみ表示されます。*w* が 7 以上の場合、日付の年は 4 桁で表示されます。

詳細

MMYY*w*.出力形式は、SAS 日付値を *mm*M<*yy*>*yy* 形式で書き出します。構文の説明は次のとおりです。

mm
月を表す整数です。

M
区切り文字です。

<*yy*>*yy*
年を表す 2 桁または 4 桁の整数です。

例

次の例では、入力値として 20999 を使用します。20999 は、2017 年 6 月 29 日に対応する SAS 日付値です。

```
data _null_;
  declare char(15) a b c d e;
  method run();
    a=put(20999,mmyy5.);
    b=put(20999,mmyy6.);
    c=put(20999,mmyy.);
    d=put(20999,mmyy7.);
    e=put(20999,mmyy10.);
    put a= b= c= d= e=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=06M17      b= 06M17      c=06M2017     d=06M2017     e=  06M2017
```

関連項目

概念:

- “DS2 式” (*SAS DS2 プログラマガイド*)

出力形式:

- “MMYYxw. 出力形式” (178 ページ)
- “YYMMw. 出力形式” (213 ページ)

MMYYxw. 出力形式

SAS 日付値を *mm*<*yy*>*yy* または *mm*-<*yy*>*yy* 形式で書き出します。出力形式名の *x* は、月と年を区切る特殊文字を表す文字で、ハイフン(-)、ピリオド(.)、空白文字、スラッシュ(/)、コロン(:)、または区切り文字なしになります。年は 2 桁または 4 桁のいずれかになります。

カテゴリ: 日付と時間

配置: 右

構文

MMYY*xw*.

必須引数

x
区切り文字を示します。または、区切り文字が月と年の間に挿入しないことを示します。有効な値は次のとおりです。

C
コロンで区切ります。

D
ハイフンで区切ります。

N
区切り文字なしを表します。

P
ピリオドで区切ります。

S
スラッシュで区切ります。

w
出力フィールドの幅を指定します。

デフォルト 7

範囲 5-32

操作 *x* を N に設定すると、区切り文字なしを指定します。この場合の幅範囲は 4–32 になり、デフォルトは 6 に変化します。

x の値が C、D、P または S で、*w* の値が 5 または 6 の場合、日付の年は下 2 桁のみ表示されます。*w* が 7 以上の場合、日付の年は 4 桁で表示されます。

x の値が *N* で、*w* の値が 4 または 5 の場合、日付の年は下 2 桁のみ表示されます。*x* の値が *N* で、*w* が 6 以上の場合、日付の年は 4 桁で表示されます。

詳細

MMYY*xw*.出力形式は、SAS 日付値を *mm*<*yy*>*yy* または *mm*X<*yy*>*yy* 形式で書き出します。構文の説明は次のとおりです。

mm
月を表す整数です。

X
指定された区切り文字です。

<*yy*>*yy*
年を表す 2 桁または 4 桁の整数です。

例

次の例では、入力値として 20999 を使用します。20999 は、2017 年 6 月 29 日に対応する SAS 日付値です。

```
data _null_;
  declare char(15) a b c d e;
  method run();
    a=put(20999,mmyyc5.);
    b=put(20999,mmyyd7.);
    c=put(20999,mmyyn4.);
    d=put(20999,mmyyp8.);
    e=put(20999,mmyys10.);
    put a= b= c= d= e=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=06:17      b=06-2017     c=0617       d= 06.2017    e=  06/2017
```

関連項目

概念:

- “DS2 式” (*SAS DS2 プログラマガイド*)

出力形式:

- “MMYYw. 出力形式” (176 ページ)
- “YYMMw. 出力形式” (213 ページ)

MONNAMEw. 出力形式

SAS 日付値を月の名前として書き出します。

カテゴリ: 日付と時間

配置: 右

構文

MONNAME*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト 9

範囲 1-32

ヒント MONNAME3.を使用すると、月名の最初の 3 文字が出力されます。

詳細

必要に応じて、出力形式の幅に合うように月名が切り捨てられます。

例

次の例では、入力値として 20999 を使用します。20999 は、2017 年 6 月 29 日に対応する SAS 日付値です。

```
data _null_;
  declare char(15) a b c;
  method run();
    a=put(20999,monname1.);
    b=put(20999,monname3.);
    c=put(20999,monname5.);
    put a= b= c=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=J          b=Jun          c= June
```

関連項目

概念:

- “DS2 式” (*SAS DS2 プログラマガイド*)

出力形式:

- “MONTHw. 出力形式” (181 ページ)

MONTHw. 出力形式

SAS 日付値を月として書き出します。

カテゴリ: 日付と時間

配置: 右

構文

MONTH*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト	2
範囲	1-32

詳細

MONTH*w*.出力形式は、SAS 日付値から月(1 から 12 まで)を書き出します。

例

次の例では、入力値として 20999 を使用します。20999 は、2017 年 6 月 29 日に対応する SAS 日付値です。

```
data _null_;
  declare char(15) a;
  method run();
    a=put(20999,month.);
    put a= ;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a= 6
```

関連項目

概念:

- “DS2 式” (*SAS DS2 プログラマガイド*)

出力形式:

- “MONNAMEw. 出力形式” (180 ページ)

MONYYw. 出力形式

SAS 日付値を月および年として *mmmyy* または *mmmyyyy* 形式で書き出します。

カテゴリ:	日付と時間
配置:	右

構文

MONYY*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト 5

範囲 5-7

詳細

MONYY*w*.出力形式は、SAS 日付値を *mmmyy* または *mmmyyyy* 形式で書き出します。構文の説明は次のとおりです。

mmm
月名の最初の 3 文字になります。

yy または *yyyy*
年を表す 2 桁または 4 桁の整数です。

比較

MONYY*w*.出力形式と DTMONYY*w*.出力形式は、どちらも日付値を書き出すという点でほぼ同じです。相違点は、MONYY*w*.が 入力として SAS 日付値を必要とし、DTMONYY*w*.は日時値が必要であることです。

例

次の例では、入力値として 20999 を使用します。20999 は、2011 年 12 月 24 日に相当する SAS 日付値です。

```
data _null_;
  declare char(15) a b;
  method run();
    a=put(20999,monyy5.);
    b=put(20999,monyy7.);
    put a= b=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=JUN17       b=JUN2017
```

関連項目

概念:

- “DS2 式” (*SAS DS2 プログラマガイド*)

出力形式:

- “DTMONYYw. 出力形式” (133 ページ)
- “DDMMYYw. 出力形式” (123 ページ)
- “MMDDYYw. 出力形式” (171 ページ)
- “YYMMDDw. 出力形式” (217 ページ)

関数:

- “MONTH 関数” (863 ページ)
- “YEAR 関数” (1263 ページ)

NEGPARENw.d 出力形式

負の数値をかっこで囲んで書き出します。

カテゴリ: 数値

配置: 右

操作: DECIMALCONV=システムオプションを STDIEEE に設定した場合、この出力形式を使用して書き出される出力が前のリリースと多少異なる可能性があります。詳細については、“DECIMALCONV= システムオプション” (*SAS システムオプション: リファレンス*)を参照してください。

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

NEGPAREN*w*.[*d*]

必須引数

w
出力フィールドの幅を指定します。

デフォルト 6

範囲 1-32

オプション引数

d
数値の小数点以下の桁数を指定します。

デフォルト 0

範囲 0-31

詳細

NEGPAREN*w.d* 出力形式は、出力値を右揃えにします。入力値が負の場合、NEGPAREN*w.d* は、指定したフィールドの幅が十分な場合は値をかっこで囲って出力を表示します。フィールドの幅が十分でない場合は、負の値を表すためにマイナス符号が使用されます。入力値が負でない場合、NEGPAREN*w.d* は、適切なカラム配置を確実にするために値を先頭および末尾の空白とともに表示します。値が正の場合でも、最後のカラムは右かっこ用に確保されます。

比較

NEGPAREN*w.d* 出力形式と COMMA*w.d* 出力形式は、各値を 3 桁ごとにカンマで区切るという点では同じです。

例

次のプログラムは、NEGPAREN*w.d* 出力形式を示しています。

```
data _null_;
  declare char(15) a b c d;
  method run();
    a=put(100,negparen6.);
    b=put(1000,negparen6.);
    c=put(-200,negparen6.);
    d=put(-2000,negparen8.);
    put a= b= c= d=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=  100       b= 1,000      c= (200)      d= (2,000)
```

NENGOw. 出力形式

SAS 日付値を *e.yymmdd* 形式で日本の日付として書き出します。

カテゴリ: 日付と時間

配置: 左

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

NENGO*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト 10

範囲 2-10

詳細

NENGO*w*.出力形式では、SAS 日付値を *e.yymmdd* 形式で書き出します。構文の説明は次のとおりです。

e
元号(明治、大正、昭和、平成、令和)の頭文字です。

yy
年を表す整数です。

mm
月を表す整数です。

dd
月の日を表す整数です。

幅が小さすぎる場合、SAS はピリオドを省略します。

例

次の例では、入力値として 20999 を使用します。20999 は、2017 年 6 月 29 日に相当する SAS 日付値です。

```
data _null_;
  declare char(15) x1 x2 x3 x4 x5;
  method run();
    x1=put(20999,nengo3.);
    x2=put(20999,nengo6.);
    x3=put(20999,nengo8.);
    x4=put(20999,nengo9.);
    x5=put(20999,nengo10.);
    put x1= x2= x3= x4= x5=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
x1=H29        x2=H29/06       x3=H.290629     x4=H29/06/29     x5=H.29/06/29
```

OCTALw. 出力形式

数値を 8 進表現に変換します。

カテゴリ: 数値

配置: 左

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

OCTAL*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト 3

範囲 1-24

詳細

必要に応じて、OCTAL*w*.出力形式は数値を整数に変換してから 8 進表現で表示します。

比較

OCTAL*w*. は、数値を 8 進表現に変換します。$OCTAL*w*.出力形式は、文字値を 8 進表現に変換します。

例

次のプログラムは、OCTAL*w*.出力形式を示しています。

```
data _null_;
  declare char(15) a;
  method run();
    a=put(3592, octal6.);
    put a=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=007010
```

関連項目

出力形式:

- “$OCTALw. 出力形式” (82 ページ)

PERCENTw.d 出力形式

数値を百分率として書き出します。

カテゴリ: 数値

配置: 右

操作: DECIMALCONV=システムオプションを STDIEEE に設定した場合、この出力形式を使用して書き出される出力が前のリリースと多少異なる可能性があります。詳細については、“DECIMALCONV= システムオプション” (*SAS システムオプション: リファレンス*)を参照してください。

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

PERCENT*w*.[*d*]

必須引数

w
出力フィールドの幅を指定します。

デフォルト 6

範囲 4-32

オプション引数

d
数値の小数点以下の桁数を指定します。

範囲 0-31

要件 *w* より小さい数にします。

詳細

PERCENT*w.d* 出力形式は、値を 100 で乗算した後に BEST*w.d* 出力形式を適用し、適用後の値の末尾にパーセント記号(%)を追加し、負の値をかっこで囲みます。
PERCENT*w.d* 出力形式では、値が負でない場合でも、パーセント記号とかっこの余地があります。

例

次のプログラムは、PERCENT*w.d* 出力形式を示しています。

```
data _null_;
```

```
    declare char(15) a b c;
    method run();
      a=put(0.1,percent10.);
      b=put(1.2,percent10.);
      c=put(-.05,percent10.);
      put a= b= c=;
    end;
  enddata;
  run;
```

SAS は次の出力をログに書き込みます。

```
a=    10%    b=   120%    c=(    5%)
```

関連項目

出力形式:

- “PERCENTNw.d 出力形式” (190 ページ)

PERCENTNw.d 出力形式

百分率を書き出します。負の値にはマイナス符号を使用します。

カテゴリ: 数値

配置: 右

操作: DECIMALCONV=システムオプションを STDIEEE に設定した場合、この出力形式を使用して書き出される出力が前のリリースと多少異なる可能性があります。詳細については、“DECIMALCONV= システムオプション” (*SAS システムオプション: リファレンス*)を参照してください。

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

PERCENTN*w*.[*d*]

必須引数

w
出力フィールドの幅を指定します。

デフォルト 6

範囲 4-32

オプション引数

d
数値の小数点以下の桁数を指定します。

範囲 0-31

要件 *w* より小さい数にします。

詳細

PERCENTN*w.d* 出力形式は、 負の値に 100 を乗算し、BEST*w.d* 出力形式を適用し、適用後の値の先頭と末尾にマイナス記号とパーセント記号(%)をそれぞれ追加します。PERCENTN*w.d* 出力形式では、値が負でない場合でも、パーセント記号とマイナス記号の余地があります。

比較

PERCENTN*w.d* 出力形式は、負の値にかっこではなくマイナス記号を使用して、百分率を生成します。PERCENT*w.d* 出力形式は、負の値にかっこを使用して、百分率を生成します。

例

次のプログラムは、PERCENTN*w.d* 出力形式を示しています。

```
data _null_;
  declare char(15) a b c d;
  method run();
    a=put(0.1,percentn.);
    b=put(1.2,percentn.);
    c=put(-.05,percentn.);
    d=put(-6.3,percentn.);
    put a= b= c= d=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=  10%       b= 120%       c=  -5%       d=-630%
```

関連項目

出力形式:

- “PERCENTw.d 出力形式” (188 ページ)

QTRw. 出力形式

SAS 日付値を年の四半期として書き出します。

カテゴリ: 日付と時間

配置: 右

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

QTR*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト 1

範囲 1-32

例

次の例では、入力値として 20999 を使用します。20999 は、2017 年 6 月 29 日に対応する SAS 日付値です。

```
data _null_;
  declare char(15) a;
  method run();
    a=put(20999,qtr.);
    put a=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=2
```

関連項目

概念:

- “DS2 式” (*SAS DS2 プログラマガイド*)

出力形式:

- “QTRRw. 出力形式” (193 ページ)

QTRRw. 出力形式

SAS 日付値を年の四半期としてローマ数字で書き出します。

カテゴリ: 日付と時間

配置: 右

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

QTRR*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト 3

範囲 3-32

例

次の例では、入力値として 20999 を使用します。20999 は、2017 年 6 月 29 日に対応する SAS 日付値です。

```
data _null_;
  declare char(15) a;
  method run();
    a=put(20999,qtrr.);
    put a=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a= II
```

関連項目

概念:

- “DS2 式” (*SAS DS2 プログラマガイド*)

出力形式:

- “QTRw. 出力形式” (192 ページ)

ROMANw. 出力形式

数値をローマ数字で書き出します。

カテゴリ:	数値
配置:	左
注:	2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

ROMAN*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト 6

範囲 2-32

詳細

ROMAN*w*.出力形式は、浮動小数点数を整数に切り捨ててから値を書き出します。

例

```
data _null_;
  declare char(15) a;
  method run();
    a=put(2017,roman.);
    put a=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=MMXVII
```

TIMEw.d 出力形式

SAS 時間値を時間、分、および秒として *hh:mm:ss.ss* 形式で書き出します。

カテゴリ: 日付と時間

配置: 右

操作: DECIMALCONV=システムオプションを STDIEEE に設定した場合、この出力形式を使用して書き出される出力が前のリリースと多少異なる可能性があります。詳細については、“DECIMALCONV= システムオプション” (*SAS システムオプション: リファレンス*)を参照してください。

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

TIME*w*.[*d*]

必須引数

w
出力フィールドの幅を指定します。

デフォルト 8

範囲 2-20

ヒント *w* には、必要な結果を生成するために十分な大きさを指定します。小数点以下 3 桁までをすべて含む時間値を取得するには、少なくとも 12 スペース(小数点の左側に 8 スペース、小数点自体に 1 スペース、秒の小数部分に 3 スペース)を指定する必要があります。

オプション引数

d
秒値の小数点以下の桁数を指定します。

デフォルト 0

範囲 0-19

要件 *w* より小さい数にします。

詳細

TIME*w.d* 出力形式は、SAS 時間値を *hh:mm:ss.ss* 形式で書き出します。構文の説明は次のとおりです。

hh
整数です。

注: *hh* が 1 桁の場合、TIME*w.d* はその桁の前に先頭の空白を置きます。たとえば、TIME*w.d.* 出力形式は、09:00 ではなく 9:00 と書き出します。

mm
00 から 59 の範囲の時間(分)です。

ss.ss
小数点以下の秒の端数を含む、00 から 59 の範囲の秒数です。

比較

TIME*w.d* 出力形式は HHMM*w.d* 出力形式と同じですが、TIME*w.d* は秒を含めます。

TIME*w.d* 出力形式と HHMM*w* は、1 桁の時間にはその先頭に空白を書き出します。TOD*w.d* 出力形式は、1 桁の時間にはその先頭にゼロを書き出します。

例

次の例では、入力値として 59083 を使用します。59083 は、午後 4:24:43 に相当する SAS 時間値です。

```
data _null_;
  declare varchar(15) a;
  method run();
    a=put(59083, time.);
    put a=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=16:24:43
```

関連項目

概念:

- “DS2 式” (*SAS DS2 プログラマガイド*)

出力形式:

- “HHMMw.d 出力形式” (161 ページ)
- “HOURw.d 出力形式” (164 ページ)
- “MMSSw.d 出力形式” (175 ページ)
- “TODw.d 出力形式” (200 ページ)

関数:

- “HOUR 関数” (699 ページ)
- “MINUTE 関数” (854 ページ)
- “SECOND 関数” (1126 ページ)
- “TIME 関数” (1187 ページ)

TIMEAMPMw.d 出力形式

SAS 時間値を時間、分、および秒として *hh:mm:ss.ss* 形式で AM または PM を付けて書き出します。

カテゴリ: 日付と時間

配置: 右

操作: DECIMALCONV=システムオプションを STDIEEE に設定した場合、この出力形式を使用して書き出される出力が前のリリースと多少異なる可能性があります。詳細については、“DECIMALCONV= システムオプション” (*SAS システムオプション: リファレンス*)を参照してください。

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

TIMEAMPM*w*.[*d*]

必須引数

w
出力フィールドの幅を指定します。

デフォルト 11

範囲 2-20

オプション引数

d
秒値の小数点以下の桁数を指定します。

デフォルト 0

範囲 0-19

要件 *w* より小さい数にします。

詳細

TIMEAMPM*w.d* 出力形式は、SAS 時間値を *hh:mm:ss.ss* 形式で AM または PM を付けて書き出します。構文の説明は次のとおりです。

hh
時間を表す整数です。

mm
分を表す整数です。

ss.ss
小数点以下 2 桁の秒数です。

23:59:59 PM を超える時間は、次の日として表示されます。

w には、必要な結果を生成するために十分な大きさを指定します。小数点以下 3 桁と AM または PM をすべて含む時間値を取得するには、少なくとも 11 スペース (*hh:mm:ss* PM)を指定する必要があります。*w* が 5 未満の場合、AM または PM だけが書き出されます。

比較

- TIMEAMPMM*w.d* 出力形式は TIMEM*w.d* 出力形式と同じですが、TIMEAMPMM*w.d* は時間の最後に AM または PM を出力します。
- TIME*w.d* は 23:59:59 PM を超える時間を書き出しますが、TIMEAMPM*w.d* は書き出しません。

例

次の例では、入力値として 59083 を使用します。59083 は、午後 4:24:43 に相当する SAS 時間値です。

```
data _null_;
  declare char(15) a b c d;
  method run();
    a=put(59083,timeampm3.);
    b=put(59083,timeampm5.);
    c=put(59083,timeampm7.);
    d=put(59083,timeampm11.);
    put a= b= c= d=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a= PM          b= 4 PM        c=4:24 PM      d= 4:24:43 PM
```

関連項目

概念:

- “DS2 式” (*SAS DS2 プログラマガイド*)

出力形式:

- “TIMEw.d 出力形式” (195 ページ)

TODw.d 出力形式

SAS 時間値と、SAS 日時値の時間部分を *hh:mm:ss.ss* 形式で書き出します。

カテゴリ: 日付と時間

配置: 右

操作: DECIMALCONV=システムオプションを STDIEEE に設定した場合、この出力形式を使用して書き出される出力が前のリリースと多少異なる可能性があります。詳細については、“DECIMALCONV= システムオプション” (*SAS システムオプション: リファレンス*)を参照してください。

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

TOD*w*.[*d*]

必須引数

w
出力フィールドの幅を指定します。

デフォルト 8

範囲 2-20

ヒント 指定した幅が十分な場合は、ゼロ時間に対しゼロが書き出されます(たとえば、02:30 または 00:30)。

オプション引数

d
秒値の小数点以下の桁数を指定します。

デフォルト	0
範囲	0-19
要件	*w* より小さい数にします。

詳細

TOD*w.d* 出力形式は、SAS 日時値を *hh:mm:ss.ss* 形式で書き出します。構文の説明は次のとおりです。

hh
時間を表す整数です。

mm
分を表す整数です。

ss.ss
小数点以下 2 桁の秒数です。

比較

TOD*w.d* 出力形式は、1 桁の時間にはその先頭にゼロを書き出します。TIME*w.d* 出力形式と HHMM*w.d* 出力形式は、1 桁の時間にはその先頭に空白を書き出します。

例

次のプログラムは、TOD*w.d* 出力形式を示しています。

```
data _null_;
  declare varchar(15) a b;
  method run();
    a=put(1850739623,tod9.);
    put a= ;
    b=put(1472049623, time.);
    put b=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a= 14:20:23
b=  408902
```

関連項目

概念:

- “DS2 式” (*SAS DS2 プログラマガイド*)

出力形式:

- “TIMEw.d 出力形式” (195 ページ)
- “TIMEAMPMw.d 出力形式” (198 ページ)

関数:

- “TIMEPART 関数” (1189 ページ)

VAXRBw.d 出力形式

VMS 形式のバイナリ実数(浮動小数点)を書き出します。

カテゴリ: 数値

配置: 右

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

VAXRB*w*.[*d*]

必須引数

w
出力フィールドの幅を指定します。

デフォルト 8

範囲 2-8

オプション引数

d
値を除算する 10 のべき乗を指定します。

デフォルト 0

範囲 0-31

詳細

VAXRB*w.d* 出力形式を使用して、ネイティブ VAX/VMS 浮動小数点数表記のデータを書き出します。

例

次の例では、VARBINARY データ型を使用して、結果が整数の 16 進表現になるようにします。

```
data _null_;
  declare varbinary(20) a;
  declare varchar(64) b;
  method run();
    a=put(1,vaxrb8.);
    put a=;
    b= put(a, $hex.);
    put b=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=
b=8040000000000000
```

w.d 出力形式

標準数値でデータを書き出します(1 バイト 1 桁)。

カテゴリ: 数値

配置: 右

別名: F*w.d*

操作: DECIMALCONV=システムオプションを STDIEEE に設定した場合、この出力形式を使用して書き出される出力が前のリリースと多少異なる可能性があります。詳細については、“DECIMALCONV= システムオプション” (*SAS システムオプション: リファレンス*)を参照してください。

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようにな

ります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

w.[*d*]

必須引数

w
出力フィールドの幅を指定します。

範囲 1-32

ヒント 必要に応じて、値、小数点、マイナス記号を書き出せる十分なスペースを指定します。

オプション引数

d
数値の小数点以下の桁数を指定します。

範囲 0-31

要件 *w* より小さい数にします。

ヒント *d* が 0、または *d* を指定しない場合、*w*.*d* は値を小数点なしで書き出します。

詳細

w.*d* 出力形式は、出力フィールドにおさまる近傍値になるように値を丸めます。*w*.*d* が小さすぎる場合、小数点揃えを BEST*w*.出力形式に合わせます。*w*.*d* 出力形式は、負の数には先頭のマイナス記号を付けて書き出します。また、*w*.*d* は、出力を右揃えにしてから書き出し、先頭に空白を挿入します。

比較

Z*w*.*d* 出力形式は、*w*.*d* 出力形式とほぼ同じですが、Z*w*.*d* は右揃えにした出力に空白ではなく 0 を挿入します。

例

次のプログラムは、*w*.*d* 出力形式を示しています。

```
data _null_;
  declare char(20) a;
  method run();
    a=put(23.45, 6.3);
    put a=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=23.450
```

関連項目

出力形式:

- “Zw.d 出力形式” (232 ページ)

WEEKDATEw. 出力形式

SAS 日付値を曜日と日付として *day-of-week*, *month-name dd*, *yy* (または *yyyy*)形式で書き出します。

カテゴリ: 日付と時間

配置: 右

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

WEEKDATE*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト 29

範囲 3-37

詳細

WEEKDATE*w*.出力形式は、SAS 日付値を *day-of-week*, *month-name dd*, *yy* (または *yyyy*)形式で書き出します。構文の説明は次のとおりです。

dd
月の日を表す整数です。

yy または *yyyy*
年を表す 2 桁または 4 桁の整数です。

w が小さすぎて曜日と月の名前を完全な形で書き出すことができない場合、必要に応じて短縮形で書き出します。

比較

WEEKDATE*w*.出力形式は WEEKDATX*w*.出力形式とほぼ同じですが、WEEKDATX*w*. は月名の前に *dd* を出力します。

例

次の例では、入力値として 20999 を使用します。20999 は、2017 年 6 月 29 日に対応する SAS 日付値です。

```
data _null_;
  declare varchar(30) a b c d;
  method run();
    a=put(20999,weekdate3.);
    b=put(20999,weekdate9.);
    c=put(20999,weekdate15.);
    d=put(20999,weekdate17.);
    put a= b= c= d=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=Thu b= Thursday c=Thu, Jun 29, 17 d=Thu, Jun 29, 2017
```

関連項目

概念:

- “DS2 式” (*SAS DS2 プログラマガイド*)

出力形式:

- “DTWKDATXw. 出力形式” (135 ページ)

- “DATEw. 出力形式” (115 ページ)
- “DDMMYYw. 出力形式” (123 ページ)
- “MMDDYYw. 出力形式” (171 ページ)
- “TODw.d 出力形式” (200 ページ)
- “WEEKDATXw. 出力形式” (207 ページ)
- “YYMMDDw. 出力形式” (217 ページ)

関数:

- “JULDATE 関数” (789 ページ)
- “MDY 関数” (847 ページ)
- “WEEKDAY 関数” (1259 ページ)

WEEKDATXw. 出力形式

SAS 日付値を曜日と日付として *day-of-week*, *dd month-name yy* (または *yyyy*)形式で書き出します。

カテゴリ: 日付と時間

配置: 右

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

WEEKDATX*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト 29

範囲 3-37

詳細

WEEKDATX*w*.出力形式は、SAS 日付値を *day-of-week*, *dd month-name*, *yy* (または *yyyy*)形式で書き出します。構文の説明は次のとおりです。

dd
月の日を表す整数です。

yy または *yyyy*
年を表す 2 桁または 4 桁の整数です。

w が小さすぎて曜日と月の名前を完全な形で書き出すことができない場合、必要に応じて短縮形で書き出します。

比較

WEEKDATE*w*.出力形式は WEEKDATX*w*.出力形式とほぼ同じですが、WEEKDATE*w*. は月名の後に *dd* を出力します。

DTWKDATX*w*.出力形式は WEEKDATX*w*.出力形式とほぼ同じですが、DTWKDATX*w*. は入力として日時値が必要です。

例

次の例では、入力値として 20999 を使用します。20999 は、2017 年 6 月 29 日に対応する SAS 日付値です。

```
data _null_;
  declare varchar(30) a;
  method run();
    a=put(20999,weekdatx.);
    put a= ;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=      Thursday, 29 June 2017
```

関連項目

概念:

- “DS2 式” (*SAS DS2 プログラマガイド*)

出力形式:

- “DTWKDATXw. 出力形式” (135 ページ)

- “DATEw. 出力形式” (115 ページ)
- “DDMMYYw. 出力形式” (123 ページ)
- “MMDDYYw. 出力形式” (171 ページ)
- “TODw.d 出力形式” (200 ページ)
- “WEEKDATEw. 出力形式” (205 ページ)
- “YYMMDDw. 出力形式” (217 ページ)

関数:

- “JULDATE 関数” (789 ページ)
- “MDY 関数” (847 ページ)
- “WEEKDAY 関数” (1259 ページ)

WEEKDAYw. 出力形式

SAS 日付値から曜日の値を書き出します。

カテゴリ: 日付と時間

配置: 右

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

WEEKDAY*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト 1

範囲 1-32

詳細

WEEKDAY*w*.出力形式は、SAS 日付値を曜日の値(1=日曜日、2=月曜日など)として書き出します。

例

次の例では、入力値として 20999 を使用します。20999 は、2017 年 6 月 29 日に相当する SAS 日付値です。

```
data _null_;
  declare char(20) a;
  method run();
    a=put(20999,weekday.);
    put a= ;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=5
```

関連項目

概念:

- “DS2 式” (*SAS DS2 プログラマガイド*)

出力形式:

- “DOWNAMEw. 出力形式” (130 ページ)

YEARw. 出力形式

SAS 日付値から年部分を書き出します。

カテゴリ:	日付と時間
配置:	右
注:	2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

YEAR*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト 4

範囲 2-32

ヒント *w* が 4 より小さい場合、年の最後の 2 桁が出力されます。それ以外の場合、年値は 4 桁で出力されます。

比較

YEAR*w*.出力形式と DTYEAR*w*.出力形式は、どちらも日付値を書き出すという点でほぼ同じです。相違点は、YEAR*w*.が入力として SAS 日付値を必要とし、DTYEAR*w*.は SAS 日時値が必要であることです。

例

次の例では、入力値として 20999 を使用します。20999 は、2017 年 6 月 29 日に対応する SAS 日付値です。

```
data _null_;
  declare varchar(20) a b;
  method run();
    a=put(20999,year2.);
    b=put(20999,year4.);
    put a= b=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=17 b=2017
```

関連項目

概念:

- “DS2 式” (*SAS DS2 プログラマガイド*)

出力形式:

- “DTYEARw. 出力形式” (137 ページ)

YENw.d 出力形式

円記号、カンマ、小数点を使用して数値を書き込みます。

カテゴリ: 数値

配置: 右

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

YEN*w.d*

必須引数

w
出力フィールドの幅を指定します。

デフォルト 8

範囲 1-32

d
数値の小数点以下の桁数を指定します。

制限事項 0 または 2 のいずれかである必要があります

ヒント *d* が 2 の場合、YEN*w.d* は小数点と小数点以下 2 桁を書き込みます。*d* が 0 の場合、YEN*w.d* は小数点と小数点以下の桁を書き込みません。

詳細

YEN*w.d* 出力形式は、先頭に円記号を付け、各値の 3 桁ごとにカンマで区切って数値を書き込みます。

円記号文字のコードの 16 進表現は、EBCDIC のシステムでは 5B、ASCII のシステムでは 5C です。これらのコードが表す通貨記号は、他の国では異なる場合があります。

例

次のプログラムは、YEN*w.d* 出力形式を示しています。

```
data _null_;
  declare char(20) a ;
  method run();
    a=put(1254.71,yen10.2);
    put a= ;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a= ¥1,254.71
```

YYMMw. 出力形式

SAS 日付値を<*yy*>*yy*M*mm* 形式で書き出します。M は月番号が M の後に続くことを示す区切り文字で、年は 2 桁または 4 桁の数字で書き出されます。

カテゴリ: 日付と時間

配置: 右

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

YYMM*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト 7

範囲 5-32

操作 *w* の値が 5 または 6 の場合、日付の年は下 2 桁でのみ表示されます。*w* が 7 以上の場合、日付の年は 4 桁で表示されます。

詳細

YYMM*w*.出力形式は、SAS 日付値を<*yy*>*yy*M*mm* 形式で書き出します。構文の説明は次のとおりです。

<*yy*>*yy*
年を表す 2 桁または 4 桁の整数です。

M
区切り文字です。

mm
月を表す整数です。

比較

- YYMM*w.d* 出力形式は YYMM*xw.d* 出力形式とほぼ同じですが、YYMM*xw.d* 出力形式では年と月の間にハイフン、コロン、スラッシュ、ピリオドなどの区切り文字が含まれている点が異なります。

例

次の例では、入力値として 20999 を使用します。20999 は、2017 年 6 月 29 日に対応する SAS 日付値です。

```
data _null_;
  declare varchar(20) a b c d e ;
  method run();
    a=put(20999,yymm5.);
    b=put(20999,yymm6.);
    c=put(20999,yymm.);
    d=put(20999,yymm7.);
    e=put(20999,yymm10.);
    put a= b= c= d= e=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=17M06 b= 17M06 c=2017M06 d=2017M06 e=   2017M06
```

関連項目

概念:

- “DS2 式” (*SAS DS2 プログラマガイド*)

出力形式:

- “MMYYw. 出力形式” (176 ページ)
- “YYMMxw. 出力形式” (215 ページ)

YYMMxw. 出力形式

SAS 日付値を*<yy>yymm* または*<yy>yy-mm* 形式で書き出します。出力形式名の *x* は、年と月を区切る特殊文字を表す文字で、ハイフン(-)、ピリオド(.)、空白文字、スラッシュ(/)、コロン(:)、または区切り文字なしになります。年は 2 桁または 4 桁の数字で書き出されます。

カテゴリ: 日付と時間

配置: 右

構文

YYMM*xw*.

必須引数

x
区切り文字を示します。または、区切り文字を年と月の間に挿入しないことを示します。有効な値は次のとおりです。

C
コロンで区切ります。

D
ハイフンで区切ります。

N
区切り文字なしを表します。

P
ピリオドで区切ります。

S
スラッシュで区切ります。

w
出力フィールドの幅を指定します。

デフォルト 7

範囲 5-32

操作 *x* を *N* に設定すると、区切り文字なしを指定します。この場合の幅範囲は 4–32 になり、デフォルトは 6 に変化します。

x の値が C、D、P または S で、*w* の値が 5 または 6 の場合、日付の年は下 2 桁のみ表示されます。*w* が 7 以上の場合、日付の年は 4 桁で表示されます。

x の値が *N* で、*w* の値が 4 または 5 の場合、日付の年は下 2 桁のみ表示されます。*x* の値が *N* で、*w* が 6 以上の場合、日付の年は 4 桁で表示されます。

詳細

YYMM*xw*.出力形式は、SAS 日付値を*<yy>yymm* または*<yy>yyXmm* 形式で書き出します。構文の説明は次のとおりです。

<yy>yy
年を表す 2 桁または 4 桁の整数です。

X
指定された区切り文字です。

mm
月を表す整数です。

比較

- YYMM*w.d* 出力形式は YYMM*xw.d* 出力形式とほぼ同じですが、YYMM*xw.d* 出力形式では年と月の間にハイフン、コロン、スラッシュ、ピリオドなどの区切り文字が含まれている点が異なります。

例

次の例では、入力値として 20999 を使用します。20999 は、2017 年 6 月 29 日に対応する SAS 日付値です。

```
data _null_;
  declare varchar(20) a b c d e;
  method run();
    a=put(20999,yymmc5.);
    b=put(20999,yymmd.);
    c=put(20999,yymmn4.);
    d=put(20999,yymmp8.);
    e=put(20999,yymms10.);
    put a= b= c= d= e=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=17:06 b=2017-06 c=1706 d= 2017.06 e=  2017/06
```

関連項目

概念:

- “DS2 式” (*SAS DS2 プログラマガイド*)

出力形式:

- “MMYYxw. 出力形式” (178 ページ)
- “YYMMw. 出力形式” (213 ページ)

YYMMDDw. 出力形式

SAS 日付値を*<yy>yymmdd* または*<yy>yy–mm–dd* 形式で書き出します。ハイフンが区切り文字として使用され、年は 2 桁または 4 桁の数字で書き出されます。

カテゴリ: 日付と時間

配置: 右

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

YYMMDD*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト 8

範囲 2-10

操作 *w* の値が 2 から 5 までの場合、日付の年と月が可能な範囲で表示されます。*w* が 7 の場合、日付は 2 桁の年としてハイフンなしで表示されます。

詳細

YYMMDD*w*.出力形式は、SAS 日付値を*<yy>yymmdd* または*<yy>yy–mm–dd* 形式で書き出します。構文の説明は次のとおりです。

<yy>yy
年を表す 2 桁または 4 桁の整数です。

–
区切り文字です。

mm
月を表す整数です。

dd
月の日を表す整数です。

比較

- YYMMDD*w.d* 出力形式は、YYMMDD*xw.d* 出力形式とほぼ同じですが、YYMMDD*xw.d* 出力形式では年、月、日の間にコロン、スラッシュ、ピリオドなどの区切り文字が含まれている点が異なります。

例

次の例では、入力値として 20999 を使用します。20999 は、2017 年 6 月 29 日に対応する SAS 日付値です。

```
ddata _null_;
  declare varchar(20) a b c d e f g h;
  method run();
    a=put(20999,yymmdd2.);
    b=put(20999,yymmdd3.);
    c=put(20999,yymmdd4.);
    d=put(20999,yymmdd5.);
    e=put(20999,yymmdd6.);
    f=put(20999,yymmdd7.);
    g=put(20999,yymmdd8.);
    h=put(20999,yymmdd10.);
    put a= b= c= d= e= f= g= h=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=17 b= 17 c=1706 d=17-06 e=170629 f= 170629 g=17-06-29 h=2017-06-29
```

関連項目

概念:

- “DS2 式” (*SAS DS2 プログラマガイド*)

出力形式:

- “DATEw. 出力形式” (115 ページ)
- “DDMMYYw. 出力形式” (123 ページ)
- “MMDDYYw. 出力形式” (171 ページ)
- “YYMMDDxw. 出力形式” (219 ページ)

関数:

- “DAY 関数” (500 ページ)
- “MDY 関数” (847 ページ)
- “MONTH 関数” (863 ページ)
- “YEAR 関数” (1263 ページ)

YYMMDDxw. 出力形式

SAS 日付値を*<yy>yymmdd* または*<yy>yy-mm-dd* 形式で書き出します。出力形式名の *x* は、年、月、日を区切る特殊文字を表す文字です。特殊文字は、ハイフン(-)、ピリオド(.)、空白文字、スラッシュ(/)、コロン(:)または区切り文字なしになります。年は 2 桁または 4 桁のいずれかになります。

カテゴリ: 日付と時間

配置: 右

構文

YYMMDD*xw*.

必須引数

x
区切り文字を示します。または、区切り文字を年、月、日の間に挿入しないことを示します。有効な値は次のとおりです。

B
空白で区切ります。

C
コロンで区切ります。

D
ハイフンで区切ります。

N
区切り文字なしを表します。

P
ピリオドで区切ります。

S
スラッシュで区切ります。

w
出力フィールドの幅を指定します。

デフォルト	8
範囲	2-10
操作	*w* の値が 2 から 5 までの場合、日付の年と月が可能な範囲で表示されます。*w* が 7 の場合、日付の年は 2 桁で、区切り文字を使用せずに表示されます。 *x* の値が N の場合、幅の範囲は 2–8 に変化します。

詳細

YYMMDD*xw*.出力形式は、SAS 日付値を*<yy>yymmdd* または*<yy>yyXmmXdd* 形式で書き出します。構文の説明は次のとおりです。

<yy>yy
年を表す 2 桁または 4 桁の整数です。

X
指定された区切り文字です。

mm
月を表す整数です。

dd
月の日を表す整数です。

比較

- YYMMDD*w.d* 出力形式は YYMMDD*xw.d* 出力形式とほぼ同じですが、YYMMDD*xw.d* 出力形式には、年、月、日の間にコロン、スラッシュ、ピリオドなどの区切り文字が含まれています。

例

次の例では、入力値として 20999 を使用します。20999 は、2017 年 6 月 29 日に対応する SAS 日付値です。

```
data _null_;
  declare varchar(20) a b c d;
  method run();
    a=put(20999,yymmddc5.);
    b=put(20999,yymmddd8.);
    c=put(20999,yymmddn8.);
    d=put(20999,yymmddp10.);
    put a= b= c= d=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=17:06 b=17-06-29 c=20170629 d=2017.06.29
```

関連項目

概念:

- “DS2 式” (*SAS DS2 プログラマガイド*)

出力形式:

- “DATEw. 出力形式” (115 ページ)
- “DDMMYYxw. 出力形式” (124 ページ)
- “MMDDYYxw. 出力形式” (173 ページ)
- “YYMMDDw. 出力形式” (217 ページ)

関数:

- “DAY 関数” (500 ページ)
- “MDY 関数” (847 ページ)
- “MONTH 関数” (863 ページ)
- “YEAR 関数” (1263 ページ)

YYMONw. 出力形式

SAS 日付値を *yymmm* または *yyyymmm* 形式で書き出します。

カテゴリ: 日付と時間

配置: 右

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

YYMON*w*.

必須引数

w
出力フィールドの幅を指定します。出力形式の幅が小さすぎて 4 桁の年を出力できない場合、年の下 2 桁だけが出力されます。

デフォルト 7

範囲 5-32

詳細

YYMON*w*.出力形式では、月名が 3 文字に省略されます。

例

次の例では、入力値として 20999 を使用します。20999 は、2017 年 6 月 29 日に対応する SAS 日付値です。

```
data _null_;
  declare char(20) a ;
  method run();
    a=put(20999,yymon.);
    put a= ;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=2017JUN
```

関連項目

概念:

- “DS2 式” (*SAS DS2 プログラマガイド*)

出力形式:

- “MMYYw. 出力形式” (176 ページ)

YYQw. 出力形式

SAS 日付値を*<yy>yyQq* 形式で書き出します。Q は区切り文字として使用され、年は 2 桁または 4 桁の数字で書き出されます。また、*q* は四半期を表します。

カテゴリ: 日付と時間

配置: 右

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようになります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

YYQ*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト 6

範囲 4-32

操作 *w* の値が 4 または 5 の場合、日付の年は下 2 桁でのみ表示されます。*w* が 6 以上の場合、日付の年は 4 桁で表示されます。

詳細

YYQ*w*.出力形式は、SAS 日付値を*<yy>yyQq* 形式で書き出します。構文の説明は次のとおりです。

<yy>yy
年を表す 2 桁または 4 桁の整数です。

Q
区切り文字です。

q
四半期を表す整数(1、2、3、4 のいずれか)です。

比較

YYQ*w*.出力形式は YYQ*xw*.出力形式とほぼ同じですが、YYQ*xw*.出力形式には、YY と Q の間にハイフン、スラッシュ、ピリオド、コロンなどの区切り文字があります。

例

次の例では、入力値として 20999 を使用します。20999 は、2017 年 6 月 29 日に対応する SAS 日付値です。

```
data _null_;
  declare char(20) a b c d e;
  method run();
    a=put(20999,yyq4.);
    b=put(20999,yyq5.);
    c=put(20999,yyq.);
    d=put(20999,yyq6.);
    e=put(20999,yyq10.);
    put a= b= c= d= e=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=17Q2          b= 17Q2          c=2017Q2
d=2017Q2          e=  2017Q2
```

関連項目

概念:

- “DS2 式” (*SAS DS2 プログラマガイド*)

出力形式:

- “YYQxw. 出力形式” (225 ページ)
- “YYQRw. 出力形式” (227 ページ)
- “YYQRxw. 出力形式” (229 ページ)

- “YYQZw. 出力形式” (231 ページ)

YYQxw. 出力形式

SAS 日付値を<*yy*>*yyq* または<*yy*>*yy-q* 形式で書き出します。出力形式名の *x* は、年と四半期を区切る特殊文字を表す文字で、ハイフン(-)、ピリオド(.)、空白文字、スラッシュ(/)、コロン(:)、または区切り文字なしになります。年は 2 桁または 4 桁の数字で書き出されます。

カテゴリ: 日付と時間

配置: 右

構文

YYQ*xw*.

必須引数

x
区切り文字を示します。または、区切り文字を年と四半期の間に挿入しないことを示します。有効な値は次のとおりです。

C
コロンで区切ります。

D
ハイフンで区切ります。

N
区切り文字なしを表します。

P
ピリオドで区切ります。

S
スラッシュで区切ります。

w
出力フィールドの幅を指定します。

デフォルト 6

範囲 4-32

操作 *x* を *N* に設定すると、区切り文字なしを指定します。この場合の幅範囲は 3–32 になり、デフォルトは 5 に変化します。

w の値が 4 または 5 の場合、日付の年は下 2 桁でのみ表示されます。*w* が 6 以上の場合、日付の年は 4 桁で表示されます。

x の値が *N* で、*w* の値が 3 または 4 の場合、日付の年は下 2 桁のみ表示されます。*x* の値が *N* で、*w* が 5 以上の場合、日付の年は 4 桁で表示されます。

詳細

YYQ*xw*.出力形式は、SAS 日付値を*<yy>yyq* または*<yy>yy*X*q* 形式で書き出します。構文の説明は次のとおりです。

<yy>yy
年を表す 2 桁または 4 桁の整数です。

X
指定された区切り文字です。

q
四半期を表す整数(1、2、3、4 のいずれか)です。

比較

YYQ*w*.出力形式は YYQ*xw*.出力形式とほぼ同じですが、YYQ*xw*.出力形式には、YY と Q の間にハイフン、スラッシュ、ピリオド、コロンなどの区切り文字があります。

例

次の例では、入力値として 20999 を使用します。20999 は、2017 年 6 月 29 日に対応する SAS 日付値です。

```
data _null_;
  declare char(20) a b c d e;
  method run();
    a=put(20999,yyqc4.);
    b=put(20999,yyqd.);
    c=put(20999,yyqn3.);
    d=put(20999,yyqp6.);
    e=put(20999,yyqs8.);
    put a= b= c= d= e=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=17:2          b=2017-2        c=172
d=2017.2          e= 2017/2
```

関連項目

概念:

- “DS2 式” (*SAS DS2 プログラマガイド*)

出力形式:

- “YYQw. 出力形式” (223 ページ)
- “YYQRw. 出力形式” (227 ページ)
- “YYQRxw. 出力形式” (229 ページ)
- “YYQZw. 出力形式” (231 ページ)

YYQRw. 出力形式

SAS 日付値を<*yy*>*yy*Q*qr* 形式で書き出します。Q は区切り文字として使用され、年は 2 桁または 4 桁の数字で書き出されます。また *qr* はローマ数字表現の四半期を表します。

カテゴリ: 日付と時間

配置: 右

構文

YYQR*w*.

必須引数

w
出力フィールドの幅を指定します。

デフォルト 8

範囲 6-32

操作 *w* の値が小さすぎて 4 桁の年を書き出せない場合、日付の年は下 2 桁でのみ表示されます。

詳細

YYQR*w*.出力形式は、SAS 日付値を<*yy*>*yy*Q*qr* 形式で書き出します。構文の説明は次のとおりです。

<*yy*>*yy*
年を表す 2 桁または 4 桁の整数です。

Q
区切り文字です。

qr
四半期を表すローマ数字表現(I、II、III、IV のいずれか)です。

比較

YYQR*w*.出力形式は YYQR*xw*.出力形式とほぼ同じですが、YYQR*xw*.出力形式には、YY と QR の間にハイフン、スラッシュ、ピリオド、コロンなどの区切り文字があります。

例

次の例では、入力値として 20999 を使用します。20999 は、2017 年 6 月 29 日に対応する SAS 日付値です。

```
 data _null_;
  declare char(20) a b c d e;
  method run();
    a=put(20999,yyqr6.);
    b=put(20999,yyqr7.);
    c=put(20999,yyqr.);
    d=put(20999,yyqr8.);
    e=put(20999,yyqr10.);
    put a= b= c= d= e=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a= 17QII        b=2017QII       c= 2017QII      d= 2017QII
e=  2017QII
```

関連項目

概念:

- “DS2 式” (*SAS DS2 プログラマガイド*)

出力形式:

- “YYQw. 出力形式” (223 ページ)
- “YYQRxw. 出力形式” (229 ページ)
- “YYQxw. 出力形式” (225 ページ)
- “YYQZw. 出力形式” (231 ページ)

YYQRxw. 出力形式

SAS 日付値を<*yy*>*yyqr* または<*yy*>*yy-qr* 形式で書き出します。出力形式名の *x* は、年と四半期を区切る特殊文字を表す文字で、ハイフン(-)、ピリオド(.)、空白文字、スラッシュ(/)、コロン(:)、または区切り文字なしになります。年は 2 桁または 4 桁の数字で書き出されます。また *qr* は、ローマ数字表現の四半期です。

カテゴリ: 日付と時間

配置: 右

構文

YYQR*xw*.

必須引数

x
区切り文字を示します。または、区切り文字を年と四半期の間に挿入しないことを示します。有効な値は次のとおりです。

C
コロンで区切ります。

D
ハイフンで区切ります。

N
区切り文字なしを表します。

P
ピリオドで区切ります。

S
スラッシュで区切ります。

w
出力フィールドの幅を指定します。

デフォルト 8

範囲 6-32

操作 *x* を *N* に設定すると、区切り文字なしを指定します。この場合の幅範囲は 5–32 になり、デフォルトは 7 に変化します。

w の値が小さすぎて 4 桁の年を書き出せない場合、日付の年は下 2 桁でのみ表示されます。

詳細

YYQR*xw*.出力形式は、SAS 日付値を*<yy>yyqr* または*<yy>yyXqr* 形式で書き出します。構文の説明は次のとおりです。

<yy>yy
年を表す 2 桁または 4 桁の整数です。

X
指定された区切り文字です。

qr
四半期を表すローマ数字表現(I、II、III、IV のいずれか)です。

比較

YYQR*w*.出力形式は YYQR*xw*.出力形式とほぼ同じですが、YYQR*xw*.出力形式には、YY と QR の間にハイフン、スラッシュ、ピリオド、コロンなどの区切り文字があります。

例

次の例では、入力値として 20999 を使用します。20999 は、2017 年 6 月 29 日に対応する SAS 日付値です。

```
data _null_;
  declare char(20) a b c d e;
  method run();
    a=put(18985,yyqrc6.);
    b=put(20999,yyqrd.);
    c=put(20999,yyqrn5.);
    d=put(20999,yyqrp8.);
    e=put(20999,yyqrs10.);
    put a= b= c= d= e=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a= 11:IV          b= 2017-II        c= 17II
d= 2017.II        e=  2017/II
```

関連項目

概念:

- “DS2 式” (*SAS DS2 プログラマガイド*)

出力形式:

- “YYQxw. 出力形式” (225 ページ)
- “YYQRw. 出力形式” (227 ページ)
- “YYQw. 出力形式” (223 ページ)
- “YYQZw. 出力形式” (231 ページ)

YYQZw. 出力形式

SAS 日付値を<*yy*><*qq*>形式で書き出します。年は 2 桁または 4 桁の数字で書き出され、*qq* は四半期を表します。

カテゴリ: 日付と時間

配置: 右

構文

YYQZ*w*.

必須引数

Z
区切り文字を年と四半期の間に挿入しないことを示します。

w
出力フィールドの幅を指定します。

デフォルト 4

範囲 4-6

詳細

YYQZ*w*.出力形式は、SAS 日付値を<*yy*> <*qq*>形式で書き出します。構文の説明は次のとおりです。

<*yy*>
年を表す 2 桁または 4 桁の整数です。

Z
区切り文字がないことを示します。

<*qq*>
四半期を表す整数(01、02、03、04 のいずれか)です。

例

次の例では、入力値として 20999 を使用します。20999 は、2017 年 6 月 29 日に対応する SAS 日付値です。

```
data _null_;
  declare char(20) a b;
  method run();
    a=put (20999,yyqz6.);
    b=put (20999,yyqz4.);
    put a= b=;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=201702          b=1702
```

関連項目

概念:

- “DS2 式” (*SAS DS2 プログラマガイド*)

出力形式:

- “YYQw. 出力形式” (223 ページ)
- “YYQxw. 出力形式” (225 ページ)
- “YYQRw. 出力形式” (227 ページ)
- “YYQRxw. 出力形式” (229 ページ)

Zw.d 出力形式

先頭に 0 を含む標準数値データを書き出します。

カテゴリ: 数値

配置: 右

操作: DECIMALCONV=システムオプションを STDIEEE に設定した場合、この出力形式を使用して書き出される出力が前のリリースと多少異なる可能性があります。詳細については、“DECIMALCONV= システムオプション” (*SAS システムオプション: リファレンス*)を参照してください。

注: 2025.12 以降、この形式は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINTPROCESSING=YES を設定すると、大きい整数値(BIGINT として保存)が丸めや切り捨てを行うことなく 19 桁までの完全な精度で表示されるようにな

ります。15 桁または 16 桁を超える大きな値の場合は、BIGINTPROCESSING=YES をお勧めします。

構文

Z*w*.[*d*]

必須引数

w
出力フィールドの幅を指定します。

デフォルト 1

範囲 1-32

ヒント 必要に応じて、値、小数点、マイナス記号を書き出せる十分なスペースを指定します。

オプション引数

d
数値の小数点以下の桁数を指定します。

デフォルト 0

範囲 0-31

ヒント *d* が 0 の場合、または *d* の指定を省略する場合、Z*w.d* は値を小数点なしで書き出します。

詳細

Z*w.d* 出力形式は、標準数値(1 バイト 1 桁)を書き出し、データ値の左側に 0 を入力します。

Z*w.d* 出力形式は、出力フィールドにおさまる近傍値になるように値を丸めます。*w.d* が大きすぎておさまらない場合、小数点揃えを BESTw.出力形式に合わせます。Z*w.d* 出力形式は、負の数には先頭のマイナス記号を付けて書き出します。また、出力を右揃えしてから書き出し、先頭にゼロを挿入します。

例

次のプログラムは、Z*w.d* 出力形式を示しています。

```
data _null_;
  declare char(20) a ;
```

```
    method run();
      a=put(1350,z8.);
      put a= ;
    end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
a=00001350
```

7

DS2 関数

DS2 関数の概要 **244**

一般的な関数構文 **244**

関数の使用 **246**
- 関数の引数の制限 246
- 関数でのデータ型変換 246
- DS2 数値関数の欠損値と null 値 246
- DS2 文字関数の欠損値と null 値 246
- システム式を使用して関数を実行する 246
- 記述統計関数に関する注記 246
- 財務関数に関する注記 246

DS2 関数の例 **250**

関数カテゴリ **250**

ディクショナリ **273**
- ABS 関数 273
- AIRY 関数 275
- ANYALNUM 関数 276
- ANYALPHA 関数 279
- ANYCNTRL 関数 282
- ANYDIGIT 関数 284
- ANYFIRST 関数 286
- ANYGRAPH 関数 289
- ANYLOWER 関数 292
- ANYNAME 関数 294
- ANYPRINT 関数 296
- ANYPUNCT 関数 299
- ANYSPACE 関数 302
- ANYUPPER 関数 304
- ANYXDIGIT 関数 307
- ARCOS 関数 309
- ARCOSH 関数 310
- ARSIN 関数 312
- ARSINH 関数 313
- ARTANH 関数 314
- ATAN 関数 316
- ATAN2 関数 317

BAND 関数 . . . 319
BETA 関数 . . . 320
BETAINV 関数 . . . 322
BHAMMING_32 関数 . . . 324
BHAMMING_HEX 関数 . . . 325
BLACKCLPRC 関数 . . . 327
BLACKPTPRC 関数 . . . 329
BLKSHCLPRC 関数 . . . 332
BLKSHPTPRC 関数 . . . 334
BLSHIFT 関数 . . . 337
BNOT 関数 . . . 338
BOR 関数 . . . 339
BRSHIFT 関数 . . . 340
BXOR 関数 . . . 342
BYTE 関数 . . . 343
CAT 関数 . . . 344
CATQ 関数 . . . 346
CATS 関数 . . . 351
CATT 関数 . . . 353
CATX 関数 . . . 356
CDF 関数 . . . 359
CDF BERNOULLI 分布 関数 . . . 362
CDF BETA 分布 関数 . . . 363
CDF BINOMIAL 分布 関数 . . . 366
CDF CAUCHY 分布 関数 . . . 368
CDF カイ 2 乗分布 関数 . . . 370
CDF Conway-Maxwell-Poisson 分布 関数 . . . 372
CDF 指数分布 関数 . . . 373
CDF F 分布 関数 . . . 375
CDF GAMMA 分布 関数 . . . 377
CDF 一般化 Poisson 分布 関数 . . . 379
CDF GEOMETRIC 分布 関数 . . . 381
CDF HYPERGEOMETRIC 分布 関数 . . . 383
CDF LAPLACE 分布 関数 . . . 385
CDF LOGISTIC 分布 関数 . . . 387
CDF LOGNORMAL 分布 関数 . . . 389
CDF NEGBINOMIAL 分布 関数 . . . 391
CDF NORMAL 分布 関数 . . . 393
CDF NORMALMIX 分布 関数 . . . 394
CDF PARETO 分布 関数 . . . 397
CDF POISSON 分布 関数 . . . 399
CDF T 分布 関数 . . . 400
CDF TWEEDIE 分布 関数 . . . 402
CDF UNIFORM 分布 関数 . . . 405
CDF WALD (逆ガウス)分布 関数 . . . 407
CDF WEIBULL 分布 関数 . . . 409
CEIL 関数 . . . 410
CEILZ 関数 . . . 412
CHOOSEC 関数 . . . 414
CHOOSEN 関数 . . . 416
CMISS 関数 . . . 417
CMP 関数 . . . 419
CMPT 関数 . . . 420
CNONCT 関数 . . . 422

COALESCE 関数 424
COALESCEC 関数 426
COMB 関数 427
COMPARE 関数 429
COMPBL 関数 433
COMPCOST 関数 435
COMPFUZZ 関数 438
COMPGED 関数 440
COMPLEV 関数 447
COMPOUND 関数 450
COMPRESS 関数 452
CONSTANT 関数 458
CONVX 関数 464
CONVXP 関数 465
COS 関数 468
COSH 関数 469
COT 関数 470
COUNT 関数 472
COUNTC 関数 474
COUNTW 関数 478
CSC 関数 483
CSS 関数 484
CUMIPMT 関数 486
CUMPRINC 関数 488
CV 関数 490
DAIRY 関数 491
DATDIF 関数 492
DATE 関数 495
DATEJUL 関数 496
DATEPART 関数 498
DATETIME 関数 499
DAY 関数 500
DEQUOTE 関数 502
DEVIANCE 関数 507
DHMS 関数 511
DIF 関数 514
DIGAMMA 関数 517
DIM 関数 518
DIVIDE 関数 520
DUR 関数 522
DURP 関数 523
EFFRATE 関数 526
ERF 関数 527
ERFC 関数 529
EXP 関数 530
FACT 関数 532
FINANCE 関数 534
FINANCE ACCRINT 関数 538
FINANCE ACCRINTM 関数 540
FINANCE AMORDEGRC 関数 542
FINANCE AMORLINC 関数 544
FINANCE COUPDAYBS 関数 546
FINANCE COUPDAYS 関数 548
FINANCE COUPDAYSNC 関数 550

FINANCE COUPNCD 関数 552
FINANCE COUPNUM 関数 553
FINANCE COUPPCD 関数 555
FINANCE CUMIPMT 関数 557
FINANCE CUMPRINC 関数 559
FINANCE DB 関数 560
FINANCE DDB 関数 562
FINANCE DISC 関数 563
FINANCE DOLLARDE 関数 565
FINANCE DOLLARFR 関数 566
FINANCE DURATION 関数 567
FINANCE EFFECT 関数 569
FINANCE FV 関数 571
FINANCE FVSCHEDULE 関数 572
FINANCE INTRATE 関数 573
FINANCE IPMT 関数 575
FINANCE IRR 関数 577
FINANCE ISPMT 関数 578
FINANCE MDURATION 関数 579
FINANCE MIRR 関数 581
FINANCE NOMINAL 関数 583
FINANCE NPER 関数 584
FINANCE NPV 関数 586
FINANCE ODDFPRICE 関数 587
FINANCE ODDFYIELD 関数 589
FINANCE ODDLPRICE 関数 592
FINANCE ODDLYIELD 関数 594
FINANCE PMT 関数 596
FINANCE PPMT 関数 598
FINANCE PRICE 関数 600
FINANCE PRICEDISC 関数 602
FINANCE PRICEMAT 関数 604
FINANCE PV 関数 606
FINANCE RATE 関数 607
FINANCE RECEIVED 関数 609
FINANCE SLN 関数 611
FINANCE SYD 関数 612
FINANCE TBILLEQ 関数 613
FINANCE TBILLPRICE 関数 614
FINANCE TBILLYIELD 関数 616
FINANCE VDB 関数 617
FINANCE XIRR 関数 619
FINANCE XNPV 関数 620
FINANCE YIELD 関数 622
FINANCE YIELDDISC 関数 624
FINANCE YIELDMAT 関数 626
FIND 関数 628
FINDC 関数 631
FINDW 関数 640
FIPNAME 関数 648
FIPNAMEL 関数 650
FIPSTATE 関数 651
FLOOR 関数 653
FLOORZ 関数 655

FMTINFO 関数 . . . 657
FNONCT 関数 . . . 659
FUZZ 関数 . . . 662
GAMINV 関数 . . . 663
GAMMA 関数 . . . 665
GARKHCLPRC 関数 . . . 666
GARKHPTPRC 関数 . . . 669
GCD 関数 . . . 671
GEODIST 関数 . . . 673
GEOMEAN 関数 . . . 676
GEOMEANZ 関数 . . . 678
GETTHREADERRORSTATUS 関数 . . . 680
HARMEAN 関数 . . . 687
HARMEANZ 関数 . . . 689
HBOUND 関数 . . . 691
HMS 関数 . . . 693
HOLIDAY 関数 . . . 695
HOUR 関数 . . . 699
IBESSEL 関数 . . . 701
IFN_164 関数 . . . 702
INDEX 関数 . . . 705
INDEXC 関数 . . . 707
INDEXW 関数 . . . 709
INPUTC 関数 . . . 712
INPUTN 関数 . . . 714
INPUTN_164 関数 . . . 715
INT 関数 . . . 717
INTCINDEX 関数 . . . 719
INTCK 関数 . . . 724
INTCYCLE 関数 . . . 732
INTDT 関数 . . . 738
INTFIT 関数 . . . 739
INTGET 関数 . . . 741
INTINDEX 関数 . . . 745
INTNEST 関数 . . . 752
INTNX 関数 . . . 755
INTRR 関数 . . . 765
INTSEAS 関数 . . . 767
INTSHIFT 関数 . . . 772
INTTEST 関数 . . . 776
INTTS 関数 . . . 779
INTZ 関数 . . . 781
IPMT 関数 . . . 783
IQR 関数 . . . 785
IRR 関数 . . . 786
JBESSEL 関数 . . . 788
JULDATE 関数 . . . 789
JULDATE7 関数 . . . 791
KURTOSIS 関数 . . . 793
LAG 関数 . . . 794
LARGEST 関数 . . . 799
LBOUND 関数 . . . 801
LCM 関数 . . . 804
LCOMB 関数 . . . 805

LEFT 関数 . . . 806
LENGTH 関数 . . . 808
LENGTHC 関数 . . . 810
LENGTHM 関数 . . . 813
LFACT 関数 . . . 815
LGAMMA 関数 . . . 816
LOG 関数 . . . 818
LOG10 関数 . . . 819
LOG1PX 関数 . . . 820
LOG2 関数 . . . 822
LOGBETA 関数 . . . 823
LOGCDF 関数 . . . 825
LOGISTIC 関数 . . . 827
LOGPDF 関数 . . . 828
LOGSDF 関数 . . . 831
LOWCASE 関数 . . . 833
LPERM 関数 . . . 835
MAD 関数 . . . 836
MARGRCLPRC 関数 . . . 837
MARGRPTPRC 関数 . . . 840
MAX 関数 . . . 843
MD5 関数 . . . 845
MDY 関数 . . . 847
MEAN 関数 . . . 849
MEDIAN 関数 . . . 850
MIN 関数 . . . 852
MINUTE 関数 . . . 854
MISSING 関数 . . . 855
MOD 関数 . . . 859
MODZ 関数 . . . 861
MONTH 関数 . . . 863
MORT 関数 . . . 865
N 関数 . . . 867
NDIMS 関数 . . . 868
NETPV 関数 . . . 870
NMISS 関数 . . . 872
NOMRATE 関数 . . . 874
NOTALNUM 関数 . . . 876
NOTALPHA 関数 . . . 878
NOTCNTRL 関数 . . . 881
NOTDIGIT 関数 . . . 883
NOTFIRST 関数 . . . 886
NOTGRAPH 関数 . . . 888
NOTLOWER 関数 . . . 891
NOTNAME 関数 . . . 893
NOTPRINT 関数 . . . 895
NOTPUNCT 関数 . . . 897
NOTSPACE 関数 . . . 900
NOTUPPER 関数 . . . 903
NOTXDIGIT 関数 . . . 906
NPV 関数 . . . 908
NULL 関数 . . . 909
NWKDOM 関数 . . . 911
ORDINAL 関数 . . . 914

PCTL 関数 916
PDF 関数 918
PDF BERNOULLI 分布 関数 920
PDF BETA 分布 関数 922
PDF BINOMIAL 分布 関数 924
PDF CAUCHY 分布 関数 926
PDF カイ 2 乗分布 関数 928
PDF Conway-Maxwell-Poisson 分布 関数 930
PDF EXPONENTIAL 分布 関数 932
PDF F 分布 関数 934
PDF GAMMA 分布 関数 936
PDF 一般化 Poisson 分布 関数 938
PDF GEOMETRIC 分布 関数 940
PDF 超幾何分布 関数 942
PDF LAPLACE 分布 関数 944
PDF LOGISTIC 分布 関数 946
PDF LOGNORMAL 分布 関数 948
PDF NEGBINOMIAL 分布 関数 949
PDF NORMAL 分布 関数 951
PDF NORMALMIX 分布 関数 953
PDF PARETO 分布 関数 955
PDF POISSON 分布 関数 957
PDF T 分布 関数 959
PDF TWEEDIE 分布 関数 961
PDF UNIFORM 分布 関数 963
PDF Wald (逆ガウス)分布 関数 965
PDF WEIBULL 分布 関数 967
PERM 関数 968
PMT 関数 970
POISSON 関数 972
POWER 関数 974
PPMT 関数 975
PROBBETA 関数 977
PROBBNML 関数 979
PROBBNRM 関数 980
PROBCHI 関数 982
PROBDF 関数 983
PROBF 関数 990
PROBGAM 関数 991
PROBHYPR 関数 992
PROBIT 関数 994
PROBMC 関数 995
PROBMED 関数 1007
PROBNEGB 関数 1009
PROBNORM 関数 1010
PROBT 関数 1011
PROGRAMBLOCKLINENUMBER 関数 1013
PROGRAMBLOCKLNAME 関数 1015
PROGRAMBLOCKTYPE 関数 1017
PRXCHANGE 関数 1019
PRXMATCH 関数 1023
PRXPAREN 関数 1027
PRXPARSE 関数 1031
PRXPOSN 関数 1034

PUT 関数 1038
PUTC 関数 1042
PUTN 関数 1045
PVP 関数 1048
QTR 関数 1051
QUANTILE 関数 1052
QUOTE 関数 1056
RANBIN 関数 1058
RANCAU 関数 1058
RAND 関数 1058
RANEXP 関数 1079
RANGAM 関数 1079
RANGE 関数 1079
RANK 関数 1081
RANNOR 関数 1082
RANPOI 関数 1082
RANTBL 関数 1082
RANTRI 関数 1083
RANUNI 関数 1083
REPEAT 関数 1083
REVERSE 関数 1084
RIGHT 関数 1085
RMS 関数 1087
ROUND 関数 1088
ROUNDE 関数 1097
ROUNDZ 関数 1100
SASLOGLINENUMBER 関数 1103
SAVING 関数 1105
SAVINGS 関数 1106
SCAN 関数 1109
SDF 関数 1120
SEC 関数 1124
SECOND 関数 1126
SETTHREADERRORSTATUS 関数 1128
SHA256 関数 1131
SHA256HEX 関数 1132
SHA256HMACHEX 関数 1135
SIGN 関数 1137
SIN 関数 1138
SINH 関数 1139
SKEWNESS 関数 1140
SLEEP 関数 1142
SMALLEST 関数 1144
SQLEXEC 関数 1146
SQRT 関数 1148
SQUANTILE 関数 1149
STD 関数 1153
STDERR 関数 1154
STFIPS 関数 1155
STNAME 関数 1157
STNAMEL 関数 1158
STREAMINIT 関数 1160
STRIP 関数 1166
擬似 SUBSTR(割り当ての左辺に用いた場合) 関数 1168

SUBSTR (=の右) 関数 1172
SUBSTRN 関数 1175
SUM 関数 1181
SUMABS 関数 1182
SYSGET 関数 1184
TAN 関数 1185
TANH 関数 1186
TIME 関数 1187
TIMEPART 関数 1189
TIMEVALUE 関数 1190
TINV 関数 1193
TNONCT 関数 1195
TO_DATE 関数 1197
TO_DOUBLE 関数 1198
TO_TIME 関数 1202
TO_TIMESTAMP 関数 1203
TODAY 関数 1205
TRANSLATE 関数 1206
TRANSTRN 関数 1208
TRANWRD 関数 1211
TRIGAMMA 関数 1215
TRIM 関数 1217
TRUNC 関数 1219
TSLVL 関数 1220
UNIFORM 関数 1224
UPCASE 関数 1224
URLDECODE 関数 1226
URLENCODE 関数 1227
USS 関数 1229
UUIDGEN 関数 1230
VAR 関数 1232
VERIFY 関数 1233
VFORMAT 関数 1235
VGETVALUE 関数 1237
VINARRAY 関数 1239
VINFORMAT 関数 1241
VLABEL 関数 1243
VLENGTH 関数 1245
VNAME 関数 1247
VSETMISSING 関数 1248
VSETVALUE 関数 1252
VTYPE 関数 1253
WEEK 関数 1255
WEEKDAY 関数 1259
WHICHC 関数 1260
WHICHN 関数 1262
YEAR 関数 1263
YIELDP 関数 1265
YRDIF 関数 1267
YYQ 関数 1270
ZIPCITY 関数 1272
ZIPCITYDISTANCE 関数 1275
ZIPFIPS 関数 1276
ZIPNAME 関数 1278

ZIPNAMEL 関数 1280
ZIPSTATE 関数 1281

DS2 関数の概要

DS2 関数は、引数に対して計算またはシステム操作を実行し、値を返します。関数式は、DS2 プログラム、メソッド、またはスレッドのどこからでも関数を呼び出します。ほとんどの関数はユーザーが指定した引数を使用しますが、一部の関数は動作環境から引数を取得します。

関数式の引数のデータ型が DS2 関数で期待されるものではない場合、DS2 は引数に対して型変換を実行して、適切なデータ型になるようにします。型変換が成功すると、関数が実行されます。それ以外の場合は、エラーが発行されます。詳細については、“DS2 型変換” (*SAS DS2 プログラマガイド*)を参照してください。

注: DS2 は CALL ルーチンをサポートしていません。関数またはメソッドの戻り型が VOID の場合、DS2 関数およびメソッドは変数引数値を変更できます。

注: 日付と時間関数は、SAS の日付、時間、および日時の値でのみ機能します。これらは、DATE、TIME、および TIMESTAMP データ型の値では機能しません。日付の操作については、“DS2 式” (*SAS DS2 プログラマガイド*)を参照してください。

一般的な関数構文

関数の構文には、次のいずれかの形式があります。

function-name (*argument*[, ...*argument*])

function-name (OF *variable-list*)

function-name ([*argument* | OF *variable-list* | OF *array-name*[*]]
[..., [*argument* | OF *variable-list* | OF *array-name*[*]]])

function-name
関数に名前を付けます。

argument
変数名、定数、または別の関数を含む任意の DS2 式にすることができます。DS2 で許可される引数の数と種類は個別の関数と併せて記載しています。引数が複数のときはカンマで区切ります。

注: 引数の値が無効な場合(たとえば、欠損しているか、指定された範囲外である場合)、エラーが発生し、関数の戻り式が欠損値または null 値に設定されます。

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

例 次に例を示します。

```
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));
if sum(cash,credit)>1000 then put 'Goal reached';
```

variable-list

個々の変数名を含む、任意の形式の DS2 変数リストにすることができます。複数の変数リストが表示される場合は、それらをスペースまたはカンマと 別の OF で区切ります。

```
a=sum(of x y z);


z=sum(of y1-y10);

z=msplint(x0,5,of x1-x5,of y1-y5,-2,2);
```

参照項目: “OF 演算子” (2109 ページ)

“関数の引数として省略変数リストを指定する” (*SAS DS2 プログラマガイド*)

例 次の 2 つの例は同等です。

```
a=sum(of x1-x10 y1-y10 z1-z10);

a=sum(of x1-x10, of y1-y10, of z1-z10);
```

array-name[*]

現在定義されている配列に名前を付けます。添え字としてアスタリスクを使用して配列を指定すると、DS2 は配列の各要素を個別の引数として扱います。

参照項目: “OF 演算子” (2109 ページ)

“変数配列を関数の引数として指定する” (*SAS DS2 プログラマガイド*)

関数の使用

関数の引数の制限

引数の値が無効な場合は、エラーが発生し、戻り式が欠損値または null 値に設定されます。関数の引数に関する一般的な制限は次のとおりです。

- 一部の関数では、引数を特定の範囲内に制限する必要があります。たとえば、LOG 関数の引数は 0 より大きくする必要があります。
- ほとんどの関数では、引数として欠損値や null 値を許可していません。例外には、記述統計関数と財務関数の一部が含まれます。

  デフォルトでは、関数の引数に欠損値または null 値が含まれている場合、エラーが発生し、メッセージが SAS ログに出力されます。DS2_OPTIONS ステートメントの MISSING_NOTE オプションを使用して、エラーを生成せず、関数の引数に欠損値または null 値が含まれている場合に SAS ログに注を書き込むことができます。詳細については、“DS2_OPTIONS ステートメント” (1348 ページ)を参照してください。
- 一般に、引数の許容範囲は、EXP 関数など、プラットフォームによって異なります。

関数でのデータ型変換

DS2 が関数に期待する引数の数と引数のデータ型は、**関数シグネチャ**と呼ばれます。関数が実行されると、関数式の引数が関数シグネチャと比較されます。引数とシグネチャが一致すると、関数が実行されます。引数の数が一致しない場合、エラーが発生します。

引数の数は一致するが、1 つ以上の引数のデータ型が一致しない場合、DS2 は引数のデータ型を関数シグネチャのデータ型に変換しようとします。型変換が成功すると、関数が実行されます。それ以外の場合は、エラーが発生します。

各関数のドキュメントには、すべての関数引数の有効なデータ型が含まれています。有効なデータ型は、DS2 が関数の実行に使用する引数データ型です。DS2 は型変換を行うため、関数式の一部の引数データ型は、関数引数の有効なデータ型と同じである必要はありません。たとえば、関数式に INTEGER のデータ型の引数が含まれていて、その引数の有効なデータ型が DOUBLE の場合、DS2 は関数の実行時にデータ型を DOUBLE に変換します。4 つのデータ型 VARBINARY と日付/時間データ型 DATE、TIME、TIMESTAMP のみが非強制的です。つまり、関数式には有効なデータ型が含まれている必要があります。

DS2 数値関数の欠損値と null 値

入力データ型が DOUBLE の関数の場合、関数に null が渡されると、null 値は欠損値に変換されます。

関数が処理された後、関数が DOUBLE を返し、関数が欠損値を返す場合、SAS モードでは欠損値が返され、ANSI モードでは null 値が返されます。

詳細については、“DS2 による null および SAS 欠損値の処理方法” (*SAS DS2 プログラマガイド*)を参照してください。

DS2 文字関数の欠損値と null 値

文字関数に null 値を渡し、関数が null 値を返す必要がある場合、ANSI モードか SAS モードかに関係なく、null 値が返されます。

システム式を使用して関数を実行する

関数の構文はメソッドの構文に似ており、DS2 がメソッドまたは関数の式を検出すると、それがどのタイプの式であるかを判別する必要があります。メソッド式と関数式の名前が同じ場合、DS2 はメソッドを実行します。メソッドと同じ名前の関数を実行する唯一の方法は、システム式を使用することです。システム式の構文は次のとおりです。

system.*function-expression*

DS2 がシステム式を検出すると、同じ名前のメソッドの呼び出しはバイパスされ、DS2 は関数を呼び出します。

記述統計関数に関する注記

DS2 は、記述統計量を返す関数を提供します。MISSING 関数を除いて、関数は MEANS プロシジャによって生成された統計量に対応します。各統計量の計算方法については、*Base SAS プロシジャガイド*の統計プロシジャセクションで説明しています。SAS は、引数の非 null または非欠損値の記述統計量を計算します。

財務関数に関する注記

財務関数の種類

SAS は、財務計算を実行する関数のグループを提供します。関数は次の種類にグループ化されます。

表 7.1 財務関数の種類

関数の種類	関数	説明
キャッシュフロー	CONVX、CONVXP	キャッシュフローのコンベクシティを計算します
	DUR、DURP	キャッシュフローの変更期間を計算します
	PVP、YIELDP	定期キャッシュフローの現在価値と満期利回りを計算します
一般	FINANCE	減価償却、満期、未払い利息、正味現在価値、定期預金、内部収益率を計算します。
内部収益率	INTRR、IRR	内部収益率を計算します
正味現在価値と将来価値	NETPV、NPV	正味現在価値と将来価値を計算します
	SAVINGS	変動金利を使用して定期預金の残高を返します
パラメーター計算	COMPOUND	複利パラメーターを計算します
	MORT	割賦返済パラメーターを計算します
価格決定	BLKSHCLPRC、BLKSHPTPRC	Black-Scholes モデルに基づき、株式のヨーロピアンオプションのコール価格とプット価格を計算します

関数の種類	関数	説明
	BLACKCLPRC、BLACKPTPRC	Black-Scholes モデルに基づき、先物のヨーロピアンオプションのコール価格とプット価格を計算します
	GARKHCLPRC、GARKHPTPRC	Garman-Kohlhagen モデルに基づき、株式のヨーロピアンオプションのコール価格とプット価格を計算します
	MARGRCLPRC、MARGRPTPRC	Margrabe モデルに基づき、株式のヨーロピアンオプションのコールオプションとプット価格を計算します

価格決定関数の使用

価格決定モデルは、金融商品の理論上の市場価値(価格)を計算するために使用されます。この値は、時価評価(MtM)値と呼ばれます。通常、価格決定関数の形式は次のとおりです。

$$price = function(rf1, rf2, rf3, ...)$$

価格決定関数では、*rf1*、*rf2*、および *rf3* は、金利や外国為替レートなどのリスクファクターです。MtM 値の計算に使用されるリスクファクターの特定の値は、ベースケース値です。ベースケース値のセットは、ベースケース市場状態として知られています。

MtM 値を決定した後、リスクファクター(*rf1*、*rf2*、および *rf3*)のベースケース値を使用して次のタスクを実行できます。

- シナリオ分析を実行するには、ベースケース値を特定の値に設定します。
- ベースケース値を値の範囲に設定して、損益曲線分析および損益曲面分析を実行します。
- ベースケース値を異なる値に自動的に設定して、感度を計算します。つまり、リスクファクターのデルタ値とガンマ値を計算します。
- ベースケースの値を摂動させて、多くの可能な市場状態を作成し、多くの可能な先物価格を計算し、シミュレーション分析を実行できるようにします。モンテカルロシミュレーションの場合、リスクファクターの値は、数学モデルとコピュラ法を使用して生成されます。

価格決定関数とその説明のリストは、“財務関数の種類” (248 ページ)に含まれています。

DS2 関数の例

```
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));
if sum(cash,credit)>1000 then
  put 'Goal reached';
```

関数カテゴリ

関数は、操作する値の種類によって分類できます。各 DS2 関数は、次のいずれかのカテゴリに属します。

表 7.2　関数カテゴリの説明

カテゴリ	説明
算術	ODS 出力の特殊欠損値を処理する除算の結果を返します。 関数のリストについては、算術 (262 ページ)を参照してください。
配列	同種データの名前付き集計コレクションを操作します。 関数のリストについては、配列を参照してください。
ビットごとの論理演算	個々のビットのレベルで 1 つ以上のビットパターンまたは 2 進数を操作します。 関数のリストについては、ビットごとの論理演算を参照してください。
文字	文字データと SQL 式を操作します。 関数のリストについては、文字を参照してください。
日付と時間	日付と時間の値を操作します。 関数のリストについては、日付と時間を参照してください。

カテゴリ	説明
記述統計	中心傾向、値間の変動、および分布値の形状を測定する値を操作します。 関数のリストについては、記述統計を参照してください。
距離	測地距離を返します。 関数のリストについては、距離を参照してください。
エラーチェック	スレッド上で行レベルのランタイムエラーチェックを実行します。 関数のリストについては、エラーチェックを参照してください。
財務	株式のヨーロッパオプションの利息、定期支払い、減価償却、価格などの財務価値を計算します。 関数のリストについては、財務を参照してください。
双曲線	正弦、余弦、正接などの双曲線計算を実行します。 関数のリストについては、双曲線を参照してください。
数学	値を操作して一般的な数学計算を実行します。 関数のリストについては、数学を参照してください。
数値	数値を操作します。 関数のリストについては、数値を参照してください。
確率	確率計算を返します。 関数のリストについては、確率を参照してください。
分位点	特定の分布から分位点を返します。 関数のリストについては、分位点を参照してください。
乱数	特定の分布からランダム変量を返します。 関数のリストについては、乱数を参照してください。
特殊	null 値と SAS 欠損値の操作、プログラムの実行中断、実行時の数値入力形式の指定、FedSQL ステートメントの実行。 関数のリストについては、特殊を参照してください。
州と ZIP コード	ZIP コード、FIPS コード、州名と都市名、郵便番号、ZIP コード間の測地距離を返します。 関数のリストについては、州と ZIP コードを参照してください。
三角関数	値を操作して三角関数の計算を実行します。 関数のリストについては、三角関数を参照してください。

カテゴリ	説明
切り捨て	値を操作して桁数を制限します。 関数のリストについては、切り捨てを参照してください。
変数の割り当て	データ値をグローバルスカラー変数、変数配列の要素、または varlist の要素に割り当て、データ値を取得して別の変数に割り当てることができるようにします。関数のリストについては、変数の割り当てを参照してください。
変数情報	変数を操作し、名前、型、長さ、入力形式、ラベルなどの変数情報を返します。 関数のリストについては、変数情報を参照してください。
Web ツール	データの文字列をエンコードおよびデコードします。 関数のリストについては、Web ツールを参照してください。

次の表に、DS2 関数の簡単な説明を示します。詳細については、各関数を参照してください。

カテゴリ	言語要素	説明
CAS	IFN_164 関数 (p. 702)	式の true、false、欠損に基づいて数値を返します。
Web ツール	URLDECODE 関数 (p. 1226)	URL のエスケープ構文を使用してデコードされた文字列を返します。
	URLENCODE 関数 (p. 1227)	URL のエスケープ構文を使用してエンコードされた文字列を返します。
インストール情報	TSLVL 関数 (p. 1220)	リリース番号、ホットフィックス、メンテナンスイメージなど、SAS インストールイメージに関する情報を提供します。
エラーチェック	GETTHREADERRORSTATUS 関数 (p. 680)	現在の DS2 スレッドのランタイムエラーステータスを取得します。
	SETTHREADERRORSTATUS 関数 (p. 1128)	致命的ではないランタイムエラーに対してユーザー指定のエラーステータス値を設定し、オプションでユーザー指定のメッセージを SAS ログに記録します。
確率	CDF 関数 (p. 359)	さまざまな連続確率分布および離散確率分布の左累積分布関数を計算します。
	CDF BERNOULLI 分布 関数 (p. 362)	Bernoulli 累積確率分布の値を返します。
	CDF BETA 分布 関数 (p. 363)	ベータ累積確率分布の値を返します。
	CDF BINOMIAL 分布 関数 (p. 366)	二項累積確率分布の値を返します。

カテゴリ	言語要素	説明
	CDF CAUCHY 分布 関数 (p. 368)	Cauchy 累積確率分布の値を返します。
	CDF カイ 2 乗分布 関数 (p. 370)	カイ 2 乗累積確率分布の値を返します。
	CDF Conway-Maxwell-Poisson 分布 関数 (p. 372)	Conway-Maxwell-Poisson 累積確率分布の値を返します。
	CDF 指数分布 関数 (p. 373)	指数累積確率分布の値を返します。
	CDF F 分布 関数 (p. 375)	F 累積確率分布の値を返します。
	CDF GAMMA 分布 関数 (p. 377)	ガンマ累積確率分布の値を返します。
	CDF 一般化 Poisson 分布 関数 (p. 379)	一般化 Poisson 累積確率分布の値を返します。
	CDF GEOMETRIC 分布 関数 (p. 381)	幾何累積確率分布の値を返します。
	CDF HYPERGEOMETRIC 分布 関数 (p. 383)	超幾何累積確率分布の値を返します。
	CDF LAPLACE 分布 関数 (p. 385)	Laplace 累積確率分布の値を返します。
	CDF LOGISTIC 分布 関数 (p. 387)	ロジスティック累積確率分布の値を返します。
	CDF LOGNORMAL 分布 関数 (p. 389)	対数正規累積確率分布の値を返します。
	CDF NEGBINOMIAL 分布 関数 (p. 391)	負の二項累積確率分布の値を返します。
	CDF NORMAL 分布 関数 (p. 393)	正規累積確率分布の値を返します。
	CDF NORMALMIX 分布 関数 (p. 394)	正規混合累積確率分布の値を返します。
	CDF PARETO 分布 関数 (p. 397)	パレート累積確率分布の値を返します。
	CDF POISSON 分布 関数 (p. 399)	Poisson 累積確率分布の値を返します。
	CDF T 分布 関数 (p. 400)	T 累積確率分布の値を返します。
	CDF TWEEDIE 分布 関数 (p. 402)	Tweedie 累積確率分布の値を返します。

カテゴリ	言語要素	説明
	CDF UNIFORM 分布 関数 (p. 405)	一様累積確率分布の値を返します。
	CDF WALD (逆ガウス)分布 関数 (p. 407)	Wald (逆ガウスとも呼ばれる)累積確率分布から値を返します。
	CDF WEIBULL 分布 関数 (p. 409)	Weibull 累積確率分布の値を返します。
	LOGCDF 関数 (p. 825)	左累積分布関数の対数を返します。
	LOGPDF 関数 (p. 828)	さまざまな連続分布および離散分布から確率密度(質量)関数の対数を計算します。
	LOGSDF 関数 (p. 831)	生存関数の対数を返します。
	PDF 関数 (p. 918)	確率密度(質量)分布の値を返します。
	PDF BERNOULLI 分布 関数 (p. 920)	Bernoulli 確率密度(質量)分布の値を返します。
	PDF BETA 分布 関数 (p. 922)	ベータ確率密度(質量)分布の値を返します。
	PDF BINOMIAL 分布 関数 (p. 924)	二項確率密度(質量)分布の値を返します。
	PDF CAUCHY 分布 関数 (p. 926)	Cauchy 確率密度(質量)分布の値を返します。
	PDF カイ 2 乗分布 関数 (p. 928)	カイ 2 乗確率密度(質量)分布の値を返します。
	PDF Conway-Maxwell-Poisson 分布 関数 (p. 930)	Conway-Maxwell-Poisson 確率密度(質量)分布の値を返します。
	PDF EXPONENTIAL 分布 関数 (p. 932)	指数確率密度(質量)分布の値を返します。
	PDF F 分布 関数 (p. 934)	F 確率密度(質量)分布の値を返します。
	PDF GAMMA 分布 関数 (p. 936)	ガンマ確率密度(質量)分布の値を返します。
	PDF 一般化 Poisson 分布 関数 (p. 938)	一般化 Poisson 確率密度(質量)分布の値を返します。
	PDF GEOMETRIC 分布 関数 (p. 940)	幾何確率密度(質量)分布の値を返します。
	PDF 超幾何分布 関数 (p. 942)	超幾何確率密度(質量)分布の値を返します。
	PDF LAPLACE 分布 関数 (p. 944)	Laplace 確率密度(質量)分布の値を返します。

カテゴリ	言語要素	説明
	PDF LOGISTIC 分布 関数 (p. 946)	ロジスティック確率密度(質量)分布の値を返します。
	PDF LOGNORMAL 分布 関数 (p. 948)	対数正規確率密度(質量)分布の値を返します。
	PDF NEGBINOMIAL 分布 関数 (p. 949)	負の二項確率密度(質量)分布の値を返します。
	PDF NORMAL 分布 関数 (p. 951)	正規確率密度(質量)分布の値を返します。
	PDF NORMALMIX 分布 関数 (p. 953)	正規混合確率密度(質量)分布の値を返します。
	PDF PARETO 分布 関数 (p. 955)	パレート確率密度(質量)分布の値を返します。
	PDF POISSON 分布 関数 (p. 957)	Poisson 確率密度(質量)分布の値を返します。
	PDF T 分布 関数 (p. 959)	T 確率密度(質量)分布の値を返します。
	PDF TWEEDIE 分布 関数 (p. 961)	Tweedie 確率密度(質量)分布の値を返します。
	PDF UNIFORM 分布 関数 (p. 963)	一様確率密度(質量)分布の値を返します。
	PDF Wald (逆ガウス)分布 関数 (p. 965)	Wald (逆ガウスとも呼ばれる)確率密度(質量)分布から値を返します。
	PDF WEIBULL 分布 関数 (p. 967)	Weibull 確率密度(質量)分布の値を返します。
	POISSON 関数 (p. 972)	Poisson 分布の確率を返します。
	PROBBETA 関数 (p. 977)	ベータ分布の確率を返します。
	PROBBNML 関数 (p. 979)	二項分布の確率を返します。
	PROBBNRM 関数 (p. 980)	2 変量正規分布の確率を返します。
	PROBCHI 関数 (p. 982)	カイ 2 乗分布の確率を返します。
	PROBDF 関数 (p. 983)	時系列の単位根に対する Dickey-Fuller 検定の有意確率を計算します。
	PROBF 関数 (p. 990)	F 分布の確率を返します。
	PROBGAM 関数 (p. 991)	ガンマ分布の確率を返します。
	PROBHYPR 関数 (p. 992)	超幾何分布の確率を返します。

カテゴリ	言語要素	説明
	PROBMC 関数 (p. 995)	平均値の多重比較を行うために各種の分布から確率または分位点を返します。
	PROBMED 関数 (p. 1007)	サンプル中央値の累積確率を計算します。
	PROBNEGB 関数 (p. 1009)	負の二項分布の確率を返します。
	PROBNORM 関数 (p. 1010)	標準正規分布の確率を返します。
	PROBT 関数 (p. 1011)	t 分布の確率を返します。
	SDF 関数 (p. 1120)	生存関数を返します。
記述統計	CMISS 関数 (p. 417)	欠損引数の数を数えます。
	CSS 関数 (p. 484)	修正済み平方和を返します。
	CV 関数 (p. 490)	変動係数を返します。
	GEOMEAN 関数 (p. 676)	幾何平均を返します。
	GEOMEANZ 関数 (p. 678)	ゼロファジーを使用して、幾何平均を返します。
	HARMEAN 関数 (p. 687)	調和平均を返します。
	HARMEANZ 関数 (p. 689)	ゼロファジーを使用して、調和平均を返します。
	IQR 関数 (p. 785)	四分位範囲を返します。
	KURTOSIS 関数 (p. 793)	尖度を返します。
	LARGEST 関数 (p. 799)	k 番目に大きい非 null または非欠損値を返します。
	MAD 関数 (p. 836)	中央値からの中央絶対偏差を返します。
	MAX 関数 (p. 843)	引数のリストから最大値を返します。
	MEAN 関数 (p. 849)	非 null または非欠損引数の算術平均(平均)を返します。
	MEDIAN 関数 (p. 850)	中央値を返します。
	MIN 関数 (p. 852)	最小値を返します。
	NMISS 関数 (p. 872)	null および SAS 欠損数値の数を返します。
	ORDINAL 関数 (p. 914)	値のリストを並べ替え、リスト内の位置に基づく値を返します。
	PCTL 関数 (p. 916)	パーセントに対応するパーセント点を返します。
	RANGE 関数 (p. 1079)	最大値と最小値の差を返します。
	RMS 関数 (p. 1087)	2 乗平均平方根を返します。

カテゴリ	言語要素	説明
	SKEWNESS 関数 (p. 1140)	歪度を返します。
	SMALLEST 関数 (p. 1144)	k 番目に小さい非 null 値または非欠損値を返します。
	STD 関数 (p. 1153)	標準偏差を返します。
	STDERR 関数 (p. 1154)	平均の標準誤差を返します。
	SUM 関数 (p. 1181)	非 null または非欠損引数の合計を返します。
	SUMABS 関数 (p. 1182)	非欠損引数の絶対値の合計を返します。
	USS 関数 (p. 1229)	無修正平方和を返します。
	VAR 関数 (p. 1232)	分散を返します。
距離	GEODIST 関数 (p. 673)	2 つの緯度と経度の座標間の測地距離を返します。
	ZIPCITYDISTANCE 関数 (p. 1275)	2 つの ZIP コード位置間の測地距離を返します。
切り捨て	CEIL 関数 (p. 410)	数値式以上の最小の整数を返します。
	CEILZ 関数 (p. 412)	ゼロファジーを使用して、引数より大きいか等しい整数のうち最小の値を返します。
	FLOOR 関数 (p. 653)	数値式以下の最大の整数を返します。
	FLOORZ 関数 (p. 655)	ゼロファジーを使用して、引数より小さいか等しい整数のうち最大の値を返します。
	FUZZ 関数 (p. 662)	引数が整数の 1E-12 以内の場合、最も近い整数を返します。
	INT 関数 (p. 717)	予期しない浮動小数点の結果を避けるためにファジー処理された整数を返します。
	INTZ 関数 (p. 781)	ゼロファジーを使用して、引数の整数部を返します。
	ROUND 関数 (p. 1088)	第 1 引数を第 2 引数の最も近い倍数(第 2 引数が省略された場合は最も近い整数)に丸めます。
	ROUNDE 関数 (p. 1097)	第 1 引数を第 2 引数の最も近い倍数に丸め、第 1 引数が 2 つの最も近い倍数の中間にある場合には偶数の倍数を返します。
	ROUNDZ 関数 (p. 1100)	第 1 引数を第 2 引数の最も近い倍数にゼロファジーを使用して丸めます。
	TRUNC 関数 (p. 1219)	数値を指定した長さに切り捨てます。
組み合わせ	COMB 関数 (p. 427)	n 個の要素を同時に r 個使用するときの組み合わせの数を計算します。

カテゴリ	言語要素	説明
	LCOMB 関数 (p. 805)	COMB 関数の対数を計算します。これは、n 個のオブジェクトから一度に r 個を取り出した組み合わせ数の対数です。
	LFACT 関数 (p. 815)	FACT(階乗)関数の対数を計算します。
	LPERM 関数 (p. 835)	n 個のオブジェクトの順列数の対数を計算し、r 個の要素を含めることができます。
	PERM 関数 (p. 968)	n 個の項目から一度に r 個の項目を抜き出す場合の順列数を計算します。
財務	BLACKCLPRC 関数 (p. 327)	Black モデルに基づき、先物のヨーロピアンオプションのコール価格を計算します。
	BLACKPTPRC 関数 (p. 329)	Black モデルに基づき、先物のヨーロピアンオプションのプット価格を計算します。
	BLKSHCLPRC 関数 (p. 332)	Black-Scholes モデルに基づき、株式のヨーロピアンオプションのコール価格を計算します。
	BLKSHPTPRC 関数 (p. 334)	Black-Scholes モデルに基づき、株式のヨーロピアンオプションのプット価格を計算します。
	COMPOUND 関数 (p. 450)	複利パラメーターを返します。
	CONVX 関数 (p. 464)	列挙キャッシュフローのコンベクシティを返します。
	CONVXP 関数 (p. 465)	債権などの定期キャッシュフローストリームのコンベクシティを返します。
	CUMIPMT 関数 (p. 486)	開始期間と終了期間の間でローンに対して支払われる累積利息を返します。
	CUMPRINC 関数 (p. 488)	開始期間と終了期間の間でローンに対して支払われる累積元本を返します。
	DUR 関数 (p. 522)	列挙キャッシュフローの修正デュレーションを返します。
	DURP 関数 (p. 523)	債権などの定期キャッシュフローストリームの修正デュレーションを返します。
	EFFRATE 関数 (p. 526)	有効な年利を返します。
	FINANCE 関数 (p. 534)	減価償却、満期、未払い利息、正味現在価値、定期的預金、内部利益率などの財務計算を行います。
	FINANCE ACCRINT 関数 (p. 538)	定期的に利息を支払う証券の未払い利息を計算します。
	FINANCE ACCRINTM 関数 (p. 540)	満期に利息を支払う証券の未払い利息を計算します。
	FINANCE AMORDEGRC 関数 (p. 542)	減価償却係数を使用して各会計期間の減価償却を計算します。

カテゴリ	言語要素	説明
	FINANCE AMORLINC 関数 (p. 544)	各会計期間の減価償却を計算します。
	FINANCE COUPDAYBS 関数 (p. 546)	クーポン期間の開始日から決済日までの日数を計算します。
	FINANCE COUPDAYS 関数 (p. 548)	決済日を含むクーポン期間の日数を計算します。
	FINANCE COUPDAYSNC 関数 (p. 550)	決済日から次のクーポン日までの日数を計算します。
	FINANCE COUPNCD 関数 (p. 552)	決済日後の次のクーポン日を計算します。
	FINANCE COUPNUM 関数 (p. 553)	決済日から満期日までの支払いクーポン数を計算します。
	FINANCE COUPPCD 関数 (p. 555)	決済日前の前のクーポン日を計算します。
	FINANCE CUMIPMT 関数 (p. 557)	2 つの期間の間に支払われる累積利息を計算します。
	FINANCE CUMPRINC 関数 (p. 559)	2 つの期間の間にローンで支払われる累積元本を計算します。
	FINANCE DB 関数 (p. 560)	定率法を使用して、指定期間の資産の減価償却を計算します。
	FINANCE DDB 関数 (p. 562)	倍額定率法またはその他の方法を指定して、指定期間の資産の減価償却を計算します。
	FINANCE DISC 関数 (p. 563)	証券の割引率を計算します。
	FINANCE DOLLARDE 関数 (p. 565)	分数で表すドル価格を小数で表すドル価格に変換します。
	FINANCE DOLLARFR 関数 (p. 566)	小数で表すドル価格を分数で表すドル価格に変換します。
	FINANCE DURATION 関数 (p. 567)	定期的に利息を支払う証券の年度期間を計算します。
	FINANCE EFFECT 関数 (p. 569)	有効な年利を計算します。
	FINANCE FV 関数 (p. 571)	投資の将来価値を計算します。
	FINANCE FVSCHEDULE 関数 (p. 572)	一連の複利率を適用した後の当初元本の将来価値を計算します。
	FINANCE INTRATE 関数 (p. 573)	完全投資済み証券の利率を計算します。

カテゴリ	言語要素	説明
	FINANCE IPMT 関数 (p. 575)	一定期間の投資の利払いを計算します。
	FINANCE IRR 関数 (p. 577)	一連のキャッシュフローの内部利益率を計算します。
	FINANCE ISPMT 関数 (p. 578)	投資の指定期間中に支払われる利息を計算します。
	FINANCE MDURATION 関数 (p. 579)	推定額面$100 の証券の修正マコーレーデュレーションを計算します。
	FINANCE MIRR 関数 (p. 581)	プラスおよびマイナスのキャッシュフローが異なる利率で資金調達される場合の内部利益率を計算します。
	FINANCE NOMINAL 関数 (p. 583)	年間の名目金利を計算します。
	FINANCE NPER 関数 (p. 584)	投資の期間数を計算します。
	FINANCE NPV 関数 (p. 586)	一連の定期的なキャッシュフローと割引率に基づいて投資の正味現在価値を計算します。
	FINANCE ODDFPRICE 関数 (p. 587)	端数の開始期間で額面$100 当たりの証券の価格を計算します。
	FINANCE ODDFYIELD 関数 (p. 589)	端数の開始期間で証券の利回りを計算します。
	FINANCE ODDLPRICE 関数 (p. 592)	端数の最終期間で額面$100 当たりの証券の価格を計算します。
	FINANCE ODDLYIELD 関数 (p. 594)	端数の最終期間で証券の利回りを計算します。
	FINANCE PMT 関数 (p. 596)	年金の定期的支払いを計算します。
	FINANCE PPMT 関数 (p. 598)	指定期間の投資の元本に対する支払いを計算します。
	FINANCE PRICE 関数 (p. 600)	定期的に利息を支払う額面$100 当たりの証券の価格を計算します。
	FINANCE PRICEDISC 関数 (p. 602)	額面$100 当たりの割引証券の価格を計算します。
	FINANCE PRICEMAT 関数 (p. 604)	満期に利息を支払う額面$100 当たりの証券の価格を計算します。
	FINANCE PV 関数 (p. 606)	投資の現在価値を計算します。
	FINANCE RATE 関数 (p. 607)	年金期間当たりの利率を計算します。
	FINANCE RECEIVED 関数 (p. 609)	完全投資済み証券の満期受け取り額を計算します。

カテゴリ	言語要素	説明
	FINANCE SLN 関数 (p. 611)	ある期間の資産の定額法による減価償却を計算します。
	FINANCE SYD 関数 (p. 612)	指定期間の資産の年次級数和法による減価償却を計算します。
	FINANCE TBILLEQ 関数 (p. 613)	米国財務省短期証券の債券換算利回りを計算します。
	FINANCE TBILLPRICE 関数 (p. 614)	額面$100 当たりの米国財務省短期証券の価格を計算します。
	FINANCE TBILLYIELD 関数 (p. 616)	米国財務省短期証券の利回りを計算します。
	FINANCE VDB 関数 (p. 617)	定率法を使用して、指定期間または一部期間の資産の減価償却を計算します。
	FINANCE XIRR 関数 (p. 619)	必ずしも定期的ではないキャッシュフロースケジュールの内部利益率を計算します。
	FINANCE XNPV 関数 (p. 620)	必ずしも定期的ではないキャッシュフロースケジュールの正味現在価値を計算します。
	FINANCE YIELD 関数 (p. 622)	定期的に利息を支払う証券の利回りを計算します。
	FINANCE YIELDDISC 関数 (p. 624)	割引証券(米国財務省短期証券など)の年間利回りを計算します。
	FINANCE YIELDMAT 関数 (p. 626)	満期に利息を支払う証券の年間利回りを計算します。
	GARKHCLPRC 関数 (p. 666)	Garman-Kohlhagen モデルに基づいて、株式のヨーロピアンオプションのコール価格を計算します。
	GARKHPTPRC 関数 (p. 669)	Garman-Kohlhagen モデルに基づいて、株式のヨーロピアンオプションのプット価格を計算します。
	INTRR 関数 (p. 765)	内部利益率を小数値で返します。
	IPMT 関数 (p. 783)	将来の残高を達成するための、均等払いローンまたは定期預金に対する指定期間の利息の支払いを返します。
	IRR 関数 (p. 786)	内部利益率をパーセントで返します。
	MARGRCLPRC 関数 (p. 837)	Margrabe モデルに基づいて、株式のヨーロピアンオプションのコール価格を計算します。
	MARGRPTPRC 関数 (p. 840)	Margrabe モデルに基づいて、株式のヨーロピアンオプションのプット価格を計算します。
	MORT 関数 (p. 865)	割賦返済パラメーターを返します。
	NETPV 関数 (p. 870)	現在正味価値をパーセントで返します。

カテゴリ	言語要素	説明
	NOMRATE 関数 (p. 874)	名目年利を返します。
	NPV 関数 (p. 908)	パーセントで表す利率を使用して現在正味価値を返します。
	PMT 関数 (p. 970)	均等払いローンまたは定期預金の将来残高に対する定期的支払いを返します。
	PPMT 関数 (p. 975)	将来の残高を達成するための、均等払いローンまたは定期預金に対する指定期間の元本の支払いを返します。
	PVP 関数 (p. 1048)	満期時の元本払い戻しで、定期的なキャッシュフローストリーム(債権など)の現在価値を返します。
	SAVING 関数 (p. 1105)	定期預金の将来価値を返します。
	SAVINGS 関数 (p. 1106)	変動金利を使用して定期預金の残高を返します。
	TIMEVALUE 関数 (p. 1190)	変動金利を使用して、基準日の参照額に相当する額を返します。
	YIELDP 関数 (p. 1265)	定期的なキャッシュフローストリーム(債権など)の最終利回りを返します。
三角関数	ARCOS 関数 (p. 309)	逆余弦をラジアンで返します。
	ARSIN 関数 (p. 312)	逆正弦をラジアンで返します。
	ATAN 関数 (p. 316)	逆正接をラジアンで返します。
	ATAN2 関数 (p. 317)	直角三角形の x 座標と y 座標の逆正接をラジアンで返します。
	COS 関数 (p. 468)	余弦をラジアンで返します。
	COSH 関数 (p. 469)	双曲線余弦をラジアンで返します。
	COT 関数 (p. 470)	余接を返します。
	CSC 関数 (p. 483)	余割を返します。
	SEC 関数 (p. 1124)	正割を返します。
	SIN 関数 (p. 1138)	三角関数の正弦を返します。
	TAN 関数 (p. 1185)	正接を返します。
算術	DIVIDE 関数 (p. 520)	ODS 出力の特殊欠損値を処理する除算の結果を返します。
州と ZIP コード	FIPNAME 関数 (p. 648)	2 桁の FIPS コードを大文字の州名に変換します。
	FIPNAMEL 関数 (p. 650)	2 桁の FIPS コードを大文字と小文字が混在する州名に変換します。

カテゴリ	言語要素	説明
	FIPSTATE 関数 (p. 651)	2 桁の FIPS コードを 2 文字の州郵便番号に変換します。
	STFIPS 関数 (p. 1155)	州郵便番号を FIPS 州コードに変換します。
	STNAME 関数 (p. 1157)	州郵便番号を大文字の州名に変換します。
	STNAMEL 関数 (p. 1158)	州郵便番号を大文字と小文字が混在する州名に変換します。
	ZIPCITY 関数 (p. 1272)	ZIP コードに対応する都市名と 2 文字の郵便番号を返します。
	ZIPCITYDISTANCE 関数 (p. 1275)	2 つの ZIP コード位置間の測地距離を返します。
	ZIPFIPS 関数 (p. 1276)	ZIP コードを 2 桁の FIPS コードに変換します。
	ZIPNAME 関数 (p. 1278)	ZIP コードを大文字の州名に変換します。
	ZIPNAMEL 関数 (p. 1280)	ZIP コードを大文字と小文字が混在する州名に変換します。
	ZIPSTATE 関数 (p. 1281)	ZIP コードを 2 文字の州郵便番号に変換します。
数学	ABS 関数 (p. 273)	数値式の絶対値を返します。
	AIRY 関数 (p. 275)	Airy 関数の値を返します。
	BETA 関数 (p. 320)	beta 関数の値を返します。
	CNONCT 関数 (p. 422)	カイ 2 乗分布の非心度パラメーターを返します。
	COALESCE 関数 (p. 424)	数値の引数のリストから最初の非 null または非欠損値を返します。
	COMPFUZZ 関数 (p. 438)	2 つの数値をファジー比較します。
	CONSTANT 関数 (p. 458)	マシン定数および数学定数を計算します。
	DAIRY 関数 (p. 491)	AIRY 関数の導関数を返します。
	DEVIANCE 関数 (p. 507)	確率分布に基づくデビアンスを返します。
	DIGAMMA 関数 (p. 517)	ディガンマ関数の値を返します。
	ERF 関数 (p. 527)	(正規)誤差関数の値を返します。
	ERFC 関数 (p. 529)	相補(正規)誤差関数の値を返します。
	EXP 関数 (p. 530)	指定された累乗の e 定数の値を返します。
	FACT 関数 (p. 532)	階乗を計算します。
	FNONCT 関数 (p. 659)	F 分布の非心度パラメーターの値を返します。

カテゴリ	言語要素	説明
	GAMMA 関数 (p. 665)	ガンマ関数の値を返します。
	GCD 関数 (p. 671)	整数の集合に対する最大公約数を返します。
	IBESSEL 関数 (p. 701)	変形ベッセル関数の値を返します。
	IFN_164 関数 (p. 702)	式の true、false、欠損に基づいて数値を返します。
	JBESSEL 関数 (p. 788)	ベッセル関数の値を返します。
	LCM 関数 (p. 804)	数の集合に対する最小公倍数を返します。
	LGAMMA 関数 (p. 816)	ガンマ関数の自然対数を返します。
	LOG 関数 (p. 818)	数値式の自然対数(底 e)を返します。
	LOG10 関数 (p. 819)	数値式の 10 を底とする対数を返します。
	LOG1PX 関数 (p. 820)	1 に引数を加えた値の対数を返します。
	LOG2 関数 (p. 822)	数値式の 2 を底とする対数を返します。
	LOGBETA 関数 (p. 823)	ベータ関数の対数を返します。
	LOGISTIC 関数 (p. 827)	引数のロジスティック変換を返します。
	MOD 関数 (p. 859)	第 1 引数を第 2 引数で除算したときの(最も期待しない浮動小数点の結果を避けるためにファジー化した)余りを返します。
	MODZ 関数 (p. 861)	第 1 引数を第 2 引数で除算したときの余りを、ゼロファジーを使用して返します。
	POWER 関数 (p. 974)	指定された累乗の数値式の値を返します。
	SIGN 関数 (p. 1137)	数値式の符号を示す数字を返します。
	SQRT 関数 (p. 1148)	値の平方根を返します。
	TNONCT 関数 (p. 1195)	スチューデントの t 分布から非心度パラメーターの値を返します。
	TRIGAMMA 関数 (p. 1215)	トリガンマ関数の値を返します。
	WHICHN 関数 (p. 1262)	数値のリストから数値の最初の位置を返します。
数値	CHOOSEN 関数 (p. 416)	引数のリストからの選択結果を表す数値を返します。
双曲線	ARCOSH 関数 (p. 310)	逆双曲線余弦を返します。
	ARSINH 関数 (p. 313)	逆双曲線正弦を返します。
	ARTANH 関数 (p. 314)	逆双曲線正接を返します。

カテゴリ	言語要素	説明
	SINH 関数 (p. 1139)	双曲線正弦を返します。
	TANH 関数 (p. 1186)	双曲線正接を返します。
特殊	DIF 関数 (p. 514)	引数とその n 番目のラグの差分を返します。
	FMTINFO 関数 (p. 657)	SAS 出力形式または入力形式の情報を返します。
	INPUTC 関数 (p. 712)	実行時に文字の入力形式を指定できるようにします。
	INPUTN 関数 (p. 714)	実行時に数値の入力形式を指定できるようにします。
	INPUTN_164 関数 (p. 715)	実行時に数値の入力形式を指定します。
	LAG 関数 (p. 794)	キューから値を返します。
	MISSING 関数 (p. 855)	引数に欠損値が含まれるかどうかを示す数字を返します。
	N 関数 (p. 867)	非 null または非欠損の数値の数を返します。
	NULL 関数 (p. 909)	引数が null の場合は 1 を返し、引数が null でない場合は 0 を返します。
	PROGRAMBLOCKLINENUMBER 関数 (p. 1013)	現在のプログラムブロックの先頭から始まる現在のステートメントの行番号を返します。
	PROGRAMBLOCKLNAME 関数 (p. 1015)	現在のプログラミングブロックの名前を返します。
	PROGRAMBLOCKTYPE 関数 (p. 1017)	現在のプログラミングブロックのタイプを返します。
	PUT 関数 (p. 1038)	指定した出力形式を使用して値を返します。
	PUTC 関数 (p. 1042)	実行時に文字の出力形式を指定できるようにします。
	PUTN 関数 (p. 1045)	実行時に数値の出力形式を指定できるようにします。
	SASLOGLINENUMBER 関数 (p. 1103)	SAS ログ内の現在のステートメントの行番号を返します。
	SLEEP 関数 (p. 1142)	指定した期間、この関数を呼び出すプログラムの実行を中断します。
	SQLEXEC 関数 (p. 1146)	FedSQL ステートメントを実行して、テーブルを作成、削除、または更新するか、テーブルに行を挿入します。
	SYSGET 関数 (p. 1184)	指定された動作環境変数の値を返します。
	UUIDGEN 関数 (p. 1230)	ユニバーサル一意識別子(UUID)の短い形式を返します。
配列	DIM 関数 (p. 518)	配列にある要素数を返します。

カテゴリ	言語要素	説明
	HBOUND 関数 (p. 691)	配列の上限を返します。
	LBOUND 関数 (p. 801)	配列の下限を返します。
	NDIMS 関数 (p. 868)	配列の次元数を返します。
日付と時間	DATDIF 関数 (p. 492)	指定された日数計算規則に従って 2 つの日付間の差を計算した後に、その日付間の日数を返します。
	DATE 関数 (p. 495)	SAS 日付値として現在の日付を返します。
	DATEJUL 関数 (p. 496)	ユリウス暦の日付を SAS 日付値に変換します。
	DATEPART 関数 (p. 498)	SAS 日時値から日付を抽出します。
	DATETIME 関数 (p. 499)	SAS 日時値として現在の日時を返します。
	DAY 関数 (p. 500)	SAS 日付値として月の日を返します。
	DHMS 関数 (p. 511)	日、時、分、秒の値から SAS 日時値を返します。
	HMS 関数 (p. 693)	SAS 時間値(時、分および秒)を返します。
	HOLIDAY 関数 (p. 695)	指定した年の特定の祝日の SAS 日付値を返します。
	HOUR 関数 (p. 699)	SAS 時間値または SAS 日時値の時間を返します。
	INTCINDEX 関数 (p. 719)	サイクルインデックスを返します。この関数には、日付、時間またはタイムスタンプの間隔と値を指定します。
	INTCK 関数 (p. 724)	DOUBLE としてエンコードされた 2 つの SAS 日付、時間、またはタイムスタンプ値の間にある特定の種類の間隔境界の数を返します。
	INTCYCLE 関数 (p. 732)	次に高い季節サイクルでの期間(日付、時間または日時の間隔)を返します。この関数には、期間(日付、時間または日時の間隔)を指定します。
	INTDT 関数 (p. 738)	DATE 値に追加する日数を指定します。
	INTFIT 関数 (p. 739)	2 つの日付に基づく時間間隔を返します。
	INTGET 関数 (p. 741)	3 つの日付値または日時値に基づく時間間隔を返します。
	INTINDEX 関数 (p. 745)	季節インデックスを返します。この関数には、日付、時間またはタイムスタンプの間隔と値を指定します。
	INTNEST 関数 (p. 752)	より大きな間隔の期間に収まる小さい間隔の全期間の数を計算します。
	INTNX 関数 (p. 755)	DOUBLE としてエンコードされた SAS 日付、時間または日時の値を増分し、DOUBLE としてエンコードされた SAS 日付、時間または日時の値を返します。

カテゴリ	言語要素	説明
	INTSEAS 関数 (p. 767)	季節サイクルの長さを返します。この関数には、日付、時間または日時の間隔を指定します。
	INTSHIFT 関数 (p. 772)	ベース間隔に対応するシフト間隔を返します。
	INTTEST 関数 (p. 776)	時間間隔が有効な場合は 1、時間間隔が無効な場合は 0 を返します。
	INTTS 関数 (p. 779)	TIMESTAMP 値に追加する秒数を指定します。
	JULDATE 関数 (p. 789)	SAS 日付値からユリウス暦の日付を返します。
	JULDATE7 関数 (p. 791)	SAS 日付値から 7 桁のユリウス暦の日付を返します。
	MDY 関数 (p. 847)	月、日および年の値から SAS 日付値を返します。
	MINUTE 関数 (p. 854)	SAS 時間または日付値から分を返します。
	MONTH 関数 (p. 863)	SAS 日付値から月を表す数字を返します。
	NWKDOM 関数 (p. 911)	指定した月および年の n 番目に発生する曜日の日付を返します。
	QTR 関数 (p. 1051)	SAS 日付値から年の四半期を返します。
	SECOND 関数 (p. 1126)	SAS 時間または日時値から秒を返します。
	TIME 関数 (p. 1187)	数値の SAS 時間値として現在時刻を返します。
	TIMEPART 関数 (p. 1189)	SAS 日時値から時間値を抽出します。
	TO_DATE 関数 (p. 1197)	SAS 日付値を指定する DOUBLE 値から DATE 値を返します。
	TO_DOUBLE 関数 (p. 1198)	DATE、TIME、または TIMESTAMP 値から、SAS 日付、時間、または日時の値を指定する DOUBLE 値を返します。
	TO_TIME 関数 (p. 1202)	SAS 時間値を指定する DOUBLE 値から TIME 値を返します。
	TO_TIMESTAMP 関数 (p. 1203)	SAS 時間値を指定する DOUBLE 値から TIMESTAMP 値を返します。
	TODAY 関数 (p. 1205)	数値の SAS 日付値として現在の日付を返します。
	WEEK 関数 (p. 1255)	週番号の値を返します。
	WEEKDAY 関数 (p. 1259)	SAS 日付値から、曜日に対応する整数を返します。
	YEAR 関数 (p. 1263)	SAS 日付値から年を返します。
	YRDIF 関数 (p. 1267)	指定した日数計算規則に従って 2 つの日付の差を年単位で返します。人の年齢を返します。

カテゴリ	言語要素	説明
	YYQ 関数 (p. 1270)	年と四半期の値から SAS 日付値を返します。
ビットごとの論理演算	BAND 関数 (p. 319)	2 つの引数のビットごとの論理積を返します。
	BHAMMING_32 関数 (p. 324)	2 つの 32 ビット整数値間のビットごとのハミング距離を返します。
	BHAMMING_HEX 関数 (p. 325)	2 つの 16 進数ビット文字列間のビットごとのハミング距離を返します。
	BLSHIFT 関数 (p. 337)	2 つの引数を使ってビットを論理左シフトした値を返します。
	BNOT 関数 (p. 338)	引数のビットごとの論理否定を返します。
	BOR 関数 (p. 339)	2 つの引数のビットごとの論理和を返します。
	BRSHIFT 関数 (p. 340)	2 つの引数を使ってビットを論理右シフトした値を返します。
	BXOR 関数 (p. 342)	2 つの引数のビットごとの排他的論理和を返します。
分位点	BETAINV 関数 (p. 322)	ベータ分布から分位点を返します。
	GAMINV 関数 (p. 663)	ガンマ分布から分位点を返します。
	PROBIT 関数 (p. 994)	標準正規分布の分位点を返します。
	QUANTILE 関数 (p. 1052)	左側確率(CDF)を指定するときに分布から分位点を返します。
	SQUANTILE 関数 (p. 1149)	右側確率(SDF)の指定した場合に分布から分位点を返します。
	TINV 関数 (p. 1193)	t 分布から分位点を返します。
変数情報	VFORMAT 関数 (p. 1235)	指定したグローバル変数に関連付けられた出力形式を返します。
	VINARRAY 関数 (p. 1239)	指定した変数が配列のメンバーかどうかを示す値を返します。
	VINFORMAT 関数 (p. 1241)	指定した変数に関連付けられた入力形式を返します。
	VLABEL 関数 (p. 1243)	指定した変数に関連付けられているラベルを返します。
	VLENGTH 関数 (p. 1245)	指定した変数のサイズを返します。
	VNAME 関数 (p. 1247)	指定した変数の名前を返します。
	VTYPE 関数 (p. 1253)	変数に関連付けられているデータ型のフルネームを返します。

カテゴリ	言語要素	説明
変数の割り当て	VGETVALUE 関数 (p. 1237)	指定された変数の値を取得し、それを別の変数に割り当てます。
	VSETMISSING 関数 (p. 1248)	指定された変数に欠損値または null 値を割り当てます。
	VSETVALUE 関数 (p. 1252)	変数を指定された値に設定します。
文字	ANYALNUM 関数 (p. 276)	文字列から英数字を検索し、その文字が見つかった最初の文字位置を返します。
	ANYALPHA 関数 (p. 279)	文字列からアルファベット文字を検索し、その文字が見つかった最初の文字位置を返します。
	ANYCNTRL 関数 (p. 282)	文字列から制御文字を検索し、その文字が見つかった最初の文字位置を返します。
	ANYDIGIT 関数 (p. 284)	文字列から数字を検索し、最初に検索された数字の文字位置を返します。
	ANYFIRST 関数 (p. 286)	VALIDVARNAME=V7 の SAS 変数名の開始文字として有効な文字を文字列から検索し、最初に検索された文字の文字位置を返します。
	ANYGRAPH 関数 (p. 289)	文字列からグラフィカル文字を検索し、その文字が見つかった最初の文字位置を返します。
	ANYLOWER 関数 (p. 292)	文字列から小文字を検索し、その文字が見つかった最初の文字位置を返します。
	ANYNAME 関数 (p. 294)	VALIDVARNAME=V7 の SAS 変数名として有効な文字を文字列から検索し、最初に検索された文字の文字位置を返します。
	ANYPRINT 関数 (p. 296)	文字列から印刷可能文字を検索し、その文字が見つかった最初の文字位置を返します。
	ANYPUNCT 関数 (p. 299)	文字列から句読文字を検索し、その文字が見つかった最初の文字位置を返します。
	ANYSPACE 関数 (p. 302)	空白文字(空白、水平タブと垂直タブ、キャリッジリターン、ラインフィード、フォームフィード)を文字列で検索し、その文字が見つかった最初の文字位置を返します。
	ANYUPPER 関数 (p. 304)	文字列から大文字を検索し、その文字が見つかった最初の文字位置を返します。
	ANYXDIGIT 関数 (p. 307)	数字を表す 16 進法の文字を文字列から検索し、最初に検索された文字の文字位置を返します。
	BYTE 関数 (p. 343)	ASCII 照合順序で 1 文字を返します。
	CAT 関数 (p. 344)	先頭または末尾の空白を削除せずに、連結文字列を返します。

カテゴリ	言語要素	説明
	CATQ 関数 (p. 346)	区切り文字を使用して各項目を区切り、区切り文字を含む文字列に引用符を追加して文字値または数値を連結します。
	CATS 関数 (p. 351)	先頭と末尾の空白を削除して、連結文字列を返します。
	CATT 関数 (p. 353)	末尾の空白を削除して、連結文字列を返します。
	CATX 関数 (p. 356)	先頭と末尾の空白を削除し、区切り文字を挿入して、連結文字列を返します。
	CHOOSEC 関数 (p. 414)	引数のリストからの選択結果を表す文字値を返します。
	CMP 関数 (p. 419)	末尾の空白を含む 2 つの文字列を比較します。
	CMPT 関数 (p. 420)	末尾の空白を除く 2 つの文字列を比較します。
	COALESCEC 関数 (p. 426)	文字引数のリストから最初の非 null または非欠損値を返します。
	COMPARE 関数 (p. 429)	2 つの文字列を比較し、異なる文字が検出された場合には最も左にある文字の位置を返し、異なる文字が検出されない場合には 0 を返します。
	COMPBL 関数 (p. 433)	文字列内の単語間にある複数の空白を取り除きます。
	COMPGED 関数 (p. 440)	2 つの文字列間の一般化編集距離を返します。
	COMPLEV 関数 (p. 447)	2 つの文字列間の Levenshtein の編集距離を返します。
	COMPRESS 関数 (p. 452)	元の文字から指定した文字を削除した文字列を返します。
	COUNT 関数 (p. 472)	指定した部分文字列が文字列内に含まれる個数を数えます。
	COUNTC 関数 (p. 474)	文字のリストに表示される(または表示されない)文字列内の文字の個数を数えます。
	COUNTW 関数 (p. 478)	文字列中の単語数を数えます。
	DEQUOTE 関数 (p. 502)	単一引用符で始まる文字列から一致する単一引用符を削除し、閉じ引用符の右側にあるすべての文字を削除します。
	FIND 関数 (p. 628)	文字列内の特定の部分文字列を検索します。
	FINDC 関数 (p. 631)	文字のリストにある文字を文字列から検索します。
	FINDW 関数 (p. 640)	文字列の単語の文字位置または文字列の単語の数を返します。
	INDEX 関数 (p. 705)	文字式から文字列を検索し、最初に検索された文字列の最初の文字の位置を返します。
	INDEXC 関数 (p. 707)	文字式から指定文字を検索し、そのいずれかの文字の最初の出現位置を返します。

カテゴリ	言語要素	説明
	INDEXW 関数 (p. 709)	文字式から単語として指定した文字列を検索し、単語の最初の文字の位置を返します。
	LEFT 関数 (p. 806)	文字式を左揃えにします。
	LENGTH 関数 (p. 808)	末尾の空白を除いた文字列の長さを文字数で返します。
	LENGTHC 関数 (p. 810)	末尾の空白を含めた文字列の長さを文字数で返します。
	LENGTHM 関数 (p. 813)	文字列に割り当て可能なメモリの量をバイト単位で返します。
	LOWCASE 関数 (p. 833)	文字式のすべての文字を小文字に変換します。
	MD5 関数 (p. 845)	指定した文字列のメッセージダイジェストの結果をバイナリ形式で返します。
	NOTALNUM 関数 (p. 876)	文字列から英数字でない文字を検索し、その文字が見つかった最初の文字位置を返します。
	NOTALPHA 関数 (p. 878)	文字列からアルファベットでない文字を検索し、その文字が見つかった最初の文字位置を返します。
	NOTCNTRL 関数 (p. 881)	文字列から制御文字でない文字を検索し、その文字が見つかった最初の文字位置を返します。
	NOTDIGIT 関数 (p. 883)	文字列から数字でない文字を検索し、その文字が見つかった最初の文字位置を返します。
	NOTFIRST 関数 (p. 886)	VALIDVARNAME=V7 の SAS 変数名として無効な開始文字を文字列から検索し、その文字が見つかった最初の文字位置を返します。
	NOTGRAPH 関数 (p. 888)	文字列からグラフィカル文字でない文字を検索し、その文字が見つかった最初の文字位置を返します。
	NOTLOWER 関数 (p. 891)	文字列から小文字でない文字を検索し、その文字が見つかった最初の文字位置を返します。
	NOTNAME 関数 (p. 893)	VALIDVARNAME=V7 の SAS 変数名として無効な文字を文字列から検索し、その文字が見つかった最初の文字位置を返します。
	NOTPRINT 関数 (p. 895)	文字列から印刷不可文字を検索し、その文字が見つかった最初の文字位置を返します。
	NOTPUNCT 関数 (p. 897)	文字列から句読文字でない文字を検索し、その文字が見つかった最初の文字位置を返します。
	NOTSPACE 関数 (p. 900)	空白文字ではない文字(空白、水平タブと垂直タブ、キャリッジリターン、改行、およびフォームフィード)を文字列で検索し、その文字が見つかった最初の文字位置を返します。

カテゴリ	言語要素	説明
	NOTUPPER 関数 (p. 903)	文字列から大文字でない文字を検索し、その文字が見つかった最初の文字位置を返します。
	NOTXDIGIT 関数 (p. 906)	文字列から 16 進文字でない文字を検索し、その文字が見つかった最初の文字位置を返します。
	QUOTE 関数 (p. 1056)	文字値に二重引用符を付加します。
	RANK 関数 (p. 1081)	ASCII 照合順序で文字の位置を返します。
	REPEAT 関数 (p. 1083)	文字式を繰り返します。
	REVERSE 関数 (p. 1084)	文字式を逆にします。
	RIGHT 関数 (p. 1085)	文字式を右揃えにします。
	SCAN 関数 (p. 1109)	文字式から n 番目の単語を返します。
	SHA256 関数 (p. 1131)	指定した文字列のメッセージダイジェストの結果を返します。
	SHA256HEX 関数 (p. 1132)	指定した文字列のメッセージダイジェストの結果を返し、文字列を 16 進表現に変換します。
	SHA256HMACHEX 関数 (p. 1135)	HMAC (Hash-based Message Authentication)アルゴリズムを使用して、指定した文字列のメッセージダイジェストの結果を返します。
	STRIP 関数 (p. 1166)	先頭と末尾の空白を削除して文字列を返します。
	擬似 SUBSTR(割り当ての左辺に用いた場合) 関数 (p. 1168)	文字変数のコンテンツを置き換えます。
	SUBSTR (=の右) 関数 (p. 1172)	引数から部分文字列を抽出します。
	SUBSTRN 関数 (p. 1175)	部分文字列を返します。長さがゼロの結果も返せます。
	TRANSLATE 関数 (p. 1206)	文字式内の特定の文字を置き換えます。
	TRANSTRN 関数 (p. 1208)	文字列に含まれる特定の部分文字列をすべて 置き換えるか取り除きます。
	TRANWRD 関数 (p. 1211)	文字列に含まれる特定の部分文字列をすべて 置き換えます。
	TRIM 関数 (p. 1217)	文字式から末尾の空白を取り除きます。文字式が欠損値の場合は、長さがゼロの文字列を返します。
	UPCASE 関数 (p. 1224)	引数内のすべての 文字を大文字に変換します。
	VERIFY 関数 (p. 1233)	式に固有の最初の文字の位置を返します。
	WHICHC 関数 (p. 1260)	文字列のリストから文字列の最初の位置を返します。

カテゴリ	言語要素	説明
文字列マッチング	PRXCHANGE 関数 (p. 1019)	パターンマッチングの置換を実行します。
	PRXMATCH 関数 (p. 1023)	パターンの一致を検索し、見つかったパターンの位置を返します。
	PRXPAREN 関数 (p. 1027)	パターンに最後に一致したかっこを返します。
	PRXPARSE 関数 (p. 1031)	文字値のパターンマッチングに使用できる Perl 正規表現 (PRX)をコンパイルします。
	PRXPOSN 関数 (p. 1034)	キャプチャバッファーの値が含まれる文字列を返します。
乱数	RAND 関数 (p. 1058)	指定する分布から乱数を生成します。
	STREAMINIT 関数 (p. 1160)	乱数を生成するための乱数ジェネレーター-とシード値を指定します。

ディクショナリ

ABS 関数

数値式の絶対値を返します。

カテゴリ: 数学

返されるデータの種類: BIGINT、DECIMAL、DOUBLE

注: 2025.12 以降、この関数では、引数が DOUBLE の場合、すべての引数が DOUBLE に変換され、結果は DOUBLE になります。PROC DS2 ステートメントで BIGINTPROCESSING=YES が設定され、いずれかの引数が BIGINT の場合、すべての引数が BIGINT に変換され、結果は BIGINT になります。

構文

ABS(*expression*)

必須引数

expression
数値に評価される有効な式を指定します。

データの種類 BIGINT、DECIMAL、DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

結果が引数のデータ型の範囲に収まらない数値の場合、ABS 関数は失敗します。

この関数の引数が数値以外の場合、その引数は DOUBLE に変換されます。いずれかの引数が DOUBLE または REAL の場合、すべての引数が(まだ変換されていなければ) DOUBLE に変換され、結果は DOUBLE になります。それ以外の場合で、いずれかの引数が DECIMAL のときは、すべての引数が(まだ変換されていなければ) DECIMAL に変換され、結果は DECIMAL になります。それ以外の場合は、すべての引数が BIGINT に変換され、結果は BIGINT になります。

例

例 1: DECIMAL と負の数の絶対値

次のプログラムは、DECIMAL と負の数の絶対値を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double x y;
  method run();
    x=abs(2.4);
    y=abs(-3);
    put x= y=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
x=2.4 y=3
```

例 2: 数式の絶対値

次のプログラムは、単純な数式の絶対値を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double z;
  method run();
    z=abs((3 * 50.5) / 5);
    put z=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
z=30.3
```

AIRY 関数

Airy 関数の値を返します。

カテゴリ:	数学
返されるデータの種類:	DOUBLE

構文

AIRY(*x*)

必須引数

x
数値の定数、変数または式を指定します。

データの種類 DOUBLE

詳細

AIRY 関数は、Airy 関数の値を返します。 (リファレンスのリストを参照。)これは、微分方程式

$$w^{(2)} - xw = 0$$

の解です(以下は条件)。

$$w(0) = \frac{1}{3^{\frac{2}{3}}\Gamma\left(\frac{2}{3}\right)}$$

および

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

例

次のプログラムは、AIRY 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double x y;
  method init();
    x=airy(2.0);
    y=airy(-2.0);
    put x= y=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
x=0.03492413042327 y=0.22740742820168
```

ANYALNUM 関数

文字列から英数字を検索し、その文字が見つかった最初の文字位置を返します。

カテゴリ:	文字
返されるデータの種類:	DOUBLE

構文

ANYALNUM('*expression*'[, *start*])

必須引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、VARCHAR、NVARCHAR

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

オプション引数

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

ANYALNUM 関数の結果は、有効になっている変換テーブル(“TRANTAB= システムオプション” (*SAS 各国語サポート(NLS): リファレンスガイド*)を参照)に直接依存し、ENCODING および LOCALE システムオプションに間接的に依存します。

ANYALNUM 関数は、文字列で最初に出現する数字、大文字または小文字を検索します。このような文字が検出されると、ANYALNUM はその文字の文字列の位置を返します。このような文字が検出されないと、ANYALNUM は 0 の値を返します。

1 つの引数のみを使用する場合、ANYALNUM は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

ANYALNUM は、次のいずれかに該当する場合にゼロ値を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

ANYALNUM 関数は、文字列から英数字を検索します。NOTALNUM 関数は、文字列から英数字でない文字を検索します。

例

例 1: 左から右への文字列のスキャン

次のプログラムでは、ANYALNUM 関数を使用して文字列の左から右へ英数字を検索します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data ltrtest;
  dcl char(15) string c;
  dcl double j;
  method run();
    string='Next = Last + 1';
    j=0;
    do until (j=0);
      j=anyalnum(string, j+1);
      if j=0 then put 'The end';
      else do;
        c=substr(string, j, 1);
        put j= c=;
      end;
    end;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
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
The end
```

例 2: 右から左への文字列のスキャン

次のプログラムでは、ANYALNUM 関数を使用して文字列の右から左へ英数字を検索します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data rtltest;
  dcl char(15) string c;
```

```
  dcl double j;
  method run();
    string='Next = Last + 1;';
    j=999999;
    do until(j=0);
      j=anyalnum(string, 1-j);
      if j=0 then put 'The end';
      else do;
        c=substr(string, j, 1);
      put j= c=;
      end;
    end;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
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
The end
```

関連項目

関数:

- “NOTALNUM 関数” (876 ページ)

ANYALPHA 関数

文字列からアルファベット文字を検索し、その文字が見つかった最初の文字位置を返します。

カテゴリ:	文字
返されるデータの種類:	DOUBLE

構文

ANYALPHA('*expression*'[, *start*])

必須引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、VARCHAR、NVARCHAR

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

オプション引数

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

ANYALPHA 関数の結果は、有効になっている変換テーブル(“TRANTAB= システムオプション” (*SAS 各国語サポート(NLS): リファレンスガイド*)を参照)に直接依存し、ENCODING および LOCALE システムオプションに間接的に依存します。

ANYALPHA 関数は、文字列で最初に出現する大文字または小文字を検索します。このような文字が検出されると、ANYALPHA はその文字の文字列の位置を返します。このような文字が検出されないと、ANYALPHA は 0 の値を返します。

1 つの引数のみを使用する場合、ANYALPHA は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

ANYALPHA は、次のいずれかに該当する場合にゼロ値を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

ANYALPHA 関数は、文字列からアルファベットを検索します。NOTALPHA 関数は、文字列からアルファベットでない文字を検索します。

例

例 1: 文字列でアルファベット文字を左から右に検索

次のプログラムでは、ANYALPHA 関数を使用して文字列の左から右へアルファベット文字を検索します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl char(18) string c;
  dcl double j i;
  method run();
    string='Next = _n_ + 12E3;';
    j=0;
    do until(j=0);
      j=anyalpha(string, j+1);
      if j=0 then put 'The end';
      else do;
        c=substr(string, j, 1);
        put j= c=;
      end;
    end;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
j=1 c=N
j=2 c=e
j=3 c=x
j=4 c=t
j=9 c=n
j=16 c=E
The end
```

例 2: ANYALPHA 関数を使用したアルファベット文字の識別

次のプログラムを実行して、ANYALPHA 関数で識別されるアルファベット文字を表示できます。

注: このプログラムは、シングルバイトエンコーディングでのみ機能します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data testany;
```

```
  dcl nchar(3) byte1 hex1;
  dcl double dec anyalpha1;

  method run();
    do dec=0 to 255;
      byte1=byte(dec);
      hex1=put(dec,hex2.);
      anyalpha1=anyalpha(byte1);
      output;
    end;
  end;
enddata;
run;
quit;

proc print data=testany;
run;
```

関連項目

関数:

- “NOTALPHA 関数” (878 ページ)

ANYCNTRL 関数

文字列から制御文字を検索し、その文字が見つかった最初の文字位置を返します。

カテゴリ:	文字
返されるデータの種類:	DOUBLE

構文

ANYCNTRL('*expression*' [, *start*])

必須引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類	CHAR、NCHAR、VARCHAR、NVARCHAR
参照項目:	“DS2 式” (*SAS DS2 プログラマガイド*)

オプション引数

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

ANYCNTRL 関数の結果は、有効になっている変換テーブル(“TRANTAB= システムオプション” (*SAS 各国語サポート(NLS): リファレンスガイド*)を参照)に直接依存し、ENCODING および LOCALE システムオプションに間接的に依存します。

ANYCNTRL 関数は、文字列で最初に出現する制御文字を検索します。このような文字が検出されると、ANYCNTRL はその文字の文字列の位置を返します。このような文字が検出されないと、ANYCNTRL は 0 の値を返します。

1 つの引数のみを使用する場合、ANYCNTRL は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

次のいずれかの条件が満たされると、ANYCNTRL は値に 0 を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

ANYCNTRL 関数は、文字列から制御文字を検索します。NOTCNTRL 関数は、文字列から制御文字でない文字を検索します。

例

次のプログラムを実行して、ANYCNTRL 関数で識別される制御文字を表示できます。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
```

```
data testany (overwrite=yes);
  dcl nchar(3) byte1 hex1;
  dcl double dec anycntrl1;

  method run();
    do dec=0 to 255;
      byte1=byte(dec);
      hex1=put(dec,hex2.);
      anycntrl1=anycntrl(byte1);
      if anycntrl1 then output;
    end;
  end;
enddata;
run;
quit;

proc print data=testany;
run;
```

関連項目

関数:

- “NOTCNTRL 関数” (881 ページ)

ANYDIGIT 関数

文字列から数字を検索し、最初に検索された数字の文字位置を返します。

カテゴリ:	文字
返されるデータの種類:	DOUBLE

構文

ANYDIGIT('*expression*'[, *start*])

必須引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、VARCHAR、NVARCHAR

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

オプション引数

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

ANYDIGIT 関数は、TRANTAB、ENCODING または LOCALE システムオプションに依存しません。

ANYDIGIT 関数は、文字列で最初に出現する数字を検索します。このような文字が検出されると、ANYDIGIT はその文字の文字列の位置を返します。このような文字が検出されないと、ANYDIGIT は 0 の値を返します。

1 つの引数のみを使用する場合、ANYDIGIT は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

ANYDIGIT は、次のいずれかに該当する場合にゼロ値を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

ANYDIGIT 関数は、文字列から数字を検索します。NOTDIGIT 関数は、文字列から数字でない文字を検索します。

例

次のプログラムでは、ANYDIGIT 関数を使用して数字を検索します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
```

```
  dcl char(18) string c;
  dcl double j i;
  method run();
    string='Next = _n_ + 12E3;';
    j=0;
    do until(j=0);
      j=anydigit(string, j+1);
      if j=0 then put 'The end';
      else do;
        c=substr(string, j, 1);
        put j= c=;
      end;
    end;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
j=14 c=1
j=15 c=2
j=17 c=3
The end
```

関連項目

関数:

- “NOTDIGIT 関数” (883 ページ)

ANYFIRST 関数

VALIDVARNAME=V7 の SAS 変数名の開始文字として有効な文字を文字列から検索し、最初に検索された文字の文字位置を返します。

カテゴリ:	文字
返されるデータの種類:	DOUBLE

構文

ANYFIRST('*expression*'[, *start*])

必須引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、VARCHAR、NVARCHAR

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

オプション引数

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

ANYFIRST 関数は、TRANTAB、ENCODING または LOCALE システムオプションに依存しません。

ANYFIRST 関数は、VALIDVARNAME=V7 の SAS 変数名の開始文字として有効な文字のうち最初に出現する文字を文字列から検索します。アンダースコア(_)および英語の大文字または小文字が有効な文字になります。このような文字が検出されると、ANYFIRST はその文字の文字列の位置を返します。このような文字が検出されないと、ANYFIRST は 0 の値を返します。

1 つの引数のみを使用する場合、ANYFIRST は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

ANYFIRST は、次のいずれかに該当する場合にゼロ値を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

ANYFIRST 関数は、VALIDVARNAME=V7 の SAS 変数名の開始文字として有効な文字のうち最初に出現する文字を文字列から検索します。NOTFIRST 関数は、

VALIDVARNAME=V7 の SAS 変数名の開始文字として有効でない文字のうち最初に出現する文字を文字列から検索します。

例

次のプログラムでは、ANYFIRST 関数を使用して VALIDVARNAME=V7 の SAS 変数名の開始文字として有効な文字を文字列から検索します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl char(18) string c;
  dcl double j i;
  method run();
    string='Next = _n_ + 12E3;';
    j=0;
    do until(j=0);
      j=anyfirst(string, j+1);
      if j=0 then put 'The end';
      else do;
        c=substr(string, j, 1);
        put j= c=;
      end;
    end;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
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=16 c=E
The end
```

関連項目

関数:

- “NOTFIRST 関数” (886 ページ)

ANYGRAPH 関数

文字列からグラフィカル文字を検索し、その文字が見つかった最初の文字位置を返します。

カテゴリ: 文字

返されるデータの種類: DOUBLE

構文

ANYGRAPH('*string*'[, *start*])

必須引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、VARCHAR、NVARCHAR

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

オプション引数

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

ANYGRAPH 関数の結果は、有効になっている変換テーブル("TRANTAB= システムオプション" (*SAS 各国語サポート(NLS): リファレンスガイド*)を参照)に直接依存し、ENCODING および LOCALE システムオプションに間接的に依存します。

ANYGRAPH 関数は、文字列で最初に出現するグラフィカル文字を検索します。グラフィカル文字は、空白以外の印刷可能文字として定義されます。このような文字が検出されると、ANYGRAPH はその文字の文字列の位置を返します。このような文字が検出されないと、ANYGRAPH は 0 の値を返します。

1 つの引数のみを使用する場合、ANYGRAPH は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。

- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

ANYGRAPH は、次のいずれかに該当する場合にゼロ値を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

ANYGRAPH 関数は、文字列からグラフィカル文字を検索します。NOTGRAPH 関数は、文字列からグラフィカル文字でない文字を検索します。

例

例 1: 文字列からのグラフィカル文字の検索

次のプログラムでは、ANYGRAPH 関数を使用して文字列からグラフィカル文字を検索します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl char(18) string c;
  dcl double j i;
  method run();
    string='Next = _n_ + 12E3;';
    j=0;
    do until(j=0);
      j=anygraph(string, j+1);
      if j=0 then put 'The end';
      else do;
        c=substr(string, j, 1);
        put j= c=;
      end;
    end;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
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=;
The end
```

例 2: ANYGRAPH 関数を使用した制御文字の識別

次のプログラムを実行して、ANYGRAPH 関数で識別される制御文字を表示できます。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data testany (overwrite=yes);
  dcl nchar(3) byte1 hex1;
  dcl double dec anygraph1;

  method run();
    do dec=0 to 255;
      byte1=byte(dec);
      hex1=put(dec,hex2.);
      anygraph=anygraph(byte1);
      output;
    end;
  end;
enddata;
run;
quit;

proc print data=testany;
run;
```

関連項目

関数:

- “NOTGRAPH 関数” (888 ページ)

ANYLOWER 関数

文字列から小文字を検索し、その文字が見つかった最初の文字位置を返します。

カテゴリ: 文字

返されるデータの種類: DOUBLE

構文

ANYLOWER('*expression*'[, *start*])

必須引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、VARCHAR、NVARCHAR

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

オプション引数

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

ANYLOWER 関数の結果は、有効になっている変換テーブル("TRANTAB= システムオプション" (*SAS 各国語サポート(NLS): リファレンスガイド*)を参照)に直接依存し、ENCODING および LOCALE システムオプションに間接的に依存します。

ANYLOWER 関数は、文字列で最初に出現する小文字を検索します。このような文字が検出されると、ANYLOWER はその文字の文字列の位置を返します。このような文字が検出されないと、ANYLOWER は 0 の値を返します。

1 つの引数のみを使用する場合、ANYLOWER は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。

- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

ANYLOWER は、次のいずれかに該当する場合にゼロ値を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

ANYLOWER 関数は、文字列から小文字を検索します。NOTLOWER 関数は、文字列から小文字でない文字を検索します。

例

次のプログラムでは、ANYLOWER 関数を使用して文字列から小文字を検索します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl char(18) string c;
  dcl double j i;
  method run();
    string='Next = _n_ + 12E3;';
    j=0;
    do until(j=0);
      j=anylower(string, j+1);
      if j=0 then put 'The end';
      else do;
        c=substr(string, j, 1);
        put j= c=;
      end;
    end;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
j=2 c=e
j=3 c=x
j=4 c=t
j=9 c=n
The end
```

関連項目

関数:

- “NOTLOWER 関数” (891 ページ)

ANYNAME 関数

VALIDVARNAME=V7 の SAS 変数名として有効な文字を文字列から検索し、最初に検索された文字の文字位置を返します。

カテゴリ: 文字

返されるデータの種類: DOUBLE

構文

ANYNAME('*expression*'[,*start*])

必須引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、VARCHAR、NVARCHAR

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

オプション引数

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

ANYNAME 関数は、TRANTAB、ENCODING または LOCALE システムオプションに依存しません。

ANYNAME 関数は、VALIDVARNAME=V7 の SAS 変数名で有効な文字のうち最初に出現する文字を文字列から検索します。アンダースコア(_)、数字、英語の大文字または小文字が有効な文字になります。このような文字が検出されると、ANYNAME

はその文字の文字列の位置を返します。このような文字が検出されないと、ANYNAME は 0 の値を返します。

1 つの引数のみを使用する場合、ANYNAME は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

ANYNAME は、次のいずれかに該当する場合にゼロ値を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

ANYNAME 関数は、VALIDVARNAME=V7 の SAS 変数名で有効な文字のうち最初に出現する文字を文字列から検索します。NOTNAME 関数は、VALIDVARNAME=V7 の SAS 変数名で有効でない文字のうち最初に出現する文字を文字列から検索します。

例

次のプログラムでは、ANYNAME 関数を使用して VALIDVARNAME=V7 の SAS 変数名として有効な文字を文字列から検索します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl char(18) string c;
  dcl double j i;
  method run();
    string='Next = _n_ + 12E3;';
    j=0;
    do until(j=0);
      j=anyname(string, j+1);
      if j=0 then put 'The end';
      else do;
        c=substr(string, j, 1);
        put j= c=;
      end;
    end;
  end;
enddata;
```

```
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
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=1
j=15 c=2
j=16 c=E
j=17 c=3
The end
```

関連項目

関数:

- “NOTNAME 関数” (893 ページ)

ANYPRINT 関数

文字列から印刷可能文字を検索し、その文字が見つかった最初の文字位置を返します。

カテゴリ:	文字
返されるデータの種類:	DOUBLE

構文

ANYPRINT('*expression*'[, *start*])

必須引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類	CHAR、NCHAR、VARCHAR、NVARCHAR
参照項目:	“DS2 式” (*SAS DS2 プログラマガイド*)

オプション引数

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

ANYPRINT 関数の結果は、有効になっている変換テーブル(“TRANTAB= システムオプション” (*SAS 各国語サポート(NLS): リファレンスガイド*)を参照)に直接依存し、ENCODING および LOCALE システムオプションに間接的に依存します。

ANYPRINT 関数は、文字列で最初に出現する印刷可能文字を検索します。このような文字が検出されると、ANYPRINT はその文字の文字列の位置を返します。このような文字が検出されないと、ANYPRINT は 0 の値を返します。

1 つの引数のみを使用する場合、ANYPRINT は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

ANYPRINT は、次のいずれかに該当する場合にゼロ値を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

ANYPRINT 関数は、文字列から印刷可能文字を検索します。NOTPRINT 関数は、文字列から印刷不可文字を検索します。

例

例 1: 文字列からの印刷可能文字の検索

次のプログラムでは、ANYPRINT 関数を使用して文字列から印刷可能文字を検索します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl char(18) string c;
  dcl double j i;
  method run();
    string='Next = _n_ + 12E3;';
    j=0;
    do until(j=0);
      j=anyprint(string, j+1);
      if j=0 then put 'The end';
      else do;
        c=substr(string, j, 1);
        put j= c=;
      end;
    end;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
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=;
The end
```

例 2: ANYPRINT 関数を使用した制御文字の識別

次のプログラムを実行して、ANYPRINT 関数で識別される制御文字を表示できます。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data testany (overwrite=yes);
  dcl nchar(3) byte1 hex1;
  dcl double dec anyprint1;
```

```
  method run();
    do dec=0 to 255;
      byte1=byte(dec);
      hex1=put(dec,hex2.);
      anyprint1=anyprint(byte1);
      output;
    end;
  end;
enddata;
run;
quit;

proc print data=testany;
run;
```

関連項目

関数:

- “NOTPRINT 関数” (895 ページ)

ANYPUNCT 関数

文字列から句読文字を検索し、その文字が見つかった最初の文字位置を返します。

カテゴリ:	文字
返されるデータの種類:	DOUBLE

構文

ANYPUNCT('*expression*'[, *start*])

必須引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、VARCHAR、NVARCHAR

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

オプション引数

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

ANYPUNCT 関数の結果は、有効になっている変換テーブル(“TRANTAB= システムオプション” (*SAS 各国語サポート(NLS): リファレンスガイド*)を参照)に直接依存し、ENCODING および LOCALE システムオプションに間接的に依存します。

ANYPUNCT 関数は、文字列で最初に出現する句読文字を検索します。このような文字が検出されると、ANYPUNCT はその文字の文字列の位置を返します。このような文字が検出されないと、ANYPUNCT は 0 の値を返します。

1 つの引数のみを使用する場合、ANYPUNCT は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

ANYPUNCT は、次のいずれかに該当する場合にゼロ値を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

ANYPUNCT 関数は、文字列から句読文字を検索します。NOTPUNCT 関数は、文字列から句読文字でない文字を検索します。

例

例 1: 文字列からの句読文字の検索

次のプログラムでは、ANYPUNCT 関数を使用して文字列から句読文字を検索します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl char(18) string c;
  dcl double j i;
  method run();
    string='Next = _n_ + 12E3;';
    j=0;
    do until(j=0);
      j=anypunct(string, j+1);
      if j=0 then put 'The end';
      else do;
        c=substr(string, j, 1);
        put j= c=;
      end;
    end;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
j=8 c=_
j=10 c=_
j=18 c=;
The end
```

例 2: ANYPUNCT 関数を使用した制御文字の識別

次のプログラムを実行すると、ANYPUNCT 関数によって識別される制御文字が表示されます。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data testany (overwrite=yes);
  dcl nchar(3) byte1 hex1;
  dcl double dec anypunct1;

  method run();
    do dec=0 to 255;
      byte1=byte(dec);
      hex1=put(dec,hex2.);
      anypunct1=anypunct(byte1);
      output;
    end;
  end;
enddata;
run;
quit;
```

関連項目

関数:

- “NOTPUNCT 関数” (897 ページ)

ANYSPACE 関数

空白文字(空白、水平タブと垂直タブ、キャリッジリターン、ラインフィード、フォームフィード)を文字列で検索し、その文字が見つかった最初の文字位置を返します。

カテゴリ: 文字

返されるデータの種類: DOUBLE

構文

ANYSPACE('*expression*'[, *start*])

必須引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、VARCHAR、NVARCHAR

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

オプション引数

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

ANYSPACE 関数の結果は、有効になっている変換テーブル(“TRANTAB= システムオプション” (*SAS 各国語サポート(NLS): リファレンスガイド*)を参照)に直接依存し、ENCODING および LOCALE システムオプションに間接的に依存します。

ANYSPACE 関数は、文字列で最初に出現する空白、水平タブ、垂直タブ、キャリッジリターン、ラインフィード、フォームフィードの文字を検索します。このような

文字が検出されると、ANYSPACE はその文字の文字列の位置を返します。このような文字が検出されないと、ANYSPACE は 0 の値を返します。

1 つの引数のみを使用する場合、ANYSPACE は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

ANYSPACE は、次のいずれかに該当する場合にゼロ値を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

ANYSPACE 関数は、文字列で最初に出現する空白、水平タブ、垂直タブ、キャリッジリターン、ラインフィード、フォームフィードの文字を検索します。NOTSPACE 関数は、文字列で最初に出現する空白、水平タブ、垂直タブ、キャリッジリターン、ラインフィード、フォームフィードでない文字を検索します。

例

例 1: 文字列からの空白文字の検索

次のプログラムでは、ANYSPACE 関数を使用して文字列から空白文字を検索します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl char(18) string c;
  dcl double j i;
  method run();
    string='Next = _n_ + 12E3;';
    j=0;
    do until(j=0);
      j=anyspace(string, j+1);
      if j=0 then put 'The end';
      else do;
        c=substr(string, j, 1);
        put j= c=;
      end;
```

```
    end;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
j=5 c=
j=7 c=
j=11 c=
j=13 c=
The end
```

例 2: ANYSPACE 関数を使用した制御文字の識別

次のプログラムを実行すると、ANYSPACE 関数によって識別される制御文字が表示されます。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data testany (overwrite=yes);
  dcl nchar(3) byte1 hex1;
  dcl double dec anyspace1;

  method run();
    do dec=0 to 255;
      byte1=byte(dec);
      hex1=put(dec,hex2.);
      anyspace1=anyspace(byte1);
      output;
    end;
  end;
enddata;
run;
quit;
```

関連項目

関数:

- “NOTSPACE 関数” (900 ページ)

ANYUPPER 関数

文字列から大文字を検索し、その文字が見つかった最初の文字位置を返します。

カテゴリ: 文字

返されるデータの種類: DOUBLE

構文

ANYUPPER('*expression*'[, *start*])

必須引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、VARCHAR、NVARCHAR

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

オプション引数

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

ANYUPPER 関数の結果は、有効になっている変換テーブル("TRANTAB= システムオプション" (*SAS 各国語サポート(NLS): リファレンスガイド*)を参照)に直接依存し、ENCODING および LOCALE システムオプションに間接的に依存します。

ANYUPPER 関数は、文字列で最初に出現する大文字を検索します。このような文字が検出されると、ANYUPPER はその文字の文字列の位置を返します。このような文字が検出されないと、ANYUPPER は 0 の値を返します。

1 つの引数のみを使用する場合、ANYUPPER は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

ANYUPPER は、次のいずれかに該当する場合にゼロ値を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。

- *start* の値が 0 になっている。

比較

ANYUPPER 関数は、文字列から大文字を検索します。NOTUPPER 関数は、文字列から大文字でない文字を検索します。

例

次のプログラムでは、ANYUPPER 関数を使用して文字列から大文字を検索します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl char(18) string c;
  dcl double j i;
  method run();
    string='Next = _n_ + 12E3;';
    j=0;
    do until(j=0);
      j=anyupper(string, j+1);
      if j=0 then put 'The end';
      else do;
        c=substr(string, j, 1);
        put j= c=;
      end;
    end;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
j=1 c=N
j=16 c=E
The end
```

関連項目

関数:

- “NOTUPPER 関数” (903 ページ)

ANYXDIGIT 関数

数字を表す 16 進法の文字を文字列から検索し、最初に検索された文字の文字位置を返します。

カテゴリ: 文字

返されるデータの種類: DOUBLE

構文

ANYXDIGIT('*expression*'[,*start*])

必須引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、VARCHAR、NVARCHAR

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

オプション引数

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

ANYXDIGIT 関数は、TRANTAB、ENCODING または LOCALE システムオプションに依存しません。

ANYXDIGIT 関数は、文字列で最初に出現する数字または A、B、C、D、E、F の大文字や小文字を検索します。このような文字が検出されると、ANYXDIGIT はその文字の文字列の位置を返します。このような文字が検出されないと、ANYXDIGIT は 0 の値を返します。

1 つの引数のみを使用する場合、ANYXDIGIT は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。

- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

ANYXDIGIT は、次のいずれかに該当する場合にゼロ値を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

ANYXDIGIT 関数は、文字列から 16 進文字を検索します。NOTXDIGIT 関数は、文字列から 16 進文字でない文字を検索します。

例

次のプログラムでは、ANYXDIGIT 関数を使用して文字列から数字を表す 16 進文字を検索します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl char(18) string c;
  dcl double j i;
  method run();
    string='Next = _n_ + 12E3;';
    j=0;
    do until(j=0);
      j=anyxdigit(string, j+1);
      if j=0 then put 'The end';
      else do;
        c=substr(string, j, 1);
        put j= c=;
      end;
    end;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
j=2 c=e
j=14 c=1
j=15 c=2
j=16 c=E
j=17 c=3
The end
```

関連項目

関数:

- “NOTXDIGIT 関数” (906 ページ)

ARCOS 関数

逆余弦をラジアンで返します。

カテゴリ:	三角関数
返されるデータの種類:	DOUBLE

構文

ARCOS(*expression*)

必須引数

expression
数値に評価される有効な式を指定します。

範囲	–1 - 1
データの種類	DOUBLE
参照項目:	“DS2 式” (*SAS DS2 プログラマガイド*)

詳細

ARCOS 関数は、引数の逆余弦(逆コサイン)を返します。戻り値はラジアンで示されます。

例

次のプログラムは、ARCOS 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double a b c;
  method run();
    a=arcos(1);
    b=arcos(0);
    c=arcos(-.5);
    put 'a=' a;
    put 'b=' b;
    put 'c=' c;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
a= 0
b= 1.57079632679489
c= 2.09439510239319
```

ARCOSH 関数

逆双曲線余弦を返します。

カテゴリ: 双曲線

返されるデータの種類: DOUBLE

構文

ARCOSH(*expression*)

必須引数

expression
数値に評価される有効な式を指定します。

範囲 *expression* >= 1

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

ARCOSH 関数は、逆双曲線余弦を計算します。ARCOSH 関数は、次の式で数学的に定義されます。ここで、*expression* >= 1 です。この式では、*expression* は *x* で表されます。

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

例

次のプログラムは、ARCOSH 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double a b;
  method run();
    a=arcosh(5);
    b=arcosh(13);
    put 'a=' a;
    put 'b=' b;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
a= 2.29243166956117
b= 3.25661395480005
```

関連項目

関数:

- “ARSINH 関数” (313 ページ)
- “ARTANH 関数” (314 ページ)
- “COSH 関数” (469 ページ)
- “TANH 関数” (1186 ページ)
- “SINH 関数” (1139 ページ)

ARSIN 関数

逆正弦をラジアンで返します。

カテゴリ: 三角関数

返されるデータの種類: DOUBLE

構文

ARSIN(*expression*)

必須引数

expression
数値に評価される有効な式を指定します。

範囲 –1 - 1

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

ARSIN 関数は、引数の逆正弦(逆サイン)を返します。戻り値はラジアンで示されます。

例

次のプログラムは、ARSIN 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double a b c;
  method run();
    a=arsin(0);
    b=arsin(1);
    c=arsin(-0.5);
```

```
    put 'a=' a;
    put 'b=' b;
    put 'c=' c;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
a= 0
b= 1.57079632679489
c= -0.52359877559829
```

ARSINH 関数

逆双曲線正弦を返します。

カテゴリ:	双曲線
返されるデータの種類:	DOUBLE

構文

ARSINH(*expression*)

必須引数

expression
数値に評価される有効な式を指定します。

範囲	−∞< x <∞
データの種類	DOUBLE
参照項目:	"DS2 式" (*SAS DS2 プログラマガイド*)

詳細

ARSINH 関数は、逆双曲線正弦を計算します。ARSINH 関数は、次の式で数学的に定義されます。ここで、$-\infty < x < \infty$です。

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

無限の記号は、マシンで使用できる最大の倍精度浮動小数に置き換えられます。この式では、*expression* は *x* で表されます。

例

次のプログラムは、ARSINH 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double a b ;
  method run();
    a=arsinh(5);
    b=arsinh(-5);
    put 'a=' a;
    put 'b=' b;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
a= 2.31243834127275
b= -2.31243834127275
```

関連項目

関数:

- “ARCOSH 関数” (310 ページ)
- “ARTANH 関数” (314 ページ)
- “COSH 関数” (469 ページ)
- “TANH 関数” (1186 ページ)
- “SINH 関数” (1139 ページ)

ARTANH 関数

逆双曲線正接を返します。

カテゴリ:	双曲線
返されるデータの種類:	DOUBLE

構文

ARTANH(*expression*)

必須引数

expression
数値に評価される有効な式を指定します。

範囲 –1 < *expression* < 1

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

ARTANH 関数は、逆双曲線正接を計算します。ARTANH 関数は、次の式で数学的に定義されます。ここで、–1 < *expression*< 1 です。この式では、*expression* は *x* で表されます。

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

例

次のプログラムは、ARTANH 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double a b ;
  method run();
    a=artanh(.5);
    b=artanh(-.5);
    put 'a=' a;
    put 'b=' b;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
a= 0.54930614433405
b= -0.54930614433405
```

関連項目

関数:

- “ARCOSH 関数” (310 ページ)
- “ARSINH 関数” (313 ページ)
- “COSH 関数” (469 ページ)
- “TANH 関数” (1186 ページ)
- “SINH 関数” (1139 ページ)

ATAN 関数

逆正接をラジアンで返します。

カテゴリ:	三角関数
別名:	ARTAN
返されるデータの種類:	DOUBLE

構文

ATAN(*expression*)

必須引数

expression
数値に評価される有効な式を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

ATAN 関数は、引数の 2 象限逆正接(逆タンジェント)を返します。戻り値は、正接が x の角度(ラジアン)です。値の範囲は– $\pi/2$から $\pi/2$までです。

比較

ATAN 関数は ATAN2 関数と類似していますが、ATAN2 は 1 つの引数ではなく 2 つの引数の比率から角度の逆正接を計算する点が異なります。

例

次のプログラムは、ATAN 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double a b c;
  method run();
    a=atan(0);
    b=atan(1);
    c=atan(-9);
    put 'a=' a;
    put 'b=' b;
    put 'c=' c;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
a= 0
b= 0.78539816339744
c= -1.460139105621
```

関連項目

関数:

- “ATAN2 関数” (317 ページ)

ATAN2 関数

直角三角形の *x* 座標と *y* 座標の逆正接をラジアンで返します。

カテゴリ: 三角関数

返されるデータの種類: DOUBLE

構文

ATAN2 (*expression-1*, *expression-2*)

必須引数

expression-1
数値に評価される有効な式を指定します。*expression-1* は、直角三角形の斜辺の端の x 座標を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

expression-2
数値に評価される有効な式を指定します。*expression-2* は、直角三角形の斜辺の端の y 座標を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

ATAN2 関数は、2 つの数値変数の逆正接(逆タンジェント)を返します。この関数の結果は、*expression-1*/*expression-2* の逆正接の計算結果と類似していますが、両方の引数の符号を使用して結果の象限を決定する点が異なります。

比較

ATAN2 関数は ATAN 関数と類似していますが、ATAN は 2 つの引数ではなく 1 つの引数の値から角度の逆正接を計算する点が異なります。

例

次のプログラムは、ATAN2 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double a b c;
  method run();
    a=atan2(-1, .5);
    b=atan2(6, 8);
    c=atan2(5, -3);
    put 'a=' a;
    put 'b=' b;
    put 'c=' c;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
a= -1.10714871779409
b= 0.64350110879328
c= 2.11121582706548
```

関連項目

- “DS2 による null および SAS 欠損値の処理方法” (*SAS DS2 プログラマガイド*)

関数:

- “ATAN 関数” (316 ページ)

BAND 関数

2 つの引数のビットごとの論理積を返します。

カテゴリ:	ビットごとの論理演算
返されるデータの種類:	DOUBLE
注:	2025.12 以降、この関数では、引数が DOUBLE の場合、すべての引数が DOUBLE に変換され、結果は DOUBLE になります。PROC DS2 ステートメントで BIGINTPROCESSING=YES が設定され、いずれかの引数が BIGINT の場合、すべての引数が BIGINT に変換され、結果は BIGINT になります。

構文

BAND(*expression-1* , *expression-2*)

必須引数

expression-1
数値に評価される有効な式を指定します。

範囲 0 から(2^{32})-1 まで(両端を含む)

データの種類 DOUBLE

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

expression-2
数値に評価される有効な式を指定します。

範囲 0 から(2^{32})-1 まで(両端を含む)

データの種類 DOUBLE

例

次のプログラムは、BAND 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double a b;
  method run();
    a=band(9, 11);
    b=band(15, 5);
    put 'a=' a;
    put 'b=' b;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
a= 9
b= 5
```

BETA 関数

beta 関数の値を返します。

カテゴリ: 数学

返されるデータの種類: DOUBLE

構文

BETA(*a*, *b*)

必須引数

a
第 1 形状パラメーターです。

範囲 *a* >0

データの種類 DOUBLE

b
第 2 形状パラメーターです。

範囲 *b* >0

データの種類 DOUBLE

詳細

BETA 関数は、次の式で数学的に求められます。

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

注:

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

前述の式で、$\Gamma(.)$はガンマ関数です。

式を計算できない場合、BETA は欠損値を返します。

例

次のプログラムは、BETA 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
```

```
data test (overwrite=yes);
  dcl double a b;
  method run();
    a=beta(5, 3);
    b=beta(15, 45);
    put 'a=' a;
    put 'b=' b;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
a= 0.0095238095238
b= -4.65396035015752
```

関連項目

関数:

- “LOGBETA 関数” (823 ページ)

BETAINV 関数

ベータ分布から分位点を返します。

カテゴリ:	分位点
返されるデータの種類:	DOUBLE

構文

BETAINV(*p*, *a*, *b*)

必須引数

p
数値の確率です。

範囲	$0 \le p \le 1$
データの種類	DOUBLE

a
数値の形状パラメーターです。

範囲 $a > 0$

データの種類 DOUBLE

b
数値の形状パラメーターです。

範囲 $b > 0$

データの種類 DOUBLE

詳細

BETAINV 関数は、ベータ分布(形状パラメーターは *a* および *b*)の *p* 分位点を返します。ベータ分布のオブザベーションが返される分位点以下になる確率は *p* です。

注: BETAINV は、PROBBETA 関数の逆数です。

例

次のプログラムは、BETAINV 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double y z;
  method run();
    y=betainv(0.001, 2, 4);
    put 'y=' y;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
y= 0.01010178788373
```

関連項目

関数:

- “PROBBETA 関数” (977 ページ)

BHAMMING_32 関数

2 つの 32 ビット整数値間のビットごとのハミング距離を返します。

カテゴリ: ビットごとの論理演算

返されるデータの種類: DOUBLE

注: 2025.12 以降、この関数では、引数が DOUBLE の場合、すべての引数が DOUBLE に変換され、結果は DOUBLE になります。PROC DS2 ステートメントで BIGINTPROCESSING=YES が設定され、いずれかの引数が BIGINT の場合、すべての引数が BIGINT に変換され、結果は BIGINT になります。

構文

BHAMMING_32(*argument-1*, *argument-2*)

必須引数

argument-1
数値の定数、変数または式を指定します。

範囲 整数値 0 から(2^{32})-1 まで(両端を含む)

argument-2
数値の定数、変数または式を指定します。

範囲 整数値 0 から(2^{32})-1 まで(両端を含む)

詳細

2 つのビット文字列間のビットごとのハミング距離は、一方の文字列のビットが他方の文字列の対応するビットと異なる文字列内の位置の数です。

いずれかの引数に欠損値が含まれている場合、関数は欠損値を返し、_ERROR_を 1 に設定します。それ以外の場合、各引数は内部で 32 ビット整数に変換されます。BHAMMING_32 は、これら 2 つの整数間のビットごとのハミング距離を計算し、その結果を整数値として返します。

例

この例は、BHAMMING_32 の機能を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double x y z;

  method run();
    x = 15;
    y = 124;
    z = BHAMMING_32(x,y);
    put x=binary8.;
    put y=binary8.;
    put z=;
  end;
enddata;
run;
quit;
```

ステートメントからの出力を次に示します。

```
x=00001111
y=01111100
z=5
```

BHAMMING_HEX 関数

2 つの 16 進数ビット文字列間のビットごとのハミング距離を返します。

カテゴリ: ビットごとの論理演算

返されるデータの種類: DOUBLE

注: 2025.12 以降、この関数では、引数が DOUBLE の場合、すべての引数が DOUBLE に変換され、結果は DOUBLE になります。PROC DS2 ステートメントで BIGINTPROCESSING=YES が設定され、いずれかの引数が BIGINT の場合、すべての引数が BIGINT に変換され、結果は BIGINT になります。

構文

BHAMMING_HEX(*hex-string-1*, *hex-string-2*);

必須引数

hex-string-1
16 進文字を含む文字列を指定します。先頭および末尾の空白は無視されます。

範囲　最大長は呼び出し環境によって異なります。

注　*hex-string-1* と *hex-string-2* は同じ長さでなければなりません。

hex-string-2
16 進文字を含む文字列を指定します。先頭および末尾の空白は無視されます。

範囲　最大長は呼び出し環境によって異なります。

注　*hex-string-1* と *hex-string-2* は同じ長さでなければなりません。

詳細

2 つのビット文字列間のビットごとのハミング距離は、一方の文字列のビットが他方の文字列の対応するビットと異なる文字列内の位置の数です。

いずれかの引数が空白の場合、関数は欠損値を返し、_ERROR_を 1 に設定します。それ以外の場合、16 進文字列は内部でビット文字列に変換されます。BHAMMING_HEX は、これら 2 つのビット文字列間のビットごとのハミング距離を計算し、その結果を整数値として返します。

例

この例は、BHAMMING_HEX の機能を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
dcl char(3) x y;
dcl double z;

method run();
  x='10F';
  y='17C';
  z = BHAMMING_HEX(x,y);
  put z=;
end;
enddata;
run;
quit;
```

これらのステートメントは、この結果を生成します。

```
z=5
```

BLACKCLPRC 関数

Black モデルに基づき、先物のヨーロピアンオプションのコール価格を計算します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

BLACKCLPRC(*E*, *t*, *F*, *r*, *sigma*)

必須引数

E
権利行使価格を指定する正の非欠損値。

要件 *E* および *F* は同じ単位で指定します。

データの種類 DOUBLE

t
満期までの時間を年単位で指定する非欠損値。

データの種類 DOUBLE

F
先物価格を指定する正の非欠損値。

要件 *F* および *E* は同じ単位で指定します。

データの種類 DOUBLE

r
連続複利の無リスク年金利を指定する正の非欠損値。

データの種類 DOUBLE

sigma
ボラティリティ (*r* の分散の平方根)を指定する正の非欠損分数。

データの種類 DOUBLE

詳細

BLACKCLPRC 関数は、Black モデルに基づき、先物のヨーロピアンオプションのコール価格を計算します。この関数は次の関係に基づきます。

$$\mathrm{CALL} = \varepsilon^{-rt}(FN(d_1) - EN(d_2))$$

引数

F
先物価格を指定します。

N
累積正規密度関数を指定します。

E
オプションの権利行使価格を指定します。

r
無リスク金利を指定します。これは、連続複利計算で表される年率です。

t
有効期限までの時間を年単位で指定します。

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

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

前述の式には次の引数が適用されます。

σ
原資産のボラティリティを 指定します。

σ^2
利益率の分散を指定します。

t=0 となる特別な場合には、次の式が真です。

$$\mathrm{CALL} = \max((F - E), 0)$$

価格設定の基本については、“価格関数の使用” (*SAS 関数と CALL ルーチン: リファレンス*)を参照してください。

比較

BLACKCLPRC 関数は、Black モデルに基づき、先物のヨーロピアンオプションのコール価格を計算します。BLACKPTPRC 関数は、Black モデルに基づき、先物のヨーロピアンオプションのプット価格を計算します。これらの関数はスカラー値を返します。

例

次のプログラムは、BLACKCLPRC 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double a b;
  method run();
    a=blackclprc(50, .25, 48, .05, .25);
    b=blackclprc(9, 1/12, 10, .05, .2);
    put 'a=' a;
    put 'b=' b;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
a= 1.55130142723116
b= 1
```

関連項目

関数:

- “BLACKPTPRC 関数” (329 ページ)

BLACKPTPRC 関数

Black モデルに基づき、先物のヨーロピアンオプションのプット価格を計算します。

カテゴリ:	財務
返されるデータの種類:	DOUBLE

構文

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

必須引数

E
権利行使価格を指定する正の非欠損値。

要件 *E* および *F* は同じ単位で指定します。

データの種類 DOUBLE

t
満期までの時間を年単位で指定する非欠損値。

データの種類 DOUBLE

F
先物価格を指定する正の非欠損値。

要件 *F* および *E* は同じ単位で指定します。

データの種類 DOUBLE

r
連続複利の無リスク年金利を指定する正の非欠損値。

データの種類 DOUBLE

sigma
ボラティリティ (*r* の分散の平方根)を指定する正の非欠損分数。

データの種類 DOUBLE

詳細

BLACKPTPRC 関数は、Black モデルに基づき、先物のヨーロピアンオプションのプット価格を計算します。この関数は次の関係に基づきます。

$$\mathrm{PUT} = \mathrm{CALL} + \varepsilon^{-rt}(E - F)$$

引数

E
オプションの権利行使価格を指定します。

r
無リスク金利を指定します。これは、連続複利計算で表される年率です。

t
有効期限までの時間を年単位で指定します。

F
先物価格を指定します。

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

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

前述の式には次の引数が適用されます。

σ
原資産のボラティリティを 指定します。

σ^2
利益率の分散を指定します。

t=0 となる特別な場合には、次の式が真です。

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

価格設定の基本については、*"価格関数の使用" (SAS 関数と CALL ルーチン: リファレンス)*を参照してください。

比較

BLACKPTPRC 関数は、Black モデルに基づき、先物のヨーロピアンオプションのプット価格を計算します。BLACKCLPRC 関数は、Black モデルに基づき、先物のヨーロピアンオプションのコール価格を計算します。これらの関数はスカラー値を返します。

例

次のプログラムは、BLACKPTPRC 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double a b;
  method run();
    a=blackptprc(298, .25, 350, .06, .25);
    b=blackptprc(145, .5, 170, .05, .2);
    put 'a=' a;
    put 'b=' b;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
a= 1.85980563934967
b= 1.41234979911583
```

関連項目

関数:

- “BLACKCLPRC 関数” (327 ページ)

BLKSHCLPRC 関数

Black-Scholes モデルに基づき、株式のヨーロピアンオプションのコール価格を計算します。

カテゴリ:	財務
返されるデータの種類:	DOUBLE

構文

BLKSHCLPRC(*E*, *t*, *S*, *r*, *sigma*)

必須引数

E
権利行使価格を指定する正の非欠損値。

要件 *E* および *S* は同じ単位で指定します。

データの種類 DOUBLE

t
満期までの時間を年単位で指定する非欠損値。

データの種類 DOUBLE

S
株価を指定する正の非欠損値。

要件 *S* および *E* は同じ単位で指定します。

データの種類 DOUBLE

r
連続複利の無リスク年金利を指定する正の非欠損値。

データの種類 DOUBLE

sigma
原資産のボラティリティを指定する正の非欠損分数。

データの種類 DOUBLE

詳細

BLKSHCLPRC 関数は、Black-Scholes モデルに基づき、株式のヨーロピアンオプションのコール価格を計算します。この関数は次の関係に基づきます。

$$\text{CALL} = SN(d_1) - EN(d_2)\varepsilon^{-rt}$$

引数

S
株価を指定する正の非欠損値。

N
累積正規密度関数を指定します。

E
オプションの権利行使価格を指定する正の非欠損値。

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

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

前述の式には次の引数が適用されます。

t
有効期限までの時間を年単位で指定します。

r
無リスク金利を指定します。これは、連続複利計算で表される年率です。

σ
ボラティリティ (分散の平方根)を指定します。

σ^2
利益率の分散を指定します。

t=0 となる特別な場合には、次の式が真です。

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

価格設定の基本については、“価格関数の使用” (*SAS 関数と CALL ルーチン: リファレンス*)を参照してください。

比較

BLKSHCLPRC 関数は、Black-Scholes モデルに基づき、株式のヨーロピアンオプションのコール価格を計算します。BLKSHPTPRC 関数は、Black-Scholes モデルに基づき、株式のヨーロピアンオプションのプットを計算します。これらの関数はスカラー値を返します。

例

次のプログラムは、BLKSHCLPRC 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double a b;
  method run();
    a=blkshclprc(50, .25, 48, .05, .25);
    b=blkshclprc(9, 1/12, 10, .05, .2);
    put 'a=' a;
    put 'b=' b;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
a= 1.79894201954462
b= 1
```

関連項目

関数:

- “BLKSHPTPRC 関数” (334 ページ)

BLKSHPTPRC 関数

Black-Scholes モデルに基づき、株式のヨーロピアンオプションのプット価格を計算します。

カテゴリ:	財務
返されるデータの種類:	DOUBLE

構文

BLKSHPTPRC(*E*, *t*, *S*, *r*, *sigma*)

必須引数

E
権利行使価格を指定する正の非欠損値。

要件 *E* および *S* は同じ単位で指定します。

データの種類 DOUBLE

t
満期までの時間を年単位で指定する非欠損値。

データの種類 DOUBLE

S
株価を指定する正の非欠損値。

要件 *S* および *E* は同じ単位で指定します。

データの種類 DOUBLE

r
連続複利の無リスク年金利を指定する正の非欠損値。

データの種類 DOUBLE

sigma
原資産のボラティリティを 指定する正の非欠損分数。

データの種類 DOUBLE

詳細

BLKSHPTPRC 関数は、Black-Scholes モデルに基づき、株式のヨーロピアンオプションのプットを計算します。この関数は次の関係に基づきます。

$$\mathrm{PUT} = \mathrm{CALL} - S + E\varepsilon^{-rt}$$

引数

S
株価を指定する正の非欠損値。

E
オプションの権利行使価格を指定する正の非欠損値。

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

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

前述の式には次の引数が適用されます。

t
有効期限までの時間を年単位で指定します。

r
無リスク金利を指定します。これは、連続複利計算で表される年率です。

σ
ボラティリティ (分散の平方根)を指定します。

σ^2
利益率の分散を指定します。

t=0 となる特別な場合には、次の式が真です。

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

価格設定の基本については、*"価格関数の使用" (SAS 関数と CALL ルーチン: リファレンス)*を参照してください。

比較

BLKSHPTPRC 関数は、Black-Scholes モデルに基づき、株式のヨーロピアンオプションのプットを計算します。BLKSHCLPRC 関数は、Black-Scholes モデルに基づき、株式のヨーロピアンオプションのコール価格を計算します。これらの関数はスカラー値を返します。

例

次のプログラムは、BLKSHPTPRC 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double a b;
  method run();
    a=blkshptprc(230,.5,290,.04,.25);
    b=blkshptprc(350,.3,400,.05,.2);
    put 'a=' a;
    put 'b=' b;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
a= 1.56597442946068
b= 1.64091943067597
```

関連項目

関数:

- “BLKSHCLPRC 関数” (332 ページ)

BLSHIFT 関数

2 つの引数を使ってビットを論理左シフトした値を返します。

カテゴリ: ビットごとの論理演算

返されるデータの種類: DOUBLE

注: 2025.12 以降、この関数では、引数が DOUBLE の場合、すべての引数が DOUBLE に変換され、結果は DOUBLE になります。PROC DS2 ステートメントで BIGINTPROCESSING=YES が設定され、いずれかの引数が BIGINT の場合、すべての引数が BIGINT に変換され、結果は BIGINT になります。

構文

BLSHIFT(*expression-1*, *expression-2*)

必須引数

expression-1
数値に評価される有効な式を指定します。

範囲 0 から(2^{32})-1 まで(両端を含む)

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

expression-2
数値に評価される有効な式を指定します。

範囲 0 から 31(0 と 31 を含む)

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

例

次のプログラムは、BLSHIFT 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double a;
  method run();
    a=blshift(7, 2);
    put a;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
28
```

関連項目

関数:

- “BRSHIFT 関数” (340 ページ)

BNOT 関数

引数のビットごとの論理否定を返します。

カテゴリ:	ビットごとの論理演算
返されるデータの種類:	DOUBLE
注:	2025.12 以降、この関数では、引数が DOUBLE の場合、すべての引数が DOUBLE に変換され、結果は DOUBLE になります。PROC DS2 ステートメントで BIGINTPROCESSING=YES が設定され、いずれかの引数が BIGINT の場合、すべての引数が BIGINT に変換され、結果は BIGINT になります。

構文

BNOT(*expression*)

必須引数

expression
数値に評価される有効な式を指定します。

範囲	0 から(2^{32})-1 まで(両端を含む)
データの種類	DOUBLE
参照項目:	“DS2 式” (*SAS DS2 プログラマガイド*)

例

次のプログラムは、BNOT 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double a;
  method run();
    a=bnot(16);
    put a;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
4294967279
```

BOR 関数

2 つの引数のビットごとの論理和を返します。

カテゴリ:	ビットごとの論理演算
返されるデータの種類:	DOUBLE
注:	2025.12 以降、この関数では、引数が DOUBLE の場合、すべての引数が DOUBLE に変換され、結果は DOUBLE になります。PROC DS2 ステートメントで BIGINTPROCESSING=YES が設定され、いずれかの引数が BIGINT の場合、すべての引数が BIGINT に変換され、結果は BIGINT になります。

構文

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

必須引数

expression-1, expression-2
数値に評価される有効な式を指定します。

範囲	0 から(2^{32})-1 まで(両端を含む)
データの種類	DOUBLE
参照項目:	“DS2 式” (*SAS DS2 プログラマガイド*)

例

次のプログラムは、BOR 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data one (overwrite=yes);
  dcl double x;
  method run();
    x=bor(4, 8);
    put x;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
12
```

BRSHIFT 関数

2 つの引数を使ってビットを論理右シフトした値を返します。

カテゴリ:	ビットごとの論理演算
返されるデータの種類:	DOUBLE
注:	2025.12 以降、この関数では、引数が DOUBLE の場合、すべての引数が DOUBLE に変換され、結果は DOUBLE になります。PROC DS2 ステートメントで BIGINTPROCESSING=YES が設定され、いずれかの引数が BIGINT の場合、すべての引数が BIGINT に変換され、結果は BIGINT になります。

構文

BRSHIFT(*expression-1*, *expression-2*)

必須引数

expression-1
数値に評価される有効な式を指定します。

範囲 0 から(2^{32})-1 まで(両端を含む)

データの種類 DOUBLE

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

expression-2
数値に評価される有効な式を指定します。

範囲 0 から 31(0 と 31 を含む)

データの種類 DOUBLE

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

例

次のプログラムは、BRSHIFT 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double a;
  method run();
    a=brshift(64, 2);
    put a;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
16
```

関連項目

関数:

- “BLSHIFT 関数” (337 ページ)

BXOR 関数

2 つの引数のビットごとの排他的論理和を返します。

カテゴリ: ビットごとの論理演算

返されるデータの種類: DOUBLE

注: 2025.12 以降、この関数では、引数が DOUBLE の場合、すべての引数が DOUBLE に変換され、結果は DOUBLE になります。PROC DS2 ステートメントで BIGINTPROCESSING=YES が設定され、いずれかの引数が BIGINT の場合、すべての引数が BIGINT に変換され、結果は BIGINT になります。

構文

BXOR(*expression-1, expression-2*)

必須引数

expression-1, expression-2
数値に評価される有効な式を指定します。

範囲 0 から(2^{32})-1 まで(両端を含む)

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

例

次のプログラムは、BXOR 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data one (overwrite=yes);
  dcl double x;
```

```
  method run();
    x=bxor(128, 64);
    put x;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
192
```

BYTE 関数

ASCII 照合順序で 1 文字を返します。

カテゴリ: 文字

返されるデータの種類: CHAR、NCHAR、NVARCHAR、VARCHAR

構文

BYTE(*n*)

必須引数

n
特定の ASCII 文字を表す整数を指定します。

範囲 0-255

データの種類 DOUBLE

詳細

ASCII 照合順序では、0 から 127 までの値に対応する文字が標準文字セットを表します。128 から 255 までの値に対応するその他の ASCII 文字は、特定の ASCII 動作環境で使用できますが、それらの文字が表す情報は動作環境によって異なります。

例

次のプログラムは、BYTE 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data one (overwrite=yes);
  dcl varchar x;
  method run();
    x=byte(80);
    put x $hex8.;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。16 進数の 50 に相当する ASCII は"P"です。

```
50
```

関連項目

関数:

- “RANK 関数” (1081 ページ)

CAT 関数

先頭または末尾の空白を削除せずに、連結文字列を返します。

カテゴリ:	文字
返されるデータの種類:	CHAR、NCHAR、NVARCHAR、VARCHAR

構文

CAT(*item-1*[, ...*item-n*])

必須引数

item
文字または数値の、定数、変数または式を指定します。*item* が数値の場合、その値は BEST*w*.出力形式を使用して文字列に変換されます。この場合、先頭の空白は削除され、ログにメモは記録されません。

詳細

返される変数の長さ

CAT 関数が、あらかじめ長さが割り当てられていない変数に値を返す場合、その変数には 200 バイトの長さが設定されます。| |や..連結演算子が、あらかじめ長さが割り当てられていない変数に値を返す場合、その変数には、連結される値の長さの合計となる長さが設定されます。

返される変数の長さ: 特殊な場合

CAT 関数は、変数または一時バッファーに値を返します。CAT 関数の戻り値の長さは次のとおりです。

- 最大 200 文字(WHERE 句および PROC SQL)
- PROC DS2 では、WHERE 句を除き、32767 文字まで
- 最大 65534 文字(CAT がマクロプロセッサから呼び出される場合)

CAT が値を一時バッファーに返す場合、バッファーの長さは呼び出し環境によって異なります。バッファーの値は CAT の処理後に切り捨てられる可能性があります。この場合、切り捨てに関するメッセージはログに出力されません。

変数またはバッファーの長さが不十分で連結結果を格納できない場合、SAS は次の手順を実行します。

- PROC DS2 と PROC SQL では、結果を空白行に変更します。
- 呼び出し環境に応じて、結果が切り捨てられたことを示す警告メッセージ、または空白値に設定されたことを示す警告メッセージがログに出力されます。
- 関数呼び出しの場所と、切り捨ての原因となった引数のリストを示すメモがログに出力されます。
- _ERROR_を 1 に設定します。

CAT 関数は、BEST*w*.出力形式で数値をフォーマットした後、数値引数から先頭と末尾の空白を削除します。

比較

CAT 関数、CATS 関数、CATT 関数、CATX 関数の結果は、*通常*、連結演算子| |および..、TRIM 関数、LEFT 関数の特定の組み合わせからの結果と同等になります。ただし、CAT 関数、CATS 関数、CATT 関数、CATX 関数のデフォルトの長さは、連結演算子を使用したときに得る長さとは異なります。詳細については、“返される変数の長さ” (345 ページ)を参照してください。

CAT 関数、CATS 関数、CATT 関数、CATX 関数を使用すると、TRIM および LEFT を使用するよりも高速になります。

例

次のプログラムでは、CAT 関数がどのように文字列を連結するのかを示します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl varchar(25) x y z a;
  dcl varchar(70) result;
  method init();
    x='  The 2012 Olym';
    y='pic Arts Festi';
    z='  val included works by D  ';
    a='ale Chihuly.';
    result=cat(x,y,z,a);
    put result=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
result=  The 2012 Olympic Arts Festi  val included works by D  ale Chihuly.
```

関連項目

関数:

- “CATQ 関数” (346 ページ)
- “CATS 関数” (351 ページ)
- “CATT 関数” (353 ページ)
- “LEFT 関数” (806 ページ)
- “STRIP 関数” (1166 ページ)

CATQ 関数

区切り文字を使用して各項目を区切り、区切り文字を含む文字列に引用符を追加して文字値または数値を連結します。

カテゴリ:	文字
返されるデータの種類:	CHAR、NCHAR、NVARCHAR、VARCHAR

構文

CATQ(*modifiers* [, *delimiter*], *item-1* [, ..., *item-n*])

必須引数

modifier
文字定数、変数または式を指定します。これらの空白以外の各文字で CATQ 関数のアクションを変更します。空白は無視されます。

次の文字を修飾子として使用できます。

1 or '
CATQ は、文字列に引用符を追加するときに単一引用符を使用します。

2 or "
CATQ は、文字列に引用符を追加するときに二重引用符を使用します。

a or A
すべての項目引数に引用符を追加します。

b or B
S または T 修飾子で削除されていない先頭や末尾の空白がある項目引数に引用符を追加します。

c or C
区切り文字としてカンマを使用します。

d or D
指定の区切り文字引数があることを示します。

h or H
区切り文字として水平タブを使用します。

m or M
最初の項目引数以降の各項目引数に区切り文字を挿入します。M 修飾子を使用しない場合、CATQ は他の修飾子で指定された処理の後に長さがゼロの項目引数の区切り文字を挿入しません。M 修飾子を使用すると、区切り文字が結果の最初または最後に表示される可能性があります。また、複数の連続する区切り文字が結果に表示される場合もあります。

n or N
値が SAS 名の通常の構文規則に準拠していない場合に項目引数を名前リテラルに変換します。名前リテラルは、引用符に囲まれた空白のない文字列で、後に文字"N"が続きます。SAS ステートメントで名前リテラルを使用するには、SAS オプション VALIDVARNAME=ANY を指定する必要があります。

q or Q
すでに引用符が含まれている項目引数に引用符を追加します。

s or S
後で処理される引数の先頭および末尾の空白を削除します。

- 区切り文字引数の先頭および末尾の空白を削除する場合、D 修飾子の*前に* S 修飾子を指定します。

- 項目引数の先頭および末尾の空白は削除するが、区切り文字引数の先頭および末尾の空白は削除*しない*場合、D 修飾子の*後に* S 修飾子を指定します。

t or T
後で処理される引数の末尾の空白を取り除きます。

- 区切り文字引数の末尾の空白を削除する場合、D 修飾子の前に T 修飾子を指定します。
- 項目引数の末尾の空白は削除するが、区切り文字引数の末尾の空白は削除しない場合、D 修飾子の後に T 修飾子を指定します。

x or X
値に印刷不可文字が含まれている場合に項目引数を 16 進リテラルに変換します。

ヒント *modifier* が定数の場合、引用符で囲みます。*modifier* を変数名または式として表すこともできます。

A、B、N、Q、S、T、X 修飾子は、CATQ 関数内部で動作します。項目引数が変数の場合、結果がその変数に割り当てられていない限り、その変数の値は CATQ によって変更されません。

item
文字または数値の、定数、変数または式を指定します。*item* が数値の場合、その値は BEST*w*.出力形式を使用して文字列に変換されます。この場合、先頭の空白は削除され、ログにメモは記録されません。

オプション引数

delimiter
連結文字列間の区切り文字として使用する文字定数、変数または式を指定します。この引数を指定する場合、D 修飾子も指定する必要があります。

詳細

返される変数の長さ

CATQ 関数は変数に値を返します。CATQ が式内で呼び出される場合は一時バッファーに値を返します。戻り値の長さは次のとおりです。

- 最大 200 文字(WHERE 句および PROC SQL)
- 最大 32767 文字(WHERE 句以外の DATA ステップ)
- 最大 65534 文字(CATQ がマクロプロセッサから呼び出される場合)

変数またはバッファーの長さが不十分で連結結果を格納できない場合、SAS は次の手順を実行します。DATA ステップおよび PROC SQL の結果を空白値に変更します。

- DATA ステップと PROC SQL では、結果を空白値に変更します。

- 呼び出し環境に応じて、結果が切り捨てられたことを示す警告メッセージ、または空白値に設定されたことを示す警告メッセージがログに出力されます。
- 関数呼び出しの場所と、切り捨ての原因となった引数のリストを示すメモがログに出力されます。
- DATA ステップで_ERROR_を 1 に設定します。

バッファーの値は CATQ の処理後に切り捨てられる可能性があります。この場合、切り捨てに関するメッセージはログに出力されません。

基本

C、D または H 修飾子を使用しない場合、CATQ は区切り文字として空白を使用します。

modifier に引用符、あるいは 1 または 2 修飾子のいずれも指定しないと、CATQ は引用符が必要な場合に各項目引数に使用する引用符の種類を独自に決定します。次の規則が適用されます。

- CATQ は、アンパサンド(&)やパーセント(%)記号を含む文字列、または単一引用符より二重引用符の方が多い文字列には、単一引用符を使用します。
- その他のすべての文字列には二重引用符を使用します。

CATQ 関数は、結果を初期化して長さをゼロにし、各項目引数で次のアクションを実行します。

1. *item* が文字列でない場合、CATQ は BEST*w* 形式を使用して *item* を文字列に変換し、 先頭の空白を削除します。
2. S 修飾子を使用している場合、CATQ は文字列から先頭の空白を削除します。
3. S または T 修飾子を使用すると、CATQ は文字列から末尾の空白を削除します。
4. CATQ は、次の条件に基づいて引用符を追加するかどうかを決定します。
   - X 修飾子を使用する場合、文字列に制御文字が含まれていると、文字列は 16 進リテラルに変換されます。
   - N 修飾子を使用する場合、次のいずれかの条件に該当すると、文字列は名前リテラルに変換されます。
     - 文字列の最初の文字がアンダースコアまたは英字ではない。
     - 文字列に数字、アンダースコアまたは英字以外の文字が含まれている。
   - X または N 修飾子を使用しない場合、次のいずれかの条件に該当すると、CATQ は引用符を文字列に追加します。
     - A 修飾子を使用している。
     - B 修飾子を使用していて、S または T 修飾子で削除されなかった先頭や末尾の空白が文字列に含まれている。
     - Q 修飾子を使用していて、文字列に引用符が含まれている。
     - 先頭および末尾の空白を削除した状態で区切り文字と同じになる部分文字列が文字列に含まれている。

5 2 番目以降の項目引数では、CATQ は次のいずれかの条件に該当する場合に区切り文字を結果に追加します。

- M 修飾子を使用している。
- 前述のステップで処理された後の文字列の長さがゼロよりも大きい。

6 CATQ は結果に文字列を追加します。

比較

CATX 関数は CATQ 関数と類似していますが、CATX は引用符を追加しない点が異なります。

例: CATQ 関数による文字列の連結

次のプログラムでは、CATQ 関数がどのように文字列を連結するのかを示します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data;
    dcl char(110) result1 result2 result3 result4 result5;
    method init();
      result1=CATQ(' ',
            'noblanks',
            'one blank',
            12345,
            '  lots of blanks    ');
      put result1=;
      result2=CATQ('CS',
            'Period (.)              ',
            'Ampersand (&)           ',
            'Comma (,)               ',
            'Double quotation marks (") ',
            '   Leading Blanks');
      put result2=;
      result3=CATQ('BCQT',
            'Period (.)              ',
            'Ampersand (&)           ',
            'Comma (,)               ',
            'Double quotation marks (") ',
            '   Leading Blanks');
      put result3=;
      result4=CATQ('ADT',
            '#=#',
            'Period (.)              ',
            'Ampersand (&)           ',
            'Comma (,)               ',
            'Double quotation marks (") ',
```

```
            '  Leading Blanks');
        put result4=;
        result5=CATQ('N',
              'ABC_123  ',
              '123   ',
              'ABC 123');
        put result5=;
      end;
  enddata;
  run;
  quit;
```

SAS は次の出力をログに書き込みます。

```
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
```

関連項目

関数:

- “CAT 関数” (344 ページ)
- “CATS 関数” (351 ページ)
- “CATT 関数” (353 ページ)
- “CATX 関数” (356 ページ)

CATS 関数

先頭と末尾の空白を削除して、連結文字列を返します。

カテゴリ:	文字
返されるデータの種類:	CHAR、NCHAR、NVARCHAR、VARCHAR

構文

CATS(*item*[, ...*item*])

必須引数

item
文字または数値の、定数、変数または式を指定します。*item* が数値の場合、その値は BEST*w*.出力形式を使用して文字列に変換されます。この場合、先頭の空白は削除され、ログにメモは記録されません。

詳細

返される変数の長さ

CATS 関数が、あらかじめ長さが割り当てられていない変数に値を返す場合、その変数には 200 バイトの長さが設定されます。| |や..連結演算子が、あらかじめ長さが割り当てられていない変数に値を返す場合、その変数には、連結される値の長さの合計となる長さが設定されます。

返される変数の長さ: 特殊な場合

CATS 関数は、変数または一時バッファーに値を返します。CATS 関数の戻り値の長さは次のとおりです。

- 最大 200 文字(WHERE 句および PROC SQL)
- PROC DS2 では、WHERE 句を除き、32767 文字まで
- 最大 65534 文字(CATS がマクロプロセッサから呼び出される場合)

CATS が値を一時バッファーに返す場合、バッファーの長さは呼び出し環境によって異なります。バッファーの値は CATS の処理後に切り捨てられる可能性があります。この場合、切り捨てに関するメッセージはログに出力されません。

変数またはバッファーの長さが不十分で連結結果を格納できない場合、SAS は次の手順を実行します。

- PROC DS2 と PROC SQL では、結果を空白値に変更します。
- 呼び出し環境に応じて、結果が切り捨てられたことを示す警告メッセージ、または空白値に設定されたことを示す警告メッセージがログに出力されます。
- 関数呼び出しの場所と、切り捨ての原因となった引数のリストを示すメモがログに出力されます。
- _ERROR_を 1 に設定します。

CATS 関数は、BEST*w*.出力形式で数値をフォーマットした後、数値引数から先頭と末尾の空白を削除します。

比較

CAT 関数、CATS 関数、CATT 関数、CATX 関数の結果は、*通常*、連結演算子| |および..、TRIM 関数、LEFT 関数の特定の組み合わせからの結果と同等になります。ただし、CAT 関数、CATS 関数、CATT 関数、CATX 関数のデフォルトの長さは、連結演

算子を使用したときに得る長さとは異なります。詳細については、“返される変数の長さ” (352 ページ)を参照してください。

CAT 関数、CATS 関数、CATT 関数、CATX 関数を使用すると、TRIM および LEFT を使用するよりも高速になります。

例

次のプログラムでは、CATS 関数がどのように文字列を連結するのかを示します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl char(25) x y z a;
  dcl char(70) result;
  method init();
    x='  The   Olym';
    y='pic Arts Festi';
    z='  val includes works by D  ';
    a='ale Chihuly.';
    result=cats(x,y,z,a);
    put result=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
result=The   Olympic Arts Festival includes works by Dale Chihuly.
```

関連項目

関数:

- “CAT 関数” (344 ページ)
- “CATT 関数” (353 ページ)
- “CATX 関数” (356 ページ)
- “STRIP 関数” (1166 ページ)

CATT 関数

末尾の空白を削除して、連結文字列を返します。

カテゴリ:	文字
返されるデータの種類:	CHAR、NCHAR、NVARCHAR、VARCHAR

構文

CATT(*item*[, ...*item*])

必須引数

item
文字または数値の、定数、変数または式を指定します。*item* が数値の場合、その値は BEST*w*.出力形式を使用して文字列に変換されます。この場合、先頭の空白は削除され、ログにメモは記録されません。

詳細

返される変数の長さ

CATT 関数が、あらかじめ長さが割り当てられていない変数に値を返す場合、その変数には 200 バイトの長さが設定されます。| |や..連結演算子が、あらかじめ長さが割り当てられていない変数に値を返す場合、その変数には、連結される値の長さの合計となる長さが設定されます。

返される変数の長さ: 特殊な場合

CATT 関数は、変数または一時バッファーに値を返します。CATT 関数の戻り値の長さは次のとおりです。

- 最大 200 文字(WHERE 句および PROC SQL)
- PROC DS2 では、WHERE 句を除き、32767 文字まで
- 最大 65534 文字(CATT がマクロプロセッサから呼び出される場合)

CATT が値を一時バッファーに返す場合、バッファーの長さは呼び出し環境によって異なります。バッファーの値は CATT の処理後に切り捨てられる可能性があります。この場合、切り捨てに関するメッセージはログに出力されません。

変数またはバッファーの長さが不十分で連結結果を格納できない場合、SAS は次の手順を実行します。

- PROC DS2 と PROC SQL では、結果を空白値に変更します。
- 呼び出し環境に応じて、結果が切り捨てられたことを示す警告メッセージ、または空白値に設定されたことを示す警告メッセージがログに出力されます。
- 関数呼び出しの場所と、切り捨ての原因となった引数のリストを示すメモがログに出力されます。
- _ERROR_を 1 に設定します。

CATT 関数は、BEST*w*.出力形式で数値をフォーマットした後、数値引数から先頭と末尾の空白を削除します。

比較

CAT 関数、CATS 関数、CATT 関数、CATX 関数の結果は、*通常*、連結演算子| |および..、TRIM 関数、LEFT 関数の特定の組み合わせからの結果と同等になります。ただし、CAT 関数、CATS 関数、CATT 関数、CATX 関数のデフォルトの長さは、連結演算子を使用したときに得る長さとは異なります。詳細については、“返される変数の長さ” (354 ページ)を参照してください。

CAT 関数、CATS 関数、CATT 関数、CATX 関数を使用すると、TRIM および LEFT を使用するよりも高速になります。

例

次のプログラムでは、CATT 関数がどのように文字列を連結するのかを示します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl char(25) x y z a;
  dcl char(70) result;
  method init();
    x='  The 2012 Olym';
    y='pic Arts Festi';
    z='  val included works by D  ';
    a='ale Chihuly.';
    result=catt(x,y,z,a);
    put result=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
result=  The 2012 Olympic Arts Festi  val included works by Dale Chihuly.
```

関連項目

関数:

- “CAT 関数” (344 ページ)
- “CATQ 関数” (346 ページ)

- “CATS 関数” (351 ページ)
- “CATX 関数” (356 ページ)
- “STRIP 関数” (1166 ページ)

CATX 関数

先頭と末尾の空白を削除し、区切り文字を挿入して、連結文字列を返します。

カテゴリ:	文字
返されるデータの種類:	CHAR、NCHAR、NVARCHAR、VARCHAR

構文

CATX(*delimiter*, *item-1*[, ...*item-n*])

必須引数

delimiter
連結項目間の区切り文字として使用する文字列を指定します。

item
文字または数値の、定数、変数または式を指定します。*item* が数値の場合、その値は BEST*w*.出力形式を使用して文字列に変換されます。この場合、ログにメモは記録されません。詳細については、“基本” (356 ページ)を参照してください。

詳細

基本

CATX 関数は、まず *item-1* を先頭および末尾の空白を削除して結果にコピーします。次に、CATX は後続の各引数 *item-i* (i=2, ..., n)で、*item-i* に 1 つ以上の空白以外の文字が含まれている場合に *delimiter* および *item-i* を結果に追加します。このとき、*item-i* から先頭および末尾の空白を削除します。CATX は、結果の先頭または末尾に区切り文字を挿入しません。空白の項目では、結果の先頭または末尾に区切り文字は生成されません。また、複数の連続する区切り文字も生成されません。

返される変数の長さ

CATX 関数は、変数または一時バッファーに値を返します。CATX 関数から返される値は、WHERE 句を除いて、最大 32767 文字まで可能です。

変数またはバッファーの長さが不十分で連結結果を格納できない場合、SAS は結果を切り捨てます。

比較

CAT 関数、CATS 関数、CATT 関数、CATX 関数の結果は、*通常*、連結演算子| |および..、TRIM 関数、LEFT 関数の特定の組み合わせからの結果と同等になります。ただし、CAT、CATS、CATT、CATX 関数のデフォルトの長さは、連結演算子の使用時に取得する長さとは異なります。詳細については、“返される変数の長さ” (356 ページ)を参照してください。

CAT 関数、CATS 関数、CATT 関数、CATX 関数を使用すると、TRIM および LEFT を使用するよりも高速になります。

注: 欠損値のある変数の場合、連結で異なる結果が生成されます。

例

次のプログラムでは、CATX 関数がどのように文字列を連結するのかを示します。最初のデータプログラムでは、値テーブルを作成します。2 番目と 3 番目のデータプログラムでは、値テーブルを入力として使用します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
/* This data program creates the Values table. */
proc ds2;
data values;
  dcl char(4) x1 x2 x3 x4;
  method init();
      /* simple values */
    x1='A'; x2='B'; x3='C'; x4='D';
    output;
    x1='XX'; x2='YY'; x3='ZZ'; x4='WW';
    output;

      /* values with leading, trailing, and embedded white space */
    x1='XX'; x2='Y Y '; x3=' Z Z'; x4=' WW ';
    output;

      /* CHAR set to missing */
    x1='E'; x2= . ; x3='F'; x4='G';
    output;
    x1='H'; x2= . ; x3= . ; x4='J';
    output;

      /* CHAR set to zero-length strings */
    x1='X'; x2='' ; x3='' ; x4='W';
    output;

      /* CHAR set to the null value */
    x1='X'; x2=null; x3='Z' ; x4=null;
    output;
```

```
  end;
run;

  /* This data program creates the Concat1 table. */
data concat1;
  dcl char(1) sp;
  dcl char(4) x1 x2 x3 x4;
  dcl char(20) test1 test2 spacey;
  method run();
   set values;
   SP='^';
   test1 = catx(sp, x1, x2, x3, x4);
   test2 = strip(x1)
         || sp || strip(x2)
         || sp || strip(x3)
         || sp || strip(x4);
   spacey = x1 || sp || x2 || sp || x3 || sp || x4;
  end;
run;

  /* This data program creates the Concat2 table.          */
  /* The example shows what happens when the delimiter contains */
  /* space characters.                                     */
data concat2;
  dcl char(3) sp;
  dcl char(4) x1 x2 x3 x4;
  dcl char(20) test1 test2 spacey;
  method run();
    set values;
    SP = ' ^ ';
    test1 = catx(sp, x1, x2, x3, x4);
    test2 = strip(x1)
          || strip(sp) || strip(x2)
          || strip(sp) || strip(x3)
          || strip(sp) || strip(x4);
    spacey = strip(x1)
          || sp || strip(x2)
          || sp || strip(x3)
          || sp || strip(x4);
  end;
run;
quit;

proc print data=concat1;
  title 'The Concat1 table';
run;

proc print data=concat2;
  title 'The Concat2 table';
run;
```

アウトプット 7.1　連結文字を示すテーブル

The Concat1 table

Obs	sp	x1	x2	x3	x4	test1	test2	spacey
1	^	A	B	C	D	A^B^C^D	A^B^C^D	A ^B ^C ^D
2	^	XX	YY	ZZ	WW	XX^YY^ZZ^WW	XX^YY^ZZ^WW	XX ^YY ^ZZ ^WW
3	^	XX	Y Y	Z Z	WW	XX^Y Y^Z Z^WW	XX^Y Y^Z Z^WW	XX ^Y Y ^ Z Z^ WW
4	^	E	.	F	G	E^.^F^G	E^.^F^G	E ^. ^F ^G
5	^	H	.	.	J	H^.^.^J	H^.^.^J	H ^. ^. ^J
6	^	X			W	X^W	X^^^W	X ^ ^ ^W
7	^	X		Z		X^Z	X^^Z^	X ^ ^Z ^

アウトプット 7.2　スペースを 含む連結文字を示すテーブル

The Concat2 table

Obs	sp	x1	x2	x3	x4	test1	test2	spacey
1	^	A	B	C	D	A ^ B ^ C ^ D	A^B^C^D	A ^ B ^ C ^ D
2	^	XX	YY	ZZ	WW	XX ^ YY ^ ZZ ^ WW	XX^YY^ZZ^WW	XX ^ YY ^ ZZ ^ WW
3	^	XX	Y Y	Z Z	WW	XX ^ Y Y ^ Z Z ^ WW	XX^Y Y^Z Z^WW	XX ^ Y Y ^ Z Z ^ WW
4	^	E	.	F	G	E ^ . ^ F ^ G	E^.^F^G	E ^ . ^ F ^ G
5	^	H	.	.	J	H ^ . ^ . ^ J	H^.^.^J	H ^ . ^ . ^ J
6	^	X			W	X ^ W	X^^^W	X ^ ^ ^ W
7	^	X		Z		X ^ Z	X^^Z^	X ^ ^ Z ^

関連項目

関数:

- “CAT 関数” (344 ページ)
- “CATQ 関数” (346 ページ)
- “CATS 関数” (351 ページ)
- “CATT 関数” (353 ページ)
- “STRIP 関数” (1166 ページ)

CDF 関数

さまざまな連続確率分布および離散確率分布の左累積分布関数を計算します。

カテゴリ: 確率

注: QUANTILE 関数は、指定した分布から分位点を返します。QUANTILE 関数は、CDF 関数の逆数です。詳細については、“QUANTILE 関数” (1052 ページ)を参照してください。この関数の入力として無効な数値が指定された場合、エラーメッセージはログに書き込まれず、欠損値が出力として返されます。

構文

CDF(*'distribution'*, *quantile* [, *parameter-1, ..., parameter-k*])

必須引数

distribution
分布を特定する文字定数、変数または式です。

有効な分布は次のとおりです。

分布	引数
Bernoulli	'BERNOULLI'
ベータ	'BETA'
二項	'BINOMIAL'
Cauchy	'CAUCHY'
カイ 2 乗	'CHISQUARE'
Conway-Maxwell-Poisson	'CONMAXPOI'
指数	'EXPONENTIAL'
F	'F'
ガンマ	'GAMMA'
一般化 Poisson	'GENPOISSON'
幾何	'GEOMETRIC'
超幾何	'HYPERGEOMETRIC'
Laplace	'LAPLACE'
ロジスティック	'LOGISTIC'

分布	引数
対数正規	'LOGNORMAL'
負の二項	'NEGBINOMIAL'
正規	'NORMAL'|'GAUSS'
正規混合	'NORMALMIX'
パレート	'PARETO'
Poisson	'POISSON'
T	'T'
Tweedie	'TWEEDIE'
一様	'UNIFORM'
Wald (逆ガウス)	'WALD'|'IGAUSS'
Weibull	'WEIBULL'

注 T、F および NORMALMIX を除き、最初の 4 文字で分布を最小限に識別できます。

quantile
確率変数の値を指定する数値定数、変数または式です。

データの種類 DOUBLE

オプション引数

parameter-1, ... , parameter-k
特定の分布に適した*形状*、*位置*または*尺度*パラメーターの値を指定する定数、変数または式です(省略可能)。

データの種類 DOUBLE

関連項目

関数:

- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)

- “LOGSDF 関数” (831 ページ)
- “PDF 関数” (918 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

CDF BERNOULLI 分布 関数

Bernoulli 累積確率分布の値を返します。

カテゴリ: 確率

返されるデータの種類: DOUBLE

注: QUANTILE 関数は、指定した分布から分位点を返します。QUANTILE 関数は、CDF 関数の逆数です。詳細については、“QUANTILE 関数” (1052 ページ)を参照してください。

構文

CDF('BERNOULLI', *x*, *p*)

必須引数

x
確率変数を指定する数値定数、変数または式です。

データの種類 DOUBLE

p
成功確率を指定する数値定数、変数または式です。

範囲 $0 \le p \le 1$

データの種類 DOUBLE

詳細

Bernoulli 分布の CDF 関数は、Bernoulli 分布(成功確率は *p*)からオブザベーションが *x* 以下となる確率を返します。

$$CDF('BERN', x, p) = \begin{cases} 0 & x < 0 \\ 1 - p & 0 \le x < 1 \\ 1 & x \ge 1 \end{cases}$$

注: この分布には、*位置*または*尺度*パラメーターはありません。

例

次のプログラムは、CDF Bernoulli 分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double y;
  method run();
    y=cdf('bern', 0, .25);
    put  y=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
y=0.75
```

関連項目

関数:

- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “PDF BERNOULLI 分布 関数” (920 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

CDF BETA 分布 関数

ベータ累積確率分布の値を返します。

カテゴリ: 確率

返されるデータの種類: DOUBLE

注: QUANTILE 関数は、指定した分布から分位点を返します。QUANTILE 関数は、CDF 関数の逆数です。詳細については、“QUANTILE 関数” (1052 ページ)を参照してください。

構文

CDF('BETA', *x*, *a*, *b* [, *l*, *r*])

必須引数

x
確率変数を指定する数値定数、変数または式です。

データの種類 DOUBLE

a
形状パラメーターを指定する数値定数、変数または式です。

範囲 $a > 0$

データの種類 DOUBLE

b
形状パラメーターを指定する数値定数、変数または式です。

範囲 $b > 0$

データの種類 DOUBLE

オプション引数

l
左位置パラメーターを指定する数値定数、変数または式です。

デフォルト 0

データの種類 DOUBLE

r
右位置パラメーターを指定する数値定数、変数または式です。

デフォルト 1

範囲 $r > l$

データの種類 DOUBLE

詳細

ベータ分布の CDF 関数は、ベータ分布(形状パラメーターは *a* および *b*)からオブザベーションが *v* 以下となる確率を返します。次の式は、ベータ分布の CDF 関数を表しています。

$$CDF('BETA', x, a, b, l, r) = \begin{cases} 0 & x \le l \\ \frac{1}{\beta(a,b)} \int_{l}^{x} \frac{(v-l)^{a-1}(r-v)^{b-1}}{(r-l)^{a+b-1}} dv & l < x \le r \\ 1 & x > r \end{cases}$$

前述の式には次の関係が適用されます。

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

前述の式には次の関係が適用されます。

$$\Gamma(a) = \int_{0}^{\infty} x^{a-1} e^{-x} dx$$

例

次のプログラムは、CDF ベータ分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double y;
  method run();
    y=cdf('beta', 0.2,3,4);
    put  y=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
y=0.09888
```

関連項目

関数:

- “LOGCDF 関数” (825 ページ)

- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “PDF BETA 分布 関数” (922 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

CDF BINOMIAL 分布 関数

二項累積確率分布の値を返します。

カテゴリ: 確率

返されるデータの種類: DOUBLE

注: QUANTILE 関数は、指定した分布から分位点を返します。QUANTILE 関数は、CDF 関数の逆数です。詳細については、“QUANTILE 関数” (1052 ページ)を参照してください。

構文

CDF('BINOMIAL', *m*, *p*, *n*)

必須引数

m
成功数を数える整数の確率変数です。

範囲 $m = 0, 1, \ldots$

データの種類 DOUBLE

p
成功確率パラメーターを指定する数値定数、変数または式です。

範囲 $0 \le p \le 1$

データの種類 DOUBLE

n
独立 Bernoulli 試行数を数える整数パラメーターを指定する数値定数、変数または式です。

範囲 $n = 0, 1, \ldots$

データの種類 DOUBLE

詳細

二項分布の CDF 関数は、二項分布(パラメーターは *p* および *n*)からオブザベーションが *m* 以下となる確率を返します。

$$CDF('BINOM', m, p, n) = \begin{cases} 0 & m < 0 \\ \sum_{j=0}^{m} \binom{n}{j} p^j (1-p)^{n-j} & 0 \le m \le n \\ 1 & m > n \end{cases}$$

注: 二項分布には、*位置*または*尺度*パラメーターはありません。

例

次のプログラムは、CDF 二項分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double y;
  method run();
    y=cdf('BINOM', 4, .5, 10);
    put  y=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
y=0.376953125
```

関連項目

関数:

- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “PDF BINOMIAL 分布 関数” (924 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)

- “SQUANTILE 関数” (1149 ページ)

CDF CAUCHY 分布 関数

Cauchy 累積確率分布の値を返します。

カテゴリ:	確率
別名:	PMF
返されるデータの種類:	DOUBLE
注:	QUANTILE 関数は、指定した分布から分位点を返します。QUANTILE 関数は、CDF 関数の逆数です。詳細については、“QUANTILE 関数” (1052 ページ)を参照してください。

構文

CDF('CAUCHY', *x* [,*θ*] [, *λ*])

必須引数

x
確率変数を指定する数値定数、変数または式です。

データの種類 DOUBLE

オプション引数

θ
位置パラメーターを指定する数値定数、変数または式です。

デフォルト 0

データの種類 DOUBLE

λ
尺度パラメーターを指定する数値定数、変数または式です。

デフォルト 1

範囲 λ > 0

データの種類 DOUBLE

詳細

Cauchy 分布の CDF 関数は、Cauchy 分布(位置パラメーターは θ、尺度パラメーターは λ)からオブザベーションが *x* 以下となる確率を返します。

$$CDF('CAUCHY', x, \theta, \lambda) = \frac{1}{2} + \frac{1}{\pi} tan^{-1}\left(\frac{x-\theta}{\lambda}\right)$$

例

次のプログラムは、CDF Cauchy 分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double y;
  method run();
    y=cdf('CAUCHY', 2);
    put y=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
y=0.85241638234956
```

関連項目

関数:

- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “PDF CAUCHY 分布 関数” (926 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

CDF カイ 2 乗分布 関数

カイ 2 乗累積確率分布の値を返します。

カテゴリ: 確率

返されるデータの種類: DOUBLE

注: QUANTILE 関数は、指定した分布から分位点を返します。QUANTILE 関数は、CDF 関数の逆数です。詳細については、“QUANTILE 関数” (1052 ページ)を参照してください。

構文

CDF('CHISQUARE', *x*, *df* [, *nc*])

必須引数

x
確率変数を指定する数値定数、変数または式です。

データの種類 DOUBLE

df
自由度パラメーターを指定する数値定数、変数または式です。

範囲 $df > 0$

データの種類 DOUBLE

オプション引数

nc
省略可能な非心度パラメーターを指定する数値定数、変数または式です。

範囲 $nc \geq 0$

データの種類 DOUBLE

詳細

カイ 2 乗分布の CDF 関数は、カイ 2 乗分布(自由度は *df*、非心度パラメーターは *nc*)からオブザベーションが *x* 以下となる確率を返します。この関数では、整数以外の自由度を使用できます。*nc* が省略されているかゼロの場合、心度カイ 2 乗分布の値

が返されます。次の式で、$\nu = df$として、そして$\lambda = nc$とします。次の式は、カイ 2 乗分布の CDF 関数を表しています。

$$CDF('CHISQ', x, \nu, \lambda) = \begin{cases} 0 & x < 0 \\ \sum_{j=0}^{\infty} e^{-\frac{\lambda}{2}} \frac{\left(\frac{\lambda}{2}\right)^j}{j!} P_c(x, \nu + 2j) & x \geq 0 \end{cases}$$

この式では、$P_c(.,.)$は、心度カイ 2 乗分布からの確率を示しています。

$$P_c(x, a) = P_g\left(\frac{x}{2}, \frac{a}{2}\right)$$

この式では、$P_g(y,b)$は、次の式によって得られるガンマ分布からの確率です。

$$P_g(y, b) = \frac{1}{\Gamma(b)} \int_0^y e^{-v} v^{b-1} dv$$

例

次のプログラムは、CDF カイ 2 乗分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double y;
  method run();
    y=cdf('CHISQ', 11.264, 11);
    put y=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
y=0.5785813293173
```

関連項目

関数:

- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “PDF カイ 2 乗分布 関数” (928 ページ)

- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

CDF Conway-Maxwell-Poisson 分布 関数

Conway-Maxwell-Poisson 累積確率分布の値を返します。

カテゴリ: 確率

返されるデータの種類: DOUBLE

注: QUANTILE 関数は、指定した分布から分位点を返します。QUANTILE 関数は、CDF 関数の逆数です。詳細については、“QUANTILE 関数” (1052 ページ)を参照してください。

構文

CDF('CONMAXPOI', *y*, *λ*,*ν*)

必須引数

y
カウントデータを表す負でない整数を指定する数値定数、変数または式です。

データの種類 DOUBLE

λ
Poisson 分布の場合と同様で、平均に似ています。

データの種類 DOUBLE

ν
ばらつきのパラメーターを指定する数値定数、変数または式です。

データの種類 DOUBLE

詳細

CDF 関数は、0 から y までの累積確率を返します。詳細については、PDF 関数の“Conway-Maxwell-Poisson”分布 (930 ページ)を参照してください。

例

次のプログラムは、CDF Conway-Maxwell-Poisson 分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double x y;
  method init();
    y=cdf('CONMAXPOI', 5, 2.3, .4);
    put y=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
y=0.2445411535065
```

関連項目

関数:

- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “PDF Conway-Maxwell-Poisson 分布 関数” (930 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

CDF 指数分布 関数

指数累積確率分布の値を返します。

カテゴリ:	確率
返されるデータの種類:	DOUBLE

注: QUANTILE 関数は、指定した分布から分位点を返します。QUANTILE 関数は、CDF 関数の逆数です。詳細については、“QUANTILE 関数” (1052 ページ)を参照してください。

構文

CDF('EXPONENTIAL', *x* [,*λ*])

必須引数

x
確率変数を指定する数値定数、変数または式です。

データの種類 DOUBLE

オプション引数

λ
尺度パラメーターを指定する数値定数、変数または式です。

デフォルト 1

範囲 λ > 0

データの種類 DOUBLE

詳細

指数分布の CDF 関数は、指数分布(尺度パラメーターは λ)からオブザベーションが *x* 以下となる確率を返します。

$$CDF('EXPO', x, \lambda) = \begin{cases} 0 & x < 0 \\ 1 - e^{-\frac{x}{\lambda}} & x \geq 0 \end{cases}$$

例

次のプログラムは、CDF 指数分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double y;
  method run();
```

```
      y=cdf('expo', 1);
      put y=;
    end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
y=0.63212055882855
```

関連項目

関数:

- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “PDF EXPONENTIAL 分布 関数” (932 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

CDF F 分布 関数

F 累積確率分布の値を返します。

カテゴリ:	確率
返されるデータの種類:	DOUBLE
注:	QUANTILE 関数は、指定した分布から分位点を返します。QUANTILE 関数は、CDF 関数の逆数です。詳細については、“CDF F 分布 関数” (375 ページ)を参照してください。

構文

CDF('F', *x*, *ndf*, *ddf* [, *nc*])

必須引数

x
確率変数を指定する数値定数、変数または式です。

ndf
分子の自由度パラメーターを指定する数値定数、変数または式です。

範囲 *ndf* > 0

データの種類 DOUBLE

ddf
分母の自由度パラメーターを指定する数値定数、変数または式です。

範囲 *ddf* > 0

データの種類 DOUBLE

オプション引数

nc
非心度パラメーターを指定する数値定数、変数または式です。

範囲 *nc* ≥ 0

データの種類 DOUBLE

詳細

F 分布の CDF 関数は、*F* 分布(分子の自由度は *ndf*、分母の自由度は *ddf*、非心度パラメーターは *nc*)からオブザベーションが *x* 以下となる確率を返します。この関数では、*ndf* および *ddf* に対して整数以外の自由度を使用できます。*nc* が除外されているかゼロの場合、心度 *F* 分布の値が返されます。次の式で、$\nu_1 = ndf$として、$\nu_2 = ddf$として、そして$\lambda = nc$とします。次の式は、*F* 分布の CDF 関数を表しています。

$$CDF('F', x, \nu_1, \nu_2, \lambda) = \begin{cases} 0 & x < 0 \\ \sum_{j=0}^{\infty} e^{-\frac{\lambda}{2}} \frac{\left(\frac{\lambda}{2}\right)^j}{j!} P_F(x, \nu_1 + 2j, \nu_2) & x \geq 0 \end{cases}$$

この式では、$P_f(f,u_1,u_2)$は、次の式を使用した心度 *F* 分布からの確率です。

$$P_F(x, u_1, u_2) = P_B\left(\frac{u_1 x}{u_1 x + u_2}, \frac{u_1}{2}, \frac{u_2}{2}\right)$$

また、$P_B(x,a,b)$は、標準ベータ分布からの確率です。

注: *F* 分布には、*位置*または*尺度*パラメーターはありません。

例

次のプログラムは、CDF F 分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double y;
  method run();
    y=cdf('F', 3.32, 2, 3);
    put y=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
y=0.82639336022431
```

関連項目

関数:

- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “PDF F 分布 関数” (934 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

CDF GAMMA 分布 関数

ガンマ累積確率分布の値を返します。

カテゴリ:	確率
返されるデータの種類:	DOUBLE

注: QUANTILE 関数は、指定した分布から分位点を返します。QUANTILE 関数は、CDF 関数の逆数です。詳細については、“QUANTILE 関数” (1052 ページ)を参照してください。

構文

CDF('GAMMA', *x*, *α* [, *λ*])

必須引数

x
確率変数を指定する数値定数、変数または式です。

データの種類 DOUBLE

α
形状パラメーターを指定する数値定数、変数または式です。

範囲 $\alpha > 0$

データの種類 DOUBLE

オプション引数

λ
尺度パラメーターを指定する数値定数、変数または式です。

デフォルト 1

範囲 $\lambda > 0$

データの種類 DOUBLE

詳細

ガンマ分布の CDF 関数は、ガンマ分布(形状パラメーターは *α*、尺度パラメーターは λ)からオブザベーションが *x* 以下となる確率を返します。

$$CDF('GAMMA', x, a, \lambda) = \begin{cases} 0 & x < 0 \\ \dfrac{1}{\lambda^a \Gamma(a)} \displaystyle\int_0^x v^{a-1} e^{-\frac{v}{\lambda}} dv & x \geq 0 \end{cases}$$

例

次のプログラムは、CDF ガンマ分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double y;
  method init();
    y=cdf('gamma',1,3);
    put y=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
y=0.08030139707139
```

関連項目

関数:

- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “PDF GAMMA 分布 関数” (936 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

CDF 一般化 Poisson 分布 関数

一般化 Poisson 累積確率分布の値を返します。

カテゴリ:	確率
返されるデータの種類:	DOUBLE
注:	QUANTILE 関数は、指定した分布から分位点を返します。QUANTILE 関数は、CDF 関数の逆数です。詳細については、“QUANTILE 関数” (1052 ページ)を参照してください。

構文

CDF('GENPOISSON', *x*, *θ*, *η*)

必須引数

x
確率変数を指定する数値定数、変数または式です。この引数は整数にする必要があります。

データの種類 DOUBLE

θ
形状パラメーターを指定する数値定数、変数または式です。

範囲 ≤5 かつ>0

データの種類 DOUBLE

η
形状パラメーターを指定する数値定数、変数または式です。

範囲 ≥0 かつ<0.95

データの種類 DOUBLE

ヒント η =0 の場合、平均と分散が θ の Poisson 分布になります。η>0 の場合、平均は$\theta \div (1-\eta)$および分散は$\theta \div (1-\eta)^3$になります。

詳細

一般化 Poisson 分布の確率質量関数の次のとおりです。

$$f(x;\theta,\eta) = \theta(\theta+\eta x)^{x-1} e^{-\theta-\eta x}/x!, \quad x = 0, 1, 2, ..., \qquad \theta > 0,\ 0 \le \eta < 1$$

η = 0 の場合、一般化 Poisson 分布は、形状パラメーターが θ の標準 Poisson 分布になります。

例

次のプログラムは、CDF 一般化 Poisson 分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double y;
```

```
  method run();
    y=cdf('GENPOISSON', 9, 1, .7);
    put y=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
y=0.90616296303524
```

関連項目

関数:

- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “PDF 一般化 Poisson 分布 関数” (938 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

CDF GEOMETRIC 分布 関数

幾何累積確率分布の値を返します。

カテゴリ:	確率
返されるデータの種類:	DOUBLE
注:	QUANTILE 関数は、指定した分布から分位点を返します。QUANTILE 関数は、CDF 関数の逆数です。詳細については、“QUANTILE 関数” (1052 ページ)を参照してください。

構文

CDF('GEOMETRIC', *m*, *p*)

必須引数

m
失敗数を指定する数値の確率変数です。

データの種類 DOUBLE

ヒント 10 進数の値は、整数から遠く離れている場合は切り捨てられます。

p
成功確率を指定する数値定数、変数または式です。

範囲 $0 \le p \le 1$

データの種類 DOUBLE

詳細

幾何分布の CDF 関数は、幾何分布(パラメーターは *p*)からオブザベーションが *m* 以下となる確率を返します。

$$CDF('GEOM', m, p) = \begin{cases} 0 & m < 0 \\ 1 - (1-p)^{(m+1)} & m \ge 0 \end{cases}$$

注: この分布には、*位置*または*尺度*パラメーターはありません。

例

次のプログラムは、CDF 幾何分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double y;
  method init();
    y=cdf('GEOMETRIC',5, .35);
    put y=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
y=0.924581109375
```

関連項目

関数:

- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “PDF GEOMETRIC 分布 関数” (940 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

CDF HYPERGEOMETRIC 分布 関数

超幾何累積確率分布の値を返します。

カテゴリ: 確率

返されるデータの種類: DOUBLE

注: QUANTILE 関数は、指定した分布から分位点を返します。QUANTILE 関数は、CDF 関数の逆数です。詳細については、“QUANTILE 関数” (1052 ページ)を参照してください。

構文

CDF('HYPER', *x*, *N*, *R*, *n* [, *o*])

必須引数

x

確率変数を指定する数値定数、変数または式です。この引数は整数にする必要があります。

データの種類 DOUBLE

N

母集団サイズパラメーターを指定する数値定数、変数または式です。この引数は整数にする必要があります。

範囲 *N* = 1, 2, ...

データの種類 DOUBLE

R
対象カテゴリの項目数を指定する数値定数、変数または式です。この引数は整数にする必要があります。

範囲 *R* = 0, 1, ..., *N*

データの種類 DOUBLE

n
サンプルサイズパラメーターを指定する数値定数、変数または式です。この引数は整数にする必要があります。

範囲 *n* = 1, 2, ..., *N*

データの種類 DOUBLE

オプション引数

o
省略可能な数値のオッズ比パラメーターを指定する数値定数、変数または式です。

範囲 *o* > 0

データの種類 DOUBLE

詳細

超幾何分布の CDF 関数は、拡張超幾何分布(母集団サイズは *N*、項目数は *R*、サンプルサイズは *n*、オッズ比は *o*)からオブザベーションが *x* 以下となる確率を返します。*o* が除外されているか 1 の場合、一般超幾何分布の値が返されます。

$$CDF\ ('HYPER', x, N, R, n, o) = \begin{cases} 0 & x < max(0, R+n-N) \\ \dfrac{\sum_{i=0}^{x}\binom{R}{i}\binom{N-R}{n-i}o^i}{\sum_{j=max(0,R+n-N)}^{min(R,n)}\binom{R}{j}\binom{N-R}{n-j}o^j} & max(0, R+n-N) \le x \le min(R, n) \\ 1 & x > min(R, n) \end{cases}$$

例

次のプログラムは、CDF 超幾何分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
```

```
data _null_;
  dcl double x y;
  method init();
    y=cdf('HYPER',2,200,50,10);
    put y=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
y=0.52367340812173
```

関連項目

関数:

- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “PDF 超幾何分布 関数” (942 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

CDF LAPLACE 分布 関数

Laplace 累積確率分布の値を返します。

カテゴリ:	確率
返されるデータの種類:	DOUBLE
注:	QUANTILE 関数は、指定した分布から分位点を返します。QUANTILE 関数は、CDF 関数の逆数です。詳細については、“QUANTILE 関数” (1052 ページ)を参照してください。

構文

CDF('LAPLACE', *x* [,*θ*, *λ*])

必須引数

x
確率変数を指定する数値定数、変数または式です。

データの種類 DOUBLE

オプション引数

θ
位置パラメーターを指定する数値定数、変数または式です。

デフォルト 0

データの種類 DOUBLE

λ
尺度パラメーターを指定する数値定数、変数または式です。

デフォルト 1

範囲 λ > 0

データの種類 DOUBLE

詳細

Laplace 分布の CDF 関数は、Laplace 分布(位置パラメーターは θ、尺度パラメーターは λ)からオブザベーションが *x* 以下となる確率を返します。

$$CDF('LAPLACE', x, \theta, \lambda) = \begin{cases} \frac{1}{2} e^{\frac{(x-\theta)}{\lambda}} & x < \theta \\ 1 - \frac{1}{2} e^{\left(-\frac{(x-\theta)}{\lambda}\right)} & x \geq \theta \end{cases}$$

例

次のプログラムは、CDF Laplace 分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double x y;
  method init();
    y=cdf('LAPLACE',1);
    put y=;
  end;
```

```
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
y=0.81606027941427
```

関連項目

関数:

- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “PDF LAPLACE 分布 関数” (944 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

CDF LOGISTIC 分布 関数

ロジスティック累積確率分布の値を返します。

カテゴリ: 確率

返されるデータの種類: DOUBLE

注: QUANTILE 関数は、指定した分布から分位点を返します。QUANTILE 関数は、CDF 関数の逆数です。詳細については、“QUANTILE 関数” (1052 ページ)を参照してください。

構文

CDF('LOGISTIC', *x* [, *θ*, *λ*])

必須引数

x
確率変数を指定する数値定数、変数または式です。

データの種類 DOUBLE

オプション引数

θ
位置パラメーターを指定する数値定数、変数または式です。

デフォルト　0

データの種類　DOUBLE

λ
尺度パラメーターを指定する数値定数、変数または式です。

デフォルト　1

範囲　λ > 0

データの種類　DOUBLE

詳細

ロジスティック分布の CDF 関数は、ロジスティック分布(位置パラメーターは θ、尺度パラメーターは λ)からオブザベーションが *x* 以下となる確率を返します。

$$CDF('LOGISTIC', x, \theta, \lambda) = \frac{1}{1 + e^{\left(-\frac{x-\theta}{\lambda}\right)}}$$

例

次のプログラムは、CDF ロジスティック分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double x y;
  method init();
    y=cdf('LOGISTIC',1);
    put y=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
y=0.73105857863
```

関連項目

関数:

- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “PDF LOGISTIC 分布 関数” (946 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

CDF LOGNORMAL 分布 関数

対数正規累積確率分布の値を返します。

カテゴリ: 確率

返されるデータの種類: DOUBLE

注: QUANTILE 関数は、指定した分布から分位点を返します。QUANTILE 関数は、CDF 関数の逆数です。詳細については、“QUANTILE 関数” (1052 ページ)を参照してください。

構文

CDF('LOGNORMAL', *x* [, *θ*, *λ*])

必須引数

x
確率変数を指定する数値定数、変数または式です。

データの種類 DOUBLE

オプション引数

θ
対数尺度パラメーターを指定する数値定数、変数または式です。e(θ)は尺度パラメーターです。

デフォルト 0

データの種類 DOUBLE

λ
形状パラメーターを指定する数値定数、変数または式です。

デフォルト	1
範囲	λ > 0
データの種類	DOUBLE

詳細

対数正規分布の CDF 関数は、対数正規分布(対数尺度パラメーターは θ、形状パラメーターは λ)からオブザベーションが *x* 以下となる確率を返します。

$$CDF('LOGN', x, \theta, \lambda) = \begin{cases} 0 & x \le 0 \\ \frac{1}{\lambda\sqrt{2\,\pi}} \int_{-\infty}^{log(x)} e\left(-\frac{(v-\theta)^2}{2\lambda^2}\right) dv & x > 0 \end{cases}$$

例

次のプログラムは、CDF 対数正規分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double x y;
  method init();
    y=cdf('LOGNORMAL',1);
    put y=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
y=0.5
```

関連項目

関数:

- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)

- “LOGSDF 関数” (831 ページ)
- “PDF LOGNORMAL 分布 関数” (948 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

CDF NEGBINOMIAL 分布 関数

負の二項累積確率分布の値を返します。

カテゴリ: 確率

返されるデータの種類: DOUBLE

注: QUANTILE 関数は、指定した分布から分位点を返します。QUANTILE 関数は、CDF 関数の逆数です。詳細については、“QUANTILE 関数” (1052 ページ)を参照してください。

構文

CDF('NEGBINOMIAL', *m*, *p*, *n*)

必須引数

m
失敗数を数える確率変数を指定する、数値定数、変数または式です。この引数は正の整数にする必要があります。

範囲 *m* = 0, 1, ...

データの種類 DOUBLE

p
成功確率を指定する数値定数、変数または式です。

範囲 $0 \le p \le 1$

データの種類 DOUBLE

n
成功数を数える値を指定する数値定数、変数または式です。

範囲 $n > 0$

データの種類 DOUBLE

詳細

負の二項分布の CDF 関数は、負の二項分布(成功確率は *p*、成功数は *n*)からオブザベーションが *m* 以下となる確率を返します。

$$CDF('NEGB', m, p, n) = \begin{cases} 0 & m < 0 \\ p^n \sum_{j=0}^{m} \binom{n+j-1}{n-1} (1-p)^j & m \geq 0 \end{cases}$$

注: 負の二項分布には、*位置*または*尺度*パラメーターはありません。

例

次のプログラムは、CDF 負の二項分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double x y;
  method init();
    y=cdf('NEGB',1,.5,2);
    put y=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
y=0.5
```

関連項目

関数:

- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “PDF NEGBINOMIAL 分布 関数” (949 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

CDF NORMAL 分布 関数

正規累積確率分布の値を返します。

カテゴリ: 確率

返されるデータの種類: DOUBLE

注: QUANTILE 関数は、指定した分布から分位点を返します。QUANTILE 関数は、CDF 関数の逆数です。詳細については、“QUANTILE 関数” (1052 ページ)を参照してください。

構文

CDF('NORMAL', *x* [, *θ*, *λ*])

必須引数

x
確率変数を指定する数値定数、変数または式です。

データの種類 DOUBLE

オプション引数

θ
位置パラメーターを指定する数値定数、変数または式です。

デフォルト 0

データの種類 DOUBLE

λ
尺度パラメーターを指定する数値定数、変数または式です。

デフォルト 1

範囲 λ > 0

データの種類 DOUBLE

詳細

正規分布の CDF 関数は、正規分布(位置パラメーターは θ、尺度パラメーターは λ)からオブザベーションが *x* 以下となる確率を返します。

$$CDF('NORMAL', x, \theta, \lambda) = \frac{1}{\lambda\sqrt{2\,\pi}} \int_{-\infty}^{x} e\left(-\frac{(v-\theta)^2}{2\lambda^2}\right) dv$$

例

次のプログラムは、CDF 正規分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double x y;
  method init();
    y=cdf('NORMAL',1.96);
    put y=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
y=0.97500210485177
```

関連項目

関数:

- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “PDF NORMAL 分布 関数” (951 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

CDF NORMALMIX 分布 関数

正規混合累積確率分布の値を返します。

カテゴリ: 確率

返されるデータの種類: DOUBLE

注: QUANTILE 関数は、指定した分布から分位点を返します。QUANTILE 関数は、CDF 関数の逆数です。詳細については、“QUANTILE 関数” (1052 ページ)を参照してください。

構文

CDF('NORMALMIX', *x*, *n*, *p*, *m*, *s*)

必須引数

x

確率変数を指定する数値定数、変数または式です。

データの種類 DOUBLE

n

混合数を指定する数値定数、変数または式です。

範囲 *n* = 1, 2, ...

データの種類 DOUBLE

p

n 個の比率$p_1, p_2, \ldots, p_n$を指定する、数値の定数、変数または式です。ここで、$\sum_{i=1}^{i=n} p_i = 1$です。

範囲 *p* = 0, 1, ...

データの種類 DOUBLE

m

n 個の平均$m_1, m_2, \ldots, m_n$を指定する数値定数、変数または式です。

データの種類 DOUBLE

s

n 個の標準偏差$s_1, s_2, \ldots, s_n$を指定する数値定数、変数または式です。

範囲 *s* > 0

データの種類 DOUBLE

詳細

正規混合分布の CDF 関数は、正規混合分布のオブザベーションが *x* 以下となる確率を返します。

$$CDF('NORMALMIX', x, n, p, m, s) = \sum_{i=1}^{i=n} p_i \, CDF('NORMAL', x, m_i, s_i)$$

正規混合分布の重み付けは、負でないことが必要です。重み付けの合計が 1 に等しくない場合は相対的な重み付けとして扱われ、合計が 1 となるように調整されます。

注: 正規混合分布には、*位置*または*尺度*パラメーターはありません。

例

次のプログラムは、CDF 正規混合分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double x y;
  method init();
    y=cdf('Normalmix',2.3,3,.33,.33,.34,.5,1.5,2.5,.79,1.6,4.3);
    put y=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
y=0.71813087231314
```

関連項目

関数:

- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “PDF NORMALMIX 分布 関数” (953 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)

- “SQUANTILE 関数” (1149 ページ)

CDF PARETO 分布 関数

パレート累積確率分布の値を返します。

カテゴリ: 確率

返されるデータの種類: DOUBLE

注: QUANTILE 関数は、指定した分布から分位点を返します。QUANTILE 関数は、CDF 関数の逆数です。詳細については、“QUANTILE 関数” (1052 ページ)を参照してください。

構文

CDF('PARETO', *x*, *a* [, *k*])

必須引数

x
確率変数を指定する数値定数、変数または式です。

データの種類 DOUBLE

a
形状パラメーターを指定する数値定数、変数または式です。

範囲 $a > 0$

データの種類 DOUBLE

オプション引数

k
尺度パラメーターを指定する数値定数、変数または式です。

デフォルト 1

範囲 $k > 0$

データの種類 DOUBLE

詳細

パレート分布の CDF 関数は、パレート分布(形状パラメーターは *a*、尺度パラメーターは *k*)からオブザベーションが *x* 以下となる確率を返します。

$$CDF('PARETO', x, a, k) = \begin{cases} 0 & x < k \\ 1 - \left(\frac{k}{x}\right)^a & x \geq k \end{cases}$$

例

次のプログラムは、CDF パレート分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double x y;
  method init();
    y=cdf('PARETO',1,1);
    put y=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
y=0
```

関連項目

関数:

- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “PDF PARETO 分布 関数” (955 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

CDF POISSON 分布 関数

Poisson 累積確率分布の値を返します。

カテゴリ: 確率

返されるデータの種類: DOUBLE

注: QUANTILE 関数は、指定した分布から分位点を返します。QUANTILE 関数は、CDF 関数の逆数です。詳細については、“QUANTILE 関数” (1052 ページ)を参照してください。

構文

CDF('POISSON', *n*, *m*)

必須引数

n
確率変数を指定する数値定数、変数または式です。この引数は整数にする必要があります。

範囲 *n* = 0, 1, ...

データの種類 DOUBLE

m
平均パラメーターを指定する数値定数、変数または式です。

範囲 *m* > 0

データの種類 DOUBLE

詳細

Poisson 分布の CDF 関数は、Poisson 分布(平均は *m*)からオブザベーションが *n* 以下となる確率を返します。

$$CDF('POISSON', n, m) = \begin{cases} 0 & n < 0 \\ \sum_{i=0}^{n} e^{-m}\frac{m^i}{i!} & n \geq 0 \end{cases}$$

注: Poisson 分布には、*位置*または*尺度*パラメーターはありません。

例

次のプログラムは、CDF Poisson 分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double x y;
  method init();
    y=cdf('POISSON',2,1);
    put y=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
y=0.9196986029286
```

関連項目

関数:

- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “PDF POISSON 分布 関数” (957 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

CDF T 分布 関数

T 累積確率分布の値を返します。

カテゴリ:	確率
返されるデータの種類:	DOUBLE

注: QUANTILE 関数は、指定した分布から分位点を返します。QUANTILE 関数は、CDF 関数の逆数です。詳細については、“QUANTILE 関数” (1052 ページ)を参照してください。

構文

CDF('T', *t*, *df* [, *nc*])

必須引数

t
確率変数を指定する数値定数、変数または式です。

データの種類 DOUBLE

df
自由度を指定する数値定数、変数または式です。

範囲 *df* > 0

データの種類 DOUBLE

オプション引数

nc
省略可能な非心度パラメーターを指定する数値定数、変数または式です。

データの種類 DOUBLE

詳細

T 分布の CDF 関数は、*T* 分布(自由度は *df*、非心度パラメーターは *nc*)からオブザベーションが *x* 以下となる確率を返します。この関数では、整数以外の自由度を使用できます。*nc* が除外されているかゼロの場合、心度 *T* 分布の値が返されます。次の式で、$\nu = df$として、そして$\delta = nc$とします。

$$CDF('T', t, \nu, \delta) = \frac{1}{2^{(\nu/2-1)}\Gamma\left(\frac{\nu}{2}\right)} \int_0^{\infty} x^{\nu-1} e^{-\frac{1}{2}x^2} \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\frac{tx}{\sqrt{\nu}}} e^{-\frac{1}{2}(u-\delta)^2} du dx$$

注: *T* 分布には、*位置*または*尺度*パラメーターはありません。

例

次のプログラムは、CDF T 分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double x y;
  method init();
    y=cdf('T',.9,5);
    put y=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
y=0.79531439982768
```

関連項目

関数:

- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “PDF T 分布 関数” (959 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

CDF TWEEDIE 分布 関数

Tweedie 累積確率分布の値を返します。

カテゴリ:	確率
返されるデータの種類:	DOUBLE
注:	QUANTILE 関数は、指定した分布から分位点を返します。QUANTILE 関数は、CDF 関数の逆数です。詳細については、“QUANTILE 関数” (1052 ページ)を参照してください。

構文

CDF ('TWEEDIE', *y*, *p* [, *μ*, *φ*])

必須引数

y

確率変数を指定する数値定数、変数または式です。

範囲	*y* ≥0
データの種類	DOUBLE
注	この引数は必須です。 *p*>1 の場合、*y* は数値になります。*p*=1 の場合、*y* は整数になります。

p

累乗パラメーターを指定する数値定数、変数または式です。

範囲	*p* ≥1
データの種類	DOUBLE
注	この引数は必須です。

オプション引数

μ

平均パラメーターを指定する数値定数、変数または式です。

デフォルト	1
範囲	*μ*>0
データの種類	DOUBLE

φ

ばらつきのパラメーターを指定する数値定数、変数または式です。

デフォルト	1
範囲	*φ*>0
データの種類	DOUBLE

詳細

Tweedie 分布の CDF 関数は、次の式の関係がある分散と平均を使用した指数散布度モデルを返します。*variance*= $\phi \times \mu^p$

$$\int_0^y \frac{1}{y} \sum_{j=1}^{\infty} \left(\frac{y^{-j\alpha}(p-1)^{ja}}{\phi^{j(1-\alpha)}(2-p)^j j!\,\Gamma(-j\alpha)} \right) e\left(\frac{1}{\phi} \left(y \frac{\mu^{1-p}-1}{1-p} - \frac{\mu^{2-p}-1}{2-p} \right) \right) dy$$

前述の式には次の関係が適用されます。

$$\alpha = \frac{2-p}{1-p}$$

注: 計算された Tweedie 確率の精度は、パラメーター空間における位置に大きく依存します。通常、10 桁の精度を利用できますが、*p* の近似値が 2 の場合やファイの近似値が 0 の場合は、6 桁の精度に低下する可能性があります。

例

次のプログラムは、CDF Tweedie 分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double x y;
  method init();
    y=cdf('TWEEDIE', .8, 5);
    put y=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
y=0.59176291643197
```

関連項目

関数:

- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “PDF TWEEDIE 分布 関数” (961 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

CDF UNIFORM 分布 関数

一様累積確率分布の値を返します。

カテゴリ: 確率

返されるデータの種類: DOUBLE

注: QUANTILE 関数は、指定した分布から分位点を返します。QUANTILE 関数は、CDF 関数の逆数です。詳細については、“QUANTILE 関数” (1052 ページ)を参照してください。

構文

CDF('UNIFORM', *x* [, *l*, *r*])

必須引数

x
確率変数を指定する数値定数、変数または式です。

データの種類 DOUBLE

オプション引数

l
左位置パラメーターを指定する数値定数、変数または式です。

デフォルト 0

データの種類 DOUBLE

r
右位置パラメーターを指定する数値定数、変数または式です。

デフォルト 1

範囲 *r* > *l*

データの種類 DOUBLE

詳細

一様分布の CDF 関数は、一様分布(左位置パラメーターは *l*、右位置パラメーターは *r*)からオブザベーションが *x* 以下となる確率を返します。

$$CDF('UNIFORM', x, l, r) = \begin{cases} 0 & x < l \\ \frac{x-l}{r-l} & l \le x < r \\ 1 & x \ge r \end{cases}$$

注: *l* と *r* のデフォルト値は、それぞれ 0 と 1 になります。

例

次のプログラムは、CDF 一様分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double x y;
  method init();
   y=cdf('UNIFORM',0.25);
   put y=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
y=0.25
```

関連項目

関数:

- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “PDF UNIFORM 分布 関数” (963 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

CDF WALD (逆ガウス)分布 関数

Wald (逆ガウスとも呼ばれる)累積確率分布から値を返します。

カテゴリ: 確率

返されるデータの種類: DOUBLE

注: QUANTILE 関数は、指定した分布から分位点を返します。QUANTILE 関数は、CDF 関数の逆数です。詳細については、“QUANTILE 関数” (1052 ページ)を参照してください。

構文

CDF('WALD', *x*, *λ* [, *μ*])

CDF('IGAUSS', *x*, *λ* [, *μ*])

必須引数

x

確率変数を指定する数値定数、変数または式です。

データの種類 DOUBLE

λ

形状パラメーターを指定する数値定数、変数または式です。

範囲 $\lambda > 0$

データの種類 DOUBLE

オプション引数

μ

平均パラメーターを指定する数値定数、変数または式です。

デフォルト 1

範囲 $\mu > 0$

データの種類 DOUBLE

詳細

Wald 分布の CDF 関数は、Wald 分布(形状パラメーターは λ)からオブザベーションが *x* 以下となる確率を返します。

$$Fx(x) = \Phi\left\{\sqrt{\frac{\lambda}{x}}\left(\frac{x}{\mu} - 1\right)\right\} + e^{2\lambda/\mu}\Phi\left\{-\sqrt{\frac{\lambda}{x}}\left(\frac{x}{\mu} + 1\right)\right\}$$

式では、Φ(.)は、正規累積分布関数です。x≤0 の場合、CDF は 0 になります。

例

次のプログラムは、CDF Wald 分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double x y;
  method init();
   y=cdf('WALD',1,2);
   put y=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
y=0.62769783815525
```

関連項目

関数:

- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “PDF Wald (逆ガウス)分布 関数” (965 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

CDF WEIBULL 分布 関数

Weibull 累積確率分布の値を返します。

カテゴリ: 確率

返されるデータの種類: DOUBLE

注: QUANTILE 関数は、指定した分布から分位点を返します。QUANTILE 関数は、CDF 関数の逆数です。詳細については、“QUANTILE 関数” (1052 ページ)を参照してください。

構文

CDF('WEIBULL', *x*, *α* [, *λ*])

必須引数

x
確率変数を指定する数値定数、変数または式です。

データの種類 DOUBLE

α
形状パラメーターを指定する数値定数、変数または式です。

範囲 $\alpha > 0$

データの種類 DOUBLE

オプション引数

λ
尺度パラメーターを指定する数値定数、変数または式です。

デフォルト 1

範囲 $\lambda > 0$

データの種類 DOUBLE

詳細

Weibull 分布の CDF 関数は、Weibull 分布(形状パラメーターは *α*、尺度パラメーターは λ)からオブザベーションが *x* 以下となる確率を返します。

$$CDF('WEIBULL', x, a, \lambda) = \begin{cases} 0 & x < 0 \\ 1 - e^{-\left(\frac{x}{\lambda}\right)^a} & x \geq 0 \end{cases}$$

例

次のプログラムは、CDF Weibull 分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double x y;
  method init();
    y=cdf('WEIBULL',1,2);
    put y=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
y=0.63212055882855
```

関連項目

関数:

- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “PDF WEIBULL 分布 関数” (967 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

CEIL 関数

数値式以上の最小の整数を返します。

カテゴリ: 切り捨て

返されるデータの種類: DECIMAL、DOUBLE、NUMERIC

構文

CEIL(*expression*)

必須引数

expression
数値に評価される有効な式を指定します。

データの種類 DECIMAL、DOUBLE、NUMERIC

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

詳細

結果が引数のデータ型の範囲に収まらない数値の場合、CEIL 関数は失敗します。

引数が DECIMAL の場合、結果は DECIMAL になります。それ以外の場合、引数は (まだ変換されていなければ) DOUBLE に変換され、結果は DOUBLE になります。

比較

CEILZ 関数とは異なり、CEIL 関数は結果をファジー処理します。引数が整数の 1E-12 内にある場合、CEIL 関数はその整数に等しくなるように結果をファジー処理します。CEILZ 関数は結果をファジー処理しません。そのため、CEILZ 関数では予期しない結果になる可能性があります。

例

次のプログラムは、CEIL 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double var1 a b c d e f g h;
  method run();
    var1=2.1;
    a=ceil(var1);
    b=ceil(-2.4);
```

```
    c=ceil(1+1.e-11);
    d=ceil(-1+1.e-11);
    e=ceil(1+1.e-13);
    f=ceil(223.456);
    g=ceil(763);
    h=ceil(-223.456);
    put 'a= ' a;
    put 'b= ' b;
    put 'c= ' c;
    put 'd= ' d;
    put 'e= ' e;
    put 'f= ' f;
    put 'g= ' g;
    put 'h= ' h;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
a= 3
b= -2
c= 2
d= 0
e= 1
f= 224
g= 763
h= -223
```

関連項目

関数:

- “CEILZ 関数” (412 ページ)
- “FLOOR 関数” (653 ページ)
- “FLOORZ 関数” (655 ページ)

CEILZ 関数

ゼロファジーを使用して、引数より大きいか等しい整数のうち最小の値を返します。

カテゴリ:	切り捨て
返されるデータの種類:	DOUBLE

構文

CEILZ(*expression*)

必須引数

expression
数値に評価される有効な式を指定します。

データの種類 DOUBLE

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

比較

CEIL 関数とは異なり、CEILZ 関数はゼロファジーを使用します。引数が整数の 1E-12 内にある場合、CEIL 関数はその整数に等しくなるように結果をファジー処理します。CEILZ 関数は結果をファジー処理しません。そのため、CEILZ 関数では予期しない結果になる可能性があります。

例

次のプログラムは、CEILZ 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double a b c d e f g h;
  method run();
    a=ceilz(2.1);
    b=ceilz(-2.4);
    c=ceilz(1+1.e-11);
    d=ceilz(-1+1.e-11);
    e=ceilz(1+1.e-13);
    f=ceilz(223.456);
    g=ceilz(763);
    h=ceilz(-223.456);
    put 'a= ' a;
    put 'b= ' b;
    put 'c= ' c;
    put 'd= ' d;
    put 'e= ' e;
    put 'f= ' f;
    put 'g= ' g;
    put 'h= ' h;
  end;
```

```
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
a= 3
b= -2
c= 2
d= 0
e= 2
f= 224
g= 763
h= -223
```

関連項目

関数:

- “CEIL 関数” (410 ページ)
- “FLOOR 関数” (653 ページ)
- “FLOORZ 関数” (655 ページ)

CHOOSEC 関数

引数のリストからの選択結果を表す文字値を返します。

カテゴリ: 文字

返されるデータの種類: VARCHAR、NVARCHAR

構文

CHOOSEC(*index-expression*, *selection-1*[, ...*selection-n*])

必須引数

index-expression
数値に評価される有効な式を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

selection
文字定数、変数または式を指定します。この引数の値は、CHOOSEC 関数によって返されます。

データの種類 CHAR、VARCHAR、NCHAR、NVARCHAR

詳細

CHOOSEC 関数は、*index-expression* の値を使用して、後続の引数から選択します。たとえば、*index-expression* が 3 の場合、CHOOSEC は *selection-3* の値を返します。第 1 引数が負の場合、関数は引数のリストを逆方向に数えてその値を返します。

比較

CHOOSEC 関数は CHOOSEN 関数と類似していますが、CHOOSEN が数値を返すのに対して CHOOSEC は文字値を返す点が異なります。

例

次のプログラムでは、CHOOSEC がどのように一連の値から選択しているのかを示します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl char fruit color planet sport;
  method init();
    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=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
fruit=apple   color=green   planet=Mercury  sport=baseball
```

関連項目

関数:

- “CHOOSEN 関数” (416 ページ)

CHOOSEN 関数

引数のリストからの選択結果を表す数値を返します。

カテゴリ: 数値

返されるデータの種類: DOUBLE

構文

CHOOSEN(*index-expression*, *selection-1*[, ...*selection-n*])

必須引数

index-expression
数値に評価される有効な式を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

selection
数値の定数、変数または式を指定します。この引数の値は、CHOOSEN 関数によって返されます。

データの種類 DOUBLE

詳細

CHOOSEN 関数は、*index-expression* の値を使用して、後続の引数から選択します。たとえば、*index-expression* が 3 の場合、CHOOSEN は *selection-3* の値を返します。第 1 引数が負の場合、関数は引数のリストを逆方向に数えてその値を返します。

比較

CHOOSEN 関数は CHOOSEC 関数と類似していますが、CHOOSEN が数値を返すのに対して CHOOSEC は文字値を返す点が異なります。

例

次のプログラムでは、CHOOSEN がどのように一連の値から選択しているのかを示します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test;
  dcl double itemnumber rank score value;
  method run();
    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= ' ItemNumber;
    put 'Rank= ' Rank;
    put 'Score= ' Score;
    put 'Value= ' Value;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
ItemNumber=  679
Rank=  4
Score=  33
Value=  1014
```

CMISS 関数

欠損引数の数を数えます。

カテゴリ: 記述統計

注: 2025.12 以降、この関数では、引数が DOUBLE の場合、すべての引数が DOUBLE に変換され、結果は DOUBLE になります。PROC DS2 ステートメントで BIGINTPROCESSING=YES が設定され、いずれかの引数が BIGINT の場合、すべての引数が BIGINT に変換され、結果は BIGINT になります。

構文

CMISS(*argument* [, ...*argument*,])

必須引数

argument
定数、変数または式を指定します。*argument* は、文字値または数値のいずれかになります。

詳細

文字式の評価結果がすべて空白の文字列または長さがゼロの文字列の場合、マクロ処理で CMISS 関数を使用していない限り、その文字式は欠損とみなされます。数値式の評価結果が数値欠損値(.、._、.A、...、.Z)の場合、その数値式は欠損として数えられます。

マクロ処理で CMISS 関数を使用する場合は、ピリオド(.)を使用して、文字欠損値と数値欠損値の両方を表します。文字引数に空白または null 値を使用した場合、SAS はエラーを返します。エラーが発生する 3 つの例を次に示します。

```
%let macvar=%sysfunc(cmiss(A,%str( )));
%let macvar=%sysfunc(cmiss(A, ));
%let macvar=%sysfunc(cmiss(A,));
```

エラー状態を回避するのに使用する例を次に示します。

```
%let macvar=%sysfunc(cmiss(A,.));
```

例

次のプログラムは、CMISS 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl char a b;
  method run();
    a=cmiss(1,0, ' ' , 2,5,' ');
    b=cmiss(1, ' ');
    put  'a=' a;
    put 'b=' b;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
a= 2
b= 1
```

関連項目

関数:

- “MISSING 関数” (855 ページ)
- “NMISS 関数” (872 ページ)

CMP 関数

末尾の空白を含む 2 つの文字列を比較します。

カテゴリ: 文字

返されるデータの種類: BIGINT

構文

CMP (*string-1*, *string-2*)

必須引数

string-1
文字列を評価するか、文字列に強制変換できる文字定数、変数または式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

string-2
文字列を評価するか、文字列に強制変換できる文字定数、変数または式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

詳細

CMP 関数では、*string–1* と *string–2* に違いがない場合はゼロ値を返します。引数が異なる場合、並べ替え順序で *string-1* が string-2 よりも前にある場合は結果の符合が負になり、*string-1* が *string-2* よりも後にある場合は正になります。

CMP 関数は、末尾の空白を削除しません。

比較

CMP 関数は、2 つの文字列を比較しますが、末尾の空白は削除しません。CMPT 関数は、2 つの文字列を比較し、末尾の空白を削除します。

例

次のプログラムは、CMP 関数を使用して 2 つの異なる文字列を比較します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double nopad pad greaterthan lessthan;
  method run();
    nopad=cmp('abc', 'def');
    pad=cmp('abc', 'abc ');
    greaterthan=cmp('abc', 'abcdef');
    lessthan=cmp('abcdef', 'abc');
    put nopad= pad= greaterthan= lessthan=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
nopad=-1 pad=-4 greaterthan=-4 lessthan=4
```

関連項目

関数:

- “CMPT 関数” (420 ページ)

CMPT 関数

末尾の空白を除く 2 つの文字列を比較します。

カテゴリ:	文字
返されるデータの種類:	BIGINT

構文

CMPT (*string-1*, *string-2*)

必須引数

string-1
文字列を評価するか、文字列に強制変換できる文字定数、変数または式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

string-2
文字列を評価するか、文字列に強制変換できる文字定数、変数または式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

詳細

CMPT 関数では、*string–1* と *string–2* に違いがない場合はゼロ値を返します。引数が異なる場合、並べ替え順序で *string-1* が string-2 よりも前にある場合は結果の符合が負になり、*string-1* が *string-2* よりも後にある場合は正になります。

CMPT 関数は、末尾の空白を削除します。

比較

CMPT 関数は、2 つの文字列を比較し、末尾の空白を削除します。CMP 関数は、2 つの文字列を比較し、末尾の空白は削除しません。

例

次のプログラムは、CMPT 関数を使用して 2 つの異なる文字列を比較します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
dcl double nopad pad greaterthan lessthan;
  method run();
    nopad=cmpt('abc', 'def');
    pad=cmpt('abc', 'abc ');
    greaterthan=cmpt('abc', 'abcdef');
```

```
    lessthan=cmpt('abcdef', 'abc');
    put nopad= pad= greaterthan= lessthan=;
  end;
enddata;
run;
quit;
```

```
nopad=0 pad=0 greaterthan=-4 lessthan=4
```

関連項目

関数:

- “CMP 関数” (419 ページ)

CNONCT 関数

カイ 2 乗分布の非心度パラメーターを返します。

カテゴリ:	数学
返されるデータの種類:	DOUBLE

構文

CNONCT(*x*, *df*, *probability*)

必須引数

x
数値の確率変数です。

範囲	$x \geq 0$
データの種類	DOUBLE

df
数値の自由度パラメーターです。

範囲	$df > 0$
データの種類	DOUBLE

probability
確率です。

範囲	0 < *probability* < 1
データの種類	DOUBLE

詳細

CNONCT 関数は、非心度カイ 2 乗分布(パラメーターは *x*、*df*、*nc*)から負でない非心度パラメーターを返します。*probability* が心度カイ 2 乗分布(パラメーターは *x* および *df*)からの確率よりも大きい場合、この問題の平方根は存在しません。この場合、欠損値が返されます。式の負でない平方根 *nc* を検索するには、ニュートン型のアルゴリズムを使用します。

$$P_c(x|\mathrm{df}, nc) - prob = 0$$

前述の式には次の関係が適用されます。

$$P_c(x|\mathrm{df}, \mathrm{nc}) = \varepsilon^{\frac{-nc}{2}} \sum_{j=0}^{\infty} \frac{\left(\frac{nc}{2}\right)^j}{j!} P_g\left(\frac{x}{2}\middle|\frac{df}{2} + j\right)$$

前述の式には次の関係が適用されます。

$$P_g(x|a)$$

次の式によって得られるガンマ分布の確率です。

$$P_g(x|a) = \frac{1}{\Gamma(a)} \int_0^x t^{a-1} \varepsilon^{-t} dt$$

アルゴリズムで固定小数点を収束できない場合、欠損値が返されます。

例

この例は、CNONCT 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data work /overwrite=yes;
  dcl double x df nc prob ncc;
  method init();
    x=2;
    df=4;
    do nc=1 to 3 by .5;
      prob=probchi(x, df, nc);
      ncc=cnonct(x, df, prob);
      output;
    end;
  end;
```

```
enddata;
run;
quit;

proc print data=work;
run;
```

アウトプット 7.3　カイ 2 乗分布の非心度パラメーターの計算

Obs	x	df	nc	prob	ncc
1	2	4	1.0	0.18611	1.0
2	2	4	1.5	0.15592	1.5
3	2	4	2.0	0.13048	2.0
4	2	4	2.5	0.10907	2.5
5	2	4	3.0	0.09109	3.0

関連項目

関数:

- “FNONCT 関数” (659 ページ)
- “TNONCT 関数” (1195 ページ)

COALESCE 関数

数値の引数のリストから最初の非 null または非欠損値を返します。

カテゴリ:	数学
返されるデータの種類:	DOUBLE
注:	2025.12 以降、この関数では、引数が DOUBLE の場合、すべての引数が DOUBLE に変換され、結果は DOUBLE になります。PROC DS2 ステートメントで BIGINTPROCESSING=YES が設定され、いずれかの引数が BIGINT の場合、すべての引数が BIGINT に変換され、結果は BIGINT になります。

構文

COALESCE(*expression*[, ...*expression*])

必須引数

expression
数値に評価される有効な式を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

COALESCE では、1 つ以上の数値式を使用できます。COALESCE 関数は、リストされている順に各式の値を確認し、最初の非 null または非欠損値を返します。リストされている値が 1 つだけの場合、COALESCE 関数はその引数の値を返します。すべての式のすべての値が null または欠損値の場合、COALESCE 関数は、ANSI モードまたは SAS モードのどちらであるかに応じて、null または欠損値を返します。詳細については、“DS2 による null および SAS 欠損値の処理方法” (*SAS DS2 プログラマガイド*)を参照してください。

比較

COALESCE 関数は数値式を検索するのに対し、COALESCEC 関数は文字式を検索します。

例

次のプログラムは、COALESCE 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data one(overwrite=yes);
  dcl double w x y z;
  method run();
    w = coalesce(., .A, 33, 22, 44, .);
    x = coalesce(42, .);
    y = coalesce(.A, .B, .C);
    z = coalesce(., 7, ., ., 42);
    put w=;
    put x=;
    put y=;
    put z=;
  end;
enddata;
run;
```

```
quit;
```

SAS は次の出力をログに書き込みます。

```
w=33
x=42
y=.
z=7
```

関連項目

関数:

- “COALESCEC 関数” (426 ページ)

COALESCEC 関数

文字引数のリストから最初の非 null または非欠損値を返します。

カテゴリ:	文字
返されるデータの種類:	CHAR、NCHAR、NVARCHAR、VARCHAR

構文

COALESCEC(*expression*[, ...*expression*])

必須引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類	CHAR、NCHAR、NVARCHAR、VARCHAR
参照項目:	“DS2 式” (*SAS DS2 プログラマガイド*)

詳細

COALESCEC では、1 つ以上の文字式を使用できます。COALESCEC 関数は、リストされている順に各式の値を確認し、最初の非 null または非欠損値を返します。リストされている値が 1 つだけの場合、COALESCEC 関数はその式の値を返します。すべての式のすべての値が null または欠損値の場合、COALESCEC 関数は、ANSI モードまたは SAS モードのどちらであるかに応じて、null または欠損値を返します。詳

細については、“DS2 による null および SAS 欠損値の処理方法” (*SAS DS2 プログラマガイド*)を参照してください。

比較

COALESCEC 関数は文字式を検索するのに対し、COALESCE 関数は数値式を検索します。

例

次のプログラムは、COALESCEC 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl char a b;
  method run();
    a=coalescec('', 'Hello');
    put a;
    b=coalescec('', 'Goodbye', 'Hello');
    put b;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
Hello
Goodbye
```

関連項目

関数:

- “COALESCE 関数” (424 ページ)

COMB 関数

n 個の要素を同時に *r* 個使用するときの組み合わせの数を計算します。

カテゴリ: 組み合わせ

返されるデータの種類: DOUBLE

構文

COMB(*n*, *r*)

必須引数

n
負でない整数で要素の合計数を表します。サンプルはこの中から選ばれます。

データの種類 DOUBLE

r
負でない整数で選ばれた要素の数を表します。

制限事項 $r \le n$

データの種類 DOUBLE

詳細

COMB 関数の数学的表現は、次の式によって得られます。

$$COMB\,(n,r) = \binom{n}{r} = \frac{n!}{r!\,(n-r)!}$$

前述の式で、$n \ge 0$、$r \ge 0$ および $n \ge r$ です。

式による計算ができない場合は欠損値が返されます。やや大きな値の場合、COMB 関数を計算できないことがあります。

例

次のプログラムは、COMB 関数を示しています。

```
proc ds2;
data _null_;
  vararray char x[5];
  dcl double k n ncomb;
  method init();
    x:=('ant' 'bee' 'cat' 'dog' 'ewe');
  end;
  method run();
```

```
    n=dim(x);
    k=3;
    ncomb=comb(n, k);
    put ncomb=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
ncomb=10
```

関連項目

関数:

- “FACT 関数” (532 ページ)
- “PERM 関数” (968 ページ)

COMPARE 関数

2 つの文字列を比較し、異なる文字が検出された場合には最も左にある文字の位置を返し、異なる文字が検出されない場合には 0 を返します。

カテゴリ:	文字
返されるデータの種類:	DOUBLE

構文

COMPARE(*string-1*, *string-2*[, *modifiers*])

必須引数

string-1
文字列を評価するか、文字列に強制変換できる文字定数、変数または式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

string-2
文字列を評価するか、文字列に強制変換できる文字定数、変数または式を指定します。

データの種類　CHAR、NCHAR、NVARCHAR、VARCHAR

オプション引数

modifiers
COMPARE 関数のアクションを変更できる文字列を指定します。

次の 1 つ以上の文字を有効な修飾子として使用できます。

i
string-1 および *string-2* の大文字と小文字を区別しません。

l
値を比較する前に *string-1* および *string-2* の先頭の空白を削除します。

n
名前リテラルの引数から引用符を削除し、*string-1* および *string-2* の大文字と小文字を区別しません。名前リテラルは、引用符内の文字列として表される名前トークンで、大文字または小文字の *n* が後に続きます。名前リテラルにより、テーブルまたは変数名に使用できなかった特殊文字(空白文字を含む)を使用できるようになります。COMPARE で文字列が名前リテラルとして認識されるように、最初の文字を引用符にする必要があります。

:
(コロン) *string-1* と *string-2* の短い方の文字列の長さまたは 1 文字(どちらか大きい方)になるように長い方の文字列を切り捨てます。この修飾子を指定しない場合、長い方の文字列の長さと同じになるように短い方の文字列に空白が埋め込まれます。

データの種類　CHAR、NCHAR、NVARCHAR、VARCHAR

ヒント　COMPARE は、修飾子として使用される空白を無視します。

詳細

基本

COMPARE 関数では、修飾子の出現順序に意味があります。

- "LN"は各文字列から先頭の空白を削除した後、名前リテラルから引用符を削除します。
- "NL"は名前リテラルから引用符を削除した後、各文字列から先頭の空白を削除します。

COMPARE 関数では、*string–1* と *string–2* に違いがない場合はゼロ値を返します。引数が異なる場合、次のようになります。

- 並べ替え順序で *string–1* が *string–2* よりも前にある場合は結果の符合が負になり、*string–1* が *string–2* よりも後にある場合は正になります。
- 結果の大きさは、文字列が異なる最も左側の文字の位置と等しくなります。

DBCS の互換性

この関数に相当する DBCS 関数は、KCOMPARE です。COMPARE 関数と KCOMPARE 関数には若干の違いがあります。どちらの関数もさまざまな数の引数を受け入れますが、第 3 引数の使用法には互換性がありません。次の例は、構文の違いを示しています。

COMPARE(*string-1*, *string-2*[, *modifiers*])

KCOMPARE(*string-1*[, *position*[, *count*]], *string-2*)

詳細については、*"KCOMPARE 関数" (SAS 各国語サポート(NLS): リファレンスガイド)* を参照してください。

例

例 1: 2 つの文字列を比較するときの比較順序の理解

次のプログラムでは、COMPARE 関数を使用して 2 つの文字列を比較します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test /overwrite=yes;
  dcl char string1 string2 modifiers having informat $char8. format $char8.;
  method init();
    string1='12345678'; string2='12345678'; output;
    string1='123'; string2='abc'; output;
    string1='abc'; string2='abx'; output;
    string1='xyz'; string2='abcdef'; output;
    string1='aBc'; string2='abc'; output;
    string1='aBc'; string2='AbC'; modifiers='i'; output;
    string1='   abc  '; string2='abc'; modifiers=' '; output;
    string1='   abc  '; string2='abc'; modifiers='l'; output;
    string1=' abc    '; string2='   abx  '; modifiers=' ';output;
    string1=' abc    '; string2='   abx  '; modifiers='l';  output;
  end;
enddata;
run;

data test_out /overwrite=yes;
  method run();
    set test;
    result=compare(string1, string2, modifiers);
    put 'String 1= ' string1 'String 2= ' string2 'Modifier= ' modifiers
      'Result= ' result;
  end;
run;
quit;

proc print data=test_out noobs;run;quit;
```

アウトプット 7.4　COMPARE 関数を使用して 2 つの文字列を比較した結果

string1	string2	modifiers	result
12345678	12345678		0
123	abc		-1
abc	abx		-3
xyz	abcdef		1
aBc	abc		-2
aBc	AbC	i	0
abc	abc		-1
abc	abc	I	0
abc	abx		2
abc	abx	I	-3

例 2: COMPARE 関数を使用した文字列の切り捨て

次のプログラムでは、: (コロン)修飾子を使用して文字列を切り捨てています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test2 /overwrite=yes;
  dcl double pad1 pad2 truncate1 truncate2 blank;
  method run();
    pad1=compare('abc','abc          ');
    pad2=compare('abc','abcdef       ');
    truncate1=compare('abc','abcdef',':');
    truncate2=compare('abcdef','abc',':');
    blank=compare('','abc',         ':');
    put pad1 pad2 truncate1 truncate2 blank;
  end;
enddata;
run;
quit;

proc print data=test2 noobs;run;quit;
```

アウトプット 7.5　コロン修飾子を使用して文字列を切り捨てた結果

pad1	pad2	truncate1	truncate2	blank
0	-4	0	0	-1

関連項目

関数

- “COMPCOST 関数” (435 ページ)
- “COMPGED 関数” (440 ページ)

COMPBL 関数

文字列内の単語間にある複数の空白を取り除きます。

カテゴリ: 文字

返されるデータの種類: CHAR、NCHAR、NVARCHAR、VARCHAR

構文

COMPBL(*character-expression*)

必須引数

character-expression
文字列を評価するか、文字列に強制変換できる有効な式であり、取り除く文字列を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

文字列中に 2 つ以上の空白が連続して出現するたびに、COMPBL 関数はそれらを 1 つの空白に変換して、複数の空白を削除します。

比較

COMPRESS 関数は、特定の文字が文字列に出現するたびにそれを削除します。ソース文字列から削除する文字として空白を指定した場合、COMPRESS 関数は COMPBL 関数と同様の動作をします。ただし、COMPRESS 関数は、ソース文字列からすべての空白を削除します。COMPBL 関数は、複数の空白を 1 つの空白に圧縮するため、1 つの空白には影響しません。

例

例 1: 文字列からの空白の削除

次のプログラムは、COMPBL 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl char(20) string1 string2;
  method run();
    string1='January      Status';
    string2=compbl(string1);
    put string2;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
January Status
```

例 2: 関数に渡される文字列からの空白の削除

次のプログラムは、COMPBL 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl char(18) string1 string2;
  method run();
    string1='125    E. Main St.';
    string2=compbl(string1);
    put string2;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
125 E. Main St.
```

関連項目

関数:

- “COMPRESS 関数” (452 ページ)

COMPCOST 関数

後から COMPGED 関数で使えるように演算のコストを設定します。

構文

COMPCOST(*operation-1*, *value-1* [..., *operation-n*, *value-n*]);

必須引数

operation
COMPGED 関数で実行する演算を指定する文字の定数、変数または式です。

value
先行する引数で示す演算のコストを指定する数値の定数、変数または式です。

制限事項 –32767 から 32767 の整数か、欠損値とする必要があります。

詳細

演算のコストの計算

演算を指定する各引数の文字列の値とする必要があります。文字列は、COMPGED 関数が実行する演算を表す語句に対応します。COMPGED 関数が使用する演算の表を確認するには、“一般化編集距離の計算” (442 ページ)を参照してください。

演算を指定する文字列には大文字、小文字、または大文字と小文字を混在させることができます。空白は無視されます。文字列はすべて等号(=)で終わる必要があります。次のテーブルに、操作の有効な値と操作のデフォルトのコストを示します。

演算	デフォルトコスト
APPEND=	very large
BLANK=	very large
DELETE=	100

演算	デフォルトコスト
DOUBLE=	very large
FDELETE=	equal to DELETE
FINSERT=	equal to INSERT
FREPLACE=	equal to REPLACE
INSERT=	100
MATCH=	0
PUNCTUATION=	very large
REPLACE=	100
SINGLE=	very large
SWAP=	very large
TRUNCATE=	very large

COMPCOST 関数で演算が出現しない場合、または演算に欠損値が続く場合、演算にはデフォルトのコストが割り当てられます。コストの"very large"は、COMPGED 関数が対応する演算を使わないほどに十分大きいコストを示します。

プログラムが COMPCOST 関数を使用した後、指定コストはプログラムが COMPCOST 関数を再度使用するまで、または COMPCOST 関数を含むデータプログラムが終了するまで有効です。

注: FCMP 関数が COMPGED 関数を呼び出す場合、COMPGED は、COMPGED 処理の前に処理される FCMP 関数内の COMPCOST 呼び出しの影響を受けます。COMPCOST が FCMP 関数の外部で呼び出された場合、FCMP 関数の内部にある COMPGED 呼び出しは、FCMP 関数の外部にある COMPCOST 呼び出しの影響を受けません。

文字列の短縮

文字列は短縮できます。特定の演算の最初の 1 文字または数文字を語全体のかわりに使えます。ただし、語を一意に区別するのに十分な数の文字を使用します。たとえば、INSERT=演算を"in="と指定し、REPLACE=演算を"r="と指定できます。DELETE=演算または DOUBLE=演算を指定するには、DELETE=および DOUBLE=の両者が"d"で始まるため、最初の 2 文字を使う必要があります。文字列は常に等号(=)で終わる必要があります。

例

この例は、COMPCOST ルーチンを呼び出して、指定した演算の一般化編集距離を計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data testdata /overwrite=yes;
  dcl char string operation;
  method init();
    string='baboon'; operation='match'; output;
    string='xbaboon'; operation='insert'; output;
    string='babon'; operation='delete'; output;
    string='baXoon'; operation='replace'; output;
  end;
enddata;
run;

data mytest /overwrite=yes;
  dcl double ged;
  method run();
    set testdata;
    if _n_ =1 then compcost('insert=', 10, 'DEL=', 11, 'r=', 12);
    ged=compged(string, 'baboon');
    put 'string=' string 'operation=' operation 'ged=' ged;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
string= baboon   operation= match    ged= 0
string= xbaboon  operation= insert   ged= 10
string= babon    operation= delete   ged= 11
string= baXoon   operation= replace  ged= 12
```

関連項目

関数:

- “COMPARE 関数” (429 ページ)
- “COMPGED 関数” (440 ページ)
- “COMPLEV 関数” (447 ページ)

COMPFUZZ 関数

2 つの数値をファジー比較します。

カテゴリ: 数学

返されるデータの種類: DOUBLE

構文

COMPFUZZ(*expression-1*, *expression-2*[, *fuzz*[, *scale*]])

必須引数

expression-1
数値に評価される有効な式を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

expression-2
数値に評価される有効な式を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

オプション引数

fuzz
比較の相対しきい値を指定する負でない数値です。1 以上の値はマシン精度の倍数単位で扱われます。

デフォルト 1024

データの種類 DOUBLE

scale
スケール係数を指定します。

デフォルト MAX (ABS (*expression-1*), ABS (*expression-2*))

データの種類 DOUBLE

詳細

COMPFUZZ 関数は、4 つの引数をすべて 指定した場合、次の値を返します。

- -1 (*expression-1* < *expression-2* – threshold の場合)
- 0 (ABS(*expression-1* - *expression-2*) ≤ threshold の場合)
- 1 (*expression-1* > *expression-2* + threshold の場合)

次の関係が存在します。

- 0 ≤ *fuzz* < 1 のとき、threshold = *fuzz* * ABS(*scale*)
- 1 ≤ *fuzz* < 1 / CONSTANT('MACEPS')のとき、threshold = *fuzz* * ABS(*scale*) * CONSTANT('MACEPS')

COMPFUZZ は浮動小数点のアンダーフローまたはオーバーフローを 回避します。

比較

COMPFUZZ 関数は 2 つの浮動小数点の数値を比較し、比較に基づく値を返します。ROUND 関数は、第 2 引数の倍数にきわめて近い値に引数を丸めます。結果は第 2 引数の正確な倍数でない場合があります。

例

浮動小数点計算では、数値を加算する順番によって合計値が異なる場合があります。x1 から x*n* までの *n* 個の数値の和を計算するときの、ある浮動小数点エラーの近似境界は次の式で表されます。

n * machine_precision * sum (abs(x1) + ... + abs(x*n*))

したがって、*n* 個の浮動小数点の数値の和を COMPFUZZ 関数で比較するのに、次の DATA ステップに示すように、*n* をファジー値、絶対値の和をスケール係数として使用できます。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double x1 x2 x3 x4 sum1 sum2 diff compfuzz1 compfuzz2 scale;
  method run();
    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=;
    put sum2=;
    put 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.                                                  */
    compfuzz1 = compfuzz(sum1, sum2, 8);
    put 'fuzz only (wrong):       ' compfuzz1=;
    /* Using a fuzz factor and a scale value gives the correct result. */
    scale = abs(x1) + abs(x2) + abs(x3) + abs(x4);
    compfuzz2 = compfuzz(sum1, sum2, 8, scale);
    put 'fuzz and scale (correct): ' compfuzz2=;
  end;
enddata;
run;
quit;
```

次の行が SAS ログに書き込まれます。

```
sum1=1.80952382087707
sum2=1.8095238095238
diff=1.1353265660929E-8
fuzz only (wrong):       compfuzz1=1
fuzz and scale (correct):  compfuzz2=0
```

関連項目

関数:

- “FUZZ 関数” (662 ページ)
- “ROUND 関数” (1088 ページ)

COMPGED 関数

2 つの文字列間の一般化編集距離を返します。

カテゴリ:	文字
返されるデータの種類:	DOUBLE

構文

COMPGED(*string-1*, *string-2* [, *cutoff*] [, *modifier(s)*])

必須引数

string-1
文字定数、変数または式を指定します。

string-2
文字定数、変数または式を指定します。

cutoff
数値定数、変数または式です。

注 実際の一般化編集距離が *cutoff* の値よりも大きい場合、*cutoff* と等しい値が返されます。

ヒント *cutoff* に小さい値を使用すると、*string–1* および *string–2* の値の長さが長い場合に COMPGED の効率が向上します。

オプション引数

modifier
COMPGED 関数のアクションを変更できる文字列を指定します。

修飾子では大文字と小文字は区別されません。次の 1 つ以上の文字を有効な修飾子として使用できます。

i
string-1 および *string-2* の大文字と小文字を区別しません。

l
値を比較する前に *string-1* および *string-2* の先頭の空白を削除します。

n
n-literal であるすべての引数から引用符を削除し、*string-1* および *string-2* の大文字と小文字を区別をしません。

:
(コロン) *string-1* と *string-2* の短い方の文字列の長さまたは 1 文字(どちらか大きい方)になるように長い方の文字列を切り捨てます。

> ヒント COMPGED は修飾子に使用される空白を無視します。

引数

詳細

修飾子の出現順序

COMPGED 関数では、修飾子の出現順序に意味があります。

- "LN"は各文字列から先頭の空白を削除した後、n-literal から引用符を削除します。

- "NL"は n-literal から引用符を削除した後、各文字列から先頭の空白を削除します。

一般化編集距離の定義

一般化編集距離は Levenshtein の編集距離を一般化し、2 つの文字列間の相違を測定したものです。Levenshtein の編集距離は、*string-1* を *string-2* に変換するために必要な単独文字の削除、挿入または置換の回数です。

一般化編集距離の計算

COMPGED 関数は *string-1* および *string-2* の間の一般化編集距離を返します。一般化編集距離は、*string-2* から *string-1* を作るためのコストが最小限の一連の演算です。

コストの和を計算するアルゴリズムは、*string-2*(入力文字列)内の文字を指すポインターを使用します。ポインターを進める、1 文字以上の文字を出力文字列に追加する、またはその両方を行う一連の演算によって出力文字列を作ります。開始時は、ポインターは入力文字列の最初の文字を指しており、出力文字列は空です。

演算とそれぞれのコストを次の表に示します。

演算	デフォルトコスト単位	演算の内容
APPEND	50	出力文字列が入力文字列より長いとき、ポインターを移動せずに 1 文字を出力文字列の末尾に追加します。
BLANK	10	次のアクションのいずれかを実行します。 ■ ポインターを移動せずに出力文字列の末尾に空白文字を 1 つ追加します。 ■ ポインター位置の文字がスペース文字のとき、出力文字列を変更せずにポインターの位置を 1 つ進めます。 ■ ポインター位置の文字がスペース文字のとき、出力文字列の末尾にスペース文字を 1 文字追加し、ポインターの位置を 1 つ進めます。 COMPCOST 関数で BLANK のコストが 0 に設定されている場合、COMPGED 関数は比較を行う前に、両方の文字列からすべてのスペース文字を削除します。

演算	デフォルトコスト単位	演算の内容
DELETE	100	出力文字列を変更せずにポインターの位置を 1 つ進めます。
DOUBLE	20	ポインターを移動せずに出力文字列の末尾に文字を追加します。
FDELETE	200	出力文字列が空のとき、出力文字を変更せずにポインターの位置を 1 つ進めます。
FINSERT	200	ポインターが第 1 位置にあるとき、ポインターを移動せずに 1 文字を出力文字列の末尾に追加します。
FREPLACE	200	ポインターが第 1 位置にあり、出力文字列が空のとき、出力文字列の末尾に 1 文字を追加し、ポインターの位置を 1 つ進めます。
INSERT	100	ポインターを移動せずに出力文字列の末尾に 1 文字を追加します。
MATCH	0	入力文字列のポインター位置の文字をコピーして出力文字列の末尾に追加し、ポインターの位置を 1 つ進めます。
PUNCTUATION	30	次のアクションのいずれかを実行します。 ■ ポインターを移動せずに出力文字列の末尾に句読空白文字を 1 つ追加します。 ■ ポインター位置の文字が句読文字のとき、出力文字列を変更せずにポインターの位置を 1 つ進めます。 ■ ポインター位置の文字が句読文字のとき、出力文字列の末尾に句読文字を 1 文字追加し、ポインターの位置を 1 つ進めます。 COMPCOST 関数で PUNCTUATION のコストが 0 に設定されている場合、COMPGED 関数は比較を行う前に、両方の文字列からすべての 句読文字を削除します。

演算	デフォルトコスト単位	演算の内容
REPLACE	100	出力文字列の末尾に 1 文字を追加し、ポインターの位置を 1 つ進めます。
SINGLE	20	ポインター位置の文字が入力文字列の続く文字と同じとき、出力文字列を変更せずにポインターの位置を 1 つ進めます。
SWAP	20	入力文字列からポインターに続く文字をコピーし、出力文字列に追加します。次に、入力文字列からポインター位置の文字をコピーし、出力文字列に追加します。ポインター位置を 2 つ進めます。
TRUNCATE	10	出力文字列が入力文字列より短いとき、出力文字を変更せずにポインターの位置を 1 つ進めます。

文字列演算のコストを設定するには、COMPCOST 関数を使用するか、デフォルトコストを使用できます。デフォルトのコストを使用する場合、COMPGED から返される値は、COMPLEV から返される値のおよそ 100 倍の大きさです。

エラー例

一般化編集距離を求める原理は、タイプミスによって起こる可能性があるエラーの数と種類に基づいています。COMPGED はそれぞれのエラーにコストを割り当て、発生する可能性のあるコストの最小和を求めます。エラーの中には、他のエラーよりも重大であるものもあります。たとえば、文字列の先頭に余分な文字が入るのは、文字列の末尾の文字が除外されることよりも重大である可能性があります。もう 1 つの例は、*string-2* に含まれる語句を入力するときにタイプミスをして、*string-2* ではなく *string-1* を生成してしまう場合があります。

一般化編集距離を対称にする

一般化編集距離は必ずしも対称ではありません。言い換えれば、COMPGED(string1, string2)で返される値は必ずしも COMPGED(string2, string1)で返される値とは同じではありません。一般化編集距離を対称にするには、COMPCOST 関数を使って、次のペアにおける演算に同等のコストを割り当てます。

- INSERT, DELETE
- FINSERT, FDELETE
- APPEND, TRUNCATE
- DOUBLE, SINGLE

比較

Levenshtein の編集距離は、COMPLEV 関数を使用して計算できます。一般化編集距離は、COMPCOST 関数および COMPGED 関数を使用して計算できます。一般化編集距離の計算は、Levenshtein の編集距離の計算よりもかなり長い時間がかかります。ただし、ファジーファイルマージやテキストマイニングなど、一般化編集距離では Levenshtein の編集距離よりも、通常は有用な測定が可能です。

注: 次の FCMP 関数は、DS2 プログラムから呼び出された場合、DS2 プログラムで設定されたシード値を尊重しません。

COMPCOST
COMPGED
RAND
STREAMINIT

DATA ステップでは、RAND からのすべての乱数はデフォルトで同じストリームから取得されますが、CALL STREAM ルーチンを使用して複数の独立したストリームを作成できます。

PROC DS2 では、予期しない機能の結果として、FCMP パッケージ内で呼び出される RAND の乱数は、別のストリームから取得されます。再現可能な乱数を取得するには、RAND を呼び出す各 FCMP 関数内で CALL STREAMINIT ルーチンを使用します。独立したストリームを取得するには、RAND を呼び出す各 FCMP 関数内で CALL STREAM ルーチンも使用します。そうすると、DS2 からの結果は DATA ステップおよび将来のリリースと一致します。例については、“FCMP パッケージを使用する際の考慮事項と制限事項” (*SAS DS2 プログラマガイド*)を参照してください。

例

次の例では、デフォルトコストを使用して一般化編集距離を計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
/**** input data ****/
data testdata(overwrite=yes);
  dcl char string1 string2;
  dcl char(30) operation;
  method init();
    string1='baboon';   string2='baboon';   operation='match'; output;
    string1='baXboon';  string2='baboon';   operation='insert'; output;
    string1='baoon';    string2='baboon';   operation='delete'; output;
    string1='baXoon';   string2='baboon';   operation='replace'; output;
    string1='baboonX';  string2='baboon';   operation='append'; output;
    string1='baboo';    string2='baboon';   operation='truncate'; output;
    string1='babboon';  string2='baboon';   operation='double'; output;
    string1='babon';    string2='baboon';   operation='single'; output;
```

```
    string1='baobon';   string2='baboon';   operation='swap'; output;
    string1='bab oon';  string2='baboon';   operation='blank'; output;
    string1='bab,oon';  string2='baboon';   operation='punctuation';
      output;
    string1='bXaoon';   string2='baboon';   operation='insert+delete';
      output;
    string1='bXaYoon';  string2='baboon';   operation='insert+replace';
      output;
    string1='bXoon';    string2='baboon';   operation='delete+replace';
      output;
    string1='Xbaboon';  string2='baboon';   operation='finsert';
      output;
    string1='aboon';    string2='baboon';
      operation='trick question: swap+delete'; output;
    string1='Xaboon';   string2='baboon';   operation='freplace'; output;
    string1='axoon';    string2='baboon';
      operation='fdelete+replace'; output;
    string1='axoo';     string2='baboon';
      operation='fdelete+replace+truncate'; output;
    string1='axon';     string2='baboon';
      operation='fdelete+replace+single'; output;
    string1='baby';     string2='baboon';
      operation='replace+truncate*2'; output;
    string1='balloon';  string2='baboon';
      operation='replace+insert'; output;
  end;
enddata;
run;

/**** comparison code *****/
data mytest (overwrite=yes) noobs;
  dcl double ged;
  method run();
    set testdata;
    ged=compged(string1, string2);
    put 'string1=' string1 'string2=' string2 'operation='
       operation 'ged=' ged;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
string1= baboon   string2= baboon   operation= match                  ged= 0
string1= baXboon  string2= baboon   operation= insert                 ged= 100
string1= baoon    string2= baboon   operation= delete                 ged= 100
string1= baXoon   string2= baboon   operation= replace                ged= 100
string1= baboonX  string2= baboon   operation= append                 ged= 50
string1= baboo    string2= baboon   operation= truncate               ged= 10
string1= babboon  string2= baboon   operation= double                 ged= 20
string1= babon    string2= baboon   operation= single                 ged= 20
string1= baobon   string2= baboon   operation= swap                   ged= 20
string1= bab oon  string2= baboon   operation= blank                  ged= 10
string1= bab,oon  string2= baboon   operation= punctuation            ged= 30
string1= bXaoon   string2= baboon   operation= insert+delete          ged= 200
string1= bXaYoon  string2= baboon   operation= insert+replace         ged= 200
string1= bXoon    string2= baboon   operation= delete+replace         ged= 200
string1= Xbaboon  string2= baboon   operation= finsert                ged= 200
string1= aboon    string2= baboon   operation= trick question: swap+delete   ged= 120
string1= Xaboon   string2= baboon   operation= freplace               ged= 200
string1= axoon    string2= baboon   operation= fdelete+replace        ged= 300
string1= axoo     string2= baboon   operation= fdelete+replace+truncate   ged= 310
string1= axon     string2= baboon   operation= fdelete+replace+single     ged= 320
string1= baby     string2= baboon   operation= replace+truncate*2       ged= 120
string1= balloon  string2= baboon   operation= replace+insert          ged= 200
```

関連項目

関数:

- “COMPARE 関数” (429 ページ)
- “COMPCOST 関数” (435 ページ)
- “COMPLEV 関数” (447 ページ)

COMPLEV 関数

2 つの文字列間の Levenshtein の編集距離を返します。

カテゴリ:	文字
返されるデータの種類:	DOUBLE

構文

COMPLEV(*string-1*, *string-2* [, *cutoff*] [, *modifier(s)*])

必須引数

string-1
文字定数、変数または式を指定します。

string-2
文字定数、変数または式を指定します。

オプション引数

cutoff
数値の定数、変数または式を指定します。

注 実際の Levenshtein の編集距離が *cutoff* の値よりも大きい場合、*cutoff* と等しい値が返されます。

ヒント *cutoff* に小さい値を使用すると、*string–1* および *string–2* の値の長さが長い場合に COMPLEV の効率が向上します。

modifier
COMPLEV 関数のアクションを変更できる文字列を指定します。修飾子では大文字と小文字は区別されません。

次の 1 つ以上の文字を有効な修飾子として使用できます。

i
string-1 および *string-2* の大文字と小文字を区別しません。

l
値を比較する前に *string-1* および *string-2* の先頭の空白を削除します。

n
n-literal であるすべての引数から引用符を削除し、*string-1* および *string-2* の大文字と小文字を区別をしません。

:
(コロン) *string-1* と *string-2* の短い方の文字列の長さまたは 1 文字(どちらか大きい方)になるように長い方の文字列を切り捨てます。

> ヒント COMPLEV は修飾子として使用される空白を無視します。

詳細

COMPLEV 関数では、修飾子の出現順序に意味があります。

- "LN"は各文字列から先頭の空白を削除した後、n-literal から引用符を削除します。
- "NL"は n-literal から引用符を削除した後、各文字列から先頭の空白を削除します。

COMPLEV 関数は末尾の空白を無視します。

COMPLEV 関数は *string-1* および *string-2* の間の Levenshtein の編集距離を返します。Levenshtein の編集距離は、ある文字列を他の文字列に変換するために必要な単独文字の挿入、削除または置換の回数です。Levenshtein の編集距離は対称的です。つまり、COMPLEV(string-1,string-2)は COMPLEV(string-2,string-1)と同じです。

比較

COMPLEV で計算される Levenshtein の編集距離は、COMPGED で計算される一般化編集距離の特別なケースです。

COMPLEV の実行にかかる時間は、COMPGED よりも大幅に短く済みます。

例

この例では、Levenshtein の編集距離を計算して 2 つの文字列を比較します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
/**** input data ****/
proc ds2;
data testdata (overwrite=yes);
  dcl char string1 string2 modifier;
  method init();
    string1='12345678'; string2='12345678'; output;
    string1='abc'; string2='abxc'; output;
    string1='ac'; string2='abc'; output;
    string1='aXc'; string2='abc'; output;
    string1='aXbZc'; string2='abc'; output;
    string1='aXYZc'; string2='abc'; output;
    string1='WaXbYcZ'; string2='abc'; output;
    string1='XYZ'; string2='abcdef'; output;
    string1='aBc'; string2='abc'; output;
    string1='aBc'; string2='AbC'; modifier='i'; output;
    string1='  abc'); string2='abc'; modifier=''; output;
    string1='  abc'); string2='abc';  modifier='l'; output;
    string1='AxC'; string2=n'abc'; modifier=''; output;
    string1='AxC'; string2=n'abc'; modifier='n'; output;
  end;
enddata;
run;
quit;

/**** comparison code *****/
proc ds2;
data test (overwrite=yes);
  dcl double result;
  method run();
    set testdata;
    result=complev(string1, string2, modifier);
    put 'string1=' string1 'string2=' string2 'modifier=' modifier
      'result=' result;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
string1= 12345678 string2= 12345678 modifier=        result= 0
string1= abc      string2= abxc     modifier=        result= 1
string1= ac       string2= abc      modifier=        result= 1
string1= aXc      string2= abc      modifier=        result= 1
string1= aXbZc    string2= abc      modifier=        result= 2
string1= aXYZc    string2= abc      modifier=        result= 3
string1= WaXbYcZ  string2= abc      modifier=        result= 4
string1= XYZ      string2= abcdef   modifier=        result= 6
string1= aBc      string2= abc      modifier=        result= 1
string1= aBc      string2= AbC      modifier= i      result= 0
string1=   abc    string2= abc      modifier=        result= 2
string1=   abc    string2= abc      modifier= l      result= 0
string1= AxC      string2= abc      modifier=        result= 3
string1= AxC      string2= abc      modifier= n      result= 1
```

関連項目

関数:

- “COMPARE 関数” (429 ページ)
- “COMPCOST 関数” (435 ページ)
- “COMPGED 関数” (440 ページ)

COMPOUND 関数

複利パラメーターを返します。

カテゴリ:	財務
返されるデータの種類:	DOUBLE

構文

COMPOUND(*a*, *f*, *r*, *n*)

必須引数

a
初期の金額を指定します。

範囲	$a \geq 0$
データの種類	DOUBLE

f
将来の金額を指定します(*n* 期間の終了時)。

範囲 $f \geq 0$

データの種類 DOUBLE

r
期間単位の金利を分数で指定します。

範囲 $r \geq 0$

データの種類 DOUBLE

n
複利期間数を指定します。

範囲 $n \geq 0$

データの種類 DOUBLE

詳細

COMPOUND 関数は、複利計算の 4 つの引数のリストの欠損引数を返します。これらの引数には次の式の関係があります。

$$f = a(1 + r)^n$$

引数は 1 つを欠損値とする必要があります。他の 3 つの値から複利パラメーターが計算されます。結果を変換して丸めた数字にする調整は行われません。

n=0 のとき、

$$f = a$$

および

$$(1 + r)^n$$

は 1 です。

注: *r* を欠損値に選ぶと COMPOUND からエラーが返されます。

例

このプログラムを使用して、$2000 の投資で名目年利 9%、30 か月複利運用したときの累計額を計算できます。第 2 引数が欠損値となっているため、将来の金額が計算されます。名目年利 9%は、月利 0.09/12 に変換されています。利率の引数は、複利計算の期間当たりの(パーセンテージではなく)分数で表される利率です。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double future;
  method run();
    future=compound(2000, ., 0.09/12, 30);
    put future;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
2502.54352764668
```

COMPRESS 関数

元の文字から指定した文字を削除した文字列を返します。

カテゴリ: 文字

返されるデータの種類: CHAR、NCHAR、NVARCHAR、VARCHAR

構文

COMPRESS(*character-expression*[, *character-list-expression*] [, *modifier(s)*])

必須引数

character-expression
文字式に評価され、指定された文字が削除される有効な式を指定します。

要件 文字のリテラル文字列は単一引用符で囲みます。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

オプション引数

character-list-expression
文字リストを初期化する変数または任意の有効な式を指定します。

デフォルトでは、このリスト内の文字が *character-expression* から削除されます。

要件 文字のリテラル文字列は単一引用符で囲みます。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

modifier
文字定数、変数または式を指定します。空白でない文字はそれぞれ COMPRESS 関数のアクションを変更します。空白は無視されます。修飾子では大文字と小文字は区別されません。

修飾子として使用できる文字は次のとおりです。

A
文字のリストにアルファベット文字を追加します。

C
文字のリストに制御文字を追加します。

D
文字のリストに数字を追加します。

F
アンダースコア文字および英文字を文字リストに追加します。

G
文字のリストにグラフィカル文字を追加します。グラフィカル文字は、現在のロケールの文字で、紙面に印刷するとイメージを生成します。

L
文字のリストに水平タブを追加します。

I
保持または削除する文字の大文字と小文字を区別しません。

K
リスト内の文字を削除するのではなく、保持します。

I
小文字を文字リストに追加します。

N
数字、アンダースコア文字および英文字を文字リストに追加します。

O
COMPRESS 関数が呼び出されるたびに第 2 引数および第 3 引数を実行するのではなく、一度だけ実行します。DATA ステップ(WHERE 句を除く)または SQL プロシジャで O 修飾子を使用すると、第 2 引数および第 3 引数が変更されないループで COMPRESS を呼び出すときに、より迅速に実行できます。

P
文字のリストに句読点を追加します。

S
文字のリストに空白文字(空白、水平タブ、垂直タブ、キャリッジリターン、ラインフィード、フォームフィード、NBSP ('A0'x - 10 進数の ASCII コードで 160))を追加します。

T
第 1 引数と第 2 引数から末尾の空白を取り除きます。

U
大文字を文字リストに追加します。

W
印刷可能文字を文字リストに追加します。

X
文字のリストに 16 進文字を追加します。

ヒント　修飾子が定数の場合、単一引用符で囲みます。一組の引用符で複数の定数を指定します。修飾子を変数または式として表すこともできます。

詳細

COMPRESS 関数は、第 2 引数内の文字に加えて、修飾子で指定される文字を含む文字を保持または削除する文字リストをコンパイルします。

引数の数に基づいて、COMPRESS 関数は次のように機能します。

引数の数	結果
第 1 引数のみ。*source*	すべての空白が削除されます。引数のすべてが空白の場合、結果は長さがゼロの文字列となります。固定長の文字変数に結果を割り当てると、その変数の値は定義した長さとなるよう空白が埋め込まれます。
2 つの引数。*source* および *chars*	第 2 引数に現れるすべての文字が結果から削除されます。
3 つの引数。*source*、*chars* および *modifier* (または *modifiers*)	K 修飾子(第 3 引数に指定)は、第 2 引数内の文字を結果に保持するか、削除するかを決定します。

これは、第 3 引数の効果を示す例です。D 修飾子は数字を指定します。次の関数呼び出しは、どちらも結果から数字を削除します。

```
compress(source, "1234567890");
compress(source, '', "d");
```

数字、プラス記号またはマイナス記号を削除するには、次の関数呼び出しを使用します。

```
COMPRESS(source, "1234567890+-");
compress(source, "+-", "d");
```

COMPRESS 関数は、第 2 引数で null または欠損(".")引数を許可します。null または欠損している引数は長さがゼロの文字列として扱われます。長さが 0 の文字列は空の文字列です。

```
compress(source, null);
compress(source, '');
```

例

例 1: 空白の取り除き

このプログラムは、文字列から空白を削除する方法を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl char a b;
  method run();
    a='AB C D ';
    b=compress(a);
    put b;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
ABCD
```

例 2: 大文字の取り除き

このプログラムは、文字列から大文字を削除する方法を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl char a b;
  method run();
    a='AB C D';
    b=compress(a, 'A');
    put b;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
B C D
```

例 3: 文字列を取り除いて長さ 0 を返す

このプログラムは、文字列から大文字を削除する方法を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl char(10) x;
  dcl double l;
  method run();
    x=' ';
    l=length(compress(x));
    put l=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
l=0
```

例 4: 母音の取り除き

この例は、文字列から母音を削除する方法を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl char(32) x;
  dcl char(32) y;
  method run();
    x='123-4567-8901 e 234-5678-9012 i';
    y=compress(x,'aeiou');
    put y;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
123-4567-8901  234-5678-9012
```

例 5: 修飾子を使用した小文字の取り除き

この例では、削除する文字のリストと、修飾子"l"で指定された小文字を使用して大文字を取り除きます。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl char(32) x y;
  method run();
    x='123-4567-8901 B 234-5678-9012 c';
    y=compress(x, 'ABCD', 'l');
    put y=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
y=123-4567-8901  234-5678-9012
```

例 6: 修飾子を使用したスペース文字の取り除き

この例は、修飾子"s"を使用して、指定された文字列からスペースを削除する方法を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl char(10) x y;
  method run();
    x='1 2 3 4 5';
    y=compress(x, '', 's');
    put y=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
y=12345
```

例 7: 修飾子を使用して文字をリストに保持する

この例は、修飾子"k"を使用して、指定された文字列に文字を保持する方法を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl char(26) x y;
  method run();
    x='Math A English B Physics A';
    y=compress(x, 'ABCD', 'k');
    put y=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
y=ABA
```

関連項目

関数:

- “COMPBL 関数” (433 ページ)
- “LEFT 関数” (806 ページ)
- “TRIM 関数” (1217 ページ)

CONSTANT 関数

マシン定数および数学定数を計算します。

カテゴリ:	数学
返されるデータの種類:	DOUBLE

構文

CONSTANT(*constant*[, *parameter*])

必須引数

constant
返す定数を特定する文字定数、変数または式を指定します。

有効な定数は次のとおりです。

定数	説明
'E'	自然数の底
'EULER'	Euler 定数
'PI'	PI
'EXACTINT' [,nbytes]	正確な整数
'BIG'	最大倍精度浮動小数点数
'LOGBIG' [,base]	*base* を底とする BIG の対数
'SQRTBIG'	BIG の平方根
'SMALL'	最小倍精度浮動小数点数
'LOGSMALL' [,base]	*base* を底とする SMALL の対数
'SQRTSMALL'	SMALL の平方根
'MACEPS'	マシン精度の定数
'LOGMACEPS' [,base]	*base* を底とする MACEPS の対数
'SQRTMACEPS'	MACEPS の平方根

オプション引数

parameter
数値パラメーターで、*constant* で指定された一部の定数と一緒にオプション引数として使用できます。この *parameter* を使用すると、CONSTANT 関数の機能が変更されます。

詳細

概要

注意
一部の動作環境では、ランタイムライブラリに制限があるために、ハードウェアが持つ浮動小数点数を最大限に活用できません。 そのような場合、CONSTANT 関数はラインタイムライブラリの制限内で使用できる値を返そうとします。たとえば、ランタイムライブラリで EXP(LOG(CONSTANT('BIG')))を計算できない場合は、CONSTANT('LOGBIG')

で LOG(CONSTANT('BIG'))と同じ値が返されず、EXP(CONSTANT('LOGBIG'))を計算できる値が返されます。

自然数の底

CONSTANT('E')

自然数の底は次の式で表されます。

$$\lim_{x \to 0} (1+x)^{\frac{1}{x}} \approx 2.718281828459045$$

Euler 定数

CONSTANT('EULER')

Euler の定数は次の式で表されます。

$$\lim_{n \to \infty} \left\{ \sum_{j=1}^{j=n} \frac{1}{j} - \log(n) \right\} \approx 0.577215664901532860$$

PI

CONSTANT('PI')

PI(円周率)は、円の周の長さと直径の比です。この定数を計算する式は多数あります。この数列を表す式の 1 例は次のとおりです。

$$4 \sum_{j=0}^{j=\infty} \frac{(-1)^j}{2j+1} \approx 3.14159265358979323846$$

正確な整数

CONSTANT('EXACTINT'[, *nbytes*])

引数

nbytes
バイト数を表す数値です。

デフォルト 8

範囲 $2 \le nbytes \le 8$

正確な整数は、絶対値で *k* 以下のすべての整数を、長さが *nbytes* の SAS 数値変数で正確に表現できる最大整数 *k* です。この情報は、容量を節約するために SAS 数値変数をデフォルトの 8 バイトから小さいバイト数に調整するときにあらかじめ知っておくと役に立つ情報です。

最大倍精度浮動小数点数

CONSTANT('BIG')

使用しているコンピューターで表現できる最大倍精度浮動小数点数(8 バイト)を返します。

BIG の対数

CONSTANT('LOGBIG'[, *base*])

引数

base
対数の底となる数値です。

デフォルト　自然数の底、E。

制限事項　指定する *base* は 1+SQRTMACEPS の値より大きくする必要があります。

base を底として、使用しているコンピューターで表現できる最大倍精度浮動小数点数(8 バイト)の対数を返します。

CONSTANT('LOGBIG', *base*)以下に累乗された指定の *base* は、累乗演算(**)を使っても、オーバーフローを発生させず安全に累乗できます。

浮動小数点数は、CONSTANT('LOGBIG')以下の任意の浮動小数点数で累乗関数 EXP を使っても、オーバーフローを発生させず安全に累乗できます。

BIG の平方根

CONSTANT('SQRTBIG')

使用しているコンピューターで表現できる最大倍精度浮動小数点数(8 バイト)の平方根を返します。

浮動小数点数は、CONSTANT('SQRTBIG')以下の任意の浮動小数点数で、オーバーフローを発生させず安全に平方根を求めることができます。

最小倍精度浮動小数点数

CONSTANT('SMALL')

使用しているコンピューターで表現できる最小倍精度浮動小数点数(8 バイト)を返します。

SMALL の対数

CONSTANT('LOGSMALL'[, *base*])

引数

base
対数の底となる数値です。

デフォルト　自然数の底、E。

制限事項　指定する *base* は 1+SQRTMACEPS の値より大きくする必要があります。

base を底として、使用しているコンピューターで表現できる最小倍精度浮動小数点数(8 バイト)の対数を返します。

CONSTANT('LOGSMALL', *base*)以上に累乗された指定の *base* は、累乗演算(**)を使って安全に累乗できます。アンダーフローまたは 0 にはなりません。

浮動小数点数は、CONSTANT('LOGSMALL')以上の任意の浮動小数点数で累乗関数 EXP を使って安全に累乗できます。アンダーフローまたは 0 にはなりません。

SMALL の平方根

CONSTANT('SQRTSMALL')

コンピューターで表現できる最小倍精度浮動小数点数(8 バイト)の平方根を返します。

浮動小数点数は、CONSTANT('SQRTBIG')以上の任意の浮動小数点数で、安全に平方根を求めることができます。アンダーフローまたはゼロにはなりません。

マシンの精度

CONSTANT('MACEPS')

この場合、$1+\varepsilon > 1$のように、ある整数 *j* に対して最小倍精度浮動小数点数(8 バイト) $\varepsilon = 2^{-j}$が返されます。

この定数は有限精度の計算を行うときに重要です。

MACEPS の対数

CONSTANT('LOGMACEPS'[, *base*])

引数

base
対数の底となる数値です。

デフォルト　自然数の底、E。

制限事項　指定する *base* は 1+SQRTMACEPS の値より大きくする必要があります。

base を底とする CONSTANT('MACEPS')の対数を返します。

MACEPS の平方根

CONSTANT('SQRTMACEPS')

CONSTANT('MACEPS')の平方根を返します。

例

次のプログラムでは、CONSTANT 関数を使用して、さまざまな定数の値を返します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test/overwrite=yes;
 dcl double a b c d e f g h i j k l m;
  method run();
    a=constant('E');
    b=constant('EULER');
    c=constant('PI');
    d=constant('EXACTINT');
    e=constant('BIG');
    f=constant('LOGBIG');
    g=constant('SQRTBIG');
    h=constant('SMALL');
    i=constant('LOGSMALL');
    j=constant('SQRTSMALL');
    k=constant('MACEPS');
    l=constant('LOGMACEPS');
    m=constant('SQRTMACEPS');
    put  'a= ' a;
    put 'b= ' b;
    put 'c= ' c;
    put 'd= ' d;
    put 'e= ' e;
    put 'f= ' f;
    put 'g= ' g;
    put 'h= ' h;
    put 'i= ' i;
    put 'j= ' j;
    put 'k= ' k;
    put 'l= ' l;
    put 'm= ' m;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
a=  2.71828182845904
b=  0.57721566490153
c=  3.14159265358979
d=  9007199254740992
e=  1.7976931348623E308
f=  709.782712893383
g=  1.3407807929942E154
h=  2.2250738585072E-308
i=  -708.396418532264
j=  1.49166814624E-154
k=  2.2204460492503E-16
l=  -36.0436533891171
m=  1.4901161193847E-8
```

CONVX 関数

列挙キャッシュフローのコンベクシティを返します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

CONVX(*y*, *f*, *c(1)*, ..., *c(k)*)

必須引数

y
期間当たりの有効満期利回りを指定します。

範囲 $0 < y < 1$

データの種類 DOUBLE

ヒント *y* を分数で表した場合、配当は小数値として記述する必要があります。DS2 では、整数除算の結果はゼロになります。ゼロは DOUBLE に変換され、CONVX 関数の最初の引数として渡されます。CONVX 関数は、最初のパラメーターとしてゼロを渡すと欠損値を返します。

f
期間当たりのキャッシュフローの頻度を指定します。

範囲 $f > 0$

データの種類 DOUBLE

c(1), ... , c(k)
キャッシュフローのリストを指定します。

データの種類 DOUBLE

詳細

CONVX 関数は、次の式から値を返します。

$$C = \sum_{k=1}^{K} \frac{k(k+f)\dfrac{c(k)}{(1+y)^{\frac{k}{f}}}}{P\left((1+y)^2\right)f^2}$$

前述の式には次の関係が適用されます。

$$P = \sum_{k=1}^{K} \frac{c(k)}{(1+y)^{\frac{k}{f}}}$$

例: CONVX 関数の使用

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test /overwrite=yes;
  dcl double c;
  method run();
    c=convx(1./20,.1,.33,.44,.55,.49,.50,.22,.4,.8,.01,.36,.2,.4);
    put c;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
976.941120481246
```

関連項目

関数:

- “CONVXP 関数” (465 ページ)

CONVXP 関数

債権などの定期キャッシュフローストリームのコンベクシティを返します。

カテゴリ:	財務
返されるデータの種類:	DOUBLE

構文

CONVXP(*A*, *c*, *n*, *K*, k_0, *y*)

必須引数

A
額面価格を指定します。

範囲 $A > 0$

A > 0

データの種類 DOUBLE

c
期間当たりの名目クーポンレートを小数で指定します。

範囲 0 ≤ *c* < 1

データの種類 DOUBLE

n
期間当たりのクーポン数を指定します。

範囲 *n* > 0

データの種類 DOUBLE

K
クーポンの残数を指定します。

範囲 *K* > 0

データの種類 DOUBLE

k_0
現在の日付から最初のクーポン日までの時間を指定します。期間数で表します。

範囲 $0 < k_0 \leq \frac{1}{n}$

$0 < k_0 \leq 1/n$

データの種類 DOUBLE

y
期間当たりの名目満期利回りを指定します。

範囲 *y* > 0

データの種類 DOUBLE

詳細

CONVXP 関数は、次の式から値を返します。

$$C = \frac{1}{n^2}\left(\frac{\sum_{k=1}^{K} t_k(t_k+1)\frac{c(k)}{\left(1+\frac{y}{n}\right)^{t_k}}}{P\left(1+\frac{y}{n}\right)^2} \right)$$

前述の式には次の関係が適用されます。

$$t_k = nk_0 + k - 1$$

$$c(k) = \frac{c}{n}A \quad for\ k = 1, \ldots, K-1$$

$$c(K) = \left(1+\frac{c}{n}\right)A$$

前述の式には次の関係が適用されます。

$$P = \sum_{k=1}^{K} \frac{c(k)}{\left(1+\frac{y}{n}\right)^{t_k}}$$

例: 債権のコンベクシティの計算

次のプログラムでは、CONVXP 関数は額面 1000、クーポン年率 0.01、年当たりのクーポン数 4、クーポン残数が 14 の債権のコンベクシティを返します。決済日から次のクーポン日までの時間は 0.165 で、年間の満期利回りは 0.08 です。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test/overwrite=yes;
  dcl double c;
  method run();
    c=convxp(1000,.01,4,14,.33/2,.08);
    put c;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
11.7290019868346
```

関連項目

関数:

- “CONVX 関数” (464 ページ)

COS 関数

余弦をラジアンで返します。

カテゴリ: 三角関数

返されるデータの種類: DOUBLE

構文

COS(*expression*)

必須引数

expression
数値に評価される有効な式です。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

例

次のプログラムは、COS 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double x y z;
  method run();
    x=cos(0.5);
    y=cos(0);
    z=cos(3.14159/3);
    put x;
    put y;
    put z;
```

```
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
0.87758256189037
1
0.50000076602519
```

COSH 関数

双曲線余弦をラジアンで返します。

カテゴリ: 三角関数

返されるデータの種類: DOUBLE

構文

COSH(*expression*)

必須引数

expression
数値に評価される有効な式です。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

COSH 関数は、次の式によって得られる引数の双曲線余弦を返します。

$$(e^{argument} + e^{-argument})/2$$

例

次のプログラムは、COSH 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double w x y z;
  method run();
    w=cosh(0);
    x=cosh(-5);
    y=cosh(4.37);
    z=cosh(0.5);
    put w;
    put x;
    put y;
    put z;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
1
74.2099485247878
39.5281414700662
1.12762596520638
```

COT 関数

余接を返します。

カテゴリ: 三角関数

返されるデータの種類: DOUBLE

構文

COT(*argument*)

必須引数

argument
ラジアンで表される数値定数、変数または式を指定します。

制限事項 *argument* は、0 または PI の倍数にはできません。

比較

COT 関数と TAN 関数は次のような関係です。

cot(x)=1/tan(x)

例

次のプログラムは、COMB 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double x y z;
  method run();
    x=cot(0.5);
    y=cot(1);
    z=cot(3.14159/3);
    put x;
    put y;
    put z;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
1.83048772171245
0.64209261593433
0.57735144856346
```

注: x=cot(0);を使用して COT 関数が欠損値を返した場合は、関数に無効な引数が入力されたことを示す NOTE がログに書き込まれます。これは正しい動作です。

関連項目

関数:

- “COS 関数” (468 ページ)
- “CSC 関数” (483 ページ)
- “SEC 関数” (1124 ページ)
- “SIN 関数” (1138 ページ)
- “TAN 関数” (1185 ページ)

COUNT 関数

指定した部分文字列が文字列内に含まれる個数を数えます。

カテゴリ: 文字

返されるデータの種類: DOUBLE

構文

COUNT(*string*, *substring*[, *modifiers*])

必須引数

string
文字定数、変数または式を指定します。この中にある部分文字列を数えます。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

ヒント 文字のリテラル文字列を引用符で囲みます。

substring
string でカウントされる文字定数、変数または式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

ヒント 文字のリテラル文字列を引用符で囲みます。

オプション引数

modifiers
1 つ以上の修飾子を文字定数、変数または式で指定します。

修飾子として使用できる文字(大文字または小文字)は次のとおりです。

I
数えるときに大文字と小文字を区別しません。この修飾子を指定しないと、COUNT は *substring* の文字の大文字と小文字に一致する部分文字列のみを数えます。

T
string および *substring* から末尾の空白を取り除きます。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

ヒント　*modifiers* が定数の場合、引用符で囲みます。一組の引用符で複数の定数を指定します。*modifiers* を変数または式として表すこともできます。

詳細

基本

COUNT 関数は左から右方向に *string* を検索し、指定した *substring* が何回出現するかを検出して出現数を返します。*string* 内に部分文字列が見つからない場合、COUNT は値 0 を返します。

注意
指定した部分文字列が文字列中に重複して 2 回出現する場合、結果は未定義になります。 たとえば、COUNT('boobooboo', 'booboo')は 1 または 2 のいずれかを返す可能性があります。

比較

COUNT 関数は文字列中の部分文字列を数えるのに対し、COUNTC 関数は文字列中の個別の文字を数えます。

例

次のプログラムは、COUNT 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl char(50) xyz expression1 expression2 expression3;
  dcl double howmanythis howmanyis howmanythis_i howmanyis_i howmanyis_it;
  dcl char(45) variable1 variable3;
  dcl char(3) variable2;
  method run();
    xyz='This is a thistle? Yes, this is a thistle.';
    howmanythis=count(xyz,'this');
    put howmanythis;
    xyz='This is a thistle? Yes, this is a thistle.';
    howmanyis=count(xyz,'is');
    put howmanyis;
    howmanythis_i=count('This is a thistle? Yes, this is a thistle.',
      'this', 'i');
    put howmanythis_i;
```

```
      variable1='This is a thistle? Yes, this is a thistle.';
      variable2='is ';
      variable3='i';
      howmanyis_i=count(variable1,variable2,variable3);
      put howmanyis_i;
      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;
    end;
  enddata;
  run;
  quit;
```

SAS は次の出力をログに書き込みます。

```
3
6
4
4
6
```

関連項目

関数:

- “COUNTC 関数” (474 ページ)
- “COUNTW 関数” (478 ページ)
- “FIND 関数” (628 ページ)
- “INDEX 関数” (705 ページ)

COUNTC 関数

文字のリストに表示される(または表示されない)文字列内の文字の個数を数えます。

カテゴリ:	文字
返されるデータの種類:	DOUBLE

構文

COUNTC(*string*, *charlist*[, *modifiers*])

必須引数

string
文字定数、変数または式を指定します。この中にある文字を数えます。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

ヒント 文字のリテラル文字列を引用符で囲みます。

charlist
文字リストを初期化する文字定数、変数または式を指定します。COUNTC はこのリストの文字を数えます(ただし、V 修飾子を *modifiers* 引数で指定しない場合)。V 修飾子を指定すると、このリストにないすべての文字を数えます。他の修飾子を使うことでリストに文字をさらに追加できます。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

ヒント 文字のリテラル文字列を引用符で囲みます。

修飾子を処理した後でリストに文字がない場合、COUNTC はゼロを返します。

オプション引数

modifiers
文字定数、変数または式を指定します。これらの空白以外の各文字で COUNTC 関数のアクションを変更します。空白は無視されます。A (アルファベット文字をリストに追加)、C (制御文字を追加)、D (数字を追加)の大文字と小文字を修飾子として使用できます。他の修飾子も使用することができます。詳細については、*SAS DS2 言語リファレンス*を参照してください。

文字定数、変数または式を指定します。これらの空白以外の各文字で COUNTC 関数のアクションを変更します。空白は無視されます。

修飾子として使用できる文字(大文字または小文字)は次のとおりです。

A
文字のリストにアルファベット文字を追加します。

B
左から右方向ではなく、右から左方向に *string* をスキャンします。

C
文字のリストに制御文字を追加します。

D
文字のリストに数字を追加します。

F
アンダースコアと英字(つまり、VALIDVARNAME=V7 を使用して SAS 変数名を開始できる文字)を文字のリストに追加します。

G
文字のリストにグラフィカル文字を追加します。

H
文字のリストに水平タブを追加します。

I
大文字と小文字を区別しません。

L
小文字を文字リストに追加します。

N
文字のリストに数字、アンダースコアおよび英文字(VALIDVARNAME=V7 を使用した SAS 変数名内に表示可能な文字)を追加します。

O
この COUNTC インスタンスを最初に呼び出す 1 回のみ、*charlist* 引数および *modifier* 引数を処理します。以降の呼び出しにおける *charlist* または *modifier* の値の変更は、COUNTC で無視されることがあります。

P
文字のリストに句読点を追加します。

S
文字のリストに空白文字(空白、水平タブ、垂直タブ、キャリッジリターン、ラインフィード、フォームフィード)を追加します。

T
string および *chars* から末尾の空白を取り除きます。両方(またはすべて)の文字引数ではなく一方のみから末尾の空白を削除する場合は、COUNTC 関数に T 修飾子を使用するかわりに、TRIM 関数を使用します。

U
大文字を文字リストに追加します。

V
文字リストに現れない文字を数えます。この修飾子を指定しないと、COUNTC はこの文字リストに現れない文字を数えます。

W
印刷可能文字を文字リストに追加します。

X
文字のリストに 16 進文字を追加します。

データの種類　CHAR、VARCHAR

ヒント　*modifier* が定数の場合、引用符で囲みます。一組の引用符で複数の定数を指定します。

詳細

COUNTC 関数では、文字引数を null に指定できます。null 引数は長さがゼロの文字列として扱われます。文字リストに数える文字がない場合、COUNTC はゼロを返します。

注: CHAR データ型の文字列には、常に宣言された長さまで空白が埋め込まれることに注意してください。VARCHAR データ型の文字列では、宣言された長さではなく、実際の文字列の長さが返されます。

比較

COUNTC 関数は文字列中の個別の文字を数えるのに対し、COUNT 関数は文字列中の部分文字列の文字を数えます。

例

COUNTC 関数で文字列中の文字数を数えるときに、修飾子を使用する場合と使用しない場合のプログラムを次に示します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl char(24) string a b b_i abc_i abc_iv abc_ivt;
    method run();
      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');
    end;
enddata;
run;
quit;
proc print data=test; run;
```

アウトプット 7.6　COUNTC 関数で修飾子を使用する場合としない場合の結果

Obs	string	a	b	b_i	abc_i	abc_iv	abc_ivt
1	Baboons Eat Bananas	5	1	3	8	16	11

関連項目

関数:

- “ANYALNUM 関数” (276 ページ)
- “ANYALPHA 関数” (279 ページ)
- “ANYCNTRL 関数” (282 ページ)
- “ANYDIGIT 関数” (284 ページ)
- “ANYGRAPH 関数” (289 ページ)
- “ANYLOWER 関数” (292 ページ)
- “ANYPRINT 関数” (296 ページ)
- “ANYPUNCT 関数” (299 ページ)
- “ANYSPACE 関数” (302 ページ)
- “ANYUPPER 関数” (304 ページ)
- “ANYXDIGIT 関数” (307 ページ)
- “COUNT 関数” (472 ページ)
- “COUNTW 関数” (478 ページ)
- “FINDC 関数” (631 ページ)
- “INDEXC 関数” (707 ページ)
- “NOTALNUM 関数” (876 ページ)
- “NOTALPHA 関数” (878 ページ)
- “NOTCNTRL 関数” (881 ページ)
- “NOTDIGIT 関数” (883 ページ)
- “NOTGRAPH 関数” (888 ページ)
- “NOTLOWER 関数” (891 ページ)
- “NOTPRINT 関数” (895 ページ)
- “NOTPUNCT 関数” (897 ページ)
- “NOTSPACE 関数” (900 ページ)
- “NOTUPPER 関数” (903 ページ)
- “NOTXDIGIT 関数” (906 ページ)
- “VERIFY 関数” (1233 ページ)

COUNTW 関数

文字列中の単語数を数えます。

カテゴリ: 文字

返されるデータの種類: DOUBLE

構文

COUNTW(*string*[, *chars*[, *modifiers*]])

必須引数

string
文字定数、変数または式を指定します。この中にある単語を数えます。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

オプション引数

chars
文字のリストを初期化する任意の文字定数、変数または式を指定します。リスト中の文字は単語を区切るための区切り文字です(ただし *modifier* 引数で K 修飾子を使用しない場合)。K 修飾子を指定すると、このリストにないすべての文字が区切り文字となります。他の修飾子を使うことでリストに文字をさらに追加できます。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

modifiers
文字定数、変数または式を指定します。これらの空白以外の各文字で COUNTW 関数のアクションを変更します。A (アルファベット文字をリストに追加)、C (制御文字を追加)、D (数字を追加)の大文字と小文字を修飾子として使用できます。他の修飾子も使用することができます。詳細については、*SAS DS2 言語リファレンス*を参照してください。

文字定数、変数または式を指定します。これらの空白以外の各文字で COUNTW 関数のアクションを変更します。

修飾子として使用できる文字(大文字または小文字)は次のとおりです。

A
文字のリストにアルファベット文字を追加します。

B
左から右方向ではなく、右から左方向に *string* をスキャンします。

C
文字のリストに制御文字を追加します。

D
文字のリストに数字を追加します。

F
アンダースコアと 英字(つまり、VALIDVARNAME=V7 を使用して SAS 変数名を開始できる文字)を文字のリストに追加します。

G
文字のリストにグラフィカル文字を追加します。

H
文字のリストに水平タブを追加します。

I
大文字と小文字を区別しません。

L
小文字を文字リストに追加します。

N
文字のリストに数字、アンダースコアおよび英文字(VALIDVARNAME=V7 を使用した SAS 変数名内に表示可能な文字)を追加します。

O
この COUNTC インスタンスを最初に呼び出す 1 回のみ、*charlist* 引数および *modifier* 引数を処理します。以降の呼び出しにおける *charlist* または *modifier* の値の変更は、COUNTC で無視されることがあります。

P
文字のリストに句読点を追加します。

Q
引用符で囲まれた部分文字列内の区切り文字を無視します。*string* の値に、対になっていない引用符が含まれている場合、左から右へのスキャンと、右から左へのスキャンとでは異なる単語が生成されます。

S
文字のリストに空白文字(空白、水平タブ、垂直タブ、キャリッジリターン、ラインフィード、フォームフィード)を追加します。

T
string および *chars* から末尾の空白を取り除きます。両方(またはすべて)の文字引数ではなく一方のみから末尾の空白を削除する場合は、COUNTC 関数に T 修飾子を使用するかわりに、TRIM 関数を使用します。

U
大文字を文字リストに追加します。

V
文字リストに現れない文字を数えます。この修飾子を指定しないと、COUNTC はこの文字リストに現れない文字を数えます。

W
印刷可能文字を文字リストに追加します。

X
文字のリストに 16 進文字を追加します。

データの種類　CHAR、VARCHAR

詳細

"単語"の定義

COUNTW 関数では、"単語"とは、次の特性のいずれかを持つ部分文字列を指します。

- 左側が区切り文字または文字列の先頭で境界設定されている
- 右側が区切り文字または文字列の末尾で境界設定されている
- 区切り文字を含まない。ただし、Q 修飾子を使用している場合で、引用符を含む部分文字列内に区切り文字が囲まれている場合を除きます。

注: "単語"の定義は SCAN 関数と COUNTW 関数の両方で共通しています。

区切り文字は、単語を区切るために指定できるいくつかの文字を示します。

ASCII 環境で COUNTW 関数を使用する

引数が 2 つしかない COUNTW 関数を使用する場合、デフォルトの区切り文字は次のとおりです。

- お使いのコンピューターが ASCII 文字を使用している場合、デフォルトの区切り文字は次のようになります。

  空白 ! $ % & ( ) * + , - ./ ; < ^ |

null 引数の使用

COUNTW 関数では、文字引数を null に指定できます。null 引数は長さがゼロの文字列として扱われます。数値引数は null にできません。

M 修飾子の使用

M 修飾子を使用しない場合、単語には少なくとも 1 つの文字が含まれている必要があります。M 修飾子を使用する場合、単語の長さをゼロとすることができます。この場合、単語数は文字列中の区切り文字数に 1 を加算した数です。Q 修飾子を使用するときは引用符に囲まれている文字列中の区切り文字は数えません。

例

次のプログラムでは、COUNTW 関数に M 修飾子および P 修飾子を使う方法を示します。

各文字列の **mp** の値の説明は次のとおりです。

- ピリオドは区切り文字であり、m 修飾子を使用すると、最後のピリオドは長さがゼロの後続の単語を参照しますが、それでも単語を参照します。つまり、ピリオドの前に 1 単語、ピリオドの後に 1 単語で、合計 2 つの単語があるということです。

- 区切り文字がないため、1 単語だけです。
- p 修飾子が区切り文字として句読点を追加するため、3 単語になります。
- p 修飾子が句読点を追加するため、/は区切り文字です。m 修飾子を使用すると、先頭の/が長さゼロの先頭の単語を参照して、合計 6 単語になります。
- 最初の\はエスケープ文字です 2 番目の\は区切り文字なので、6 単語があります。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl char(60) string1 having informat $char60. format $char60.;
  method init();
    string1='The quick brown fox jumps over the lazy dog.'; output;
    string1='       Leading blanks'; output;
    string1='2+2=4'; output;
    string1='/unix/path/names/use/slashes'; output;
    string1='\Windows\Path\Names\Use\Backslashes'; output;
  end;
enddata;
run;

data test_out (overwrite=yes);
  dcl double default blanks mp;
  method run();
    set test;
    default = countw(string1);
    blanks = countw(string1, ' ');
    mp = countw(string1, '.', 'mp');
    put 'String= ' string1 'Default= ' default 'Blanks= ' blanks 'MP= ' mp;
  end;
enddata;
run;
quit;
```

アウトプット 7.7　M および P 修飾子で COUNTW 関数を使用した結果

default	blanks	mp	string1
9	9	2	The quick brown fox jumps over the lazy dog.
2	2	1	Leading blanks
2	1	3	2+2=4
5	1	6	/unix/path/names/use/slashes
1	1	6	\Windows\Path\Names\Use\Backslashes

注: Windows のパス名にはダブルバックスラッシュが使用されています。

関連項目

関数:

- “COUNT 関数” (472 ページ)
- “COUNTC 関数” (474 ページ)
- “FINDW 関数” (640 ページ)
- “SCAN 関数” (1109 ページ)

CSC 関数

余割を返します。

カテゴリ:	三角関数
返されるデータの種類:	DOUBLE

構文

CSC(*argument*)

必須引数

argument
ラジアンで表される数値定数、変数または式を指定します。

制限事項	*argument* は、0 または PI の倍数にはできません。
データの種類	DOUBLE

比較

CSC 関数と SIN 関数は次のような関係です。

csc(x)=1/sin(x)

例

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double x y z;
  method run();
    x=csc(.5);
    y=csc(1);
    z=csc(3.14159/3);
    put x;
    put y;
    put z;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
2.08582964293348
1.18839510577812
1.15470112806662
```

注: x=csc(0);を使用して CSC 関数が欠損値を返した場合は、関数に無効な引数が入力されたことを示す NOTE がログに書き込まれます。これは正しい動作です。

関連項目

関数:

- “COS 関数” (468 ページ)
- “COT 関数” (470 ページ)
- “SEC 関数” (1124 ページ)
- “SIN 関数” (1138 ページ)
- “TAN 関数” (1185 ページ)

CSS 関数

修正済み平方和を返します。

カテゴリ:	記述統計
返されるデータの種類:	DOUBLE

構文

CSS(*expression*[, ...*expression*])

必須引数

expression
数値に評価される有効な式を指定します。

要件 少なくとも 1 つの非 null または非欠損の式が必要です。

データの種類 DOUBLE

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

例

次のプログラムは、CSS 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double x1 x2 x3 x4;
  method run();
    x1=css(5,9,3,6);
    put x1=;
    x2=css(5,8,9,6,.);
    put x2=;
    x3=css(8,9,6,.);
    put x3=;
    x4=css(of x1-x3);
    put x4=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
x1=18.75
x2=10
x3=4.66666666666666
x4=101.11574074074
```

関連項目

関数:

- “USS 関数” (1229 ページ)

CUMIPMT 関数

開始期間と終了期間の間でローンに対して支払われる累積利息を返します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

CUMIPMT(*rate*, *number-of-periods*, *principal-amount*[, *start-period*][, *end-period*][, *type*])

必須引数

rate
支払期間ごとの利率を指定します。

データの種類 DOUBLE

number-of-periods
支払期間の数を指定します。*number-of-periods* は正の整数にする必要があります。

データの種類 DOUBLE

principal-amount
ローンの元金を指定します。欠損値が指定されている場合、ゼロとみなされます。

データの種類 DOUBLE

オプション引数

start-period
計算の開始期間を指定します。

データの種類 DOUBLE

end-period
計算の終了期間を指定します。

データの種類 DOUBLE

type
期首と期末のどちらで支払がなされるのかを指定します。0 は期末の支払を表し、1 は期首の支払を表します。*type* が省略された場合、または欠損値が指定されている場合は、ゼロとみなされます。

データの種類 DOUBLE

例

- 名目年利 9%で$125,000 の 30 年ローンを組み、期末の月次払いとした場合に 2 年目に支払う累積利息は次のように計算します。

  注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test/overwrite=yes;
  dcl double TotalInterest having format dollar10.2;
  method run();
    TotalInterest= CUMIPMT(0.09/12, 360, 125000, 13, 24, 0);
    put 'Total Interest=' TotalInterest;
  end;
enddata;
run;
quit;
```

  計算によって$11,135.23 という値が返されます。

- 同じローンの最初の期間に支払われる利息は、次の方法で計算されます。

  注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test/overwrite=yes;
  dcl double first_period_interest having format dollar10.2;
  method run();
    first_period_interest= CUMIPMT(0.09/12, 360, 125000, 1, 1, 0);
    put 'Total Interest=' first_period_interest;
  end;

enddata;
run;
quit;
```

  計算によって$937.50 という値が返されます。

関連項目

関数:

- “CUMPRINC 関数” (488 ページ)

CUMPRINC 関数

開始期間と終了期間の間でローンに対して支払われる累積元本を返します。

カテゴリ:	財務
返されるデータの種類:	DOUBLE

構文

CUMPRINC(*rate*, *number-of-periods*, *principal-amount*[, *start-period*] [, *end-period*][, *type*])

必須引数

rate
支払期間ごとの利率を指定します。

データの種類	DOUBLE

number-of-periods
支払期間の数を指定します。

要件	*number-of-periods* は正の整数にする必要があります。
データの種類	DOUBLE

principal-amount
ローンの元金を指定します。

データの種類	DOUBLE
注	欠損値または null 値が指定されている場合、ゼロとみなされます。

オプション引数

start-period
計算の開始期間を指定します。

データの種類	DOUBLE

end-period
計算の終了期間を指定します。

データの種類 DOUBLE

type
期首と期末のどちらで支払がなされるのかを指定します。0 は期末の支払を表し、1 は期首の支払を表します。*type* が省略された場合、または欠損値が指定されている場合は、ゼロとみなされます。

データの種類 DOUBLE

例

- 名目年利 9%で$125,000 の 30 年ローンを組み、期末の月次払いとした場合に 2 年目に支払う累積元本は次のように計算します。

  注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test/overwrite=yes;
  dcl double PrincipalYear2 having format dollar10.2;
  method run();
    PrincipalYear2=CUMPRINC(0.09/12, 360, 125000, 12, 24, 0);
    put 'Principal Year 2 EOP=' PrincipalYear2;
  end;
enddata;
run;
quit;
```

  計算によって$1008.23 という値が返されます。

- 同じローンで、期首支払の場合に 2 年目に支払う元本は次のように計算します。

  注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test/overwrite=yes;
  dcl double PrincipalYear2b having format dollar10.2;
  method run();
    PrincipalYear2b = CUMPRINC(0.09/12, 360, 125000, 12, 24, 1);
    put 'Principal Year 2 BOP=' PrincipalYear2b;
  end;
enddata;
run;
quit;
```

  計算によって$1000.73 という値が返されます。

関連項目

関数:

- “CUMIPMT 関数” (486 ページ)

CV 関数

変動係数を返します。

カテゴリ: 記述統計

返されるデータの種類: DOUBLE

構文

CV(*expression-1*, *expression-2* [, ...*expression-n*])

必須引数

expression
数値に評価される有効な式を指定します。

要件 少なくとも 2 つの引数が必要です。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

例

次のプログラムは、CV 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double x1 x2 x3 x4;
  method run();
    x1=cv(5, 9, 3, 6);
    put x1=;
    x2=cv(5, 8, 9, 6, .);
    put x2=;
```

```
      x3=cv(8, 9, 6, .);
      put x3=;
      x4=cv(of x1-x3);
      put x4=;
    end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
x1=43.4782608695652
x2=26.082026547865
x3=19.9242421519819
x4=40.9535392160926
```

DAIRY 関数

AIRY 関数の導関数を返します。

カテゴリ: 数学

返されるデータの種類: DOUBLE

構文

DAIRY(*x*)

必須引数

x
数値の定数、変数または式を指定します。

詳細

DAIRY 関数は AIRY 関数の導関数の値を返します。 (リファレンスのリストを参照。)

例

次のプログラムは、DAIRY 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double x y;
  method run();
    x=dairy(2.0);
    put x;
    y=dairy(-2.0);
    put y;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
-0.05309038443365
0.61825902074169
```

DATDIF 関数

指定された日数計算規則に従って 2 つの日付間の差を計算した後に、その日付間の日数を返します。

カテゴリ: 日付と時間

返されるデータの種類: DOUBLE

構文

DATDIF(*sdate*, *edate*, *basis*)

必須引数

sdate
開始日を識別する SAS 日付値を指定します。

データの種類 DATE

ヒント *sdate* が月末の場合、SAS はその日付を 1 か月が 30 日の月の最終日として処理します。

edate
終了日を識別する SAS 日付値を指定します。

データの種類 DATE

ヒント *edate* が月末の場合、SAS はその日付を 1 か月が 30 日の月の最終日として処理します。

basis
日数計算基準を表す文字列を指定します。*basis* の有効な値は、'30/360'、'ACT/ACT'、'ACT/360'、'ACT/365'です。たとえば、'ACT/365'は、各月に実際のカレンダー日数を使用し、すべての年の日数に実際の日数に関わらず 365 日を使用します。詳細については、*SAS DS2 言語リファレンス*を参照してください。

日数計算基準を表す文字列を指定します。*basis* の有効な値は次のとおりです。

'30/360'
その月または年の実際のカレンダー日数に関わらず、1 か月を 30 日、1 年を 360 日として指定します。

月の最終日に利息を支払う証券では、その利息を常にその月の最終日に支払うか、または常に各月の同じ日付に支払います。唯一の例外は、2 月 30 日など、その日がその月の有効な日ではない場合です。詳細については、“日数計算基準(30/360)の計算方法” (*SAS 関数と CALL ルーチン: リファレンス*)を参照してください。

別名 '360'

'ACT/ACT'
日付間の実際の日数を使用します。各月にその月の実際のカレンダー日数を使用し、各年にその年の実際のカレンダー日数を使用します。

別名 *'Actual'*

'ACT/360'
各月に実際のカレンダー日数を使用し、すべての年の日数に実際の日数に関わらず 360 日を使用します。

ヒント *ACT/360* は短期間の証券で使用されます。

'ACT/365'
各月に実際のカレンダー日数を使用し、すべての年の日数に実際の日数に関わらず 365 日を使用します。

ヒント *ACT/365* は短期間の証券で使用されます。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

詳細

基本

DATDIF 関数は証券業界にとって特定の意味を持ち、その計算方法は実際の日数の数え方とは異なります。実際の日数を含む月と年を計算に使用できます。1 か月を 30 日または 1 年を 360 日として計算することもできます。標準の証券計算方法の詳細については、この関数の最後にあるリファレンスセクションを参照してください。

注: 月の日数を数える場合、DATDIF では*常に*開始日が含まれ、終了日は除外されます。

日数計算基準(30/360)の計算方法

2 つの日付間の日数を計算するには、次の式を使用します。

$$Number of days = [(Y2 - Y1) * 360] + [(M2 - M1) * 30] + (D2 - D1)$$

引数

Y2
後の日付の年を指定します。

Y1
前の日付の年を指定します。

M2
後の日付の月を指定します。

M1
前の日付の月を指定します。

D2
後の日付の日を指定します。

D1
前の日付の日を指定します。

すべての月は 30 日のみになるため、30 日以外の日数が含まれる月は調整する必要があります。この調整は、2 つの日付間の日数を計算する前に行います。

次の規則が適用されます。

- 証券が月末ルールを使用していて、D2 と D1 の両方が 2 月(うるう年以外は 28 日、うるう年は 29 日)の最終日の場合、D2 を 30 に変更します。
- 証券が月末ルールを使用していて、D1 が 2 月の最終日の場合、D1 を 30 に変更します。
- D2 の値が 31 で D1 の値が 30 または 31 の場合、D2 を 30 に変更します。
- D1 の値が 31 の場合、D1 を 30 に変更します。

例

次のプログラムでは、DATDIF は 2 つの日付間の実際の日数、および 1 か月を 30 日、1 年を 360 日とした場合の日数を返します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl date sdate edate;
```

```
  dcl double actual days360;
  method run();
    sdate= date'1978-10-16';
    edate= date'1996-02-16';
    sassdate=to_double(sdate);
    sasedate=to_double(edate);
    actual=datdif(sassdate, sasedate, 'act/act');
    days360=datdif(sassdate, sasedate, '30/360');
    put  'Actual= ' actual;
    put 'Days 360 = ' days360;
  end;
enddata;
run;
quit;
```

次の行が SAS ログに書き込まれます。

```
Actual= 6332
Days 360= 6240
```

関連項目

関数:

- “YRDIF 関数” (1267 ページ)

参考文献

Securities Industry Association 1994. *Standard Securities Calculation Methods - Fixed Income Securities Formulas for Analytic Measures*. 2 巻 New York, NY: Securities Industry Association.

DATE 関数

SAS 日付値として現在の日付を返します。

カテゴリ:	日付と時間
別名:	TODAY
返されるデータの種類:	DOUBLE

構文

DATE( )

引数なし

DATE 関数には引数がありません。

比較

DATE 関数は引数を取りません。返される SAS 日付値は、1960 年 1 月 1 日から現在の日付までの日数です。

DS2 が日付を処理する方法の詳細については、“DS2 式” (*SAS DS2 プログラマガイド*)を参照してください。

例

次のプログラムは、DATE 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double x having format nldate.;
  method run();
    x=date();
    put x;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
2019/01/22
```

関連項目

関数:

- “TODAY 関数” (1205 ページ)

DATEJUL 関数

ユリウス暦の日付を SAS 日付値に変換します。

カテゴリ: 日付と時間

返されるデータの種類: DOUBLE

構文

DATEJUL(*julian-date*)

必須引数

julian-date
数値に評価され、ユリウス暦の日付を表す有効な式を指定します。ユリウス暦の日付は *yyddd* または *yyyyddd* 形式の日付です。このとき、*yy* と *yyyy* はそれぞれ年を表す 2 桁または 4 桁の整数、*ddd* は該当日がその年の何日目かを表す数です。*ddd* の値は 1 から 365(閏年の場合は 366)の間である必要があります。

データの種類 DOUBLE

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

詳細

SAS 日付値は、1960 年 1 月 1 日から指定日付までの日数です。DATEJUL 関数は、1960 年 1 月 1 日から *julian-date* で指定されたユリウス暦の日付までの日数を返します。

DS2 での日付の処理方法の詳細については、"DS2 式" (*SAS DS2 プログラマガイド*)を参照してください。

例

次のプログラムは、DATEJUL 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double xstart xend having format nldate.;
  method run();
    xstart=datejul(94365);
    put xstart;
    xend=datejul(2001001);
    put xend;
  end;
enddata;
```

```
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
December 31, 1994
January 01, 2001
```

関連項目

関数:

- “JULDATE 関数” (789 ページ)

DATEPART 関数

SAS 日時値から日付を抽出します。

カテゴリ: 日付と時間

返されるデータの種類: DOUBLE

構文

DATEPART(*datetime*)

必須引数

datetime
SAS 日時値を表す有効な式を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

SAS 日時値は、1960 年 1 月 1 日から特定の日付内の時、分、秒までの秒数です。DATEPART 関数は、SAS 日時値の日付部分を決定し、日付を SAS 日付値として返します。これは 1960 年 1 月 1 日からの日数です。

例

次のプログラムは、DATEPART 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double dtvalue;
  dcl double dp having format date9.;
  method run();
    dtvalue=1652165417;
    dp=datepart(dtvalue);
    put dp;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
09MAY2012
```

関連項目

- “DS2 式” (*SAS DS2 プログラマガイド*)

関数:

- “DATETIME 関数” (499 ページ)
- “TIMEPART 関数” (1189 ページ)

DATETIME 関数

SAS 日時値として現在の日時を返します。

カテゴリ:	日付と時間
返されるデータの種類:	DOUBLE

構文

DATETIME( )

比較

DATETIME 関数は引数を取りません。返される SAS 日時値は、1960 年 1 月 1 日から現在の日時までの秒数です。

例

次のプログラムは、DATETIME 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double dt ;
  method run();
    dt=datetime();
    put dt;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
1865079581.31599
```

関連項目

概念:

- “DS2 の日付と時間” (*SAS DS2 プログラマガイド*)

関数:

- “DATE 関数” (495 ページ)
- “TIME 関数” (1187 ページ)

DAY 関数

SAS 日付値として月の日を返します。

カテゴリ: 日付と時間

返されるデータの種類: DOUBLE

構文

DAY(*date*)

必須引数

date
SAS 日付値を表す有効な式を指定します。

データの種類 DOUBLE

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

詳細

DAY 関数は、月の日を表す 1 から 31 までの整数を生成します。

SAS 日付値は、1960 年 1 月 1 日から特定の日付までの日数です。

例

次のプログラムは、SAS 日付値である dayvalue の値が 22541 で、その月の 18 日目である場合の DAY 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double dv;
  method run();
    dv=day(22541);
    put dv;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
18
```

関連項目

- “DS2 式” (*SAS DS2 プログラマガイド*)

関数:

- “MONTH 関数” (863 ページ)
- “YEAR 関数” (1263 ページ)

DEQUOTE 関数

単一引用符で始まる文字列から一致する単一引用符を削除し、閉じ引用符の右側にあるすべての文字を削除します。

カテゴリ: 文字

返されるデータの種類: CHAR、NCHAR、NVARCHAR、VARCHAR

構文

DEQUOTE(*expression*)

必須引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

DEQUOTE 関数で返される値は、*expression* の最初の文字または最初の 2 文字に依存します。

- *expression* の最初の文字が引用符でない場合、DEQUOTE は構文エラーを返します。
- *expression* の最初の文字が単一引用符の場合、DEQUOTE 関数は結果からその単一引用符を削除します。DEQUOTE は、次に *expression* を左から右にスキャンし、他の単一引用符または二重引用符を探します。

  ペアになっている単一引用符はすべて、1 つの単一引用符に置き換えられます。

  ペアになっている二重引用符はすべて保持されます。

二重引用符が 2 文字目の場合、DEQUOTE は結果から二重引用符を削除します。DEQUOTE は、次に、*expression* を左から右にスキャンしていきます。一致する二重引用符が見つかった場合は、二重引用符の間のテキストが返されます。閉じ二重引用符の右側、つまり *expression* の末尾までのテキストは、結果から削除されます。

expression でペアになっていない最初の単一引用符は閉じ単一引用符であり、削除されます。

閉じかっこが単一引用符の後に続く場合、関数は引用符で囲まれていない文字列を返します。閉じ単一引用符の右側に文字が存在する場合、関数は構文エラーになり、エラーが SAS ログに出力されます。

- *expression* が二重引用符で囲まれている場合、DEQUOTE 関数は null または欠損値を返します。

注: *expression* が引用符に囲まれた定数の場合、それらの引用符は *expression* の値には含まれません。そのため、定数を示す引用符は DEQUOTE を使用して削除する必要はありません。

例

例 1: 引用符なし

次のプログラムは、DEQUOTE 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl char(75) string;
  method run();
    string= dequote(Omitting quotation marks causes an error);
    put string;
  end;
enddata;
run;
quit;
```

SAS は次のエラーをログに書き込みます。

```
ERROR: Compilation error.
ERROR: Line 8: Parse encountered identifier when expecting ')'.
ERROR: Line 8: Parse failed:  string= dequote(Omitting  >>> quotation <<<  marks causes an error
```

例 2: 文字列に先頭の引用符なし

次のプログラムは、DEQUOTE 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl char(75) string;
  method run();
    string= dequote(No 'leading' quotation marks cause an error);
    put string;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
ERROR: Compilation error.
ERROR: Line 28: Parse encountered constant when expecting ')'.
ERROR: Line 28: Parse failed:  string= dequote(No  >>> 'leading' <<<  quotation marks caus
```

例 3: 単一の一致する引用符

次のプログラムは、DEQUOTE 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl char(75) string;
  method run();
    string = dequote(' "Single matched quotation marks are removed" ');
    put string;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
"Single matched quotation marks are removed"
```

例 4: 一致する二重引用符

次のプログラムは、DEQUOTE 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl char(75) string;
  method run();
```

```
    string= dequote("Matched double quotation marks cause an error.");
    put string;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
ERROR: Compilation error.
WARNING: Line 68: No DECLARE for referenced variable "matched double quotation marks cause an
error."; creating it as a global variable of type double.
ERROR: BASE driver, Column name Matched double quotation marks cause an error. is too long for a
 SAS name
ERROR: Unable to execute CREATE TABLE statement for table work.test.
```

例 5: ペアの単一引用符

次のプログラムは、DEQUOTE 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl char(75) string;
  method run();
    string= dequote('Paired '' single '' quotation marks are reduced to a
      single quotation mark');
    put string;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
Paired ' single ' quotation marks are reduced to a single quotation mark
```

例 6: 始め引用符の前にスペースがある 単一引用符内の二重引用符

次のプログラムは、DEQUOTE 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl char(100) string;
  method run();
    string= dequote(' "Double quotation marks within single quotation marks,
      with space before open quotation mark" ');
    put string;
```

```
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
"Double quotation marks within single quotation marks, with space before open quotation mark"
```

例 7: 始め引用符の前にスペースがない 単一引用符内の二重引用符

次のプログラムは、DEQUOTE 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl char(110) string;
  method run();
    string= dequote(dequote('"Double quotation marks within single quotation
      marks, without space before open quotation mark causes an error"'));
    put string;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
ERROR: Compilation error.
ERROR: Line 103: Parse encountered ';' when expecting ')'.
ERROR: Line 103: Parse failed: ) >>> ; <<< );
```

例 8: 終わり二重引用符の後のテキスト

次のプログラムは、DEQUOTE 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl char(75) string;
  method run();
    string= dequote(' "Text after closing double quotation mark"
      is removed ');
    put string;
  end;
enddata;
run;
```

```
quit;
```

SAS は次の出力をログに書き込みます。

```
"Text after closing double quotation mark" is removed
```

例 9: 一致する引用符なし

次のプログラムは、DEQUOTE 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl char(75) string;
  method run();
      string= dequote(' No matching quotation mark);
      put string;
  end;
enddata;
run;
quit;
```

要求は応答を停止し、閉じ引用符を検索します。コードをサブミットする前に、対になっていない引用がないかコードを詳しく調べてください。

DEVIANCE 関数

確率分布に基づくデビアンスを返します。

カテゴリ:	数学
返されるデータの種類:	DOUBLE

構文

DEVIANCE(*distribution*, *variable*, *shape-parameter(s)*[, *ε*])

必須引数

distribution
分布を特定する文字定数、変数または式です。有効な分布は、'BERNOULLI', 'BINOMIAL'、'GAMMA'、'IGAUSS'、'NORMAL'、'POISSON'です。これらの各分布でサポートされているパラメーターの説明については、*SAS DS2 言語リファレンス*を参照してください。

分布を特定する文字定数、変数または式です。有効な分布については、次の表を参照してください。

分布	引数
Bernoulli (p. 508)	'BERNOULLI' | 'BERN'
二項 (p. 509)	'BINOMIAL' | 'BINO'
ガンマ (p. 509)	'GAMMA'
逆ガウス(Wald) (p. 510)	'IGAUSS' | 'WALD'
正規 (p. 510)	'NORMAL' | 'GAUSSIAN'
Poisson (p. 510)	'POISSON' | 'POIS'

variable
数値定数、変数または式です。

shape-parameter(s)
分布の形状の特性を示す 1 つ以上の分布固有の数値パラメーターです。

オプション引数

ε
正規分布以外のすべての分布に使用される任意の小さい数値です。

詳細

Bernoulli 分布

DEVIANCE('BERNOULLI', *variable*, *p*[, *ε*])

引数

variable
成功した場合は 1、失敗した場合は 0 の二値の数値の確率変数です。

p
ε ≤ *p* ≤ 1–*ε* の成功率の数値です。

ε
限界 *p* に使用される任意の正の数値です。間隔 0 ≤ *p* ≤ *ε* のすべての *p* 値は、*ε* で置換されます。間隔 1 – *ε* ≤ *p* ≤ 1 のすべての *p* 値は、1 – *ε* で置換されます。

DEVIANCE 関数は、成功が 1 の確率変数値として定義された成功率 *p* の Bernoulli 分布からデビアンスを返します。式は次のとおりです。

$$\text{DEVIANCE}('BERN', variable, p, \varepsilon) = \begin{cases} -2\log(1-p) & x = 0 \\ -2\log(p) & x = 1 \\ . & otherwise \end{cases}$$

二項分布

DEVIANCE('BINO', *variable*, *μ*, *n*[, *ε*])

引数

variable
成功数が含まれる数値の確率変数です。

範囲 $0 \le variable \le 1$

μ
数値の平均パラメーターです。

範囲 $n\varepsilon \le \mu \le n(1-\varepsilon)$

n
Bernoulli 試行数の整数パラメーターです。

範囲 $n \ge 0$

ε
限界 *μ* に使用される任意の正の数値です。間隔 $0 \le \mu \le n\varepsilon$ のすべての *μ* 値は、*nε* で置換されます。間隔 $n(1-\varepsilon) \le \mu \le n$ のすべての *μ* 値は、$n(1-\varepsilon)$で置換されます。

DEVIANCE 関数は、成功率 *p*、独立した Bernoulli 試行数 *n* の二項分布からデビアンスを返します。正規分布の DEVIANCE 関数は次の式で表されます。*x* は確率変数です。

$$\text{DEVIANCE}('BINO', x, \mu, n) = \begin{cases} . & x < 0 \\ 2\left(x\log\left(\frac{x}{\mu}\right) + (n-x)\log\left(\frac{n-x}{n-\mu}\right)\right) & 0 \le x \le n \\ . & x > n \end{cases}$$

ガンマ分布

DEVIANCE('GAMMA', *variable*, *μ*[, *ε*])

引数

variable
数値の確率変数です。

範囲 $variable \ge \varepsilon$

μ
数値の平均パラメーターです。

範囲 $\mu \ge \varepsilon$

ε
限界 *variable* および *μ* に使用される任意の正の数値です。間隔 $0 \le variable \le \varepsilon$ のすべての *variable* 値は、*ε* で置換されます。間隔 $0 \le \mu \le \varepsilon$ のすべての *μ* 値は、*ε* で置換されます。

DEVIANCE 関数は、平均パラメーター μ のガンマ分布からデビアンスを返します。ガンマ分布の DEVIANCE 関数は次の式で表されます。x は確率変数です。

$$\text{DEVIANCE}('GAMMA', x, \mu) = \begin{cases} . & x < 0 \\ 2\left(-\log\left(\frac{x}{\mu}\right) + \frac{x-\mu}{\mu}\right) & x \geq \varepsilon, \mu \geq \varepsilon \end{cases}$$

逆ガウス(Wald)分布

DEVIANCE('IGAUSS' | 'WALD', *variable*, *μ*[, *ε*])

引数

variable
数値の確率変数です。

範囲 *variable* ≥ *ε*

μ
数値の平均パラメーターです。

範囲 *μ* ≥*ε*

ε
限界 *variable* および *μ* に使用される任意の正の数値です。間隔 0 ≤ *variable* ≤ *ε* のすべての *variable* 値は、*ε* で置換されます。間隔 0 ≤ *μ* ≤ *ε* のすべての *μ* 値は、*ε* で置換されます。

DEVIANCE 関数は、平均パラメーター μ の逆ガウス分布からデビアンスを返します。逆ガウス分布の DEVIANCE 関数は次の式で表されます。x は確率変数です。

$$\text{DEVIANCE}('IGAUSS', x, \mu) = \begin{cases} . & x < 0 \\ \frac{(x-\mu)^2}{\mu^2 x} & x \geq \varepsilon, \mu \geq \varepsilon \end{cases}$$

正規分布

DEVIANCE('NORMAL' | 'GAUSSIAN', *variable*, *μ*)

引数

variable
数値の確率変数です。

μ
数値の平均パラメーターです。

DEVIANCE 関数は、平均パラメーター μ の正規分布からデビアンスを返します。正規分布の DEVIANCE 関数は次の式で表されます。x は確率変数です。

$$\text{DEVIANCE}('NORMAL', x, \mu) = (x-\mu)^2$$

Poisson 分布

DEVIANCE('POISSON', *variable*, *μ*[, *ε*])

引数

variable
数値の確率変数です。

範囲 *variable* ≥ 0

μ
数値の平均パラメーターです。

範囲 *μ* ≥*ε*

ε
限界 *μ* に使用される任意の正の数値です。間隔 0 ≤ *μ* ≤ *ε* のすべての *μ* 値は、*ε* で置換されます。

DEVIANCE 関数は、平均パラメーター *μ* の Poisson 分布からデビアンスを返します。Poisson 分布の DEVIANCE 関数は次の式で表されます。*x* は確率変数です。

$$\text{DEVIANCE}('POISSON', x, \mu) = \begin{cases} . & x < 0 \\ 2\left(x log\left(\frac{x}{\mu}\right) - (x - \mu)\right) & x \geq 0, \mu \geq \varepsilon \end{cases}$$

DHMS 関数

日、時、分、秒の値から SAS 日時値を返します。

カテゴリ: 日付と時間

返されるデータの種類: DOUBLE

構文

DHMS(*date*, *hour*, *minute*, *second*)

必須引数

date
SAS 日付値を表す有効な式を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

hour
1 から 12 までの整数を表す数値式を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

minute
1 から 59 までの整数を表す数値式を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

second
1 から 59 までの整数を表す数値式を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

DHMS 関数は、SAS 日時値を表す数値を返します。正または負の数値が返されます。

例

例 1: DHMS 関数の使用

次のプログラムは、DHMS 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double dtvalue;
  method run();
    dtvalue=dhms(mdy(12,31,2018),12,01,01);
    put dtvalue;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
1861876861
```

例 2: 日付と時間の値の組み合わせ

次のプログラムでは、現在の日時を使用して、SAS 日付値と SAS 時間値を組み合わせて SAS 日時値にする方法を示します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double day tme dt;
  method run();
    day=date();
    tme=time();
    dt=dhms(day,0,0,tme);
    put dt;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
1863788289.77899
```

例 3: 日時値の使用

次のプログラムでは、日時値で DHMS 関数を使用する方法を示します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double dtid dtid1 dtid2 having format datetime.;
  method run();
    dtid=dhms(mdy(01, 03, 2018), 15, 30, 15);
    put dtid;
    dtid1=dhms(mdy(01, 03, 2018), 15, 30, 61);
    put dtid1;
    dtid2=dhms(mdy(01, 03, 2018), 15, .5, 15);
    put dtid2;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
03JAN18:15:30:15
03JAN18:15:31:01
03JAN18:15:00:45
```

関連項目

概念:

- “DS2 の日付と時間” (*SAS DS2 プログラマガイド*)

関数:

- “HMS 関数” (693 ページ)

DIF 関数

引数とその *n* 番目のラグの差分を返します。

カテゴリ:	特殊
返されるデータの種類:	式のデータ型と同じ

構文

DIF[*n*] (*expression*)

必須引数

expression
数値に評価される有効な式を指定します。

データの種類	BIGINT、DECIMAL、DOUBLE、INTEGER、TINYINT
参照項目:	“DS2 式” (*SAS DS2 プログラマガイド*)

オプション引数

n
ラグ数を指定します。

範囲	1-100
データの種類	INTEGER

詳細

DIF 関数、DIF1、DIF2、...、DIF*n* は、式の値とその *n* 番目のラグの差分を返します。DIF1 は DIF と記述することもできます。DIF*n* は、DIF*n*(*x*)=*x*-LAG*n*(*x*)として定義されます。

注: データのアンダーフローまたはオーバーフローによる 予期しない動作を回避するために、式が SMALLINT または TINYINT の場合、DS2 は DIF 関数を処理する前

に式を INTEGER に変換します。処理後、DS2 は式を SMALLINT と TINYINT に変換し直し、他のすべての処理と同じ方法でアンダーフローまたはオーバーフローを 処理します。

LAG*n* キューからの値の保存と戻り値の詳細については、LAG 関数を参照してください。

比較

関数 DIF2(X)は、差分 DIF(DIF(X))とは異なります。

例

例 1

次のプログラムでは、LAG 関数と DIF 関数の違いを示します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data one (overwrite=yes);
   declare int x z d;
   method run();
     do x=1,2,6,4,7;
       z=lag(x);
       d=dif(x);
       output;
     end;
   end;
enddata;
run;
quit;

proc print data=one;
run;
```

アウトプット 7.8　DIF 関数と LAG 関数の違い

Obs	x	z	d
1	1	.	.
2	2	1	1
3	6	2	4
4	4	6	-2
5	7	4	3

例 2: 式を引数として使用する

次のプログラムは、文字列式がどのように double に変換されるかを示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl char(8) i difi;
  method init();
    do i=' 2', '  4', '   6', '10  ', 'abc';
      difi=dif(i);
      put i= difi=;
    end;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
i= 2      difi=.
i=  4     difi=2
i=   6    difi=2
i=10      difi=4
i=abc     difi=.
```

関連項目

関数:

- “LAG 関数” (794 ページ)

DIGAMMA 関数

ディガンマ関数の値を返します。

カテゴリ: 数学

返されるデータの種類: DOUBLE

構文

DIGAMMA(*expression*)

必須引数

expression
数値に評価される有効な式を指定します。

制限事項 ゼロおよび負の整数は無効です。

データの種類 DOUBLE

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

詳細

DIGAMMA 関数は、次の式で求められる比率を返します。

$$\Psi(x) = \Gamma'(x)/\Gamma(x)$$

$\Gamma(.)$および$\Gamma'(.)$は、それぞれガンマ関数とその導関数を示します。*expression*>0 の場合、DIGAMMA 関数は lgamma 関数の導関数です。

例

次のプログラムは、DIGAMMA 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double x;
```

```
  method run();
    x=digamma(1.0);
    put x;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
-0.57721566490153
```

DIM 関数

配列にある要素数を返します。

カテゴリ:	配列
返されるデータの種類:	INTEGER

構文

DIM(*array-name*[, *bound-n*])

必須引数

array-name
一時配列または変数配列の名前を指定します。

オプション引数

bound-n
多次元配列にある要素数を確認する次元を指定する、数値定数、変数または式です。*bound-n* 値が指定されていない場合、DIM 関数は配列の最初の次元の要素の数を返します。

bound-n は整数値に評価されます。

データの種類 INTEGER

詳細

DIM 関数は、1 次元配列にある要素数、または多次元配列内の指定した次元にある要素数を返します。

配列の次元外にある *bound-n* 次元値を使用して DIM 関数が呼び出された場合、ランタイムエラーが発生し、関数は NULL 整数値を返します。

比較

- DIM は、配列次元の要素数を返します。
- HBOUND は、配列次元の上限の値を返します。
- LBOUND は、配列次元の下限の値を返します。
- NDIMS は、配列の次元数を返します。

例

次のプログラムは、DIM、HBOUND、LBOUND、NDIMS 配列関数の使用方法を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;

  method init();
    declare char(15) a1[4];
    declare double   a2[2,3,4] sum;

    a1 := ('red' 'yellow' 'green' 'blue');
    a2 := (24*2.0);

    do i = 1 to dim(a1);
      put a1[i];
    end;

    numelems = 0;
    do i = 1 to ndims(a2);
      numelems = numelems + dim(a2, i);
    end;

    sum = 0;

    do i = lbound(a2, 1) to hbound(a2, 1);
     do j = lbound(a2, 2) to hbound(a2, 2);
       do k = lbound(a2, 3) to hbound(a2, 3);
         sum = sum + a2[i,j,k];
       end;
     end;
    end;

    put sum=;
```

```
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
red
yellow
green
blue
sum=48
```

関連項目

関数:

- “HBOUND 関数” (691 ページ)
- “LBOUND 関数” (801 ページ)
- “NDIMS 関数” (868 ページ)

DIVIDE 関数

ODS 出力の特殊欠損値を処理する除算の結果を返します。

カテゴリ: 算術

返されるデータの種類: DOUBLE

構文

DIVIDE(*x*, *y*)

必須引数

x
数値に評価される有効な式を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

y
数値に評価される有効な式を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

DIVIDE 関数は、2 つの数値を除算して ODS 規則に適合する結果を返します。この関数は、ODS 出力の特殊欠損値を処理します。次のリストに、特定の特殊欠損値が ODS で解釈される方法を示します。

- .I は無限大として解釈
- .M は負の無限大として解釈
- ._は空白として解釈

次の表に、*x* および *y* の値に基づいて DIVIDE 関数によって返される値を示します。

図 7.1 DIVIDE 関数によって返される値

		x						
		positive	zero	negative	.I	.M	._	other
y	positive	x/y or .I	0	x/y or .M	.I	.M	._	x
	zero	.I	.	.M	.I	.M	._	x
	negative	x/y or .M	0	x/y or .I	.M	.I	._	x
	.I	0	0	0	.	.	._	x
	.M	0	0	0	.	.	._	x
	._	._	._	._	._	._	._	._
	other	y	y	y	y	y	._	x

注: DIVIDE 関数は、欠損値、ゼロ除算またはオーバーフローに関するメモを SAS ログに書き込むことはありません。

例

次のプログラムでは、DIVIDE 関数を使用した結果を示します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double a b c d;
  method run();
    a=divide(1, 0);
    put a='(infinity)';
    b=divide(2, .I);
    put b=;
```

```
    c=divide(.I, -1);
    put c='(minus infinity)';
    d=divide(constant('big'), constant('small'));
    put d='(infinity because of overflow)';
  end;
enddata;
run;
quit;
```

次の行が SAS ログに書き込まれます。

```
a=I (infinity)
b=0
c=M (minus infinity)
d=I (infinity because of overflow)
```

DUR 関数

列挙キャッシュフローの修正デュレーションを返します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

DUR(*y*, *f*, *c(1), ..., c(k)*)

必須引数

y
期間当たりの有効満期利回りを指定します。分数で表します。

範囲 $y > 0$

データの種類 DOUBLE

f
期間当たりのキャッシュフローの頻度を指定します。

範囲 $f > 0$

データの種類 DOUBLE

c(1), ... ,c(k)
キャッシュフローのリストを指定します。

データの種類 DOUBLE

詳細

DUR 関数は、次の式から値を返します。

$$C = \sum_{k=1}^{K} \frac{k\left(\frac{c(k)}{(1+y)^{\frac{k}{f}}}\right)}{(P(1+y)f)}$$

前述の式には次の関係が適用されます。

$$P = \sum_{k=1}^{K} \frac{c(k)}{(1+y)^{\frac{k}{f}}}$$

例: DUR 関数の使用

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double d;
  method run();
    d=dur(.05,1,.33,.44,.55,.49,.50,.22,.4,.8,.01,.36,.2,.4);
    put d;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
5.28402498798216
```

関連項目

関数:

- “DURP 関数” (523 ページ)

DURP 関数

債権などの定期キャッシュフローストリームの修正デュレーションを返します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

DURP(*A*, *c*, *n*, *K*, k_0, *y*)

必須引数

A
額面価格を指定します。

範囲 $A > 0$

データの種類 DOUBLE

c
期間当たりの名目クーポンレートを分数で指定します。

範囲 $0 \le c < 1$

データの種類 DOUBLE

n
期間当たりのクーポン数を指定します。

範囲 $n > 0$ で整数

データの種類 DOUBLE

K
クーポンの残数を指定します。

範囲 $K > 0$ で整数

データの種類 DOUBLE

k_0
現在の日付から最初のクーポン日までの時間を指定します。期間数で表します。

範囲 $0 < k_0 \le 1/n$

データの種類 DOUBLE

y
期間当たりの名目有効満期利回りを指定します。分数で表します。

範囲 $y > 0$

データの種類 DOUBLE

詳細

DURP 関数は、次の式から値を返します。

$$D = \frac{1}{n}\frac{\sum_{k=1}^{K} t_k \frac{c(k)}{\left(1+\frac{y}{n}\right)^{t_k}}}{P\left(1+\frac{y}{n}\right)}$$

前述の式には次の関係が適用されます。

- $t_k = nk_0 + k - 1$
- $c(k) = \frac{c}{n}A \quad for\ k = 1, \ldots, K-1$
- $c(K) = \left(1+\frac{c}{n}\right)A$

前述の式には次の関係が適用されます。

$$P = \sum_{k=1}^{K} \frac{c(k)}{\left(1+\frac{y}{n}\right)^{t_k}}$$

例: DURP 関数の使用

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double d;
  method run();
    d=durp(1000,1/100,4,14,.33/2,.10);
    put d;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
3.33170731707317
```

関連項目

関数:

- “DUR 関数” (522 ページ)

EFFRATE 関数

有効な年利を返します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

EFFRATE(*compounding-interval*, *rate*)

必須引数

compounding-interval
SAS 間隔です。この値は、*rate* を複利計算する頻度を表します。

データの種類 CHAR

rate
数値です。*rate* は、各利息計算期間で複利計算する名目年利(パーセント)です。

データの種類 DOUBLE

詳細

EFFRATE 関数は、有効な年利を返します。この関数は、名目年利に対応する有効な年利を計算します。

EFFRATE 関数には次の詳細が適用されます。

- rate の値は–99 以上にする必要があります。
- 名目金利と利息計算期間を考えるに当たって、*compounding-interval* が 'CONTINUOUS'の場合、EFFRATE によって返される値は $e^{rate/100}–1$ です。

  compounding-interval が'CONTINUOUS'以外で、1 年に *m* 回の利息計算期間が含まれる場合、EFFRATE によって返される値は$(1+[rate/100\ m])^{m–1}$ です。
- *compounding-interval* で有効な値は次のとおりです。
  - 'CONTINUOUS'
  - 'DAY'
  - 'SEMIMONTH'
  - 'MONTH'

- 'QUARTER'
- 'SEMIYEAR'
- 'YEAR'

- 間隔が'DAY'の場合、*m*=365 です。

例

次のプログラムでは、有効な利率の計算方法を示します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

- このプログラムは、名目利率が 10%で利息が毎月複利計算される場合に対応する有効な利率を示します。

```
proc ds2;
data test(overwrite=yes);
  dcl double effectiverate1;
  method run();
    effectiverate1=effrate('month', 10);
    put effectiverate1;
  end;
enddata;
run;
quit;
```

有効な利率は 10.4713067441296 です。

- このプログラムは、名目利率が 10%で利息が四半期ごとに複利計算される場合に対応する有効な利率を示します。

```
proc ds2;
data test(overwrite=yes);
  dcl double effectiverate1;
  method run();
    effectiverate1=effrate('quarter', 10);
    put effectiverate1;
  end;
enddata;
run;
quit;
```

有効な利率は 10.3812890624999 です。

ERF 関数

(正規)誤差関数の値を返します。

カテゴリ: 数学

返されるデータの種類: DOUBLE

構文

ERF(*expression*)

必須引数

expression
数値に評価される有効な式を指定します。

データの種類 DOUBLE

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

詳細

ERF 関数は、次の式で求められる積分を返します。

$$\mathrm{ERF}(x) = \frac{2}{\sqrt{\pi}} \int_0^x \varepsilon^{-z^2} dz$$

ERF 関数を使用して、平均値が 0 で標準偏差が 1 の正規分布に従う確率変数の値が X よりも小さい確率(p)を求めることができます。たとえば、次のステートメントで求められる数量は PROBNORM(X)と同等です。

```
p=.5+.5*erf(x/sqrt(2));
```

例

次のプログラムは、ERF 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double x y;
  method run();
    x=erf(1);
    y=erf(-1);
    put x=;
    put y=;
  end;
enddata;
run;
```

```
quit;
```

SAS は次の出力をログに書き込みます。

```
x=0.84270079294971
y=-0.84270079294971
```

関連項目

関数:

- “ERFC 関数” (529 ページ)
- “PROBNORM 関数” (1010 ページ)

ERFC 関数

相補(正規)誤差関数の値を返します。

カテゴリ: 数学

返されるデータの種類: DOUBLE

構文

ERFC(*expression*)

必須引数

expression
数値に評価される有効な式を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

ERFC 関数は、ERF 関数(1 − ERF(*argument*))の補数を返します。

例

次のプログラムは、ERFC 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double x y;
  method run();
    x=erfc(1);
    y=erfc(-1);
    put x=;
    put y=;
 end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
x=0.15729920705028
y=1.84270079294971
```

関連項目

関数:

- “ERF 関数” (527 ページ)

EXP 関数

指定された累乗の e 定数の値を返します。

カテゴリ:	数学
返されるデータの種類:	DOUBLE

構文

EXP(*expression*)

必須引数

expression
数値に評価される有効な式を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

EXP 関数は、定数 **e** に対する指定した引数の累乗を返します。**e** は約 2.71828 で指定されます。この結果は、コンピューターの倍精度小数値の最大値によって制限されます。

例

例 1: EXP 関数の使用

次のプログラムは、EXP 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double x y;
  method run();
    x=exp(1.0);
    y=exp(0);
    put x=;
    put y=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
x=2.71828182845904
y=1
```

例 2: 行列パッケージ配列の入力

次のプログラムは、2 つの行列パッケージを作成し、それらに EXP 関数の値を入力します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double a[3, 3];
  dcl double c[3, 3];

  method run();
    dcl package matrix m;
    dcl package matrix r;
    dcl double i j;

    a := (1, 2, 3, 1, 2, 3, 1, 2, 3);

    m=_new_ matrix(a, 3, 3);
    r=m.exp();
    r.toarray(c);

    do i=1 to 3;
      do j=1 to 3;
        put c[i, j];
      end;
    end;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
2.71828182845904
7.38905609893065
20.0855369231876
2.71828182845904
7.38905609893065
20.0855369231876
2.71828182845904
7.38905609893065
20.0855369231876
```

FACT 関数

階乗を計算します。

カテゴリ:	数学
返されるデータの種類:	DOUBLE

構文

FACT(*expression*)

必須引数

expression
数値に評価される有効な式を指定します。

データの種類 DOUBLE

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

詳細

FACT 関数の数学的表現は、次の式によって得られます。

$$FACT(n) = n!$$

この式では、$n \geq 0$ です。

式による計算ができない場合は欠損値が返されます。やや大きな値の場合、FACT 関数を計算できないことがあります。

例

次のプログラムは、FACT 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data;
  dcl double x y z;
  method run();
     x=fact(5);
     /* negative #s don't work */
     y=fact(-27);
     /* decimal #s do not work */
     z=fact(7.3);
     put x=;
     put y=;
     put z=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
x=120
y=.
z=.
ERROR: Line 170: Invalid argument to function fact.
ERROR: Line 172: Invalid argument to function fact.
```

関連項目

関数:

- “COMB 関数” (427 ページ)
- “PERM 関数” (968 ページ)

FINANCE 関数

減価償却、満期、未払い利息、正味現在価値、定期的預金、内部利益率などの財務計算を行います。

カテゴリ: 財務

構文

FINANCE(*string-identifier*, *parameter–1*, *parameter–2*, ...)

必須引数

string-identifier
文字定数、変数または式を指定します。次の表に、*string-identifier* の有効な値を示します。

string-identifier	説明
'ACCRINT'	定期的に利息を支払う証券の未払い利息を計算します。
'ACCRINTM'	満期に利息を支払う証券の未払い利息を計算します。
' AMORDEGRC'	減価償却係数を使用して各会計期間の減価償却を計算します。
'AMORLINC'	各会計期間の減価償却を計算します。
'COUPDAYBS'	クーポン期間の開始日から決済日までの日数を計算します。
'COUPDAYS'	決済日を含むクーポン期間の日数を計算します。

string-identifier	説明
'COUPDAYSNC'	決済日から次のクーポン日までの日数を計算します。
'COUPNCD'	決済日後の次のクーポン日を計算します。
'COUPNUM'	決済日から満期日までの支払いクーポン数を計算します。
'COUPPCD'	決済日前の前のクーポン日を計算します。
'CUMIPMT'	2 つの期間の間に支払われる累積利息を計算します。
'CUMPRINC'	2 つの期間の間にローンで支払われる累積元本を計算します。
'DB'	定率法を使用して、指定期間の資産の減価償却を計算します。
'DDB'	倍額定率法またはその他の方法を指定して、指定期間の資産の減価償却を計算します。
'DISC'	証券の割引率を計算します。
'DOLLARDE'	分数で表すドル価格を小数で表すドル価格に変換します。
'DOLLARFR'	小数で表すドル価格を分数で表すドル価格に変換します。
'DURATION'	定期的に利息を支払う証券の年度期間を計算します。
'EFFECT'	有効な年利を計算します。
'FV'	投資の将来価値を計算します。
'FVSCHEDULE'	一連の複利率を適用した後の当初元本の将来価値を計算します。
'INTRATE'	完全投資済み証券の利率を計算します。
'IPMT'	一定期間の投資の利払いを計算します。
'IRR'	一連のキャッシュフローの内部利益率を計算します。
'ISPMT'	投資の指定期間中に支払われる利息を計算します。
'MDURATION'	推定額面$100 の証券の修正マコーレーデュレーションを計算します。

string-identifier	説明
'MIRR'	プラスおよびマイナスのキャッシュフローが異なる利率で資金調達される場合の内部利益率を計算します。
'NOMINAL'	年間の名目金利を計算します。
'NPER'	投資の期間数を計算します。
'NPV'	一連の定期的なキャッシュフローと割引率に基づいて投資の正味現在価値を計算します。
'ODDFPRICE'	端数の開始期間で証券の額面$100 当たりの価格を計算します。
'ODDFYIELD'	端数の開始期間で証券の利回りを計算します。
'ODDLPRICE'	端数の最終期間で証券の額面$100 当たりの価格を計算します。
'ODDLYIELD'	端数の最終期間で証券の利回りを計算します。
'PMT'	年金の定期的支払いを計算します。
'PPMT'	一定期間の投資の元本に対する支払いを計算します。
'PRICE'	定期的に利息を支払う証券の額面$100 当たりの価格を計算します。
'PRICEDISC'	割引証券の額面$100 当たりの価格を計算します。
'PRICEMAT'	満期に利息を支払う証券の額面$100 当たりの価格を計算します。
'PV'	投資の現在価値を計算します。
'RATE'	年金期間当たりの利率を計算します。
'RECEIVED'	完全投資済み証券の満期受け取り額を計算します。
'SLN'	ある期間の資産の定額法による減価償却を計算します。
'SYD'	指定期間の資産の年次級数和法による減価償却を計算します。
'TBILLEQ'	米国財務省短期証券の債券換算利回りを計算します。
'TBILLPRICE'	米国財務省短期証券の額面$100 当たりの価格を計算します。

string-identifier	説明
'TBILLYIELD'	米国財務省短期証券の利回りを計算します。
'VDB'	定率法を使用して、指定期間または一部期間の資産の減価償却を計算します。
'XIRR'	必ずしも定期的ではないキャッシュフロースケジュールの内部利益率を計算します。
'XNPV'	必ずしも定期的ではないキャッシュフロースケジュールの正味現在価値を計算します。
'YIELD'	定期的に利息を支払う証券の利回りを計算します。
'YIELDDISC'	割引証券(米国財務省短期証券など)の年間利回りを計算します。
'YIELDMAT'	満期に利息を支払う証券の年間利回りを計算します。

parameter
各 *string-identifier* に関連付けられたパラメーターを指定します。

次のパラメーターを使用できます。

basis
使用する日数計算基準の種類を示す文字または数値の値を示すパラメーターです(省略可能)。

数値	文字列値	日数計算方法
0	"30/360"	US (NASD) 30/360
1	"ACTUAL"	実日数/実日数
2	"ACT/360"	実日数/360
3	"ACT/365"	実日数/365
4	"EU30/360"	ヨーロッパ 30/360

interest-rates
パーセントではなく数値で示す利率を指定します。

dates
財務関数のすべての日付が SAS 日付であることを指定します。

sign-of-cash-values
すべての引数に対し、貯蓄預金口座への入金やその他の出金など、支払う金額を負の数値で指定します。また、配当小切手やその他の入金など、受け取る金額を正の数値で指定します。

FINANCE ACCRINT 関数

定期的に利息を支払う証券の未払い利息を計算します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

FINANCE('ACCRINT', *issue*, *first-interest*, *settlement*, *rate*, *par-value*, *frequency*, [*basis*]);

必須引数

issue
証券の発行日を指定します。

要件 *issue* は SAS 日付です。

データの種類 DOUBLE

first-interest
証券の最初の利息日を指定します。

要件 *first-interest* は SAS 日付です。

データの種類 DOUBLE

settlement
決済日を指定します。

要件 *settlement* は SAS 日付です。

データの種類 DOUBLE

rate
利率を指定します。

要件 *rate* は、パーセンテージではなく数値として提供されます。

データの種類 DOUBLE

par-value
証券の額面価格を指定します。*par-value* を省略すると、**$1000** という値が使用されます。

データの種類 DOUBLE

frequency
年間のクーポン支払い回数を指定します。年 1 回支払いの場合は *frequency*=1、年 2 回支払いの場合は *frequency*=2、年 4 回支払いの場合は *frequency*=4 となります。

データの種類 DOUBLE

オプション引数

basis
使用する日数計算基準の種類を示す文字または数値としてオプションのパラメーターを指定します。

数値	文字列値	日数計算方法
0	"30/360"	US (NASD) 30/360
1	"ACTUAL"	実日数/実日数
2	"ACT/360"	実日数/360
3	"ACT/365"	実日数/365
4	"EU30/360"	ヨーロッパ 30/360

データの種類 DOUBLE

例: 未払い利息を計算する: ACCRINT

次のプログラムでは、定期的に利息を支払う証券の未払い利息を計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double issue firstinterest settlement rate;
  dcl double par frequency basis r;
  method init();
    issue=mdy(2, 28, 2016);
    firstinterest=mdy(8, 31, 2016);
```

```
      settlement=mdy(5, 1, 2016);
      rate=0.1;
      par=1000;
      frequency=2;
      basis=1;
      r=finance('accrint', issue, firstinterest,
        settlement, rate, par, frequency, basis);
      put r=;
    end;
  enddata;
  run;
  quit;
```

SAS は次の出力をログに書き込みます。

```
r=17.1225513616818
```

FINANCE ACCRINTM 関数

満期に利息を支払う証券の未払い利息を計算します。

カテゴリ:	財務
返されるデータの種類:	DOUBLE

構文

FINANCE('ACCRINTM', *issue*, *settlement*, *rate*, *par-value*, [*basis*]);

必須引数

issue
証券の発行日を指定します。

要件	*issue* は SAS 日付です。
データの種類	DOUBLE

settlement
決済日を指定します。

要件	*settlement* は SAS 日付です。
データの種類	DOUBLE

rate
利率を指定します。

要件 *rate* は、パーセンテージではなく数値として提供されます。

データの種類 DOUBLE

par-value
証券の額面価格を指定します。*par-value* を省略すると、**$1000** という値が使用されます。

データの種類 DOUBLE

オプション引数

basis
使用する日数計算基準の種類を示す文字または数値としてオプションのパラメーターを指定します。

数値	文字列値	日数計算方法
0	"30/360"	US (NASD) 30/360
1	"ACTUAL"	実日数/実日数
2	"ACT/360"	実日数/360
3	"ACT/365"	実日数/365
4	"EU30/360"	ヨーロッパ 30/360

データの種類 DOUBLE

例: 未払い利息を計算する: ACCRINTM

次の例では、満期に利息を支払う証券の未払い利息を計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double issue maturity rate;
  dcl double par basis r;
  method init();
    issue=mdy(2, 28, 2015);
    maturity=mdy(8, 31, 2015);
    rate=0.1;
    par=1000;
    basis=0;
```

```
      r=finance('accrintm', issue, maturity, rate, par, basis);
      put r=;
    end;
  enddata;
  run;
  quit;
```

SAS は次の出力をログに書き込みます。

```
r=50.5555555555555
```

FINANCE AMORDEGRC 関数

減価償却係数を使用して各会計期間の減価償却を計算します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

FINANCE('AMORDEGRC', *cost*, *date-purchased*, *first-period*, *salvage*, *period*, *rate*, [*basis*]);

必須引数

cost
資産の初期コストを指定します。

データの種類 DOUBLE

date-purchased
資産の購入日を指定します。

要件 *date-purchased* は SAS 日付です。

データの種類 DOUBLE

first-period
第 1 期間の最終日を指定します。

要件 *first-period* は SAS 日付です。

データの種類 DOUBLE

salvage
減価償却を終了した時点の価値(資産の救済価額とも呼ばれる)を指定します。

データの種類 DOUBLE

period
減価償却期間を指定します。

データの種類 DOUBLE

rate
減価償却率を指定します。

要件 *rate* は、パーセンテージではなく数値として提供されます。

データの種類 DOUBLE

オプション引数

basis
使用する日数計算基準の種類を示す文字または数値としてオプションのパラメーターを指定します。

数値	文字列値	日数計算方法
0	"30/360"	US (NASD) 30/360
1	"ACTUAL"	実日数/実日数
2	"ACT/360"	実日数/360
3	"ACT/365"	実日数/365
4	"EU30/360"	ヨーロッパ 30/360

ヒント FINANCE 関数の第 1 引数が AMORDEGR で、*basis* の値が 2 の場合、欠損値が返されます。

データの種類 DOUBLE

例: 減価償却を計算する: AMORDEGRC

次のプログラムでは、減価償却係数を使用して各会計期間の減価償却を計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
```

```
data test(overwrite=yes);
  dcl double cost datepurchased firstperiod salvage;
  dcl double period rate basis r;
  method init();
    cost=2400;
    datepurchased=mdy(8, 19, 2016);
    firstperiod=mdy(12, 31, 2016);
    salvage=300;
    period=1;
    rate=0.15;
    basis=1;
    r=finance('amordegrc', cost, datepurchased,
      firstperiod, salvage, period, rate, basis);
    put r=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
r=776
```

FINANCE AMORLINC 関数

各会計期間の減価償却を計算します。

カテゴリ:	財務
返されるデータの種類:	DOUBLE

構文

FINANCE('AMORLINC', *cost*, *date-purchased*, *first-period*, *salvage*, *period*, *rate*, [*basis*]);

必須引数

cost
資産の初期コストを指定します。

データの種類 DOUBLE

date-purchased
資産の購入日を指定します。

要件 *date-purchased* は SAS 日付です。

データの種類 DOUBLE

first-period
第 1 期間の最終日を指定します。

要件 *first-period* は SAS 日付です。

データの種類 DOUBLE

salvage
減価償却を終了した時点の価値(資産の救済価額とも呼ばれる)を指定します。

データの種類 DOUBLE

period
減価償却期間を指定します。

データの種類 DOUBLE

rate
減価償却率を指定します。

要件 *rate* は、パーセンテージではなく数値として提供されます。

データの種類 DOUBLE

オプション引数

basis
使用する日数計算基準の種類を示す文字または数値としてオプションのパラメーターを指定します。

数値	文字列値	日数計算方法
0	"30/360"	US (NASD) 30/360
1	"ACTUAL"	実日数/実日数
2	"ACT/360"	実日数/360
3	"ACT/365"	実日数/365
4	"EU30/360"	ヨーロッパ 30/360

ヒント FINANCE 関数の第 1 引数が AMORLINC で、*basis* の値が 2 の場合、欠損値が返されます。

データの種類 DOUBLE

例: 計算の説明: AMORLINC

次のプログラムでは、各会計期間の減価償却を計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double cost dp fp salvage;
  dcl double period rate basis r;
  method init();
    cost=2400;
    dp=mdy(9, 30, 2016);
    fp=mdy(12, 31, 2016);
    salvage=245;
    period=0;
    rate=0.115;
    basis=0;
    r=finance('amorlinc', cost, dp, fp, salvage, period, rate, basis);
    put r=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
r=69
```

FINANCE COUPDAYBS 関数

クーポン期間の開始日から決済日までの日数を計算します。

カテゴリ:	財務
返されるデータの種類:	DOUBLE

構文

FINANCE('COUPDAYBS', *settlement*, *maturity*, *frequency*, [*basis*]);

必須引数

settlement
証券の決済日を指定します。証券決済日は、証券が買い手に対して売買される発行日の翌日になります。

要件 *settlement* は SAS 日付です。

データの種類 DOUBLE

maturity
証券の満期日を指定します。満期日は証券の有効期限が切れる日です。

要件 *maturity* は SAS 日付です。

データの種類 DOUBLE

frequency
年間のクーポン支払い回数を指定します。年 1 回支払いの場合は *frequency*=1、年 2 回支払いの場合は *frequency*=2、年 4 回支払いの場合は *frequency*=4 となります。

データの種類 DOUBLE

オプション引数

basis
使用する日数計算基準の種類を示す文字または数値としてオプションのパラメーターを指定します。

数値	文字列値	日数計算方法
0	"30/360"	US (NASD) 30/360
1	"ACTUAL"	実日数/実日数
2	"ACT/360"	実日数/360
3	"ACT/365"	実日数/365
4	"EU30/360"	ヨーロッパ 30/360

データの種類 DOUBLE

例: 計算の説明: COUPDAYBS

次のプログラムでは、クーポン期間の開始日から決済日までの日数を計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double settlement maturity frequency basis r;
  method init();
    settlement=mdy(12,30,2013);
    maturity=mdy(11,29,2016);
    frequency=4;
    basis=2;
    r=finance('coupdaybs', settlement, maturity, frequency, basis);
    put r=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
r=31
```

注: 日付は DATE 関数を使用するか、他の計算式や関数の結果として入力する必要があります。たとえば、2016 年 5 月 23 日には DATE(2016,5,23)を使用します。日付をテキストとして入力すると問題が発生する場合があります。

FINANCE COUPDAYS 関数

決済日を含むクーポン期間の日数を計算します。

カテゴリ:	財務
返されるデータの種類:	DOUBLE

構文

FINANCE('COUPDAYS', *settlement*, *maturity*, *frequency*, [*basis*]);

必須引数

settlement
決済日を指定します。

要件	*settlement* は SAS 日付です。
データの種類	DOUBLE

maturity
満期日を指定します。

要件 *maturity* は SAS 日付です。

データの種類 DOUBLE

frequency
年間のクーポン支払い回数を指定します。年 1 回支払いの場合は *frequency*=1、年 2 回支払いの場合は *frequency*=2、年 4 回支払いの場合は *frequency*=4 となります。

データの種類 DOUBLE

オプション引数

basis
使用する日数計算基準の種類を示す文字または数値としてオプションのパラメーターを指定します。

数値	文字列値	日数計算方法
0	"30/360"	US (NASD) 30/360
1	"ACTUAL"	実日数/実日数
2	"ACT/360"	実日数/360
3	"ACT/365"	実日数/365
4	"EU30/360"	ヨーロッパ 30/360

データの種類 DOUBLE

例: 計算の説明: COUPDAYS

次のプログラムでは、決済日を含むクーポン期間の日数を計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double settlement maturity frequency basis r;
  method init();
    settlement=mdy(1,25,2015);
    maturity=mdy(11,15,2016);
```

```
      frequency=2;
      basis=1;
      r=finance('coupdays', settlement, maturity, frequency, basis);
      put r=;
    end;
  enddata;
  run;
  quit;
```

SAS は次の出力をログに書き込みます。

```
r=181
```

FINANCE COUPDAYSNC 関数

決済日から次のクーポン日までの日数を計算します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

FINANCE('COUPDAYSNC', *settlement*, *maturity*, *frequency*, [*basis*]);

必須引数

settlement
決済日を指定します。

要件 *settlement* は SAS 日付です。

データの種類 DOUBLE

maturity
満期日を指定します。

要件 *maturity* は SAS 日付です。

データの種類 DOUBLE

frequency
年間のクーポン支払い回数を指定します。年 1 回支払いの場合は *frequency*=1、年 2 回支払いの場合は *frequency*=2、年 4 回支払いの場合は *frequency*=4 となります。

データの種類 DOUBLE

オプション引数

basis
使用する日数計算基準の種類を示す文字または数値としてオプションのパラメーターを指定します。

数値	文字列値	日数計算方法
0	"30/360"	US (NASD) 30/360
1	"ACTUAL"	実日数/実日数
2	"ACT/360"	実日数/360
3	"ACT/365"	実日数/365
4	"EU30/360"	ヨーロッパ 30/360

データの種類 DOUBLE

例: 計算の説明: COUPDAYSNC

次のプログラムでは、決済日から次のクーポン日までの日数を計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double settlement maturity frequency basis r;
  method init();
    settlement=mdy(1,25,2007);
    maturity=mdy(11,15,2008);
    frequency=2;
    basis=1;
    r=finance('coupdaysnc', settlement, maturity, frequency, basis);
    put r=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
r=110
```

FINANCE COUPNCD 関数

決済日後の次のクーポン日を計算します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

FINANCE('COUPNCD', *settlement*, *maturity*, *frequency*, [*basis*]);

必須引数

settlement
決済日を指定します。

要件 *settlement* は SAS 日付です。

データの種類 DOUBLE

maturity
満期日を指定します。

要件 *maturity* は SAS 日付です。

データの種類 DOUBLE

frequency
年間のクーポン支払い回数を指定します。年 1 回支払いの場合は *frequency*=1、年 2 回支払いの場合は *frequency*=2、年 4 回支払いの場合は *frequency*=4 となります。

データの種類 DOUBLE

オプション引数

basis
使用する日数計算基準の種類を示す文字または数値としてオプションのパラメーターを指定します。

数値	文字列値	日数計算方法
0	"30/360"	US (NASD) 30/360
1	"ACTUAL"	実日数/実日数

数値	文字列値	日数計算方法
2	"ACT/360"	実日数/360
3	"ACT/365"	実日数/365
4	"EU30/360"	ヨーロッパ 30/360

データの種類 DOUBLE

例: 計算の説明: COUPNCD

次のプログラムでは、決済日後の次のクーポン日を計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double settlement maturity frequency basis r;
  method init();
    settlement=mdy(1, 25, 2007);
    maturity=mdy(11, 15, 2008);
    frequency=2;
    basis=1;
    r=finance('coupncd', settlement, maturity, frequency, basis);
    put r date7.;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
15MAY07
```

注: *r* は数値の SAS 値で、DATE7 形式を使用して出力できます。

FINANCE COUPNUM 関数

決済日から満期日までの支払いクーポン数を計算します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

FINANCE('COUPNUM', *settlement*, *maturity*, *frequency*, [*basis*]);

必須引数

settlement
決済日を指定します。

要件 *settlement* は SAS 日付です。

データの種類 DOUBLE

maturity
満期日を指定します。

要件 *maturity* は SAS 日付です。

データの種類 DOUBLE

frequency
年間のクーポン支払い回数を指定します。年 1 回支払いの場合は *frequency*=1、年 2 回支払いの場合は *frequency*=2、年 4 回支払いの場合は *frequency*=4 となります。

データの種類 DOUBLE

オプション引数

basis
使用する日数計算基準の種類を示す文字または数値としてオプションのパラメーターを指定します。

数値	文字列値	日数計算方法
0	"30/360"	US (NASD) 30/360
1	"ACTUAL"	実日数/実日数
2	"ACT/360"	実日数/360
3	"ACT/365"	実日数/365
4	"EU30/360"	ヨーロッパ 30/360

データの種類 DOUBLE

例: 計算の説明: COUPNUM

次のプログラムでは、決済日から満期日までの支払いクーポン数を計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double settlement maturity frequency basis r;
  method init();
    settlement=mdy(1,25,2015);
    maturity=mdy(11,15,2016);
    frequency=2;
    basis=1;
    r=finance('coupnum', settlement, maturity, frequency, basis);
    put r=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
r=4
```

FINANCE COUPPCD 関数

決済日前の前のクーポン日を計算します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

FINANCE('COUPPCD', *settlement*, *maturity*, *frequency*, [*basis*]);

必須引数

settlement
決済日を指定します。

要件　*settlement* は SAS 日付です。

データの種類　DOUBLE

maturity
満期日を指定します。

要件　*maturity* は SAS 日付です。

データの種類　DOUBLE

frequency
年間のクーポン支払い回数を指定します。年 1 回支払いの場合は *frequency*=1、年 2 回支払いの場合は *frequency*=2、年 4 回支払いの場合は *frequency*=4 となります。

データの種類　DOUBLE

オプション引数

basis
使用する日数計算基準の種類を示す文字または数値としてオプションのパラメーターを指定します。

数値	文字列値	日数計算方法
0	"30/360"	US (NASD) 30/360
1	"ACTUAL"	実日数/実日数
2	"ACT/360"	実日数/360
3	"ACT/365"	実日数/365
4	"EU30/360"	ヨーロッパ 30/360

データの種類　DOUBLE

例: 計算の説明: COUPPCD

次のプログラムでは、決済日前の前のクーポン日を計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
```

```
  dcl double settlement maturity frequency basis r;
  method init();
    settlement=mdy(1, 25, 2007);
    maturity=mdy(11, 15, 2008);
    frequency=2;
    basis=1;
    r=finance('couppcd', settlement, maturity, frequency, basis);
    put r date7.;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
15NOV06
```

FINANCE CUMIPMT 関数

2 つの期間の間に支払われる累積利息を計算します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

FINANCE('CUMIPMT', *rate*, *nper*, *pv*, *start-period*, *end-period*, [*type*]);

必須引数

rate
利率を指定します。

要件 *rate* は、パーセンテージではなく数値として提供されます。

データの種類 DOUBLE

nper
支払い期間の総数を指定します。

データの種類 DOUBLE

pv
現在価値、または一連の将来の支払いに相当する現時点での一括払い金額を指定します。

データの種類 DOUBLE

start-period
計算の開始期間を指定します。支払い期間は 1 から番号が付けられます。

データの種類 DOUBLE

end-period
計算の最終期間を指定します。

データの種類 DOUBLE

オプション引数

type
0 か 1 を指定し、支払い期日がいつであるかを示します。*type* を省略すると、0 とみなされます。

支払い期日が期末の場合は、*type* 引数を省略するか、0 に設定します。支払い期日が期首の場合は、*type* を 1 に設定します。

データの種類 DOUBLE

例: 計算の説明: CUMIPMT

次のプログラムでは、2 つの期間の間に支払われる累積利息を計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double rate nper pv startperiod endperiod type r;
  method init();
    rate=0.09;
    nper=30;
    pv=125000;
    startperiod=13;
    endperiod=24;
    type=0;
    r=finance('cumipmt', rate, nper, pv, startperiod, endperiod, type);
    put r=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
r=-94054.8203251095
```

FINANCE CUMPRINC 関数

2 つの期間の間にローンで支払われる累積元本を計算します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

FINANCE('CUMPRINC', *rate*, *nper*, *pv*, *start-period*, *end-period*, [*type*]);

必須引数

rate
利率を指定します。

要件 *rate* は、パーセンテージではなく数値として提供されます。

データの種類 DOUBLE

nper
支払い期間の総数を指定します。

データの種類 DOUBLE

pv
現在価値、または一連の将来の支払いに相当する現時点での一括払い金額を指定します。

データの種類 DOUBLE

start-period
計算の開始期間を指定します。支払い期間は 1 から番号が付けられます。

データの種類 DOUBLE

end-period
計算の最終期間を指定します。

データの種類 DOUBLE

オプション引数

type
0 か 1 を指定し、支払い期日がいつであるかを示します。*type* を省略すると、0 とみなされます。

支払い期日が期末の場合は、*type* 引数を省略するか、0 に設定します。支払い期日が期首の場合は、*type* を 1 に設定します。

データの種類　DOUBLE

例: 計算の説明: CUMPRINC

次のプログラムでは、2 つの期間の間に支払われる累積利息を計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double rate nper pv startperiod endperiod type r;
  method init();
    rate=0.09;
    nper=30;
    pv=125000;
    startperiod=13;
    endperiod=24;
    type=0;
    r=finance('cumprinc', rate, nper, pv, startperiod, endperiod, type);
    put r=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
r=-51949.7067612251
```

FINANCE DB 関数

定率法を使用して、指定期間の資産の減価償却を計算します。

カテゴリ:	財務
返されるデータの種類:	DOUBLE

構文

FINANCE('DB', *cost*, *salvage*, *life*, *period*, [*month*]);

必須引数

cost
資産の初期コストを指定します。

データの種類 DOUBLE

salvage
減価償却を終了した時点の価値(資産の救済価額とも呼ばれる)を指定します。

データの種類 DOUBLE

life
資産が減価償却される期間数(資産の耐用年数とも呼ばれる)を指定します。

データの種類 DOUBLE

period
減価償却を計算する期間を指定します。*Period* には、*life* と同じ時間単位を使用する必要があります。

データの種類 DOUBLE

オプション引数

month
月数を指定します(month は数値引数で、省略可能です)。month を省略すると、デフォルト値 12 が使用されます。

データの種類 DOUBLE

例: 計算の説明: DB

次のプログラムでは、定率法を使用して、指定した期間の資産の減価償却を計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double cost salvage life period month r;
  method init();
    cost=1000000;
    salvage=100000;
    life=6;
    period=2;
    month=7;
    r=finance('db', cost, salvage, life, period, month);
    put r=;
  end;
```

```
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
r=259639.416666666
```

FINANCE DDB 関数

倍額定率法またはその他の方法を指定して、指定期間の資産の減価償却を計算します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

FINANCE('DDB', *cost*, *salvage*, *life*, *period*, [*factor*]);

必須引数

cost
資産の初期コストを指定します。

データの種類 DOUBLE

salvage
減価償却を終了した時点の価値(資産の救済価額とも呼ばれる)を指定します。

データの種類 DOUBLE

life
資産が減価償却される期間数(資産の耐用年数とも呼ばれる)を指定します。

データの種類 DOUBLE

period
減価償却を計算する期間を指定します。*Period* には、*life* と同じ時間単位を使用する必要があります。

データの種類 DOUBLE

オプション引数

factor
減価償却率を指定します。*factor* を省略すると、2 とみなされます(倍額定率法)。

データの種類 DOUBLE

例: 計算の説明: DDB

次のプログラムでは、倍額定率法またはその他の方法を指定して、指定した期間の資産の減価償却を計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double cost salvage life period factor r;
  method init();
    cost=2400;
    salvage=300;
    life=10*365;
    period=1;
    factor=.;
    r=finance('ddb', cost, salvage, life, period, factor);
    put r=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
r=1.31506849315068
```

FINANCE DISC 関数

証券の割引率を計算します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

FINANCE('DISC', *settlement*, *maturity*, *price*, *redemption*, [*basis*]);

必須引数

settlement
決済日を指定します。

要件 *settlement* は SAS 日付です。

データの種類 DOUBLE

maturity
満期日を指定します。

要件 *maturity* は SAS 日付です。

データの種類 DOUBLE

price
額面$100 当たりの証券の価格を指定します。

データの種類 DOUBLE

redemption
満期受け取り額を指定します。

データの種類 DOUBLE

オプション引数

basis
使用する日数計算基準の種類を示す文字または数値としてオプションのパラメーターを指定します。

数値	文字列値	日数計算方法
0	"30/360"	US (NASD) 30/360
1	"ACTUAL"	実日数/実日数
2	"ACT/360"	実日数/360
3	"ACT/365"	実日数/365
4	"EU30/360"	ヨーロッパ 30/360

データの種類 DOUBLE

例: 計算の説明: DISC

次のプログラムでは、証券の割引率を計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double settlement maturity price redemption basis r;
  method init();
    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=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
r=0.05242021276595
```

FINANCE DOLLARDE 関数

分数で表すドル価格を小数で表すドル価格に変換します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

FINANCE('DOLLARDE', *fractional-dollar*, *fraction*);

必須引数

fractional-dollar
小数で表す数値を指定します。

データの種類 DOUBLE

fraction
分母に使用する整数を指定します。

データの種類 DOUBLE

例: 計算の説明: DOLLARDE

次のプログラムでは、分数で表すドル価格を小数で表すドル価格に変換します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double fractionaldollar fraction r;
  method init();
    fractionaldollar=1.125;
    fraction=16;
    r=finance('dollarde', fractionaldollar, fraction);
    put r=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
r=1.78125
```

FINANCE DOLLARFR 関数

小数で表すドル価格を分数で表すドル価格に変換します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

FINANCE('DOLLARFR', *decimal-dollar*, *fraction*);

必須引数

decimal-dollar
小数を指定します。

データの種類 DOUBLE

fraction
分母に使用する整数を指定します。

データの種類 DOUBLE

例: 計算の説明: DOLLARFR

次のプログラムでは、小数で表すドル価格を分数で表すドル価格に変換します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double decimaldollar fraction r;
  method init();
    decimaldollar=1.125;
    fraction=16;
    r=finance('dollarfr', decimaldollar, fraction);
    put r=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
r=1.02
```

分数形式では、*r* の値は次のようになります。$1\frac{2}{16}$

FINANCE DURATION 関数

定期的に利息を支払う証券の年度期間を計算します。

カテゴリ:	財務
返されるデータの種類:	DOUBLE

構文

FINANCE('DURATION', *settlement*, *maturity*, *coupon*, *yield*, *frequency*, [*basis*]);

必須引数

settlement
決済日を指定します。

要件 *settlement* は SAS 日付です。

データの種類 DOUBLE

maturity
満期日を指定します。

要件 *maturity* は SAS 日付です。

データの種類 DOUBLE

coupon
証券の年間のクーポン率を指定します。

要件 *coupon* は、パーセンテージではなく数値として提供されます。

データの種類 DOUBLE

yield
証券の年間利回りを指定します。

データの種類 DOUBLE

frequency
年間のクーポン支払い回数を指定します。年 1 回支払いの場合は *frequency*=1、年 2 回支払いの場合は *frequency*=2、年 4 回支払いの場合は *frequency*=4 となります。

データの種類 DOUBLE

オプション引数

basis
使用する日数計算基準の種類を示す文字または数値としてオプションのパラメーターを指定します。

数値	文字列値	日数計算方法
0	"30/360"	US (NASD) 30/360
1	"ACTUAL"	実日数/実日数
2	"ACT/360"	実日数/360

数値	文字列値	日数計算方法
3	"ACT/365"	実日数/365
4	"EU30/360"	ヨーロッパ 30/360

データの種類 DOUBLE

例: 計算の説明: DURATION

次のプログラムでは、定期的に利息を支払う証券の年度期間を計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double settlement maturity couponrate yield
    frequency basis r;
    method init();
    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=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
r=5.99377495554518
```

FINANCE EFFECT 関数

有効な年利を計算します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

FINANCE('EFFECT', *nominal-rate*, *npery*);

必須引数

nominal-rate
名目金利を指定します。

データの種類 DOUBLE

npery
年間の複利計算期間数を指定します。

データの種類 DOUBLE

例: 計算の説明: EFFECT

次のプログラムでは、有効な年利を計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double nominalrate npery r;
  method init();
    nominalrate=0.0525;
    npery=4;
    r=finance('effect', nominalrate, npery);
    put r=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
r=0.05354266737075
```

FINANCE FV 関数

投資の将来価値を計算します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

FINANCE('FV', *rate*, *nper*, [*payment*], [*present-value*], [*type*]);

必須引数

rate
利率を指定します。

要件 *rate* は、パーセンテージではなく数値として提供されます。

データの種類 DOUBLE

nper
支払い期間の総数を指定します。

データの種類 DOUBLE

オプション引数

payment
各期間に行われる支払いを指定します。年金の有効期間中に支払いが変更されることはありません。通常、*payment* には元本と利息が含まれますが、手数料や税金は含まれません。*payment* を省略する場合、*present-value* 引数を含める必要があります。

データの種類 DOUBLE

present-value
現在価値、または一連の将来の支払いに相当する現時点での一括払い金額を指定します。*present-value* を省略すると、0(ゼロ)とみなされ、*payment* 引数を含める必要があります。

データの種類 DOUBLE

type
0 か 1 を指定し、支払い期日がいつであるかを示します。*type* を省略すると、0 とみなされます。

支払い期日が期末の場合は、*type* 引数を省略するか、0 に設定します。支払い期日が期首の場合は、*type* を 1 に設定します。

データの種類　DOUBLE

例: 計算の説明: FV

次のプログラムでは、投資の将来価値を計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double rate nper payment present_value type r;
  method init();
    rate=0.06/12;
    nper=10;
    payment=-200;
    present_value=-500;
    type=1;
    r=finance('fv', rate, nper, payment, present_value, type);
    put r=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
r=2581.40337406018
```

FINANCE FVSCHEDULE 関数

一連の複利率を適用した後の当初元本の将来価値を計算します。

カテゴリ:　財務

返されるデータの種類:　DOUBLE

構文

FINANCE('FVSCHEDULE', *principal*, *schedule-1*, *schedule-2* ...);

必須引数

principal
現在価値を指定します。

データの種類 DOUBLE

schedule
適用する一連の利率を指定します。

要件 *schedule* 率は、パーセンテージではなく数値として提供されます。

データの種類 DOUBLE

例: 計算の説明: FVSCHEDULE

次のプログラムでは、一連の複利率を適用した後の当初元本の将来価値を計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double principal r1 r2 r3 r;
  method init();
    principal=1;
    r1=0.09;
    r2=0.11;
    r3=0.1;
    r=finance('fvschedule', principal, r1, r2, r3);
    put r=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
r=1.33089
```

FINANCE INTRATE 関数

完全投資済み証券の利率を計算します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

FINANCE('INTRATE', *settlement*, *maturity*, *investment*, *redemption*, [*basis*]);

必須引数

settlement
決済日を指定します。

要件 *settlement* は SAS 日付です。

データの種類 DOUBLE

maturity
満期日を指定します。

要件 *maturity* は SAS 日付です。

データの種類 DOUBLE

investment
証券に投資される金額を指定します。

データの種類 DOUBLE

redemption
満期受け取り額を指定します。

データの種類 DOUBLE

オプション引数

basis
使用する日数計算基準の種類を示す文字または数値としてオプションのパラメーターを指定します。

数値	文字列値	日数計算方法
0	"30/360"	US (NASD) 30/360
1	"ACTUAL"	実日数/実日数
2	"ACT/360"	実日数/360
3	"ACT/365"	実日数/365
4	"EU30/360"	ヨーロッパ 30/360

データの種類 DOUBLE

例: 計算の説明: INTRATE

次のプログラムでは、完全投資済み証券の利率を計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double settlement maturity investment redemption basis r;
  method init();
    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=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
r=0.05768
```

FINANCE IPMT 関数

一定期間の投資の利払いを計算します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

FINANCE('IPMT', *rate*, *period*, *nper*, *pv*, [*fv*], [*type*]);

必須引数

rate
利率を指定します。

要件 *rate* は、パーセンテージではなく数値として提供されます。

データの種類 DOUBLE

period
減価償却を計算する期間を指定します。*Period* には、*life* と同じ単位を使用する必要があります。

データの種類 DOUBLE

nper
支払い期間の総数を指定します。

データの種類 DOUBLE

pv
現在価値、または一連の将来の支払いに相当する現時点での一括払い金額を指定します。*pv* を省略すると、0(ゼロ)とみなされ、*fv* 引数を含める必要があります。

データの種類 DOUBLE

オプション引数

fv
将来価値、または最終支払い後に達する現金残高を指定します。*fv* を省略すると、0(ゼロ)とみなされます(たとえば、ローンの将来価値が 0)。

データの種類 DOUBLE

type
0 か 1 を指定し、支払い期日がいつであるかを示します。*type* を省略すると、0 とみなされます。

支払い期日が期末の場合は、*type* 引数を省略するか、0 に設定します。支払い期日が期首の場合は、*type* を 1 に設定します。

データの種類 DOUBLE

例: 計算の説明: IPMT

次のプログラムでは、指定期間の投資の利払いを計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
```

```
data test(overwrite=yes);
  dcl double rate per nper pv fv type r;
  method init();
    rate=0.1/12;
    per=2;
    nper=3;
    pv=100;
    fv=.;
    type=.;
    r=finance('ipmt', rate, per, nper, pv, fv, type);
    put r=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
r=-0.5578575637229
```

FINANCE IRR 関数

一連のキャッシュフローの内部利益率を計算します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

FINANCE('IRR', *value-1*, *value-2*, ..., *value-n*);

必須引数

value
内部利益率を計算する対象の数値を含む数値引数のリストを指定します。

データの種類 DOUBLE

例: 計算の説明: IRR

次のプログラムでは、一連のキャッシュフローの内部利益率を計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double v1 v2 v3 v4 v5 v6 r;
  method init();
    v1=-70000;
    v2=12000;
    v3=15000;
    v4=18000;
    v5=21000;
    v6=26000;
    r=finance('irr', v1, v2, v3, v4, v5, v6);
    put r=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
r=0.08663094803653
```

FINANCE ISPMT 関数

投資の指定期間中に支払われる利息を計算します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

FINANCE('ISPMT', *interest-rate*, *period*, *number-payments*, *pv*);

必須引数

interest-rate
投資の利率を指定します。

要件 *rate* は、パーセンテージではなく数値として提供されます。

データの種類 DOUBLE

period
利率を計算する期間を指定します。*Period* は 1 から *number-payments* までの間の値にする必要があります。

データの種類 DOUBLE

number-payments
年金の支払い回数を指定します。

データの種類 DOUBLE

pv
ローン金額または支払いの現在価値を指定します。

データの種類 DOUBLE

例: 計算の説明: ISPMT

次のプログラムでは、年間 7.5%を得る$5000 の投資の 2 年間の利払いを計算します。利息の支払いは 8 か月目に計算されます。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double interest;
  method init();
    interest=finance('ispmt', 0.075/12, 8, 2*12, 5000);
    put interest=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
interest=-20.8333333333333
```

FINANCE MDURATION 関数

推定額面$100 の証券の修正マコーレーデュレーションを計算します。

カテゴリ:	財務
返されるデータの種類:	DOUBLE

構文

FINANCE('MDURATION', *settlement*, *maturity*, *coupon*, *yield*, *frequency*, [*basis*]);

必須引数

settlement
決済日を指定します。

要件 *settlement* は SAS 日付です。

データの種類 DOUBLE

maturity
満期日を指定します。

要件 *maturity* は SAS 日付です。

データの種類 DOUBLE

coupon
証券の年間のクーポン率を指定します。

要件 *coupon* は、パーセンテージではなく数値として提供されます。

データの種類 DOUBLE

yield
証券の年間利回りを指定します。

データの種類 DOUBLE

frequency
年間のクーポン支払い回数を指定します。年 1 回支払いの場合は *frequency*=1、年 2 回支払いの場合は *frequency*=2、年 4 回支払いの場合は *frequency*=4 となります。

データの種類 DOUBLE

オプション引数

basis
使用する日数計算基準の種類を示す文字または数値としてオプションのパラメーターを指定します。

数値	文字列値	日数計算方法
0	"30/360"	US (NASD) 30/360
1	"ACTUAL"	実日数/実日数
2	"ACT/360"	実日数/360

数値	文字列値	日数計算方法
3	"ACT/365"	実日数/365
4	"EU30/360"	ヨーロッパ 30/360

データの種類 DOUBLE

例: 計算の説明: MDURATION

次のプログラムでは、推定額面$100 の証券の修正マコーレーデュレーションを計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double settlement maturity couponrate yield frequency basis r;
  method init();
    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=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
r=5.73566981391883
```

FINANCE MIRR 関数

プラスおよびマイナスのキャッシュフローが異なる利率で資金調達される場合の内部利益率を計算します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

FINANCE('MIRR', *value-1*, ..., *value-n*, *finance-rate*, *reinvest-rate*);

必須引数

value
数値を含む数値引数のリストを指定します。これらの数値は、定期的に発生する一連の支払い(負の値)と収入(正の値)を表します。*value* は、修正内部利益率を計算するためには、正の値と負の値が少なくとも 1 つずつ含まれている必要があります。

データの種類 DOUBLE

finance-rate
キャッシュフローで使用する資金に対して支払う利率を指定します。

要件 *finance-rate* は、パーセンテージではなく数値として提供されます。

データの種類 DOUBLE

reinvest-rate
投資するときにキャッシュフローに対して受け取る利率を指定します。

要件 *reinvest-rate* は、パーセンテージではなく数値として提供されます。

データの種類 DOUBLE

例: 計算の説明: MIRR

次のプログラムでは、プラスおよびマイナスのキャッシュフローが異なる利率で資金調達される場合の内部利益率を計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double v1 v2 v3 v4 financerate reinvestrate r;
  method init();
    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=;
    end;
  enddata;
  run;
  quit;
```

SAS は次の出力をログに書き込みます。

```
r=1.35314201717001
```

FINANCE NOMINAL 関数

年間の名目金利を計算します。

カテゴリ:	財務
返されるデータの種類:	DOUBLE

構文

FINANCE('NOMINAL', *effective-rate*, *npery*);

必須引数

effective-rate
有効な利率を指定します。

要件	*effective-rate* は、パーセンテージではなく数値として提供されます。
データの種類	DOUBLE

npery
年間の複利計算期間数を指定します。

データの種類	DOUBLE

例: 計算の説明: NOMINAL

次のプログラムでは、年間の名目金利を計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double effectrate npery r;
  method init();
    effectrate=0.08;
    npery=4;
    r= finance('nominal', effectrate, npery);
    put r=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
r=0.07770618763309
```

FINANCE NPER 関数

投資の期間数を計算します。

カテゴリ:	財務
返されるデータの種類:	DOUBLE

構文

FINANCE('NPER', *rate*, *payment*, *pv*, [*fv*], [*type*]);

必須引数

rate
利率を指定します。

要件	*rate* は、パーセンテージではなく数値として提供されます。
データの種類	DOUBLE

payment
各期間に行われる支払いを指定します。年金の有効期間中に支払いが変更されることはありません。通常、*payment* には元本と利息が含まれますが、その他の手数料や税金は含まれません。*payment* を省略する場合、*pv* 引数を含める必要があります。

データの種類 DOUBLE

pv
現在価値、または一連の将来の支払いに相当する現時点での一括払い金額を指定します。*pv* を省略すると、0(ゼロ)とみなされ、*payment* 引数を含める必要があります。

データの種類 DOUBLE

オプション引数

fv
将来価値、または最終支払い後に達する現金残高を指定します。*fv* を省略すると、0(ゼロ)とみなされます(たとえば、ローンの将来価値が 0)。

データの種類 DOUBLE

type
0 か 1 を指定し、支払い期日がいつであるかを示します。*type* を省略すると、0 とみなされます。

支払い期日が期末の場合は、*type* 引数を省略するか、0 に設定します。支払い期日が期首の場合は、*type* を 1 に設定します。

データの種類 DOUBLE

例: 計算の説明: NPER

次のプログラムでは、投資の期間数を計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double rate payment pv fv type r;
  method init();
    rate=0.08;
    payment=200;
    pv=1000;
    fv=0;
    type=0;
    r=finance('nper', rate, payment, pv, fv, type);
    put r=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
r=-4.37198135126657
```

FINANCE NPV 関数

一連の定期的なキャッシュフローと割引率に基づいて投資の正味現在価値を計算します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

FINANCE('NPV', *rate*, *value-1*, [, ... *value-n*]);

必須引数

rate
利率を指定します。

要件 *rate* は、パーセンテージではなく数値として提供されます。

データの種類 DOUBLE

value
一連のキャッシュフローを表します。

データの種類 DOUBLE

例: 計算の説明: NPV

次のプログラムでは、一連の定期的なキャッシュフローと割引率に基づいて投資の正味現在価値を計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double rate v1 v2 v3 r;
  method init();
    rate=0.08;
    v1=200;
    v2=1000;
    v3=0.;
```

```
    r=finance('npv', rate, v1, v2, v3);
    put r=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
r=1042.52400548696
```

FINANCE ODDFPRICE 関数

端数の開始期間で額面$100 当たりの証券の価格を計算します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

FINANCE('ODDFPRICE', *settlement*, *maturity*, *issue*, *first-coupon*, *rate*, *yield*, *redemption*, *frequency*, [*basis*]);

必須引数

settlement
決済日を指定します。

要件 *settlement* は SAS 日付です。

データの種類 DOUBLE

maturity
満期日を指定します。

要件 *maturity* は SAS 日付です。

データの種類 DOUBLE

issue
証券の発行日を指定します。

要件 *issue* は SAS 日付です。

データの種類 DOUBLE

first-coupon
証券の開始クーポン日を指定します。

要件 *first-coupon* は SAS 日付です。

データの種類 DOUBLE

rate
利率を指定します。

要件 *rate* は、パーセンテージではなく数値として提供されます。

データの種類 DOUBLE

yield
証券の年間利回りを指定します。

データの種類 DOUBLE

redemption
満期受け取り額を指定します。

データの種類 DOUBLE

frequency
年間のクーポン支払い回数を指定します。年 1 回支払いの場合は *frequency*=1、年 2 回支払いの場合は *frequency*=2、年 4 回支払いの場合は *frequency*=4 となります。

データの種類 DOUBLE

オプション引数

basis
使用する日数計算基準の種類を示す文字または数値としてオプションのパラメーターを指定します。

数値	文字列値	日数計算方法
0	"30/360"	US (NASD) 30/360
1	"ACTUAL"	実日数/実日数
2	"ACT/360"	実日数/360
3	"ACT/365"	実日数/365
4	"EU30/360"	ヨーロッパ 30/360

データの種類 DOUBLE

例: 計算の説明: ODDFPRICE

次のプログラムでは、端数の開始期間で額面$100 当たりの証券の価格を計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double settlement maturity issue firstcoupon
  rate yield redemption frequency basis r;
  method init();
    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=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
r=103.941039839766
```

FINANCE ODDFYIELD 関数

端数の開始期間で証券の利回りを計算します。

カテゴリ:	財務
返されるデータの種類:	DOUBLE

構文

FINANCE('ODDFYIELD', *settlement*, *maturity*, *issue*, *first-coupon*, *rate*, *price*, *redemption*, *frequency*, [*basis*]);

必須引数

settlement
決済日を指定します。

要件 *settlement* は SAS 日付です。

データの種類 DOUBLE

maturity
満期日を指定します。

要件 *maturity* は SAS 日付です。

データの種類 DOUBLE

issue
証券の発行日を指定します。

要件 *issue* は SAS 日付です。

データの種類 DOUBLE

first-coupon
証券の開始クーポン日を指定します。

要件 *first-coupon* は SAS 日付です。

データの種類 DOUBLE

rate
利率を指定します。

要件 *rate* は、パーセンテージではなく数値として提供されます。

データの種類 DOUBLE

price
額面$100 当たりの証券の価格を指定します。

データの種類 DOUBLE

redemption
満期受け取り額を指定します。

データの種類 DOUBLE

frequency
年間のクーポン支払い回数を指定します。年 1 回支払いの場合は *frequency*=1、年 2 回支払いの場合は *frequency*=2、年 4 回支払いの場合は *frequency*=4 となります。

データの種類 DOUBLE

オプション引数

basis
使用する日数計算基準の種類を示す文字または数値としてオプションのパラメーターを指定します。

数値	文字列値	日数計算方法
0	"30/360"	US (NASD) 30/360
1	"ACTUAL"	実日数/実日数
2	"ACT/360"	実日数/360
3	"ACT/365"	実日数/365
4	"EU30/360"	ヨーロッパ 30/360

データの種類 DOUBLE

例: 計算の説明: ODDFYIELD

次のプログラムでは、端数の開始期間で利回りの利息を計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double settlement maturity issue firstcoupon
    rate price redemption frequency basis r;
    method init();
    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=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
r=0.05999999999946
```

FINANCE ODDLPRICE 関数

端数の最終期間で額面$100 当たりの証券の価格を計算します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

FINANCE('ODDLPRICE', *settlement*, *maturity*, *last-interest*, *rate*, *yield*, *redemption*, *frequency*, [*basis*]);

必須引数

settlement
決済日を指定します。

要件 *settlement* は SAS 日付です。

データの種類 DOUBLE

maturity
満期日を指定します。

要件 *maturity* は SAS 日付です。

データの種類 DOUBLE

last-interest
証券の最終クーポン日を指定します。

要件 *last-interest* は SAS 日付です。

データの種類 DOUBLE

rate
利率を指定します。

要件 *rate* は、パーセンテージではなく数値として提供されます。

データの種類 DOUBLE

yield
証券の年間利回りを指定します。

データの種類 DOUBLE

redemption
満期受け取り額を指定します。

データの種類 DOUBLE

frequency
年間のクーポン支払い回数を指定します。年 1 回支払いの場合は *frequency*=1、年 2 回支払いの場合は *frequency*=2、年 4 回支払いの場合は *frequency*=4 となります。

データの種類 DOUBLE

オプション引数

basis
使用する日数計算基準の種類を示す文字または数値としてオプションのパラメーターを指定します。

数値	文字列値	日数計算方法
0	"30/360"	US (NASD) 30/360
1	"ACTUAL"	実日数/実日数
2	"ACT/360"	実日数/360
3	"ACT/365"	実日数/365
4	"EU30/360"	ヨーロッパ 30/360

データの種類 DOUBLE

例: 計算の説明: ODDLPRICE

次のプログラムでは、端数の最終期間で額面$100 当たりの証券の価格を計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data;
```

```
  dcl double settlement maturity lastinterest
    rate yield redemption frequency basis r;
  method init();
    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=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
r=99.8782860147213
```

FINANCE ODDLYIELD 関数

端数の最終期間で証券の利回りを計算します。

カテゴリ:	財務
返されるデータの種類:	DOUBLE

構文

FINANCE('ODDLYIELD', *settlement*, *maturity*, *last-interest*, *rate*, *price*, *redemption*, *frequency*, [*basis*]);

必須引数

settlement
決済日を指定します。

要件	*settlement* は SAS 日付です。
データの種類	DOUBLE

maturity
満期日を指定します。

要件 *maturity* は SAS 日付です。

データの種類 DOUBLE

last-interest
証券の最終クーポン日を指定します。

要件 *last-interest* は SAS 日付です。

データの種類 DOUBLE

rate
利率を指定します。

要件 *rate* は、パーセンテージではなく数値として提供されます。

データの種類 DOUBLE

price
額面$100 当たりの証券の価格を指定します。

データの種類 DOUBLE

redemption
満期受け取り額を指定します。

データの種類 DOUBLE

frequency
年間のクーポン支払い回数を指定します。年 1 回支払いの場合は *frequency*=1、年 2 回支払いの場合は *frequency*=2、年 4 回支払いの場合は *frequency*=4 となります。

データの種類 DOUBLE

オプション引数

basis
使用する日数計算基準の種類を示す文字または数値としてオプションのパラメーターを指定します。

数値	文字列値	日数計算方法
0	"30/360"	US (NASD) 30/360
1	"ACTUAL"	実日数/実日数
2	"ACT/360"	実日数/360
3	"ACT/365"	実日数/365
4	"EU30/360"	ヨーロッパ 30/360

データの種類 DOUBLE

例: 計算の説明: ODDLYIELD

次のプログラムでは、端数の最終期間で証券の利回りを計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double settlement maturity lastinterest
    rate price redemption frequency basis r;
  method init();
    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('oddlyield', settlement, maturity, lastinterest,
      rate, price, redemption, frequency, basis);
    put r=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
r=0.04049999999213
```

FINANCE PMT 関数

年金の定期的支払いを計算します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

FINANCE('PMT', *rate*, *nper*, *pv*, [*fv*], [*type*]);

必須引数

rate
利率を指定します。

要件 *rate* は、パーセンテージではなく数値として提供されます。

データの種類 DOUBLE

nper
支払い期間の総数を指定します。

データの種類 DOUBLE

pv
現在価値、または一連の将来の支払いに相当する現時点での一括払い金額を指定します。*pv* を省略すると、0(ゼロ)とみなされ、*fv* 引数を含める必要があります。

データの種類 DOUBLE

オプション引数

fv
将来価値、または最終支払い後に達する現金残高を指定します。*fv* を省略すると、0(ゼロ)とみなされます(たとえば、ローンの将来価値が 0)。

データの種類 DOUBLE

type
0 か 1 を指定し、支払い期日がいつであるかを示します。*type* を省略すると、0 とみなされます。

支払い期日が期末の場合は、*type* 引数を省略するか、0 に設定します。支払い期日が期首の場合は、*type* を 1 に設定します。

データの種類 DOUBLE

例: 計算の説明: PMT

次のプログラムでは、年金の定期的支払いを計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
```

```
    dcl double rate nper pv fv type r;
    method init();
      rate=0.08;
      nper=5;
      pv=91;
      fv=3;
      type=0;
      r=finance('pmt', rate, nper, pv, fv, type);
      put r=;
    end;
  enddata;
  run;
  quit;
```

SAS は次の出力をログに書き込みます。

```
r=-23.3029067292826
```

FINANCE PPMT 関数

指定期間の投資の元本に対する支払いを計算します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

FINANCE('PPMT', *rate*, *period*, *nper*, *pv*, [*fv*], [*type*]);

必須引数

rate
利率を指定します。

要件 *rate* は、パーセンテージではなく数値として提供されます。

データの種類 DOUBLE

period
期間を指定します。

範囲: 1–*nper*

データの種類 DOUBLE

nper
支払期間の数を指定します。

データの種類 DOUBLE

pv
現在価値、または一連の将来の支払いに相当する現時点での一括払い金額を指定します。*pv* を省略すると、0(ゼロ)とみなされ、*fv* 引数を含める必要があります。

データの種類 DOUBLE

オプション引数

fv
将来価値、または最終支払い後に達する現金残高を指定します。*fv* を省略すると、0(ゼロ)とみなされます(たとえば、ローンの将来価値が 0)。

データの種類 DOUBLE

type
0 か 1 を指定し、支払い期日がいつであるかを示します。*type* を省略すると、0 とみなされます。

支払い期日が期末の場合は、*type* 引数を省略するか、0 に設定します。支払い期日が期首の場合は、*type* を 1 に設定します。

データの種類 DOUBLE

例: 計算の説明: PPMT

次のプログラムでは、指定期間の投資の元本に対する支払いを計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double rate period nper pv fv type r;
  method init();
    rate=0.08;
    period=10;
    nper=10;
    pv=200000;
    fv=0;
    type=0;
    r=finance('ppmt', rate, period, nper, pv, fv, type);
    put r=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
r=-27598.0534624213
```

FINANCE PRICE 関数

定期的に利息を支払う額面$100 当たりの証券の価格を計算します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

FINANCE('PRICE', *settlement*, *maturity*, *rate*, *yield*, *redemption*, *frequency*, [*basis*]);

必須引数

settlement
決済日を指定します。

要件 *settlement* は SAS 日付です。

データの種類 DOUBLE

maturity
満期日を指定します。

要件 *maturity* は SAS 日付です。

データの種類 DOUBLE

rate
利率を指定します。

要件 *rate* は、パーセンテージではなく数値として提供されます。

データの種類 DOUBLE

yield
証券の年間利回りを指定します。

データの種類 DOUBLE

redemption
満期受け取り額を指定します。

データの種類 DOUBLE

frequency
年間のクーポン支払い回数を指定します。年 1 回支払いの場合は *frequency*=1、年 2 回支払いの場合は *frequency*=2、年 4 回支払いの場合は *frequency*=4 となります。

データの種類 DOUBLE

オプション引数

basis
使用する日数計算基準の種類を示す文字または数値としてオプションのパラメーターを指定します。

数値	文字列値	日数計算方法
0	"30/360"	US (NASD) 30/360
1	"ACTUAL"	実日数/実日数
2	"ACT/360"	実日数/360
3	"ACT/365"	実日数/365
4	"EU30/360"	ヨーロッパ 30/360

データの種類 DOUBLE

例: 計算の説明: PRICE

次のプログラムでは、定期的に利息を支払う額面$100 当たりの証券の価格を計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double settlement maturity rate yield redemption
    frequency basis r;
  method init();
    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=;
    end;
  enddata;
  run;
  quit;
```

SAS は次の出力をログに書き込みます。

```
r=94.6343616213221
```

FINANCE PRICEDISC 関数

額面$100 当たりの割引証券の価格を計算します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

FINANCE('PRICEDISC', *settlement*, *maturity*, *discount*, *redemption*, [*basis*]);

必須引数

settlement
決済日を指定します。

要件 *settlement* は SAS 日付です。

データの種類 DOUBLE

maturity
満期日を指定します。

要件 *maturity* は SAS 日付です。

データの種類 DOUBLE

discount
証券の割引率を指定します。

要件 *discount* は、パーセンテージではなく数値として提供されます。

データの種類 DOUBLE

redemption
満期受け取り額を指定します。

データの種類 DOUBLE

オプション引数

basis
使用する日数計算基準の種類を示す文字または数値としてオプションのパラメーターを指定します。

数値	文字列値	日数計算方法
0	"30/360"	US (NASD) 30/360
1	"ACTUAL"	実日数/実日数
2	"ACT/360"	実日数/360
3	"ACT/365"	実日数/365
4	"EU30/360"	ヨーロッパ 30/360

データの種類 DOUBLE

例: 計算の説明: PRICEDISC

次のプログラムでは、額面$100 当たりの割引証券の価格を計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double settlement maturity discount redemption basis r;
  method init();
    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=;
  end;
enddata;
run;
```

```
quit;
```

SAS は次の出力をログに書き込みます。

```
r=48.8125
```

FINANCE PRICEMAT 関数

満期に利息を支払う額面$100 当たりの証券の価格を計算します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

FINANCE('PRICEMAT', *settlement*, *maturity*, *issue*, *rate*, *yield*, [*basis*]);

必須引数

settlement
決済日を指定します。

要件 *settlement* は SAS 日付です。

データの種類 DOUBLE

maturity
満期日を指定します。

要件 *maturity* は SAS 日付です。

データの種類 DOUBLE

issue
証券の発行日を指定します。

要件 *issue* は SAS 日付です。

データの種類 DOUBLE

rate
利率を指定します。

要件 *rate* は、パーセンテージではなく数値として提供されます。

データの種類 DOUBLE

yield
証券の年間利回りを指定します。

データの種類 DOUBLE

オプション引数

basis
使用する日数計算基準の種類を示す文字または数値としてオプションのパラメーターを指定します。

数値	文字列値	日数計算方法
0	"30/360"	US (NASD) 30/360
1	"ACTUAL"	実日数/実日数
2	"ACT/360"	実日数/360
3	"ACT/365"	実日数/365
4	"EU30/360"	ヨーロッパ 30/360

データの種類 DOUBLE

例: 計算の説明: PRICEMAT

次のプログラムでは、満期に利息を支払う額面$100 当たりの証券の価格を計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double settlement maturity issue rate yield basis r;
  method init();
    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=;
  end;
enddata;
```

```
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
r=99.9844988755569
```

FINANCE PV 関数

投資の現在価値を計算します。

カテゴリ:	財務
返されるデータの種類:	DOUBLE

構文

FINANCE('PV', *rate*, *nper*, *payment*, [*fv*], [*type*]);

必須引数

rate
利率を指定します。

要件	*rate* は、パーセンテージではなく数値として提供されます。
データの種類	DOUBLE

nper
支払い期間の総数を指定します。

データの種類	DOUBLE

payment
各期間に行われる支払いを指定します。年金の有効期間中に支払いが変更されることはありません。通常、*payment* には元本と利息が含まれますが、その他の手数料や税金は含まれません。

データの種類	DOUBLE

オプション引数

fv
将来価値、または最終支払い後に達する現金残高を指定します。*fv* を省略すると、0(ゼロ)とみなされます(たとえば、ローンの将来価値が 0)。

データの種類	DOUBLE

type
0 か 1 を指定し、支払い期日がいつであるかを示します。*type* を省略すると、0 とみなされます。

支払い期日が期末の場合は、*type* 引数を省略するか、0 に設定します。支払い期日が期首の場合は、*type* を 1 に設定します。

データの種類 DOUBLE

例: 計算の説明: PV

次のプログラムでは、投資の現在価値を計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double rate nper payment fv type r;
  method init();
    rate=0.05;
    nper=10;
    payment=1000;
    fv=200;
    type=0;
    r=finance('pv', rate, nper, payment, fv, type);
    put r=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
r=-7844.51757989297
```

FINANCE RATE 関数

年金期間当たりの利率を計算します。

カテゴリ:	財務
返されるデータの種類:	DOUBLE

構文

FINANCE('RATE', *nper*, *payment*, *pv*, [*fv*], [*type*]);

必須引数

nper
支払い期間の総数を指定します。

データの種類 DOUBLE

payment
各期間に行われる支払いを指定します。年金の有効期間中に支払いが変更されることはありません。通常、*payment* には元本と利息が含まれますが、その他の手数料や税金は含まれません。*payment* を省略する場合、*pv* 引数を含める必要があります。

データの種類 DOUBLE

pv
現在価値、または一連の将来の支払いに相当する現時点での一括払い金額を指定します。*pv* を省略すると、0(ゼロ)とみなされ、*fv* 引数を含める必要があります。

データの種類 DOUBLE

オプション引数

fv
将来価値、または最終支払い後に達する現金残高を指定します。*fv* を省略すると、0(ゼロ)とみなされます(たとえば、ローンの将来価値が 0)。

データの種類 DOUBLE

type
0 か 1 を指定し、支払い期日がいつであるかを示します。*type* を省略すると、0 とみなされます。

支払い期日が期末の場合は、*type* 引数を省略するか、0 に設定します。支払い期日が期首の場合は、*type* を 1 に設定します。

データの種類 DOUBLE

例: 計算の説明: RATE

次のプログラムでは、年金期間当たりの利率を計算します。

```
proc ds2;
data test(overwrite=yes);
  dcl double nper payment pv r;
  method init();
    nper=4;
```

```
      payment=-2481;
      pv=8000;
      r=finance('rate', nper, payment, pv);
      put r=;
    end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
r=0.09214768406875
```

FINANCE RECEIVED 関数

完全投資済み証券の満期受け取り額を計算します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

FINANCE('RECEIVED', *settlement*, *maturity*, *investment*, *discount*, [*basis*]);

必須引数

settlement
決済日を指定します。

要件 *settlement* は SAS 日付です。

データの種類 DOUBLE

maturity
満期日を指定します。

要件 *maturity* は SAS 日付です。

データの種類 DOUBLE

investment
証券に投資される金額を指定します。

データの種類 DOUBLE

discount
証券の割引率を指定します。

要件　*discount* は、パーセンテージではなく数値として提供されます。

データの種類　DOUBLE

オプション引数

basis
使用する日数計算基準の種類を示す文字または数値としてオプションのパラメーターを指定します。

数値	文字列値	日数計算方法
0	"30/360"	US (NASD) 30/360
1	"ACTUAL"	実日数/実日数
2	"ACT/360"	実日数/360
3	"ACT/365"	実日数/365
4	"EU30/360"	ヨーロッパ 30/360

データの種類　DOUBLE

例: 計算の説明: RECEIVED

次のプログラムでは、完全投資済み証券の満期受け取り額を計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double settlement maturity investment discount basis r;
  method init();
    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=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
r=1014584.6544071
```

FINANCE SLN 関数

ある期間の資産の定額法による減価償却を計算します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

FINANCE('SLN', *cost*, *salvage*, *life*);

必須引数

cost
資産の初期コストを指定します。

データの種類 DOUBLE

salvage
減価償却を終了した時点の価値(資産の救済価額とも呼ばれる)を指定します。

データの種類 DOUBLE

life
資産が減価償却される期間数(資産の耐用年数とも呼ばれる)を指定します。

データの種類 DOUBLE

例: 計算の説明: SLN

次のプログラムでは、ある期間の資産の定額法による減価償却を計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double cost salvage life r;
  method init();
```

```
      cost=2000;
      salvage=200;
      life=11;
      r=finance('sln', cost, salvage, life);
      put r=;
    end;
  enddata;
  run;
  quit;
```

SAS は次の出力をログに書き込みます。

```
r=163.636363636363
```

FINANCE SYD 関数

指定期間の資産の年次級数和法による減価償却を計算します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

FINANCE('SYD', *cost*, *salvage*, *life*, *period*);

必須引数

cost
資産の初期コストを指定します。

データの種類 DOUBLE

salvage
減価償却を終了した時点の価値(資産の救済価額とも呼ばれる)を指定します。

データの種類 DOUBLE

life
資産が減価償却される期間数(資産の耐用年数とも呼ばれる)を指定します。

データの種類 DOUBLE

period
引数 *life* に使用するものと同じ時間単位で期間を指定します。

データの種類 DOUBLE

例: 計算の説明: SYD

次のプログラムでは、指定期間の資産の年次級数和法による減価償却を計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double cost salvage life period r;
  method init();
    cost=2000;
    salvage=200;
    life=11;
    period=1;
    r=finance('syd', cost, salvage, life, period);
    put r=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
r=300
```

FINANCE TBILLEQ 関数

米国財務省短期証券の債券換算利回りを計算します。

カテゴリ:	財務
返されるデータの種類:	DOUBLE

構文

FINANCE('TBILLEQ', *settlement*, *maturity*, *discount*);

必須引数

settlement
決済日を指定します。

要件	*settlement* は SAS 日付です。
データの種類	DOUBLE

maturity
満期日を指定します。

要件 *maturity* は SAS 日付です。

データの種類 DOUBLE

discount
証券の割引率を指定します。

要件 *discount* は、パーセンテージではなく数値として提供されます。

データの種類 DOUBLE

例: 計算の説明: TBILLEQ

次のプログラムでは、米国財務省短期証券の債券換算利回りを計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double settlement maturity discount r;
  method init();
    settlement=mdy(3, 31, 2008);
    maturity=mdy(6, 1, 2008);
    discount=0.0914;
    r=finance('tbilleq', settlement, maturity, discount);
    put r=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
r=0.09415149356594
```

FINANCE TBILLPRICE 関数

額面$100 当たりの米国財務省短期証券の価格を計算します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

FINANCE('TBILLPRICE', *settlement*, *maturity*, *discount*);

必須引数

settlement
決済日を指定します。

要件 *settlement* は SAS 日付です。

データの種類 DOUBLE

maturity
満期日を指定します。

要件 *maturity* は SAS 日付です。

データの種類 DOUBLE

discount
証券の割引率を指定します。

要件 *discount* は、パーセンテージではなく数値として提供されます。

データの種類 DOUBLE

例: 計算の説明: TBILLPRICE

次のプログラムでは、額面$100 当たりの米国財務省短期証券の価格を計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double settlement maturity discount r;
  method init();
    settlement=mdy(3, 31, 2008);
    maturity=mdy(6, 1, 2008);
    discount=0.09;
    r=finance('tbillprice', settlement, maturity, discount);
    put r=;
  end;
enddata;
run;
quit;
```

返される *r* の値は 98.45 です。

FINANCE TBILLYIELD 関数

米国財務省短期証券の利回りを計算します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

FINANCE('TBILLYIELD', *settlement*, *maturity*, *price*);

必須引数

settlement
決済日を指定します。

要件 *settlement* は SAS 日付です。

データの種類 DOUBLE

maturity
満期日を指定します。

要件 *maturity* は SAS 日付です。

データの種類 DOUBLE

price
額面$100 当たりの証券の価格を指定します。

データの種類 DOUBLE

例: 計算の説明: TBILLYIELD

次のプログラムでは、米国財務省短期証券の利回りを計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double settlement maturity price r;
  method init();
```

```
      settlement=mdy(3, 31, 2008);
      maturity=mdy(6, 1, 2008);
      price=98;
      r=finance('tbillyield', settlement, maturity, price);
      put r=;
    end;
  enddata;
  run;
  quit;
```

SAS は次の出力をログに書き込みます。

```
r=0.11849901250822
```

FINANCE VDB 関数

定率法を使用して、指定期間または一部期間の資産の減価償却を計算します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

FINANCE('VDB', *cost*, *salvage*, *life*, *start-period*, *end-period*, [*factor*], [*noswitch*]);

必須引数

cost
資産の初期コストを指定します。

データの種類 DOUBLE

salvage
減価償却を終了した時点の価値(資産の救済価額とも呼ばれる)を指定します。

データの種類 DOUBLE

life
資産が減価償却される期間数(資産の耐用年数とも呼ばれる)を指定します。

データの種類 DOUBLE

start-period
計算の開始期間を指定します。支払い期間は 1 から番号が付けられます。

データの種類 DOUBLE

end-period
計算の最終期間を指定します。

データの種類 DOUBLE

オプション引数

factor
減価償却率を指定します。*factor* を省略すると、2 とみなされます(倍額定率法)。

データの種類 DOUBLE

noswitch
減価償却費が定率法による計算よりも大きい場合、定額法による減価償却に切り替えるかどうかを決定する論理値を指定します。*noswitch* を省略すると、1 とみなされます。

データの種類 DOUBLE

例: 計算の説明: VDB

次のプログラムでは、定率法を使用して、指定期間または一部期間の資産の減価償却を計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double cost salvage life startperiod endperiod factor r;
  method init();
    cost=2400;
    salvage=300;
    life=10;
    startperiod=0;
    endperiod=1;
    factor=1.5;
    r=finance('vdb', cost, salvage, life, startperiod, endperiod, factor);
    put r=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
r=360
```

FINANCE XIRR 関数

必ずしも定期的ではないキャッシュフロースケジュールの 内部利益率を計算します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

FINANCE('XIRR', *values*, *dates*, [*guess*]);

必須引数

values
支払いのスケジュール(日付)に対応する一連のキャッシュフローを指定します。1 回目の支払いは任意で、投資開始時に発生するコストまたは支払いに対応します。開始値がコストまたは支払いの場合、負の値にする必要があります。それ以降のすべての 支払いは、1 年 365 日に基づいて割引されます。一連の値には、正の値と負の値が少なくとも 1 つずつ含まれている必要があります。

データの種類 DOUBLE

dates
キャッシュフローの支払いに対応する支払い日のスケジュールを指定します。1 回目の支払い日は、支払いスケジュールの開始時を示します。その他のすべての日付は、この日付より後にする必要がありますが、どの順序で発生してもかまいません。

要件 *dates* は SAS 日付です。

データの種類 DOUBLE

オプション引数

guess
XIRR の結果に近い推測する数値を指定します(省略可能)。

データの種類 DOUBLE

例: 計算の説明: XIRR

次のプログラムでは、必ずしも定期的ではないキャッシュフロースケジュールの 内部利益率を計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double v1 v2 v3 v4 v5 d1 d2 d3 d4 d5 r;
  method init();
    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=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
r=0.37336253351883
```

FINANCE XNPV 関数

必ずしも定期的ではないキャッシュフロースケジュールの 正味現在価値を計算します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

FINANCE('XNPV', *rate*, *values*, *dates*);

必須引数

rate
利率を指定します。

要件 *rate* は、パーセンテージではなく数値として提供されます。

データの種類 DOUBLE

values
支払いのスケジュール(日付)に対応する一連のキャッシュフローを指定します。1 回目の支払いは任意で、投資開始時に発生するコストまたは支払いに対応しま

す。開始値がコストまたは支払いの場合、負の値にする必要があります。それ以降のすべての 支払いは、1 年 365 日に基づいて割引されます。一連の値には、正の値と負の値が少なくとも 1 つずつ含まれている必要があります。

データの種類 DOUBLE

dates
キャッシュフローの支払いに対応する支払い日のスケジュールを指定します。1 回目の支払い日は、支払いスケジュールの開始時を示します。その他のすべての日付は、この日付より後にする必要がありますが、どの順序で発生してもかまいません。

要件 *dates* は SAS 日付です。

データの種類 DOUBLE

例: 計算の説明: XNPV

次のプログラムでは、必ずしも定期的ではないキャッシュフロースケジュールの正味現在価値を計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double rate v1 v2 v3 v4 v5 d1 d2 d3 d4 d5 r;
  method init();
    rate=.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', rate, v1, v2, v3, v4, v5, d1, d2, d3, d4, d5);
    put r=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
r=2086.64760203153
```

FINANCE YIELD 関数

定期的に利息を支払う証券の利回りを計算します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

FINANCE('YIELD', *settlement*, *maturity*, *rate*, *price*, *redemption*, *frequency*, [*basis*]);

必須引数

settlement
決済日を指定します。

要件 *settlement* は SAS 日付です。

データの種類 DOUBLE

maturity
満期日を指定します。

要件 *maturity* は SAS 日付です。

データの種類 DOUBLE

rate
利率を指定します。

要件 *rate* は、パーセンテージではなく数値として提供されます。

データの種類 DOUBLE

price
額面$100 当たりの証券の価格を指定します。

データの種類 DOUBLE

redemption
満期受け取り額を指定します。

データの種類 DOUBLE

frequency
年間のクーポン支払い回数を指定します。年 1 回支払いの場合は *frequency*=1、年 2 回支払いの場合は *frequency*=2、年 4 回支払いの場合は *frequency*=4 となります。

データの種類 DOUBLE

オプション引数

basis
使用する日数計算基準の種類を示す文字または数値としてオプションのパラメーターを指定します。

数値	文字列値	日数計算方法
0	"30/360"	US (NASD) 30/360
1	"ACTUAL"	実日数/実日数
2	"ACT/360"	実日数/360
3	"ACT/365"	実日数/365
4	"EU30/360"	ヨーロッパ 30/360

データの種類 DOUBLE

例: 計算の説明: YIELD

次のプログラムでは、定期的に利息を支払う証券の利回りを計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double settlement maturity rate pr redemption frequency basis r;
  method init();
    settlement=mdy(2, 15, 2008);
    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=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
r=0.06500000688075
```

FINANCE YIELDDISC 関数

割引証券(米国財務省短期証券など)の年間利回りを計算します。

カテゴリ:	財務
返されるデータの種類:	DOUBLE

構文

FINANCE('YIELDDISC', *settlement*, *maturity*, *price*, *redemption*, [*basis*]);

必須引数

settlement
決済日を指定します。

要件	*settlement* は SAS 日付です。
データの種類	DOUBLE

maturity
満期日を指定します。

要件	*maturity* は SAS 日付です。
データの種類	DOUBLE

price
額面$100 当たりの証券の価格を指定します。

データの種類	DOUBLE

redemption
満期受け取り額を指定します。

データの種類	DOUBLE

オプション引数

basis
使用する日数計算基準の種類を示す文字または数値としてオプションのパラメーターを指定します。

数値	文字列値	日数計算方法
0	"30/360"	US (NASD) 30/360
1	"ACTUAL"	実日数/実日数
2	"ACT/360"	実日数/360
3	"ACT/365"	実日数/365
4	"EU30/360"	ヨーロッパ 30/360

データの種類　DOUBLE

例: 計算の説明: YIELDDISC

次のプログラムでは、割引証券(米国財務省短期証券など)の年間利回りを計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double settlement maturity pr redemption basis r;
  method init();
    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=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
r=0.0059607747836
```

FINANCE YIELDMAT 関数

満期に利息を支払う証券の年間利回りを計算します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

FINANCE('YIELDMAT', *settlement*, *maturity*, *issue*, *rate*, *price*, [*basis*]);

必須引数

settlement
決済日を指定します。

要件 *settlement* は SAS 日付です。

データの種類 DOUBLE

maturity
満期日を指定します。

要件 *maturity* は SAS 日付です。

データの種類 DOUBLE

issue
証券の発行日を指定します。

要件 *issue* は SAS 日付です。

データの種類 DOUBLE

rate
利率を指定します。

要件 *rate* は、パーセンテージではなく数値として提供されます。

データの種類 DOUBLE

price
額面$100 当たりの証券の価格を指定します。

データの種類 DOUBLE

オプション引数

basis
使用する日数計算基準の種類を示す文字または数値としてオプションのパラメーターを指定します。

数値	文字列値	日数計算方法
0	"30/360"	US (NASD) 30/360
1	"ACTUAL"	実日数/実日数
2	"ACT/360"	実日数/360
3	"ACT/365"	実日数/365
4	"EU30/360"	ヨーロッパ 30/360

データの種類 DOUBLE

例: 計算の説明: YIELDMAT

次のプログラムでは、満期に利息を支払う証券の年間利回りを計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double settlement maturity issue rate pr basis r;
  method init();
    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=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
r=0.06095433369153
```

FIND 関数

文字列内の特定の部分文字列を検索します。

カテゴリ: 文字

返されるデータの種類: DOUBLE

ヒント: かわりに“KINDEX 関数” (*SAS 各国語サポート(NLS): リファレンスガイド*) を使用してエンコーディングの非依存コードを書き込みます。

構文

FIND(*string*, *substring*[, *modifier(s)*][, *startpos*])

FIND(*string*, *substring*[, *startpos*][, *modifier(s)*])

必須引数

string
文字列を評価するか、文字列に強制変換できて、部分文字列の検索対象になる文字定数、変数、または式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

ヒント 文字のリテラル文字列を引用符で囲みます。

substring
文字列を評価するか、文字列に強制変換できる文字定数、変数、または式であり、*string* で検索する文字の部分文字列を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

ヒント 文字のリテラル文字列を引用符で囲みます。

オプション引数

modifier(s)
1 つ以上の修飾子を文字定数、変数または式で指定します。修飾子として使用できる文字(大文字または小文字)は、I (大文字と小文字の区別を無視する)または T (末尾の空白を取り除く)です。両方(またはすべて)の文字引数ではなく一方のみから末尾の空白を削除する場合は、FIND 関数に T 修飾子を使用するかわりに、TRIM 関数を使用します。

1 つ以上の修飾子を文字定数、変数または式で指定します。修飾子として使用できる文字(大文字または小文字)は次のとおりです。

i or I
検索時に大文字と小文字を区別しません。この修飾子を指定しないと、FIND は *substring* の文字の大文字と小文字に一致する部分文字列のみを検索します。

t or T
string および *substring* から末尾の空白を取り除きます。

注: 両方(またはすべて)の文字引数ではなく一方のみから末尾の空白を削除する場合は、FIND 関数に T 修飾子を使用するかわりに、TRIM 関数を使用します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

ヒント *modifier* が定数の場合、引用符で囲みます。一組の引用符で複数の定数を指定します。*modifier* を変数または式として表すこともできます。

startpos
検索を開始する位置と検索の方向を指定する数値定数、変数、または式です。式は整数に解決される必要があります

データの種類 DOUBLE

詳細

FIND 関数は *string* で最初に出現する指定の *substring* を検索し、その部分文字列の最初の文字の位置を返します。*string* 内に部分文字列が見つからない場合、FIND は値 0 を返します。

sstartpos を指定しないと、FIND は *string* の最初から検索を開始し、左から右に *string* を検索します。*startpos* を指定すると、*startpos* の絶対値で検索の開始位置が決定されます。検索方向は、*startpos* の符号で決まります。

startpos の値	アクション
0 より大きい	*startpos* の位置から検索を開始し、右方向に検索します。*startpos* が *string* の長さよりも大きい場合、FIND は値 0 を返します。
0 より小さい	*startpos* の位置から検索を開始し、左方向に検索します。*startpos* が *string* の長さよりも大きい場合、検索は *string* の最後から開始します。
0	値 0 を返します。

比較

- FIND 関数は文字列の文字の部分文字列を検索しますが、FINDC 関数は文字列の個々の文字を検索します。
- FIND 関数と INDEX 関数はどちらも文字列の文字の部分文字列を検索します。ただし、INDEX 関数には *modifiers* 引数も *startpos* 引数もありません。

例

次のプログラムは、さまざまなデータ型を使用した FIND 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl char(35) char1 char2 char3;
  dcl varchar(35) varchar1 varchar2 varchar3;
  dcl double i;
  method run();
    i=find('She sells seashells? Yes, she does.', 'she ', 'i');
    put 'literal' i=;
    char1='She sells seashells? Yes, she does.';
    char2='she ';
    char3='i';
    i=find(char1, char2, char3);
    put 'char' i=;
    varchar1='She sells seashells? Yes, she does.';
    varchar2='she ';
    varchar3='i';
    i=find(varchar1, varchar2, varchar3);
    put 'varchar' i=;
    char1='She sells seashells? Yes, she does.';
    char2='she ';
    char3='i';
    i=find(char1, trim(char2), char3);
    put 'trim char' i=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
literal i=1
char i=0
varchar i=1
trim char i=1
```

関連項目

関数:

- “COUNT 関数” (472 ページ)
- “FINDC 関数” (631 ページ)
- “FINDW 関数” (640 ページ)
- “INDEX 関数” (705 ページ)

FINDC 関数

文字のリストにある文字を文字列から検索します。

カテゴリ: 文字

返されるデータの種類: DOUBLE

ヒント: かわりに“KINDEXC 関数” (*SAS 各国語サポート(NLS): リファレンスガイド*)を使用してエンコーディングの非依存コードを書き込みます。

構文

FINDC(*string*, *charlist*[, *startpos*][, *modifiers*])

FINDC(*string*, *charlist*, *modifiers*[, *startpos*])

FINDC(*string*, *charlist*[, *modifiers*])

FINDC(*string*[, *charlist*])

必須引数

string
文字列を評価するか、文字列に強制変換できる文字定数、変数、または式であり、検索する文字列を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

ヒント 文字のリテラル文字列を引用符で囲みます。

charlist
文字リストを初期化する定数、変数または文字式です。*modifiers* 引数で K 修飾子を指定しないと、FINDC はこのリストの文字を検索します。K 修飾子を指定すると、FINDC はこの文字のリストにないすべての文字を検索します。他の修飾子を使うことでリストに文字をさらに追加できます。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

ヒント　　文字のリテラル文字列を引用符で囲みます。

オプション引数

modifiers
文字定数、変数または式です。これらの各文字で FINDC 関数のアクションを変更します。修飾子として使用できる文字(大文字または小文字)は次のとおりです。

A
文字のリストにアルファベット文字を追加します。

B
startpos 引数の符号に関係なく、左から右へではなく右から左へ検索します。

C
文字のリストに制御文字を追加します。

D
文字のリストに数字を追加します。

F
アンダースコア文字および英文字(VALIDVARNAME=V7 で SAS 変数名の先頭に使用できる文字)を文字リストに追加します。

G
文字のリストにグラフィカル文字を追加します。

H
文字のリストに水平タブを追加します。

I
検索時に大文字と小文字を区別しません。

K
文字のリストにない文字を検索します。この修飾子を指定しないと、FINDC は文字のリストにある文字を検索します。V 修飾子と K 修飾子は同じ機能を実行します。

L
小文字を文字リストに追加します。

N
文字のリストに数字、アンダースコアおよび英文字(VALIDVARNAME=V7 を使用した SAS 変数名内に表示可能な文字)を追加します。

O
charlist 引数および *modifiers* 引数を 1 回だけ処理します。FINDC 関数の呼び出し時毎には処理されません。DS2 (WHERE 句を除く)で O 修飾子を使用すると、*charlist* 引数と *modifiers* 引数が変化しないループで FINDC を呼び出すと、FINDC がより速く実行されます。

P
文字のリストに句読点を追加します。

S
文字のリストに空白文字(空白、水平タブ、垂直タブ、キャリッジリターン、ラインフィード、フォームフィード)を追加します。

T
string 引数と *charlist* 引数から末尾の空白をトリムします。両方(またはすべて)の文字引数ではなく一方のみから末尾の空白を削除する場合は、FINDC 関数に T 修飾子を使用するかわりに、TRIM 関数を使用することに注意してください。

U
大文字を文字リストに追加します。

V
文字のリストにないすべての文字が区切り文字として扱われるようにします。V が指定されていない場合、文字のリストにあるすべての文字が区切り文字として扱われます。V 修飾子と K 修飾子は同じ機能を実行します。

W
印刷可能文字を文字リストに追加します。

X
文字のリストに 16 進文字を追加します。

データの種類　CHAR、NCHAR、NVARCHAR、VARCHAR

ヒント　*modifiers* が定数の場合、引用符で囲みます。一組の引用符で複数の定数を指定します。*modifiers* を変数または式として表すこともできます。空白は無視されます。

startpos
検索を開始する位置と検索する方向を指定するオプションの数値定数、変数、または式です。式は整数に解決される必要があります。

データの種類　DOUBLE

詳細

FINDC 関数は *string* で最初に出現する指定の文字を検索し、最初に検出された文字の位置を返します。*string* 内に文字が見つからない場合、FINDC は値 0 を返します。

FINDC 関数では、文字引数を null に指定できます。null 引数は、長さがゼロの文字列として扱われます。数値引数は null にできません。

startpos を指定しないと、FINDC は、B 修飾子を使用している場合は文字列の最後から、B 修飾子を使用していない場合は文字列の最初から検索を開始します。

startpos を指定すると、*startpos* の絶対値で検索の開始位置が指定されます。B 修飾子を使用する場合、検索は常に右から左に進みます。B 修飾子を使用しない場合、*startpos* の符号で検索方向を指定します。次の表に、検索方向の要約を示します。

startpos の値	アクション
0 より大きい	検索は *startpos* の位置から開始され、右に進みます。*startpos* が文字列の長さよりも大きい場合、FINDC は値 0 を返します。

startpos の値	アクション
0 より小さい	検索は–*startpos* の位置から開始され、左に進みます。*startpos* が文字列の長さの負の値よりも小さい場合、検索は文字列の末尾から開始されます。
0	値 0 を返します。

比較

- FINDC 関数は文字列の個々の文字を検索しますが、FIND 関数は文字列の文字の部分文字列を検索します。
- FINDC 関数と INDEXC 関数はどちらも文字列の個々の文字を検索します。ただし、INDEXC 関数には *modifiers* 引数も *startpos* 引数もありません。
- FINDC 関数は文字列の個々の文字を検索しますが、VERIFY 関数は式に固有の最初の文字を検索します。VERIFY 関数には *modifiers* 引数も *startpos* 引数もありません。

例

例 1: 文字列の文字の検索

次のプログラムでは、文字列を検索して、検出された文字を返します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double j i;
  dcl char c;
  method run();
    j=0;
    do until(j=0);
      j = findc('Hi, ho!','hi',j+1);
      if j= 0 then put 'The End';
      else do;
        c = substr('Hi, ho!', j, 1);
        put j= c=;
      end;
    end;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
j=2 c=i
j=5 c=h
The End
```

例 2: 文字列の文字の検索(大文字と小文字を区別しない)

次のプログラムでは、文字列を検索して、検出された文字を返します。I 修飾子を使用して、文字の大文字と小文字を区別しないようにします。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl char string charlist c;
  dcl double j i;
  method run();
    string='Hi, ho!';
    charlist='ho';
    j=0;
    do until(j=0);
      j=findc(string,charlist,j+1,'i');
      if j=0 then put 'The End';
      else do;
        c=substr(string, j, 1);
        put j= c=;
      end;
    end;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
j=1 c=H
j=4 c=
j=5 c=h
j=6 c=o
j=8 c=
The End
```

例 3: K 修飾子を使用した文字の検索

次のプログラムでは、文字列を検索して、文字リストにない文字を返します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double j;
```

```
  dcl char c string charlist;
  method run();
    string='Hi, ho!';
    charlist='hi';
    j=0;
    do until(j = 0);
    j = findc(string,charlist,'k',j+1);
    if j=0 then put 'The End';
      else do;
        c = substr(string,j,1);
        put j= c=;
      end;
    end;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
j=1 c=H
j=3 c=,
j=6 c=o
j=7 c=!
The End
```

例 4: h、i および空白文字の検索

次のプログラムでは、3 つの文字(h、i および空白文字)を検索します。文字 h および i は小文字です。この検索では、大文字の H および I は無視されます。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double whereishi;
  dcl char whatfound;
  method run();
    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;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
whereishi=2 whatfound=i
whereishi=3 whatfound=
whereishi=5 whatfound=h
whereishi=10 whatfound=
The End
```

例 5: 文字 h および i の検索(大文字と小文字を区別しない)

次のプログラムでは、4 つの文字(h、i、H、I)を検索します。i 修飾子を指定すると、FINDC は検索時に大文字と小文字を区別しません。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double whereishi_i;
  dcl char whatfound variable1 variable2 variable3;
  method run();
    whereishi_i=0;
    do until(whereishi_i=0);
      whereishi_i=findc('Hi there, Ian!','hi ',whereishi_i+1);
      if whereishi_i=0 then put 'The End';
        else do;
          whatfound=substr('Hi there, Ian!',whereishi_i,1);
          put whereishi_i= whatfound=;
        end;
    end;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
whereishi_i=2 whatfound=i
whereishi_i=3 whatfound=
whereishi_i=5 whatfound=h
whereishi_i=10 whatfound=
The End
```

例 6: 文字 h および i の検索(末尾の空白を削除)

次のプログラムでは、2 つの文字(h および i)を検索します。t 修飾子を指定すると、FINDC は文字列引数および文字引数から末尾の空白を削除します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
dcl char(14) expression1 expression2 expression3 whatfound;
dcl double whereishi_t;
method run();
```

```
  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;
  end;
enddata;
run;
quit;
```

SAS は次の行の出力をログに書き込みます。

```
whereishi_t=2 whatfound=i
whereishi_t=5 whatfound=h
The End
```

例 7: h、i、H、I を除くすべての文字の検索

次のプログラムでは、文字列の h、i、H、I を除くすべての文字を検索します。v 修飾子を指定すると、FINDC は文字引数にない文字のみを数えます。この例には i 修飾子もあるため、検索時に大文字と小文字は区別されません。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl char(14) xyz;
  dcl double whereishi_iv;
  method run();
    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;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
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
```

関連項目

関数:

- “ANYALNUM 関数” (276 ページ)
- “ANYALPHA 関数” (279 ページ)
- “ANYCNTRL 関数” (282 ページ)
- “ANYDIGIT 関数” (284 ページ)
- “ANYGRAPH 関数” (289 ページ)
- “ANYLOWER 関数” (292 ページ)
- “ANYPRINT 関数” (296 ページ)
- “ANYPUNCT 関数” (299 ページ)
- “ANYSPACE 関数” (302 ページ)
- “ANYUPPER 関数” (304 ページ)
- “ANYXDIGIT 関数” (307 ページ)
- “COUNTC 関数” (474 ページ)
- “INDEXC 関数” (707 ページ)
- “NOTALNUM 関数” (876 ページ)
- “NOTALPHA 関数” (878 ページ)
- “NOTCNTRL 関数” (881 ページ)
- “NOTDIGIT 関数” (883 ページ)
- “NOTGRAPH 関数” (888 ページ)
- “NOTLOWER 関数” (891 ページ)
- “NOTPRINT 関数” (895 ページ)
- “NOTPUNCT 関数” (897 ページ)
- “NOTSPACE 関数” (900 ページ)
- “NOTUPPER 関数” (903 ページ)
- “NOTXDIGIT 関数” (906 ページ)
- “VERIFY 関数” (1233 ページ)

FINDW 関数

文字列の単語の文字位置または文字列の単語の数を返します。

カテゴリ: 文字

返されるデータの種類: DOUBLE

構文

FINDW(*string*, *word*[, *chars*])

FINDW(*string*, *word*, *chars*, *modifier(s)*[, *startpos*])

FINDW(*string*, *word*, *chars*, *startpos*[, *modifier(s)*])

FINDW(*string*, *word*, *startpos*[, *chars*[, *modifier(s)*]])

必須引数

string
文字列を評価するか、文字列に強制変換できる文字定数、変数、または式であり、検索する文字列を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

ヒント 文字のリテラル文字列を引用符で囲みます。

word
文字列を評価するか、文字列に強制変換できる文字定数、変数、または式であり、検索する単語を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

ヒント 文字のリテラル文字列を引用符で囲みます。

chars
文字のリストを初期化する文字定数、変数または式です(省略可能)。リスト中の文字は単語を区切るための区切り文字です(ただし *modifier* 引数に K 修飾子を指定しない場合)。K 修飾子を指定すると、このリストにないすべての文字が区切り文字となります。他の修飾子を使用して、このリストに文字をさらに追加できます。

文字のリストを初期化する文字定数、変数または式です(省略可能)。

リスト中の文字は単語を区切るための区切り文字です(ただし *modifier* 引数に K 修飾子を指定しない場合)。K 修飾子を指定すると、このリストにないすべての文字が区切り文字となります。他の修飾子を使用して、このリストに文字をさらに追加できます。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

ヒント 文字のリテラル文字列を引用符で囲みます。

startpos
オプションの数値定数、変数、または式であり、検索を開始する位置と検索する方向を指定する整数です。

データの種類 DOUBLE

'*modifier(s)*'
文字定数、変数または式を指定します。これらの空白以外の各文字で FINDW 関数のアクションを変更します。A (アルファベット文字をリストに追加)、C (制御文字を追加)、D (数字を追加)の大文字と小文字を修飾子として使用できます。他の修飾子も使用することができます。詳細については、*SAS DS2 言語リファレンス*を参照してください。

文字定数、変数または式を指定します。これらの空白以外の各文字で FINDW 関数のアクションを変更します。

次の文字を修飾子として使用できます。

blank
無視されます。

a or A
文字のリストにアルファベット文字を追加します。

b or B
startpos 引数の符号に関係なく、左から右へではなく右から左へ検索します。

c or C
文字のリストに制御文字を追加します。

d or D
文字のリストに数字を追加します。

e or E
指定した単語の文字列の文字位置を決定するかわりに、指定した単語が検出されるまでにスキャンした単語を数えます。単語の断片はカウントされません。

f or F
アンダースコアと英字(つまり、VALIDVARNAME=V7 を使用して SAS 変数名を開始できる文字)を文字のリストに追加します。

g or G
文字のリストにグラフィカル文字を追加します。

h or H
文字のリストに水平タブを追加します。

i or I
大文字か小文字かは無視します。

k or K
文字のリストにないすべての文字が区切り文字として扱われるようにします。K を指定しない場合、文字のリストに含まれているすべての文字が区切り文字として扱われます。K 修飾子と V 修飾子は同じ機能を実行します。

l or L
小文字を文字リストに追加します。

m or M
複数の連続する区切り文字、および *string* 引数の先頭または末尾の区切り文字が、長さがゼロの単語を参照するように指定します。

n or N
文字のリストに数字、アンダースコアおよび英文字(VALIDVARNAME=V7 を使用した SAS 変数名内に表示可能な文字)を追加します。

o or O
FINDW 関数が呼び出されるたびに、*chars* 引数と *modifier* 引数を処理するのではなく、一度のみ処理します。DS2 (WHERE 句を除く)で O 修飾子を使用すると、*chars* 引数と *modifier* 引数が変化しないループで FINDW を呼び出すと、FINDW がより速く実行されます。

p or P
文字のリストに句読点を追加します。

q or Q
引用符で囲まれた部分文字列内の区切り文字を無視します。*string* 引数の値に、対になっていない引用符が含まれている場合、左から右へのスキャンと、右から左へのスキャンとでは異なる単語が生成されます。

r or R
word 引数から先頭および末尾の区切り文字を削除します。

s or S
文字リストに空白文字(空白、水平タブ、垂直タブ、キャリッジリターン、ラインフィード、フォームフィード)を追加します。

t or T
string 引数、*word* 引数、*chars* 引数の末尾の空白をトリムします。

u or U
大文字を文字リストに追加します。

v or V
文字のリストにないすべての文字が区切り文字として扱われるようにします。V が指定されていない場合、文字のリストにあるすべての文字が区切り文字として扱われます。V 修飾子と K 修飾子は同じ機能を実行します。

w or W
印刷可能文字を文字リストに追加します。

x or X
文字のリストに 16 進文字を追加します。

データの種類　CHAR、NCHAR、NVARCHAR、VARCHAR

ヒント　*modifier* 引数を使用する場合は、*chars* 引数の後に配置する必要があります。

詳細

"区切り文字"の定義

"区切り文字"は、単語を区切るために使用されるすべての文字を示します。区切り文字は、*chars* 引数、*modifier* 引数、またはその両方を使って指定できます。Q 修飾子を指定する場合、引用符で囲まれる部分文字列内の文字は区切り文字として扱われません。

"単語"の定義

"単語"は、次の両方の特徴を持つ部分文字列を示します。

- 左側にある区切り文字で区切られる部分文字列または文字列の最初の部分文字列
- 右側にある区切り文字で区切られる部分文字列または文字列の最後の部分文字列

注: 単語には区切り文字を含めることができます。その場合、FINDW 関数は、単語に区切り文字が含まれない SCAN 関数とは異なります。

文字列の検索

FINDW 関数は、指定した単語と単語の定義の両方を満たす部分文字列を検出できない場合、値 0 を返します。

指定した単語と単語の定義の両方を満たす部分文字列を FINDW 関数が検出した場合に返される値は、E 修飾子が指定されているどうかによって異なります。

- E 修飾子を指定すると、FINDW は、指定した単語の検索中にスキャンした完全な単語の数を返します。*startpos* が単語の中心の位置を指定した場合、その単語はカウントされません。
- E 修飾子を指定しないと、FINDW は検出された部分文字列の文字位置を返します。

startpos 引数を指定した場合、*startpos* の絶対値で検索を開始する位置が指定されます。*startpos* の符号で検索する方向が指定されます。

startpos の値	アクション
0 より大きい	検索は *startpos* の位置から開始され、右に進みます。*startpos* が文字列の長さよりも大きい場合、FINDW は 0 を返します。
0 より小さい	検索は–*startpos* の位置から開始され、左に進みます。*startpos* が文字列の長さの負の値よりも小さい場合、検索は文字列の末尾から開始されます。
0	FINDW は値 0 を返します。

startpos 引数または B 修飾子を指定しない場合、FINDW は文字列の先頭から左から右に検索します。B 修飾子を指定しても、*startpos* 引数を使用しない場合、FINDW は文字列の末尾から右から左に検索します。

ASCII 環境で FINDW 関数を使用する

引数が 2 つしかない FINDW 関数を使用する場合、デフォルトの区切り文字は次のとおりです。

- お使いのコンピューターが ASCII 文字を使用している場合、デフォルトの区切り文字は次のようになります。

  空白 ! $ % & ( ) * + , - ./ ; < ^ |

null 引数の使用

FINDW 関数では、文字引数を null に指定できます。null 引数は長さがゼロの文字列として扱われます。数値引数は null にできません。

例

例 1: 文字列からの単語の検索

次のプログラムでは、文字列から単語"she"を検索し、単語の開始位置を返します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double whereisshe;
  method run();
    whereisshe=findw('She sells sea shells? Yes, she does.','she');
    put whereisshe=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
whereisshe=28
```

例 2: 文字列の検索と、chars 引数と startpos 引数の使用

次のプログラムでは、単語"rain"が 2 回現れます。FINDW は、25 番目の位置から検索を開始するため、2 番目の単語のみが検出されます。*chars* 引数は区切り文字としてスペースを指定しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double result;
  method run();
    result = findw('At least 2.5 meters of rain falls in a rain forest.',
      'rain', ' ', 25);
    put result=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
result=40
```

例 3: I 修飾子および Startpos 引数を使用した文字列の検索

次のプログラムでは、I 修飾子を使用して単語の開始位置を返します。I 修飾子を使用すると大文字と小文字が区別されなくなります。*startpos* 引数では検索の開始位置を指定します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double result;
  dcl char(75) string;
  method run();
    string='Artists from around the country display their art at
      an art festival.';
    result=findw(string, 'Art',' ', 'i', 10);
    put result=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
result=47
```

例 4: E 修飾子を使用した文字列の検索

次のプログラムでは、E 修飾子を使用して、単語"art"の検索中にスキャンされた完全な単語の数を返します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double result;
  dcl char(75) string;
  method run();
    string='Artists from around the country display their art at
      an art festival.';
    result=findw(string,'art',' ','E');
    put result=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
result=8
```

例 5: E 修飾子および Startpos 引数を使用した文字列の検索

次のプログラムでは、E 修飾子を使用して文字列の単語を数えます。単語のカウントを文字列の 50 番目の位置から開始します。"art"は 50 番目の文字位置から 3 番目の単語であるため、結果は 3 になります。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double result;
  dcl char(75) string;
  method run();
    string='Artists from around the country display their art at
      an art festival.';
    result=findw(string, 'art',' ','E',50);
    put result=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
result=3
```

例 6: 2 つの修飾子を使用した文字列の検索

次のプログラムでは、I および E 修飾子を使用して、文字列の単語を検出します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double result;
  dcl char(130) string;
  method run();
    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=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
result=5
```

例 7: R 修飾子を使用した文字列の検索

次のプログラムでは、R 修飾子を使用して、単語から先頭および末尾の区切り文字を削除します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double result;
  dcl char(75) string;
  dcl char word;
  method run();
    string='Artists from around the country display their art at
      an art festival.';
    word='  art  ';
    result=findw(string, word, ' ', 'R');
    put result=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
result=47
```

関連項目

関数:

- “COUNTW 関数” (478 ページ)
- “FIND 関数” (628 ページ)
- “FINDC 関数” (631 ページ)
- “INDEXW 関数” (709 ページ)
- “SCAN 関数” (1109 ページ)

FIPNAME 関数

2 桁の FIPS コードを大文字の州名に変換します。

カテゴリ: 州と ZIP コード

制限事項: この関数は CAS サーバーではサポートされていません。

返されるデータの種類: CHAR

構文

FIPNAME(*expression*)

必須引数

expression
米国の FIPS コードを表す数値定数、変数または式を指定します。

データの種類 CHAR()、DOUBLE

詳細

まだ長さが割り当てられていない変数に FIPNAME 関数から値が返される場合、その変数にはデフォルトで長さ 20 が割り当てられます。

FIPNAME 関数は、FIPS (U.S. Federal Information Processing Standards)コードを大文字の対応する州名または米国領名に変換し、最大 20 文字の値を返します。

比較

FIPNAME、FIPNAMEL、および FIPSTATE 関数は同じ引数を取りますが、異なる値を返します。

- FIPNAME は大文字の州名を返します。
- FIPNAMEL は、大文字と小文字が混在する州名を返します。
- FIPSTATE は、2 文字の州郵便番号(または米国領の場合は世界の GSA 地理コード)を大文字で返します。

例

次の例は、ノースカロライナ州の 2 桁の FIPS コードに対して FIPNAME、FIPNAMEL、FIPSTATE を使用した場合の違いを示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
 data _null_;
  method run();
  dcl char(15) state;
  dcl char(2) statecode;
  state=fipname(37);
  put state=;
  state=fipnamel(37);
  put state=;
  statecode=fipstate(37);
  put statecode=;
  end;
 enddata;
run;
quit;
```

これらのステートメントは、次の結果を生成します。

```
state=NORTH CAROLINA
state=North Carolina
statecode=NC
```

関連項目

関数:

- “FIPNAMEL 関数” (650 ページ)
- “FIPSTATE 関数” (651 ページ)
- “STFIPS 関数” (1155 ページ)

- “STNAME 関数” (1157 ページ)
- “STNAMEL 関数” (1158 ページ)

FIPNAMEL 関数

2 桁の FIPS コードを大文字と小文字が混在する州名に変換します。

カテゴリ: 州と ZIP コード

制限事項: この関数は CAS サーバーではサポートされていません。

返されるデータの種類: CHAR

構文

FIPNAMEL(*expression*)

必須引数

expression
米国の FIPS コードを表す数値定数、変数または式を指定します。

データの種類 CHAR()、DOUBLE

詳細

まだ長さが割り当てられていない変数に FIPNAMEL 関数から値が返される場合、その変数にはデフォルトで長さ 20 が割り当てられます。

FIPNAMEL 関数は、FIPS (U.S. Federal Information Processing Standards)コードを、大文字と小文字が混在する対応する州名または米国領名に変換し、最大 20 文字の値を返します。

比較

FIPNAME 関数と FIPNAMEL 関数はどちらも、引数として 2 桁の FIPS コードを取り、州名を返します。FIPNAME は大文字の州名を返します。FIPNAMEL は、大文字と小文字が混在する州名を返します。

例

次の例は、ノースカロライナ州の 2 桁の FIPS コードに対する FIPNAMEL 関数の結果を示しています。

```
proc ds2;
  data _null_;
  method run();
  dcl char(15) state;
  state=fipnamel(37);
  put state=;
  end;
  enddata;
run;
quit;
```

これらのステートメントは、次の結果を生成します。

```
state=North Carolina
```

関連項目

関数:

- “FIPNAME 関数” (648 ページ)
- “FIPSTATE 関数” (651 ページ)
- “STFIPS 関数” (1155 ページ)
- “STNAME 関数” (1157 ページ)
- “STNAMEL 関数” (1158 ページ)

FIPSTATE 関数

2 桁の FIPS コードを 2 文字の州郵便番号に変換します。

カテゴリ:	州と ZIP コード
制限事項:	この関数は CAS サーバーではサポートされていません。
返されるデータの種類:	CHAR(2)

構文

FIPSTATE(*expression*)

必須引数

expression
米国の FIPS コードを表す数値定数、変数または式を指定します。

詳細

まだ長さが割り当てられていない変数に FIPSTATE 関数から値が返される場合、その変数にはデフォルトで長さ 20 が割り当てられます。

FIPSTATE 関数は、FIPS (U.S. Federal Information Processing Standards)コードを、大文字の 2 文字の州郵便番号(または米国領の場合は世界の GSA 地理コード)に変換します。

例

次の例は、ノースカロライナ州の 2 桁の FIPS コードに対する FIPSTATE 関数の結果を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
 data _null_;
  method run();
  dcl char(2) statecode;
  statecode=fipstate(37);
  put statecode=;
  end;
 enddata;
run;
quit;
```

これらのステートメントは、次の結果を生成します。

```
statecode=NC
```

関連項目

関数:

- “FIPNAME 関数” (648 ページ)
- “FIPNAMEL 関数” (650 ページ)
- “STFIPS 関数” (1155 ページ)
- “STNAME 関数” (1157 ページ)

- “STNAMEL 関数” (1158 ページ)

FLOOR 関数

数値式以下の最大の整数を返します。

カテゴリ: 切り捨て

返されるデータの種類: DECIMAL、DOUBLE、NUMERIC

構文

FLOOR(*expression*)

必須引数

expression
数値に評価される有効な式を指定します。

データの種類 DECIMAL、DOUBLE、NUMERIC

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

expression が整数の 1E-12 以内の場合、関数はその整数を返します。結果が DOUBLE の範囲に収まらない数値の場合、FLOOR 関数は失敗します。

引数が DECIMAL の場合、結果は DECIMAL になります。それ以外の場合、引数は (まだ変換されていなければ) DOUBLE に変換され、結果は DOUBLE になります。

比較

FLOOR 関数は、結果が整数の 1E-12 内にある場合はその整数を返すように、結果をファジー処理します。FLOORZ 関数はゼロファジーを使用します。そのため、FLOORZ 関数では予期しない結果になる可能性があります。

例

例 1: FLOOR 関数の例

次のプログラムは、FLOOR 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double x;
  method run();
    x=floor(1.95);
    put x;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
1
```

例 2: 行列パッケージを使用した FLOOR 関数の例

次のプログラムは、FLOOR 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double a[2, 2];
  dcl double c[2, 2];

  method run();
    dcl package matrix m;
    dcl package matrix r;
    dcl double i j;

    a := (1323.43, -.72, 3.38, 45);

    m=_new_ matrix(a, 2, 2);
    r=m.floor();
    r.toarray(c);

    do i=1 to 2;
      do j=1 to 2;
        put c[i, j];
      end;
    end;
  end;
```

```
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
1323
0
3
45
```

関連項目

関数:

- “CEIL 関数” (410 ページ)
- “CEILZ 関数” (412 ページ)
- “FLOORZ 関数” (655 ページ)

FLOORZ 関数

ゼロファジーを使用して、引数より小さいか等しい整数のうち最大の値を返します。

カテゴリ: 切り捨て

返されるデータの種類: DOUBLE

構文

FLOORZ(*expression*)

必須引数

expression
数値に評価される有効な式を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

比較

FLOOR 関数とは異なり、FLOORZ 関数はゼロファジーを使用します。引数が整数の 1E-12 内にある場合、FLOOR 関数はその整数に等しくなるように結果をファジー処理します。FLOORZ 関数は結果をファジー処理しません。そのため、FLOORZ 関数では予期しない結果になる可能性があります。

例

次のプログラムは、FLOORZ 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double a b c d e f var1 var2 var6;
  method run();
    var1=2.1;
    a=floorz(var1);
    put a=;

    var2=-2.4;
    b=floorz(var2);
    put b=;

    c=floorz(-1.6);
    put c=;

    var6=(1.-1.e-13);
    d=floorz(1-1.e-13);
    put d=;

    e=floorz(763);
    put e=;

    f=floorz(-223.456);
    put f=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
a=2
b=-3
c=-2
d=0
e=763
f=-224
```

関連項目

関数:

- “CEIL 関数” (410 ページ)
- “CEILZ 関数” (412 ページ)
- “FLOOR 関数” (653 ページ)

FMTINFO 関数

SAS 出力形式または入力形式の情報を返します。

カテゴリ: 特殊

制限事項: この関数は、SAS で提供されている出力形式の情報を返します。FORMAT プロシジャで作成されるユーザー定義出力形式には使用できません。

構文

FMTINFO('*format-name*', '*information-type*');

必須引数

'*format-name*'
SAS 出力形式または入力形式の名前を指定します。

要件 *format-name* は、単一引用符で囲む必要があります。

information-type
返される情報の種類を指定します。*format-information* には、'DESC' (簡単な説明)、'MINW' (最小幅)、'MAXW' (最大幅)のいずれかの値を指定できます。追加の情報の種類が使用可能です。詳細については、*SAS DS2 言語リファレンス*を参照してください。

返される情報の種類を指定します。*format-information* には、次のいずれかの値を指定できます。

'CAT'
関数カテゴリを返します。

参照項目: 完全なリストについては、“関数カテゴリ” (250 ページ)を参照してください。

'TYPE'
format-name が出力形式、入力形式、またはその両方のいずれであるかを返します。

'DESC'
出力形式または入力形式の簡単な説明を返します。

'MIND'
出力形式または入力形式の小数点以下の最小桁数を返します。

'MAXD'
出力形式または入力形式の小数点以下の最大桁数を返します。

'DEFD'
出力形式または入力形式の小数点以下のデフォルト桁数を返します。

'MINW'
出力形式または入力形式の最小の幅の値を返します。

'MAXW'
出力形式または入力形式の最大の幅の値を返します。

'DEFW'
出力形式または入力形式のデフォルトの幅の値を返します。

制限事項 *information-type* 引数は 1 つだけ指定できます。

要件 *information-type* は、単一引用符で囲む必要があります。

詳細

FMTINFO 関数は、出力形式または入力形式の情報を返します。言語要素の種類、言語要素の説明、小数点以下の値の最小/最大/デフォルト値、幅の値の最小/最大/デフォルト値などの、出力形式または入力形式のカテゴリに関する情報を返すことができます。

FMTINFO 関数で複数の引数を指定することはできません。

FMTINFO 関数は、数値引数 MIND、MAXD、DEFD、MINW、MAXW、DEFW を含めて、すべてのデータ値に対して文字列を返します。

例

次のプログラムでは、DATE 出力形式についての情報を返します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl char(30) fdesc fcat ftype;
  dcl double fmind fmaxd fdefd fminw fmaxw fdefw;
  method run();
    ftype=fmtinfo('date','type');
    fcat= fmtinfo('date','cat');
```

```
      fdesc= fmtinfo('date','desc');
      fmind= fmtinfo('date','mind');
      fmaxd= fmtinfo('date','maxd');
      fdefd= fmtinfo('date','defd');
      fminw= fmtinfo('date','minw');
      fmaxw= fmtinfo('date','maxw');
      fdefw= fmtinfo('date','defw');
      put ftype=;
      put fcat=;
      put fdesc= ;
      put fmind= ;
      put fmaxd= ;
      put fdefd= ;
      put fminw= ;
      put fmaxw= ;
      put fdefw= ;
    end;
  enddata;
  run;
  quit;
```

次の行が SAS ログに書き込まれます。

```
ftype=BOTH
fcat=date
fdesc=date value
fmind=0
fmaxd=8
fdefd=0
fminw=5
fmaxw=11
fdefw=7
```

FNONCT 関数

F 分布の非心度パラメーターの値を返します。

カテゴリ: 数学

返されるデータの種類: DOUBLE

構文

FNONCT(*x*, *ndf*, *ddf*, *probability*)

必須引数

x
数値の確率変数です。

範囲 $x \geq 0$

データの種類 DOUBLE

ndf
数値の分子の自由度パラメーターです。

範囲 $ndf > 0$

データの種類 DOUBLE

ddf
数値の分母の自由度パラメーターです。

範囲 $ddf > 0$

データの種類 DOUBLE

probability
確率です。

範囲 $0 < probability < 1$

データの種類 DOUBLE

詳細

FNONCT 関数は、非心度 *F* 分布(パラメーターは *x*、*ndf*、*ddf*、*nc*)から負でない非心度パラメーターを返します。*probability* が心度 *F* 分布(パラメーターは *x*、*ndf* および *ddf*)からの確率よりも大きい場合、この問題の平方根は存在しません。この場合、欠損値が返されます。式の負でない平方根 *nc* を検索するには、ニュートン型のアルゴリズムを使用します。

$$P_f(x|ndf, ddf, nc) - prob = 0$$

前述の式には次の関係が適用されます。

$$P_f(x|ndf, ddf, nc) = \varepsilon^{\frac{-nc}{2}} \sum_{j=0}^{\infty} \frac{\left(\frac{nc}{2}\right)^j}{j!} I_{\frac{(ndf)x}{ddf+(ndf)x}}\left(\frac{ddf}{2} + j, \frac{ddf}{2}\right)$$

この式では、$I(...)$は、次の式によって得られるベータ分布の確率です。

$$I_x(a,b) = \frac{}{\frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}} \int_0^x t^{a-1}(1-t)^{b-1} dt$$

アルゴリズムで固定小数点を収束できない場合、欠損値が返されます。

例

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data work /overwrite=yes;
  dcl double x df ddf nc prob ncc;
  method init();
    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;
  end;
enddata;
run;
quit;

proc print data=work;
run;
```

アウトプット 7.9　FNONCT 関数からの出力

The SAS System

Obs	x	df	ddf	nc	prob	ncc
1	2	4	5	1.0	0.69277	1.00000
2	2	4	5	1.5	0.65701	1.50000
3	2	4	5	2.0	0.62232	2.00000
4	2	4	5	2.5	0.58878	2.50000
5	2	4	5	3.0	0.55642	3.00000

関連項目

関数:

- “CNONCT 関数” (422 ページ)
- “TNONCT 関数” (1195 ページ)

FUZZ 関数

引数が整数の 1E-12 以内の場合、最も近い整数を返します。

カテゴリ: 切り捨て

返されるデータの種類: DOUBLE

構文

FUZZ(*expression*)

必須引数

expression
数値に評価される有効な式を指定します。

データの種類 DOUBLE

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

詳細

式が整数の 1E-12 以内の場合(整数と引数間の絶対差が 1E-12 よりも小さい場合)、FUZZ 関数は最も近い整数値を返します。それ以外の場合は、その式を返します。

例

次のプログラムは、FUZZ 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
/**** returns nearest whole number ******/
proc ds2;
data test (overwrite=yes);
  dcl double var1 x;
  method run();
    var1=5.9999999999999;
    x=put(fuzz(var1),16.14);
    put x;
  end;
```

```
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
6
```

```
/**** returns the original expression ******/
proc ds2;
data test (overwrite=yes);
  dcl double var1 x;
  method run();
    var1=5.99999999;
    x=put(fuzz(var1),16.14);
    put x;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
5.99999999
```

GAMINV 関数

ガンマ分布から分位点を返します。

カテゴリ: 分位点

返されるデータの種類: DOUBLE

構文

GAMINV(*p*, *a*)

必須引数

p
数値の確率に評価される有効な式を指定します。

範囲 $0 \le p < 1$

データの種類 DOUBLE

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

a
数値の形状パラメーターに評価される有効な式を指定します。

範囲 $a > 0$

データの種類 DOUBLE

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

詳細

GAMINV 関数は、ガンマ分布(形状パラメーターは *a*)の *p*分位点を返します。ガンマ分布の行が返される分位点以下になる確率は *p* です。

注: GAMINV は、PROBGAM 関数の逆数です。

例

次のプログラムは、GAMINV 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double x y;
  method run();
     x=gaminv(0.5, 9);
     y=gaminv(.1, 2.1);
     put x;
     put y;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
8.66895118437038
0.5841932368992
```

関連項目

関数:

- "PROBGAM 関数" (991 ページ)

GAMMA 関数

ガンマ関数の値を返します。

カテゴリ: 数学

返されるデータの種類: DOUBLE

構文

GAMMA(*expression*)

必須引数

expression
数値に評価される有効な式を指定します。

制限事項 正でない整数は無効です。

データの種類 DOUBLE

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

詳細

GAMMA 関数は、次の式で求められる積分を返します。

$$\mathrm{GAMMA}(x) = \int_0^{\infty} t^{x-1} e^{-t}\, dt.$$

正の整数の場合、GAMMA(x)は(x − 1)!になります。この関数は、一般的に$\Gamma(x)$で表されます。

例

次のプログラムは、GAMMA 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
```

```
data _null_;
  dcl double x;
  method run();
    x=gamma(6);
    put x=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
120
```

GARKHCLPRC 関数

Garman-Kohlhagen モデルに基づいて、株式のヨーロピアンオプションのコール価格を計算します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

GARKHCLPRC(*E*, *t*, *S*, R_d, R_f, *sigma*)

必須引数

E
権利行使価格を指定する正の非欠損値。

要件 *E* および *S* は同じ単位で指定します。

データの種類 DOUBLE

t
満期までの時間を指定する非欠損値です。

データの種類 DOUBLE

S
通貨の現物価格を指定する非欠損値の正の値です。

要件 *S* および *E* は同じ単位で指定します。

データの種類 DOUBLE

R_d
期間 *t* の国内の安全利率を指定する非欠損値の正の分数です。

要件 単位 *t* と同じ期間の R_d の値を指定します。

データの種類 DOUBLE

R_f
期間 *t* の海外の安全利率を指定する非欠損値の正の分数です。

要件 単位 *t* と同じ期間の R_f の値を指定します。

データの種類 DOUBLE

sigma
為替相場のボラティリティを 指定する非欠損値の正の分数です。

要件 単位 *t* と同じ期間の *sigma* の値を指定します。

データの種類 DOUBLE

詳細

GARKHCLPRC 関数は、Garman-Kohlhagen モデルに基づいて、株式のヨーロピアンオプションのコール価格を計算します。この関数は次の関係に基づきます。

$$\text{CALL} = SN(d_1)\left(\varepsilon^{-R_f t}\right) - EN(d_2)\left(\varepsilon^{-R_d t}\right)$$

引数

S
通貨の現物価格を指定します。

N
累積正規密度関数を指定します。

E
オプションの権利行使価格を指定します。

t
有効期限までの時間を指定します。

R_d
期間 *t* の国内の安全利率を指定します。

R_f
期間 *t* の海外の安全利率を指定します。

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

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

前述の式には次の引数が適用されます。

σ
原資産のボラティリティを指定します。

σ^2
利益率の分散を指定します。

t=0 となる特別な場合には、次の式が真です。

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

価格設定の基本については、*"価格関数の使用" (SAS 関数と CALL ルーチン: リファレンス)*を参照してください。

比較

GARKHCLPRC 関数は、Garman-Kohlhagen モデルに基づいて、株式のヨーロピアンオプションのコール価格を計算します。GARKHPTPRC 関数は、Garman-Kohlhagen モデルに基づいて、株式のヨーロピアンオプションのプット価格を計算します。これらの関数はスカラー値を返します。

例

次のプログラムは、GARKHCLPRC 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double a b;
  method run();
    a=garkhclprc(40, .5, 38, .06, .04, .2);
    b=garkhclprc(19, .25, 20, .05, .03, .09);
    put a=;
    put b=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
1.44942510595479
1.1304209447635
```

関連項目

関数:

■ “GARKHPTPRC 関数” (669 ページ)

GARKHPTPRC 関数

Garman-Kohlhagen モデルに基づいて、株式のヨーロピアンオプションのプット価格を計算します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

GARKHPTPRC(*E*, *t*, *S*, R_d, R_f, *sigma*)

必須引数

E

権利行使価格を指定する正の非欠損値。

要件 *E* および *S* は同じ単位で指定します。

データの種類 DOUBLE

t

満期までの時間を年単位で指定する非欠損値。

データの種類 DOUBLE

S

通貨の現物価格を指定する非欠損値の正の値です。

要件 *S* および *E* は同じ単位で指定します。

データの種類 DOUBLE

R_d

期間 *t* の国内の安全利率を指定する非欠損値の正の分数です。

要件 単位 *t* と同じ期間の R_d の値を指定します。

データの種類 DOUBLE

R_f

期間 *t* の海外の安全利率を指定する非欠損値の正の分数です。

要件 単位 *t* と同じ期間の R_t の値を指定します。

データの種類 DOUBLE

sigma
為替相場のボラティリティを 指定する非欠損値の正の分数です。

データの種類 DOUBLE

詳細

GARKHPTPRC 関数は、Garman-Kohlhagen モデルに基づいて、株式のヨーロピアンオプションのプット価格を計算します。この関数は次の関係に基づきます。

$$\text{PUT} = \text{CALL} - S\left(\varepsilon^{-R_f t}\right) + E\left(\varepsilon^{-R_d t}\right)$$

引数

S
通貨の現物価格を指定します。

E
オプションの権利行使価格を指定します。

t
有効期限までの時間を年単位で指定します。

R_d
期間 *t* の国内の安全利率を指定します。

R_f
期間 *t* の海外の安全利率を指定します。

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

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

前述の式には次の引数が適用されます。

σ
原資産のボラティリティを 指定します。

σ^2
利益率の分散を指定します。

t=0 となる特別な場合には、次の式が真です。

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

価格設定の基本については、*“価格関数の使用” (SAS 関数と CALL ルーチン: リファレンス)*を参照してください。

比較

GARKHPTPRC 関数は、Garman-Kohlhagen モデルに基づいて、株式のヨーロピアンオプションのプット価格を計算します。GARKHCLPRC 関数は、Garman-

Kohlhagen モデルに基づいて、株式のヨーロピアンオプションのコール価格を計算します。これらの関数はスカラー値を返します。

例

次のプログラムは、GARKHPTPRC 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  method run();
    a=garkhptprc(50, .7, 55, .05, .04, .2);
    b=garkhptprc(32, .3, 33, .05, .03, .3);
    put a=;
    put b=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
1.4050880944848
1.56473205137371
```

関連項目

関数:

- “GARKHCLPRC 関数” (666 ページ)

GCD 関数

整数の集合に対する最大公約数を返します。

カテゴリ:	数学
返されるデータの種類:	DOUBLE
注:	2025.12 以降、この関数では、引数が DOUBLE の場合、すべての引数が DOUBLE に変換され、結果は DOUBLE になります。PROC DS2 ステートメントで BIGINTPROCESSING=YES が設定され、いずれかの引数が BIGINT の場合、すべての引数が BIGINT に変換され、結果は BIGINT になります。

構文

GCD(*expression-1*, *expression-2* [, ...*expression-n*])

必須引数

expression
数値に評価される有効な式を指定します。

要件 少なくとも 2 つの引数が必要です。

データの種類 DOUBLE

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

詳細

GCD (最大公約数)関数は、1 つ以上の整数の最大公約数を返します。たとえば、30 と 42 の最大公約数は 6 になります。最大公約数は、最高共通因子とも呼ばれます。

例

次のプログラムは、GCD 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double x y;
  method run();
    x=gcd(10, 15);
    y=gcd(36, 45);
    put x;
    put y;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
5
9
```

関連項目

関数:

- “LCM 関数” (804 ページ)

GEODIST 関数

2 つの緯度と経度の座標間の測地距離を返します。

カテゴリ: 距離

返されるデータの種類: DOUBLE

構文

GEODIST(*latitude-1*, *longitude-1*, *latitude-2*, *longitude-2* [,*options*])

必須引数

latitude
赤道の南北にある特定の場所の座標を指定する数値定数、変数または式です。赤道の北にある座標は正の値、赤道の南にある座標は負の値になります。

制限事項 値を度数で表す場合、90 から–90 までの値にする必要があります。値をラジアンで表す場合、pi/2 から-pi/2 までの値にする必要があります。

データの種類 DOUBLE

longitude
英国のグリニッジを通過するグリニッジ子午線の東西にある特定の場所の座標を指定する数値定数、変数または式です。グリニッジ子午線の東にある座標は正の値、グリニッジ子午線の西にある座標は負の値になります。

制限事項 値を度数で表す場合、540 から–540 までの値にする必要があります。値をラジアンで表す場合、3*pi から-3*pi までの値にする必要があります。

データの種類 DOUBLE

オプション引数

options
距離の記述方法を示す文字定数を指定します。

次のディスクリプターがサポートされています。

M
マイル単位で距離を指定します。

K
キロメートル単位で距離を指定します。K は、距離のデフォルト値です。

D
入力値を度数で表すように指定します。D は、入力値のデフォルトです。

R
入力値をラジアンで表すように指定します。

データの種類　CHAR、NCHAR、NVARCHAR、VARCHAR

詳細

GEODIST 関数は、2 つの緯度と経度の座標間の測地距離を計算します。入力値は、度数またはラジアンで表すことができます。

例

例 1: 測地距離(キロメートル単位)の計算

次のプログラムでは、AL の Mobile (緯度 30.68 N、経度 88.25 W)と NC の Asheville (緯度 35.43 N、経度 82.55 W)間の測地距離(キロメートル単位)を示します。プログラムでは、デフォルトの K オプションが使用されます。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double distance;
  method run();
    distance=geodist(30.68, -88.25, 35.43, -82.55);
    put 'Distance= ' distance 'kilometers';
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
Distance= 748.652914703183 kilometers
```

例 2: 測地距離(マイル単位)の計算

次のプログラムでは、M オプションを使用して、AL の Mobile (緯度 30.68 N、経度 88.25 W)と NC の Asheville (緯度 35.43 N、経度 82.55 W)間の測地距離(マイル単位)を計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double distance;
  method run();
    distance=geodist(30.68, -88.25, 35.43, -82.55, 'M');
    put 'Distance = ' distance 'miles';
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
Distance = 465.191354181072 miles
```

例 3: 度数単位の入力を使用した測地距離の計算

次のプログラムでは、度数で表される緯度と経度の値を使用して、2 つの場所の測地距離を計算します。プログラムでは、D オプションと M オプションの両方が指定されます。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double distance lat1 long1 lat2 long2;
  method run();
    lat1=35.2;
    long1=-78.1;
    lat2=37.6;
    long2=-79.8;
    Distance = geodist(lat1,long1,lat2,long2,'DM');
    put 'Distance= ' Distance 'miles';
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
Distance= 190.683975083577 miles
```

例 4: ラジアン単位の入力を使用した測地距離の計算

次のプログラムでは、ラジアンで表される緯度と経度の値を使用して、2 つの場所の測地距離を計算します。プログラムでは、GEODIST 関数を実行する前に度数がラジアンに変換されます。このプログラムでは、R オプションと M オプションの両方が指定されます。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double distance pi lat1 long1 lat2 long2;
  method run();
    lat1=35.2;
    long1=-78.1;
    lat2=37.6;
    long2=-79.8;
    pi = constant('pi');
    lat1 = (pi*lat1)/180;
    long1 = (pi*long1)/180;
    lat2 = (pi*lat2)/180;
    long2 = (pi*long2)/180;
    Distance = geodist(lat1,long1,lat2,long2,'RM');
    put 'Distance= ' Distance 'miles';
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
Distance= 190.683975083578 miles
```

参考文献

Vincenty, T. 1975. “Direct and Inverse Solutions of Geodesics on the Ellipsoid with Application of Nested Equations.” *Survey Review* 22: 99-93.

GEOMEAN 関数

幾何平均を返します。

カテゴリ:	記述統計
返されるデータの種類:	DOUBLE

構文

GEOMEAN(*expression* [, ...*expression*])

必須引数

expression
負でない数値に評価される有効な式です。

データの種類 DOUBLE

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

詳細

いずれかの引数が負の場合、結果は null または欠損値になります。負の引数が無効であること示すメッセージがログに表示されます。いずれかの引数がゼロの場合、幾何平均はゼロになります。すべての引数が null または欠損値の場合、結果は null または欠損値になります。それ以外の場合、結果は非 null または非欠損値の幾何平均になります。

nを非 null または非欠損値を持つ引数の数とし、$x_1, x_2, \ldots, x_n$をそれらの引数の値とします。幾何平均は、次の値の積のn^{th}の平方根です。

$$\sqrt[n]{(x_1 * x_2 * \ldots * x_n)}$$

同様に、幾何平均はこの式で示されます。

$$\exp\left(\frac{(\log(x_1) + \log(x_2) + \ldots + \log(x_n))}{n}\right)$$

多くの場合、浮動小数点の計算では僅かな数値誤差が生じます。正確に計算すればゼロになる計算でも、浮動小数点の計算が使用されると僅かな非ゼロ値になる場合があります。そのため、GEOMEAN はほぼゼロの引数の値をファジー処理します。ある引数の値が最大の引数と比較して極端に小さい場合、前者の引数はゼロとして扱われます。極端に小さい値を SAS でファジー処理しない場合、GEOMEANZ 関数を使用します。

比較

MEAN 関数は算術平均(平均)、HARMEAN 関数は調和平均、GEOMEAN 関数は非 null または非欠損値の幾何平均を返します。GEOMEANZ や GEOMEAN とは異なり、ほぼゼロの引数の値をファジー処理します。

例

次のプログラムは、GEOMEAN 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double x1 x2;
  method run();
    x1=geomean(1,2,4,4);
    x2=geomean(.,4,12,24);
    put x1=;
    put x2=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
x1=2.37841423000544
x2=10.4829655768355
```

関連項目

関数:

- “GEOMEANZ 関数” (678 ページ)
- “HARMEAN 関数” (687 ページ)
- “HARMEANZ 関数” (689 ページ)
- “MEAN 関数” (849 ページ)

GEOMEANZ 関数

ゼロファジーを使用して、幾何平均を返します。

カテゴリ:	記述統計
返されるデータの種類:	DOUBLE

構文

GEOMEANZ(*expression* [, ...*expression*])

必須引数

expression
負でない数値に評価される有効な式を指定します。

データの種類 DOUBLE

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

詳細

いずれかの引数が負の場合、結果は null または欠損値になります。負の引数が無効であること示すメッセージがログに表示されます。いずれかの引数がゼロの場合、幾何平均はゼロになります。すべての引数が null または欠損値の場合、結果は null または欠損値になります。それ以外の場合、結果は非 null または非欠損値の幾何平均になります。

nを非 null または非欠損値を持つ引数の数とし、$x_1, x_2, \ldots, x_n$をそれらの引数の値とします。幾何平均は、次の値の積のn^{th}の平方根です。

$$\sqrt[n]{(x_1 * x_2 * \ldots * x_n)}$$

同様に、幾何平均はこの式で示されます。

$$\exp\left(\frac{(\log(x_1) + \log(x_2) + \ldots + \log(x_n))}{n}\right)$$

比較

MEAN 関数は算術平均(平均)、HARMEAN 関数は調和平均、GEOMEANZ 関数は非 null または非欠損値の幾何平均を返します。GEOMEAN や GEOMEANZ とは異なり、ほぼゼロの引数の値をファジー処理しません。

例

次のプログラムは、GEOMEANZ 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
```

```
data _null_;
  dcl double x1 x2;
  method run();
    x1=geomeanz(1,2,4,4);
    x2=geomeanz(.,4,12,24);
    put x1=;
    put x2=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
x1=2.37841423000544
x2=10.4829655768355
```

関連項目

関数:

- “GEOMEAN 関数” (676 ページ)
- “HARMEAN 関数” (687 ページ)
- “HARMEANZ 関数” (689 ページ)
- “MEAN 関数” (849 ページ)

GETTHREADERRORSTATUS 関数

現在の DS2 スレッドのランタイムエラーステータスを取得します。

カテゴリ:	エラーチェック
返されるデータの種類:	INTEGER

構文

GETTHREADERRORSTATUS()

引数なし

GETTHREADERRORSTATUS()関数には引数がありません。

詳細

GETTHREADERRORSTATUS()関数は、スレッド内で致命的ではないランタイムエラーまたは警告が発生するたびに更新される内部ステータス変数をクエリし、変数のステータスを返します。デフォルトでは、エラーステータスの戻り値は次のとおりです。

0
: エラーは発生していません

1
: 致命的ではないエラーが発生しました

2
: 警告が発生しました

デフォルトでは、内部変数は無効なパラメーター値や無効なデータ型変換などのプログラミングエラーを検出します。同じスレッド内で警告の前にエラーが発生した場合、エラー値が警告よりも優先されます。

SET ステートメントとともに使用すると、この関数は入力データセットの各行のエラーステータスをチェックし、_N_事前定義変数と組み合わせて使用すると、エラーの行番号を返すために使用できます。RUN メソッドの各ループの後、エラーステータスは 0 (ゼロ)にリセットされます。

SETTHREADERRORSTATUS 関数および IF ステートメントとともに GETTHREADERRORSTATUS 関数を使用すると、ユーザー指定の条件に対するカスタムエラーステータス値を返すことができます。SETTHREADERRORSTATUS 関数を使用すると、内部ステータス変数のカスタムエラーステータス値と、オプションのユーザー指定メッセージを定義できます。SETTHREADERRORSTATUS 関数を使用して、内部エラーステータス変数を明示的に 0 (ゼロ)にリセットすることもできます。

この関数は、シングルスレッド、マルチスレッド、または部分スレッド DS2 プログラムの現在のスレッドで実行されます。マルチスレッド環境で実行すると、GETTHREADERRORSTATUS()関数は、DS2 THREAD ブロックまたは DS2 DATA ブロックを実行しているスレッドのエラーステータス値を返します。DS2 THREAD ブロックまたは DS2 DATA ブロックを実行している各スレッドは、個別のエラーステータス値を保持します。たとえば、THREADS=5 を指定した SET FROM ステートメントによって 5 つのスレッドが生成された場合、DS2 THREAD ブロックを実行している 5 つのスレッドはそれぞれ、独自のエラーステータス値を維持します。さらに、DATA ブロックを実行しているスレッドも独自のエラーステータス値を維持するため、DS2 プログラムに対して 6 つのエラーステータス値が維持されます。

GETTHREADERRORSTATUS 関数は、DATA、PACKAGE、または THREAD ブロック内のメソッドを実行するときにエラーを検出するのに役立ちます。

例

例 1: シングルスレッド環境での GETTHREADERRORSTATUS 関数の使用

次のプログラムは、SET ステートメントで GETTHREADERRORSTATUS 関数を使用する方法を示しています。FACT 関数は、sashelp.class データセットの Height 列に対して実行されます。GETTHREADERRORSTATUS 関数は、19 行のデータセットのうち、負の値または小数値が格納されているためにエラーを報告する行を特定します。

```
data indata;
 set sashelp.class;
run;

proc ds2;
ds2_options msgorder=temporal;                    /* 1 */
data _null_;
 method run();
   set indata;
   fact(Height);                                  /* 2 */
   if (getThreadErrorStatus() ne 0) then
      put 'ERROR for row' _n_;                    /* 3 */
 end;
enddata;
run;
quit;
```

1 DS2_OPTIONS ステートメントで MSGORDER=TEMPORAL を指定すると、SAS ログに PUT ステートメントのメッセージと診断メッセージが分散されます。

2 FACT 関数は、指定された列で負の値または小数値が見つかった場合にエラーを報告するように指定します。

3 自動_N_変数は、エラーが発生した各行の行番号を出力するために PUT ステートメントで指定されるユーザー定義エラーメッセージに含まれています。

ログには、エラーに関する情報がメッセージのペアで提供されます。各ペアの最初の行は、エラーの原因となった操作のシステム生成ログ行番号と、関数が返したエラーメッセージです。FACT 関数はログの 91 行目にあります。2 行目は PUT ステートメントによって定義され、19 行の入力データセット内のエラーの場所を特定します。

```
... [snip] ...
ERROR: Line 91: Invalid argument to function fact.
ERROR for row 2
ERROR: Line 91: Invalid argument to function fact.
ERROR for row 3
ERROR: Line 91: Invalid argument to function fact.
ERROR for row 4
ERROR: Line 91: Invalid argument to function fact.
ERROR for row 5
ERROR: Line 91: Invalid argument to function fact.
ERROR for row 6
ERROR: Line 91: Invalid argument to function fact.
ERROR for row 7
ERROR: Line 91: Invalid argument to function fact.
ERROR for row 8
ERROR: Line 91: Invalid argument to function fact.
ERROR for row 9
ERROR: Line 91: Invalid argument to function fact.
ERROR for row 11
ERROR: Line 91: Invalid argument to function fact.
ERROR for row 12
ERROR: Line 91: Invalid argument to function fact.
ERROR for row 13
ERROR: Line 91: Invalid argument to function fact.
ERROR for row 14
ERROR: Line 91: Invalid argument to function fact.
ERROR for row 16
ERROR: Line 91: Invalid argument to function fact.
ERROR for row 18
ERROR: Line 91: Invalid argument to function fact.
ERROR for row 19
```

例 2: スレッドプログラムでの GETTHREADERRORSTATUS 関数の使用

この例では、10,000 行を含む LARGENUMBERSET という名前のテーブルを作成します。このテーブルの 1000 行目ごとに null 値が含まれます。SETTHREADERRORSTATUS 関数と GETTHREADERRORSTATUS 関数は、IF ステートメントを使用してスレッドプログラムで指定され、値の部分的な合計から null 値を持つ行を識別して省略します。次に、スレッドプログラムが 2 つのスレッドで実行され、数値の最終的な合計が作成されます。

```
proc ds2;
   data largeNumberSet (overwrite=yes);                  /* 1 */
      dcl int row;
      method run();
        dcl int x;
        do x = 1 to 10000;
           if (mod(x, 1000) = 0) then row = NULL;
           else row = x;
           output;
        end;
      end;
   enddata;
   run;
quit;

proc ds2;                                          /* 2 */
   thread processDataset / overwrite=yes;
```

```
    dcl int partial_sum;
    retain partial_sum;

    /* Set the partial sum to 0 */
    method init();
      partial_sum = 0;
    end;

    /* Start processing each row */
    method run();
      set largeNumberSet;
      if (missing(row) or null(row)) then
        do;
          setThreadErrorStatus(4);              /* 3 */
          put 'Row' _n_ 'on thread' _threadid_ 'was null';
        end;
      if (getThreadErrorStatus() = 0) then partial_sum = partial_sum + row;
    end;

    method term();
      output;
    end;
  endthread;

  data _null_;                                  /* 4 */
    dcl bigint  summation;
    dcl thread  processDataset t;
    retain summation;

    /* Initialize summation to 0 */
    method init();
      summation = 0;
    end;

    method run();
      set from t threads=2;
      summation = summation + partial_sum;
    end;

    method term();
      put summation=;
    end;
  enddata;
run;
quit;
```

1 DATA ブロックはテーブルを作成し、1000 行目ごとに null 値を挿入します。

2 THREAD ブロックは次のことを行います。

- テーブルを入力データセットとして設定する
- カスタムエラー条件を定義する IF ステートメントを指定する
- SETTHREADERRORSTATUS 関数を発行して、エラー条件を満たすすべての行にカスタムエラーステータス値を設定する
- エラーを返さない行に対して部分合計計算を実行するように指定する

3 SETTHREADERRORSTATUS 関数は、エラーステータス値 4 を割り当てます。選択した任意のエラーステータス値を指定できます。PUT ステートメントには、ユーザー指定のエラーメッセージ内の省略された各行の行番号とスレッド ID を出力するための自動変数_N_および_THREADID_が含まれています。

4 DATA ブロックはスレッドプログラムを実行し、SUMMATION=値を計算します。

エラーステータス値はカスタムエラー条件を指定するため、ログにはユーザー定義のエラーメッセージのみが含まれます。

```
... [ snip] ...
Row 808 on thread 1 was null
Row 1000 on thread 2 was null
Row 2000 on thread 2 was null
Row 1808 on thread 1 was null
Row 3000 on thread 2 was null
Row 4000 on thread 2 was null
Row 5000 on thread 2 was null
Row 6000 on thread 2 was null
Row 7000 on thread 2 was null
Row 8000 on thread 2 was null
summation=49950000
```

例 3: パッケージエラーが GETTHREADERRORSTATUS の結果に与える影響

この例は、パッケージメソッドが原因でスレッドプログラムとデータプログラムのエラーが発生した場合の GETTHREADERRORSTATUS 関数の結果を示しています。

```
proc ds2;
   package mypkg / inline;                    /* 1 */
      method getFactorial(int x);
         fact(x);
      end;
   endpackage;

   thread mythread;                           /* 2 */
      dcl double result;
      method run();
      dcl package mypkg p();
      dcl int status;
      p.getFactorial(-3);
      status = getThreadErrorStatus();
      put 'The error status was' status 'on thread' _threadid_ 'after an error occurred in the
PACKAGE.';
      end;
   endthread;

   data _null_;                               /* 3 */
      dcl thread mythread t();

      method run();
       set from t threads=4;
      end;

      method term();
      dcl package mypkg m();
```

```
      dcl int status;
      m.getFactorial(-2);
      status = getThreadErrorStatus();
      put 'The error status was' status 'in the DATA block after an error occurred in the
PACKAGE.';
      end;
   enddata;
run;
quit;
```

1 パッケージ MYPKG は、整数を入力として受け入れるメソッド GETFACTORIAL を作成します。このメソッドは、指定された列で負の値または小数値が見つかった場合にエラーを報告します。

2 MYTHREAD という名前のスレッドプログラムが定義されています。スレッドプログラムは次のことを行います。

- RESULT という名前のグローバル変数を DOUBLE として宣言します。
- パッケージ MYPKG をパッケージ変数 P()としてインスタンス化します。
- ローカル変数 STATUS を整数として宣言します。
- パッケージ変数 P()を使用して GETFACTORIAL メソッドを呼び出し、入力値-3 を指定します。値-3 を指定すると、GETFACTORIAL メソッドでエラーが発生します。
- 割り当てステートメントを使用して GETTHREADERRORSTATUS 関数を呼び出し、結果を STATUS 変数に割り当てます。
- パッケージ内でエラーが発生した後、PUT ステートメントを使用して、エラーステータスとスレッド ID を含むメッセージをログに記録します。

3 データプログラムは次のことを行います。

- スレッド MYTHREAD を T()として宣言します。
- スレッドプログラム T を入力として設定し、それを 4 つのスレッドで実行するように指定します。
- パッケージ MYPKG を M()としてインスタンス化します。
- STATUS という名前のローカル変数を宣言します。
- パッケージ変数 M()を使用して GETFACTORIAL メソッドを呼び出し、入力値-2 を指定します。値-2 を指定すると、GETFACTORIAL メソッドでエラーが発生します。
- 割り当てステートメントを使用して GETTHREADERRORSTATUS 関数を呼び出し、結果を STATUS 変数に割り当てます。
- パッケージ内でエラーが発生した後、PUT ステートメントを使用してエラーステータスのメッセージをログに記録します。

ログでは、カスタムエラーメッセージがシステム生成の ERROR メッセージの前に表示されます。システム生成の ERROR メッセージでは、行 83 がエラーの場所として識別されます。行 83 は、パッケージ定義内の FACT 関数のログ行番号です。

```
... [ snip] ...
The error status was 1 on thread 1 after an error occurred in the PACKAGE.
The error status was 1 on thread 3 after an error occurred in the PACKAGE.
The error status was 1 on thread 2 after an error occurred in the PACKAGE.
The error status was 1 on thread 4 after an error occurred in the PACKAGE.
The error status was 1 in the DATA block after an error occurred in the PACKAGE.
NOTE: Created thread mythread in data set work.mythread.
ERROR: Line 83: Invalid argument to function fact.
ERROR: Line 83: Invalid argument to function fact.
ERROR: Line 83: Invalid argument to function fact.
ERROR: Line 83: Invalid argument to function fact.
ERROR: Line 83: Invalid argument to function fact.
NOTE: Execution succeeded.No rows affected.
113     quit;
```

関連項目

概念:

- “DS2 スレッドで役立つ自動変数” (*SAS DS2 プログラマガイド*)

関数:

- “SETTHREADERRORSTATUS 関数” (1128 ページ)

ステートメント:

- “DS2_OPTIONS ステートメント” (1348 ページ)

HARMEAN 関数

調和平均を返します。

カテゴリ:	記述統計
返されるデータの種類:	DOUBLE

構文

HARMEAN(*expression* [, ...*expression*])

必須引数

expression
負でない数値に評価される有効な式を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

いずれかの引数が負の場合、結果は null または欠損値になります。負の引数が無効であること示すメッセージがログに表示されます。すべての引数が null または欠損値の場合、結果は null または欠損値になります。それ以外の場合、結果は非 null または非欠損値の調和平均になります。

いずれかの引数がゼロの場合、調和平均はゼロになります。それ以外の場合、調和平均は値の逆数の算術平均の逆数になります。

nを非 null または非欠損値を持つ引数の数とし、$x_1, x_2, \ldots, x_n$をそれらの引数の値とします。調和平均はこの式で示されます。

$$\frac{n}{\frac{1}{x_1}+\frac{1}{x_2}+\ldots+\frac{1}{x_n}}$$

多くの場合、浮動小数点の計算では僅かな数値誤差が生じます。正確に計算すればゼロになる計算でも、浮動小数点の計算が使用されると僅かな非ゼロ値になる場合があります。そのため、HARMEAN はほぼゼロの引数の値をファジー処理します。ある引数の値が最大の引数と比較して極端に小さい場合、前者の引数はゼロとして扱われます。極端に小さい値を SAS でファジー処理しない場合、HARMEANZ 関数を使用します。

比較

MEAN 関数は算術平均(平均)、GEOMEAN 関数は幾何平均、HARMEAN 関数は非 null または非欠損値の調和平均を返します。HARMEANZ や HARMEAN とは異なり、ほぼゼロの引数の値をファジー処理します。

例

次のプログラムは、HARMEAN 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double x1 x2;
  method run();
    x1=harmean(1,2,4,4);
    x2=harmean(.,4,12,24);
    put x1=;
    put x2=;
```

```
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
x1=2
x2=8
```

関連項目

関数:

- “GEOMEAN 関数” (676 ページ)
- “GEOMEANZ 関数” (678 ページ)
- “HARMEANZ 関数” (689 ページ)
- “MEAN 関数” (849 ページ)

HARMEANZ 関数

ゼロファジーを使用して、調和平均を返します。

カテゴリ:	記述統計
返されるデータの種類:	DOUBLE

構文

HARMEANZ(*expression* [, ...*expression*])

必須引数

expression
負でない数値に評価される有効な式を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

いずれかの引数が負の場合、結果は null または値になります。負の引数が無効であること示すメッセージがログに表示されます。すべての引数が null または値の場合、結果は null または値になります。それ以外の場合、結果は非 null または非欠損値の調和平均になります。

いずれかの引数がゼロの場合、調和平均はゼロになります。それ以外の場合、調和平均は値の逆数の算術平均の逆数になります。

nを非 null または非欠損値を持つ引数の数とし、$x_1, x_2, \ldots, x_n$をそれらの引数の値とします。調和平均はこの式で示されます。

$$\frac{n}{\frac{1}{x_1} + \frac{1}{x_2} + \ldots + \frac{1}{x_n}}$$

比較

MEAN 関数は算術平均(平均)、GEOMEAN 関数は幾何平均、HARMEANZ 関数は非 null または非欠損値の調和平均を返します。HARMEAN や HARMEANZ とは異なり、ほぼゼロの引数の値をファジー処理しません。

例

次のプログラムは、HARMEANZ 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double x1 x2;
  method run();
    x1=harmeanz(1,2,4,4);
    x2=harmeanz(.,4,12,24);
    put x1= x2=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
x1=2 x2=8
```

関連項目

関数:

- “GEOMEAN 関数” (676 ページ)
- “GEOMEANZ 関数” (678 ページ)
- “HARMEAN 関数” (687 ページ)
- “MEAN 関数” (849 ページ)

HBOUND 関数

配列の上限を返します。

カテゴリ: 配列

返されるデータの種類: INTEGER

構文

HBOUND(*array-name*[, *bound-n*])

必須引数

array-name
一時配列または変数配列の名前を指定します。

データの種類 CHAR

オプション引数

bound-n
多次元配列にある上限を確認する次元を指定する、数値定数、変数または式です。*bound-n* の値を指定しない場合、HBOUND 関数は配列の最初の次元の上限を返します。

bound-n は整数値に評価されます。

データの種類 INTEGER

詳細

HBOUND 関数は、1 次元配列の上限、または多次元配列の指定した次元の上限を返します。

HBOUND と LBOUND を併用して、配列の次元の上限と下限の値を返すことができます。

配列の次元外にある次元値を使用して HBOUND 関数が呼び出された場合、ランタイムエラーが発生し、関数は NULL 整数値を返します。

比較

- DIM は、配列次元の要素数を返します。
- HBOUND は、配列次元の上限の値を返します。
- LBOUND は、配列次元の下限の値を返します。
- NDIMS は、配列の次元数を返します。

例

次のプログラムは、DIM、HBOUND、LBOUND、NDIMS 配列関数の使用方法を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl char(15) a1[4];
  dcl double a2[2,3,4] sum;
  method init();
    declare char(15) a1[4];
    declare double   a2[2,3,4] sum;
    a1 := ('red' 'yellow' 'green' 'blue');
    a2 := (24*2.0);
    do i = 1 to dim(a1);
      put a1[i];
    end;
    numelems = 0;
    do i = 1 to ndims(a2);
      numelems = numelems + dim(a2, i);
    end;
    sum = 0;
    do i = lbound(a2, 1) to hbound(a2, 1);
      do j = lbound(a2, 2) to hbound(a2, 2);
        do k = lbound(a2, 3) to hbound(a2, 3);
          sum = sum + a2[i,j,k];
        end;
      end;
    end;
    put sum=;
  end;
enddata;
```

```
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
red
yellow
green
blue
sum=48
```

関連項目

関数:

- “DIM 関数” (518 ページ)
- “LBOUND 関数” (801 ページ)
- “NDIMS 関数” (868 ページ)

HMS 関数

SAS 時間値(時、分および秒)を返します。

カテゴリ:	日付と時間
返されるデータの種類:	DOUBLE

構文

HMS(*hour*, *minute*, *second*)

必須引数

hour
数値式。

データの種類	DOUBLE
参照項目:	“DS2 式” (*SAS DS2 プログラマガイド*)

minute
数値式。

データの種類	DOUBLE
参照項目:	“DS2 式” (*SAS DS2 プログラマガイド*)

second
数値式。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

HMS 関数は、SAS 時間値を表す数値を返します。SAS 時間値は、現在日の午前 0 時からの秒数を表す数値です。

例

次のプログラムは、HMS 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double a;
  dcl varchar(10) b;
  method run();
    a=hms(12,45,10);
    b=put(a, time.);
    put a;
    put b;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
45910
12:45:10
```

関連項目

概念:

- “DS2 の日付と時間” (*SAS DS2 プログラマガイド*)

関数:

- “DHMS 関数” (511 ページ)

- “HOUR 関数” (699 ページ)
- “MINUTE 関数” (854 ページ)
- “SECOND 関数” (1126 ページ)

HOLIDAY 関数

指定した年の特定の祝日の SAS 日付値を返します。

カテゴリ: 日付と時間

返されるデータの種類: DOUBLE

構文

HOLIDAY('*holiday*', *year*)

必須引数

'*holiday*'
次のいずれかの値を指定する文字定数、変数または式です。BOXING、CANADA、CANADAOBSERVED、CHRISTMAS、COLUMBUS、EASTER、FATHERS、HALLOWEEN、LABOR、MLK、MEMORIAL、MOTHERS、NEWYEAR、THANKSGIVING、THANKSGIVINGCANADA、USINDEPENDENCE、USPRESIDENTS、VALENTINES、VETERANS、VETERANSUSG、VETERANSUSPS、VICTORIA。*holiday* の値には、大文字または小文字を使用できます。詳細については、*SAS DS2 言語リファレンス*を参照してください。

次の表に記載されているいずれかの値を指定する文字定数、変数または式です。

holiday の値には、大文字または小文字を使用できます。

表 7.3 祝日の値とその説明

祝日の値	説明	祝日の日付
BOXING	ボクシングデー	12 月 26 日
CANADA	カナダ独立記念日	7 月 1 日
CANADAOBSERVED	カナダ独立記念日振替休日	7 月 1 日または 7 月 2 (7/1 が日曜日の場合)
CHRISTMAS	クリスマス	12 月 25 日
COLUMBUS	コロンブス記念日	10 月の第 2 月曜日

祝日の値	説明	祝日の日付
EASTER	復活の主日	毎年変わる
FATHERS	父の日	6 月の第 3 日曜日
HALLOWEEN	ハロウィーン	10 月 31 日
LABOR	労働祭	9 月の第 1 月曜日
MLK	マーティンルーサーキング牧師の誕生日	1986 以降の 1 月の第 3 月曜日
MEMORIAL	メモリアルデー	5 月の最終月曜日(1971 以降)
MOTHERS	母の日	5 月の第 2 日曜日
NEWYEAR	元日	1 月 1 日
THANKSGIVING	U.S. 感謝祭	11 月の第 4 木曜日
THANKSGIVINGCANADA	カナダ感謝祭	10 月の第 2 月曜日
USINDEPENDENCE	U.S. 独立記念日	7 月 4 日
USPRESIDENTS	大統領記念日の祝日	2 月の第 3 月曜日 (1971 以降)
VALENTINES	バレンタインデー	2 月 14 日
VETERANS	退役軍人の日	11 月 11 日
VETERANSUSG	退役軍人の日(米国政府の祝日)	月曜日から金曜日までにスケジュールされる米国政府の祝日
VETERANSUSPS	退役軍人の日(米国郵政公社の祝日)	月曜日から日曜日までにスケジュールされる米国政府の祝日(米国郵政公社)
VICTORIA	ビクトリアデー	5 月 24 日以前の直近の月曜日

データの種類　CHAR

year
4 桁の年を指定する数値定数、変数または式です。2 桁の年を使用する場合、YEARCUTOFF=システムオプションを指定する必要があります。

データの種類 DOUBLE

詳細

HOLIDAY 関数は、指定した年に発生する特定の祝日の日付を計算します。米国およびカナダの特定の一般的な祝日のみがこの関数で使用できるように定義されています。(有効な祝日のリストについては、*"祝日の値とその説明" (SAS 関数と CALL ルーチン: リファレンス)*を参照)。

多くの祝日の定義は時代とともに変わります。米国では、1971 年 2 月 11 日に出された大統領命令 11582 で、多くの米国連邦休日の遵守を定着させました。

多くの休日はそれらが設立された日付が確立していますが、現在の祝日の定義は過去および将来に無限に拡張されます。一貫した定義がこれまでにされていない休日もあります。

HOLIDAY 関数は、SAS 日付値を返します。SAS 日付値をカレンダー日付に変換するには、DATE9.形式などの有効な SAS 日付形式を使用します。

比較

HOLIDAY 関数と NWKDOM 関数が同じ結果を返す場合があります。たとえば、ステートメント holiday('thanksgiving', 2012);は、nwkdom(4, 5, 11, 2012);.と同じ値を返します。

また、HOLIDAY 関数と MDY 関数が同じ結果を返す場合もあります。たとえば、ステートメント holiday('christmas', 2012);は、mdy(12, 25, 2012);.と同じ値を返します。

例

次のプログラムは、HOLIDAY 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
/*** Thanksgiving ***/

proc ds2;
data _null_;
  dcl double thanks having format date9.;
  method init();
    thanks = holiday('thanksgiving', 2019);
    put thanks;
```

```
  end;
enddata;
run;

/*** Boxing ***/
data _null_;
  method run();
    dcl double boxing having format date9.;
    boxing = holiday('boxing', 2019);
    put boxing;
  end;
enddata;
run;

/*** Easter ***/
data _null_;
  method run();
    dcl double easter having format date9.;
    easter = holiday('easter', 2019);
    put easter;
  end;
enddata;
run;

/*** Canada Day ***/
data _null_;
  method run();
    dcl double canada having format date9.;
    canada = holiday('canada', 2019);
    put canada;
  end;
enddata;
run;

/*** Father's Day ***/
data _null_;
  method run();
   dcl double fathers having format date9.;
    fathers = holiday('fathers', 2019);
    put fathers;
  end;
enddata;
run;

/*** Valentine's Day ***/
data _null_;
  method run();
    dcl double valentines having format date9.;
    valentines = holiday('valentines', 2019);
    put valentines;
  end;
enddata;
run;

/*** Victoria Day ***/
data _null_;
```

```
    method run();
      dcl double victoria having format date9.;
      victoria = holiday('victoria', 2019);
      put victoria;
    end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
28NOV2019

26DEC2019

21APR2019

01JUL2019

16JUN2019

14FEB2019

20MAY2019
```

関連項目

関数:

- “MDY 関数” (847 ページ)
- “NWKDOM 関数” (911 ページ)

HOUR 関数

SAS 時間値または SAS 日時値の時間を返します。

カテゴリ:	日付と時間
返されるデータの種類:	DOUBLE

構文

HOUR(*time* | *datetime*)

必須引数

time
SAS 時間値を表す有効な式を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

datetime
SAS 日時値を表す有効な式を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

HOUR 関数は、SAS 時間値または SAS 日時値の時間を表す数値を返します。0-23 の数値が表示されます。HOUR は常に正の数を返します。

例

次のプログラムは、HOUR 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double a;
  method run();
    a=hour(time());
    put a=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
14
```

関連項目

- “DS2 の日付と時間” (*SAS DS2 プログラマガイド*)

関数:

- “MINUTE 関数” (854 ページ)
- “SECOND 関数” (1126 ページ)

IBESSEL 関数

変形ベッセル関数の値を返します。

カテゴリ: 数学

返されるデータの種類: DOUBLE

構文

IBESSEL(*nu*, *x*, *kode*)

必須引数

nu
数値の定数、変数または式を指定します。

範囲 *nu* ≥ 0

データの種類 DOUBLE

x
数値の定数、変数または式を指定します。

範囲 *x* ≥ 0

データの種類 DOUBLE

kode
負でない整数を指定する数値定数、変数または式です。

データの種類 DOUBLE

詳細

IBESSEL 関数は、*x* で評価される順序 *nu* の変形ベッセル関数(Abramowitz, Stegun 1964; Amos, Daniel, Weston 1977)の値を返します。*kode* が 0 の場合、ベッセル関数が返されます。それ以外の場合、次の関数の値が返されます。

$$\varepsilon^{-x} I_{nu}(x)$$

例

次のプログラムは、IBESSEL 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double x;
  method run();
    x=ibessel(2, 2, 0);
    put x=;
    x=ibessel(2, 2, 1);
    put x=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
x=0.68894844769873
x=0.09323903330473
```

関連項目

関数:

- “JBESSEL 関数” (788 ページ)

IFN_164 関数

式の true、false、欠損に基づいて数値を返します。

カテゴリ: 数学
CAS

注: 2025.12 以降、この関数は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINIT が機能するようにするには、BIGINTPROCESSING=YES を設定します。PROC DS2 ステートメントで BIGINTPROCESSING=YES が設定され、いずれかの引数が BIGINT の場合、すべての引数が BIGINT に変換され、結果は BIGINT になります。

IFN_I64 は BIGINT 引数をサポートし、BIGINT 結果を返します。

構文

IFN_164(*logical-expression*, *value-returned-when-true*, *value-returned-when-false* [, *value-returned-when-missing*])

必須引数

logical-expression
数値の定数、変数または式を指定します。

value-returned-when-true
logical-expression の値が true の場合に返される数値定数、変数または式を指定します。

value-returned-when-false
logical-expression の値が false の場合に返される数値定数、変数または式を指定します。

オプション引数

value-returned-when-missing
logical-expression の値が欠損値の場合に返される数値定数、変数または式を指定します。

詳細

IFN_164 関数は、論理式の値に基づいて複数の値から選択できる条件ロジックを使用します。

IFN_164 は、第 1 引数 *logical-expression* を評価します。*logical-expression* が true の場合(ゼロや欠損値ではない場合)、IFN_164 は第 2 引数の値を返します。*logical-expression* が欠損値で、第 4 引数がある場合、IFN_164 は第 4 引数の値を返します。それ以外の場合、 *logical-expression* が false であれば、IFN_164 は第 3 引数の値を返します。

IFN_164 関数、IF/THEN/ELSE 構文または WHERE ステートメントで同じ結果を生成できます。(例を参照)。ただし、IFN_164 関数は、IF/THEN/ELSE 構文や WHERE ステートメントを使用しづらいまたは使用できない場合に DATA ステップの式で役立ちます。

比較

IFN_164 関数は IFC 関数と類似していますが、IFC が文字値を返すのに対して IFN_164 は数値を返す点が異なります。

例

例 1: IFN 関数を使用した手数料の計算

次の例では、IFN_164 は式 TotalSales > 10000 を評価します。売り上げ合計が $10,000 を超える場合、売り上げ手数料は売り上げ合計の 5%になります。売り上げ合計が$10,000 未満の場合、売り上げ手数料は売り上げ合計の 2%になります。

```
data _null_;
  input TotalSales;
  commission=ifn_164(TotalSales > 10000, TotalSales*.05, TotalSales*.02);
  put commission=;
  datalines;
25000
10000
500
10300
;
```

SAS は次の出力をログに書き込みます。

```
commitment=1250
commitment=200
commitment=10
commitment=515
```

例 2: IF/THEN/ELSE 構文を使用した手数料の計算

次の例では、IF/THEN/ELSE 構文で式 TotalSales > 10000 を評価します。売り上げ合計が$10,000 を超える場合、売り上げ手数料は売り上げ合計の 5%になります。売り上げ合計が$10,000 未満の場合、売り上げ手数料は売り上げ合計の 2%になります。

```
data _null_;
  input TotalSales;
  if TotalSales > 10000 then commission = .05 * TotalSales;
    else commission = .02 * TotalSales;
  put commission=;
  datalines;
25000
10000
500
10300
;
```

SAS は次の出力をログに書き込みます。

```
commitment=1250
commitment=200
commitment=10
commitment=515
```

例 3: WHERE ステートメントを使用した手数料の計算

次の例では、WHERE ステートメントで式 TotalSales > 10000 を評価します。売り上げ合計が$10,000 を超える場合、売り上げ手数料は売り上げ合計の 5%になります。売

り上げ合計が$10,000 未満の場合、売り上げ手数料は売り上げ合計の 2%になります。出力には、売り上げ合計が$10,000 を超えている販売員のみが表示されます。

```
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;
```

アウトプット 7.10　WHERE ステートメントからの出力

Commission for Total Sales > 1000

Obs	SalesPerson	TotalSales	commission
1	Michaels	25000	1250
2	Gupta	10300	515

関連項目

関数:

- *"IFC 関数" (SAS 関数と CALL ルーチン: リファレンス)*

INDEX 関数

文字式から文字列を検索し、最初に検索された文字列の最初の文字の位置を返します。

カテゴリ:	文字
返されるデータの種類:	INTEGER

構文

INDEX(*target-expression*, *search-expression*)

必須引数

target-expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

search-expression
文字列を評価するか文字列に強制変換できて、*target-expression* での検索に使用される有効な式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

ヒント 文字のリテラル文字列は単一引用符で囲みます。

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

詳細

INDEX 関数は、*target-expression* を左から右へ、*search-expression* で指定される文字列の最初の出現箇所を検索し、*target-expression* 内での文字列の最初の文字の位置を返します。*target-expression* 内に文字列が見つからない場合、INDEX は値 0 を返します。文字列が複数回出現する場合、INDEX は、最初の出現箇所の位置のみを返します。

注: *target-expression* または *search-expression* のいずれかの長さが 0 の場合、INDEX 関数は値 0 を返します。宣言されていない VARCHAR 変数の長さは 0 であることにも注意してください。

比較

VERIFY 関数は、*search-expression* 式を含まない *target-expression* の最初の文字の位置を返し、INDEX 関数は、*target-expression* に存在する *search-expression* の最初の出現位置を返します。

例

次のプログラムは、INDEX ステートメントを示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double c;
  dcl varchar(20) a b;
  method run();
    a='ABC.DEF (X=Y)';
    b='X=Y';
    c=index(a,b);
    put c=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
c=10
```

関連項目

関数:

- “INDEXC 関数” (707 ページ)
- “INDEXW 関数” (709 ページ)
- “VERIFY 関数” (1233 ページ)

INDEXC 関数

文字式から指定文字を検索し、そのいずれかの文字の最初の出現位置を返します。

カテゴリ:	文字
返されるデータの種類:	DOUBLE

構文

INDEXC(*target-expression*, *search-expression*[, ...*search-expression*])

必須引数

target-expression
文字列を評価するか、文字列に強制変換できる検索先の有効な式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

search-expression
target-expression で検索する文字を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

ヒント 文字のリテラル文字列は単一引用符で囲みます。

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

INDEXC 関数は、検索式に存在する *target-expression* のいずれかの文字の最初の出現箇所を検索し、その文字の *target-expression* 内の位置を返します。検索は左から右に実行されます。*target-expression* で検索式内の文字が見つからない場合、INDEXC は値 0 を返します。

比較

INDEXC 関数が、検索式に含まれる個々の文字が最初に出現する箇所を検索するのに対し、INDEX 関数は、検索式がパターンとして最初に出現する箇所を検索します。

例

次のプログラムは、INDEXC 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double x y;
  dcl varchar(20) a b c;
  method run();
    a='ABC.DEP (X2=Y1)';
    b='()';
    c='=.';
    x=indexc(a,'0123',';()=.');
    put x=;
```

```
    x=indexc(a,b);
    put x=;
    x=indexc(a,b,c);
    put x=;
    c='have a good day';
    x=indexc(c,'pleasant','very');
    put x=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
x=4
x=9
x=4
x=2
```

関連項目

関数:

- “INDEX 関数” (705 ページ)
- “INDEXW 関数” (709 ページ)

INDEXW 関数

文字式から単語として指定した文字列を検索し、単語の最初の文字の位置を返します。

カテゴリ:	文字
返されるデータの種類:	DOUBLE

構文

INDEXW(*target-expression*, *search-expression* [, *delimiter*])

必須引数

target-expression
文字列を評価するか、文字列に強制変換できる検索先の有効な式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

search-expression
文字列を評価するか文字列に強制変換できて、*target-expression* で検索される有効な式を指定します。SAS は、*search-expression* から先頭および末尾の区切り文字を削除します。

データの種類　CHAR、NCHAR、NVARCHAR、VARCHAR

参照項目:　"DS2 式" (*SAS DS2 プログラマガイド*)

オプション引数

delimiter
INDEXW で文字列の単語区切り文字として使用する文字式を指定します。デフォルトの区切り文字は空白文字です。

データの種類　CHAR、NCHAR、NVARCHAR、VARCHAR

ヒント　空白文字が区切り文字の場合は、*delimiter* の最後の文字にならないように並べ替えてください。*delimiter* は末尾の空白から取り除かれるため、末尾の空白は無視されます。

参照項目:　"DS2 式" (*SAS DS2 プログラマガイド*)

詳細

INDEXW 関数は、*target-expression* を左から右へ、*search-expression* の最初の出現箇所を検索し、*target-expression* 内での部分文字列の最初の文字の位置を返します。*target-expression* 内に部分文字列が見つからない場合、INDEXW は値 0 を返します。文字列が複数回出現する場合、INDEXW は、最初の出現箇所の位置のみを返します。

部分文字列パターンは、単語の境界で開始および終了する必要があります。INDEXW の場合、単語の境界は区切り文字、*target-expression* の最初、および *target-expression* の最後になります。

ヒント　*search-expression* に空白が含まれている場合や長さが 0 の場合、INDEXW は次のように動作します。

- *target-expression* と *search-expression* の両方に空白のみが含まれている場合や長さが 0 の場合、INDEXW は値 1 を返します。
- *search-expression* に空白のみが含まれている場合や長さが 0 の場合、*target-expression* に文字または数値のデータが含まれていると、INDEXW は値 0 を返します。

比較

INDEXW 関数は、単語の文字列を検索しますが、INDEX 関数は区切られた単語または他の単語の一部としてパターンを検索します。INDEXC は、excerpt 内に含まれる文字を検索します。

例

次のプログラムは、INDEXW 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double c b;
  dcl varchar(20) a word x xyz;
  method run();
    a='The power to know.';
    word='power';
    c=indexw(a,word);
    put c=;
    a='The power to know.';
    b=indexw(a,'know');
    put b=;
    a='The power to know.';
    b=indexw(a,'know','. ');
    put b=;
    a='abc,def@ xyz';
    b=indexw(a,',', '@');
    put b=;
    a='abc,def@ xyz';
    b=indexw(a,'def', '@,');
    put b=;
    x='abc,def@ xyz';
    xyz=indexw(x, ' xyz', '@');
    put xyz=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
c=5
b=0
b=14
b=0
b=5
xyz=9
```

関連項目

関数:

- “INDEX 関数” (705 ページ)
- “INDEXC 関数” (707 ページ)

INPUTC 関数

実行時に文字の入力形式を指定できるようにします。

カテゴリ: 特殊

返されるデータの種類: CHAR、NCHAR、NVARCHAR、VARCHAR

構文

INPUTC(*source*, *informat*[, *w*])

必須引数

source
入力形式を適用する文字定数、変数または式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

informat
source に適用する文字の入力形式が含まれる文字定数、変数または式です。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

オプション引数

w
入力形式に適用する数値幅に評価される有効な式を指定します。

操作 ここで指定した幅は、入力形式での幅の指定より優先されます。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

INPUTC 関数が長さの割り当てられていない変数に値を返した場合、デフォルトでは変数の長さは最初の引数の長さによって決定されます。

比較

INPUTN 関数は、実行時に数値の入力形式を指定できるようにします。

例

次のプログラムは、文字の入力形式を指定する方法を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl char(10) type type2;
  method init();
    type=inputc('positive', '$upcase15.');
    type2=inputc('positive', '$upcase15.', 3);
    put type=;
    put type2=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
type=POSITIVE
type2=POS
```

関連項目

関数:

- “INPUTN 関数” (714 ページ)
- “PUT 関数” (1038 ページ)

INPUTN 関数

実行時に数値の入力形式を指定できるようにします。

カテゴリ: 特殊

返されるデータの種類: DOUBLE

構文

INPUTN(*source*, *informat*[, *w*[, *d*]])

必須引数

source
入力形式を適用する文字定数、変数または式を指定します。

データの種類 CHAR

informat
source に適用する数値の入力形式が含まれる文字定数、変数または式です。

データの種類 CHAR

オプション引数

w
入力形式に適用する幅を指定する数値定数、変数または式です。

操作 ここで指定した幅は、入力形式での幅の指定より優先されます。

データの種類 DOUBLE

d
使用する小数点以下の桁数を指定する数値定数、変数または式です。

操作 ここで指定した桁数は、入力形式での小数点以下の桁数の指定より優先されます。

データの種類 DOUBLE

比較

INPUTC 関数は、実行時に文字の入力形式を指定できるようにします。コンパイル時に入力形式を指定するため、PUT 関数を使用した方が高速になります。

例

次のプログラムは、数値の入力形式を指定する方法を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  method init();
    declare double salary;
    salary = inputn('20,000.00', 'comma10.2');
    put salary=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
20000
```

関連項目

関数:

- “INPUTC 関数” (712 ページ)
- “PUT 関数” (1038 ページ)

INPUTN_164 関数

実行時に数値の入力形式を指定します。

カテゴリ:	特殊
返されるデータの種類:	DOUBLE
注:	2025.12 以降、この関数は DS2 言語で実行する場合に BIGINT データ型をサポートします。BIGINIT が機能するようにするには、BIGINTPROCESSING=YES を設定し

ます。PROC DS2 ステートメントで BIGINTPROCESSING=YES が設定され、いずれかの引数が BIGINT の場合、すべての引数が BIGINT に変換され、結果は BIGINT になります。

INPUTN_I64 は、BIGINT の結果を返します。

構文

INPUTN_164(*source*, *informat*[, *w*[, *d*]])

必須引数

source
入力形式を適用する文字定数、変数または式を指定します。

データの種類 CHAR

informat
source に適用する数値の入力形式が含まれる文字定数、変数または式です。

データの種類 CHAR

オプション引数

w
入力形式に適用する幅を指定する数値定数、変数または式です。

操作 ここで指定した幅は、入力形式での幅の指定より優先されます。

データの種類 DOUBLE

d
使用する小数点以下の桁数を指定する数値定数、変数または式です。

操作 ここで指定した桁数は、入力形式での小数点以下の桁数の指定より優先されます。

データの種類 DOUBLE

比較

INPUTN_I64 関数は、実行時に文字の入力形式を指定できるようにします。コンパイル時に入力形式を指定するため、PUT 関数を使用した方が高速になります。

例

次のプログラムは、数値の入力形式を指定する方法を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  method init();
    declare double salary;
    salary = INPUTN_164('20,000.00', 'comma10.2');
    put salary=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
20000
```

関連項目

関数:

- “INPUTC 関数” (712 ページ)
- “PUT 関数” (1038 ページ)

INT 関数

予期しない浮動小数点の結果を避けるためにファジー処理された整数を返します。

カテゴリ: 切り捨て

返されるデータの種類: DOUBLE

構文

INT(*expression*)

必須引数

expression
数値に評価される任意の式を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

INT 関数は、引数の整数部(小数点以下は切り捨て)を返します。引数の値が 1E-12 内の整数である場合、関数はその整数を返します。*expression* が正の値の場合、INT 関数の結果は FLOOR 関数の結果と同じです。*expression* が負の値の場合、INT 関数の結果は CEIL 関数の結果と同じです。

比較

INTZ 関数とは異なり、INT 関数は結果をファジー処理します。引数が整数の 1E-12 内にある場合、INT 関数はその整数に等しくなるように結果をファジー処理します。INTZ 関数は結果をファジー処理しません。そのため、INTZ 関数では予期しない結果になる可能性があります。

例

次のプログラムは、INT 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double var1 a b c d;
  method run();
    var1=2.1;
    a=int(var1);
    put a;
    b=int(-2.4);
    put b;
    c=int(1+1.e-11);
    put c;
    d=int(-1.6);
    put d;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
2
-2
1
-1
```

関連項目

関数:

- “CEIL 関数” (410 ページ)
- “FLOOR 関数” (653 ページ)
- “INTZ 関数” (781 ページ)
- “MOD 関数” (859 ページ)
- “MODZ 関数” (861 ページ)
- “ROUND 関数” (1088 ページ)

INTCINDEX 関数

サイクルインデックスを返します。この関数には、日付、時間またはタイムスタンプの間隔と値を指定します。

カテゴリ:	日付と時間
制限事項:	DS2 は、カスタムの日付または時間間隔をサポートしていません。
返されるデータの種類:	DOUBLE

構文

INTCINDEX({*interval*[*multiple*][.*shift-index*]}, *date-time-value*)

必須引数

interval
文字列を評価するか、文字列に強制変換でき、WEEK、MONTH、QTR などの間隔名を含む文字定数、変数、または式を指定します。

データの種類	CHAR、NCHAR、NVARCHAR、VARCHAR
注	*interval* の可能な値は、“日付間隔と時間間隔について” (*SAS 出力形式と入力形式: リファレンス*)にリストされています。
ヒント	*Interval* は、大文字または小文字で表示できます。

例 YEAR は、年ベースの間隔を指定します。

date-time-value
指定した間隔の期間を表す日付、時間またはタイムスタンプの値を指定します。

データの種類 DOUBLE

オプション引数

multiple
乗数を指定します(省略可能)。基本タイプの間隔の期間に対する倍数と同等の間隔を設定します。

データの種類 DOUBLE

参照項目: 詳細については、"乗数とシフト間隔を使用した日時の増分" (*SAS 関数と CALL ルーチン: リファレンス*)を参照してください。

例 YEAR2 は、2 年、つまり 2 年間続く間隔タイプを指定します。

shift-index
シフトインデックスを指定します(省略可能)。間隔の開始時点を指定したサブ期間にシフトします。

制限事項 シフトインデックスは、間隔全体のサブ期間の数以下にする必要があります。たとえば、2 年間隔では 25 番目の月は存在しないため、YEAR2.24 は使用できますが YEAR2.25 はエラーになります。

デフォルトのシフト期間が間隔の種類と同じ場合、複数期間の間隔のみを任意のシフトインデックスでシフトできます。たとえば、MONTH の種類の間隔はデフォルトでは MONTH のサブ期間でシフトされるため、シフトインデックスで月間隔をシフトできません。ただし、各 MONTH2 間隔には 2 つの MONTH 間隔が含まれるため、シフトインデックスで 2 か月間隔をシフトできます。たとえば、間隔名 MONTH2.2 では、偶数月の 1 日目に開始する 2 か月の期間が指定されます。

データの種類 DOUBLE

参照項目: 詳細については、"乗数とシフト間隔を使用した日時の増分" (*SAS 関数と CALL ルーチン: リファレンス*)を参照してください。

例 YEAR.3 は、暦年の 3 月 1 日に開始し、翌年の 2 月に終了するようにシフトされた年次期間を指定します。

詳細

基本

INTCINDEX 関数は、季節サイクルのインデックスを返します。この関数には、間隔と DATE、TIME または TIMESTAMP の値を指定します。たとえば、間隔が MONTH の場合、データ内の各オブザベーションは特定の月に対応します。月単位のデータは、1 年間で周期的とみなされます。1 年には 12 か月あるため、季節サイクル(年)内の間隔(月)数は 12 です。WEEK は、DAY と等しい間隔の季節サイクルです。2013 年 9 月 1 日はその年の 35 番目の週で 6 番目の曜日なので、この例では 36 の値が返されます。

```
sasdate1=to_double(date'2013-09-01');
cycle_index1 = intcindex('day', sasdate1);
```

日付間隔と時間間隔の操作の詳細については、“日付間隔と時間間隔” (*SAS 関数と CALL ルーチン: リファレンス*)を参照してください。

INTCINDEX 関数は、小売業界によるカレンダーの間隔でも使用できます。これらの間隔は ISO 8601 に準拠します。これらの間隔のリストについては、*SAS プログラマガイド: エッセンシャル*の"Retail Calendar Intervals: ISO 8601 Compliant"を参照してください。

間隔

間隔は、基本的なものから複雑なものまであります。基本間隔は、SAS が経過時間内にカウントできる測定単位(DAY、MONTH、HOUR など)です。より複雑な間隔を指定するには、基本間隔名に乗数とシフトインデックスを組み合わせて使用できます。

間隔構文は次のとおりです。

interval[*multiple*][.*shift-index*]

詳細については、“必須引数” (719 ページ)を参照してください。

比較

INTCINDEX 関数はサイクルインデックスを返しますが、INTINDEX 関数は季節インデックスを返します。

この例では、INTCINDEX 関数は年間通算週を返します。

```
sasdate1=to_double(date'04apr2013');
cycle_index = intcindex('day', sasdate1);
```

この例では、INTINDEX 関数は曜日を返します。

```
sasdate1=to_double(date'04apr2013');
index = intindex('day','04APR2013'd);
```

この例では、INTCINDEX 関数は時刻を返します。

```
sasts=to_double(timestamp '2012-09-01 00:00:00');
a= intcindex('minute', sasts);
```

この例では、INTINDEX 関数は分を返します。

```
sasts=to_double(timestamp'2012-09-01 00:00:00');
a= intindex('minute', sasts);
```

例 intseas(intcycle('interval'));では、INTSEAS 関数は intcindex('interval', date);によって返される最大数を返します。

例

次のプログラムは、INTCINDEX 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double sasdate1 cycle_index1;
  method run();
    sasdate1=as_double(date'2013-09-01');
    cycle_index1 = intcindex('day', sasdate1);
    put cycle_index1;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
36
```

```
proc ds2;
data test (overwrite=yes);
  dcl double sascydate cycle_index1;
  method run();
    sascydate=as_double(timestamp '2013-05-23 05:03:01');
    cycle_index1 = intcindex('dtqtr', sascydate);
    put cycle_index1;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
1
```

```
proc ds2;
data test (overwrite=yes);
  dcl double sascydate cycle_index1;
  method run();
    sascydate=as_double(date'2013-12-13');
    cycle_index1 = intcindex('tenday', sascydate);
```

```
    put cycle_index1;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
1
```

```
proc ds2;
data test (overwrite=yes);
  dcl double sascydate cycle_index1;
  method run();
    sascydate=as_double(time '23:13:02');
    cycle_index1 = intcindex('minute', sascydate);
    put cycle_index1;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
24
```

```
proc ds2;
data test (overwrite=yes);
  dcl double sascydate cycle_index1;
  dcl char(10) var1;
  method run();
    sascydate=as_double(timestamp '2013-05-05 10:54:03');
    var1='semimonth';
    cycle_index1 = intcindex(var1, sascydate);
    put cycle_index1;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
1
```

関連項目

関数:

- “INTCYCLE 関数” (732 ページ)
- “INTINDEX 関数” (745 ページ)
- “INTSEAS 関数” (767 ページ)

INTCK 関数

DOUBLE としてエンコードされた 2 つの SAS 日付、時間、またはタイムスタンプ値の間にある特定の種類の間隔境界の数を返します。

カテゴリ: 日付と時間

制限事項: DS2 は、カスタムの日付または時間間隔をサポートしていません。

返されるデータの種類: DOUBLE

構文

INTCK({*interval*[*multiple*][.*shift-index*]}, *start-date*, *end-date*[, '*method*'])

INTCK(*start-date*, *end-date*[, '*method*'])

必須引数

interval
文字列を評価するか、文字列に強制変換でき、WEEK、MONTH、QTR などの間隔名を含む文字定数、変数、または式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

注 *interval* の可能な値は、"日付間隔と時間間隔について" (*SAS 出力形式と入力形式: リファレンス*)にリストされています。

ヒント *Interval* は、大文字または小文字で表示できます。

例 YEAR は、年ベースの間隔を指定します。

start-date
SAS を開始する日付、時間、またはタイムスタンプ値を表す式を指定します。

データの種類 DOUBLE

end-date
SAS を終了する日付、時間、またはタイムスタンプ値を表す式を指定します。

データの種類 DOUBLE

オプション引数

multiple
乗数を指定します(省略可能)。基本タイプの間隔の期間に対する倍数と同等の間隔を設定します。

データの種類 DOUBLE

参照項目: 詳細については、"乗数とシフト間隔を使用した日時の増分" (*SAS 関数と CALL ルーチン: リファレンス*)を参照してください。

例 YEAR2 は、2 年、つまり 2 年間続く間隔タイプを指定します。

shift-index
シフトインデックスを指定します(省略可能)。間隔の開始時点を指定したサブ期間にシフトします。

制限事項 シフトインデックスは、間隔全体のサブ期間の数以下にする必要があります。たとえば、2 年間隔では 25 番目の月は存在しないため、YEAR2.24 は使用できますが YEAR2.25 はエラーになります。

デフォルトのシフト期間が間隔の種類と同じ場合、複数期間の間隔のみを任意のシフトインデックスでシフトできます。たとえば、MONTH の種類の間隔はデフォルトでは MONTH のサブ期間でシフトされるため、シフトインデックスで月間隔をシフトできません。ただし、各 MONTH2 間隔には 2 つの MONTH 間隔が含まれるため、シフトインデックスで 2 か月間隔をシフトできます。たとえば、間隔名 MONTH2.2 では、偶数月の 1 日目に開始する 2 か月の期間が指定されます。

データの種類 DOUBLE

参照項目: 詳細については、"乗数とシフト間隔を使用した日時の増分" (*SAS 関数と CALL ルーチン: リファレンス*)を参照してください。

例 YEAR.3 は、暦年の 3 月 1 日に開始し、翌年の 2 月に終了するようにシフトされた年次期間を指定します。

'*method*'
DISCRETE メソッドと CONTINUOUS メソッドのどちらを使用して間隔を数えるかを指定します。*method* は引用符で囲みます。*method* は、CONTINUOUS と DISCRETE (間隔境界をカウント)のいずれかの値になります。

DISCRETE メソッドと CONTINUOUS メソッドのどちらを使用して間隔を数えるかを指定します。

method は引用符で囲む必要があります。*Method* には次のいずれかの値を指定できます。

CONTINUOUS
連続した期間を測定単位に指定します。間隔は開始日に基づいてシフトされます。

たとえば、2013 年 1 月 15 日から 2013 年 2 月 15 日までの月単位の距離は 1 か月です。

別名 C または CONT

DISCRETE
連続しない期間を測定の単位に指定します。DISCRETE メソッドは、間隔の境界(月末など)を数えます。

デフォルトの DISCRETE メソッドは、時系列オブザベーションをビンに分類して処理する場合に役立ちます。たとえば、日単位のデータを月単位の時系列として処理するために月単位のデータに累積できます。

DISCRETE メソッドの場合、2000 年 1 月 31 日から 2000 年 2 月 1 日までの間隔を月単位で示すと、1 か月になります。

別名　D または DISC

デフォルト　DISCRETE

データの種類　CHAR、NCHAR、NVARCHAR、VARCHAR

詳細

間隔

間隔は、基本的なものから複雑なものまであります。基本間隔は、SAS が経過時間内にカウントできる測定単位(DAY、MONTH、HOUR など)です。より複雑な間隔を指定するには、基本間隔名に乗数とシフトインデックスを組み合わせて使用できます。

間隔構文は次のとおりです。

interval[*multiple*][.*shift-index*]

詳細については、“必須引数” (724 ページ)を参照してください。

カレンダー間隔の計算

個々の時間間隔内に含まれる値は、すべて同等とみなします。つまり、月間隔を指定した場合、2013 年 1 月 1 日と 2013 年 1 月 15 日は同等です。これら両方の日付は、2013 年 1 月 1 日に開始して 2013 年 1 月 31 日に終了する間隔を表します。この間隔を示すには、間隔の開始日(2013 年 1 月 1 日)または間隔の終了日(2013 年 1 月 31 日)を使用できます。これらの日付によって、月間隔内のすべての日付が表されます。

次の例では、*start-date* (2013 年 1 月 14 日)は、2013 年の第 1 四半期と同等です。

```
sasdate1=to_double(date'2013-01-14');
sasdate2 = to_double(date'2013-09-02');
qtr=intck('qtr', sasdate1, sasdate2);
```

end-date (2013 年 9 月 2 日)は、2013 年の第 3 四半期と同等です。間隔数(*start-date* と *end-date* の間に間隔の開始点が含まれる回数)は 2 です。

デフォルトの DISCRETE メソッドを使用する INTCK 関数は、1 番目の日付と 2 番目の日付の間に次の間隔の開始点が含まれる回数を数えます。2 つの日付間に含まれる間隔数は計算しません。

- 次の例では、2 つの日付が同じ月内にあるため、0 が返されます。

```
sasdate1=to_double(date'2013-01-01');
sasdate2 = to_double(date'2013-01-31');
month=intck(month, sasdate1, sasdate2);
put month;
```

- 次の例では、2 つの日付が 1 か月離れた異なる月にあるため、1 が返されます。

```
sasdate1=to_double(date'2013-01-31');
sasdate2 = to_double(date'2013-02-01');
month=intck(month, sasdate1, sasdate2);
put month;
```

- 次の例では、最初の日付が 2 番目の日付よりも遅い離散間隔にあるため、-1 が返されます。(1 番目の日付が 2 番目の日付よりも後で、2 つの日付が同じ間隔内に存在しない場合、INTCK は常に負の値を返します)。

```
sasdate1=to_double(date'2013-02-01');
sasdate2 = to_double(date'2013-01-31');
month=intck('month', sasdate1, sasdate2);
put month;
```

DISCRETE メソッドを使用する場合、WEEK 間隔は *start-date* と *end-date* の間に 7 日間がいくつ含まれるかではなく、2 つの日付間に存在する日曜日(週のデフォルトの開始曜日)の数で決定されます。*start-date* と *end-date* の間に 7 日間が含まれる数を数えるには、CONTINUOUS メソッドを使用します。

multiple 引数と *shift-index* 引数は両方とも任意で、デフォルトで 1 になっています。たとえば、YEAR、YEAR1、YEAR.1、YEAR1.1 は、すべて通常のカレンダー年を指定します。

日付間隔と時間間隔の操作の詳細については、“日付間隔と時間間隔” (*SAS 関数と CALL ルーチン: リファレンス*)を参照してください。

カテゴリ別の間隔

表 7.4　日付と時間の関数で使用される間隔

カテゴリ	間隔	定義	デフォルトの開始点	シフト期間	例	説明
日付	DAY	日間隔	毎日	日	DAY3	日曜日から始まる 3 日間隔
	WEEK	7 日間の週間隔	毎週日曜日	日(1=日曜日 ...7=土曜日)	WEEK.7	土曜日を週の最初の日とする毎週
	WEEKDAY <*days*W>	金土日を含む日間隔	毎日	日	WEEKDAY1W	日曜日を週末日、6 日間を就業日とする週
		同じ日としてカウントされ			WEEKDAY35W	火曜日と木曜日を週末

カテゴリ	間隔	定義	デフォルトの開始点	シフト期間	例	説明
		ます(土曜日と日曜日を週末、5 日間を就業日とする週)。*days* は、週末日を数値で識別します(1=日曜日 ... 7=土曜日)。デフォルトでは、*days*=17 です。				日とし、5 日間を就業日とする週(W で 3 日目と 5 日目を週末日に指定)
	TENDAY	10 日間隔(米国自動車産業の慣例)	毎月 1 日、11 日、21 日	10 日間	TENDAY4.2	TENDAY 単位の 2 番目を開始時点とする 40 日単位
	SEMIMONTH	半月間隔	毎月 1 日と 16 日	半月間	SEMIMONTH2.2	月の 16 日目から翌月の 15 日目までの間隔
	MONTH	月間隔	毎月 1 日	月	MONTH2.2	翌年の 2 月-3 月、4 月-5 月、6 月-7 月、8 月-9 月、10 月-11 月、12 月-1 月
	QTR	四半期(3 か月)間隔	1 月 1 日	月	QTR3.2	4 月 1 日、7 月 1 日、10 月 1 日、1 月 1 日から 3 か月間隔
			4 月 1 日			
			7 月 1 日			
			10 月 1 日			
	SEMIYEAR	半年(6 か月)間隔	1 月 1 日	月	SEMIYEAR.3	3 月-8 月と 9 月-2 月の 6 か月間隔
			7 月 1 日			
	YEAR	年間隔	1 月 1 日	月		

カテゴリ	間隔	定義	デフォルトの開始点	シフト期間	例	説明
日時	日付間隔のいずれかに DT を追加	関連する日付間隔に対応する間隔	1960 年 1 月 1 日の午前 0 時		DTMONTH	
					DTWEEKDAY	
時間	SECOND	秒間隔	1 日の始まり(午前 0 時)	秒		
	MINUTE	分間隔	1 日の始まり(午前 0 時)	分		
	HOUR	時間隔	1 日の始まり(午前 0 時)	時		

小売カレンダーの間隔

小売業界では、1 年のカレンダーを 13 週間からなる 4 つの期間に分けてデータを計算することがよくあります。期間の形式は、4-4-5、4-5-4 または 5-4-4 のいずれかに基づきます。1 番目、2 番目、3 番目の数値には、それぞれ各期間の 1 か月目、2 か月目、3 か月目の週数を指定します。詳細については、“ISO 8601 規格の基本表記と拡張表記を使用した日付と時間の処理” (*SAS 出力形式と入力形式: リファレンス*)を参照してください。

例

次のプログラムは、INTCK 関数を示しています。

2 番目の例では、経過日数は 1 日のみでも INTCK は 1 の値を返します。これは、2012 年 12 月 31 日から 2013 年 1 月 1 日までの間に YEAR 間隔の開始点が含まれるためです。ただし、3 番目の例では、経過日数は 364 日でも 0 の値が返されます。この結果が発生するのは、2013 年 1 月 1 日から 2013 年 12 月 31 日の間に YEAR 間隔の開始点が含まれないためです。

4 番目の例では、2010 年 1 月 1 日から 2013 年 1 月 1 日までの期間に半年の間隔が 6 回含まれるため、SAS は 6 の値を返します。(終了日が 2012 年 12 月 31 日の場合、SAS は 5 回の間隔を数えます)。5 番目の例では、2013 年 1 月 7 日から 2013 年 4 月 1 日の間には第 1 日を月曜日とする 2 週間の間隔が 6 回含まれるため、SAS は 6 の値を返します。6 番目の例では、SAS は値 27 を返します。この例では 2013 年 1 月 1 日から 2013 年 2 月 1 日までの間に土曜日のみを週末として数えるため、この期間には 27 日の平日が含まれることになります。

7 番目の例では、引数に変数が使用されています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double sasdate1 sasdate2 qtr;
  method run();
    sasdate1 = to_double(date'2013-01-10');
    sasdate2 = to_double(date'2013-07-01');
    qtr=intck('qtr', sasdate1, sasdate2);
    put qtr;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
2
```

```
proc ds2;
data test (overwrite=yes);
  dcl double sasdate1 sasdate2 year;
  method run();
    sasdate1 = to_double(date'2012-12-31');
    sasdate2 = to_double(date'2013-01-01');
    year=intck('year', sasdate1, sasdate2);
    put year;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
1
```

```
proc ds2;
data test (overwrite=yes);
  dcl double sasdate1 sasdate2 year;
  method run();
    sasdate1 = to_double(date'2013-01-01');
    sasdate2 = to_double(date'2013-12-31');
    year=intck('year', sasdate1, sasdate2);
    put year;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
0
```

```
proc ds2;
```

```
data test (overwrite=yes);
  dcl double sasdate1 sasdate2 semiyear;
  method run();
    sasdate1 = to_double(date'2010-01-01');
    sasdate2 = to_double(date'2013-01-01');
    semiyear=intck('semiyear', sasdate1, sasdate2);
    put semiyear;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
6
```

```
proc ds2;
data test (overwrite=yes);
  dcl double sasdate1 sasdate2 weekvar;
  method run();
    sasdate1 = to_double(date'2013-01-07');
    sasdate2 = to_double(date'2013-04-01');
    weekvar=intck('week2.2', sasdate1, sasdate2);
    put weekvar;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
6
```

```
proc ds2;
data test (overwrite=yes);
  dcl double sasdate1 sasdate2 wdvar;
  method run();
    sasdate1 = to_double(date'2013-01-01');
    sasdate2 = to_double(date'2013-02-01');
    wdvar=intck('weekday7w', sasdate1, sasdate2);
    put wdvar;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
27
```

```
proc ds2;
data test (overwrite=yes);
  dcl double sasdate1 sasdate2 newyears;
  dcl char y;
  method run();
    y='year';
```

```
      sasdate1 = to_double(date'2003-09-01');
      sasdate2 = to_double(date'2013-09-01');
      newyears=intck(y, sasdate1, sasdate2);
      put newyears;
    end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
10
```

関連項目

関数:

- “INTDT 関数” (738 ページ)
- “INTNX 関数” (755 ページ)
- “INTTS 関数” (779 ページ)

その他のリファレンス:

- “DS2 の日付と時間” (*SAS DS2 プログラマガイド*)

INTCYCLE 関数

次に高い季節サイクルでの期間(日付、時間または日時の間隔)を返します。この関数には、期間(日付、時間または日時の間隔)を指定します。

カテゴリ:	日付と時間
制限事項:	DS2 は、カスタムの日付または時間間隔をサポートしていません。
返されるデータの種類:	CHAR、NCHAR、NVARCHAR、VARCHAR

構文

INTCYCLE({*interval*[*multiple*][.*shift-index*]}[, *seasonality*])

必須引数

interval
文字列を評価するか、文字列に強制変換でき、WEEK、MONTH、QTR などの間隔名を含む文字定数、変数、または式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

注 *interval* の可能な値は、"日付間隔と時間間隔について" (*SAS 出力形式と入力形式: リファレンス*)にリストされています。

ヒント *Interval* は、大文字または小文字で表示できます。

例 YEAR は、年ベースの間隔を指定します。

オプション引数

multiple
乗数を指定します(省略可能)。基本タイプの間隔の期間に対する倍数と同等の間隔を設定します。

データの種類 DOUBLE

参照項目: 詳細については、"乗数とシフト間隔を使用した日時の増分" (*SAS 関数と CALL ルーチン: リファレンス*)を参照してください。

例 YEAR2 は、2 年、つまり 2 年間続く間隔タイプを指定します。

shift-index
シフトインデックスを指定します(省略可能)。間隔の開始時点を指定したサブ期間にシフトします。

制限事項 シフトインデックスは、間隔全体のサブ期間の数以下にする必要があります。たとえば、2 年間隔では 25 番目の月は存在しないため、YEAR2.24 は使用できますが YEAR2.25 はエラーになります。

デフォルトのシフト期間が間隔の種類と同じ場合、複数期間の間隔のみを任意のシフトインデックスでシフトできます。たとえば、MONTH の種類の間隔はデフォルトでは MONTH のサブ期間でシフトされるため、シフトインデックスで月間隔をシフトできません。ただし、各 MONTH2 間隔には 2 つの MONTH 間隔が含まれるため、シフトインデックスで 2 か月間隔をシフトできます。たとえば、間隔名 MONTH2.2 では、偶数月の 1 日目に開始する 2 か月の期間が指定されます。

データの種類 DOUBLE

参照項目: 詳細については、"乗数とシフト間隔を使用した日時の増分" (*SAS 関数と CALL ルーチン: リファレンス*)を参照してください。

例 YEAR.3 は、暦年の 3 月 1 日に開始し、翌年の 2 月に終了するようにシフトされた年次期間を指定します。

seasonality
数値を指定します。この引数では、日付と時間のサイクルをより柔軟に操作できます。1 年間の季節性を 52 週にするか 53 週にするかを指定できます。

数値を指定します。

この引数では、日付と時間のサイクルをより柔軟に操作できます。1 年間の季節性を 52 週にするか 53 週にするかを指定できます。

デフォルト　52

データの種類　DOUBLE、CHAR、NCHAR、NVARCHAR、VARCHAR

例　次の例では、*seasonality* 引数により
INTCYCLE('MONTH', 3);
関数呼び出しで値 QTR が返されます。この関数呼び出しは
INTCYCLE('MONTH');
seasonality 引数がないため、値 YEAR を返します。

詳細

基本

INTCYCLE 関数は、日付、時間または日時の間隔に応じて、季節サイクルの間隔を返します。たとえば、INTCYCLE('MONTH');では、1 月から 12 月までで 1 年サイクルを構成するため、値 YEAR を返します。INTCYCLE('DAY');では、日曜日から土曜日までで 1 週間サイクルを構成するため、値 WEEK を返します。

乗数とシフトインデックスの詳細については、“乗数とシフト間隔を使用した日時の増分” (*SAS 関数と CALL ルーチン: リファレンス*)を参照してください。間隔の計算方法については、“よく使用される時間間隔” (*SAS 関数と CALL ルーチン: リファレンス*)を参照してください。

日付間隔と時間間隔の操作の詳細については、“日付間隔と時間間隔” (*SAS 関数と CALL ルーチン: リファレンス*)を参照してください。

INTCYCLE 関数は、小売業界によるカレンダーの間隔でも使用できます。これらの間隔は ISO 8601 に準拠します。詳細については、“小売カレンダーの間隔: ISO 8601 準拠” (*SAS 関数と CALL ルーチン: リファレンス*)を参照してください。

間隔

間隔は、基本的なものから複雑なものまであります。基本間隔は、SAS が経過時間内にカウントできる測定単位(DAY、MONTH、HOUR など)です。より複雑な間隔を指定するには、基本間隔名に乗数とシフトインデックスを組み合わせて使用できます。

間隔構文は次のとおりです。

interval[*multiple*][.*shift-index*]

詳細については、“必須引数” (732 ページ)を参照してください。

季節性

季節性は、年内の異なる間隔での周期的変動を測定する時系列の概念です。季節性の指定では、季節が最も一般的な変動ソースです。たとえば、家庭暖房用の灯油は、

一般的に他の季節よりも冬の売り上げの方が高くなります。多くの場合、日単位の時系列では曜日による定期的な変動(週末のレジャーで 支出が増えるなど)が発生します。INTCYCLE 関数は季節性の概念を使用して、次に高い季節サイクルでの期間(日付、時間または日時の間隔)を返します。この関数には、期間(日付、時間または日時の間隔)を指定します。季節性と PROC FORECAST での予測手法の使用方法の詳細については、*SAS/ETS ユーザーガイド*を参照してください。

例

次のプログラムは、INTCYCLE 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl char(10) cycle_year;
  method init();
    cycle_year=intcycle('year');
    put cycle_year;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
YEAR
```

```
proc ds2;
data _null_;
  dcl char(10) cycle_quarter;
  method init();
    cycle_quarter=intcycle('qtr');
    put cycle_quarter;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
YEAR
```

```
proc ds2;
data test(overwrite=yes);
  dcl char(10) cycle_3;
  method init();
    cycle_3=intcycle('month', 3);
    put cycle_3;
  end;
enddata;
```

```
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
QTR
```

```
proc ds2;
data test(overwrite=yes);
  dcl char(10) cycle_month;
  method init();
    cycle_month=intcycle('month');
    put cycle_month;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
YEAR
```

```
proc ds2;
data test(overwrite=yes);
  dcl char(10) cycle_weekday;
  method init();
    cycle_weekday=intcycle('weekday');
    put cycle_weekday;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
WEEK
```

```
proc ds2;
data  test(overwrite=yes);
  dcl char(10) cycle_weekday2;
  method init();
    cycle_weekday2=intcycle('weekday', 5);
    put cycle_weekday2;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
WEEK
```

```
proc ds2;
data test(overwrite=yes);
  dcl char(10) cycle_day;
  method init();
```

```
    cycle_day=intcycle('day');
    put cycle_day;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
WEEK
```

```
proc ds2;
data  test(overwrite=yes);
  dcl char(10) cycle_day2;
  method init();
    cycle_day2=intcycle('day', 10);
    put cycle_day2;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
TENDAY
```

```
proc ds2;
data test(overwrite=yes);
  dcl char(10) cycle_second;
  dcl char(6) var1;
  method init();
    var1='second';
    cycle_second=intcycle(var1);
    put cycle_second;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
DTMINUTE
```

関連項目

関数:

- “INTCINDEX 関数” (719 ページ)
- “INTINDEX 関数” (745 ページ)
- “INTSEAS 関数” (767 ページ)

その他のリファレンス:

- *SAS/ETS User's Guide*

INTDT 関数

DATE 値に追加する日数を指定します。

カテゴリ: 日付と時間

返されるデータの種類: DATE

構文

INTDT(*expression*, *increment*)

必須引数

expression
DATE 値を表す有効な式を指定します。

データの種類 DATE

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

increment
日付に追加する日数を表す負、正またはゼロの整数を指定します。

データの種類 DOUBLE

詳細

INTDT 関数は、指定した日数だけ DATE 値を増分します。

比較

INTNX 関数は、DOUBLE 値としてエンコードされた SAS 日付、時間、または日時の値を増分します。

例

次のプログラムは、INTDT 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl date a b c d;
  method run();
    a= date '2019-05-01';
    b=intdt(a, 25);
    put a;
    put b;
    c= date '2019-05-01';
    d=intdt(c, -25);
    put c;
    put d;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
2019-05-01
2019-05-26
2019-05-01
2019-04-06
```

関連項目

- *"DS2 式" (SAS DS2 プログラマガイド)*

関数:

- "INTCK 関数" (724 ページ)
- "INTNX 関数" (755 ページ)
- "INTTS 関数" (779 ページ)

INTFIT 関数

2 つの日付に基づく時間間隔を返します。

カテゴリ:	日付と時間
制限事項:	DS2 は、カスタムの日付または時間間隔をサポートしていません。
返されるデータの種類:	CHAR、NCHAR、NVARCHAR、VARCHAR

構文

INTFIT(*expresssion-1*, *expression-2*, '*type*')

必須引数

expression
文字列を評価するか、文字列に強制変換できて、SAS 日付または日時値を表す有効な式を指定します。

データの種類 DOUBLE

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

'*type*'
引数が SAS 日付値、日時値、行のいずれであるかを指定します。*type* の有効な値は、d (日付)、dt (日時)、および obs (行)です。

引数が SAS 日付値、日時値、行のいずれであるかを指定します。

type の有効な値は次のとおりです。

d
expression-1 と *expression-2* が日付値であることを指定します。

dt
expression-1 と *expression-2* が日時値であることを指定します。

obs
expression-1 と *expression-2* が行であることを指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

詳細

INTFIT 関数は、間隔内に配置された 2 つの日付値、日時値または行に基づいて、間隔内の最も可能性の高い時間間隔を返します。INTFIT では配置の値は SAME とみなされ、対応する間隔を増分したカレンダー日付と同じ日付を使用します。*alignment* 引数の詳細については、"INTNX 関数" (*SAS 関数と CALL ルーチン: リファレンス*)を参照してください。

INTFIT で行の引数を使用する場合、行番号を使用してそのサイクルを確認できます。次の例では、INTFIT の最初の 2 つの引数は行番号で、*type* 引数は obs です。調査員がデータを記録した 1 回目と 25 回目に Jason がジムを使用した場合、次のステートメントを使用して間隔を確認できます。interval=intfit(1, 25, 'obs');.この場合、間隔の値は 24.2 です。

時系列の詳細については、*SAS/ETS ユーザーガイド*を参照してください。

INTFIT 関数は、小売業界によるカレンダーの間隔でも使用できます。これらの間隔は ISO 8601 に準拠します。詳細については、"ISO 8601 規格の基本表記と拡張表記を使用した日付と時間の処理" (*SAS 出力形式と入力形式: リファレンス*)を参照してください。

例

次のプログラムは、2 つの日付に基づく間隔を示しています。このプログラムの *type* 引数は、入力を日付値として識別します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl char(10) c;
  dcl double sasdate1 sasdate2;
  method run();
    sasdate1=to_double(date'2013-08-01');
    sasdate2=to_double(date'2013-09-01');
    c=intfit(sasdate1, sasdate2, 'd');
    put c;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
MONTH
```

関連項目

関数:

- “INTCK 関数” (724 ページ)
- “INTNX 関数” (755 ページ)

INTGET 関数

3 つの日付値または日時値に基づく時間間隔を返します。

カテゴリ:	日付と時間
制限事項:	DS2 は、カスタムの日付または時間間隔をサポートしていません。
返されるデータの種類:	CHAR、NCHAR、NVARCHAR、VARCHAR

構文

INTGET(*date-1*, *date-2*, *date-3*)

必須引数

date
SAS 日付または日時の値に評価される有効な式を指定します。

データの種類 DOUBLE

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

詳細

INTGET 関数の間隔

INTGET 関数は、3 つの日付値または日時値に基づく時間間隔を返します。この関数は最初の 2 つの日付間で可能なすべての間隔を最初に確認し、次に 2 番目と 3 番目の日付間で可能なすべての間隔を確認します。すべての間隔が同じ場合、INTGET はその間隔を返します。1 番目と 2 番目の日付間に異なる間隔があり、2 番目と 3 番目の日付間に異なる間隔がある場合、INTGET はそれらの間隔を比較します。他の間隔の倍数となる間隔がある場合、INTGET はその小さい方の間隔を返します。それ以外の場合、INTGET は欠損値を返します。INTGET は、配置の値が BEGIN の INTNX 関数によって生成された日付に適しています。

次の例では、INTGET は間隔 DAY2 を返します。

```
interval=intget('01mar00'd, '03mar00'd, '09mar00'd);
```

2000 年 3 月 1 日から 2000 年 3 月 3 日までの日数は 2 であるため、1 番目と 2 番目の日付間の間隔は DAY2 です。2000 年 3 月 3 日から 2000 年 3 月 9 日までの日数は 6 であるため、2 番目と 3 番目の日付間の間隔は DAY6 です。DAY6 は DAY2 の倍数です。INTGET はこの 2 つの間隔のうち小さい方を返します。

次の例では、INTGET は間隔 MONTH4 を返します。

```
interval=intget('01jan00'd, '01may00'd, '01may01'd);
```

2000 年 1 月 1 日から 2000 年 5 月 1 日までの月数は 4 であるため、最初の 2 つの日付間の間隔は MONTH4 です。2 番目と 3 番目の日付間の間隔は YEAR です。YEAR は MONTH4 の倍数(YEAR には 3 つの MONTH4 間隔が含まれる)であるため、INTGET は 2 つの間隔のうち小さい方の間隔を返します。

次の例では、INTGET は欠損値を返します。

```
interval=intget('01Jan2006'd, '01Apr2006'd, '01Dec2006'd);
```

最初の 2 つの日付間の間隔は MONTH3、2 番目と 3 番目の日付間の間隔は MONTH8 です。MONTH8 は MONTH3 の倍数ではないため、INTGET は欠損値を返します。

返される間隔は有効な SAS 間隔で、間隔とシフト間隔の倍数が含まれます。有効な SAS 間隔は、"日付間隔と時間間隔について" (*SAS 出力形式と入力形式: リファレンス*)にリストされています。

注: INTGET が一致する間隔を確認できない場合、関数は欠損値を返します。SAS ログにメッセージは書き込まれません。

小売カレンダーの間隔

INTGET 関数は、小売業界によるカレンダーの間隔でも使用できます。これらの間隔は ISO 8601 に準拠します。詳細については、"ISO 8601 規格の基本表記と拡張表記を使用した日付と時間の処理" (*SAS 出力形式と入力形式: リファレンス*)を参照してください。

例

次のプログラムは、INTGET 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl char(10) c;
  dcl double sasdate1 sasdate2 sasdate3;
  method run();
    sasdate1=to_double(date'2013-01-01');
    sasdate2=to_double(date'2014-01-01');
    sasdate3=to_double(date'2014-05-01');
    c=intget(sasdate1, sasdate2, sasdate3);
    put c;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
MONTH4
```

```
proc ds2;
data test (overwrite=yes);
  dcl char(10) c;
  dcl double sasdate1 sasdate2 sasdate3;
  method run();
    sasdate1=to_double(date'2012-02-29');
    sasdate2=to_double(date'2014-02-28');
    sasdate3=to_double(date'2016-02-29');
    c=intget(sasdate1, sasdate2, sasdate3);
    put c;
```

```
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
YEAR2.2
```

```
proc ds2;
data test (overwrite=yes);
  dcl char(10) c;
  dcl double sasdate1 sasdate2 sasdate3;
  method run();
    sasdate1=to_double(date'2013-02-01');
    sasdate2=to_double(date'2013-02-16');
    sasdate3=to_double(date'2013-03-01');
    c=intget(sasdate1, sasdate2, sasdate3);
    put c;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
SEMIMONTH
```

```
proc ds2;
data test (overwrite=yes);
  dcl char(10) c;
  dcl double sasdate1 sasdate2 sasdate3;
  method run();
    sasdate1=to_double(date'2013-01-02');
    sasdate2=to_double(date'2014-02-02');
    sasdate3=to_double(date'2015-03-02');
    c=intget(sasdate1, sasdate2, sasdate3);
    put c;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
MONTH13.13
```

```
proc ds2;
data test (overwrite=yes);
  dcl char(10) c;
  dcl double sasdate1 sasdate2 sasdate3;
  method run();
    sasdate1=to_double(date'2013-02-10');
    sasdate2=to_double(date'2013-02-19');
    sasdate3=to_double(date'2013-02-28');
    c=intget(sasdate1, sasdate2, sasdate3);
```

```
    put c;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
DAY9.5
```

```
proc ds2;
data test (overwrite=yes);
  dcl char(10) c;
  dcl double sasdate1 sasdate2 sasdate3;
  method run();
    sasdate1=to_double(timestamp'2014-04-01 01:03:00.0000');
    sasdate2=to_double(timestamp'2014-04-01 01:04:00.0000');
    sasdate3=to_double(timestamp'2014-04-01 01:05:00.0000');
    c=intget(sasdate1, sasdate2, sasdate3);
    put c;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
MINUTE
```

関連項目

関数:

- “INTFIT 関数” (739 ページ)
- “INTNX 関数” (755 ページ)

INTINDEX 関数

季節インデックスを返します。この関数には、日付、時間またはタイムスタンプの間隔と値を指定します。

カテゴリ:	日付と時間
制限事項:	DS2 は、カスタムの日付または時間間隔をサポートしていません。
返されるデータの種類:	DOUBLE

構文

INTINDEX({*interval*[*multiple*][.*shift-index*]}, *date-value*[, *seasonality*])

必須引数

interval
文字列を評価するか、文字列に強制変換でき、WEEK、MONTH、QTR などの間隔名を含む文字定数、変数、または式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

注 *interval* の可能な値は、"日付間隔と時間間隔について" (*SAS 出力形式と入力形式: リファレンス*)にリストされています。

ヒント *Interval* は、大文字または小文字で表示できます。

例 YEAR は、年ベースの間隔を指定します。

date-value
指定した間隔の期間を表す日付、時間またはタイムスタンプの値を指定します。

データの種類 DOUBLE

オプション引数

multiple
乗数を指定します(省略可能)。基本タイプの間隔の期間に対する倍数と同等の間隔を設定します。

データの種類 DOUBLE

参照項目: 詳細については、"乗数とシフト間隔を使用した日時の増分" (*SAS 関数と CALL ルーチン: リファレンス*)を参照してください。

例 YEAR2 は、2 年、つまり 2 年間続く間隔タイプを指定します。

shift-index
シフトインデックスを指定します(省略可能)。間隔の開始時点を指定したサブ期間にシフトします。

制限事項 シフトインデックスは、間隔全体のサブ期間の数以下にする必要があります。たとえば、2 年間隔では 25 番目の月は存在しないため、YEAR2.24 は使用できますが YEAR2.25 はエラーになります。

デフォルトのシフト期間が間隔の種類と同じ場合、複数期間の間隔のみを任意のシフトインデックスでシフトできます。たとえば、MONTH の種類の間隔はデフォルトでは MONTH のサブ期間でシフトされるため、シフトインデックスで月間隔をシフトできません。ただし、各 MONTH2 間隔には 2 つの MONTH 間隔が含まれるため、シフトインデックスで 2 か月

間隔をシフトできます。たとえば、間隔名 MONTH2.2 では、偶数月の 1 日目に開始する 2 か月の期間が指定されます。

データの種類 DOUBLE

参照項目: 詳細については、"乗数とシフト間隔を使用した日時の増分" (*SAS 関数と CALL ルーチン: リファレンス*)を参照してください。

例 YEAR.3 は、暦年の 3 月 1 日に開始し、翌年の 2 月に終了するようにシフトされた年次期間を指定します。

seasonality
数値またはサイクルを指定します。この引数では、日付と時間のサイクルをより柔軟に操作できます。1 年間の季節性を 52 週にするか 53 週にするかを指定できます。

数値またはサイクルを指定します。

この引数では、日付と時間のサイクルをより柔軟に操作できます。1 年間の季節性を 52 週にするか 53 週にするかを指定できます。

データの種類 DOUBLE、CHAR、NCHAR、NVARCHAR、VARCHAR

例 この例では、次の関数で同じ結果が生成されます。

```
INTINDEX('MONTH', sasdate, 3);
INTINDEX('MONTH', sasdate, 'QTR');
```

最初の例の *seasonality* は数値(月数)で、2 番目の例の *seasonality* はサイクル(QTR)です。

詳細

基本

INTINDEX 関数は、季節インデックスを返します。この関数には、間隔と適切な日付、時間またはタイムスタンプの値を指定します。季節インデックスは、指定した間隔の季節サイクルでの日付、時間またはタイムスタンプの値の位置を表す数値です。1 年サイクルには 12 か月があり 12 月はその年の 12 番目の月であるため、この例では 12 の値が返されます。

```
sasdate=to_double(date'2012-12-01');
x=intindex('month', sasdate);
put x;
```

次の例では、両方のステートメントが 2013 年第 1 四半期内の日付を含むため、INTINDEX は同じ値(1)を返します。

```
sasdate=to_double(date'2013-01-01');
x=intindex('qtr', sasdate);
put x;
```

```
sasdate=to_double(date'2013-03-31');
y=intindex('qtr', sasdate);
put y;
```

次の例では、日単位のデータは週単位の周期に含まれ、2012 年 12 月 7 日は金曜日で 6 番目の曜日なので、6 の値を返します。

```
sasdate=to_double(date'2012-12-07');
x=intindex('day', sasdate);
put x;
```

間隔

間隔は、基本的なものから複雑なものまであります。基本間隔は、SAS が経過時間内にカウントできる測定単位(DAY、MONTH、HOUR など)です。より複雑な間隔を指定するには、基本間隔名に乗数とシフトインデックスを組み合わせて使用できます。

間隔構文は次のとおりです。

interval[*multiple*][.*shift-index*]

詳細については、“必須引数” (746 ページ)を参照してください。

Interval と *Date-Time-Value* の関係

季節インデックスを正しく識別するには、間隔が日付、時間またはタイムスタンプの値に適合する必要があります。たとえば、intindex('month', '01DEC2012'd);では、1 年の間隔には 12 か月があり 12 月はその年の 12 番目の月であるため、12 の値を返します。MONTH 間隔には SAS 日付値が必要です。週間隔には 7 日あり、2012 年 12 月 7 日は金曜日で 6 番目の曜日なので、次の例では 6 の値が返されます。

```
sasdate=to_double(date'2012-12-07');
x=intindex('day', sasdate);
put x;
```

DAY 間隔には SAS 日付値が必要です。

```
select intget('day', date'2012-12-07');
```

この例では、QTR 間隔が TIMESTAMP 値でなく SAS 日付値を日付に必要とするため、欠損値が返されます。

```
sasdate=to_double(timestamp'2013-01-01 00:00:00');
x=intindex('qtr', sasdate);
put x;
```

この例では、12 の値が返されます。DTMONTH 間隔には TIMESTAMP 値が必要です。

```
sasdate=to_double(timestamp'2013-12-01 00:00:00');
x=intindex('dtmonth', sasdate);
put x;
```

日付間隔と時間間隔の操作の詳細については、“日付間隔と時間間隔” (*SAS 関数と CALL ルーチン: リファレンス*)を参照してください。

小売カレンダーの間隔

INTINDEX 関数は、小売業界によるカレンダーの間隔でも使用できます。これらの間隔は ISO 8601 に準拠します。詳細については、"小売カレンダーの間隔: ISO 8601 準拠" (*SAS 関数と CALL ルーチン: リファレンス*)を参照してください。

季節性

季節性は、年内の異なる間隔での周期的変動を測定する時系列の概念です。季節性の指定では、季節が最も一般的な変動ソースです。たとえば、家庭暖房用の灯油は、一般的に他の季節よりも冬の売り上げの方が高くなります。多くの場合、日単位の時系列では曜日による定期的な変動(週末のレジャーで支出が増えるなど)が発生します。INTINDEX 関数は、季節性の概念を使用して季節インデックスを返します。この関数には、日付、時間またはタイムスタンプの間隔と値を指定します。季節性と PROC FORECAST での予測手法の使用方法の詳細については、*SAS/ETS ユーザーガイド*を参照してください。

比較

INTINDEX 関数は季節インデックスを返しますが、INTCINDEX 関数はサイクルインデックスを返します。

次の例では、2013 年 4 月 4 日は木曜日で週の 5 番目の曜日なので、INTINDEX 関数は 5 を返します。

```
sasdate=to_double(date'2013-04-04');
x = intindex('day', sasdate);
put x;
```

同じ日を使用しても、2013 年 4 月 4 日は年の 14 番目の週なので、INTCINDEX 関数は 14 を返します。

```
sasdate=to_double(date'2013-04-04');
x = intcindex('day', sasdate);
put x;
```

この例では、INTINDEX 関数は分を返します。

```
sasdate=to_double(timestamp'2012-09-01 06:05:04');
x = intindex('minute', sasdate);
put x;
```

同じ日付と時間を使用しても、INTCINDEX 関数は時刻を返します。

```
sasdate=to_double(timestamp'2012-09-01 06:05:04');
y = intcindex('minute', sasdate);
put y;
```

例 intseas('interval');では、INTSEAS は intindex('interval', date);によって返される最大数を返します。

例

次のプログラムは、INTINDEX 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double sasdate1 interval1;
  method run();
    sasdate1=to_double(date'2013-08-14');
    interval1 = intindex('qtr', sasdate1);
    put interval1;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
3
```

```
proc ds2;
data test (overwrite=yes);
  dcl double sasts1 interval2;
  method run();
    sasts1=to_double(timestamp'2013-12-23 15:09:19');
    interval2 = intindex('dtqtr', sasts1);
    put interval2;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
4
```

```
proc ds2;
data test (overwrite=yes);
  dcl double sastime1 interval3;
  method run();
    sastime1=to_double(time'09:05:15');
    interval3 = intindex('hour', sastime1);
    put interval3;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
10
```

```
proc ds2;
data test (overwrite=yes);
  dcl double sasdate1 interval4;
  method run();
    sasdate1=to_double(date '2013-02-26');
    interval4 = intindex('month', sasdate1);
    put interval4;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
2
```

```
proc ds2;
data test (overwrite=yes);
  dcl double sasts1 interval5;
  method run();
    sasts1=to_double(timestamp'2013-05-28 05:15:00');
    interval5 = intindex('dtmonth', sasts1);
    put interval5;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
5
```

```
proc ds2;
data test (overwrite=yes);
  dcl double sasdate1 interval6;
  method run();
    sasdate1=to_double(date '2013-09-09');
    interval6 = intindex('week', sasdate1);
    put interval6;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
37
```

```
proc ds2;
data test (overwrite=yes);
  dcl double sasdate1 interval7;
  method run();
    sasdate1=to_double(date '2013-04-16');
    interval7 = intindex('tenday', sasdate1);
    put interval7;
  end;
```

```
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
11
```

関連項目

関数:

- “INTCINDEX 関数” (719 ページ)
- “INTCYCLE 関数” (732 ページ)
- “INTSEAS 関数” (767 ページ)

その他のリファレンス:

- *SAS/ETS User's Guide*

INTNEST 関数

より大きな間隔の期間に収まる小さい間隔の全期間の数を計算します。

カテゴリ:	日付と時間
制限事項:	DS2 は、カスタムの日付または時間間隔をサポートしていません。
返されるデータの種類:	DOUBLE

構文

INTNEST(*interval1*, *interval2*)

必須引数

interval1
最初の間隔を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

interval2
2 番目の間隔を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

詳細

最初の間隔の整数がすべての期間の 2 番目の間隔と同じ期間にまたがる場合、間隔は別の間隔内にネストされます。ネストするためには、2 つの間隔で、配置する開始日と終了日を生成する必要があります。

最初の間隔 *interval1* が 2 番目の間隔 *interval2* よりも長期間にわたる場合、返されるのは正の数です。2 番目の間隔が最初の間隔よりも長期間にわたる場合、返されるのは負の数です。

次の時系列タスクは INTNEST 関数に関連しています。

累積
: 1 つの間隔が別の間隔にネストされている場合、可変数であっても、短い期間から長い期間への累積は単純なルールで達成されます。間隔がネストされていない場合は、より複雑なルール(補間など)を使用して、時系列をある頻度から別の頻度に変換することを検討してください。

季節性
: 多くの季節モデルでは、期間数が固定されている高頻度間隔を低頻度季節間隔にネストする必要があります。

時間調整
: 時間調整では、高頻度間隔が低頻度間隔にネストする必要があります。

次の表には、INTNEST 関数によって返される結果の種類と説明が含まれています。

表 7.5　INTNEST 関数の結果と説明

結果	説明	説明または例
0	同一	2 つの入力間隔は、すべての 期間に対して同じ期間を定義します。 intnest('month12','year')
1	可変数	最初の間隔には 2 番目の間隔の整数の期間が含まれますが、その数は時間とともに変化します。 intnest('month','day')
-1	可変数	2 番目の間隔には最初の間隔の整数の期間が含まれますが、その数は時間とともに変化します。 intnest('day',''year')
n > 1	個定数	最初の間隔には 2 番目の間隔の整数 *n* 期間が含まれますが、これは常に固定されています。 intnest('week', 'day')
n < -1	個定数	2 番目の間隔には最初の間隔の整数 *n* 期間が含まれますが、これは常に固定されています。

結果	説明	説明または例
		intnest('dthour', 'day')
M の欠損値	複数の不一致	どちらの間隔も、もう一方の間隔にネストできません。ただし、これらのタイプの間隔は、一部の複数の値に対してネストできます。 intnest('semimonth3', 'month')
S の欠損値	シフトの不一致	どちらの間隔も、もう一方の間隔にネストできません。ただし、シフト値が変更されると、間隔が同じになるか、1 つの間隔が別の間隔にネストされます。 intnest('semimonth2.2', 'month')
B の欠損値	ベースの不一致	間隔ベースで、まったく異なるため任意の複数またはシフトに対するネストが不可能な期間が定義されています。たとえば、YEAR は常に毎年 1 月 1 日に始まり、月単位でシフトされます。ただし、YEARV は常に 1 月 4 日の直前または月曜日に始まり、月曜日に始まる ISO 8601 週単位でシフトされます。年によっては 1 月 1 日は単なる月曜日であるため、間隔が一貫して同じ日に開始されることはありません。月の最初がすべての年の月曜日ではないため、YEAR 間隔が月単位でシフトされる場合も同じ問題が発生します。 intnest('year', 'yearv')

例

次のプログラムは、INTNEST 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double a;
  dcl char(10) x y;
  method run();
    x='SEMIMONTH3';
    y='MONTH';
    a=intnest(x, y);
    put a;
  end;
enddata;
run;
```

```
quit;
```

SAS は次の出力をログに書き込みます。

```
M
```

INTNX 関数

DOUBLE としてエンコードされた SAS 日付、時間または日時の値を増分し、DOUBLE としてエンコードされた SAS 日付、時間または日時の値を返します。

カテゴリ: 日付と時間

制限事項: DS2 は、カスタムの日付または時間間隔をサポートしていません。

返されるデータの種類: DOUBLE

構文

INTNX({*interval*[*multiple*][.*shift-index*]}, *start-from*, *increment*[, '*alignment*'])

INTNX(*start-from*, *increment*[, '*alignment*'])

必須引数

interval
文字列を評価するか、文字列に強制変換でき、WEEK、MONTH、QTR などの間隔名を含む文字定数、変数、または式を指定します。

データの種類: CHAR、NCHAR、NVARCHAR、VARCHAR

注: *interval* の可能な値は、"日付間隔と時間間隔について" (*SAS 出力形式と入力形式: リファレンス*)にリストされています。

ヒント: *Interval* は、大文字または小文字で表示できます。

例: YEAR は、年ベースの間隔を指定します。

start-from
DOUBLE としてエンコードされた SAS 日付、時間または日時の値を表し、開始点を識別する式を指定します。

データの種類: DOUBLE

increment
日付、時間または日時の間隔を表す負、正またはゼロの整数を指定します。*Increment* は、*start-from* の値をシフトする間隔数です。

データの種類 DOUBLE

オプション引数

multiple
乗数を指定します(省略可能)。基本タイプの間隔の期間に対する倍数と同等の間隔を設定します。

データの種類 DOUBLE

参照項目: 詳細については、"乗数とシフト間隔を使用した日時の増分" (*SAS 関数と CALL ルーチン: リファレンス*)を参照してください。

例 YEAR2 は、2 年、つまり 2 年間続く間隔タイプを指定します。

shift-index
シフトインデックスを指定します(省略可能)。間隔の開始時点を指定したサブ期間にシフトします。

制限事項 シフトインデックスは、間隔全体のサブ期間の数以下にする必要があります。たとえば、2 年間隔では 25 番目の月は存在しないため、YEAR2.24 は使用できますが YEAR2.25 はエラーになります。

デフォルトのシフト期間が間隔の種類と同じ場合、複数期間の間隔のみを任意のシフトインデックスでシフトできます。たとえば、MONTH の種類の間隔はデフォルトでは MONTH のサブ期間でシフトされるため、シフトインデックスで月間隔をシフトできません。ただし、各 MONTH2 間隔には 2 つの MONTH 間隔が含まれるため、シフトインデックスで 2 か月間隔をシフトできます。たとえば、間隔名 MONTH2.2 では、偶数月の 1 日目に開始する 2 か月の期間が指定されます。

データの種類 DOUBLE

参照項目: 詳細については、"乗数とシフト間隔を使用した日時の増分" (*SAS 関数と CALL ルーチン: リファレンス*)を参照してください。

例 YEAR.3 は、暦年の 3 月 1 日に開始し、翌年の 2 月に終了するようにシフトされた年次期間を指定します。

'*alignment*'
間隔内の SAS 日付の位置を制御します。*alignment* は引用符で囲む必要があります。*alignment* には、BEGINNING、MIDDLE、END、SAME のいずれかの値を指定します。SAME では、返された日付を入力日付と同じ場所に配置するように指定されます。詳細については、SAS DS2 言語リファレンスを参照してください。

間隔内の SAS 日付の位置を制御します。*alignment* は引用符で囲む必要があります。*Alignment* には次のいずれかの値を指定できます。

BEGINNING
返された日付値または日時値を間隔の開始点に配置するように指定します。

別名 B

MIDDLE
返された日付値または日時値を間隔の中間点(開始位置と終了位置の値の平均)に配置するように指定します。

別名 M

END
返された日付値または日時値を間隔の終了点に配置するように指定します。

別名 E

SAME
返された日付を入力日付と同じ場所に配置するように指定します。

別名 S

SAMEDAY

参照項目: “SAME 配置” (758 ページ)

デフォルト BEGINNING

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

参照項目: “SAS 日付出力の間隔内の配置” (758 ページ)

詳細

基本

INTNX 関数は、DAY、WEEK、QTR、MINUTE などの間隔、または定義したカスタム間隔単位で日付、時間または日時の値を増分します。間隔は、開始日付、時間または日時の値、および指定した時間間隔に基づきます。

INTNX 関数は、*start-from* 引数で指定した間隔の開始日付、時間または日時の値に対する SAS 日付値を返します。(日付値をカレンダー日付に変換するには、DATE9. 形式などの有効な DS2 日付形式を使用します)。2011 年 10 月 17 日の週から 6 週間後の週の開始日を確認する方法の例を次に示します。

```
sasdate=to_double(date'2011-10-17');
x=intnx('week', sasdate, 6);
put x date9.;
```

INTNX は値 27NOV2011 を返します。

日付間隔と時間間隔の操作の詳細については、“日付間隔と時間間隔” (*SAS 関数と CALL ルーチン: リファレンス*)を参照してください。

間隔

間隔は、基本的なものから複雑なものまであります。基本間隔は、SAS が経過時間内にカウントできる測定単位(DAY、MONTH、HOUR など)です。より複雑な間隔を指定するには、基本間隔名に乗数とシフトインデックスを組み合わせて使用できます。

間隔構文は次のとおりです。

interval[*multiple*][.*shift-index*]

詳細については、“必須引数” (755 ページ)を参照してください。

SAS 日付出力の間隔内の配置

通常、SAS 日付値は *interval* 引数で指定した時間間隔の開始点に配置されます。

任意の *alignment* 引数を使用して、返された値の配置を指定できます。値 BEGINNING、MIDDLE または END は、それぞれ間隔の開始点、中間点または終了点に日付を配置します。

SAME 配置

alignment 引数の SAME 値を使用した場合、INTNX は指定した間隔の増分を計算した後に同じカレンダー日付を返します。同じカレンダー日付は、間隔ではなく間隔のシフト期間に基づいて配置されます。有効なシフト期間を表示するには、“カテゴリ別の間隔” (727 ページ)を参照してください。

シフト期間のほとんどの値は、その対応する間隔と等しくなります。この例外は、間隔 WEEK、WEEKDAY、QTR、SEMIYEAR、YEAR、およびこれらに対応する DT です。WEEK および WEEKDAY 間隔には DAYS のシフト期間が含まれ、QTR、SEMIYEAR および YEAR 間隔には MONTH のシフト期間が含まれます。たとえば、YEAR で SAME 配置を使用した場合、この間隔のシフト期間である MONTH に基づいて同日に配置されます。YEAR 間隔の同日に配置されるわけではありません。倍数の間隔を指定した場合、デフォルトのシフト間隔は倍数の間隔ではなく元の間隔に基づきます。

QTR、SEMIYEAR および YEAR 間隔に SAME 配置を使用する場合、計算結果日付は間隔の開始点(入力日付)からの月数と同じです。月の日は可能な限り同じ日に合わせられます。月によっては日数が異なるため、月の日を常に一致させることは不可能です。

シフト期間の詳細については、“カテゴリ別の間隔” (727 ページ)を参照してください。

配置の間隔

入力日付の配置に基づいて計算結果日付を配置するには、*alignment* 引数に SAME 値を使用します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
```

```
  dcl double x having format date9.;
  dcl double y having format datetime.;
  dcl double z having format date9.;
  dcl double sasdatex sasdatey sasdatez;
  method init();
    sasdatex=to_double(date'2011-03-15');
    x=intnx('week', sasdatex, 1, 'same');
    sasdatey=to_double(timestamp'2011-03-15 08:45:00');
    y=intnx('dtweek', sasdatey, 1, 'same');
    sasdatez=to_double(date'2011-03-15');
    z=intnx('year', sasdatez, 5, 'same');
    put x;
    put y;
    put z;
  end;
enddata;
run;
quit;
```

日付の調整

増分された間隔の日付が存在しない場合、INTNX 関数はその日付を自動的に調整します。次に例を示します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double a b c  d e having format date9.;
  dcl double sasdate;
  method init();
    sasdate=to_double(date'2011-03-15');
    a=intnx('month', sasdate, 5, 'same');
    put a;
    sasdate=to_double(date'2012-02-29');
    b=intnx('year', sasdate, 2, 'same');
    put b;
    sasdate=to_double(date'2011-08-31');
    c=intnx('month', sasdate, 1, 'same');
    put c;
    sasdate=to_double(date'2011-03-01');
    d=intnx('year', sasdate, 1, 'same');
    put d;
    sasdate=to_double(date'2011-03-01');
    e=intnx('year', sasdate, 1, 'same', 'day');
    put e;
  end;
enddata;
run;
quit;
```

次の例では、INTNX 関数は値 01JAN2014 を返します。この値は、開始日 (29FEB2012)の年から 2 年後の最初の日付です。

```
dcl double a having format date9.;
```

```
sasdate=to_double(date'2012-02-29');
a=intnx('year', sasdate, 2);
put a;
```

この例では、INTNX 関数は値 28FEB2014 を返します。この場合、開始日は 2012 年で、結果となる年は 2 年後(2014)、月は同月(2 月)、日付は 28 日が 2014 年 2 月 29 日に最も近いため、これらの値になります。

```
dcl double b having format date9.;
sasdate=to_double(date'2012-02-29');
b=intnx('year', sasdate, 2, 'same');
put b;
```

小売カレンダーの間隔

小売業界では、1 年のカレンダーを 13 週間からなる 4 つの期間に分けてデータを計算することがよくあります。期間の形式は、4-4-5、4-5-4 または 5-4-4 のいずれかに基づきます。1 番目、2 番目、3 番目の数値には、それぞれ各期間の 1 か月目、2 か月目、3 か月目の週数を指定します。詳細については、"ISO 8601 規格の基本表記と拡張表記を使用した日付と時間の処理" (*SAS 出力形式と入力形式: リファレンス*) を参照してください。

例

例 1: INTNX 関数の使用

次のプログラムは、INTNX 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double sasdate1 yr;
  method run();
    sasdate1 = to_double(date'2019-02-05');
    yr=intnx('year', sasdate1, 3);
    put yr;
    put yr date7.;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
22646
01JAN22
```

```
proc ds2;
data test (overwrite=yes);
  dcl double sasdate1 x;
```

```
  method run();
    sasdate1 = to_double(date'2019-01-05');
    x=intnx('month', sasdate1, 0);
    put x;
    put x date7.;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
21550
01JAN19
```

```
proc ds2;
data test (overwrite=yes);
  dcl double sasdate1 next;
  method run();
    sasdate1 = to_double(date'2019-01-01');
    next=intnx('semiyear', sasdate1, 1);
    put next;
    put next date7.;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
21731
01JUL19
```

```
proc ds2;
data test (overwrite=yes);
  dcl double sasdate1 past;
  method run();
    sasdate1 = to_double(date'2019-08-01');
    past=intnx('month2', sasdate1, -1);
    put past;
    put past date7.;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
21670
01MAY19
```

```
proc ds2;
data test (overwrite=yes);
  dcl double sasdate1 sm;
  method run();
    sasdate1 = to_double(date'2019-04-01');
    sm=intnx('semimonth2.2', sasdate1, 4);
```

```
    put sm;
    put sm date7.;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
21746
16JUL19
```

```
proc ds2;
data test (overwrite=yes);
  dcl double sasdate1 nextmon;
  dcl char x;
  method run();
    x='month';
    sasdate1 = to_double(date'2019-06-01');
    nextmon=intnx(x, sasdate1, 1);
    put nextmon;
    put nextmon date7.;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
21731
01JUL19
```

```
proc ds2;
data test (overwrite=yes);
  dcl double sasdate1 x;
  dcl char m1 m2;
  method run();
    m1='month    ';
    m2=trim(m1);
    sasdate1 = to_double(date'2019-06-15') - 100;
    x=intnx(m2, sasdate1, 1);
    put x;
    put x date7.;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
21640
01APR19
```

例 2: ALIGNMENT 引数の使用

任意の *alignment* 引数を使用して日付を進めるプログラムを次に示します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double sasdate1 x;
  method run();
    sasdate1 = to_double(date'2019-01-01');
    x=intnx('month', sasdate1, 5, 'beginning');
    put x;
    put x date7.;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
21701
01JUN19
```

```
proc ds2;
data test (overwrite=yes);
  dcl double sasdate1 x;
  method run();
    sasdate1 = to_double(date'2019-01-01');
    x=intnx('month', sasdate1, 5, 'middle');
    put x;
    put x date7.;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
21715
15JUN19
```

```
proc ds2;
data test (overwrite=yes);
  dcl double sasdate1 x;
  method run();
    sasdate1 = to_double(date'2019-01-01');
    x=intnx('month', sasdate1, 5, 'end');
    put x;
    put x date7.;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
21730
30JUN19
```

```
proc ds2;
data test (overwrite=yes);
  dcl double sasdate1 x;
  method run();
    sasdate1 = to_double(date'2019-01-01');
    x=intnx('month', sasdate1, 5, 'sameday');
    put x;
    put x date7.;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
21701
01JUN19
```

```
proc ds2;
data test (overwrite=yes);
  dcl double sasdate1 x;
  method run();
    sasdate1 = to_double(date'2019-03-15');
    x=intnx('month', sasdate1, 5, 'same');
    put x;
    put x date7.;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
21776
15AUG19
```

```
proc ds2;
data test (overwrite=yes);
  dcl double sasdate1 x;
  dcl char interval align;
  method run();
    interval='month';
    align='m';
    sasdate1 = to_double(date'2019-09-01');
    x=intnx(interval, sasdate1,2, align);
    put x;
    put x date7.;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
21868
15NOV19
```

```
proc ds2;
data test (overwrite=yes);
  dcl double sasdate1 x;
  dcl char m1 m2;
  method run();
    m1='month    ';
    m2=trim(m1);
    sasdate1 = to_double(date'2019-09-01');
    x=intnx(m2, sasdate1, 2, 'm');
    put x;
    put x date7.;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
21868
15NOV19
```

関連項目

関数:

- “INTCK 関数” (724 ページ)
- “INTDT 関数” (738 ページ)
- “INTSHIFT 関数” (772 ページ)
- “INTTS 関数” (779 ページ)

その他のリファレンス:

- “DS2 の日付と時間” (*SAS DS2 プログラマガイド*)

INTRR 関数

内部利益率を小数値で返します。

カテゴリ:	財務
返されるデータの種類:	DOUBLE

構文

INTRR(*freq*, *c0, c1*[..., *cn*])

必須引数

freq
目的の内部利益率に関連付けられた指定ベース期間での支払い数を示す数値です。

範囲 *freq* > 0

データの種類 DOUBLE

ヒント *freq* = 0 は、連続複利計算を許可するフラグです。

c0, c1[..., *cn*]
任意の現金払いを示す数値です。

データの種類 DOUBLE

詳細

INTRR 関数は、現金払いセット *c0*, *c1*,..., *cn* の指定ベース期間での内部利益率を返します。2 つの連続した支払い間の時間間隔は同じとみなされます。引数 *freq* > 0 は、指定ベース期間で発生する支払い数を表します。各インスタンスから発行される情報の数は制限されます。

内部利益率は、一連の支払いの正味現在価値が 0 となる利率です。(“NPV 関数” (908 ページ)を参照してください。)次の式で表されます。

$$r = \begin{cases} \frac{1}{x^{freq}} - 1 & freq > 0 \\ -log_e(x) & freq = 0 \end{cases}$$

この式では、*x* は多項式の実根です。

$$\sum_{i=0}^{n} c_i x^i = 0$$

重根の場合、1 つの実根が返され、返される内部利益率の非一意性に関する警告が発行されます。支払いの値によっては、方程式の根が存在しないことがあります。その場合、欠損値が返されます。

支払いの欠損値は 0 値として扱われます。*freq* > 0 の場合、計算される利益率は、指定ベース期間で有効な利率です。月払いでの四半期の内部利益率(ベース期間は 3 か月)を計算するには、*freq* を 3 に設定します。

freq が 0 の場合は連続複利計算とみなされ、ベース期間は 2 つの連続する支払い間の時間間隔です。計算される内部利益率は、ベース期間の名目利益率です。月単位の支払いを連続複利計算するには、*freq* を 0 に設定します。計算される内部利益率は、月単位の利率です。

比較

IRR 関数は INTRR と似ていますが、IRR 関数では内部利益率がパーセントという点で異なります。

例

当初の支払いが$400 で、次の 3 年間の予定支払い額がそれぞれ$100、$200、$300 の場合、年間内部利益率は次のように計算できます。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double rate;
  method init();
    rate=intrr(1,-400,100,200,300);
    put rate;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
0.19437709962747
```

```
0.1943770996
```

関連項目

関数:

- “IRR 関数” (786 ページ)

INTSEAS 関数

季節サイクルの長さを返します。この関数には、日付、時間または日時の間隔を指定します。

カテゴリ:	日付と時間
制限事項:	DS2 は、カスタムの日付または時間間隔をサポートしていません。
返されるデータの種類:	DOUBLE

構文

INTSEAS({*interval*[*multiple*][.*shift-index*]}[, *seasonality*])

必須引数

interval
文字列を評価するか、文字列に強制変換でき、WEEK、MONTH、QTR などの間隔名を含む文字定数、変数、または式を指定します。

データの種類　CHAR、NCHAR、NVARCHAR、VARCHAR

注　*interval* の可能な値は、"日付間隔と時間間隔について" (*SAS 出力形式と入力形式: リファレンス*)にリストされています。

ヒント　*Interval* は、大文字または小文字で表示できます。

例　YEAR は、年ベースの間隔を指定します。

オプション引数

multiple
乗数を指定します(省略可能)。基本タイプの間隔の期間に対する倍数と同等の間隔を設定します。

データの種類　DOUBLE

参照項目:　詳細については、"乗数とシフト間隔を使用した日時の増分" (*SAS 関数と CALL ルーチン: リファレンス*)を参照してください。

例　YEAR2 は、2 年、つまり 2 年間続く間隔タイプを指定します。

shift-index
シフトインデックスを指定します(省略可能)。間隔の開始時点を指定したサブ期間にシフトします。

制限事項　シフトインデックスは、間隔全体のサブ期間の数以下にする必要があります。たとえば、2 年間隔では 25 番目の月は存在しないため、YEAR2.24 は使用できますが YEAR2.25 はエラーになります。

デフォルトのシフト期間が間隔の種類と同じ場合、複数期間の間隔のみを任意のシフトインデックスでシフトできます。たとえば、MONTH の種類の間隔はデフォルトでは MONTH のサブ期間でシフトされるため、シフトインデックスで月間隔をシフトできません。ただし、各 MONTH2 間隔には 2 つの MONTH 間隔が含まれるため、シフトインデックスで 2 か月間隔をシフトできます。たとえば、間隔名 MONTH2.2 では、偶数月の 1 日目に開始する 2 か月の期間が指定されます。

データの種類　DOUBLE

参照項目: 詳細については、"乗数とシフト間隔を使用した日時の増分" (*SAS 関数と CALL ルーチン: リファレンス*)を参照してください。

例 YEAR.3 は、暦年の 3 月 1 日に開始し、翌年の 2 月に終了するようにシフトされた年次期間を指定します。

seasonality
数値を指定します。この引数では、日付と時間のサイクルをより柔軟に操作できます。1 年間の季節性を 52 週にするか 53 週にするかを指定できます。

数値を指定します。

この引数では、日付と時間のサイクルをより柔軟に操作できます。1 年間の季節性を 52 週にするか 53 週にするかを指定できます。

デフォルト 52

データの種類 DOUBLE、CHAR、NCHAR、NVARCHAR、VARCHAR

例 次の例では、*seasonality* 引数により
INTSEAS('MONTH', 'qtr');
関数呼び出しで値 3 が返されます。この関数呼び出しは
INTSEAS('MONTH');
seasonality 引数がないため、値 12 を返します。

詳細

基本

INTSEAS 関数は、季節サイクル内の間隔数を返します。たとえば、時系列の間隔が月単位の場合、多くのプロシジャでオプション INTERVAL=MONTH が使用されます。この場合、データ内の各オブザベーションは特定の月に対応します。月単位のデータは、1 年間で周期的とみなされます。1 年には 12 か月あるため、季節サイクル(年)内の間隔(月)数は 12 です。

四半期のデータは、1 年間で周期的とみなされます。1 年には 4 つの四半期があるため、季節サイクル内の間隔数は 4 です。

周期性は常に 1 年とは限りません。たとえば、INTERVAL=DAY は 1 週間の周期とみなされます。1 週間には 7 日間あるため、季節サイクルの間隔数は 7 です。

日付間隔と時間間隔の操作の詳細については、"日付間隔と時間間隔" (*SAS 関数と CALL ルーチン: リファレンス*)を参照してください。

間隔

間隔は、基本的なものから複雑なものまであります。基本間隔は、SAS が経過時間内にカウントできる測定単位(DAY、MONTH、HOUR など)です。より複雑な間隔を指定するには、基本間隔名に乗数とシフトインデックスを組み合わせて使用できます。

間隔構文は次のとおりです。

interval[*multiple*][.*shift-index*]

詳細については、“必須引数” (768 ページ)を参照してください。

小売カレンダーの間隔

小売業界では、1 年のカレンダーを 13 週間からなる 4 つの期間に分けてデータを計算することがよくあります。期間の形式は、4-4-5、4-5-4 または 5-4-4 のいずれかに基づきます。1 番目、2 番目、3 番目の数値には、それぞれ各期間の 1 か月目、2 か月目、3 か月目の週数を指定します。詳細については、“ISO 8601 規格の基本表記と拡張表記を使用した日付と時間の処理” (*SAS 出力形式と入力形式: リファレンス*)を参照してください。

季節性

季節性は、年内の異なる間隔での周期的変動を測定する時系列の概念です。季節性の指定では、季節が最も一般的な変動ソースです。たとえば、家庭暖房用の灯油は、一般的に他の季節よりも冬の売り上げの方が高くなります。多くの場合、日単位の時系列では曜日による定期的な変動(週末のレジャーで支出が増えるなど)が発生します。INTSEAS 関数は、季節性の概念を使用して季節サイクルの長さを返します。この関数には、日付、時間または日時の間隔を指定します。季節性と予測の詳細については、*SAS/ETS ユーザーガイド*を参照してください。

例

次のプログラムは、INTSEAS 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double cycle_years cycle_smiyears cycle_quarters cycle_number;
  dcl double cycle_months cycle_smimonths cycle_tendays;
  dcl double cycle_weeks cycle_wkdays cycle_hours  cycle_minutes;
  dcl double cycle_minutes cycle_month2 cycle_week2 cycle_var1;
  dcl double cycle_day1;
  dcl char var1;
  method run();
    cycle_years = intseas('year');
    put cycle_years;

    cycle_smiyears = intseas('semiyear');
    put cycle_smiyears;

    cycle_quarters = intseas('quarter');
    put cycle_quarters;

    cycle_number = intseas('month', 'qtr');
    put cycle_number;
```

```
    cycle_months = intseas('month');
    put cycle_months;

    cycle_smimonths = intseas('semimonth');
    put cycle_smimonths;

    cycle_tendays = intseas('tenday');
    put cycle_tendays;

    cycle_weeks = intseas('week');
    put cycle_weeks;

    cycle_wkdays = intseas('weekday');
    put cycle_wkdays;

    cycle_hours = intseas('hour');
    put cycle_hours;

    cycle_minutes = intseas('minute');
    put cycle_minutes;

    cycle_month2 = intseas('month2.2');
    put cycle_month2;

    cycle_week2 = intseas('week2');
    put cycle_week2;

    var1 = 'month4.3';
    cycle_var1 = intseas(var1);
    put cycle_var1;

    cycle_day1 = intseas('day1');
    put cycle_day1;

  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
1
2
4
3
12
24
36
52
5
24
60
6
26
3
7
```

関連項目

関数:

- “INTCYCLE 関数” (732 ページ)
- “INTINDEX 関数” (745 ページ)

その他のリファレンス:

- *SAS/ETS User's Guide*

INTSHIFT 関数

ベース間隔に対応するシフト間隔を返します。

カテゴリ:	日付と時間
制限事項:	DS2 は、カスタムの日付または時間間隔をサポートしていません。
返されるデータの種類:	CHAR、NCHAR、NVARCHAR、VARCHAR

構文

INTSHIFT(*interval*[*multiple*][*.shift-index*])

必須引数

interval
文字列を評価するか、文字列に強制変換でき、WEEK、MONTH、QTR などの間隔名を含む文字定数、変数、または式を指定します。

データの種類	CHAR、NCHAR、NVARCHAR、VARCHAR
注	*interval* の可能な値は、“日付間隔と時間間隔について” (*SAS 出力形式と入力形式: リファレンス*)にリストされています。
ヒント	*Interval* は、大文字または小文字で表示できます。
例	YEAR は、年間隔を指定します。

オプション引数

multiple
乗数を指定します(省略可能)。基本タイプの間隔の期間に対する倍数と同等の間隔を設定します。

データの種類 DOUBLE

参照項目: 詳細については、"乗数とシフト間隔を使用した日時の増分" (*SAS 関数と CALL ルーチン: リファレンス*)を参照してください。

例 YEAR2 は 2 年の期間で構成されます。

shift-index
シフトインデックスを指定します(省略可能)。間隔の開始時点を指定したサブ期間にシフトします。

制限事項 シフトインデックスは、間隔全体のサブ期間の数以下にする必要があります。たとえば、2 年間隔では 25 番目の月は存在しないため、YEAR2.24 は使用できますが YEAR2.25 はエラーになります。

デフォルトのシフト期間が間隔の種類と同じ場合、複数期間の間隔のみを任意のシフトインデックスでシフトできます。たとえば、MONTH の種類の間隔はデフォルトでは MONTH のサブ期間でシフトされるため、シフトインデックスで月間隔をシフトできません。ただし、各 MONTH2 間隔には 2 つの MONTH 間隔が含まれるため、シフトインデックスで 2 か月間隔をシフトできます。たとえば、間隔名 MONTH2.2 では、偶数月の 1 日目に開始する 2 か月の期間が指定されます。

データの種類 DOUBLE

参照項目: 詳細については、"乗数とシフト間隔を使用した日時の増分" (*SAS 関数と CALL ルーチン: リファレンス*)を参照してください。

例 YEAR.3 は、暦年の 3 月 1 日に開始し、翌年の 2 月に終了するようにシフトされた年次期間を指定します。

詳細

基本

INTSHIFT 関数は、ベース間隔に対応するシフト間隔を返します。カスタム間隔の場合、返される値はベースカスタム間隔名です。INTSHIFT は、間隔と間隔シフトの倍数は無視します。

INTSHIFT 関数は、小売業界によるカレンダーの間隔でも使用できます。これらの間隔は ISO 8601 に準拠します。詳細については、"ISO 8601 規格の基本表記と拡張表記を使用した日付と時間の処理" (*SAS 出力形式と入力形式: リファレンス*)を参照してください。

間隔

間隔は、基本的なものから複雑なものまであります。基本間隔は、SAS が経過時間内にカウントできる測定単位(DAY、MONTH、HOUR など)です。より複雑な間隔を

指定するには、基本間隔名に乗数とシフトインデックスを組み合わせて使用できます。

間隔構文は次のとおりです。

interval[*multiple*][.*shift-index*]

詳細については、“必須引数” (772 ページ)を参照してください。

例

次のプログラムは、INTSHIFT 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl char(10) c;
  method run();
    c=intshift('year');
    put c;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
MONTH
```

```
proc ds2;
data test (overwrite=yes);
  dcl char(10) c;
  method run();
    c=intshift('dtyear');
    put c;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
DTMONTH
```

```
proc ds2;
data test (overwrite=yes);
  dcl char(10) c;
  method run();
    c=intshift('minute');
    put c;
  end;
```

```
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
DTMINUTE
```

```
proc ds2;
data test (overwrite=yes);
  dcl char(10) c x;
  method run();
    x='weekdays';
    c=intshift(x);
    put c;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
WEEKDAY
```

```
proc ds2;
data test (overwrite=yes);
  dcl char(10) c;
  method run();
    c=intshift('weekdays5.4');
    put c;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
WEEKDAY
```

```
proc ds2;
data test (overwrite=yes);
  dcl char(10) c;
  method run();
    c=intshift('qtr');
    put c;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
MONTH
```

```
proc ds2;
data test (overwrite=yes);
```

```
  dcl char(10) c;
  method run();
    c=intshift('dttenday');
    put c;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
DTTENDAY
```

INTTEST 関数

時間間隔が有効な場合は 1、時間間隔が無効な場合は 0 を返します。

カテゴリ:	日付と時間
制限事項:	DS2 は、カスタムの日付または時間間隔をサポートしていません。
返されるデータの種類:	DOUBLE

構文

INTTEST(*interval*[*multiple*][*.shift-index*])

必須引数

interval
文字列を評価するか、文字列に強制変換でき、WEEK、MONTH、QTR などの間隔名を含む文字定数、変数、または式を指定します。

データの種類	CHAR、NCHAR、NVARCHAR、VARCHAR
注	*interval* の可能な値は、“日付間隔と時間間隔について” (*SAS 出力形式と入力形式: リファレンス*)にリストされています。
ヒント	*Interval* は、大文字または小文字で表示できます。
例	YEAR は、年ベースの間隔を指定します。

オプション引数

multiple
乗数を指定します(省略可能)。基本タイプの間隔の期間に対する倍数と同等の間隔を設定します。

データの種類 DOUBLE

参照項目: 詳細については、"乗数とシフト間隔を使用した日時の増分" (*SAS 関数と CALL ルーチン: リファレンス*)を参照してください。

例 YEAR2 は、2 年、つまり 2 年間続く間隔タイプを指定します。

shift-index
シフトインデックスを指定します(省略可能)。間隔の開始時点を指定したサブ期間にシフトします。

制限事項 シフトインデックスは、間隔全体のサブ期間の数以下にする必要があります。たとえば、2 年間隔では 25 番目の月は存在しないため、YEAR2.24 は使用できますが YEAR2.25 はエラーになります。

デフォルトのシフト期間が間隔の種類と同じ場合、複数期間の間隔のみを任意のシフトインデックスでシフトできます。たとえば、MONTH の種類の間隔はデフォルトでは MONTH のサブ期間でシフトされるため、シフトインデックスで月間隔をシフトできません。ただし、各 MONTH2 間隔には 2 つの MONTH 間隔が含まれるため、シフトインデックスで 2 か月間隔をシフトできます。たとえば、間隔名 MONTH2.2 では、偶数月の 1 日目に開始する 2 か月の期間が指定されます。

データの種類 DOUBLE

参照項目: 詳細については、"乗数とシフト間隔を使用した日時の増分" (*SAS 関数と CALL ルーチン: リファレンス*)を参照してください。

例 YEAR.3 は、暦年の 3 月 1 日に開始し、翌年の 2 月に終了するようにシフトされた年次期間を指定します。

詳細

基本

INTTEST 関数は、有効な間隔名を確認します。この関数は、*multiple* と *shift-index* の有効な値を確認するときに役立ちます。乗数とシフトインデックスの詳細については、*SAS プログラマガイド: エッセンシャル*の"Multiunit Intervals"を参照してください。

INTTEST 関数は、小売業界によるカレンダーの間隔でも使用できます。これらの間隔は ISO 8601 に準拠します。詳細については、"日付、時間、および間隔" (*SAS プログラマガイド: エッセンシャル*)を参照してください。

間隔

間隔は、基本的なものから複雑なものまであります。基本間隔は、SAS が経過時間内にカウントできる測定単位(DAY、MONTH、HOUR など)です。より複雑な間隔を

指定するには、基本間隔名に乗数とシフトインデックスを組み合わせて使用できます。

間隔構文は次のとおりです。

interval[*multiple*][.*shift-index*]

詳細については、“必須引数” (776 ページ)を参照してください。

例

次のプログラムでは、SAS は *interval* 引数が有効な場合は 1、interval 引数が無効な場合は 0 を返します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double test1 test2 test3 test4 test5;
  dcl char var1;
  method run();
    test1 = inttest('month');
    put test1;

    test2 = inttest('week6.13');
    put test2;

    test3 = inttest('tenday');
    put test3;

    test4 = inttest('twoweeks');
    put test4;

    var1 = 'hour2.2';
    test5 = inttest(var1);
    put test5;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
1
1
1
0
1
```

INTTS 関数

TIMESTAMP 値に追加する秒数を指定します。

カテゴリ: 日付と時間

返されるデータの種類: TIMESTAMP

構文

INTTS(*expression*, *increment*)

必須引数

expression
TIMESTAMP 値を表す有効な式を指定します。

データの種類 TIMESTAMP

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

increment
時間に追加する秒数を表す負、正またはゼロの整数を指定します。

データの種類 DOUBLE

詳細

INTTS 関数は、指定した秒数だけ TIMESTAMP 値を増分します。

比較

INTNX 関数は、DOUBLE 値としてエンコードされた SAS 日付、時間、または日時の値を増分します。

例

次のプログラムは、INTTS 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl timestamp y z;
  method init();
    y= timestamp '2019-03-01 16:51:36.00';
    z=intts(y, 43);
    put y;
    put z;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
2019-03-01 16:51:36
2019-03-01 16:52:19
```

```
proc ds2;
data _null_;
  dcl timestamp y z;
  dcl date a;
  method init();
    a= date '2019-01-01';
    y= timestamp '2019-05-01 02:58:17.00';
    z=intts(y, -2500);
    put a;
    put y;
    put z;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
2019-01-01
2019-05-01 02:58:17
2019-05-01 02:16:37
```

関連項目

- “DS2 式” (*SAS DS2 プログラマガイド*)

関数:

- “INTCK 関数” (724 ページ)
- “INTDT 関数” (738 ページ)
- “INTNX 関数” (755 ページ)

INTZ 関数

ゼロファジーを使用して、引数の整数部を返します。

カテゴリ: 切り捨て

返されるデータの種類: DOUBLE

構文

INTZ(*expression*)

必須引数

expression
数値に評価される有効な式を指定します。

データの種類 DOUBLE

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

詳細

次の規則が適用されます。

- 引数の値が整数のみの場合、INTZ はその整数を返します。
- 引数が整数ではない正の値の場合、INTZ は引数より小さい最大の整数を返します。
- 引数が整数ではない負の値の場合、INTZ は引数より大きい最小の整数を返します。

比較

INT 関数とは異なり、INTZ 関数はゼロファジーを使用します。引数が整数の 1E-12 内にある場合、INT 関数はその整数に等しくなるように結果をファジー処理します。INTZ 関数は結果をファジー処理しません。そのため、INTZ 関数では予期しない結果になる可能性があります。

例

次のプログラムは、INTZ 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double var1 var2 var3 var4 a b c d ;
  method run();
    var1=2.1;
    a=intz(var1);
    put a;
    var2=-2.4;
    b=intz(var2);
    put b;
    var3=1+1.e-11;
    c=intz(var3);
    put c;
    var4=-1.6;
    d=intz(var4);
    put d;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
2
-2
1
-1
```

関連項目

関数:

- “CEIL 関数” (410 ページ)
- “CEILZ 関数” (412 ページ)
- “FLOOR 関数” (653 ページ)
- “FLOORZ 関数” (655 ページ)
- “INT 関数” (717 ページ)
- “MOD 関数” (859 ページ)
- “MODZ 関数” (861 ページ)
- “ROUND 関数” (1088 ページ)
- “ROUNDZ 関数” (1100 ページ)

IPMT 関数

将来の残高を達成するための、均等払いローンまたは定期預金に対する指定期間の利息の支払いを返します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

IPMT(*rate*, *period*, *number-of-periods*, *principal-amount*[, *future-amount*][, *type*])

必須引数

rate
支払期間ごとの利率を指定します。

データの種類 DOUBLE

period
利息の支払いを計算するための支払い期間を指定します。

要件 *period* は、*number-of-periods* の値以下の正の整数で指定する必要があります。

データの種類 INTEGER

number-of-periods
支払期間の数を指定します。

要件 *number-of-periods* は正の整数にする必要があります。

データの種類 INTEGER

principal-amount
ローンの元金を指定します。

データの種類 DOUBLE

注 欠損値が指定されている場合、ゼロとみなされます。

オプション引数

future-amount
将来の残高を指定します。

データの種類 DOUBLE

注 *future-amount* は、指定した支払い期間数終了後のローンの残高、または定期預金の将来の残高を指定できます。

0 を指定した場合、*future-amount* が省略されたか欠損値が指定されたとみなされます。

type
期首と期末のどちらで支払がなされるのかを指定します。0 は期末の支払を表し、1 は期首の支払を表します。

データの種類 INTEGER

注 *type* が省略された場合、または欠損値が指定されている場合は、ゼロとみなされます。

例

$8,000 のローンで名目年利が 10%、月単位の期末払いが 36 の場合、初回の定期的支払いでの利息の支払いは次のとおりです。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double interestpaid1;
  method run();
    InterestPaid1 = ipmt(0.1/12, 1, 36, 8000);
    put interestpaid1;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
66.6666666666666

66.66667
```

同じローンに期首払いがある場合、利息の支払いは次のように計算されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double interestpaid2;
  method run();
    InterestPaid2 = ipmt(0.1/12, 1, 36, 8000, 0, 1);
    put interestpaid2;
  end;
enddata;
run;
```

```
quit;
```

計算によって 0 という値が返されます。

- ```
  proc ds2;
  data test (overwrite=yes);
    dcl double interestpaid3;
    method run();
      InterestPaid3 = ipmt(0.1, 3, 3, 8000);
      put interestpaid3;
    end;
  enddata;
  run;
  quit;
  ```

  計算によって 292.447129909366 という値が返されます。

- ```
  proc ds2;
  data test (overwrite=yes);
    dcl double interestpaid4;
    method run();
      InterestPaid4 = ipmt(0.09/12, 359, 360, 125000, 0, 1);
      put interestpaid4;
    end;
  enddata;
  run;
  quit;
  ```

  計算によって 14.8075736630449 という値が返されます。

IQR 関数

四分位範囲を返します。

カテゴリ:	記述統計
返されるデータの種類:	DOUBLE

構文

IQR(*expression* [, ...*expression*])

必須引数

expression
数値に評価される有効な式を指定します。

データの種類	DOUBLE
参照項目:	“DS2 式” (*SAS DS2 プログラマガイド*)

詳細

すべての引数に null または欠損値がある場合、ANSI モードまたは SAS モードのどちらであるかに応じて、結果は null または欠損値になります。 詳細については、“DS2 による null および SAS 欠損値の処理方法” (*SAS DS2 プログラマガイド*)を参照してください。

それ以外の場合、結果は非 null または非欠損値の四分位範囲になります。四分位範囲の式は、UNIVARIATE プロシジャで使用される式と同じです。詳細については、*Base SAS プロシジャガイド: 統計プロシジャ*を参照してください。

例

次のプログラムは、IQR 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double a;
  method run();
    a=iqr(2,4,1,3,999999);
    put  'a= ' a;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
2
```

関連項目

関数:

- “MAD 関数” (836 ページ)
- “PCTL 関数” (916 ページ)

IRR 関数

内部利益率をパーセントで返します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

IRR(*freq*, *c1, c2* [*..., cn*])

必須引数

freq
目的の内部利益率に関連付けられた指定ベース期間での支払い数を示す数値です。

範囲 *freq* > 0。

データの種類 DOUBLE

ヒント *freq* = 0 は、連続複利計算を許可するフラグです。

c1, c2[*..., cn*]
任意の現金払いを示す数値です。

要件 最小で 2 つの現金払い値が必要です。

データの種類 DOUBLE

詳細

IRR 関数は、現金払いセット *c1*, *c2*, ..., *cn* の指定ベース期間での内部利益率を返します。2 つの連続した支払い間の時間間隔は同じとみなされます。引数 *freq* > 0 は、指定ベース期間で発生する支払い数を表します。各インスタンスから発行される情報の数は制限されます。

比較

IRR 関数は INTRR と似ていますが、IRR 関数では内部利益率がパーセントという点で異なります。

例

当初の支払いが$400 で、次の 3 年間の予定支払い額がそれぞれ$100、$200、$300 の場合、年間内部利益率はパーセントで次のプログラムで表されます。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double rate2;
  method run();
    rate2=irr(1,-400,100,200,300);
    put rate2;
  end;
enddata;
run;
quit;
```

返される値は 19.437709962747 です。

関連項目

関数:

- “INTRR 関数” (765 ページ)

JBESSEL 関数

ベッセル関数の値を返します。

カテゴリ: 数学

返されるデータの種類: DOUBLE

構文

JBESSEL(*nu*, *x*)

必須引数

nu
数値の定数、変数または式を指定します。

範囲 $nu \geq 0$

x
数値の定数、変数または式を指定します。

範囲 $x \geq 0$

詳細

JBESSEL 関数は、*x* で評価された次数 *nu* のベッセル関数の値を返します(詳細は Abramowitz and Stegun 1964 および Amos, Daniel, and Weston 1977 を参照)。

例

次のプログラムは、JBESSEL 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double x;
  method run();
    x=jbessel(2,2);
    put x;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
0.35283402861563
```

関連項目

関数:

- “IBESSEL 関数” (701 ページ)

JULDATE 関数

SAS 日付値からユリウス暦の日付を返します。

カテゴリ:	日付と時間
返されるデータの種類:	DOUBLE

構文

JULDATE(*date*)

必須引数

date
SAS 日付値を表す有効な式を指定します。

データの種類 DOUBLE

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

詳細

SAS 日付値は、1960 年 1 月 1 日から特定の日付までの日数を表す数値です。JULDATE 関数は、SAS 日付値をユリウス暦の日付に変換します。*date* がシステムオプション YEARCUTOFF=で定義された 100 年の期間内にある場合、結果は 3 桁、4 桁または 5 桁です。5 桁の結果では、最初の 2 桁は年を表し、残りの 3 桁は年間通算日(1 から 365、またはうるう年の場合は 1 から 366)を表します。先頭の 0 は結果から削除されるため、ユリウス暦の日付の年は省略されたり(末尾が 00 の年の場合)、1 桁のみになったり(年の末尾が 01 から 09 の場合)することがあります。それ以外の場合、結果は 7 桁になります。最初の 4 桁は年を表し、残りの 3 桁は年間通算日を表します。

00-09 で終わる年については、Z5.出力形式を使用して 5 桁のユリウス暦の日付をフォーマットできます。

DS2 が日付を処理する方法の詳細については、"DS2 式" (*SAS DS2 プログラマガイド*) を参照してください。

比較

関数 JULDATE7 は JULDATE に似ていますが、JULDATE7 は常に 4 桁の年を返す点で異なります。JULDATE7 は 2 桁の年を考慮する必要がなく、2000 年問題に対応しています。

例

次のプログラムは、JULDATE 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
```

```
data test(overwrite=yes);
  dcl double julian1 julian3 julian4;
  dcl double julian2 having format z5.;
  method run();
    julian1=juldate(mdy(12,31,2007));
    put julian1;
    julian2=juldate(mdy(12,31,2007));
    put julian2;
    julian3=juldate(mdy(9,1,1999));
    put julian3;
    julian4=juldate(mdy(7,1,1886));
    put julian4;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
7365
07365
99244
1886182
```

関連項目

関数:

- “DATEJUL 関数” (496 ページ)
- “JULDATE7 関数” (791 ページ)

JULDATE7 関数

SAS 日付値から 7 桁のユリウス暦の日付を返します。

カテゴリ:	日付と時間
返されるデータの種類:	DOUBLE

構文

JULDATE7(*date*)

必須引数

date
SAS 日付値を表す有効な式を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

SAS 日付値は、1960 年 1 月 1 日から特定の日付までの日数を表す数値です。JULDATE7 関数は、SAS 日付値から 7 桁のユリウス暦の日付を返します。最初の 4 桁は年を表し、残りの 3 桁は年間通算日を表します。

DS2 が日付を処理する方法の詳細については、“DS2 式” (*SAS DS2 プログラマガイド*)を参照してください。

比較

関数 JULDATE7 は JULDATE に似ていますが、JULDATE7 は常に 4 桁の年を返す点で異なります。JULDATE7 は 2 桁の年を考慮する必要がなく、2000 年問題に対応しています。

例

次のプログラムは、JULDATE7 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double julian7;
  method run();
    julian7=juldate7(mdy(12,31,1996));
    put julian7=;
    julian7=juldate7(mdy(1,1,2099));
    put julian7=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
julian7=1996366
julian7=2099001
```

関連項目

関数:

- “JULDATE 関数” (789 ページ)

KURTOSIS 関数

尖度を返します。

カテゴリ:	記述統計
返されるデータの種類:	DOUBLE

構文

KURTOSIS(*expression-1, expression-2, expression-3, expression-4* [, ...*expression-n*])

必須引数

expression
数値に評価される有効な式を指定します。

要件	少なくとも 4 つの非 null または非欠損の引数が必要です。それがない場合は、関数から null または欠損値が返されます。
データの種類	DOUBLE
参照項目:	“DS2 式” (*SAS DS2 プログラマガイド*)

詳細

尖度は、主に分布の裾の重さの尺度です。尖度の値が大きい場合は、分布の裾が重いことを示しています。

null 値と欠損値は無視され、計算には含まれません。

すべての非 null または非欠損引数の値が等しい場合、尖度は数学的に定義されておらず、KURTOSIS 関数は null または欠損値を返します。

例

次のプログラムは、KURTOSIS 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double x1 x2 x3 x4 x5;
  method run();
    x1=kurtosis(5, 9, 3, 6);
    x2=kurtosis(5, 8, 9, 6, .);
    x3=kurtosis(8, 9, 6, 1);
    x4=kurtosis(8, 1, 6, 1);
    x5=kurtosis(of x1-x4);
    put _all_;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
x1=0.92799999999999 x2=-3.3 x3=1.5 x4=-4.48337950138504 x5=-5.06569275379433 _N_=1
```

LAG 関数

キューから値を返します。

カテゴリ: 特殊

返されるデータの種類: 式のデータ型と同じ

構文

LAG [*n*] (*expression*)

必須引数

expression
任意の有効な式を指定します。

データの種類 ANY

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

オプション引数

n

ラグ値の数を指定します。

範囲	1-100
データの種類	Integer

詳細

基本

まだ長さが割り当てられていない文字変数に LAG 関数から値が返される場合、その変数には、デフォルトで、引数で使用した変数と同じ長さが割り当てられます。

LAG 関数、LAG1、LAG2、...、LAG*n* は、キューから値を返します。LAG1 は LAG と記述することもできます。LAG*n* 関数は、値をキューに格納し、*n* 番目の前の呼び出しによって提供された値を返します。プログラム内の各 LAG*n* 関数によって、値の独自のキューが生成されます。

各 LAG*n* の発生時のキューは、*n* 個の欠損値で初期化されます。*n* はキューの長さです(たとえば、LAG2 キューは 2 個の欠損値で初期化されます)。各 LAG*n* の発生時に、そのキューの先頭の値が削除されて返され、残りの値が前にシフトされ、引数の新しい値がキューの最後に配置されます。そのため、引数のラグ値が登場し始めると、各 LAG*n* の発生時の最初の *n* 回の実行で欠損値が返されます。

注: この関数が実行された場合にのみ、値がキューの最後に格納されてキューの先頭から値が返されます。条件付きで実行される LAG*n* 関数は、条件を満たすオブザベーションからのみの値を格納して返します。

ヒント　LAG*n* の引数が X[i]などの配列要素である場合、配列へのインデックスごとに個別のキューが保持されます。

LAG 関数のメモリ制限

LAG 関数がコンパイルされるときに、SAS はキューにメモリを割り当て、LAG 関数にリストされた変数の値を保持します。たとえば、関数 LAG100(x)の変数が 8 バイトの長さの数値の場合、必要なメモリは 8 の 100 倍の 800 バイトです。そのため、LAG 関数のメモリ制限は SAS が割り当てるメモリに基づき、動作環境によって異なります。

例

例 1: 2 つのラグ値を生成する

次のプログラムでは、各オブザベーションで 2 つのラグ値を生成します。

LAG1 は 1 つの欠損値と X の値を返します(1 回のラグ)。LAG2 は 2 つの欠損値と X の値を返します(2 回のラグ)。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data two (overwrite=yes);
  dcl char(8) x y z;
  method run();
    do x='abc', 'def','ghi','jkl','mno','pqr';
      y=lag1(x);
      z=lag2(x);
      output;
    end;
  end;
enddata;
run;
quit;

proc print data=two;
  title LAG Output;
run;
```

アウトプット 7.11　2 つのラグ値の生成からの出力

LAG Output

Obs	x	y	z
1	abc		
2	def	abc	
3	ghi	def	abc
4	jkl	ghi	def
5	mno	jkl	ghi
6	pqr	mno	jkl

例 2: フィボナッチ数列を生成する

次のプログラムでは、フィボナッチ数列を生成します。0 と 1 で開始し、前の 2 つのフィボナッチ数を追加して次のフィボナッチ数を生成します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl int n f;
  method run();
    put 'Fibonacci Sequence';
    n=1;
    f=1;
    put n= f=;
    do n=2 to 10;
      f=sum(f, lag(f));
      put n=f=;
    end;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
Fibonacci Sequence
n=1 f=1
n=2 f=1
n=3 f=2
n=4 f=3
n=5 f=5
n=6 f=8
n=7 f=13
n=8 f=21
n=9 f=34
n=10 f=55
```

例 3: LAG 関数の引数に式を使用する

次のプログラムでは、X、Y および Z の値を含むデータセットを作成する式を使用します。LAG は式の前の値をキューから取り出し、現在の値をキューに追加します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data one (overwrite=yes);
  declare int x y z;
  method run();
    do x=1 to 6;
      y=lag1(x+10);
      z=lag2(x);
      output;
    end;
  end;
enddata;
run;
quit;

proc print data=one;
  title LAG Output: Using an Expression;
run;
```

アウトプット 7.12　LAG 関数からの出力: 式の使用

LAG Output: Using an Expression

Obs	x	y	z
1	1	.	.
2	2	11	.
3	3	12	1
4	4	13	2
5	5	14	3
6	6	15	4

例 4: 配列要素のラグキューの生成

次のプログラムは、配列へのインデックスごとに個別のキューを生成します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double x[2];
  method init();
    dcl int i j k y;
    y=1;
    i=1;
    j=1;
    do while (y > 0 or missing(y));
      x[1]=i * 3;
      x[2]=i * 7;
      if i < 9 or i > 15 then
        k=1;
      else
      k=2;
      y=lag(x[k]);
      put i=y=;
      i=i + 1;
      j=1 - j;
      if (i > 20) then
        y=-1;
    end;
end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
i=1 y=
i=2 y=3
i=3 y=6
i=4 y=9
i=5 y=12
i=6 y=15
i=7 y=18
i=8 y=21
i=9 y=
i=10 y=63
i=11 y=70
i=12 y=77
i=13 y=84
i=14 y=91
i=15 y=98
i=16 y=24
i=17 y=48
i=18 y=51
i=19 y=54
i=20 y=57
```

関連項目

関数:

- “DIF 関数” (514 ページ)

LARGEST 関数

k 番目に大きい非 null または非欠損値を返します。

カテゴリ:	記述統計
返されるデータの種類:	DOUBLE
注:	2025.12 以降、この関数では、引数が DOUBLE の場合、すべての引数が DOUBLE に変換され、結果は DOUBLE になります。PROC DS2 ステートメントで BIGINTPROCESSING=YES が設定され、いずれかの引数が BIGINT の場合、すべての引数が BIGINT に変換され、結果は BIGINT になります。

構文

LARGEST(*k*, *expression* [, ...*expression*])

必須引数

k
数値に評価され、最大の戻り値を表す有効な式を指定します。たとえば、*k* が 2 の場合、LARGEST 関数は、式のリストから 2 番目に大きい値を返します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

expression
数値に評価される検索先の有効な式を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

k が null または欠損値、0 未満、または値の数よりも大きい場合、結果は null または欠損値になります。そうでない場合は、*k* が非 null または非欠損値の数よりも大きいと、結果は null または欠損値になります。

例

次のプログラムは、LARGEST 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double k largest1 largest2 largest3 largest4;
  method run();
    k=1;
    largest1=largest(k, 456, 789, .Q, 123);
    k=2;
    largest2=largest(k, 456, 789, .Q, 123);
    k=3;
    largest3=largest(k, 456, 789, .Q, 123);
    k=4;
    largest4=largest(k, 456, 789, .Q, 123);
    put largest1=;
    put largest2=;
    put largest3=;
    put largest4=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
largest1=789
largest2=456
largest3=123
largest4=.
```

関連項目

関数:

- “ORDINAL 関数” (914 ページ)
- “PCTL 関数” (916 ページ)
- “SMALLEST 関数” (1144 ページ)

LBOUND 関数

配列の下限を返します。

カテゴリ:	配列
返されるデータの種類:	INTEGER

構文

LBOUND(*array-name*[, *bound-n*])

必須引数

array-name
一時配列または変数配列の名前を指定します。

オプション引数

bound-n
多次元配列にある下限を確認する次元を指定する、数値定数、変数または式です。

bound-n 値が指定されていない場合、LBOUND 関数は配列の最初の次元の下限を返します。

bound-n は整数値に評価されます。

詳細

LBOUND 関数は、1 次元配列の下限、または多次元配列内の指定した次元の下限を返します。LBOUND と HBOUND を同時に使用して、配列のディメンションの下限と上限の値を返すことができます。

配列の次元外にある次元値を使用して LBOUND 関数が呼び出された場合、ランタイムエラーが発生し、関数は NULL 整数値を返します。

比較

- DIM は、配列次元の要素数を返します。
- HBOUND は、配列次元の上限の値を返します。
- LBOUND は、配列次元の下限の値を返します。
- NDIMS は、配列の次元数を返します。

例

例 1: 1 次元配列

次のプログラムは、1 次元配列に対して DIM、HBOUND、LBOUND、NDIMS 配列関数を使用する方法を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  method init();
    declare char(15) a1[4];
    declare double i a2[2,3,4] sum numelems j k;
    a1 := ('red' 'yellow' 'green' 'blue');
    a2 := (24*2.0);
    do i = 1 to dim(a1);
      put a1[i];
    end;
    numelems = 0;
    do i = 1 to ndims(a2);
      numelems = numelems + dim(a2, i);
    end;
    sum = 0;
    do i = lbound(a2, 1) to hbound(a2, 1);
     do j = lbound(a2, 2) to hbound(a2, 2);
       do k = lbound(a2, 3) to hbound(a2, 3);
         sum = sum + a2[i,j,k];
       end;
     end;
```

```
    end;
    put sum=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
red
yellow
green
blue
sum=48
```

例 2: 多次元配列

次のプログラムは、多次元配列に対して LBOUND 配列関数を使用する方法を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double x1 x2 x3;
  vararray double mult[2:6, 4:13, 2] mult1-mult100;
  method run();
    x1=LBOUND(MULT,1);
    x2=LBOUND(MULT,2);
    x3=LBOUND(MULT,3);
    put x1= x2= x3=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
x1=2 x2=4 x3=1
```

関連項目

関数:

- “DIM 関数” (518 ページ)
- “HBOUND 関数” (691 ページ)
- “NDIMS 関数” (868 ページ)

LCM 関数

数の集合に対する最小公倍数を返します。

カテゴリ:	数学
返されるデータの種類:	DOUBLE
注:	2025.12 以降、この関数では、引数が DOUBLE の場合、すべての引数が DOUBLE に変換され、結果は DOUBLE になります。PROC DS2 ステートメントで BIGINTPROCESSING=YES が設定され、いずれかの引数が BIGINT の場合、すべての引数が BIGINT に変換され、結果は BIGINT になります。

構文

LCM(*expression*-1, *expression*-2 [, ...*expression-n*])

必須引数

expression
整数に評価される有効な式を指定します。

要件	少なくとも 2 つの引数が必要です。
データの種類	DOUBLE
注	いずれかの式がゼロに評価されると、エラーが発生し、欠損値が返されます。
参照項目:	“DS2 式” (*SAS DS2 プログラマガイド*)

詳細

最小公倍数とは、2 つ以上の数を均等に割り切れる最小の数です。

例

次のプログラムは、LCM 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double x;
  method run();
    x=lcm(10, 15);
    put x;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
30
```

関連項目

関数:

- “GCD 関数” (671 ページ)

LCOMB 関数

COMB 関数の対数を計算します。これは、*n* 個のオブジェクトから一度に *r* 個を取り出した組み合わせ数の対数です。

カテゴリ:	組み合わせ
返されるデータの種類:	DOUBLE

構文

LCOMB(*n*, *r*)

必須引数

n
負でない整数で要素の合計数を表します。サンプルはこの中から選ばれます。

r
負でない整数で選ばれた要素の数を表します。

制限事項 $r \le n$

比較

LCOMB 関数は、COMB 関数の対数を計算します。

例

次のプログラムは、LCOMB 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double x y;
  method init();
    x=lcomb(5000,500);
    y=lcomb(100,10);
    put x= y=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
1621.44113611415
30.4823233622786
```

関連項目

関数:

- “COMB 関数” (427 ページ)

LEFT 関数

文字式を左揃えにします。

カテゴリ:	文字
返されるデータの種類:	CHAR、NCHAR、NVARCHAR、VARCHAR

構文

LEFT(*expression*)

必須引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

詳細

LEFT は、先頭の空白を値の末尾に移動した文字列を返します。

例

次のプログラムは、LEFT 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl varchar(12) a;
  method run();
    a=left(' END-OF-YEAR');
    put a;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
END-OF-YEAR
```

関連項目

関数:

- "COMPRESS 関数" (452 ページ)
- "RIGHT 関数" (1085 ページ)

- “STRIP 関数” (1166 ページ)
- “TRIM 関数” (1217 ページ)

LENGTH 関数

末尾の空白を除いた文字列の長さを文字数で返します。

カテゴリ:	文字
別名:	LENGTHN
返されるデータの種類:	BIGINT、INTEGER

構文

LENGTH(*expression*)

必須引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

LENGTH 関数は、*expression* 内で最も右にある空白以外の文字または数値の位置を表す整数を返します。*expression* の値が空白で埋められているか空の文字列の場合、LENGTH は値 0 を返します。*expression* が数値定数、変数、または式(初期化済みまたは未初期化)の場合、SAS では数値が自動的に右揃えの文字列に変換されます。

比較

- LENGTH 関数は末尾の空白を除いた文字列の長さを返すのに対し、LENGTHC 関数は末尾の空白を含む文字列の長さを返します。LENGTH は必ず LENGTHC が返す値以下の値を返します。
- LENGTH 関数が末尾の空白を除いた文字列の長さを文字数で返すのに対して、LENGTHM 関数は文字列に割り当てられているメモリ量をバイト単位で返します。LENGTH は必ず LENGTHM が返す値以下の値を返します。ただし、

expression 引数がある場合は、LENGHTM が常に null 値を返すのに対して、LENGTH は文字値の長さを文字数で返します。

例

次のプログラムは、LENGTH 関数を示しています。**g** のデータ型は、値が 20943 で長さが 5 の NCHAR に変換されます。**d** および **e** のデータ型である null または欠損値(.)は、DOUBLE から NCHAR に変換され、値は.になります。その長さは 1 です。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double c d e f g;
  method run();
    c=length('ABDCEF    ');
    d=length('   ');
    e=length(.);
    g=date();
    f=length(g);
    put c;
    put d;
    put e;
    put f;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
6
0
1
5
```

関連項目

関数:

- “LENGTHC 関数” (810 ページ)
- “LENGTHM 関数” (813 ページ)

LENGTHC 関数

末尾の空白を含めた文字列の長さを文字数で返します。

カテゴリ: 文字

返されるデータの種類: BIGINT、INTEGER

構文

LENGTHC(*expression*)

必須引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

詳細

固定長変数の場合、LENGTHC は、末尾の空白を含む文字列の文字長を返します。*expression* が固定長変数の場合、LENGTHC が返す値は、宣言された変数の長さと同じになります。*expression* が可変長変数の場合、LENGTHC が返す値は、宣言された変数の長さ以下になります。*expression* の値が欠損していて空白が含まれる場合、LENGTHC は *expression* の空白の数を返します。*expression* が数値式(初期化済みまたは未初期化)の場合、SAS では数値が自動的に右揃えの文字列に変換されます。

比較

- LENGTHC 関数が末尾の空白を含む文字列の長さを返すのに対し、LENGTH 関数は末尾の空白を除いた文字列の長さを返します。LENGTHC は常に LENGTH が返す値以上の値を返します。
- LENGTHC 関数は末尾の空白を含む文字列の長さを返しますが、LENGTHM 関数は文字列に割り当てられたメモリの量(バイト単位)を返します。固定長の文字列の場合、LENGTHC と LENGTHM は常に同じ値を返します。可変長の文字列の場合、LENGTHC は常に LENGTHM で返される値以下の値を返します。ただし、expression 引数がある場合は、LENGHTM が常に null 値を返すのに対して、LENGTHC は文字値の長さを文字数で返します。

例

例 1: LENGTHC が文字列の実際の長さを返す

次のプログラムは、LENGTHC 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double c;
  method run();
    c=lengthc('string with trailing blanks     ');
    put c;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
32
```

例 2: 可変長文字列を使用する LENGTHC

次のプログラムは、LENGTHC 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test2 (overwrite=yes);
  dcl varchar(25) string;
  dcl double a;
  method run();
    string='The power to know.  ';
    a=lengthc(string);
    put a;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
20
```

例 3: 固定長文字列を使用する LENGTHC

次のプログラムは、LENGTHC 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test2 (overwrite=yes);
  dcl char(25) string;
  dcl double b;
  method run();
    string='The power to know.  ';
    b=lengthc(string);
    put b;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
25
```

例 4: 長さゼロの文字列を使用する LENGTHC

次のプログラムは、LENGTHC 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test2 (overwrite=yes);
  dcl double d;
  method run();
    d=lengthc('');
    put d;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
0
```

例 5: 空白文字列を使用する LENGTHC

次のプログラムは、LENGTHC 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test2 (overwrite=yes);
  dcl double d;
  method run();
    d=lengthc(' ');
```

```
    put d;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
1
```

関連項目

関数:

- “LENGTH 関数” (808 ページ)
- “LENGTHM 関数” (813 ページ)

LENGTHM 関数

文字列に割り当て可能なメモリの量をバイト単位で返します。

カテゴリ:	文字
返されるデータの種類:	BIGINT、INTEGER

構文

LENGTHM(*variable*)

必須引数

variable
有効な文字列変数を指定します。文字列値を評価するか文字列値に強制変換できる式は無効な引数です。

制限事項	変数引数のみが許可されます。式とリテラルは無効であり、null バイト長を返します。
データの種類	CHAR、NCHAR、NVARCHAR、VARCHAR

詳細

LENGTHM 関数は、文字変数に割り当て可能なメモリの最大量をバイト単位で返します。これは、NATIONAL、VARYING、CHARACTER SET などの文字データ型修飾子に関係なく当てはまります。

LENGTHM 関数によって返される値は、変数に保存されている現在の値とは無関係であり、現在割り当てられている変数のバイト長よりも大きい場合があります。これは、固定長文字変数と可変長文字変数の両方に当てはまります。たとえば、CHAR(100) CHARACTER SET UTF-8 変数または VARCHAR(100) CHARACTER SET UTF-8 変数がある場合、LENGTHM 関数は、変数に保存されている値に関係なく 400 バイトを返します。

リテラルまたは式の場合、LENGTHM 関数によって返される値は null バイト長であり、引数が関数に対して無効であるという警告が発行されます。

比較

LENGTHM 関数が文字列に割り当てられたメモリ量をバイト単位で返すのに対して、LENGTH および LENGTHC 関数は文字値の長さを文字数で返します。variable 引数がある場合は、LENGTHM は常に LENGTH および LENGTHC によって返される値以上の値を返します。expression 引数がある場合は、LENGHTM が常に null 値を返すのに対して、LENGTH および LENGTHC は文字値の長さを文字数で返します。

例

例 1: Latin1 文字セットを使用した LENGHTM

次のプログラムは、LENGTHM 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl char(25) character set latin1 string;
  dcl double a;
  method run();
    string='The power to know.   ';
    a=lengthm(string);
    put a;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
25
```

例 2: UTF-8 文字セットを使用した LENGTHM

次のプログラムは、LENGTHM 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl varchar(15) character set utf8 a;
  dcl double b;
  method run();
    a='ABCDEF    ';
    b=lengthm(a);
    put b;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
60
```

関連項目

関数:

- “LENGTH 関数” (808 ページ)
- “LENGTHC 関数” (810 ページ)

LFACT 関数

FACT(階乗)関数の対数を計算します。

カテゴリ:	組み合わせ
返されるデータの種類:	DOUBLE

構文

LFACT(*n*)

必須引数

n
整数で要素の合計数を表します。サンプルはこの中から選ばれます。

詳細

LFACT 関数は、FACT 関数の対数を計算します。

例

次のプログラムは、LFACT 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double x y ;
  method init();
    x=lfact(5000);
    y=lfact(100);
    put x=  y=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
x=37591.1435088767 y=363.739375555563
```

関連項目

関数:

- “FACT 関数” (532 ページ)

LGAMMA 関数

ガンマ関数の自然対数を返します。

カテゴリ: 数学

返されるデータの種類: DOUBLE

構文

LGAMMA(*expression*)

必須引数

expression
数値に評価される有効な式を指定します。

要件 正数にする必要があります。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

例

次のプログラムは、LGAMMA 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double a;
  method run();
    a=lgamma(2);
    put a=;
    a=lgamma(1.5);
    put a=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
a=0
a=-0.12078223763524
```

LOG 関数

数値式の自然対数(底 e)を返します。

カテゴリ: 数学

返されるデータの種類: DOUBLE

構文

LOG(*expression*)

必須引数

expression
数値に評価される有効な式を指定します。

要件 正数にする必要があります。

データの種類 DOUBLE

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

例

次のプログラムは、LOG 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double a b;
  method run();
    a=log(1.0);
    b=log(10.0);
    put a=;
    put b=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
a=0
b=2.30258509299404
```

関連項目

関数:

- “LOG10 関数” (819 ページ)
- “LOG2 関数” (822 ページ)

LOG10 関数

数値式の 10 を底とする対数を返します。

カテゴリ:	数学
返されるデータの種類:	DOUBLE

構文

LOG10(*expression*)

必須引数

expression
数値に評価される有効な式を指定します。

要件	正数にする必要があります。
データの種類	DOUBLE
参照項目:	“DS2 式” (*SAS DS2 プログラマガイド*)

例

次のプログラムは、LOG10 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
```

```
  dcl double a b;
  method run();
    a=log10(1.0);
    b=log10(10.0);
    put a=;
    put b=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
a=0
b=1
```

関連項目

関数:

- “LOG 関数” (818 ページ)
- “LOG2 関数” (822 ページ)

LOG1PX 関数

1 に引数を加えた値の対数を返します。

カテゴリ:	数学
返されるデータの種類:	DOUBLE

構文

LOG1PX(*x*)

必須引数

x
数値定数、変数または式を指定します。

データの種類 DOUBLE

詳細

LOG1PX 関数は、1 に引数を加えた値の対数を計算します。LOG1PX 関数は、次の式(–1 < x の場合)で数学的に定義されます。

$$LOG1PX(x) = log(1 + x)$$

x が 0 に近い場合、LOG1PX(x)は LOG(1+x)より精度が高くなることがあります。

例

例 1: LOG1PX 関数を使用した対数の計算

次のプログラムでは、1 に値 0.5 を加えた値の対数を計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double x;
  method run();
    x=log1px(0.5);
    put x=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
x=0.40546510810816
```

例 2: LOG1PX 関数と LOG 関数の比較

次のプログラムでは、LOG1PX 関数を使用して値 X を計算します。値 Y は LOG 関数を使用して計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double x y;
  method run();
    x=log1px(1.e-5);
    put x= hex16.;
    y=log(1+1.e-5);
    put y= hex16.;
  end;
```

```
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
x=3EE4F8AEA9AE7317
y=3EE4F8AEA9AF0A25
```

関連項目

関数:

- “LOG 関数” (818 ページ)

LOG2 関数

数値式の 2 を底とする対数を返します。

カテゴリ:	数学
返されるデータの種類:	DOUBLE

構文

LOG2(*expression*)

必須引数

expression
数値に評価される有効な式を指定します。

要件	正数にする必要があります。
データの種類	DOUBLE
参照項目:	“DS2 式” (*SAS DS2 プログラマガイド*)

例

次のプログラムは、LOG2 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double a;
  method run();
    a=log2(8.0);
    put a=;
    a=log2(4);
    put a=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
a=3
a=2
```

関連項目

関数:

- “LOG 関数” (818 ページ)
- “LOG10 関数” (819 ページ)

LOGBETA 関数

ベータ関数の対数を返します。

カテゴリ:	数学
返されるデータの種類:	DOUBLE

構文

LOGBETA(*a*, *b*)

必須引数

a
第 1 形状パラメーターです(*a*>0)。

データの種類 DOUBLE

b
第 2 形状パラメーターです(*b*>0)。

データの種類 DOUBLE

詳細

LOGBETA 関数は、式で数学的に求められます。

$$log(\beta(a,b)) = log(\Gamma(a)) + log(\Gamma(b)) - log(\Gamma(a+b))$$

この式で、$\Gamma(.)$はガンマ関数です。

式が計算できない場合、LOGBETA は欠損値を返します。

例

次の DS2 プログラムは、ベータ関数の対数を計算します。第 1 形状パラメーターは 5 で、第 2 形状パラメーターは 3 です。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double y;
  method run();
    y=logbeta(5,3);
    put y=;
  end;
enddata;
run;
quit;
```

次の行が SAS ログに書き込まれます。

```
y=-4.65396035015752
```

関連項目

関数:

- “BETA 関数” (320 ページ)

LOGCDF 関数

左累積分布関数の対数を返します。

カテゴリ: 確率

参照項目: “CDF 関数” (359 ページ)

構文

LOGCDF('*distribution*', *quantile* [, *parameter-1*, ..., *parameter-k*])

必須引数

distribution
分布を特定する文字定数、変数または式です。

注: 各 LOGCDF 分布関数の引数は、対応する CDF 分布関数の引数と同じです。

有効な分布は次のとおりです。

分布	引数
Bernoulli	'BERNOULLI'
ベータ	'BETA'
二項	'BINOMIAL'
Cauchy	'CAUCHY'
カイ 2 乗	'CHISQUARE'
Conway-Maxwell-Poisson	'CONMAXPOI'
指数	'EXPONENTIAL'
F	'F'
ガンマ	'GAMMA'
一般化 Poisson	'GENPOISSON'
幾何	'GEOMETRIC'

分布	引数
超幾何	'HYPERGEOMETRIC'
Laplace	'LAPLACE'
ロジスティック	'LOGISTIC'
対数正規	'LOGNORMAL'
負の二項	'NEGBINOMIAL'
正規	'NORMAL'|'GAUSS'
正規混合	'NORMALMIX'
パレート	'PARETO'
Poisson	'POISSON'
T	'T'
Tweedie	'TWEEDIE'
一様	'UNIFORM'
Wald (逆ガウス)	'WALD'|'IGAUSS'
Weibull	'WEIBULL'

注: T、F および NORMALMIX を除き、最初の 4 文字で分布を最小限に識別できます。

quantile
確率変数の値を指定する、数値変数、定数または式です。

データの種類 DOUBLE

オプション引数

parameter-1, ..., parameter-k
特定の分布に適した任意の *shape*、*location* または *scale* パラメーターです。

データの種類 DOUBLE

詳細

LOGCDF 関数は、さまざまな連続分布および離散分布から左累積分布関数の対数(左側の対数)を計算します。詳細については、前述の表の個々の分布を参照してください。

表に示された分布の詳細については、“CDF 関数” (359 ページ)を参照してください。

関連項目

関数:

- “CDF 関数” (359 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “PDF 関数” (918 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

LOGISTIC 関数

引数のロジスティック変換を返します。

カテゴリ: 数学

構文

LOGISTIC(*argument*)

必須引数

argument
数値の確率変数の値を指定する、数値変数、定数または式です。*argument* が欠損値または null 値の場合、LOGISTIC 関数は欠損値または null 値を返します。

詳細

LOGISTIC 関数は引数のロジスティック変換を返します。通常は、対数オッズ値を確率尺度の値に変換するために使用されます。この関数は、数学的に次の式で表現されます。

$$logistic=\frac{e^x}{1+e^x}$$

引数に欠損値が含まれている場合、LOGISTIC 関数は欠損値または null 値を返します。

例

次のプログラムは、LOGISTIC 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double a b;
  method run();
    a=logistic(.5);
    b=logistic(7.3);
    put  'a= ' a;
    put 'b=' b;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
a=  0.62245933120185
b= 0.99932491726936
```

LOGPDF 関数

さまざまな連続分布および離散分布から確率密度(質量)関数の対数を計算します。

カテゴリ: 確率

別名: LOGPMF

参照項目: “PDF 関数” (918 ページ)

構文

LOGPDF('*distribution*', *quantile*, *parameter-1, ..., parameter-k*)

必須引数

distribution
分布を特定する文字定数、変数または式です。

注: 各 LOGPDF 分布関数の引数は、対応する PDF 分布関数の引数と同じです。

有効な分布は次のとおりです。

分布	引数
Bernoulli	'BERNOULLI'
ベータ	'BETA'
二項	'BINOMIAL'
Cauchy	'CAUCHY'
カイ 2 乗	'CHISQUARE'
Conway-Maxwell-Poisson	'CONMAXPOI'
指数	'EXPONENTIAL'
F	'F'
ガンマ	'GAMMA'
一般化 Poisson	'GENPOISSON'
幾何	'GEOMETRIC'
超幾何	'HYPERGEOMETRIC'
Laplace	'LAPLACE'
ロジスティック	'LOGISTIC'
対数正規	'LOGNORMAL'
負の二項	'NEGBINOMIAL'
正規	'NORMAL'|'GAUSS'

分布	引数
正規混合	'NORMALMIX'
パレート	'PARETO'
Poisson	'POISSON'
T	'T'
Tweedie	'TWEEDIE'
一様	'UNIFORM'
Wald (逆ガウス)	'WALD'|'IGAUSS'
Weibull	'WEIBULL'

注: T、F および NORMALMIX を除き、最初の 4 文字で分布を最小限に識別できます。

quantile
確率変数の値を指定する、数値定数、変数または式です。

データの種類 DOUBLE

parameter-1, ..., parameter-k
特定の分布に適した任意の *shape*、*location* または *scale* パラメーターです。

データの種類 DOUBLE

詳細

LOGPDF 関数は、さまざまな連続分布および離散分布から確率密度(質量)関数の対数を計算します。詳細については、前述の表の個々の分布を参照してください。

表に示された分布の詳細については、“PDF 関数” (918 ページ)を参照してください。

関連項目

関数:

- “CDF 関数” (359 ページ)
- “LOGCDF 関数” (825 ページ)

- “LOGSDF 関数” (831 ページ)
- “PDF 関数” (918 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

LOGSDF 関数

生存関数の対数を返します。

カテゴリ: 確率

参照項目: “SDF 関数” (1120 ページ)

構文

LOGSDF('*distribution*', *quantile*, *parameter-1, ..., parameter-k*)

必須引数

distribution
分布を特定する文字定数、変数または式です。

注: 各 LOGSDF 分布関数の引数は、対応する CDF 分布関数の引数と同じです。

有効な分布は次のとおりです。

分布	引数
Bernoulli	'BERNOULLI'
ベータ	'BETA'
二項	'BINOMIAL'
Cauchy	'CAUCHY'
カイ 2 乗	'CHISQUARE'
Conway-Maxwell-Poisson	'CONMAXPOI'
指数	'EXPONENTIAL'

分布	引数
F	'F'
ガンマ	'GAMMA'
一般化 Poisson	'GENPOISSON'
幾何	'GEOMETRIC'
超幾何	'HYPERGEOMETRIC'
Laplace	'LAPLACE'
ロジスティック	'LOGISTIC'
対数正規	'LOGNORMAL'
負の二項	'NEGBINOMIAL'
正規	'NORMAL'|'GAUSS'
正規混合	'NORMALMIX'
パレート	'PARETO'
Poisson	'POISSON'
T	'T'
Tweedie	'TWEEDIE'
一様	'UNIFORM'
Wald (逆ガウス)	'WALD'|'IGAUSS'
Weibull	'WEIBULL'

注: T、F および NORMALMIX を除き、最初の 4 文字で分布を最小限に識別できます。

quantile
確率変数の値を指定する、数値定数、変数または式です。

データの種類 DOUBLE

parameter-1, ..., parameter-k
特定の分布に適した任意の *shape*、*location* または *scale* パラメーターです。

データの種類 DOUBLE

詳細

LOGSDF 関数は、さまざまな連続分布および離散分布から生存関数の対数を計算します。詳細については、“SDF 関数” (1120 ページ)を参照してください。

表に示された分布の詳細については、“CDF 関数” (359 ページ)を参照してください。

関連項目

関数:

- “CDF 関数” (359 ページ)
- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “PDF 関数” (918 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

LOWCASE 関数

文字式のすべての文字を小文字に変換します。

カテゴリ:	文字
別名:	LOWER
返されるデータの種類:	CHAR、NCHAR、NVARCHAR、VARCHAR

構文

LOWCASE(*expression*)

必須引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

要件	リテラル文字表現は、単一引用符で囲む必要があります。
データの種類	CHAR、NCHAR、NVARCHAR、VARCHAR
参照項目:	“DS2 式” (*SAS DS2 プログラマガイド*)

詳細

LOWCASE 関数は、文字式をコピーし、すべての大文字を小文字に変換して、変更された値を結果として返します。

比較

UPCASE 関数は、引数内のすべての文字を大文字に変換します。LOWCASE 関数は、引数内のすべての文字を小文字に変換します。

例

次のプログラムは、LOWCASE 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl char(12) x;
  method run();
    x=lowcase('INTRODUCTION');
    put x;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
introduction
```

関連項目

関数:

- “UPCASE 関数” (1224 ページ)

LPERM 関数

n 個のオブジェクトの順列数の対数を計算し、r 個の要素を含めることができます。

カテゴリ: 組み合わせ

返されるデータの種類: DOUBLE

構文

LPERM (*n* [, *r*])

必須引数

n
選択するサンプルの要素の合計数を表します。

データの種類 INTEGER

オプション引数

r
選択された要素の数を表します。*r* を省略すると、関数は *n* の階乗を返します。

制限事項 *n*<=*r*

データの種類 INTEGER

詳細

LPERM 関数は、PERM 関数の対数を計算します。

例

次のプログラムは、LPERM 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
```

```
  dcl double x y;
  method run();
    x=lperm(5000,500);
    y=lperm(100,10);
    put x= y=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
x=4232.77159457431 y=45.5867359353541
```

関連項目

関数:

- “PERM 関数” (968 ページ)

MAD 関数

中央値からの中央絶対偏差を返します。

カテゴリ:	記述統計
返されるデータの種類:	DOUBLE

構文

MAD(*expression-1*[, ...*expression-n*])

必須引数

expression
中央値からの中央絶対偏差を計算する数値に評価される有効な式を指定します。

データの種類 DOUBLE

詳細

すべての引数が欠損値または null 値の場合、結果は欠損値または null 値になります。それ以外の場合は、結果は非欠損値または非 null 値の中央値からの中央絶対偏

差になります。中央値の計算式は、UNIVARIATE プロシジャで使用するものと同じです。詳細については、Base SAS プロシジャガイド: 統計プロシジャを参照してください。

例

次のプログラムは、MAD 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double c;
  method run();
    c=mad(2, 4, 1, 3, 5, 999999);
    put c;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
1.5
```

関連項目

関数:

- “IQR 関数” (785 ページ)
- “MEDIAN 関数” (850 ページ)
- “PCTL 関数” (916 ページ)

MARGRCLPRC 関数

Margrabe モデルに基づいて、株式のヨーロピアンオプションのコール価格を計算します。

カテゴリ:	財務
返されるデータの種類:	DOUBLE

構文

MARGRCLPRC(*X_1*, *t*, *X_2*, *sigma1*, *sigma2*, *rho12*)

必須引数

X_1
第 1 資産の価格を指定する正の非欠損値です。

要件　同じ単位で X_1 と X_2 を指定します。

データの種類　DOUBLE

t
有効期限までの時間を年単位で指定する非欠損値です。

データの種類　DOUBLE

X_2
第 2 資産の価格を指定する正の非欠損値です。

要件　同じ単位で X_2 と X_1 を指定します。

データの種類　DOUBLE

sigma1
第 1 資産のボラティリティを指定する非欠損の正の分数です。

データの種類　DOUBLE

sigma2
第 2 資産のボラティリティを指定する非欠損の正の分数です。

データの種類　DOUBLE

rho12
第 1 資産と第 2 資産の間の相関を指定します。詳細については、*SAS DS2 言語リファレンス*を参照してください。

第 1 資産と第 2 資産の間の相関$\rho_{x_1x_2}$を指定します。

範囲　-1 から 1 まで

データの種類　DOUBLE

詳細

MARGRCLPRC 関数は、Margrabe モデルに基づいて、株式のヨーロピアンオプションのコール価格を計算します。この関数は次の関係に基づきます。

$$\text{CALL} = X_1 N(d_1) - X_2 N(d_2)$$

引数

X_1
第 1 資産の価格を指定します。

X_2
第 2 資産の価格を指定します。

N
累積正規密度関数を指定します。

$$d_1 = \frac{\left(\ln\left(\frac{N_1}{N_2}\right) + \left(\frac{\sigma^2}{2}\right)t\right)}{\sigma\sqrt{t}}$$

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

$$\sigma^2 = \sigma_{x_1}^2 + \sigma_{x_2}^2 - 2\rho_{x_1,x_2}\sigma_{x_1}\sigma_{x_2}$$

前述の式には次の引数が適用されます。

t
有効期限までの時間を指定します。

$\sigma_{x_1}^2$
第 1 資産の分散を指定します。

$\sigma_{x_2}^2$
第 2 資産の分散を指定します。

σ_{x_1}
第 1 資産のボラティリティを指定します。

σ_{x_2}
第 2 資産のボラティリティを指定します。

ρ_{x_1, x_2}
第 1 資産と第 2 資産の間の相関を指定します。

t=0 となる特別な場合には、次の式が真です。

$$\text{CALL} = \max((X_1 - X_2), 0)$$

注: この関数は、2 つの資産から利益配当金がないことを想定しています。

価格設定の基本については、*"価格関数の使用" (SAS 関数と CALL ルーチン: リファレンス)*を参照してください。

比較

MARGRCLPRC 関数は、Margrabe モデルに基づいて、株式のヨーロピアンオプションのコール価格を計算します。MARGRCLPRC 関数は、Margrabe モデルに基づいて、株式のヨーロピアンオプションのプット価格を計算します。これらの関数はスカラー値を返します。

例

次のプログラムは、MARGRCLPRC 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double a b;
  method run();
    a=margrclprc(15, .5, 13, .06, .05, 1);
    put a;
    b=margrclprc(2, .25, 1, .3, .2, 1);
    put b;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
2
1
```

関連項目

関数:

- “MARGRPTPRC 関数” (840 ページ)

MARGRPTPRC 関数

Margrabe モデルに基づいて、株式のヨーロピアンオプションのプット価格を計算します。

カテゴリ:	財務
返されるデータの種類:	DOUBLE

構文

MARGRPTPRC(*X_1*, *t*, *X_2*, *sigma1*, *sigma2*, *rho12*)

必須引数

X_1
第 1 資産の価格を指定する正の非欠損値です。

要件 同じ単位で X_1 と X_2 を指定します。

データの種類 DOUBLE

t
有効期限までの時間を年単位で指定する非欠損値です。

データの種類 DOUBLE

X_2
第 2 資産の価格を指定する正の非欠損値です。

要件 同じ単位で X_2 と X_1 を指定します。

データの種類 DOUBLE

sigma1
第 1 資産のボラティリティを指定する非欠損の正の分数です。

データの種類 DOUBLE

sigma2
第 2 資産のボラティリティを指定する非欠損の正の分数です。

データの種類 DOUBLE

rho12
第 1 資産と第 2 資産の間の相関を指定します。詳細については、*SAS DS2 言語リファレンス*を参照してください。

第 1 資産と第 2 資産の間の相関$\rho_{x_1y_1}$を指定します。

範囲 -1 から 1 まで

データの種類 DOUBLE

詳細

MARGRCLPRC 関数は、Margrabe モデルに基づいて、株式のヨーロピアンオプションのプット価格を計算します。この関数は次の関係に基づきます。

$$\text{PUT} = X_2 N(pd_1) - X_1 N(pd_2)$$

引数

X_1
第 1 資産の価格を指定します。

X_2
第 2 資産の価格を指定します。

N
累積正規密度関数を指定します。

$$pd_1 = \frac{\left(\ln\left(\frac{N_1}{N_2}\right) + \left(\frac{\sigma^2}{2}\right)t\right)}{\sigma\sqrt{t}}$$

$$pd_2 = pd_1 - \sigma\sqrt{t}$$

$$\sigma^2 = \sigma_{x1}^2 + \sigma_{x2}^2 - 2\rho_{x1,\,x2}\sigma_{x1}\sigma_{x2}$$

前述の式には次の引数が適用されます。

t
有効期限までの時間を年単位で指定する非欠損値です。

σ_{x1}^2
第 1 資産の分散を指定します。

σ_{x2}^2
第 2 資産の分散を指定します。

σ_{x1}
第 1 資産のボラティリティを指定します。

σ_{x2}
第 2 資産のボラティリティを指定します。

$\rho_{x1,\,x2}$
第 1 資産と第 2 資産の間の相関を指定します。

対応する CALL 関係を表示するには、“MARGRCLPRC 関数” (837 ページ)を参照してください。

t=0 となる特別な場合には、次の式が真です。

$$\text{PUT} = \max((X_2 - X_1), 0)$$

注: この関数は、2 つの資産から利益配当金がないことを想定しています。

価格設定の基本については、“価格関数の使用” (*SAS 関数と CALL ルーチン: リファレンス*)を参照してください。

比較

MARGRCLPRC 関数は、Margrabe モデルに基づいて、株式のヨーロピアンオプションのプット価格を計算します。MARGRCLPRC 関数は、Margrabe モデルに基づいて、株式のヨーロピアンオプションのコール価格を計算します。これらの関数はスカラー値を返します。

例

次のプログラムは、MARGRPTPRC 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double a b;
  method run();
    a=margrptprc(2, .25, 3, .06, .2, 1);
    put a;
    b=margrptprc(3, .25, 4, .05, .3, 1);
    put b;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
1.00000000009729
1.00157624907711
```

関連項目

関数:

- “MARGRCLPRC 関数” (837 ページ)

MAX 関数

引数のリストから最大値を返します。

カテゴリ:	記述統計
返されるデータの種類:	BIGINT、DECIMAL、DOUBLE
注:	2025.12 以降、この関数では、引数が DOUBLE の場合、すべての引数が DOUBLE に変換され、結果は DOUBLE になります。PROC DS2 ステートメントで BIGINTPROCESSING=YES が設定され、いずれかの引数が BIGINT の場合、すべての引数が BIGINT に変換され、結果は BIGINT になります。

構文

MAX(*expression-1, expression-2* [, ...*expression-n*])

必須引数

expression
数値に評価される有効な式です。

要件	少なくとも 2 つの引数が必要です。
データの種類	BIGINT、DECIMAL、DOUBLE
参照項目:	“DS2 式” (*SAS DS2 プログラマガイド*)

詳細

この関数の引数が数値以外の場合、その引数は DOUBLE に変換されます。いずれかの引数が DOUBLE または REAL の場合、すべての引数が(まだ変換されていなければ) DOUBLE に変換され、結果は DOUBLE になります。それ以外の場合で、いずれかの引数が DECIMAL のときは、すべての引数が(まだ変換されていなければ) DECIMAL に変換され、結果は DECIMAL になります。それ以外の場合は、すべての引数が BIGINT に変換され、結果は BIGINT になります。

比較

MAX 関数は、引数のリストから最大値を返します。MAX 演算子(<>)は、2 つのオペランドのうち大きい方を返します。

MAX 関数は、すべての引数が null または欠損の場合にのみ、null 値または欠損値を返します。MAX 演算子(<>)は、両方のオペランドが null または欠損の場合にのみ、null 値または欠損値を返します。この場合、null 値または欠損値の並べ替え順がより高いオペランドの値が返されます。

例

次のプログラムは、MAX 関数を示しています。

```
proc ds2;
data test(overwrite=yes);
  dcl double x;
  method run();
    x=max(8, 3);
    put x;
    x=max(2, 6, .);
    put x;
    x=max(2, -3, 1, -1);
```

```
    put x;
    x=max(3, ., -3);
    put x;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
8
6
2
3
```

関連項目

関数:

- “MIN 関数” (852 ページ)

MD5 関数

指定した文字列のメッセージダイジェストの結果をバイナリ形式で返します。

カテゴリ:	文字
制限事項:	文字引数を使用している場合、MD5 は NATIONAL 文字セット(NCHAR)を含む任意の文字セットの CHAR を受け入れます。
返されるデータの種類:	BINARY

構文

MD5(*string*)

必須引数

string
文字定数、変数または式を指定します。

データの種類	BINARY、CHAR
ヒント	文字のリテラル文字列は単一引用符で囲みます。

スカラー文字変数引数の場合、変数が MD5 関数に渡される前に、DECLARE ステートメントで指定された初期文字セットエンコーディングを使用して変数がトランスコードされます。バイナリ引数の場合、バイナリ値は文字列として扱われ、セッションエンコーディングを使用して、MD5 関数に渡される値がトランスコードされます。

詳細

基本

MD5 関数は、MD5 アルゴリズムに基づいて文字列を 128 ビットのハッシュ値に変換します。このハッシュ値は*メッセージダイジェスト*(デジタル署名)と呼ばれ、関数に渡される各文字列に対してほぼ重複することはありません。

MD5 関数では、出力形式は適用されません。$BINARY*w*.出力形式または$HEX*w*.出力形式を使用して、読み取り可能な結果を表示します。

メッセージダイジェストアルゴリズム

メッセージダイジェストは、任意の長いバイナリデータストリームを操作および圧縮した結果です。理想的なメッセージダイジェストアルゴリズムでは、2 つの異なる入力セットに対し同じ結果は生成されません。ただし、このような重複しない結果を生成するには、入力自体と同じぐらいの長さのメッセージダイジェストが必要になります。したがって MD5P は、ほぼ重複することのない結果を出力するために設計されたアルゴリズムで作成される、適度なサイズ(16 バイト)のメッセージダイジェストを生成します。

MD5 関数を使用する

MD5 関数を使用して、テーブル内の変更を追跡できます。MD5 関数は、テーブル内のレコードの列の値セットのダイジェストを生成できます。このダイジェストをレコードの署名として扱い、レコードに加えられる変更を追跡するために使用できます。新しいレコードからのダイジェストが、テーブル内のレコードの既存のダイジェストと一致する場合、2 つのレコードは同一です。ダイジェストが異なる場合、レコードの列の値が変更されています。新たに変更されたレコードは、既存のキーのある値への変更を表す新しい代理キーとともにテーブルに追加されます。

MD5 関数は、ソフトウェアのインストール、ファイル比較、ファイル破損および改ざん検出のためのシェルスクリプトまたは Perl プログラムを開発するときに便利な場合があります。

MD5 関数は、ハッシュパッケージのキーとして使用するオブザベーションの重複しない ID を作成するためにも使用できます。詳細については、*“ハッシュパッケージの使用” (SAS DS2 プログラマガイド)*を参照してください。

例

MD5 関数によって返される結果を生成する方法の例を次に示します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  method init();
  dcl binary(32) y z having format $hex32.;
    y = md5('abc');
    z = md5('access method');
    put y= ;
    put z= ;
  end;
enddata;
run;
quit;
```

SAS は次の結果をログに書き込みます。

```
y=900150983CD24FB0D6963F7D28E17F72
z=53128C19421A8E0C7F6436D06A026537
```

MDY 関数

月、日および年の値から SAS 日付値を返します。

カテゴリ: 日付と時間

返されるデータの種類: DOUBLE

構文

MDY(*month*, *day*, *year*)

必須引数

month
1 から 12 までの整数を表す数値式を指定します。

データの種類 DOUBLE

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

day
1 から 31 までの整数を表す数値式を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

year
2 桁または 4 桁の年を表す数値式を指定します。YEARCUTOFF=システムオプションは、2 桁の日付の年の値を定義します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

例

次のプログラムは、MDY 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double d m y;
  dcl double birthday having format mmddyy10.;
  method run();
     d=8; m=27; y=19;
     birthday= mdy(d,m,y);
     put birthday;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
2019/08/27
```

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test2 (overwrite=yes);
  dcl double d m y;
  dcl double anniversary having format date9.;
  method run();
     d=7; m=11; y=19;
     anniversary= mdy(d,m,y);
     put anniversary;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

11JUL2019

関連項目

概念:

- “DS2 の日付と時間” (*SAS DS2 プログラマガイド*)

関数:

- “DAY 関数” (500 ページ)
- “MONTH 関数” (863 ページ)
- “YEAR 関数” (1263 ページ)

MEAN 関数

非 null または非欠損引数の算術平均(平均)を返します。

カテゴリ:	記述統計
返されるデータの種類:	DOUBLE

構文

MEAN(*expression-1*[, ...*expression-n*])

必須引数

expression
数値に評価される有効な式を指定します。

要件	少なくとも 1 つの非 null または非欠損の引数が必要です。それがない場合は、関数から null または欠損値が返されます。
データの種類	DOUBLE
参照項目:	“DS2 式” (*SAS DS2 プログラマガイド*)

比較

GEOMEAN 関数が幾何平均を返し、HARMEAN 関数が調和平均を返すのに対し、MEAN 関数は算術平均(平均)を返します。

例

次のプログラムは、MEAN 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double x1 x2 x3;
  method run();
    x1=mean(2, ., ., 6);
    put x1;
    x2=mean(1, 2, 3, 2);
    put x2;
    x3=mean(of x1-x2);
    put x3;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
4
2
3
```

関連項目

関数:

- “GEOMEAN 関数” (676 ページ)
- “GEOMEANZ 関数” (678 ページ)
- “HARMEAN 関数” (687 ページ)
- “HARMEANZ 関数” (689 ページ)
- “MEDIAN 関数” (850 ページ)

MEDIAN 関数

中央値を返します。

カテゴリ:	記述統計
返されるデータの種類:	DOUBLE

構文

MEDIAN(*expression-1*[, ...*expression-n*])

必須引数

expression
数値に評価される有効な式を指定します。

データの種類 DOUBLE

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

詳細

MEDIAN 関数は、非欠損値または非 null 値の中央値を返します。すべての引数が欠損値または null 値の場合、結果は欠損値または null 値になります。

注: MEDIAN 関数で使用される式は、Base SAS プロシジャガイド: 統計プロシジャに説明されている、PROC UNIVARIATE で使用される式と同じです。詳細については、SAS Elementary Statistics Procedures を参照してください。

比較

MEDIAN 関数が非欠損値または非 null 値の中央値を返すのに対し、MEAN 関数は算術平均(平均)を返します。

例

次のプログラムは、MEDIAN 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double x y;
  method run();
    x=median(2, 4, 1, 3);
    put x;
    y=median(5, 8, 0, 3, 4);
    put y;
  end;
enddata;
```

```
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
2.5
4
```

関連項目

関数:

- “MEAN 関数” (849 ページ)

MIN 関数

最小値を返します。

カテゴリ:	記述統計
返されるデータの種類:	BIGINT、DECIMAL、DOUBLE
注:	2025.12 以降、この関数では、引数が DOUBLE の場合、すべての引数が DOUBLE に変換され、結果は DOUBLE になります。PROC DS2 ステートメントで BIGINTPROCESSING=YES が設定され、いずれかの引数が BIGINT の場合、すべての引数が BIGINT に変換され、結果は BIGINT になります。

構文

MIN(*expression-1, expression-2* [, ...*expression-n*])

必須引数

expression
数値に評価される有効な式を指定します。

要件	少なくとも 2 つの引数が必要です。
データの種類	BIGINT、DECIMAL、DOUBLE
参照項目:	“DS2 式” (*SAS DS2 プログラマガイド*)

詳細

この関数の引数が数値以外の場合、その引数は DOUBLE に変換されます。いずれかの引数が DOUBLE または REAL の場合、すべての引数が(まだ変換されていなければ) DOUBLE に変換され、結果は DOUBLE になります。それ以外の場合で、いずれかの引数が DECIMAL のときは、すべての引数が(まだ変換されていなければ) DECIMAL に変換され、結果は DECIMAL になります。それ以外の場合は、すべての引数が BIGINT に変換され、結果は BIGINT になります。

比較

MIN 関数は、値のリストから最小値を返します。MIN 演算子(><)は、2 つのオペランドのうち小さい方を返します。

MIN 関数は、すべての引数が null または欠損の場合にのみ、null 値または欠損値を返します。MIN 演算子は、オペランドのどちらかが null または欠損の場合にのみ、null 値または欠損値を返します。この場合、null 値または欠損値の並べ替え順がより低いオペランドの値が返されます。

例

次のプログラムは、MIN 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double x x1 x2 x3 x4 x5;
  method run();
    x=min(7,4);
    put x;
    x1=min(2, ., 6);
    put x1;
    x2=min(2, -3, 1, -1);
    put x2;
    x3=min(0, 4);
    put x3;
    x4=min(of x1-x3);
    put x4;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
4
2
-3
0
-3
```

MINUTE 関数

SAS 時間または日付値から分を返します。

カテゴリ: 日付と時間

返されるデータの種類: DOUBLE

構文

MINUTE(*time* | *datetime*)

必須引数

time
SAS 時間値を表す有効な式を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

datetime
SAS 日時値を表す有効な式を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

MINUTE 関数は、特定の分を表す整数を返します。MINUTE は必ず 0 から 59 までの範囲の正の値を返します。null 値または欠損値は無視されます。

例

次のプログラムは、MINUTE 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double d;
  method init();
    d=minute(time());
    put d;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
38
```

```
proc ds2;
data test(overwrite=yes);
  dcl double m dt;
  dcl time t;
  method run();
    t=time '03:19:24';
    dt=to_double(t);
    m=minute(dt);
    put m;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
19
```

関連項目

- “DS2 式” (*SAS DS2 プログラマガイド*)

関数:

- “HOUR 関数” (699 ページ)
- “SECOND 関数” (1126 ページ)

MISSING 関数

引数に欠損値が含まれるかどうかを示す数字を返します。

カテゴリ:	特殊
返されるデータの種類:	INTEGER
注:	2025.12 以降、この関数では、引数が DOUBLE の場合、すべての引数が DOUBLE に変換され、結果は DOUBLE になります。PROC DS2 ステートメントで BIGINTPROCESSING=YES が設定され、いずれかの引数が BIGINT の場合、すべての引数が BIGINT に変換され、結果は BIGINT になります。

構文

MISSING(*expression*)

必須引数

expression
値に評価される有効な式を指定します。

データの種類	すべてのデータ型
注	SAS Federation Server を使用している場合、DS2_SASMISSING 環境変数が TRUE に設定されていると、FedSQL CALL 呼び出しで ANSI null 値が SAS 欠損値に変換されます。
参照項目:	“DS2 式” (*SAS DS2 プログラマガイド*)

詳細

- MISSING 関数は、値が null 値または欠損値かどうかを確認し、結果を数値で返します。引数に欠損値が含まれていない場合、SAS は値 0 を返します。引数に欠損値が含まれている場合、SAS は値 1 を返します。
- SAS モードでは、空白で埋められた文字値は、SAS 欠損値として定義されています。ANSI モードでは、空白で埋められた文字値は、非欠損および非 null として定義されています。
- SAS モードでは、DOUBLE 値は SAS 欠損値(.、.A から Z)である可能性があります。その他の数値型では、SAS 欠損値はサポートされていません。
- パッケージインスタンスが存在しない場合、MISSING 関数は 1 を返します。つまり、パッケージ変数は欠損しているパッケージ参照です。パッケージ変数がパッケージインスタンスを参照している場合、MISSING 関数は 0 を返します。

比較

MISSING 関数には引数を 1 つのみ指定できます。NMISS 関数には数値引数が必要で、引数リスト内の欠損値の数が返されます。

注: 欠損値と null 値は、SAS モードと ANSI モードで異なる方法で処理されます。モードに応じて、欠損値および null 値が変換される場合があります。

例

例 1: MISSING 関数の使用

次のプログラムは、MISSING 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl int a[3];
  dcl double i;
  method run();
    a[1]=2;
    a[2]=4;
    a[3]=.;
    do i = 1 to 3;
      if missing(a[i]) then put 'Missing';
        else put 'Not Missing';
    end;
  end;
enddata;
run;
quit;
```

次の行が SAS ログに書き込まれます。

```
Not Missing
Not Missing
Missing
```

例 2: SAS モードおよび ANSI モードでの MISSING 関数

このプログラムは、MISSING 関数を含む DS2 プログラムがモードに基づいて異なる結果を返す方法を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl char(1) a[3];
  dcl double  b[3];
  dcl int c[3];
  dcl int i;
  method run();
```

```
    a := ('a', '', NULL);
    b := (1, ., NULL);
    c := (1, NULL, NULL);
    do i = 1 to 3;
      if (missing(a[i])) then put a[i]= 'missing';
        else put a[i]= 'not missing';
       if (missing(b[i])) then put b[i]= 'missing';
        else put b[i]= 'not missing';
       if (missing(c[i])) then put c[i]= 'missing';
        else put c[i]= 'not missing';
    end;
  end;
enddata;
run;
quit;
```

SAS モードでは、次の行が SAS ログに書き込まれます。

```
a[1]=a not missing
b[1]=1 not missing
c[1]=1 not missing
a[2]=  missing
b[2]=. missing
c[2]= missing
a[3]=  missing
b[3]=. missing
c[3]= missing
```

ANSI モードでは、次の行が SAS ログに書き込まれます。

```
a[1]=a not missing
b[1]=1 not missing
c[1]=1 not missing
a[2]=  not missing
b[2]= missing
c[2]= missing
a[3]= missing
b[3]= missing
c[3]= missing
```

関連項目

- “DS2 による null および SAS 欠損値の処理方法” (*SAS DS2 プログラマガイド*)

関数:

- “CMISS 関数” (417 ページ)
- “NMISS 関数” (872 ページ)
- “N 関数” (867 ページ)
- “NULL 関数” (909 ページ)

MOD 関数

第 1 引数を第 2 引数で除算したときの(最も期待しない浮動小数点の結果を避けるためにファジー化した)余りを返します。

カテゴリ: 数学

返されるデータの種類: DOUBLE

注: 2025.12 以降、この関数では、引数が DOUBLE の場合、すべての引数が DOUBLE に変換され、結果は DOUBLE になります。PROC DS2 ステートメントで BIGINTPROCESSING=YES が設定され、いずれかの引数が BIGINT の場合、すべての引数が BIGINT に変換され、結果は BIGINT になります。

構文

MOD(*dividend-expression*, *divisor-expression*)

必須引数

dividend-expression
数値に評価される有効な式である被除数を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

divisor-expression
数値に評価される有効な式である除数を指定します。

制限事項 *divisor-expression* を 0 にすることはできません

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

MOD 関数は、*dividend-expression* を *divisor-expression* で除算したときの余りを返します。結果がゼロ以外のとき、結果は第 1 引数と同じ符号になります。第 2 引数の符号は無視されます。

次の条件の両方を満たす場合、MOD 関数で実行される計算は厳密になります。

- 両方の引数が厳密な整数である。

- すべての整数が、厳密に 8 バイトの浮動小数点で表されるいずれかの引数より小さい。

前述の条件のいずれかを満たしていない場合、浮動小数点計算で小さな数値誤差が発生する可能性があります。この場合、次のようになります。

- 余りが 0 か第 2 引数の値にとても近いと、MOD はゼロを返す。
- 約 3 桁以上の精度で余りを計算できなければ、MOD は null 値または欠損値を返す。この場合、ログにエラーメッセージが書き込まれる。

比較

MOD 関数と MODZ 関数の比較を次に示します。

- MOD 関数は、ファジーと呼ばれる追加の計算を実行し、厳密なゼロを返す(ファジーを実行しなければ数値誤差のため結果はゼロにならない)。
- MODZ 関数はファジーを実行しない。
- 約 3 桁以上の精度で余りを計算できなければ、MOD 関数と MODZ 関数はどちらも null 値または欠損値を返します。

例

次のプログラムは、MOD および MODZ 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double x1 x2 x3  xa xb xc having format 9.4;
  dcl double x4 xd having format 24.20;
  method run();
    x1=mod(10, 3);
    put x1=;
    xa=modz(10, 3);
    put xa=;
    x2=mod(.3, -.1);
    put x2=;
    xb=modz(.3, -.1);
    put xb=;
    x3=mod(1.7, .1);
    put x3=;
    xc=modz(1.7, .1);
    put xc=;
    x4=mod(.9, .3);
    put x4=;
    xd=modz(.9, .3);
    put xd=;
  end;
```

```
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
x1=  1.0000
xa=  1.0000
x2=  0.0000
xb=  0.1000
x3=  0.0000
xc=  0.0000
x4=  0.00000000000000000000
xd=  0.00000000000000011102
```

関連項目

関数:

- “MODZ 関数” (861 ページ)
- “INT 関数” (717 ページ)
- “INTZ 関数” (781 ページ)

MODZ 関数

第 1 引数を第 2 引数で除算したときの余りを、ゼロファジーを使用して返します。

カテゴリ: 数学

返されるデータの種類: DOUBLE

構文

MODZ(*dividend-expression*, *divisor-expression*)

必須引数

dividend-expression
数値に評価される有効な式である被除数を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

divisor-expression
数値に評価される有効な式である除数を指定します。

制限事項	*divisor-expression* を 0 にすることはできません
データの種類	DOUBLE
参照項目:	“DS2 式” (*SAS DS2 プログラマガイド*)

詳細

MODZ 関数は、*dividend-expression* を *divisor-expression* で除算したときの余りを返します。結果がゼロ以外のとき、結果は第 1 引数と同じ符号になります。第 2 引数の符号は無視されます。

次の条件の両方を満たす場合、MODZ 関数で実行される計算は厳密になります。

- 両方の引数が厳密な整数である。
- すべての整数が、厳密に 8 バイトの浮動小数点で表されるいずれかの引数より小さい。

前述の条件のいずれかを満たしていない場合、浮動小数点計算で小さな数値誤差が発生する可能性があります。たとえば、厳密算術を使用して結果がゼロのとき、MODZ は正の極小値または第 2 引数より若干小さい値を返す場合があります。

比較

MODZ 関数と MOD 関数の比較を次に示します。

- MODZ 関数はファジーを実行しない。
- MOD 関数は、ファジーを実行し、厳密なゼロを返す(ファジーを実行しなければ数値誤差のため結果はゼロにならない)。
- 約 3 桁以上の精度で余りを計算できなければ、MODZ 関数と MOD 関数はどちらも null 値または欠損値を返します。

例

次のステートメントは、MOD および MODZ 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double x1 x2 x3  xa xb xc having format 9.4;
  dcl double x4 xd having format 24.20;
  method run();
    x1=mod(10, 3);
    put x1=;
```

```
    xa=modz(10, 3);
    put xa=;
    x2=mod(.3, -.1);
    put x2=;
    xb=modz(.3, -.1);
    put xb=;
    x3=mod(1.7, .1);
    put x3=;
    xc=modz(1.7, .1);
    put xc=;
    x4=mod(.9, .3);
    put x4=;
    xd=modz(.9, .3);
    put xd=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
x1=  1.0000
xa=  1.0000
x2=  0.0000
xb=  0.1000
x3=  0.0000
xc=  0.0000
x4=  0.099999999999999999
xd=  0.099999999999999911102
```

関連項目

関数:

- “INT 関数” (717 ページ)
- “INTZ 関数” (781 ページ)
- “MOD 関数” (859 ページ)

MONTH 関数

SAS 日付値から月を表す数字を返します。

カテゴリ:	日付と時間
返されるデータの種類:	DOUBLE

構文

MONTH(*date*)

必須引数

date
SAS 日付値を表す有効な式を指定します。

範囲 1-12

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

例

次のプログラムは、月が 3 月の場合の MONTH 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double d;
  method init();
    d=month(date());
    put d;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
3
```

次のプログラムは、日付が 2019 年 2 月 5 日の場合の MONTH 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double m dat;
  dcl date d;
  method run();
    d=date '2019-02-05';
    dat=to_double(d);
    m=month(dat);
```

```
    put m;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
2
```

関連項目

- “DS2 式” (*SAS DS2 プログラマガイド*)

関数:

- “DAY 関数” (500 ページ)
- “YEAR 関数” (1263 ページ)

MORT 関数

割賦返済パラメーターを返します。

カテゴリ:	財務
返されるデータの種類:	DOUBLE

構文

MORT(*a*, *p*, *r*, *n*)

必須引数

a
初期の金額に評価される有効な式を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

p
定期的支払いに評価される有効な式を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

r
分数で表す定期的な利率に評価される有効な式を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

n
複利期間数に評価される有効な式を指定します。

範囲 $n \geq 0$

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

結果の計算

MORT 関数は毎期間複利計算される固定利率の割賦返済計算から 4 つの引数のリスト内の欠損引数を返します。これらの引数には次の式の関係があります。

$$p = \frac{ar(1+r)^n}{(1+r)^n - 1}$$

引数は 1 つを欠損値とする必要があります。値は残りの 3 つから計算されます。結果を変換して丸めた数字にする調整は行われません。

結果を計算するときの制限事項

次の引数の組み合わせのいずれかに該当する場合、MORT 関数は無効な引数というメモを SAS ログに返し、_ERROR_を 1 に設定します。

- rate < –1 or n < 0
- principal <= 0 または payment <= 0 または n <= 0
- principal <= 0 または payment <= 0 または rate <= –1
- principal * rate > payment
- principal > payment * n

例

次のプログラムでは、$50,000 を毎月複利計算される年利 10%の 30 年で借入します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double payment;
  method run();
    payment=mort(50000, . , .10/12, 30*12);
    put payment;
  end;
enddata;
run;
quit;
```

返される値は 438.79(丸め値)です。第 2 引数は欠損に設定されています。これは定期的支払いが計算されることを示します。名目年利 10%は、月利 0.10/12 に変換されています。rate は分数(パーセントではない)で表す複利計算期間当たりの利率です。30 年は 360 か月に変換されます。

N 関数

非 null または非欠損の数値の数を返します。

カテゴリ: 特殊

返されるデータの種類: INTEGER

注: 2025.12 以降、この関数では、引数が DOUBLE の場合、すべての引数が DOUBLE に変換され、結果は DOUBLE になります。PROC DS2 ステートメントで BIGINTPROCESSING=YES が設定され、いずれかの引数が BIGINT の場合、すべての引数が BIGINT に変換され、結果は BIGINT になります。

構文

N(*expression* [, ...*expression*])

必須引数

expression
数値に評価される有効な式を指定します。

要件 少なくとも 1 つの引数が必要です。

データの種類 DECIMAL、DOUBLE、NUMERIC

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

詳細

null 値は欠損値に変換され、欠損値として数えられます。

比較

N 関数は非 null 値と非欠損値を数えますが、NMISS 関数は欠損値を数えます。N 関数には数値引数が必要です。

例

次のプログラムは、N 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double x1 x2 x3;
  method run();
    x1=n(1, 0,., 2, 5, .);
    x2=n(1, 2);
    x3=n(of x1-x2);
    put x1= x2= x3=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
x1=4 x2=2 x3=2
```

関連項目

関数:

- “NMISS 関数” (872 ページ)

NDIMS 関数

配列の次元数を返します。

カテゴリ: 配列

返されるデータの種類: INT

構文

NDIMS(*array-name*)

必須引数

array-name
一時配列または変数配列の名前を指定します。

詳細

NDIMS 関数は、多次元配列の次元数を返すか、1 次元配列の場合は 1 を返します。

比較

- DIM は、配列次元の要素数を返します。
- HBOUND は、配列次元の上限の値を返します。
- LBOUND は、配列次元の下限の値を返します。
- NDIMS は、配列の次元数を返します。

例

次のプログラムは、DIM、HBOUND、LBOUND、NDIMS 配列関数の使用方法を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl char(15) a1[4];
  dcl double   a2[2,3,4] sum i j k numelems;
  method init();
    a1 := ('red' 'yellow' 'green' 'blue');
    a2 := (24*2.0);
    do i = 1 to dim(a1);
      put a1[i];
    end;
```

```
    numelems = 0;
    do i = 1 to ndims(a2);
      numelems = numelems + dim(a2, i);
    end;
    sum = 0;
    do i = lbound(a2, 1) to hbound(a2, 1);
      do j = lbound(a2, 2) to hbound(a2, 2);
        do k = lbound(a2, 3) to hbound(a2, 3);
          sum = sum + a2[i,j,k];
        end;
      end;
    end;
    put sum=;
  end;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
red
yellow
green
blue
sum=48
```

関連項目

関数:

- “DIM 関数” (518 ページ)
- “HBOUND 関数” (691 ページ)
- “LBOUND 関数” (801 ページ)

NETPV 関数

現在正味価値をパーセントで返します。

カテゴリ:	財務
返されるデータの種類:	DOUBLE

構文

NETPV(*r*, *freq*, *c0*, *c1*, ..., *cn*)

必須引数

r
指定したベース期間の利率を分数で表した数値。

範囲 *r* >= 0

データの種類 DOUBLE

freq
利率 *r* で指定したベース期間中の支払い回数を表す数値。

範囲 *freq* > 0

データの種類 DOUBLE

注 *freq* = 0 は、連続割引を許可するフラグです。

c0, c1 ,..., cn
時間 0,1,...n に発生する現金支出(支払い)または現金流入(収入)を表す数値のキャッシュフローです。これらのキャッシュフローは、等間隔の期間開始時の値であると想定されます。負の値は支払い、正の値は収入、値 0 はその時点でキャッシュフローがないことを表します。*c0* 引数と *c1* 引数は必須です。

データの種類 DOUBLE

詳細

NETPV 関数は、指定したベース期間中に利率 *r* を使用して、一連の現金支払い *c0*,*c1*, ...,*cn* に対する時間 0 の正味現在価値を返します。引数 *freq*>0 は、指定ベース期間で発生する支払い数を表します。

正味現在価値は次の式で求められます。

$$\mathrm{NETPV}(r, freq, c_0, c_1, ..., c_n) = \sum_{i=0}^{n} c_i x^i$$

前述の式には次の関係が適用されます。

$$x = \begin{cases} \frac{1}{(1+r)^{(1/freq)}} & freq > 0 \\ \varepsilon^{-r} & freq = 0 \end{cases}$$

支払いの欠損値は 0 値として扱われます。*freq*>0 のとき、利率 *r* は指定したベース期間の実効金利です。四半期金利 4%(ベース期間は 3 か月)の毎月現金支払いで計算するには、*freq* を 3、*r* を.04 に設定します。

freq が 0 の場合は、連続割引とみなされます。ベース期間は 2 つの連続する支払い間の時間間隔で、利率 *r* は名目金利です。

名目年利 11%、連続割引、毎月支払いで計算するには、*freq* を 0、*r* を.11/12 に設定します。

例

翌 6 年間にわたり$200、$300、$400 の年 2 回支払いおよび年間割引率 10%を返す初期投資$500 では、投資の現在正味価値は次のように表すことができます。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double value;
  method run();
    value=netpv(.10,.5,-500,200,300,400);
    put value;
  end;
enddata;
run;
quit;
```

返される値は 95.982864829379 です。

関連項目

関数:

- “NPV 関数” (908 ページ)

NMISS 関数

null および SAS 欠損数値の数を返します。

カテゴリ:	記述統計
返されるデータの種類:	INTEGER
注:	2025.12 以降、この関数では、引数が DOUBLE の場合、すべての引数が DOUBLE に変換され、結果は DOUBLE になります。PROC DS2 ステートメントで BIGINTPROCESSING=YES が設定され、いずれかの引数が BIGINT の場合、すべての引数が BIGINT に変換され、結果は BIGINT になります。

構文

NMISS(*expression* [, ...*expression*])

必須引数

expression
数値に評価される有効な式を指定します。

要件 少なくとも 1 つの引数が必要です。

データの種類 DECIMAL、DOUBLE、NUMERIC

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

null 値は SAS 欠損値に変換され、欠損値として数えられます。

比較

NMISS 関数が null 値または SAS 欠損値の数を返すのに対し、N 関数は非 null および非欠損値の数を返します。NMISS が数値を必要とし、複数の数値を使用できるのに対し、MISSING は数値か文字のいずれかの値を 1 つだけ使用できます。

例

次のプログラムは、NMISS 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double x1 x2 x3;
  method run();
    x1=nmiss(1,0,.,2,5,.);
    x2=nmiss(1,0);
    x3=nmiss(of x1-x2);
    put x1= x2= x3=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
x1=2 x2=0 x3=0
```

関連項目

- “DS2 による null および SAS 欠損値の処理方法” (*SAS DS2 プログラマガイド*)

関数:

- “CMISS 関数” (417 ページ)
- “MISSING 関数” (855 ページ)
- “N 関数” (867 ページ)
- “NULL 関数” (909 ページ)

NOMRATE 関数

名目年利を返します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

NOMRATE(*compounding-interval*, *rate*)

必須引数

compounding-interval
SAS 間隔です。この値は、返される値が複利計算される頻度を表します。

データの種類 CHAR

rate
数値です。*rate* は各期間複利計算される実効年利(パーセント)です。

データの種類 DOUBLE

詳細

NOMRATE 関数は名目年利を返します。NOMRATE は実効年利に対応する名目年利を計算します。

次の詳細は NOMRATE 関数に適用されます。

- rate の値は–99 以上にする必要があります。

- 実効金利と複利計算間隔を検討するとき、*compounding-interval* が 'CONTINUOUS'であれば、NOMRATE によって返される値は $\log_e(1+[rate/100])$に等しくなります。

  compounding-interval が'CONTINUOUS'でなく、1 年に *m* 個の間隔が発生する場合、NOMRATE によって返される値は次に等しくなります。

  $$m\left((1+\frac{rate}{100})^{\frac{1}{m}}-1\right)$$

- *compounding-interval* で有効な値は次のとおりです。
  - 'CONTINUOUS'
  - 'DAY'
  - 'SEMIMONTH'
  - 'MONTH'
  - 'QUARTER'
  - 'SEMIYEAR'
  - 'YEAR'
- 間隔が'DAY'の場合、*m*=365 です。

例

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

- 毎月複利で実効金利が 10%の場合、対応する名目金利は次のように表すことができます。

```
proc ds2;
data test(overwrite=yes);
  dcl double effective_rate1;
  method run();
    effective_rate1 = NOMRATE('MONTH', 10);
    put effective_rate1=;
  end;
enddata;
run;
quit;
```

  SAS は次の出力をログに書き込みます。

```
effective_rate1=9.56896851468451
```

- 四半期複利で実効金利が 10%の場合、対応する名目金利は次のように表すことができます。

```
proc ds2;
data test(overwrite=yes);
  dcl double effective_rate2;
  method run();
```

```
    effective_rate2 = NOMRATE('QUARTER', 10);
    put effective_rate2=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
effective_rate2=9.64547563377804
```

NOTALNUM 関数

文字列から英数字でない文字を検索し、その文字が見つかった最初の文字位置を返します。

カテゴリ: 文字

返されるデータの種類: DOUBLE

構文

NOTALNUM('*expression*'[, *start*])

必須引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、VARCHAR、NVARCHAR

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

オプション引数

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

NOTALNUM 関数の結果は、有効になっている変換テーブル("TRANTAB= システムオプション" (*SAS 各国語サポート(NLS): リファレンスガイド*)を参照)に直接依存し、ENCODING および LOCALE システムオプションに間接的に依存します。

NOTALNUM 関数は、文字列から数字、大文字または小文字でない文字の最初の出現を検索します。対象の文字が検出されると、NOTALNUM は文字列内の該当文字の位置を返します。対象の文字が検出されない場合、NOTALNUM は値 0 を返します。

引数を 1 つのみ使用すると、NOTALNUM は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

次のいずれかの条件が満たされると、NOTALNUM は値 0 を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

NOTALNUM 関数は、文字列から英数字でない文字を検索します。ANYALNUM 関数は、文字列から英数字を検索します。

例

次のプログラムでは、NOTALNUM 関数を使用して文字列の左から右へ英数字でない文字を検索します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl nchar(16) string c;
  dcl double j i;
  method run();
    string='Next = Last + 1;';
    j=0;
    do until(j=0);
      j=notalnum(string, j+1);
      if j=0 then put 'The end';
      else do;
        c=substr(string, j, 1);
        put j= c=;
      end;
    end;
```

```
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
j=5 c=
j=6 c==
j=7 c=
j=12 c=
j=13 c=+
j=14 c=
j=16 c=;
The end
```

関連項目

関数:

- “ANYALNUM 関数” (276 ページ)

NOTALPHA 関数

文字列からアルファベットでない文字を検索し、その文字が見つかった最初の文字位置を返します。

カテゴリ:	文字
返されるデータの種類:	DOUBLE

構文

NOTALPHA('*expression*'[, *start*])

必須引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、VARCHAR、NVARCHAR

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

オプション引数

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

NOTALPHA 関数の結果は、有効になっている変換テーブル(“TRANTAB= システムオプション” (*SAS 各国語サポート(NLS): リファレンスガイド*)を参照)に直接依存し、ENCODING および LOCALE システムオプションに間接的に依存します。

NOTALPHA 関数は、文字列から大文字または小文字でない文字の最初の出現を検索します。対象の文字が検出されると、NOTALPHA は文字列内の該当文字の位置を返します。対象の文字が検出されない場合、NOTALPHA は値 0 を返します。

引数を 1 つのみ使用すると、NOTALPHA は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

次のいずれかの条件が満たされると、NOTALPHA は値 0 を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

NOTALPHA 関数は、文字列からアルファベットでない文字を検索します。
ANYALPHA 関数は、文字列からアルファベットを検索します。

例

例 1: 文字列からアルファベットでない文字を検索する

次のプログラムでは、NOTALPHA 関数を使用して文字列の左から右へアルファベットでない文字を検索します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl char(18) string c;
  dcl double j i;
  method run();
    string='Next = _n_ + 12E3;';
    j=0;
    do until(j=0);
      j=notalpha(string, j+1);
      if j=0 then put 'The end';
      else do;
        c=substr(string, j, 1);
        put j= c=;
      end;
    end;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
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=;
The end
```

例 2: NOTALPHA 関数を使用した制御文字の識別

次のプログラムを実行して、NOTALPHA 関数で識別される制御文字を表示できます。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test/overwrite=yes;
  dcl nchar(3) byte1 hex1;
  dcl double dec notalpha1;

  method run();
    do dec=0 to 255;
      byte1=byte(dec);
      hex1=put(dec,hex2.);
```

```
      notalpha1=notalpha(byte1);
      output;
    end;
  end;
enddata;
run;
quit;
```

関連項目

関数:

- “ANYALPHA 関数” (279 ページ)

NOTCNTRL 関数

文字列から制御文字でない文字を検索し、その文字が見つかった最初の文字位置を返します。

カテゴリ: 文字

返されるデータの種類: DOUBLE

構文

NOTCNTRL('*expression*'[, *start*])

必須引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、VARCHAR、NVARCHAR

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

オプション引数

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

NOTCNTRL 関数の結果は、有効になっている変換テーブル(“TRANTAB= システムオプション” (*SAS 各国語サポート(NLS): リファレンスガイド*)を参照)に直接依存し、ENCODING および LOCALE システムオプションに間接的に依存します。

NOTCNTRL 関数は、文字列から制御文字でない文字の最初の出現を検索します。対象の文字が検出されると、NOTCNTRL は文字列内の該当文字の位置を返します。対象の文字が検出されない場合、NOTCNTRL は値 0 を返します。

引数を 1 つのみ使用すると、NOTCNTRL は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

次のいずれかの条件が満たされると、NOTCNTRL は値 0 を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

NOTCNTRL 関数は、文字列から制御文字でない文字を検索します。ANYCNTRL 関数は、文字列から制御文字を検索します。

例

次のプログラムを実行して、NOTCNTRL 関数で識別される制御文字を表示できます。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double dec notcntrl1;
  dcl char byte1 hex1;
  method run();
    do dec=0 to 255;
      byte1=byte(dec);
      hex1=put(dec, hex2.);
      notcntrl1=notcntrl(byte1);
      output;
```

```
    end;
  end;
enddata;
run;
quit;
```

関連項目

関数:

- “ANYCNTRL 関数” (282 ページ)

NOTDIGIT 関数

文字列から数字でない文字を検索し、その文字が見つかった最初の文字位置を返します。

カテゴリ: 文字

返されるデータの種類: DOUBLE

構文

NOTDIGIT('*expression*'[, *start*])

必須引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、VARCHAR、NVARCHAR

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

オプション引数

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

NOTDIGIT 関数の結果は、有効になっている変換テーブル(“TRANTAB= システムオプション” (*SAS 各国語サポート(NLS): リファレンスガイド*)を参照)に直接依存し、ENCODING および LOCALE システムオプションに間接的に依存します。

NOTDIGIT 関数は、文字列から数字でない文字の最初の出現を検索します。対象の文字が検出されると、NOTDIGIT は文字列内の該当文字の位置を返します。対象の文字が検出されない場合、NOTDIGIT は値 0 を返します。

引数を 1 つのみ使用すると、NOTDIGIT は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

次のいずれかの条件が満たされると、NOTDIGIT は値 0 を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

NOTDIGIT 関数は、文字列から数字でない文字を検索します。ANYDIGIT 関数は、文字列から数字を検索します。

例

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

例 1: 数字でない文字の検索

次のプログラムでは、NOTDIGIT 関数を使用して数字でない文字を検索します。

```
proc ds2;
data _null_;
  dcl nchar(18) string c;
  dcl double j i;
  method run();
    string='Next = _n_ + 12E3;';
    j=0;
    do until(j=0);
      j=notdigit(string, j+1);
```

```
        if j=0 then put 'The end';
        else do;
          c=substr(string, j, 1);
          put j= c=;
        end;
      end;
    end;
  enddata;
  run;
  quit;
```

SAS は次の出力をログに書き込みます。

```
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=16 c=E
j=18 c=;
The end
```

例 2: NOTDIGIT 関数への圧縮された値のサブミット

次のプログラムでは、圧縮された値を NOTDIGIT 関数にサブミットする方法を示します。

```
proc ds2;
data _null_;
  dcl char(5) x z;
  dcl double y;
  method run();
    x='1 ';
    y=notdigit(compress(x));
    put y=;
  end;
enddata;
run;
quit;
```

```
y=0
```

関連項目

関数:

- “ANYDIGIT 関数” (284 ページ)

NOTFIRST 関数

VALIDVARNAME=V7 の SAS 変数名として無効な開始文字を文字列から検索し、その文字が見つかった最初の文字位置を返します。

カテゴリ: 文字

返されるデータの種類: DOUBLE

構文

NOTFIRST('*expression*'[, *start*])

必須引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、VARCHAR、NVARCHAR

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

オプション引数

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

NOTFIRST 関数は、TRANTAB、ENCODING または LOCALE システムオプションに依存しません。

NOTFIRST 関数は、VALIDVARNAME=V7 の SAS 変数名の開始文字として有効でない文字のうち最初に出現する文字を文字列から検索します。これらの文字は、アンダースコア(_)、大文字または小文字の英文字以外の文字です。対象の文字が検出されると、NOTFIRST は文字列内の該当文字の位置を返します。対象の文字が検出されない場合、NOTFIRST は値 0 を返します。

引数を 1 つのみ使用すると、NOTFIRST は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

次のいずれかの条件が満たされると、NOTFIRST は値 0 を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

NOTFIRST 関数は、VALIDVARNAME=V7 の SAS 変数名の開始文字として有効でない文字のうち最初に出現する文字を文字列から検索します。ANYFIRST 関数は、VALIDVARNAME=V7 の SAS 変数名の開始文字として有効な文字のうち最初に出現する文字を文字列から検索します。

例

次のプログラムでは、NOTFIRST 関数を使用して VALIDVARNAME=V7 の SAS 変数名の開始文字として有効でない文字を文字列から検索します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl nchar(18) string c;
  dcl double j i;
  method run();
    string='Next = _n_ + 12E3;';
    j=0;
    do until(j=0);
      j=notfirst(string, j+1);
      if j=0 then put 'The end';
      else do;
        c=substr(string, j, 1);
        put j= c=;
      end;
    end;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
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=;
The end
```

関連項目

関数:

- “ANYFIRST 関数” (286 ページ)

NOTGRAPH 関数

文字列からグラフィカル文字でない文字を検索し、その文字が見つかった最初の文字位置を返します。

カテゴリ: 文字

返されるデータの種類: DOUBLE

構文

NOTGRAPH('*expression*'[, *start*])

必須引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、VARCHAR、NVARCHAR

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

オプション引数

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

NOTGRAPH 関数の結果は、有効になっている変換テーブル(“TRANTAB= システムオプション” (*SAS 各国語サポート(NLS): リファレンスガイド*)を参照)に直接依存し、ENCODING および LOCALE システムオプションに間接的に依存します。

NOTGRAPH 関数は、文字列からグラフィカル文字でない文字の最初の出現を検索します。グラフィカル文字は、空白以外の印刷可能文字として定義されます。対象の文字が検出されると、NOTGRAPH は文字列内の該当文字の位置を返します。対象の文字が検出されない場合、NOTGRAPH は値 0 を返します。

引数を 1 つのみ使用すると、NOTGRAPH は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

次のいずれかの条件が満たされると、NOTGRAPH は値 0 を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

NOTGRAPH 関数は、文字列からグラフィカル文字でない文字を検索します。ANYGRAPH 関数は、文字列からグラフィカル文字を検索します。

例

例 1: 文字列からグラフィカルでない文字を検索

次のプログラムでは、NOTGRAPH 関数を使用して文字列からグラフィカルでない文字を検索します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl nchar(18) string c;
  dcl double j i;
  method run();
    string='Next = _n_ + 12E3;';
```

```
    j=0;
    do until(j=0);
      j=notgraph(string, j+1);
      if j=0 then put 'The end';
      else do;
        c=substr(string, j, 1);
        put j= c=;
      end;
    end;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
j=5 c=
j=7 c=
j=11 c=
j=13 c=
The end
```

例 2: NOTGRAPH 関数を使用した制御文字の識別

次のプログラムを実行して、NOTGRAPH 関数で識別される制御文字を表示できます。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl nchar byte1 hex1;
  dcl double dec notgraph1;

  method run();
    do dec=0 to 255;
      byte1=byte(dec);
      hex1=put(dec,hex2.);
      notgraph1=notgraph(byte1);
      output;
    end;
  end;
enddata;
run;
quit;
```

関連項目

関数:

- “ANYGRAPH 関数” (289 ページ)

NOTLOWER 関数

文字列から小文字でない文字を検索し、その文字が見つかった最初の文字位置を返します。

カテゴリ: 文字

返されるデータの種類: DOUBLE

構文

NOTLOWER('*expression*'[, *start*])

必須引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、VARCHAR、NVARCHAR

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

オプション引数

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

NOTLOWER 関数の結果は、有効になっている変換テーブル("TRANTAB= システムオプション" (*SAS 各国語サポート(NLS): リファレンスガイド*)を参照)に直接依存し、ENCODING および LOCALE システムオプションに間接的に依存します。

NOTLOWER 関数は、文字列から小文字でない文字の最初の出現を検索します。対象の文字が検出されると、NOTLOWER は文字列内の該当文字の位置を返します。対象の文字が検出されない場合、NOTLOWER は値 0 を返します。

引数を 1 つのみ使用すると、NOTLOWER は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。

- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

次のいずれかの条件が満たされると、NOTLOWER は値 0 を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

NOTLOWER 関数は、文字列から小文字でない文字を検索します。ANYLOWER 関数は、文字列から小文字を検索します。

例

次のプログラムでは、NOTLOWER 関数を使用して文字列から小文字でない文字を検索します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl nchar(18) string c;
  dcl double j i;
  method run();
    string='Next = _n_ + 12E3;';
    j=0;
    do until(j=0);
      j=notlower(string, j+1);
      if j=0 then put 'The end';
      else do;
        c=substr(string, j, 1);
        put j= c=;
      end;
    end;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
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=;
The end
```

関連項目

関数:

- “ANYLOWER 関数” (292 ページ)

NOTNAME 関数

VALIDVARNAME=V7 の SAS 変数名として無効な文字を文字列から検索し、その文字が見つかった最初の文字位置を返します。

カテゴリ:	文字
返されるデータの種類:	DOUBLE

構文

NOTNAME('*expression*'[, *start*])

必須引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、VARCHAR、NVARCHAR

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

オプション引数

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

NOTNAME 関数は、TRANTAB、ENCODING または LOCALE システムオプションに依存しません。

NOTNAME 関数は、VALIDVARNAME=V7 の SAS 変数名で有効でない文字のうち最初に出現する文字を文字列から検索します。これらの文字は、アンダースコア(_)、数字、大文字または小文字の英文字以外の文字です。対象の文字が検出されると、NOTNAME は文字列内の該当文字の位置を返します。対象の文字が検出されない場合、NOTNAME は値 0 を返します。

引数を 1 つのみ使用すると、NOTNAME は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

次のいずれかの条件が満たされると、NOTNAME は値 0 を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

NOTNAME 関数は、VALIDVARNAME=V7 の SAS 変数名で有効でない文字のうち最初に出現する文字を文字列から検索します。ANYNAME 関数は、VALIDVARNAME=V7 の SAS 変数名で有効な文字のうち最初に出現する文字を文字列から検索します。

例

次のプログラムでは、NOTNAME 関数を使用して VALIDVARNAME=V7 の SAS 変数名として有効でない文字を文字列から検索します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl nchar(18) string c;
  dcl double j i;
  method run();
    string='Next = _n_ + 12E3;';
    j=0;
    do until(j=0);
      j=notname(string, j+1);
      if j=0 then put 'The end';
      else do;
        c=substr(string, j, 1);
        put j= c=;
      end;
    end;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
j=5 c=
j=6 c==
j=7 c=
j=11 c=
j=12 c=+
j=13 c=
j=18 c=;
The end
```

関連項目

関数:

- “ANYNAME 関数” (294 ページ)

NOTPRINT 関数

文字列から印刷不可文字を検索し、その文字が見つかった最初の文字位置を返します。

カテゴリ:	文字
返されるデータの種類:	DOUBLE

構文

NOTPRINT('*expression*'[, *start*])

必須引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、VARCHAR、NVARCHAR

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

オプション引数

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

NOTPRINT 関数の結果は、有効になっている変換テーブル("TRANTAB= システムオプション" (*SAS 各国語サポート(NLS): リファレンスガイド*)を参照)に直接依存し、ENCODING および LOCALE システムオプションに間接的に依存します。

NOTPRINT 関数は、文字列で最初に出現する印刷不可文字を検索します。対象の文字が検出されると、NOTPRINT は文字列内の該当文字の位置を返します。対象の文字が検出されない場合、NOTPRINT は値 0 を返します。

引数を 1 つのみ使用すると、NOTPRINT は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

次のいずれかの条件が満たされると、NOTPRINT は値 0 を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

NOTPRINT 関数は、文字列から印刷不可文字を検索します。ANYPRINT 関数は、文字列から印刷可能文字を検索します。

例

次のプログラムを実行して、NOTPRINT 関数で識別される制御文字を表示できます。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double dec notprint1;
  dcl nchar byte1 hex1;
  method run();
    do dec=0 to 255;
      byte1=byte(dec);
      hex1=put(dec, hex2.);
      notprint1=notprint(byte1);
      output;
    end;
  end;
enddata;
run;
quit;
```

関連項目

関数:

- “ANYPRINT 関数” (296 ページ)

NOTPUNCT 関数

文字列から句読文字でない文字を検索し、その文字が見つかった最初の文字位置を返します。

カテゴリ:	文字
返されるデータの種類:	DOUBLE

構文

NOTPUNCT('*expression*'[, *start*])

必須引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、VARCHAR、NVARCHAR

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

オプション引数

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

NOTPUNCT 関数の結果は、有効になっている変換テーブル("TRANTAB= システムオプション" (*SAS 各国語サポート(NLS): リファレンスガイド*)を参照)に直接依存し、ENCODING および LOCALE システムオプションに間接的に依存します。

NOTPUNCT 関数は、文字列から句読文字でない文字の最初の出現を検索します。対象の文字が検出されると、NOTPUNCT は文字列内の該当文字の位置を返します。対象の文字が検出されない場合、NOTPUNCT は値 0 を返します。

引数を 1 つのみ使用すると、NOTPUNCT は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

次のいずれかの条件が満たされると、NOTPUNCT は値 0 を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

NOTPUNCT 関数は、文字列から句読文字でない文字を検索します。ANYPUNCT 関数は、文字列から句読文字を検索します。

例

例 1: 文字列からの句読文字でない文字の検索

次のプログラムでは、NOTPUNCT 関数を使用して文字列から句読文字でない文字を検索します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl char(18) string c;
  dcl double j i;
  method run();
    string='Next = _n_ + 12E3;';
    j=0;
    do until(j=0);
      j=notpunct(string, j+1);
      if j=0 then put 'The end';
      else do;
        c=substr(string, j, 1);
        put j= c=;
      end;
    end;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
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=9 c=n
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
The end
```

例 2: NOTPUNCT 関数を使用した制御文字の識別

次のプログラムを実行して、NOTPUNCT 関数で識別される制御文字を表示できます。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test/overwrite=yes;
  dcl nchar(3) byte1 hex1;
  dcl double dec notpunct1;

  method run();
    do dec=0 to 255;
      byte1=byte(dec);
      hex1=put(dec,hex2.);
      notpunct1=notpunct(byte1);
      output;
    end;
  end;
enddata;
run;
quit;

 proc print data=test;
 run;
quit;
```

関連項目

関数:

- “ANYPUNCT 関数” (299 ページ)

NOTSPACE 関数

空白文字ではない文字(空白、水平タブと垂直タブ、キャリッジリターン、改行、およびフォームフィード)を文字列で検索し、その文字が見つかった最初の文字位置を返します。

カテゴリ:	文字
返されるデータの種類:	DOUBLE

構文

NOTSPACE('*expression*'[, *start*])

必須引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、VARCHAR、NVARCHAR

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

オプション引数

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

NOTSPACE 関数の結果は、有効になっている変換テーブル("TRANTAB= システムオプション" (*SAS 各国語サポート(NLS): リファレンスガイド*)を参照)に直接依存し、ENCODING および LOCALE システムオプションに間接的に依存します。

NOTSPACE 関数は、文字列から空白、水平タブ、垂直タブ、キャリッジリターン、ラインフィード、フォームフィードでない文字の最初の出現を検索します。対象の文字が検出されると、NOTSPACE は文字列内の該当文字の位置を返します。対象の文字が検出されない場合、NOTSPACE は値 0 を返します。

引数を 1 つのみ使用すると、NOTSPACE は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

次のいずれかの条件が満たされると、NOTSPACE は値 0 を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

NOTSPACE 関数は、文字列で最初に出現する空白、水平タブ、垂直タブ、キャリッジリターン、ラインフィード、フォームフィードでない文字を検索します。
ANYSPACE 関数は、文字列で最初に出現する空白、水平タブ、垂直タブ、キャリッジリターン、ラインフィード、フォームフィードの文字を検索します。

例

例 1: 文字列からの空白文字でない文字の検索

次のプログラムでは、NOTSPACE 関数を使用して、文字列から空白文字でない文字を検索します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl char(18) string c;
  dcl double j i;
  method run();
    string='Next = _n_ + 12E3;';
    j=0;
    do until(j=0);
      j=notspace(string, j+1);
      if j=0 then put 'The end';
      else do;
        c=substr(string, j, 1);
        put j= c=;
      end;
    end;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
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=;
The end
```

例 2: NOTSPACE 関数を使用した制御文字の識別

次のプログラムを実行して、NOTSPACE 関数で識別される制御文字を表示できます。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl nchar(3) byte1 hex1;
  dcl double dec notspace1;

  method run();
    do dec=0 to 255;
      byte1=byte(dec);
      hex1=put(dec,hex2.);
      notspace1=notspace(byte1);
      output;
    end;
  end;
enddata;
run;
quit;
```

関連項目

関数:

- “ANYSPACE 関数” (302 ページ)

NOTUPPER 関数

文字列から大文字でない文字を検索し、その文字が見つかった最初の文字位置を返します。

カテゴリ: 文字

返されるデータの種類: DOUBLE

構文

NOTUPPER('*expression*'[, *start*])

必須引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、VARCHAR、NVARCHAR

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

オプション引数

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

NOTUPPER 関数の結果は、有効になっている変換テーブル("TRANTAB= システムオプション" (*SAS 各国語サポート(NLS): リファレンスガイド*)を参照)に直接依存し、ENCODING および LOCALE システムオプションに間接的に依存します。

NOTUPPER 関数は、文字列から大文字でない文字の最初の出現を検索します。対象の文字が検出されると、NOTUPPER は文字列内の該当文字の位置を返します。対象の文字が検出されない場合、NOTUPPER は値 0 を返します。

引数を 1 つのみ使用すると、NOTUPPER は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

次のいずれかの条件が満たされると、NOTUPPER は値 0 を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

NOTUPPER 関数は、文字列から大文字でない文字を検索します。ANYUPPER 関数は、文字列から大文字を検索します。

例

次のプログラムでは、NOTUPPER 関数を使用して文字列から大文字でない文字を検索します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl char(18) string c;
  dcl double j i;
  method run();
    string='Next = _n_ + 12E3;';
    j=0;
    do until(j=0);
      j=notupper(string, j+1);
      if j=0 then put 'The end';
      else do;
        c=substr(string, j, 1);
        put j= c=;
      end;
    end;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
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=;
The end
```

関連項目

関数:

- “ANYUPPER 関数” (304 ページ)

NOTXDIGIT 関数

文字列から 16 進文字でない文字を検索し、その文字が見つかった最初の文字位置を返します。

カテゴリ: 文字

返されるデータの種類: DOUBLE

構文

NOTXDIGIT('*expression*'[, *start*])

必須引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、VARCHAR、NVARCHAR

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

オプション引数

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

NOTXDIGIT 関数は、文字列から数字、大文字または小文字の A、B、C、D、E、F でない文字の最初の出現を検索します。対象の文字が検出されると、NOTXDIGIT は文字列内の該当文字の位置を返します。対象の文字が検出されない場合、NOTXDIGIT は値 0 を返します。

引数を 1 つのみ使用すると、NOTXDIGIT は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

次のいずれかの条件が満たされると、NOTXDIGIT は値 0 を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

NOTXDIGIT 関数は、文字列から 16 進文字でない文字を検索します。ANYXDIGIT 関数は、文字列から 16 進文字を検索します。

例

次のプログラムでは、NOTXDIGIT 関数を使用して文字列から 16 進文字でない文字を検索します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl char(18) string c;
  dcl double j i;
  method run();
    string='Next = _n_ + 12E3;';
    j=0;
    do until(j=0);
      j=notxdigit(string, j+1);
      if j=0 then put 'The end';
      else do;
        c=substr(string, j, 1);
        put j= c=;
      end;
    end;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
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=;
The end
```

関連項目

関数:

- “ANYXDIGIT 関数” (307 ページ)

NPV 関数

パーセントで表す利率を使用して現在正味価値を返します。

カテゴリ:	財務
返されるデータの種類:	DOUBLE

構文

NPV(*r*, *freq*, *c0*, *c1*, ..., *cn*)

必須引数

r
指定したベース期間の利率をパーセントで表した数値。

データの種類	DOUBLE

freq
利率 *r* を使用して指定したベース期間中の支払い回数を表す数値。

範囲	*freq* > 0
データの種類	DOUBLE
注	*freq* = 0 は、連続割引を許可するフラグです。

c0, c1, ..., cn
時間 0,1,...n に発生する現金支出(支払い)または現金流入(収入)を表す数値のキャッシュフローです。これらのキャッシュフローは、等間隔の期間開始時の値であると想定されます。負の値は支払い、正の値は収入、値 0 はその時点でキャッシュフローがないことを表します。*c0* 引数と *c1* 引数は必須です。

データの種類 DOUBLE

比較

NPV 関数は、*r* 引数をパーセントで指定する点を除き、NETPV と同じです。

関連項目

関数:

- “NETPV 関数” (870 ページ)

NULL 関数

引数が null の場合は 1 を返し、引数が null でない場合は 0 を返します。

カテゴリ: 特殊

返されるデータの種類: INTEGER

構文

NULL(*expression*)

必須引数

expression
任意の有効な式を指定します。

データの種類 すべてのデータ型

注 SAS Federation Server LIBNAME エンジンを使用している場合、DS2_SASMISSING 環境変数が TRUE に設定されていると、FedSQL CALL 呼び出しで ANSI null 値が SAS 欠損値に変換されます。

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

NULL 関数は、null 値に対してのみ **1** を返します。SAS 欠損値を含め、非 null 値に対しては **0** を返します。

パッケージインスタンスが存在しない場合、つまりパッケージ変数が null パッケージ参照である場合、NULL 関数は **1** を返します。パッケージ変数がパッケージインスタンスを参照している場合、NULL 関数は **0** を返します。

注: 欠損値と null 値は、SAS モードと ANSI モードで異なる方法で処理されます。モードに応じて、欠損値および null 値が変換される場合があります。

例

次のプログラムは、SAS モードと ANSI モードで NULL がどのように異なるかを示しています。

SAS モードでは、次の行が SAS ログに書き込まれます。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl char(1) a[3];
  dcl double  b[3];
  dcl int c[3];
  dcl int i;
  method run();
    a := ('a', '', NULL);
    b := (1, ., NULL);
    c := (1, NULL, NULL);
    do i = 1 to 3;
      if (null(a[i])) then put a[i]= 'null';
        else put a[i]= 'not null';
      if (null(b[i])) then put b[i]= 'null';
        else put b[i]= 'not null';
      if (null(c[i])) then put c[i]= 'null';
        else put c[i]= 'not null';
    end;
  end;
enddata;
run;
quit;
```

```
a[1]=a not null
b[1]=1 not null
c[1]=1 not null
a[2]=  not null
b[2]=. not null
c[2]= null
a[3]=  not null
b[3]=. not null
c[3]= null
```

ANSI モードでは、次の行が SAS ログに書き込まれます。

```
a[1]=a not null
b[1]=1 not null
c[1]=1 not null
a[2]=  not null
b[2]= null
c[2]= null
a[3]= null
b[3]= null
c[3]= null
```

ANSI モードでは、SAS 欠損値(.)が b := (1, ., NULL);で **b[2]**に割り当てられる前に null に変換されるため、**b[2]**は null であることに注意してください。SAS モードでは、null 値が if (null(a[i])) then put a[i]= 'null';と else put a[i]= 'not null';で **a[3]**と **b[3]**に割り当てられる前に SAS 欠損値(**a[3]**の場合は空白で埋められた文字列、**b[3]**の場合は欠損.)に変換されるため、**a[3]**と b[3]は null ではありません。

関連項目

- “DS2 による null および SAS 欠損値の処理方法” (*SAS DS2 プログラマガイド*)

関数:

- “MISSING 関数” (855 ページ)
- “N 関数” (867 ページ)
- “NMISS 関数” (872 ページ)

NWKDOM 関数

指定した月および年の *n* 番目に発生する曜日の日付を返します。

カテゴリ:	日付と時間
返されるデータの種類:	DOUBLE

構文

NWKDOM(*n*, *weekday*, *month*, *year*)

必須引数

n
指定した日を含む月の週を数値で指定します。

範囲 1-5

データの種類 DOUBLE

ヒント *N*=5 は、指定した日がその月の最終週に発生することを示します。*n*=4 と *n*=5 の結果が同じになる場合もあります。

weekday
週の曜日に対応する数値を指定します。

範囲 1-7

データの種類 DOUBLE

ヒント 日曜日は、週の最初の日として見なされ、*weekday* の値は 1 になります。

month
年の月に対応する数値を指定します。

範囲 1-12

データの種類 DOUBLE

year
4 桁のカレンダー年を指定します。

データの種類 DOUBLE

詳細

NWKDOM 関数は、指定した月および年の *n* 番目の曜日の SAS 日付値を返します。DATE9.形式などの有効な SAS 日付形式を使用して、カレンダー日付を表示します。月の最後に発生する特定の曜日の場合、*n*=5 を指定できます。

n=5 と *n*=4 の結果が同じになる場合もあります。要求された曜日がその月に 4 回しか発生しない場合にこのような結果になります。たとえば、1 月の初日が日曜日の場合、日曜日、月曜日および火曜日は 5 回発生しますが、水曜日、木曜日、金曜日、土曜日は 4 回しか発生しません。この場合、水曜日、木曜日、金曜日、土曜日に対して *n*=5 または *n*=4 を指定すると、結果が同じになります。

その年がうるう年でない場合、2 月は 28 日間あり、週の各曜日は 4 回発生します。この場合、すべての曜日で *n*=5 と *n*=4 の結果が同じになります。

比較

NWKDOM 関数では、*weekday* の値は、日曜日から始まる週の曜日の数値に対応しています。この値は、WEEKDAY 関数で使用される値と同じで、日曜日=1 (以下同様)になります。*month* の値は、1 月から始まる年の月の数値に対応します。この値は、MONTH 関数で使用される値と同じで、1 月=1 (以下同様)になります。

NWKDOM 関数を使用して、HOLIDAY 関数で定義されていないイベントを計算できます。たとえば、大学の卒業が常に 6 月の第 1 土曜日にスケジュールされている場合、次のステートメントを使用してその日付を計算できます。UnivGrad = nwkdom(1, 7, 6, *year*);

例

例 1: 日付値を返す

次のプログラムでは、NWKDOM 関数を使用して、指定した月および年に発生する特定の曜日の日付を返します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double a b c d e f;
  method run();
    /* 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.;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
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
```

例 2: 5 月の最終月曜日の日付を返す

次のプログラムでは、2012 年 5 月の最終月曜日に対応する日付を返します。

```
data _null_;
  dcl double x;
  method init();
    /* The last Monday in May. */
    x=nwkdom(5,2,5,2012);
    put x date9.;
  end;
enddata;
run;
```

SAS は次の出力をログに書き込みます。

```
28MAY2012
```

関連項目

関数:

- “HOLIDAY 関数” (695 ページ)
- “INTNX 関数” (755 ページ)
- “MONTH 関数” (863 ページ)
- “WEEKDAY 関数” (1259 ページ)

ORDINAL 関数

値のリストを並べ替え、リスト内の位置に基づく値を返します。

カテゴリ:	記述統計
返されるデータの種類:	DOUBLE

構文

ORDINAL(*position*, *expression-1, expression-2* [, ...*expression-n*])

必須引数

position
引数リストの要素数以下の整数を指定します。

要件 *position* は正数にする必要があります。

データの種類 DOUBLE

expression
数値に評価される有効な式を指定します。

要件 少なくとも 2 つの引数が必要です。

データの種類 DOUBLE

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

詳細

ORDINAL 関数はリストを並べ替えて、*position* で指定されたリスト内の引数を返します。欠損値は下位に並べ替えられ、数値の前に配置されます。

比較

ORDINAL 関数は、null 値、欠損値、非欠損値、非 null 値の両方を数えますが、SMALLEST 関数は非 null 値および非欠損値のみを数えます。

例

次のプログラムは、ORDINAL 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double x;
  method run();
    x=ordinal(4,1,2,3,-4,5,6,7);
    put x;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
3
```

PCTL 関数

パーセントに対応するパーセント点を返します。

カテゴリ: 記述統計

返されるデータの種類: DOUBLE

構文

PCTL[*n*](*percentage*, *expression*[, ...*expression*])

必須引数

percentage
計算するパーセント点を指定します。

データの種類 DOUBLE

ヒント *percentage* は、0≤*percentage*≤100 の数値です。

percentage は、0 ≤ *percentage* ≤ 100 の数値です。

オプション引数

n
計算するパーセント点の定義を指定する 1 から 5 までの数字です。

デフォルト 定義 5

データの種類 DOUBLE

参照項目: *Base SAS プロシジャガイド*の"Methods for Computing Quantile Statistics"の表にある QNTLDEF= (エイリアス PCTLDEF=)の説明。

expression
数値に評価される有効な式を指定します。その値はパーセント点計算で計算されます。

データの種類 DOUBLE

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

詳細

PCTL 関数は、パーセントに対応する非 null 値または非欠損値のパーセント点を返します。*percentage* が null 値または欠損値、ゼロ未満、あるいは 100 を超える場合、PCTL 関数はエラーメッセージを生成します。

例

次のプログラムは、PCTL 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double lower_quartile percentile_def2 lower_tertile percentile_def3;
  dcl double median upper_tertile upper_quartile;
  method run();
    lower_quartile=pctl(25, 2, 4, 1, 3);
    put lower_quartile=;
    percentile_def2=pctl2(25, 2, 4, 1, 3);
    put percentile_def2=;
    lower_tertile=pctl(100/3, 2, 4, 1, 3);
    put lower_tertile=;
    percentile_def3=pctl3(100/3, 2, 4, 1, 3);
    put percentile_def3=;
    median=pctl(50, 2, 4, 1, 3);
    put median=;
    upper_tertile=pctl(200/3, 2, 4, 1, 3);
    put upper_tertile=;
    upper_quartile=pctl(75, 2, 4, 1, 3);
    put upper_quartile=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
lower_quartile=1.5
percentile_def2=1
lower_tertile=2
percentile_def3=2
median=2.5
upper_tertile=3
upper_quartile=3.5
```

PDF 関数

確率密度(質量)分布の値を返します。

カテゴリ: 確率

別名: PMF

データの種類: DOUBLE

構文

PDF(*'distribution'*, *quantile* [, *parameter-1, ..., parameter-k*])

必須引数

distribution
分布を特定する文字定数、変数または式です。有効な分布は次のとおりです。

分布	引数
Bernoulli	'BERNOULLI'
ベータ	'BETA'
二項	'BINOMIAL'
Cauchy	'CAUCHY'
カイ 2 乗	'CHISQUARE'
Conway-Maxwell-Poisson	'CONMAXPOI'
指数	'EXPONENTIAL'
F	'F'
ガンマ	'GAMMA'
一般化 Poisson	'GENPOISSON'
幾何	'GEOMETRIC'
超幾何	'HYPERGEOMETRIC'

分布	引数
Laplace	'LAPLACE'
ロジスティック	'LOGISTIC'
対数正規	'LOGNORMAL'
負の二項	'NEGBINOMIAL'
正規	'NORMAL'|'GAUSS'
正規混合	'NORMALMIX'
パレート	'PARETO'
Poisson	'POISSON'
T	'T'
Tweedie	'TWEEDIE'
一様	'UNIFORM'
Wald (逆ガウス)	'WALD'|'IGAUSS'
Weibull	'WEIBULL'

注　T、F および NORMALMIX を除き、最初の 4 文字で分布を識別できます。

quantile
確率変数の値を指定する数値定数、変数または式です。

データの種類　DOUBLE

parameter-1, ..., parameter-k
特定の分布に適した*形状*、*位置*または*尺度*パラメーターの値を指定する数値定数、変数または式です(省略可能)。

データの種類　DOUBLE

関連項目

関数:

- “CDF 関数” (359 ページ)

- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

参考文献

Shmueli, G., T. Minka,, S. Borle, and P. Boatwright. 2005. “A Useful Distribution for Fitting Discrete Data: Revival of the Conway-Maxwell-Poisson Distribution.” *Applied Statistics* : 54:127–142.

PDF BERNOULLI 分布 関数

Bernoulli 確率密度(質量)分布の値を返します。

カテゴリ:	確率
別名:	PMF
返されるデータの種類:	DOUBLE

構文

PDF ('BERNOULLI', *x*, *p*)

必須引数

x
確率変数を指定する数値定数、変数または式です。

データの種類 DOUBLE

p
成功確率を指定する数値定数、変数または式です。

範囲 $0 \le p \le 1$

データの種類 DOUBLE

詳細

Bernoulli 分布の PDF 関数は、*p* に等しい成功確率で確率密度関数を返します。この PDF 関数は、値 *x* で評価されます。

$$PDF('BERN', x, p) = \begin{cases} 0 & x < 0 \\ 1 - p & x = 0 \\ 0 & 0 < x < 1 \\ p & x = 1 \\ 0 & x > 1 \end{cases}$$

注: この分布には、*位置*または*尺度*パラメーターはありません。

例

次のプログラムは、PDF Bernoulli 分布関数を示しています。

注: このサンプルコードでは、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double y;
  method init();
    y=pdf('BERN',0,.25);
    put 'Bern dist:     ' y;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
Bern dist:     0.75
```

関連項目

関数:

- “CDF BERNOULLI 分布 関数” (362 ページ)
- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “QUANTILE 関数” (1052 ページ)

- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

PDF BETA 分布 関数

ベータ確率密度(質量)分布の値を返します。

カテゴリ:	確率
別名:	PMF
返されるデータの種類:	DOUBLE

構文

PDF ('BETA', *x*, *a*, *b* [, *l*, *r*])

必須引数

x
確率変数を指定する数値定数、変数または式です。

データの種類	DOUBLE

a
形状パラメーターを指定する数値定数、変数または式です。

範囲	$a > 0$
データの種類	DOUBLE

b
形状パラメーターを指定する数値定数、変数または式です。

範囲	$b > 0$
データの種類	DOUBLE

オプション引数

l
左位置パラメーターを指定する数値定数、変数または式です。

デフォルト	0
データの種類	DOUBLE

r

右位置パラメーターを指定する数値定数、変数または式です。

デフォルト　1

範囲　*r* > *l*

データの種類　DOUBLE

詳細

ベータ分布の PDF 関数は、形状パラメーター *a* と *b* で確率密度関数を返します。この PDF 関数は、値 *x* で評価されます。

$$PDF('BETA', x, a, b, l, r) = \begin{cases} 0 & x < l \\ \frac{1}{\beta(a,b)} \frac{(x-l)^{a-1}(r-x)^{b-1}}{(r-l)^{a+b-1}} & l \le x \le r \\ 0 & x > r \end{cases}$$

注: 数量$\frac{x-l}{r-l}$は、$\varepsilon \le \frac{x-l}{r-l} \le 1 - 2\varepsilon$へと強制されます。

例

次のプログラムは、PDF ベータ分布関数を示しています。

注: このサンプルコードでは、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double y;
  method init();
    y=pdf('BETA', 0.2, 3, 4);
    put 'Beta dist:     ' y;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
Beta dist:     1.2288
```

関連項目

関数:

- “CDF BETA 分布 関数” (363 ページ)
- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

PDF BINOMIAL 分布 関数

二項確率密度(質量)分布の値を返します。

カテゴリ:	確率
別名:	PMF
返されるデータの種類:	DOUBLE

構文

PDF ('BINOMIAL', *m*, *p*, *n*)

必須引数

m
成功数を数える確率変数です。この引数は整数にする必要があります。

範囲	*m*=0, 1, ...
データの種類	DOUBLE

p
成功確率を指定する数値定数、変数または式です。

範囲	$0 \le p \le 1$
データの種類	DOUBLE

n
独立 Bernoulli 試行数を数えるパラメーターです。この引数は整数にする必要があります。

範囲	*n*=0, 1, ...
データの種類	DOUBLE

詳細

二項分布の PDF 関数は、パラメーター *p* と *n* で確率密度関数を返します。この PDF 関数は、値 *m* で評価されます。

$$PDF('BINOM', m, p, n) = \begin{cases} 0 & m < 0 \\ \binom{n}{m} p^m (1-p)^{n-m} & 0 \le m \le n \\ 0 & m > n \end{cases}$$

注: 二項分布には、*位置*または*尺度*パラメーターはありません。

例

次のプログラムは、PDF 二項分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double y;
  method init();
    y=pdf('BINOM',4,.5,10);
    put 'Binom dist:    ' y;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
Binom dist:    0.205078125
```

関連項目

関数:

- “CDF BINOMIAL 分布 関数” (366 ページ)
- “LOGCDF 関数” (825 ページ)

- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

PDF CAUCHY 分布 関数

Cauchy 確率密度(質量)分布の値を返します。

カテゴリ:	確率
別名:	PMF
返されるデータの種類:	DOUBLE

構文

PDF ('CAUCHY', *x* [, *θ*, *λ*])

必須引数

x
確率変数を指定する数値定数、変数または式です。

データの種類 DOUBLE

オプション引数

θ
位置パラメーターを指定する数値定数、変数または式です。

デフォルト 0

データの種類 DOUBLE

λ
尺度パラメーターを指定する数値定数、変数または式です。

デフォルト 1

範囲 λ > 0

データの種類 DOUBLE

詳細

Cauchy 分布の PDF 関数は、位置パラメーターは θ、尺度パラメーターは λ の確率密度関数を返します。PDF 関数は、値 *x* で評価されます。

$$PDF('CAUCHY', x, \theta, \lambda) = \frac{1}{\pi}\left(\frac{\lambda}{\lambda^2 + (x-\theta)^2}\right)$$

例

次のプログラムは、PDF Cauchy 分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double y;
  method init();
    y=pdf('CAUCHY',2);
    put 'Cauchy dist:    ' y;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
Cauchy dist:    0.06366197723675
```

関連項目

関数:

- “CDF CAUCHY 分布 関数” (368 ページ)
- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

PDF カイ 2 乗分布 関数

カイ 2 乗確率密度(質量)分布の値を返します。

カテゴリ:	確率
別名:	PMF
返されるデータの種類:	DOUBLE

構文

PDF ('CHISQUARE', *x*, *df* [, *nc*])

必須引数

x
確率変数を指定する数値定数、変数または式です。

データの種類 DOUBLE

df
自由度を指定する数値定数、変数または式です。

範囲 $df > 0$

データの種類 DOUBLE

オプション引数

nc
省略可能な非心度パラメーターを指定する数値定数、変数または式です。

範囲 $nc \geq 0$

データの種類 DOUBLE

詳細

カイ 2 乗分布の PDF 関数は、カイ 2 乗分布(自由度は *df*、非心度パラメーターは *nc*)の確率密度関数を返します。PDF 関数は、値 *x* で評価されます。この関数では、整数以外の自由度を使用できます。*nc* が省略されているかゼロの場合、心度カイ 2 乗分布の値が返されます。

$$PDF('CHISQ', x, \nu, \lambda) = \begin{cases} 0 & x < 0 \\ \sum_{j=0}^{\infty} e^{-\frac{\lambda}{2}} \frac{\left(\frac{\lambda}{2}\right)^j}{j!} p_c(x, \nu + 2j) & x \geq 0 \end{cases}$$

この式では、$p_c(.,.)$は、心度カイ 2 乗分布の密度を示しています。

$$p_c(x, a) = \frac{1}{2} p_g\left(\frac{x}{2}, \frac{a}{2}\right)$$

この式では、$p_g(y,b)$はガンマ分布からの密度です。

$$p_g(y, b) = \frac{1}{\Gamma(b)} e^{-y} y^{b-1}$$

例

次のプログラムは、PDF カイ 2 乗分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double y;
  method init();
    y=pdf('CHISQ',11.264,11);
    put 'Chisq dist:    ' y;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
Chisq dist:    0.08168618682435
```

関連項目

関数:

- “CDF カイ 2 乗分布 関数” (370 ページ)
- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

PDF Conway-Maxwell-Poisson 分布 関数

Conway-Maxwell-Poisson 確率密度(質量)分布の値を返します。

カテゴリ:	確率
別名:	PMF
返されるデータの種類:	DOUBLE

構文

PDF('CONMAXPOI',*y*,*λ*,*ν*)

必須引数

y
カウントを表す負でない整数を指定する数値定数、変数または式です。

データの種類 DOUBLE

λ
Poisson 平均パラメーターに似た位置パラメーターを指定する数値定数、変数または式です。

データの種類 DOUBLE

ν
ばらつきのパラメーターを指定する数値定数、変数または式です。

データの種類 DOUBLE

詳細

Conway-Maxwell-Poisson (CMP)分布は Poisson 分布の一般化です。これにより、過小分散データと過大分散データをモデル化できます。CMP 分布は、次の式に従って定義されます。

$$P\left(Y = y\,;\lambda,\nu\right) = \frac{1}{Z(\lambda,\nu)}\frac{\lambda^{y}}{(y!)^{\nu}} \qquad y = 0, 1, 2, \ldots$$

正規化因子は次の式で表されます。

$$Z\left(\lambda,\nu\right) = \sum_{n=0}^{\infty}\frac{\lambda^{n}}{(n!)^{\nu}}$$

λ および ν は負ではなく、同時にゼロにもなりません。

追加パラメーター ν は、分布の裾挙動のモデル化における柔軟性が考慮されています。ν=1 の場合、比は Poisson 分布の減衰率に等しくなります。ν<1 の場合、減衰率が減少し、Poisson 分布より裾が長いプロセス(過大分散データ)をモデル化できます。ν>1 の場合、減衰率が非線形方法で増加し、したがって、分布の裾が短くなります(過小分散データ)。

Conway-Maxwell-Poisson 分布には、いくつか特殊なケースがあります。λ<1 および ν→∞の場合、結果的に Conway-Maxwell-Poisson 分布が Bernoulli 分布になります。このケースでは、データが取り得る値は 0 および 1 です。これは極過小分散を表します。ν=1 の場合、等分散の特性を持つ Poisson 分布に戻ります。ν=0 および λ<1 の場合、正規化因子は収束性であり、次の幾何級数を形成します。

$$Z\left(\lambda,0\right)=\frac{1}{1-\lambda}$$

確率密度関数は、次の式で表されます。

$$P\left(Y=y\,;\lambda,\nu=0\right)=\left(1-\lambda\right)\lambda^{y}$$

幾何分布は、著しい過大分散のケースを表します。

Conway-Maxwell-Poisson モデルの平均、分散およびばらつき

Conway-Maxwell-Poisson 分布の平均および分散は、次の式で定義されます。

$$E\left[Y\right]=\frac{\partial\,1n\;Z}{\partial\,1n\,\lambda}$$

$$V\left[Y\right]=\frac{\partial^{2}\,1n\,Z}{\partial^{2}\,1n\,\lambda}$$

Conway-Maxwell-Poisson 分布は、そのパラメーター λ および ν に関して、その積率の閉形式を持ちません。ただし、積率は近似できます。(Conway-Maxwell-Poisson 分布および離散データの詳細については、この関数の最後にあるリファレンスセクションを参照してください。)次の式で示すように、Z に対する漸近表現を使用すると、E(Y)および V(Y)を導き出せます。

$$E\left[Y\right]\approx\lambda^{1/\nu}+\frac{1}{2\nu}-\frac{1}{2}$$

$$V\left[Y\right]\approx\frac{1}{\nu}\lambda^{1/\nu}$$

Conway-Maxwell-Poisson モデルでは、対数展開を使用して無限級数の総和が評価されます。(Conway-Maxwell-Poisson 分布および離散データの詳細については、この関数の最後にあるリファレンスセクションを参照してください。)Conway-Maxwell-Poisson モデルについては、次のように平均および分散が計算されます。

$$E\left(Y\right)=\frac{1}{Z(\lambda,\nu)}\sum_{j=0}^{\infty}\frac{j\lambda^{j}}{(j!)^{\nu}}$$

$$V\left(Y\right)=\frac{1}{Z(\lambda,\nu)}\sum_{j=0}^{\infty}\frac{j^{2}\lambda^{j}}{(j!)^{\nu}}-E(Y)^{2}$$

ばらつきは、次のように定義されます。

$$D\left(Y\right) = \frac{V(Y)}{E(Y)}$$

例

次のプログラムは、PDF Conway-Maxwell-Poisson 分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double y;
  method init();
    y=pdf('CONMAXPOI', .2, 2.3, .4);
    put 'ConMaxPoi dist:     ' y;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
ConMaxPoi dist:     0.00977326351239
```

関連項目

関数:

- “CDF Conway-Maxwell-Poisson 分布 関数” (372 ページ)
- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

PDF EXPONENTIAL 分布 関数

指数確率密度(質量)分布の値を返します。

カテゴリ: 確率

別名: PMF

返されるデータの種類: DOUBLE

構文

PDF ('EXPONENTIAL', *x* [, *λ*])

必須引数

x
確率変数を指定する数値定数、変数または式です。

データの種類 DOUBLE

オプション引数

λ
尺度パラメーターを指定する数値定数、変数または式です。

デフォルト 1

範囲 λ > 0

データの種類 DOUBLE

詳細

指数分布の PDF 関数は、指数分布(尺度パラメーターは λ)の確率密度関数を返します。PDF 関数は、値 *x* で評価されます。

$$PDF('EXPO',x,\lambda)=\begin{cases}0 & x<0\\ \frac{1}{\lambda}exp\left(-\frac{x}{\lambda}\right) & x\geq 0\end{cases}$$

例

次のプログラムは、PDF 指数分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double y;
  method init();
    y=pdf('EXPO',1);
```

```
      put 'Expo dist:     ' y;
    end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
Expo dist:     0.36787944117144
```

関連項目

関数:

- “CDF 指数分布 関数” (373 ページ)
- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

PDF F 分布 関数

F 確率密度(質量)分布の値を返します。

カテゴリ:	確率
別名:	PMF
返されるデータの種類:	DOUBLE

構文

PDF ('F', *x*, *ndf*, *ddf* [, *nc*])

必須引数

x
確率変数を指定する数値定数、変数または式です。

データの種類 DOUBLE

ndf
分子の自由度を指定する数値定数、変数または式です。

範囲 *ndf*> 0

データの種類 DOUBLE

ddf
分母の自由度を指定する数値定数、変数または式です。

範囲 *ddf* > 0

データの種類 DOUBLE

オプション引数

nc
省略可能な非心度パラメーターを指定する数値定数、変数または式です。

範囲 *nc* ≥ 0

データの種類 DOUBLE

詳細

F 分布の PDF 関数は、*F* 分布(分子の自由度は *ndf*、分母の自由度は *ddf*、非心度パラメーターは *nc*)の確率密度関数を返します。PDF 関数は、値 *x* で評価されます。この PDF 関数では、*ndf* および *ddf* に対して整数以外の自由度を使用できます。*nc* が除外されているかゼロの場合、心度 *F* 分布の値が返されます。次の式で、$\nu_1 = ndf$として、$\nu_2 = ddf$として、そして$\lambda = nc$とします。次の式は、*F* 分布の PDF 関数を表しています。

$$PDF('F', x, \nu_1, \nu_2, \lambda) = \begin{cases} 0 & x < 0 \\ \sum_{j=0}^{\infty} e^{-\frac{\lambda}{2}} \frac{\left(\frac{\lambda}{2}\right)^j}{j!} p_f(f, \nu_1 + 2j, \nu_2) & x \geq 0 \end{cases}$$

この式では、$p_f(f,u_1,u_2)$は、次の式を使用した心度 *F* 分布の密度です。

$$p_f(f, u_1, u_2) = p_B\left(\frac{u_1 f}{uf + u_2}, \frac{u_1}{2}, \frac{u_2}{2}\right) \frac{u_1 u_2}{(u_2 + u_1 f)^2}$$

この式では、$p_B(x,a,b)$は、標準ベータ分布の密度です。

注: *F* 分布には、*位置*または*尺度*パラメーターはありません。

例

次のプログラムは、PDF F 分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double y;
  method init();
    y=pdf('F',3.32,2,3);
    put 'F dist:        ' y;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
F dist:       0.05402696258579
```

関連項目

関数:

- “CDF F 分布 関数” (375 ページ)
- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

PDF GAMMA 分布 関数

ガンマ確率密度(質量)分布の値を返します。

カテゴリ:	確率
別名:	PMF
返されるデータの種類:	DOUBLE

構文

PDF ('GAMMA', *x*, *α* [, *λ*])

必須引数

x
確率変数を指定する数値定数、変数または式です。

データの種類 DOUBLE

α
形状パラメーターを指定する数値定数、変数または式です。

範囲 $\alpha > 0$

データの種類 DOUBLE

オプション引数

λ
尺度パラメーターを指定する数値定数、変数または式です。

デフォルト 1

範囲 $\lambda > 0$

データの種類 DOUBLE

詳細

ガンマ分布の PDF 関数は、ガンマ分布(形状パラメーターは *α*、尺度パラメーターは λ)の確率密度関数を返します。PDF 関数は、値 *x* で評価されます。

$$PDF('GAMMA', x, a, \lambda) = \begin{cases} 0 & x < 0 \\ \frac{1}{\lambda^a \Gamma(a)} x^{a-1} exp\left(-\frac{x}{\lambda}\right) & x \geq 0 \end{cases}$$

例

次のプログラムは、PDF ガンマ分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double y;
```

```
  method init();
    y=pdf('GAMMA',1,3);
    put 'Gamma dist:    ' y;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
Gamma dist:    0.18393972058572
```

関連項目

関数:

- “CDF GAMMA 分布 関数” (377 ページ)
- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “SDF 関数” (1120 ページ)

PDF 一般化 Poisson 分布 関数

一般化 Poisson 確率密度(質量)分布の値を返します。

カテゴリ:	確率
別名:	PMF
返されるデータの種類:	DOUBLE

構文

PDF ('GENPOISSON', *x*, *θ*, *η*)

必須引数

x
確率変数を指定する数値定数、変数または式です。この引数は整数にする必要があります。

データの種類 DOUBLE

θ
形状パラメーターを指定する数値定数、変数または式です。

範囲 <10^5 かつ>0

データの種類 DOUBLE

η
形状パラメーターを指定する数値定数、変数または式です。

範囲 ≥0 かつ<0.95

データの種類 DOUBLE

ヒント η =0 の場合、平均と分散が θ の Poisson 分布になります。η>0 の場合、平均は$\theta \div (1-\eta)$および分散は$\theta \div (1-\eta)^3$になります。

詳細

一般化 Poisson 分布の確率質量関数の次のとおりです。

$$f(x;\theta,\eta) = \theta(\theta+\eta x)^{x-1}e^{-\theta-\eta x}/x!, \quad x = 0,1,2,..., \qquad \theta > 0,\ 0 \le \eta < 1$$

例

次のプログラムは、PDF 一般化 Poisson 分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double y;
  method init();
    y=pdf('GENPOISSON', 9, 1, .7);
    put 'GENPOISSON dist:     ' y;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
GENPOISSON dist:     0.01501309145504
```

関連項目

関数:

- “CDF 一般化 Poisson 分布 関数” (379 ページ)
- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

PDF GEOMETRIC 分布 関数

幾何確率密度(質量)分布の値を返します。

カテゴリ:	確率
別名:	PMF
返されるデータの種類:	DOUBLE

構文

PDF ('GEOMETRIC', *m*, *p*)

必須引数

m
初めて成功する前の失敗数を指定する数値定数、変数または式です。

範囲	$m \geq 0$
データの種類	DOUBLE

p
成功確率を指定する数値定数、変数または式です。

範囲	$0 \leq p \leq 1$
データの種類	DOUBLE

詳細

幾何分布の PDF 関数は、パラメーター *p* をもつ確率密度関数を返します。この PDF 関数は、値 *m* で評価されます。

$$PDF('GEOM', m, p) = \begin{cases} 0 & m < 0 \\ p(1-p)^m & m \geq 0 \end{cases}$$

注: この分布には、*位置*または*尺度*パラメーターはありません。

例

次のプログラムは、PDF 幾何分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double y;
  method init();
    y=pdf('GEOMETRIC', 5, .3);
    put 'geometric dist:     ' y;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
geometric dist:     0.050421
```

関連項目

関数:

- “CDF GEOMETRIC 分布 関数” (381 ページ)
- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

PDF 超幾何分布 関数

超幾何確率密度(質量)分布の値を返します。

カテゴリ:	確率
別名:	PMF
返されるデータの種類:	DOUBLE

構文

PDF ('HYPER', *x*, *N*, *R*, *n* [, *o*])

必須引数

x
確率変数を指定する数値定数、変数または式です。この引数は整数にする必要があります。

データの種類 DOUBLE

N
母集団サイズパラメーターを指定する数値定数、変数または式です。この引数は整数にする必要があります。

範囲 *N*=1, 2, ...

データの種類 DOUBLE

R
対象カテゴリの項目数を指定する数値定数、変数または式です。この引数は整数にする必要があります。

範囲 *R*=0, 1, ..., *N*

データの種類 DOUBLE

n
サンプルサイズパラメーターを 指定する数値定数、変数または式です。この引数は整数にする必要があります。

範囲 *n*=1, 2, ..., *N*

データの種類 DOUBLE

オプション引数

o

省略可能なオッズ比パラメーターを指定する数値定数、変数または式です。

範囲 *o* > 0

データの種類 DOUBLE

詳細

超幾何分布の PDF 関数は、拡張超幾何分布(母集団サイズは *N*、項目数は *R*、サンプルサイズは *n*、オッズ比は *o*)の確率密度関数を返します。この PDF 関数は、値 *x* で評価されます。*o* が除外されているか 1 の場合、一般超幾何分布の値が返されます。

$$PDF\ ('HYPER', x, N, R, n, o) = \begin{cases} 0 & x < max(0, R+n-N) \\ \dfrac{\binom{R}{x}\binom{N-R}{n-x}o^x}{\sum_{j=max(0,R+n-N)}^{min(R,n)}\binom{R}{j}\binom{N-R}{n-j}o^j} & max(0, R+n-N) \le x \le min(R, n) \\ 0 & x > min(R, n) \end{cases}$$

例

次のプログラムは、PDF 超幾何分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double y;
  method init();
    y=pdf('HYPER', 2, 200, 50, 10);
    put 'Hyper dist:     ' y;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
Hyper dist:     0.28685059643368
```

関連項目

関数:

- “CDF HYPERGEOMETRIC 分布 関数” (383 ページ)
- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

PDF LAPLACE 分布 関数

Laplace 確率密度(質量)分布の値を返します。

カテゴリ:	確率
別名:	PMF
返されるデータの種類:	DOUBLE

構文

PDF ('LAPLACE', *x* [, *θ*, *λ*])

必須引数

x
確率変数を指定する数値定数、変数または式です。

データの種類 DOUBLE

オプション引数

θ
位置パラメーターを指定する数値定数、変数または式です。

デフォルト 0

データの種類 DOUBLE

λ
尺度パラメーターを指定する数値定数、変数または式です。

デフォルト	1
範囲	λ > 0
データの種類	DOUBLE

詳細

Laplace 分布の PDF 関数は、Laplace 分布(位置パラメーターは θ、尺度パラメーターは λ)の確率密度関数を返します。PDF 関数は、値 *x* で評価されます。

$$PDF('LAPLACE', x, \theta, \lambda) = \frac{1}{2\lambda} exp\left(- \frac{|x - \theta|}{\lambda} \right)$$

例

次のプログラムは、PDF Laplace 分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double y;
  method init();
    y=pdf('LAPLACE', 1);
    put 'Laplace dist:   ' y;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
Laplace dist:   0.18393972058572
```

関連項目

関数:

- “CDF LAPLACE 分布 関数” (385 ページ)
- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “QUANTILE 関数” (1052 ページ)

- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

PDF LOGISTIC 分布 関数

ロジスティック確率密度(質量)分布の値を返します。

カテゴリ: 確率

別名: PMF

返されるデータの種類: DOUBLE

構文

PDF ('LOGISTIC', *x* [, *θ*, *λ*])

必須引数

x
確率変数を指定する数値定数、変数または式です。

データの種類 DOUBLE

オプション引数

θ
位置パラメーターを指定する数値定数、変数または式です。

デフォルト 0

データの種類 DOUBLE

λ
尺度パラメーターを指定する数値定数、変数または式です。

デフォルト 1

範囲 λ > 0

データの種類 DOUBLE

詳細

ロジスティック分布の PDF 関数は、ロジスティック分布(位置パラメーターは θ、尺度パラメーターは λ)の確率密度関数を返します。PDF 関数は、値 *x* で評価されます。

$$PDF('LOGISTIC', x, \theta, \lambda) = \frac{exp\left(-\frac{x-\theta}{\lambda}\right)}{\lambda\left(1+exp\left(-\frac{x-\theta}{\lambda}\right)\right)^2}$$

例

次のプログラムは、PDF ロジスティック分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double y;
  method init();
    y=pdf('LOGISTIC', 1);
    put 'Logistic dist:   ' y;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
Logistic dist:   0.19661193324148
```

関連項目

関数:

- “CDF LOGISTIC 分布 関数” (387 ページ)
- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

PDF LOGNORMAL 分布 関数

対数正規確率密度(質量)分布の値を返します。

カテゴリ:	確率
別名:	PMF
返されるデータの種類:	DOUBLE

構文

PDF ('LOGNORMAL', *x* [, *θ*, *λ*])

必須引数

x
確率変数を指定する数値定数、変数または式です。

データの種類 DOUBLE

オプション引数

θ
対数尺度パラメーターを指定する数値定数、変数または式です。exp(θ)は尺度パラメーターです。

デフォルト 0

データの種類 DOUBLE

λ
形状パラメーターを指定する数値定数、変数または式です。

デフォルト 1

範囲 λ > 0

データの種類 DOUBLE

詳細

対数正規分布の PDF 関数は、対数正規分布(対数尺度パラメーターは θ、形状パラメーターは λ)の確率密度関数を返します。PDF 関数は、値 *x* で評価されます。

$$PDF('LOGN', x, \theta, \lambda) = \begin{cases} 0 & x \le 0 \\ \frac{1}{x\lambda\sqrt{2\,\pi}}\ exp\left(-\frac{(log(x)-\theta)^2}{2\lambda^2}\right) & x > 0 \end{cases}$$

例

次のプログラムは、PDF 対数正規分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double y;
  method init();
    y=pdf('LOGNORMAL', 1);
    put 'LogNormal dist:     ' y;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
LogNormal dist:     0.39894228040143
```

関連項目

関数:

- “CDF LOGNORMAL 分布 関数” (389 ページ)
- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

PDF NEGBINOMIAL 分布 関数

負の二項確率密度(質量)分布の値を返します。

カテゴリ:	確率
別名:	PMF
返されるデータの種類:	DOUBLE

構文

PDF ('NEGBINOMIAL', *m*, *p*, *n*)

必須引数

m
失敗数を数える確率変数を指定する、数値定数、変数または式です。この引数は正の整数にする必要があります。

範囲	*m*=0, 1, ...
データの種類	DOUBLE

p
成功確率を指定する数値定数、変数または式です。

範囲	$0 \le p \le 1$
データの種類	DOUBLE

n
成功数を数える値を指定する数値定数、変数または式です。

範囲	*n*>0
データの種類	DOUBLE

詳細

負の二項分布の PDF 関数は、負の二項分布(成功確率は *p*、成功数は *n*)の確率密度関数を返します。この PDF 関数は、値 *m* で評価されます。

$$PDF('NEGB', m, p, n) = \begin{cases} 0 & m < 0 \\ \binom{n+m-1}{n-1} p^n (1-p)^m & m \ge 0 \end{cases}$$

注: 負の二項分布には、*位置*または*尺度*パラメーターはありません。

例

次のプログラムは、PDF 負の二項分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double y;
  method init();
    y=pdf('NEGB',1,.5,2);
    put 'NegB dist:      ' y;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
NegB dist:     0.25
```

関連項目

関数:

- “CDF NEGBINOMIAL 分布 関数” (391 ページ)
- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “SDF 関数” (1120 ページ)

PDF NORMAL 分布 関数

正規確率密度(質量)分布の値を返します。

カテゴリ:	確率
別名:	PMF
返されるデータの種類:	DOUBLE

構文

PDF ('NORMAL', *x* [, *θ*, *λ*])

必須引数

x
確率変数を指定する数値定数、変数または式です。

データの種類 DOUBLE

オプション引数

θ
位置パラメーターを指定する数値定数、変数または式です。

デフォルト 0

データの種類 DOUBLE

λ
尺度パラメーターを指定する数値定数、変数または式です。

デフォルト 1

範囲 λ > 0

データの種類 DOUBLE

詳細

正規分布の PDF 関数は、正規分布(位置パラメーターは θ、尺度パラメーターは λ)の確率密度関数を返します。PDF 関数は、値 *x* で評価されます。

$$PDF('NORMAL', x, \theta, \lambda) = \frac{1}{\lambda\sqrt{2\,\pi}}\ exp\left(-\frac{(x-\theta)^2}{2\lambda^2}\right)$$

例

次のプログラムは、PDF 正規分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double y;
  method init();
```

```
    y=pdf('NORMAL',1.96);
    put 'Normal dist:    ' y;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
Normal dist:    0.05844094433345
```

関連項目

関数:

- “CDF NORMAL 分布 関数” (393 ページ)
- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

PDF NORMALMIX 分布 関数

正規混合確率密度(質量)分布の値を返します。

カテゴリ:	確率
別名:	PMF
返されるデータの種類:	DOUBLE

構文

PDF ('NORMALMIX', *x*, *n*, p_1, p_2, ... p_n, m_1, m_2, ... m_n, s_1, s_2, ... s_n)

必須引数

x
確率変数を指定する数値定数、変数または式です。

データの種類 DOUBLE

n
混合数を指定する数値定数、変数または式です。この引数は整数にする必要があります。

範囲 *n*=1, 2, ...

データの種類 DOUBLE

p_i
n 個の比率$p_1, p_2, \ldots, p_n$を指定する、数値の定数、変数または式のリストです。ここで、$\sum_{i=1}^{i=n} p_i = 1$です。

範囲 *p*=0, 1, ...

データの種類 DOUBLE

m_i
n 個の平均$m_1, m_2, \ldots, m_n$を指定する、数値の定数、変数または式のリストです。

s_i
n 個の標準偏差$s_1, s_2, \ldots, s_n$を指定する、数値の定数、変数または式のリストです。

範囲 *s* > 0

データの種類 DOUBLE

詳細

正規混合分布の PDF 関数は、正規混合分布のオブザベーションが *x* 以下となる確率を返します。

$$PDF('NORMALMIX', x, n, p, m, s) = \sum_{i=1}^{i=n} p_i \, PDF('NORMAL', x, m_i, s_i)$$

正規混合分布の重み付けは、負でないことが必要です。重み付けの合計が 1 に等しくない場合は相対的な重み付けとして扱われ、合計が 1 となるように調整されます。

注: 正規混合分布には、*位置*または*尺度*パラメーターはありません。

例

次のプログラムは、PDF 正規混合分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double y;
  method init();
    y=pdf('NORMALMIX',2.3, 3, .33, .33, .34,
     .5, 1.5, 2.5, .79, 1.6, 4.3);
    put 'Normal mix dist:' y;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
Normal mix dist: 0.11655396435559
```

関連項目

関数:

- “CDF NORMALMIX 分布 関数” (394 ページ)
- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

PDF PARETO 分布 関数

パレート確率密度(質量)分布の値を返します。

カテゴリ:	確率
別名:	PMF
返されるデータの種類:	DOUBLE

構文

PDF ('PARETO', *x*, *a* [, *k*])

必須引数

x
数値の確率変数を指定する数値定数、変数または式です。

データの種類 DOUBLE

a
形状パラメーターを指定する数値定数、変数または式です。

範囲 $a > 0$

データの種類 DOUBLE

オプション引数

k
尺度パラメーターを指定する数値定数、変数または式です。

デフォルト 1

範囲 $k > 0$

データの種類 DOUBLE

詳細

パレート分布の PDF 関数は、パレート分布(形状パラメーターは *a*、尺度パラメーターは *k*)の確率密度関数を返します。PDF 関数は、値 *x* で評価されます。

$$PDF('PARETO', x, a, k) = \begin{cases} 0 & x < k \\ \frac{a}{k}\left(\frac{k}{x}\right)^{a+1} & x \geq k \end{cases}$$

例

次のプログラムは、PDF パレート分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double y;
  method init();
    y=pdf('PARETO', 1, 1);
    put 'Pareto dist:     ' y;
  end;
enddata;
```

```
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
Pareto dist:    1
```

関連項目

関数:

- “CDF PARETO 分布 関数” (397 ページ)
- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

PDF POISSON 分布 関数

Poisson 確率密度(質量)分布の値を返します。

カテゴリ:	確率
別名:	PMF
返されるデータの種類:	DOUBLE

構文

PDF ('POISSON', *n*, *m*)

必須引数

n

確率変数を指定する数値定数、変数または式です。この引数は整数にする必要があります。

範囲	*n*=0, 1, ...
データの種類	DOUBLE

m
平均パラメーターを指定する数値定数、変数または式です。

範囲 $m > 0$

データの種類 DOUBLE

詳細

Poisson 分布の PDF 関数は、平均が *m* の確率密度関数を返します。この PDF 関数は、値 *n* で評価されます。

$$PDF('POISSON', n, m) = \begin{cases} 0 & n < 0 \\ e^{-m}\frac{m^n}{n!} & n \geq 0 \end{cases}$$

注: Poisson 分布には、*位置*または*尺度*パラメーターはありません。

例

次のプログラムは、PDF Poisson 分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double y;
  method init();
    y=pdf('POISSON', 2, 1);
    put 'Poisson dist:     ' y;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
Poisson dist:     0.18393972058572
```

関連項目

関数:

- “CDF POISSON 分布 関数” (399 ページ)

- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

PDF T 分布 関数

T 確率密度(質量)分布の値を返します。

カテゴリ:	確率
別名:	PMF
返されるデータの種類:	DOUBLE

構文

PDF ('T', *t*, *df* [, *nc*])

必須引数

t
確率変数を指定する数値定数、変数または式です。

データの種類 DOUBLE

df
自由度を指定する数値定数、変数または式です。

範囲 *df* > 0

データの種類 DOUBLE

オプション引数

nc
省略可能な非心度パラメーターを指定する数値定数、変数または式です。

データの種類 DOUBLE

詳細

T 分布の PDF 関数は、*T* 分布(自由度は *df*、非心度パラメーターは *nc*)の確率密度関数を返します。PDF 関数は、値 *x* で評価されます。この PDF 関数では、整数以外の自由度を使用できます。*nc* が除外されているかゼロの場合、心度 *T* 分布の値が返されます。この式で、$\nu = df$として、そして$\delta = nc$とします。

$$PDF('T', t, \nu, \delta) = \frac{1}{2^{\frac{\nu}{2}-1}\Gamma\left(\frac{\nu}{2}\right)} \int_0^{\infty} x^{\nu-1} e^{-\frac{1}{2}x^2} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{tx}{\sqrt{\nu}}-\delta\right)^2} \frac{x}{\sqrt{\nu}} dx$$

注: *T* 分布には、*位置*または*尺度*パラメーターはありません。

例

次のプログラムは、PDF T 分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double y;
  method init();
    y=pdf('T',.9,5);
    put 'T dist:      ' y;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
T dist:     0.24194434361358
```

関連項目

関数:

- “CDF T 分布 関数” (400 ページ)
- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)

- “SQUANTILE 関数” (1149 ページ)

PDF TWEEDIE 分布 関数

Tweedie 確率密度(質量)分布の値を返します。

カテゴリ:	確率
別名:	PMF
返されるデータの種類:	DOUBLE

構文

PDF ('TWEEDIE', *y*, *p* [, *μ*, *φ*])

必須引数

y

確率変数を指定する数値定数、変数または式です。

範囲	*y* ≥0
データの種類	DOUBLE
注	この引数は必須です。 *y*>1 の場合、*y* は数値になります。*p*=1 の場合、*y* は整数になります。

p

累乗パラメーターを指定する数値定数、変数または式です。

範囲	*p* ≥1
データの種類	DOUBLE
注	この引数は必須です。

オプション引数

μ

平均パラメーターを指定する数値定数、変数または式です。

デフォルト	1
範囲	*μ* >0
データの種類	DOUBLE

φ
ばらつきのパラメーターを指定する数値定数、変数または式です。

デフォルト 1

範囲 *φ*>0

データの種類 DOUBLE

詳細

Tweedie 分布の PDF 関数は、*variance* の式の関係がある分散と平均を使用した指数散布度モデルを返します。$= \phi * \mu^p$

$$\frac{1}{y}\sum_{j=1}^{\infty}\left(\frac{y^{-j\alpha}(p-1)^{j\alpha}}{\phi^{j(1-\alpha)}(2-p)^{j}j!\Gamma(-j\alpha)}\right)\exp\left(\frac{1}{\phi}\left(y\frac{\mu^{1-p}-1}{1-p}-\frac{\mu^{2-p}-1}{2-p}\right)\right)$$

前述の式には次の関係が適用されます。

$$\alpha=\frac{2-p}{1-p}$$

注: 計算された Tweedie 確率の精度は、パラメーター空間における位置に大きく依存します。通常、10 桁の精度を利用できますが、次の場合を除きます。*p* の近似値が 2 の場合、または*ϕ*の近似値が 0 の場合どちらのケースにしても、6 桁の精度に低下する可能性があります。

注: 数値データでの問題を避けるため、μ および Φ は定数 SQRTMACEPS より小さくしないようにします。

例

次のプログラムは、PDF Tweedie 分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double y;
  method init();
    y=pdf('TWEEDIE', .8, 5);
    put 'Tweedie dist:     ' y;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
Tweedie dist:     0.74229082358846
```

関連項目

関数:

- “CDF TWEEDIE 分布 関数” (402 ページ)
- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

PDF UNIFORM 分布 関数

一様確率密度(質量)分布の値を返します。

カテゴリ:	確率
別名:	PMF
返されるデータの種類:	DOUBLE

構文

PDF ('UNIFORM', *x* [, *l*, *r*])

必須引数

x
確率変数を指定する数値定数、変数または式です。

データの種類 DOUBLE

オプション引数

l
左位置パラメーターを指定する数値定数、変数または式です。

デフォルト 0

データの種類 DOUBLE

r

右位置パラメーターを指定する数値定数、変数または式です。

デフォルト 1

範囲 *r* > *l*

データの種類 DOUBLE

詳細

一様分布の PDF 関数は、一様分布(左位置パラメーターは *l*、右位置パラメーターは *r*)の確率密度関数を返します。PDF 関数は、値 *x* で評価されます。

$$PDF('UNIFORM', x, l, r) = \begin{cases} 0 & x < l \\ \frac{1}{r-l} & l \le x \le r \\ 0 & x > r \end{cases}$$

例

次のプログラムは、PDF 一様分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double y;
  method init();
    y=pdf('UNIFORM', 0.25);
    put 'Uniform dist:     ' y;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
Uniform dist:     1
```

関連項目

関数:

- “CDF UNIFORM 分布 関数” (405 ページ)
- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

PDF Wald (逆ガウス)分布 関数

Wald (逆ガウスとも呼ばれる)確率密度(質量)分布から値を返します。

カテゴリ:	確率
別名:	PMF
返されるデータの種類:	DOUBLE

構文

PDF ('WALD', *x*, *λ* [, *μ*])
PDF ('IGAUSS', *x*, *λ* [, *μ*])

必須引数

x
確率変数を指定する数値定数、変数または式です。

λ
形状パラメーターを指定する数値定数、変数または式です。

範囲 $\lambda > 0$

オプション引数

μ
平均パラメーターを指定する数値定数、変数または式です。

デフォルト 1

範囲	μ > 0

詳細

Wald 分布の PDF 関数は、値 *x* で評価される Wald 分布(形状パラメーターは λ)の確率密度関数を返します。

$$fx(x) = \left[\frac{\lambda}{2\pi x^3}\right]^{1/2} \exp\left\{-\frac{\lambda}{2\mu^2 x}(x-\mu)^2\right\}, \qquad x > 0$$

例

次のプログラムは、PDF Wald 分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double y;
  method init();
    y=pdf('WALD', 1, 2);
    put 'Wald dist:     ' y;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
Wald dist:     0.56418958354775
```

関連項目

関数:

- “CDF WALD (逆ガウス)分布 関数” (407 ページ)
- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

PDF WEIBULL 分布 関数

Weibull 確率密度(質量)分布の値を返します。

カテゴリ:	確率
別名:	PMF
返されるデータの種類:	DOUBLE

構文

PDF('WEIBULL', *x*, *α* [, *λ*])

必須引数

x
確率変数を指定する数値定数、変数または式です。

データの種類	DOUBLE

α
形状パラメーターを指定する数値定数、変数または式です。

範囲	*α* > 0
データの種類	DOUBLE

オプション引数

λ
尺度パラメーターを指定する数値定数、変数または式です。

デフォルト	1
範囲	λ > 0
データの種類	DOUBLE

詳細

Weibull 分布の PDF 関数は、Weibull 分布(形状パラメーターは *α*、尺度パラメーターは λ)の確率密度関数を返します。PDF 関数は、値 *x* で評価されます。

$$PDF('WEIBULL', x, a, \lambda) = \begin{cases} 0 & x < 0 \\ exp\left(-\left(\frac{x}{\lambda}\right)^{a}\right)\frac{a}{\lambda}\left(\frac{x}{\lambda}\right)^{a-1} & x \geq 0 \end{cases}$$

例

次のプログラムは、PDF Weibull 分布関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double y;
  method init();
    y=pdf('WEIBULL', 1, 2);
    put 'WEIBULL dist:     ' y;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
WEIBULL dist:     0.73575888234288
```

関連項目

関数:

- “CDF WEIBULL 分布 関数” (409 ページ)
- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “QUANTILE 関数” (1052 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

PERM 関数

n 個の項目から一度に *r* 個の項目を抜き出す場合の順列数を計算します。

カテゴリ: 組み合わせ

返されるデータの種類: DOUBLE

構文

PERM(*n*[, *r*])

必須引数

n
要素の合計数を表す有効な式を指定します。サンプルはこの中から選ばれます。

データの種類 DOUBLE

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

オプション引数

r
選択した要素の数を表す有効な式を指定します。

制限事項 $r \le n$

データの種類 DOUBLE

注 *r* を省略すると、関数は *n* の階乗を返します。

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

詳細

PERM 関数の数学的表現は、次の式によって得られます。

$$PERM(n, r) = \frac{n!}{(n-r)!}$$

$n \ge 0$、$r \ge 0$ および $n \ge r$ となります。

式による計算ができない場合は欠損値が返されます。やや大きな値の場合、PERM 関数を計算できないことがあります。

例

次のプログラムは、PERM 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
 proc ds2;
data _null_;
  dcl double x y z;
  method run();
    x=perm(5,1);
    y=perm(5);
    z=perm(5, 2);
    put x y z;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
5 120 20
```

関連項目

関数:

- “COMB 関数” (427 ページ)
- “FACT 関数” (532 ページ)
- “LPERM 関数” (835 ページ)

PMT 関数

均等払いローンまたは定期預金の将来残高に対する定期的支払いを返します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

PMT(*rate*, *number-of-periods*, *principal-amount*[, *future-amount*][, *type*])

必須引数

rate
支払期間ごとの利率を指定します。

データの種類 DOUBLE

number-of-periods
支払期間の数を指定します。

要件 *number-of-periods* は正の整数にする必要があります。

データの種類 DOUBLE

principal-amount
ローンの元金を指定します。null 値または欠損値が指定されている場合、ゼロとみなされます。

データの種類 DOUBLE

オプション引数

future-amount
将来の残高を指定します。*future-amount* は、指定した支払い期間数終了後のローンの残高、または定期預金の将来の残高を指定できます。0 を指定した場合、*future-amount* が省略されたか欠損値が指定されたとみなされます。

データの種類 DOUBLE

type
期首と期末のどちらで支払がなされるのかを指定します。0 は期末の支払を表し、1 は期首の支払を表します。*type* が省略された場合や、null 値または欠損値が指定された場合は、ゼロとみなされます。

データの種類 DOUBLE

例

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

- 名目年利が 8%で支払い期間が 10 か月の$10,000 のローンの月々の支払いは次のように計算されます。

```
proc ds2;
data _null_;
  dcl double Payment1;
  method init();
    Payment1 = PMT(0.08/12., 10, 10000);
    /* same results with PMT(0.08/12., 10, 10000, 0, 0) ***/
    put Payment1=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
1037.03208935915
```

- 同じローンに期首払いがある場合、支払いは次のように計算できます。

```
proc ds2;
data _null_;
  dcl double Payment2;
  method init();
    Payment2 = PMT(0.08/12., 10, 10000, 0, 1);
    put Payment2= ;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
1030.16432717796
```

- 名目年利 12%の$5,000 のローンを 5 か月の支払いで返済する場合の支払い額は、次のように計算されます。

```
proc ds2;
data _null_;
  dcl double Payment3;
  method init();
    Payment3 = PMT(.01/12., 5, 5000);
    put Payment3= ;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
1002.50138831008
```

- 名目年利 6%で 18 年以上積み立て、18 年終了時に$50,000 積み立てる毎月の定期預金の支払い額は、次のように計算されます。

```
proc ds2;
data _null_;
  dcl double Payment4;
  method init();
    /* this generates a negative number */
    Payment4 = PMT(.06/12., 216, 0, 50000, 0);
    put Payment4= ;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
-129.081160867993
```

POISSON 関数

Poisson 分布の確率を返します。

カテゴリ: 確率

返されるデータの種類: DOUBLE

構文

POISSON(*m*, *n*)

必須引数

m
数値の平均パラメーターに評価される有効な式を指定します。

範囲 *m* ≥ 0

データの種類 DOUBLE

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

n
確率変数に評価される有効な式を指定します。

範囲 *n* ≥ 0

データの種類 DOUBLE

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

詳細

POISSON 関数は、平均が *m* の Poisson 分布のオブザベーションが *n* 以下となる確率を返します。オブザベーションが指定した値の *n* と等しくなる確率を計算するには、Poisson 分布からの 2 つの確率(*n* の確率と *n*-1 の確率)の差異を計算します。

例

次のプログラムは、POISSON 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double x;
  method run();
    x=poisson(1,2);
```

```
    put x;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
0.9196986029286
```

POWER 関数

指定された累乗の数値式の値を返します。

カテゴリ: 数学

返されるデータの種類: DOUBLE

構文

POWER(*numeric-expression*, *whole-number-expression*)

必須引数

numeric-expression
数値に評価される有効な式を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

whole-number-expression
数値に評価される有効な式を指定します。この引数は整数にする必要があります。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

numeric-expression が null の場合、POWER 関数は null を返します。結果が引数のデータ型の範囲に収まらない数値の場合、POWER 関数は失敗します。

例

次のプログラムは、POWER 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double x;
  method run();
    x=power(5*3, 2);
    put x;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
225
```

PPMT 関数

将来の残高を達成するための、均等払いローンまたは定期預金に対する指定期間の元本の支払いを返します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

PPMT(*rate*, *period*, *number-of-periods*, *principal-amount*[, *future-amount*][, *type*])

必須引数

rate
支払期間ごとの利率を指定します。

データの種類 DOUBLE

period
元本支払いを計算するための支払い期間を指定します。

要件　*period* は、*number-of-periods* の値以下の整数で指定する必要があります。

データの種類　DOUBLE

number-of-periods
支払期間の数を指定します。

要件　*number-of-periods* は正の整数にする必要があります。

データの種類　DOUBLE

principal-amount
ローンの元金を指定します。null 値または欠損値が指定されている場合、ゼロとみなされます。

データの種類　DOUBLE

オプション引数

future-amount
将来の残高を指定します。*future-amount* は、指定した支払い期間数終了後のローンの残高、または定期預金の将来の残高を指定できます。*future-amount* が省略された場合や、null 値または欠損値が指定された場合は、ゼロとみなされます。

データの種類　DOUBLE

type
期首と期末のどちらで支払がなされるのかを指定します。0 は期末の支払を表し、1 は期首の支払を表します。*type* が省略された場合や、null 値または欠損値が指定された場合は、ゼロとみなされます。

データの種類　DOUBLE

例

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

- 名目年利 10%で 2 年間、$2,000 のローンの最初の毎月の定期支払いの元本支払い額は、次のように計算されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double PrincipalPayment;
  method run();
    PrincipalPayment = PPMT(0.1/12., 1, 24, 2000);
    put PrincipalPayment;
  end;
enddata;
```

```
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
75.6231860083663
```

- $20,000 の 3 年ローンで期首支払いの場合、元本支払いは次のように計算されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double PrincipalPayment2;
  method run();
    PrincipalPayment2 = PPMT(0.1/12, 1, 36, 20000, 0, 1);
    put PrincipalPayment2;
  end;
enddata;
run;
quit;
```

SAS は、最初の支払いで支払われた元本として、次の出力をログに書き込みます。

```
640.010324505867
```

- 3 年終了時の残高が 5,000 ドルの月末払いローンの元本支払いは、次のように計算されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double PrincipalPayment3;
  method run();
    PrincipalPayment3 = PPMT(0.1/12, 1, 36, 20000, 5000, 0);
    put PrincipalPayment3;
  end;
enddata;
run;
quit;
```

SAS は、最初の支払いで支払われた元本として、次の出力をログに書き込みます。

```
359.007807907562
```

PROBBETA 関数

ベータ分布の確率を返します。

カテゴリ:	確率
返されるデータの種類:	DOUBLE

構文

PROBBETA(*x*, *a*, *b*)

必須引数

x
数値の確率変数です。

範囲 $0 \leq x \leq 1$

データの種類 DOUBLE

a
数値の形状パラメーターです。

範囲 $a > 0$

データの種類 DOUBLE

b
数値の形状パラメーターです。

範囲 $b > 0$

データの種類 DOUBLE

詳細

PROBBETA 関数は、形状パラメーターが *a* と *b* のベータ分布のオブザベーションが *x* 以下となる確率を返します。

例

次のプログラムは、PROBBETA 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double x;
  method run();
    x=probbeta(.2,3,4);
    put x= ;
  end;
enddata;
run;
```

```
quit;
```

SAS は次の出力をログに書き込みます。

```
x=0.09888
```

PROBBNML 関数

二項分布の確率を返します。

カテゴリ: 確率

返されるデータの種類: DOUBLE

構文

PROBBNML(*p*, *n*, *m*)

必須引数

p
数値の成功率パラメーターです。

範囲 $0 \le p \le 1$

データの種類 DOUBLE

n
独立した Bernoulli 試行数のパラメーターです。この引数は整数にする必要があります。

範囲 $n > 0$

データの種類 DOUBLE

m
成功数の確率変数です。この引数は整数にする必要があります。

範囲 $0 \le m \le n$

データの種類 DOUBLE

詳細

PROBBNML 関数は、成功確率が *p*、試行数が *n*、成功数が *m* の二項分布のオブザベーションが *m* 以下となる確率を返します。オブザベーションが指定した値の *m* と等しくなる確率を計算するには、二項分布の 2 つの確率(*m* の成功と *m*-1 の成功)の差異を計算します。

例

次のプログラムは、PROBBNML 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  method run();
    x=probbnml(0.5,10,4);
    put x= ;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
x=0.376953125
```

PROBBNRM 関数

2 変量正規分布の確率を返します。

カテゴリ:	確率
返されるデータの種類:	DOUBLE

構文

PROBBNRM(*x*, *y*, *r*)

必須引数

x
数値の定数、変数または式を指定します。

データの種類 DOUBLE

y
数値の定数、変数または式を指定します。

データの種類 DOUBLE

r
数値相関係数です。

範囲 $-1 \le r \le 1$

データの種類 DOUBLE

詳細

PROBBNRM 関数は、平均が 0、分散が 1、相関係数が *r* の標準 2 変量正規分布のオブザベーション(X, Y)が(*x*, *y*)以下となる確率を返します。すなわち、X≤*x* かつ Y≤*y* となる確率を返します。次の等式は、*u* と *v* がそれぞれ確率変数 *x* と *y* を表す PROBBNRM 関数を示しています。

$$\mathrm{PROBBNRM}(x, y, r) = \frac{1}{2\pi\sqrt{1-r^2}} \int_{-\infty}^{x} \int_{-\infty}^{y} \exp\left[-\frac{u^2 - 2ruv + v^2}{2\left(1-r^2\right)} \right] dv\, du$$

例

次のプログラムは、PROBBNRM 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double p;
  method run();
    p=probbnrm(.4, -.3, .2);
    put p= ;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
0.27831833451901
```

PROBCHI 関数

カイ 2 乗分布の確率を返します。

カテゴリ:	確率
返されるデータの種類:	DOUBLE

構文

PROBCHI(*x*, *df*[, *nc*])

必須引数

x
数値の確率変数です。

範囲	$x \geq 0$
データの種類	DOUBLE

df
数値の自由度パラメーターです。

範囲	$df > 0$
データの種類	DOUBLE

オプション引数

nc
数値の非心度パラメーターです(省略可能)。

範囲	$nc \geq 0$
データの種類	DOUBLE

詳細

PROBCHI 関数は、自由度が *df*、非心度パラメーターが *nc* のカイ 2 乗分布のオブザベーションが *x* 以下となる確率を返します。この関数では、整数以外の自由度パラ

メーター *df* を使用できます。パラメーター *nc* (省略可能)が指定されていないかゼロの場合、心度カイ 2 乗分布の値が返されます。

例

次のプログラムは、PROBCHI 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double x;
  method run();
    x=probchi(11.264,11);
    put x= ;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
0.5785813293173
```

PROBDF 関数

時系列の単位根に対する Dickey-Fuller 検定の有意確率を計算します。

カテゴリ:	確率
返されるデータの種類:	DOUBLE

構文

PROBDF(*x*, *n*[, *d*[, *type*]])

必須引数

x
検定統計量です。

データの種類 DOUBLE

n

サンプルサイズです。許可される *n* の最小値は、第 3 引数 *d* に指定された値によって異なります。集合(1,2,4,6,12)の *d* の場合、*n* は max(2d,5)以下の整数でなければなりません。*d* の他の値の場合、*n* の最小値は 24 です。

データの種類 DOUBLE

オプション引数

d

検定される単位根の次数を与える整数です。単純単位根(1–B)の検定では、*d*=1 を指定します。季節単位根の検定では、*d* が検定の季節サイクルの長さに等しいことを指定します。*d* のデフォルト値は 1 です。つまり、*d* が指定されていない場合は、単純単位根の検定が想定されます。*d* の最大値は 12 です。

データの種類 DOUBLE

type

使用する検定統計量の種類を指定する文字引数です。*type* の値は次のとおりです。RSM (回帰単一平均切片)、RTR (回帰決定論的時間トレンド)、RZM (回帰ゼロ平均、切片なし)、SSM (スチューデント化単一平均切片)、STR (スチューデント化決定論的時間トレンド)、SZM (スチューデント化ゼロ平均、切片なし)。詳細については、SAS DS2 言語リファレンスを参照してください。

使用する検定統計量の種類を指定する文字引数です。*type* の値は次のとおりです。

RSM
単一平均(切片)の場合の回帰検定統計量を指定します。

RTR
決定論的時間トレンドの場合の回帰検定統計量を指定します。

RZM
ゼロ平均(切片なし)の場合の回帰検定統計量を指定します。

SSM
単一平均(切片)の場合のスチューデント化検定統計量を指定します。

STR
決定論的時間トレンドの場合のスチューデント化検定統計量を指定します。

SZM
ゼロ平均(切片なし)の場合のスチューデント化検定統計量を指定します。

デフォルト SZM

制限事項 値 STR および RTR は、*d*=1 の場合にのみ許可されます。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

詳細

理論的背景

時系列に単位根がある場合、系列は非定常であり、通常の最小 2 乗(OLS)推定量は正規分布していません。Dickey (1976)および Dickey and Fullerr (1979)は、単純単位根を持つ時系列の自己回帰モデルの OLS 推定量の極限分布を研究しました。Dickey, Hasza, and Fullerr (1984)は、季節単位根を持つ時系列の極限分布を得ました。このセクションでは、非季節性検定を紹介し、非季節性の参考文献をリストします。

Dicky-Fuller 回帰では、帰無仮説は自己回帰単位根$H_0 : \alpha = 1$があるとし、対立仮説は$H_\alpha : |\alpha| < 1$とし、α は次の時系列の自己回帰係数とします。

$$y_t = \alpha y_{t-1} + \varepsilon_t$$

このモデルはゼロ平均(ZM)モデルと呼ばれます。標準の Dickey-Fuller (DF)検定では、誤差がホワイトノイズであると想定されています。定数または時間トレンドを含む回帰モデルには、他に 2 つのタイプがあります。

$$y_t = \mu + \alpha y_{t-1} + \varepsilon_t$$

$$y_t = \mu + \beta t + \alpha y_{t-1} + \varepsilon_t$$

これら 2 つのモデルは、それぞれ一定平均モデル(SM)とトレンドモデル(TR)と呼ばれます。一定平均モデルには、時系列の一定平均 μ が含まれます。ただし、μ の解釈は次の意味で定常性に依存します。つまり、α 1 のときの定常性の場合の平均は、α < 1 のときの積分の場合のトレンドです。したがって、帰無仮説は、α = 1 かつ μ = 0 であるという共同仮説でなければなりません。ただし、単位根検定の場合、検定統計量は α = 1 の帰無仮説に関係しています。共同帰無仮説は一般的には使用されません。この問題は、Bhargava, A. (1986)で異なるネストモデルで扱われています。

Dickey-Fuller 検定の I(1)の帰無のもとでは、差分過程は連続的に相関していません。この仕様の一般化が大いに必要です。最初に Dickey and Fuller (1979)で提案された拡張 Dickey-Fuller (ADF)検定は、次のように自己回帰モデルにラグ付き第一階差を追加することで、時系列の系列相関を調整します。(*p* +1)次の自己回帰時系列を考えてみましょう。

$$y_t = \alpha_1 y_{t-1} + \alpha_2 y_{t-2} + \ldots + \alpha_{p+1} y_{t-p-1} + e_t$$

特性方程式は次のとおりです。

$$m^{p+1} - \alpha_1 m^p - \alpha_2 m^{p-1} - \ldots - \alpha_{p+1} = 0$$

すべての特性根が絶対値で 1 未満の場合、y_tは定常です。y_tは単位根があれば非定常です。単位根がある場合、自己回帰パラメーターの合計は 1 であるため、自己回帰パラメーターの合計が 1 であるかどうかを検定することで、単位根を検定できます。切片なしモデルは、次のようにパラメーター化されます。

$$\nabla y_t = \delta y_{t-1} + \theta_1 \nabla y_{t-1} + \ldots + \theta_p \nabla y_{t-p} + e_t$$

前述の式では、次の関係が適用されます。

$$\nabla y_t = y_t - y_{t-1}$$

および

$$\delta = a_1 + \ldots + \alpha_{p+1} - 1$$

$$\theta_k = -\alpha_{k+1} - \ldots - \alpha_{p+1}$$

推定量は、∇y_tを$y_{t-1}, \nabla y_{t-1}, \ldots, \nabla y_{t-p}$で回帰することによって取得されます。δ の通常の最小 2 乗推定量の *t* 統計量は、単位根検定の検定統計量です。

type 引数の値がゼロ以外の平均(切片の場合)の検定を指定する場合、自己回帰モデルには平均項α_0が含まれます。*type* 引数値が時間トレンドの検定を指定する場合、モデルには時間トレンド項も含まれ、モデルは次のようになります。

$$\nabla y_t = \alpha_0 + \gamma t + \delta y_{t-1} + \theta_1 \nabla y_{t-1} + \ldots + \theta_p \nabla y_{t-p} + e_1$$

季節単位根の検定については、乗法モデルを検討してください。

$$(1 - \alpha_d \mathrm{B}^d)(1 - \theta_1 \mathrm{B} - \ldots - \theta_p \mathrm{B}^p) y_t = e_t$$

$\nabla^d y_t \equiv y_t - y_{t-d}$であるとします。検定統計量は、次のステップで計算されます。

1. $\nabla^d y_t$を$\nabla^d y_{t-1} \ \ldots \ \nabla^d y_{t-p}$で回帰して、初期推定量$\hat{\theta}_i$ を取得し、残差$\hat{e}_t$ を計算します。$\alpha_d = 1$という帰無仮説のもとでは、$\hat{\theta}_i$ はθ_iの一致推定量です。

2. $\hat{e}_t$ を$(1 - \hat{\theta}_1 \mathrm{B} - \ldots - \hat{\theta}_p \mathrm{B}^p) y_{t-d}, \ \nabla^d y_{t-1}, \ldots, \nabla^d y_{t-p}$で回帰して、$\delta = \alpha_d - 1$と$\theta_i - \hat{\theta}_i$の推定値を取得します。

2 番目のステップで生成される δ の推定値の *t* 比は、季節単位根を検定するための検定統計量として使用されます。θ_iの推定値は、第 2 ステップの$\theta_i - \hat{\theta}_i$の推定値を第 1 ステップの$\hat{\theta}_i$に加算することによって得られます。

系列$(1 - \mathrm{B}^d) y_t$は定常と見なされます。ここで、*d* は PROBDF 関数の第 3 引数の値です。

系列が ARMA 過程である場合、信頼できる検定統計量を取得するには、*p* の値を大きくすることが望ましい場合があります。*p* の適切な値を決定するには、Said and Dickey (1984)を参照してください。[1].

検定統計量

Dickey-Fuller 検定は、時系列が定常性の対立仮説に対してラグ *d* 単位根を示すという帰無仮説を検定するために使用されます。PROBDF 関数は、帰無仮説が真であるという仮定のもとで、*x* より極端な検定統計量が観測される確率を計算します。PROBDF が小さい(有意な)確率値を返す場合、単位根仮説を棄却する必要があります。

最初に Dickey-Fuller 回帰を考えてみましょう。Dickey-Fuller 検定にはいくつかのバージョンがあります。PROBDF 関数は、*type* 引数で選択される 6 つのバージョンをサポートします。検定統計量 *x* の計算方法に対応する *type* 値を指定します。

1. Said, S. E., and Dickey, D. A. (1984). “Testing for Unit Roots in ARMA Models of Unknown Order. Biometrika 71:599–607.

type 値の最後の 2 文字は、Dickey-Fuller 検定統計量の計算に使用される回帰モデルのタイプを指定します。*type* 値の最後の 2 文字の意味は次のとおりです。

SM
: 単一平均または切片の場合を指定します。検定統計量 *x* は、次の回帰モデルから計算されると想定されます。

$$y_t = \mu + \alpha y_{t-1} + e_t$$

TR
: 切片および決定論的時間トレンドの場合を指定します。検定統計量 *x* は、次の回帰モデルから計算されると想定されます。

$$y_t = \mu + \gamma t + \alpha y_{t-1} + e_t$$

ZM
: ゼロ平均または切片なしの場合を指定します。検定統計量 *x* は、次の回帰モデルから計算されると想定されます。

$$y_t = \alpha y_{t-1} + e_t$$

type 値の最初の文字は、回帰検定統計量またはスチューデント化検定統計量のどちらを使用するかを指定します。$\hat{\alpha}$を系列のラグの推定回帰係数とし、se_αを$\hat{\alpha}$の標準誤差とします。*type* 値の最初の文字の意味は次のとおりです。

R
: 回帰係数ベースの検定統計量を指定します。検定統計量は次のとおりです。

$$\rho = n(\hat{\alpha} - 1)$$

S
: スチューデント化検定統計量を指定します。検定統計量は次のとおりです。

$$DF_t = \frac{(\hat{\alpha} - 1)}{se_\alpha}$$

type 値 R の式は、ρ 検定とも呼ばれます。*type* 値 S の式は、τ 検定とも呼ばれます。ゼロ平均モデルの場合、Dickey-Fuller 検定統計量の漸近分布は次のようになります。

$$n(\hat{\alpha} - 1) \Rightarrow \left(\int_0^1 W(r)\,dW(r)\right)\ \left(\int_0^1 W(r)^2 dr\right)^{-1}$$

$$DF_\tau \Rightarrow \left(\int_0^1 W(r)\,dW(r)\right)\ \left(\int_0^1 W(r)^2\,dr\right)^{-\frac{1}{2}}$$

一定平均モデルの場合、漸近分布は次のようになります。

$$n(\hat{\alpha} - 1) \Rightarrow \left([W(1)^2 - 1]/2 - W(1)\int_0^1 W(r)\,dr\right)\ \left(\int_0^1 W(r)^2\,dr - \left(\int_0^1 W(r)\,dr\right)^2\right)^{-1}$$

$$DF\tau \Rightarrow \left([W(1)^2 - 1]/2 - W(1)\int_0^1 W(r)\,dr\right)\ \left(\int_0^1 W(r)^2\,dr - \left(\int_0^1 W(r)\,dr\right)^2\right)^{-1/2}$$

トレンドモデルの場合、漸近分布は次のようになります。

$$n(\hat{\alpha}-1) \Rightarrow \left[W(r)dW + 12\left(\int_0^1 rW(r)\,dr - \frac{1}{2}\int_0^1 W(r)\,dr\right)\left(\int_0^1 W(r)\,dr - \frac{1}{2}W(1)\right) - W(1)\int_0^1 W(r)\,dr\right]D^1$$

$$DF\tau \Rightarrow \left[W(r)dW + 12\left(\int_0^1 rW(r)\,dr - \frac{1}{2}\int_0^1 W(r)\,dr\right)\left(\int_0^1 W(r)\,dr - \frac{1}{2}W(1)\right) - W(1)\int_0^1 W(r)\,dr\right]D^{1/2}$$

次の式は、前述の式に適用されます。

$$D = \int_0^1 W(r)^2\,dr - 12\left(\int_0^1 rW(r)dr\right)^2 + 12\int_0^1 W(r)\,dr\int_0^1 rW(r)\,dr - 4\left(\int_0^1 W(r)dr\right)^2$$

Dickey-Fuller 検定の帰無分布の詳細については、Dickey and Fuller (1979)、Dickey, Hasza, and Fuller (1984)、および Hamilton (1994)を参照してください。前述の式は、基本的な Dickey-Fuller 検定用です。PROBDF 関数は、拡張 Dickey-Fuller 検定にも使用できます。この検定では、誤差項が自己回帰過程としてモデル化されます。ただし、拡張 Dickey-Fuller 検定では、検定統計量の計算方法が多少異なります。非季節拡張 Dickey-Fuller 検定の場合、検定統計量は、Dickey-Fuller 検定に似た 2 つの形式のいずれかになります。形式の 1 つは、OLS *t* 値$\frac{\hat{\alpha}-1}{sd(\alpha)}$であり、もう一方の形式は$\frac{n(\hat{\alpha}-1)}{1-\hat{\alpha}_1-\ldots-\hat{\alpha}_p}$です。

例

次のプログラムでは、テーブル Test に時系列変数 *Y* の 104 個のオブザベーションが含まれています。このプログラムは、Y 系列にラグ 4 の季節単位根が存在するという帰無仮説を検定します。次のプログラムは、PROC REG と PROBDF 関数を使用して単一平均 Dickey-Fuller 回帰係数検定を実行する方法を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test1 (overwrite=yes);
  set test;
  y4 = lag4(y);
end;
enddata;
run;
quit;

proc reg data=test1 outest=alpha;
  model y = y4 / noprint;
run;

proc ds2;
data _null_;
  dcl double x p;
  method run();
    set alpha;
    x = 100 * ( y4 - 1 );
    p = probdf( x, 100, 4, 'RSM' );
```

```
    put p= pvalue5.3;
  end;
enddata;
run;
quit;
```

拡張 Dickey-Fuller 検定を実行するには、系列のラグ差とラグ値で系列の差を回帰し、ラグ系列の回帰係数から検定統計量を計算します。次のプログラムは、PROC REG と PROBDF 関数を使用して、単純単位根に対する単一平均拡張 Dickey-Fuller ステューデント化検定を実行する方法を示しています。

```
proc ds2;
data test1 (overwrite=yes);
  set test;
  yl  = lag(y);
  yd  = dif(y);
  yd1 = lag1(yd); yd2 = lag2(yd);
  yd3 = lag3(yd); yd4 = lag4(yd);
 enddata;
run;
quit;

proc reg data=test1 outest=alpha covout;
  model yd = yl yd1-yd4 / noprint;
run;

proc ds2;
data _null_;
  dcl double a x p;
  method run();
    set alpha;
    retain a;
    if _type_ = 'PARMS' then
      a = yl;
    if _type_ = 'COV' & _NAME_ = 'Y1' then do;
      x = a / sqrt(yl);
      p = probdf( x, 99, 1, "SSM" );
      put p= ;
  end;
enddata;
run;
quit;
```

%DFTEST マクロは、Dickey-Fuller 検定を実行するためのより簡単な方法を提供します。次のステートメントは、前の例と同じ検定を実行します。

```
%dftest(test, y, ar=4);
%put p=&dftest;
```

注: DS2 が SAS の外部(SAS Federation Server グリッドコンピューティング環境など)で実行されている場合、SAS マクロ機能は使用できず、マクロを含む DS2 プログラムはコンパイルに失敗します。

PROBF 関数

F 分布の確率を返します。

カテゴリ: 確率

返されるデータの種類: DOUBLE

構文

PROBF(*x*, *ndf*, *ddf*[, *nc*])

必須引数

x
数値の確率変数です。

範囲 $x \geq 0$

データの種類 DOUBLE

ndf
数値の分子の自由度パラメーターです。

範囲 $ndf > 0$

データの種類 DOUBLE

ddf
数値の分母の自由度パラメーターです。

範囲 $ddf > 0$

データの種類 DOUBLE

オプション引数

nc
数値の非心度パラメーターです(省略可能)。

範囲 $nc \geq 0$

データの種類 DOUBLE

詳細

PROBF 関数は、分子の自由度が *ndf*、分母の自由度が *ddf*、非心度パラメーターが *nc* の *F* 分布のオブザベーションが *x* 以下となる確率を返します。PROBF 関数では、自由度パラメーター *ndf* と *ddf* に非整数を指定できます。パラメーター *nc* (省略可能)が指定されていないかゼロの場合、心度 *F* 分布の値が返されます。

F 検定統計の有意水準は次の式で求められます。

```
p=1-probf(x,ndf,ddf);
```

例

次のプログラムは、PROBF 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double x;
  method run();
    x=probf(3.32,2,3);
    put x= ;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
0.82639336022431
```

PROBGAM 関数

ガンマ分布の確率を返します。

カテゴリ:	確率
返されるデータの種類:	DOUBLE

構文

PROBGAM(*x*, *a*)

必須引数

x
数値の確率変数です。

範囲 $x \geq 0$

データの種類 DOUBLE

a
数値の形状パラメーターです。

データの種類 DOUBLE

詳細

PROBGAM 関数は、形状パラメーターが *a* のガンマ分布のオブザベーションが *x* 以下となる確率を返します。

例

次のプログラムは、PROBGAM 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  method run();
    x=probgam(1,3);
    put x= ;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
0.08030139707139
```

PROBHYPR 関数

超幾何分布の確率を返します。

カテゴリ: 確率

返されるデータの種類: DOUBLE

構文

PROBHYPR(*N*, *K*, *n*, *x*[, *r*])

必須引数

N
母集団サイズパラメーターです。この引数は整数にする必要があります。

範囲 $N \geq 1$

データの種類 DOUBLE

K
対象カテゴリの項目数のパラメーターです。この引数は整数にする必要があります。

範囲 $0 \leq K \leq N$

データの種類 DOUBLE

n
サンプルサイズパラメーターです。この引数は整数にする必要があります。

範囲 $0 \leq n \leq N$

データの種類 DOUBLE

x
確率変数です。この引数は整数にする必要があります。

範囲 $max(0, K + n-N) \leq x \leq min(K,n)$

オプション引数

r
数値オッズ比パラメーターです。

範囲 $r \geq 0$

データの種類 DOUBLE

詳細

PROBHYPR 関数は、拡張超幾何分布(母集団サイズは *N*、項目数は *K*、サンプルサイズは *n*、オッズ比は *r*)からオブザベーションが *x* 以下となる確率を返します。パラメ

ーター *r* (省略可能)が指定されていないか 1 に設定されている場合、一般超幾何分布の値が返されます。

例

次のプログラムは、PROBHYPR 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  method run();
    x=probhypr(200,50,10,2);
    put x= ;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
0.52367340812173
```

PROBIT 関数

標準正規分布の分位点を返します。

カテゴリ:	分位点
返されるデータの種類:	DOUBLE

構文

PROBIT(*p*)

必須引数

p
数値の確率です。

範囲	$0 < p < 1$
データの種類	DOUBLE

詳細

PROBIT 関数は、標準正規分布の *p*番目の分位点を返します。標準正規分布のオブザベーションが、返される分位点以下となる確率は *p* です。

注意
結果は-8.222 から 7.941 の間の値に切り捨てられる場合があります。

注: PROBIT は PROBNORM 関数の逆数です。

例

次のプログラムは、PROBIT 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double x y;
  method run();
    x=probit(.025);
    put x= ;
    y=probit(1.e-7);
    put y= ;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
x=-1.95996398454005
y=-5.19933758219281
```

PROBMC 関数

平均値の多重比較を行うために各種の分布から確率または分位点を返します。

カテゴリ:	確率
返されるデータの種類:	DOUBLE

構文

PROBMC(*distribution*, *q*, *prob*, *df*, *nparms*[, *parameters*])

必須引数

distribution
文字列を評価するか、文字列に強制変換できる文字定数、変数、または式であり、分布を識別します。分布として、ANOM (平均の分析)、DUNNETT1、DUNNETT2、MAXMOD (最大モジュール)、PARTRANGE (パーティション分割範囲)、RANGE (Student 化範囲)、WILLIAMS が有効です。

分布を特定する文字定数、変数または式です。有効な分布は次のとおりです。

分布	引数
平均分析	ANOM
片側 Dunnett	DUNNETT1
両側 Dunnett	DUNNETT2
最大係数	MAXMOD
分割された範囲	PARTRANGE
スチューデント化された範囲	RANGE
Williams	WILLIAMS

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

q
分布からの分位点です。

制限事項 *q* または *prob* のいずれか一方を指定できます(両方指定することはできません)。

データの種類 DOUBLE

prob
分布からの左側確率です。

制限事項 *prob* または *q* のいずれか一方を指定できます(両方指定することはできません)。

データの種類 DOUBLE

df
自由度です。

注: 欠損値は無限値として解釈されます。

nparms
治療の数です。

データの種類 DOUBLE

注 DUNNETT1 と DUNNETT2 については、コントロールグループを計算に入れません。

オプション引数

parameters
標本サイズが等しくない場合の処理に指定する必要のある *nparms* パラメーターのセットです。*parameters* の意味は *distribution* の値によって異なります。*parameters* が指定されていない場合、サンプルサイズは揃っているものと見なされます。通常、帰無仮説のケースはサンプルサイズが揃っています。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

詳細

概要

PROBMC 関数は、分散推定値に有限および無限の自由度を使用して、各種の分布の確率または分位点を返します。

prob 引数は確率変数が *q* 未満となる確率です。したがって、*p*-values は 1– *prob* のように計算できます。たとえば、有意水準 5%の臨界値を計算するには、*prob*= 0.95 と設定します。計算された確率の精度は $O(10^{-8})$ (絶対誤差)、計算された分位点の精度は $O(10^{-5})$です。

注: 自由度が有限のときと、サンプルサイズが不揃いのときは、スチューデント化された範囲は計算されません。

注: Williams 検定はサンプルサイズが揃っているときにのみ計算されます。

式とパラメーター

ここに掲げる等式は、各種の分布と、それぞれの分布におけるさまざまな状況で確率 *prob* と分位点 *q* を関連付ける等式に使用する式を定義します。これらの式では、*v* を自由度 *df* とします。

$$d\mu_\nu(x) = \frac{\nu^{\frac{\nu}{2}}}{\Gamma\left(\frac{\nu}{2}\right)2^{\frac{\nu}{2}-1}} x^{\nu-1}\varepsilon^{-\frac{\nu x^2}{2}}dx$$

$$\phi(x) = \frac{1}{\sqrt{2\,\pi}}\varepsilon^{-\frac{x^2}{2}}$$

$$\Phi(x) = \int_{-\infty}^{x} \phi(u)du$$

平均の分析の計算

平均の分析(ANOM)は、*k* (ガウス)標本として編成されるデータに適用されます。ここでは、*i*番目の標本のサイズが n_i となります。$I = \sqrt{-1}$であるとします。分布関数[1, 2, 3, 4, 5]は、多次元 *k* 多変量$\mathbb{T}$ベクトルの最大絶対値、ν 自由度、関連付けられた相関行列$\rho_{ij} = -\alpha_i\alpha_j$の CDF です。この式は次のように記述することができます。

$$prob = r(|t_1| < h, |t_2| < h, ..., |t_k| < h)$$
$$= \int_0^\infty \left\{ \int_0^\infty \prod_{j=0}^{j=k} g(sh, y, \alpha_j)\phi(y)dy \right\} d\mu_\nu(s)$$

前述の式には次の関係が適用されます。

$$g(sh, y, \alpha_j) = \Phi\left(\frac{sh - y\alpha_j}{\sqrt{1+\alpha_j^2}}\right) - \Phi\left(\frac{-sh - y\alpha_j}{\sqrt{1+\alpha_j^2}}\right)$$

この式では、$\Gamma(.)$、$\phi(.)$、$\Phi(.)$はそれぞれ、標準正規分布からのガンマ関数、密度、および CDF です。

$\nu = \infty$では、分布はこの式になります。

$$r(|t_1| < h, |t_2| < h, ..., |t_k| < h) = \int_0^\infty \prod_{j=0}^{j=k} g(h, y, \alpha_j)\phi(y)dy$$

前述の式には次の関係が適用されます。

$$g(h, y, \alpha_j) = \Phi\left(\frac{h - y\alpha_j}{\sqrt{1+\alpha_j^2}}\right) - \Phi\left(\frac{-h - y\alpha_j}{\sqrt{1+\alpha_j^2}}\right)$$

バランスの取れている場合、分布は次の式になります。

$$r(|t_1| < h, |t_2| < h, ..., |t_n| < h) = \int_0^\infty f(h, y, \rho)^n \phi(y)dy$$

前述の式には次の関係が適用されます。

$$f(h, y, \rho) = \Phi\left(\frac{h - y\sqrt{\rho}}{\sqrt{1+\rho}}\right) - \Phi\left(\frac{-h - y\sqrt{\rho}}{\sqrt{1+\rho}}\right)$$

$$\rho = \frac{1}{n-1}$$

この分布の構文は次のとおりです。

x=*probmc*('*anom*', *q*, *p*, *nu*, *n*[, *alpha*$_1$, ..., *alpha*$_n$]);

引数

x
返される結果の数値です。

q
分位点を示す数値です。

p
確率を示す数値です。*p* と *q* は、いずれかが欠損値である必要があります。

nu
自由度を示す数値です。

n
サンプル数を示す数値です。

alpha$_i$, *i* = 1, ...,*k*
この分布の最初の式のアルファ値を示すオプションの数値です。"平均の分析の計算" (998 ページ)を参照してください。

多対 1 の *t* 統計量: Dunnett の片側検定

- 自由度が有限の不揃いなケースで確率 *prob* と分位点 *q* を相関させます。*parameters* は $\lambda_1, \ldots, \lambda_k$、*nparms* の設定値は *k*、*df* の設定値は *v* です。式は次のとおりです。

$$prob = \int_0^{\infty} \int_{-\infty}^{\infty} \phi(y) \prod_{i=1}^{k} \Phi\left(\frac{\lambda_i y + qx}{\sqrt{1-\lambda_i^2}}\right) dy\, du_v(x)$$

- 自由度が有限の揃っているケースで確率 *prob* と分位点 *q* を相関させます。*parameters* は渡されず、$\left(\lambda = \sqrt{\frac{1}{2}}\right)$*nparms* の値は *k* に設定され、*df* の値は *v* に設定されます。式は次のとおりです。

$$prob = \int_0^{\infty} \int_{-\infty}^{\infty} \phi(y)[\Phi(y + \sqrt{2qx})]^k dy\, du_v(x)$$

- 自由度が無限の不揃いなケースで確率 *prob* と分位点 *q* を相関させます。*parameters* は $\lambda_1, \ldots, \lambda_k$、*nparms* の設定値は *k*、*df* の設定値は欠損です。式は次のとおりです。

$$prob = \int_{-\infty}^{\infty} \phi(y) \prod_{i=1}^{k} \Phi\left(\frac{\lambda_i y + q}{\sqrt{1-\lambda_i^2}}\right) dy$$

- 自由度が無限の揃っているケースで確率 *prob* と分位点 *q* を相関させます。*parameters* は渡されず、$\left(\lambda = \sqrt{\frac{1}{2}}\right)$*nparms* の値は *k* に設定され、*df* の値は欠損値に設定されます。式は次のとおりです。

$$prob = \int_{-\infty}^{\infty} \phi(y)[\Phi(y + \sqrt{2q})]^k dy$$

多対 1 の *t* 統計量: Dunnett の両側検定

- 自由度が有限の不揃いなケースで確率 *prob* と分位点 *q* を相関させます。*parameters* は $\lambda_1, \ldots, \lambda_k$、*nparms* の設定値は *k*、*df* の設定値は *v* です。式は次のとおりです。

$$prob = \int_{0}^{\infty}\int_{-\infty}^{\infty} \phi(y) \prod_{i=1}^{k} \left[\Phi\left(\frac{\lambda_i y + qx}{\sqrt{1-\lambda_i^2}} \right) - \Phi\left(\frac{\lambda_i y - qx}{\sqrt{1-\lambda_i^2}} \right) \right] dy\, du_\nu(x)$$

- 自由度が有限の揃っているケースで確率 *prob* と分位点 *q* を相関させます。*parameters* は渡されず、*nparms* の設定は *k*、*df* の設定値は *v* です。式は次のとおりです。

$$prob = \int_{0}^{\infty}\int_{-\infty}^{\infty} \phi(y)[\Phi(y+\sqrt{2qx}) - \Phi(y-\sqrt{2qx})]^k dy\, du_\nu(x)$$

- 自由度が無限の不揃いなケースで確率 *prob* と分位点 *q* を相関させます。*parameters* は $\lambda_1, \ldots, \lambda_k$、*nparms* の設定値は *k*、*df* の設定値は欠損です。式は次のとおりです。

$$prob = \int_{-\infty}^{\infty} \phi(y) \prod_{i=1}^{k} \left[\Phi\left(\frac{\lambda_i y + q}{\sqrt{1-\lambda_i^2}} \right) - \Phi\left(\frac{\lambda_i y - q}{\sqrt{1-\lambda_i^2}} \right) \right] dy$$

- 自由度が無限の揃っているケースで確率 *prob* と分位点 *q* を相関させます。*parameters* は渡されず、*nparms* の設定は *k*、*df* の設定値は欠損です。式は次のとおりです。

$$prob = \int_{-\infty}^{\infty} \phi(y)[\Phi(y+\sqrt{2q}) - \Phi(y-\sqrt{2q})]^k dy$$

パーティション分割範囲の計算

RANGE は、*n* のグループ平均のスチューデント化範囲の分布で使用されます。PARTRANGE は、分割されたスチューデント化範囲の分布で使用されます。*n* グループを、大きさが $n_1 + \ldots + n_k = n$ の *k* 個のサブセットに分割します。このとき、分割された範囲はそれぞれのサブセットのスチューデント化範囲の最大です。スチューデント化係数はすべてのケースで 同一です。

$$prob = \int_{0}^{\infty} \prod_{i=1}^{i=k} \left(\int_{-\infty}^{\infty} k\phi(y)(\Phi(y) - \Phi(y-qx))^{k-1} dy \right)^{n_i} d\mu_\nu(x)$$

この分布の構文は次のとおりです。

x=*probmc*('*partrange*', *q*, *p*,*nu*, *k*, n_1 ,...,n_k);

引数

x
返される結果の数値です(確率または分位点)。

q
分位点を示す数値です。

p
確率を示す数値です。*p* と *q* は、いずれかが欠損値である必要があります。

nu
自由度を示す数値です。

k
グループ数を示す数値です。

n_i i=1,...,k

この分布の式の *n* 値を示すオプションの数値です。“パーティション分割範囲の計算” (1000 ページ)を参照してください。

Student 化範囲

- 自由度が有限の揃っているケースで確率 *prob* と分位点 *q* を相関させます。*parameters* は渡されず、*nparms* の設定は *k*、*df* の設定値は *v* です。式は次のとおりです。

$$prob = \int_0^\infty \int_{-\infty}^\infty k\phi(y)[\Phi(y) - \Phi(y - qx)]^{k-1} dy\, du_\nu(x)$$

- 自由度が無限の不揃いなケースで確率 *prob* と分位点 *q* を相関させます。*parameters* は $\sigma_1, \ldots, \sigma_k$、*nparms* の設定値は *k*、*df* の設定値は欠損です。式は次のとおりです。

$$prob = \int_{-\infty}^\infty \sum_{j=1}^k \left\{ \prod_{i=1}^k \left[\Phi\left(\frac{y}{\sigma_i}\right) - \Phi\left(\frac{y-q}{\sigma_i}\right) \right] \right\} \phi\left(\frac{y}{\sigma_j}\right) \frac{1}{\sigma_j} dy$$

- 自由度が無限の揃っているケースで確率 *prob* と分位点 *q* を相関させます。*parameters* は渡されず、*nparms* の設定は *k*、*df* の設定値は欠損です。式は次のとおりです。

$$prob = \int_{-\infty}^\infty k\phi(y)[\Phi(y) - \Phi(y - q)]^{k-1} dy$$

Student 化最大モジュール

- 自由度が有限の不揃いなケースで確率 *prob* と分位点 *q* を相関させます。*parameters* は $\sigma_1, \ldots, \sigma_k$、*nparms* の設定値は *k*、*df* の設定値は *v* です。式は次のとおりです。

$$prob = \int_0^\infty \prod_{i=1}^k \left[2\Phi\left(\frac{qx}{\sigma_i}\right) - 1 \right] d\mu_\nu(x)$$

- 自由度が有限の揃っているケースで確率 *prob* と分位点 *q* を相関させます。*parameters* は渡されず、*nparms* の設定は *k*、*df* の設定値は *v* です。式は次のとおりです。

$$prob = \int_0^\infty [2\Phi(qx) - 1]^k d\mu_\nu(x)$$

- 自由度が無限の不揃いなケースで確率 *prob* と分位点 *q* を相関させます。*parameters* は $\sigma_1, \ldots, \sigma_k$、*nparms* の設定値は *k*、*df* の設定値は欠損です。式は次のとおりです。

$$prob = \prod_{i=1}^k \left[2\Phi\left(\frac{q}{\sigma_i}\right) - 1 \right]$$

- 自由度が無限の揃っているケースで確率 *prob* と分位点 *q* を相関させます。*parameters* は渡されず、*nparms* の設定は *k*、*df* の設定値は欠損です。式は次のとおりです。

$$prob = [2\Phi(q) - 1]^k$$

Williams の検定

PROBMC は、Williams (1971、1972)によって定義されている分布の確率または分位点を計算します。""参考文献"" (*SAS 関数と CALL ルーチン: リファレンス*)を参照してください。Williams 検定の必要性は、処方治療の平均をコントロール平均と比較し、治療で有効な最小の処方量を判断するときに発生します。

注: Williams 検定はサンプルサイズが揃っているときにのみ計算されます。

$X_1, X_2, ..., X_k$ は同一の独立している N(0,1)の確率変数であるものとします。ここで、Y_k は次の式で与えられる平均値を表します。

$$Y_k = \frac{X_1 + X_2 + ... + X_k}{k}$$

次の値の分布を計算する必要があります。

$$(Y_k - Z)/S$$

引数

Y_k
前述の定義のとおりです。

Z
N(0,1)の独立した確率変数です。

S
½νS^2 が χ^2 変数です(自由度 ν)。

Williams (1971)の説明によれば、全体の計算は非常に冗長であり、3 段階で行われます。""参考文献"" (*SAS 関数と CALL ルーチン: リファレンス*)を参照してください。

1 Y_k の分布を計算します。分布の計算はこの演算の基本的な(高コストな)部分で、Y_k の密度と確率の両方を求めるのに使われます。U_i をこの式に定義するとします。

$$U_i = \frac{X_1 + X_2 + ... + X_i}{i},\ i = 1, 2, ..., k$$

$Y_k > d$ の確率の再帰式を書くことができます。d の値は任意の実数になります。

$$\begin{aligned}\Pr(Y_k > d) &= \Pr(U_1 > d) \\ &\quad +\Pr(U_2 > d, U_1 < d) \\ &\quad +\Pr(U_3 > d, U_2 < d, U_1 < d) \\ &\quad +\ldots \\ &\quad +\Pr(U_k > d, U_{k-1} < d, \ldots, U_1 < d) \\ &\quad = \Pr(Y_{k-1} > d) + \Pr(X_k + (k-1)U_{k-1} > kd)\end{aligned}$$

この確率を計算するには、N(0,1)密度関数から開始します。

$$D(U_1 = x) = \phi(x)$$

そして、畳み込みを再帰的に計算します。

$$D(U_k = x, U_{k-1} < d, \ldots, U_1 < d) =$$
$$\int_{-\infty}^{d} D(U_{k-1} = y, U_{k-2} < d, \ldots, U_1 < d)(k-1)\phi(kx-(k-1)y)dy$$

この順次畳み込みから、前に示した$\Pr(Y_k < d)$の再帰方程式のすべての要素を計算することができます。

2 $Y_k - Z$ の分布を計算します。この計算には、確率を計算するための別の畳み込みが必要です。

$$\Pr((Y_k - Z) > d) = \int_{-\infty}^{\infty} \Pr(Y_k > \sqrt{2}d + y)\phi(y)dy$$

3 $(Y_k - Z)/S$ の分布を計算します。この計算には、確率を計算するための別の畳み込みが含まれます。

$$\Pr((Y_k - Z) > tS) = \int_{0}^{\infty} \Pr((Y_k - Z) > ty)d\mu_\nu(y)$$

ν = ∞のときは、第 3 段階は必要ありません。演算が複雑なため、この冗長なアルゴリズムは有限および無限の自由度 ν の両方で $k \le 15$ のとき、高速なアルゴリズムに置き換えられます。$k \ge 16$ のときは、冗長計算が実行されます。この計算は、アルゴリズムが複雑なため、きわめて高コストで非常に時間がかかります。

比較

SAS/STAT ソフトウェアの GLM プロシジャの MEANS ステートメントは次の検定を計算します。

- Dunnett の片側検定
- Dunnett の両側検定
- スチューデント化された範囲

例

例 1: PROBMC を使用した確率の計算

次のプログラムでは、確率の計算方法を示します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double par[5];
  dcl double df q prob;
  dcl char(20) test[3];
```

```
  dcl int i;
  method run();
    par := (.5 .51 .55 .45 .2);
    df = 40;
    q = 1;
    test := ('dunnett1' 'dunnett2' 'maxmod');
    do i = 1 to dim(test);
      prob=probmc(test[i], q, ., df, 5, par[1], par[2], par[3], par[4],
        par[5]);
      put test[i] $10. df q e18.13 prob e18.13;
    end;
  end;
enddata;
run;
quit;
```

SAS は次の結果をログに書き込みます。

```
dunnett1  40  1.00000000000E+00 4.82992196083E-01
dunnett2  40  1.00000000000E+00 1.64023105316E-01
maxmod    40  1.00000000000E+00 8.02784203408E-01
```

例 2: 平均の分析の計算

次のプログラムは、平均の分析を計算する方法を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double q1 q2 q3 q4;
  method run();
    q1=probmc('anom',.,0.9,.,20);
    put q1=;
    q2=probmc('anom',.,0.9,20,5,0.1,0.1,0.1,0.1,0.1);
    put q2=;
    q3=probmc('anom',.,0.9,20,5,0.5,0.5,0.5,0.5,0.5);
    put q3=;
    q4=probmc('anom',.,0.9,20,5,0.1,0.2,0.3,0.4,0.5);
    put q4=;
  end;
enddata;
run;
quit;
```

SAS は次の結果をログに書き込みます。

```
q1=2.78950610163346
q2=2.45499619666786
q3=2.45499619666786
q4=2.45323199941123
```

例 3: パーティション分割範囲の計算

次のプログラムは、パーティション分割範囲を計算する方法を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double q1 q2;
  method run();
    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=;
  end;
enddata;
run;
quit;
```

SAS は次の結果をログに書き込みます。

```
q1=4.1022397989482
q2=4.7888626337511
```

例 4: Williams の検定の計算

次のプログラムでは、8 つのブロックからなる乱塊法を使用し、7 つの水準である物質を検定しました。観測された処理の平均は次のとおりです。

処理	平均
X_0	10.4
X_1	9.9
X_2	10.0
X_3	10.6
X_4	11.4
X_5	11.9
X_6	11.7

自由度が (7 – 1)(8 – 1) = 42 である平均平方は、$s^2 = 1.16$ です。

平均化プロセスを経て最大尤度推定 M_i を求めます。

- $X_0 > X_1$ であるため、$X_{0,1} = (X_0 + X_1)/2 = 10.15$ となります。
- $X_{0,1} > X_2$ であるため、$X_{0,1,2} = (X_0 + X_1 + X_2)/3 = (2X_{0,1} + X_2)/3 = 10.1$ となります。
- $X_{0,1,2} < X_3 < X_4 < X_5$
- $X_5 > X_6$ であるため、$X_{5,6} = (X_5 + X_6)/2 = 11.8$ となります。

これで順序制約が満たされました。

対立仮説の最尤推定値は、次のようになります。

- $M_0 = M_1 = M_2 = X_{0,1,2} = 10.1$
- $M_3 = X_3 = 10.6$
- $M_4 = X_4 = 11.4$
- $M_5 = M_6 = X_{5,6} = 11.8$

$t = (11.8 - 10.4)/\sqrt{2s^2/8} = 2.60$を計算します。$k = 6$、$\nu = 42$、$t = 2.60$ の確率は、.9924467341 です。この物質に対する反応があるという強力な証拠を示しています。

また、次のプログラムに示すように、上の 5%および裾の 1%の分位点も計算できます。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double prob quant5 quant1;
  method run();
    prob=probmc('WILLIAMS',2.6,.,42,6);
    put prob=;
    quant5=probmc('WILLIAMS',.,.95,42,6);
    put quant5=;
    quant1=probmc('WILLIAMS',.,.99,42,6);
    put quant1= ;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
prob=0.99244668715827
quant5=1.80656253603894
quant1=2.49090827298707
```

参考文献

Guirguis, G. H., and R. D. Tobias. 2004. “On the Computation of the Distribution for the Analysis of Means.” *Communications in Statistics: Simulation and Computation* 33: 861-887.

Nelson, P. R. 1981. “Numerical evaluation of an equi-correlated multivariate non-central t distribution.” *Communications in Statistics: Part B - Simulation and Computation* 10: 41-50.

Nelson, P.R. 1982a. “An Approximation for the Complex Normal Probability Integral.” *BIT* 22(1): 94-100.

Nelson, P. R. 1982b. “Exact critical points for the analysis of means.” *Communications in Statistics: Part A - Theory and Methods* 11: 699-709.

Nelson, P. R. 1988. “Application of the Analysis of Means.” *Proceedings of the SAS Users Group International Conference* 13: 225-230.

Nelson, P. R. 1991. “Numerical evaluation of multivariate normal integrals with correlations.” *The Frontiers of Statistical Scientific Theory and Industrial Applications* 2: 97-114.

Nelson, P. R. 1993. “Additional Uses for the Analysis of Means and Extended Tables of Critical Values.” *Technometrics* 35: 61-71.

PROBMED 関数

サンプル中央値の累積確率を計算します。

カテゴリ: 確率

返されるデータの種類: DOUBLE

構文

PROBMED(*n*, *x*)

必須引数

n
サンプルサイズを指定します。

データの種類 DOUBLE

x
関心対象です。つまり、PROBMED 関数は、中央値が *x* 以下である確率を計算します。

データの種類　DOUBLE

詳細

PROBMED 関数は、 *n 個*の独立した標準正規確率変数(平均 0、分散 1)のサンプルについて、サンプルの中央値が *x* 以下である確率を計算します。

n がサンプルサイズを表し、 $x_{(i)}$が *i* 次統計量を表すとします。次に、 *n* が奇数の場合、関数は次の計算を行います。

$$\Pr\left[X_{((n+1)/2)} \le x\right] = \mathrm{I}_{\Phi(x)}\left(\frac{n+1}{2}, \frac{n+1}{2}\right)$$

次の式では、前述の式を参照します。

$$\mathrm{I}_p(a,b) = \frac{\mathrm{I}}{\mathrm{B}(a,b)}\int_0^p t^{a-1}(\mathrm{I}-t)^{b-1}dt$$

式$\mathrm{B}(a,b) = \Gamma(a)\Gamma(b)/\Gamma(a+b)$の Γ(。)はガンマ関数です。*n* が偶数の場合、PROBMED 関数は次の計算を実行します。

$$\Pr[\frac{X_{(n/2)} + X_{((n/2)+1)}}{2} \le x] =$$

$$\frac{2}{\mathrm{B}(\frac{n}{2}, \frac{n}{2})}\int_{-\infty}^{x}\{[\mathrm{I}-\Phi(u)]^{n/2} - [\mathrm{I}-\Phi(2x-u)]^{n/2}\}[\Phi(u)]^{(n/2)-1}\phi(u)du$$

この式では、$\mathrm{B}(n/2,\ n/2) = [\Gamma(n/2)]^2/\Gamma(n)$ 、および Φ(.)と φ(.)は、それぞれ、標準正規累積分布関数と密度関数です。

例

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double b;
  method run();
    b=probmed(5,-0.1);
    put b;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
0.42563808966747
```

参考文献

David, H.A. 1981. *Order Statistics*. 2nd 版 New York, , New York: John Wiley & Sons.

PROBNEGB 関数

負の二項分布の確率を返します。

カテゴリ: 確率

返されるデータの種類: DOUBLE

構文

PROBNEGB(*p*, *n*, *m*)

必須引数

p
数値の成功率パラメーターです。

範囲 $0 \le p \le 1$

データの種類 DOUBLE

n
成功数パラメーターです。この引数は整数にする必要があります。

範囲 $n \ge 1$

データの種類 DOUBLE

m
確率変数の失敗数です。この引数は正の整数にする必要があります。

範囲 $m \ge 0$

データの種類 DOUBLE

詳細

PROBNEGB 関数は、成功確率が *p*、成功数が *n* の負の二項分布のオブザベーションが *m* 以下となる確率を返します。

オブザベーションが指定した値の *m* と等しくなる確率を計算するには、負の二項分布からの 2 つの確率(*m* の確率と *m*-1 の確率)の差異を計算します。

例

次のプログラムは、PROBNEGB 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double x;
  method run();
    x=probnegb(0.5,2,1);
    put x;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
0.5
```

PROBNORM 関数

標準正規分布の確率を返します。

カテゴリ: 確率

返されるデータの種類: DOUBLE

構文

PROBNORM(*x*)

必須引数

x
数値の確率変数です。

データの種類 DOUBLE

詳細

PROBNORM 関数は、標準正規分布からのオブザベーションが *x* 以下となる確率を返します。

注: PROBNORM は PROBIT の逆関数です。

例

次のプログラムは、PROBNORM 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double x;
  method run();
    x=probnorm(1.96);
    put x;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
0.97500210485177
```

PROBT 関数

t 分布の確率を返します。

カテゴリ:	確率
返されるデータの種類:	DOUBLE

構文

PROBT(*x*, *df*[, *nc*])

必須引数

x
数値の確率変数です。

データの種類 DOUBLE

df
数値の自由度パラメーターです。

範囲 *df* > 0

データの種類 DOUBLE

オプション引数

nc
数値の非心度パラメーターです。

データの種類 DOUBLE

詳細

PROBT 関数は、自由度が *df* で非心度パラメーターが *nc* のスチューデントの *t* 分布のオブザベーションが *x* 以下となる確率を返します。この関数は、自由度パラメーター *df* に非整数を指定できます。パラメーター *nc* (省略可能)が指定されていないかゼロの場合、値は心度スチューデントの *t* 分布から返されます。

両側 *t* 検定の有意水準は、次の式で求められます。

```
p=(1-probt(abs(x),df))*2;
```

例

次のプログラムは、PROBT 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double x;
  method run();
    x=probt(0.9,5);
    put x;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
0.79531439982768
```

PROGRAMBLOCKLINENUMBER 関数

現在のプログラムブロックの先頭から始まる現在のステートメントの行番号を返します。

カテゴリ:	特殊
返されるデータの種類:	INTEGER

構文

PROGRAMBLOCKLINENUMBER()

引数なし

PROGRAMBLOCKLINENUMBER()関数には引数がありません。

詳細

ProgramBlockLineNumber 関数は、現在のパッケージ、スレッド、またはデータブロックの先頭からオフセットされた現在のステートメントの行番号を返します。プログラムブロックの行番号は、プログラムブロックを開始する PACKAGE、THREAD、または DATA ステートメントで 1 から始まり、プログラムブロックを終了する ENDPACKAGE、ENDTHREAD、または ENDDATA ステートメントまで順番に続きます。

例

次のプログラムは、PROGRAMBLOCKLINENUMBER 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
resetline;
proc ds2;
thread testthread / overwrite=yes;
  dcl char a;
  dcl int threadline;
```

```
    method init();
      a = 'apple';
      threadline = programblocklinenumber();
      put threadline=;
    end;
  endthread;

  data _null_;
    dcl thread testthread t();
    dcl int dataline;
    dcl char b;
    method init();
      b = 'banana';
      dataline = programblocklinenumber();
      put dataline=;
    end;
    method run();
      set from t threads = 1;
   end;
  enddata;
  run;
  quit;
```

SAS は次の出力をログに書き込みます。

```
78   resetline;
1    proc ds2;
2
3    thread testthread / overwrite=yes;
4     dcl char a;
5     dcl int threadline;
6     method init();7       a = 'apple';
8       threadline = programblocklinenumber();
9     put threadline=;
10    end;
11   endthread;
12
13   data _null_;
14   dcl thread testthread t();
15   dcl int dataline;
16   dcl char b;
17    method init();
18      b = 'banana';
19      dataline = programblocklinenumber();
20      put dataline=;
21    end;
22
23    method run();
24     set from t threads = 1;
25    end;
26   enddata;
27   run;
dataline=7

threadline=6
```

関連項目

“SASLOGLINENUMBER 関数” (1103 ページ)

PROGRAMBLOCKLNAME 関数

現在のプログラミングブロックの名前を返します。

カテゴリ: 特殊

返されるデータの種類: CHAR

構文

PROGRAMBLOCKNAME()

引数なし

PROGRAMBLOCKNAME()関数には引数がありません。

詳細

ProgramBlockName 関数は、現在のプログラムブロックの名前を返します。パッケージおよびスレッドプログラムブロックの 場合、パッケージまたはスレッドのユーザー定義名が返されます。データプログラムブロックの場合、'block'が返されるため、プログラムブロックタイプとプログラムブロック 名は'data block'になります。

例

次のプログラムは、PROGRAMBLOCKNAME 関数を示しています。

注:

この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
resetline;
proc ds2;

thread testthread / overwrite=yes;
  dcl char a;
  dcl varchar(100) threadname;
  method init();
    a = 'apple';
    threadname = programblockname();
    put threadname=;
  end;
```

```
  endthread;

  data _null_;
    dcl thread testthread t();
    dcl varchar(100) dataname;
    dcl char b;
    method init();
      b = 'banana';
      dataname = programblockname();
      put dataname=;
    end;

    method run();
      set from t threads = 1;
    end;
  enddata;
  run;
  quit;
```

SAS は次の出力をログに書き込みます。

```
78   resetline;
1    proc ds2;
2
3    thread testthread / overwrite=yes;
4     dcl char a;
5     dcl varchar(100) threadname;
6     method init();
7      a = 'apple';
8       threadname = programblockname();
9      put threadname=;
10     end;
11   endthread;
12
13   data _null_;
14   dcl thread testthread t();
15   dcl varchar(100) dataname;
16   dcl char b;
17     method init();
18       b = 'banana';
19       dataname = programblockname();
20     put dataname=;
21     end;
22
23     method run();
24      set from t threads = 1;
25     end;
26   enddata;
27   run;
dataname=block
threadname=testthread
```

関連項目

“PROGRAMBLOCKTYPE 関数” (1017 ページ)

PROGRAMBLOCKTYPE 関数

現在のプログラミングブロックのタイプを返します。

カテゴリ: 特殊

返されるデータの種類: CHAR

構文

PROGRAMBLOCKTYPE()

引数なし

PROGRAMBLOCKTYPE()関数には引数がありません。

詳細

ProgramBlockType 関数は、現在のプログラムブロックのタイプ('package'、'thread'、または'data')を返します。

例

次のプログラムは、PROGRAMBLOCKTYPE 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
resetline;
proc ds2;
thread testthread / overwrite=yes;
  dcl char a;
  dcl varchar(100) threadtype;
  method init();
    a = 'apple';
    threadtype = programblocktype();
    put threadtype=;
 end;
endthread;

data _null_;
  dcl thread testthread t();
```

```
    dcl varchar(100) datatype;
    dcl char b;
    method init();
      b = 'banana';
      datatype = programblocktype();
      put datatype=;
    end;

    method run();
      set from t threads = 1;
    end;
  enddata;
  run;
  quit;
```

SAS は次の出力をログに書き込みます。

```
78   resetline;
1    proc ds2;
2
3    thread testthread / overwrite=yes;
4      dcl char a;
5      dcl varchar(100) threadtype;
6      method init();
7    a = 'apple';
8        threadtype = programblocktype();
9    put threadtype=;
10     end;
11   endthread;
12
13   data _null_;
14   dcl thread testthread t();
15   dcl varchar(100) datatype;
16   dcl char b;
17     method init();
18       b = 'banana';
19       datatype = programblocktype();
20     put datatype=;
21     end;
22
23     method run();
24   set from t threads = 1;
25     end;
26   enddata;
27   run;
27 !      quit;
datatype=data
threadtype=thread
```

関連項目

“PROGRAMBLOCKLNAME 関数” (1015 ページ)

PRXCHANGE 関数

パターンマッチングの置換を実行します。

カテゴリ: 文字列マッチング

制限事項: この関数は、PCRX パッケージによって置き換えられました。

返されるデータの種類: CHAR

注: SAS は、Unicode 文字列データに一致する正規表現を実装するために International Components for Unicode (ICU)を採用しています。詳細については、正規表現を参照してください。

構文

PRXCHANGE(*perl-regular-expression* | *regular-expression-id*, *times*, *source*)

引数

perl-regular-expression
Perl 正規表現を値とする文字定数、変数または式を指定します。

データの種類 CHAR

regular-expression-id
PRXPARSE 関数によって返されるパターン識別子の値が含まれる数値変数を指定します。

制限事項 この引数を使用する場合、PRXPARSE 関数も使用する必要があります。

データの種類 INTEGER

times
一致を検索して一致するパターンを置換する回数を指定する数値の定数、変数または式です。

データの種類 INTEGER

ヒント *times* の値が–1 の場合、*source* の末尾に到達するまで一致したパターンが置換され続けます。

source
検索する文字定数、変数または式を指定します。

データの種類 CHAR

詳細

基本

regular-expression-id を使用すると、PRXCHANGE 関数は PRXPARSE から返された *regular-expression-id* で *source* を検索します。*source* 中の値は正規表現で指定された変更が加えられて返されます。一致がない場合、PRXCHANGE は未変更の *source* の値を返します。

perl-regular-expression を使用すると、PRXCHANGE は *perl-regular-expression* を使って *source* を検索します。PRXPARSE を呼び出す必要はありません。PRXCHANGE は WHERE 句と PROC SQL の中の *perl-regular-expression* で使用できます。

注: DS2 の PRX 関数には次の制限が適用されます。

- PRX 形式の/.../.../で使用できるのは **m**、**i**、**s**、**x** のみで、前にも後にも 1 文字を置くことができます。
- マッチングモード修飾子の **p**、**o**、**c**、**a**、**l** はサポートされていません。
- マッチングモード修飾子 **g** がサポートされています。

パターンマッチングの詳細については、“Perl 正規表現(PRX)を使用したパターンマッチング ” (*SAS 関数と CALL ルーチン: リファレンス*)を参照してください。

Perl 正規表現のコンパイル

perl-regular-expression が定数の場合、または/o オプションが指定されている場合、一度コンパイルされた Perl 正規表現が PRXCHANGE の使用のたびに再利用されます。*perl-regular-expression* が定数ではなく、/o オプションが指定されていない場合は、PRXCHANGE の呼び出しのたびに Perl 正規表現が再コンパイルされます。

注: コンパイル 1 回のみの動作は、DS2 環境、WHERE 句または PROC SQL の中で PRXCHANGE を使用する場合に発生します。他のすべての使用では、PRXCHANGE を呼び出すたびに *perl-regular-expression* が再コンパイルされます。

照合の実行

Perl 正規表現は、メタ文字と呼ばれる文字と特殊文字で構成されます。照合を実行すると、SAS は Perl 正規表現で指定された部分文字列をソース文字列から検索します。

コードを記述するときに使用可能な Perl 正規表現のメタ文字の簡易リストは、“Perl 正規表現(PRX)のメタ文字テーブル” (*SAS 関数と CALL ルーチン: リファレンス*)の表で確認できます。メタ文字の完全なリストについては、Perl の Web サイトを参照してください。

比較

SAS PRX*関数は、DATA ステップとの互換性のために DS2 で提供されています。
PCRX パッケージは、DS2 言語によって提供される文字マッチング機能です。

例

例 1: 姓名の順序を入れ替える

次のプログラムは、姓名の順序を入れ替えます。

```
/* Create a table that contains a list of names. */
proc ds2;
data names;
  dcl char(32) name;
   method run();
     name='Jones, Fred'; output;
     name='Kavich, Kate'; output;
     name='Turley, Ron'; output;
     name='Dulix, Yolanda'; output;
  end;
enddata;
run;
quit;

  /* Reverse last and first names */
proc ds2;
data ReversedNames;
  method run();
     set names;
     name=prxchange('s/(\w+), (\w+)/$2 $1/', -1, name);
  end;
enddata;
run;
quit;

proc print data=ReversedNames;
run;
```

アウトプット 7.13　PRXCHANGE 関数の結果

The SAS System

Obs	name
1	Fred Jones
2	Kate Kavich
3	Ron Turley
4	Yolanda Dulix

例 2: 一致するパターンの固定値への変更

このプログラムでは、変数中のパターンを検索し、事前に定義した値に変数を置換します。プログラムは電話番号を検索し、情報メッセージに置き換えます。

```
/* Create table that contains confidential information. */
proc ds2;
data a;
  dcl char(95) text;
  method run();
    text='The phone number for Ed is (801)443-9876 but not until tonight.';
    output;
    text='He can be reached at (910)998-8762 tomorrow for testing purposes.';
    output;
  end;
enddata;
run;
quit;

proc print data=a;
run;

  /* Locate confidential phone numbers and replace them with message */
  /* indicating that they have been removed.                         */
proc ds2;
data b;
  method run();
    set a;
    text=prxchange('s/\([2-9]\d\d\) ?[2-9]\d\d-\d\d\d\d/*PHONE NUMBER REMOVED*/'
      , -1, text);
  end;
enddata;
run;
quit;

proc print data=b;
run;
```

アウトプット 7.14　一致するパターンから固定値への変更前の結果

The SAS System

Obs	text
1	The phone number for Ed is (801)443-9876 but not until tonight.
2	He can be reached at (910)998-8762 tomorrow for testing purposes.

アウトプット 7.15　一致するパターンから固定値への変更の結果

The SAS System

Obs	text
1	The phone number for Ed is *PHONE NUMBER REMOVED* but not until tonight.
2	He can be reached at *PHONE NUMBER REMOVED* tomorrow for testing purposes.

関連項目

関数:

- “PRXMATCH 関数” (1023 ページ)
- “PRXPAREN 関数” (1027 ページ)
- “PRXPARSE 関数” (1031 ページ)
- “PRXPOSN 関数” (1034 ページ)

パッケージ:

- “DS2 PCRXFIND パッケージのメソッド、演算子、およびステートメント” (1893 ページ)
- “DS2 PCRXREPLACE パッケージのメソッド、演算子、およびステートメント” (1911 ページ)

PRXMATCH 関数

パターンの一致を検索し、見つかったパターンの位置を返します。

カテゴリ:	文字列マッチング
制限事項:	この関数は、PCRX パッケージによって置き換えられました。
返されるデータの種類:	INTEGER

注: SAS は、Unicode 文字列データに一致する正規表現を実装するために International Components for Unicode (ICU)を採用しています。詳細については、正規表現を参照してください。

構文

PRXMATCH(*perl-regular-expression*, *source*)

引数

perl-regular-expression
Perl 正規表現を値とする文字定数、変数または式を指定します。

データの種類 CHAR

参照項目: Perl 正規表現(PRX)メタ文字のテーブル

source
検索する文字定数、変数または式を指定します。

データの種類 CHAR

詳細

基本

perl-regular-expression を使用すると、PRXMATCH 関数は *perl-regular-expression* で *source* を検索し、文字列の開始位置を返します。一致が見つからない場合、PRXMATCH は 0 を返します。

WHERE 句と PROC SQL で、PRXMATCH に Perl 正規表現を使用できます。

注: DS2 の PRX 関数には次の制限が適用されます。

- PRX 形式の/.../.../で使用できるのは **m**、**i**、**s**、**x** のみで、前にも後にも 1 文字を置くことができます。
- マッチングモード修飾子の **p**、**o**、**c**、**a**、**l** はサポートされていません。

パターンマッチングの詳細については、 "Perl 正規表現(PRX)を使用したパターンマッチング " (*SAS 関数と CALL ルーチン: リファレンス*)を参照してください。

Perl 正規表現のコンパイル

perl-regular-expression が定数の場合、または/o オプションが指定されている場合、一度コンパイルされた Perl 正規表現が PRXMATCH の使用のたびに再利用されます。*perl-regular-expression* が定数ではなく、/o オプションを使用しない場合、PRXMATCH を呼び出すたびに Perl 正規表現が再コンパイルされます。

比較

SAS PRX*関数は、DATA ステップとの互換性のために DS2 で提供されています。PCRX パッケージは、DS2 言語によって提供される文字マッチング機能です。

例

例 1: PRXPARSE を使用した部分文字列の位置の検索

次のプログラムでは、文字列から部分文字列を検索し、その部分文字列の文字列内の位置を返します。

```
proc ds2;
 data _null_;
  dcl double patternID position;
  method init();
    /* 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=;
 end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
position=7
```

例 2: Perl 正規表現を使用した部分文字列の位置の検索

次のプログラムでは、Perl 正規表現を使用して文字列(Hello world)から部分文字列(world)を検索し、その部分文字列の文字列内の位置を返します。

```
proc ds2;
data _null_;
  dcl double position;
  method run();
    position=prxmatch('/world/', 'Hello world!');
    put position=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
position=7
```

例 3: ZIP コードの抽出

次のプログラムでは、テーブルの各行で 9 桁の ZIP コードを検索し、それらの行をテーブル ZipPlus4 に書き込みます。

注: バックスラッシュ(\)の前に、エスケープ文字として機能する別のバックスラッシュ(\)を付ける必要があります。

```
proc ds2;
data ZipCodes (overwrite=yes);
  dcl char(16) name;
  dcl char(10) zip;
  method run();
    name='Jonathan'; zip='32532-2343'; output;
    name='Seth'; zip='85030'; output;
    name='Kim'; zip='39204'; output;
    name='Samuel'; zip='93849-3843'; output;
  end;
 enddata;
 run;
quit;

proc print data=ZipCodes;
run;

proc ds2;
data ZipPlus4 (overwrite=yes);
  method run();
    set zipcodes;
    select;
      when (prxmatch('/\d{5}-\d{4}/', zip))
      output;
    end;
  end;
enddata;
run;
quit;

proc print data=ZipPlus4;
run;
```

アウトプット 7.16　オリジナルの ZIP コードテーブルの出力

The SAS System

Obs	name	zip
1	Jonathan	32532-2343
2	Seth	85030
3	Kim	39204
4	Samuel	93849-3843

アウトプット 7.17　9 桁の ZIP コードの出力

The SAS System

Obs	name	zip
1	Jonathan	32532-2343
2	Samuel	93849-3843

関連項目

関数:

- “PRXCHANGE 関数” (1019 ページ)
- “PRXPARSE 関数” (1031 ページ)
- “PRXPOSN 関数” (1034 ページ)

パッケージ:

- “DS2 PCRXFIND パッケージのメソッド、演算子、およびステートメント” (1893 ページ)
- “DS2 PCRXREPLACE パッケージのメソッド、演算子、およびステートメント” (1911 ページ)

PRXPAREN 関数

パターンに最後に一致したかっこを返します。

カテゴリ:	文字列マッチング
制限事項:	この関数は、PCRX パッケージによって置き換えられました。
要件	PRXPARSE 関数と一緒に使用します。
返されるデータの種類:	INTEGER
注:	SAS は International Components for Unicode (ICU)を採用しました。ICU により、SAS は Unicode 文字列データに正規表現マッチングを適用できるようになります。詳細については、正規表現を参照してください。

構文

PRXPAREN(*regular-expression-id*)

必須引数

regular-expression-id
PRXPARSE 関数によって返される数値パターン識別子を指定します。

データの種類 INTEGER

詳細

PRXPAREN 関数は、PRXPOSN 関数に渡すことができる最大のキャプチャバッファー番号を見つけたり、パターンのどの部分が一致したかを識別するのに役立ちます。

詳細については、“Perl 正規表現(PRX)を使用したパターンマッチング ” (*SAS 関数と CALL ルーチン: リファレンス*)を参照してください。

比較

SAS PRX*関数は、DATA ステップとの互換性のために DS2 で提供されています。PCRX パッケージは、DS2 言語によって提供される文字マッチング機能です。

例: PRXPAREN を使用した引用符のグループ化

次のプログラムは、PRXPAREN 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
```

```
 /* Create some source data to work with */
data all_quotes/overwrite=yes;
  dcl char(120) quote;
  method init();
    quote='A room without books is like a body without a soul.';output;
    quote='When I have a little money, I buy books; and if I have any left, I buy food and
clothes.';output;
    quote='Books are mirrors: you only see in them what you already have inside
you.';output;
    quote='The secret of getting ahead is getting started.';output;
    quote='When I was a kid, National Geographic was my favorite magazine.';output;
    quote='If you don''t read the newspaper, you''re uninformed. If you read the
newspaper, you''re mis-informed.';output;
    quote='To look at the newspaper is to raise a seashell to one''s ear and to be
overwhelmed by the roar of humanity.';output;
  end;
run;

/* Read source data and distribute rows to appropriate subsets */
data book_quotes
   magazine_quotes
   newspaper_quotes/overwrite=yes;
  dcl int rxID ;
  retain rxID 0;
  drop rxID;
  method init();
    rxID=prxparse('/(book)|(magazine)|(newspaper)/i');
  end;
  method run();
    dcl int rc pattern;
    set all_quotes;
    /* Use PRXMATCH to search for a match and return a position, if found.  */
    rc=prxmatch(rxID,quote);
    if not rc then do;
      /* Message for quotes that did not match any pattern */
      put 'This quote didn''t match:';
      put quote=;
    end;
    else do;
      /* When a match is found, identify which pattern matched. */
      pattern=prxparen(rxID);
      /* Output the quote to the appropriate subset:
        1=book, 2=magazine, 3=newspaper */
      if pattern=1 then output book_quotes;
      else if pattern=2 then output magazine_quotes;
      else if pattern=3  then output newspaper_quotes;
    end;
  end;
enddata;
run;
quit;

/* Print the input and output data sets. */
title "All Quotes";
proc print data=all_quotes; run;
title "Book Quotes";
```

```
proc print data=book_quotes; run;
title "Magazine Quotes";
proc print data=magazine_quotes; run;
title "Newspaper Quotes";
proc print data=newspaper_quotes; run;
title;
```

例の出力は次のとおりです。

All Quotes

Obs	quote
1	A room without books is like a body without a soul.
2	When I have a little money, I buy books; and if I have any left, I buy food and clothes.
3	Books are mirrors: you only see in them what you already have inside you.
4	The secret of getting ahead is getting started.
5	When I was a kid, National Geographic was my favorite magazine.
6	If you don't read the newspaper, you're uninformed. If you read the newspaper, you're mis-informed.
7	To look at the newspaper is to raise a seashell to one's ear and to be overwhelmed by the roar of humanity.

Book Quotes

Obs	quote
1	A room without books is like a body without a soul.
2	When I have a little money, I buy books; and if I have any left, I buy food and clothes.
3	Books are mirrors: you only see in them what you already have inside you.

Magazine Quotes

Obs	quote
1	When I was a kid, National Geographic was my favorite magazine.

Newspaper Quotes

Obs	quote
1	If you don't read the newspaper, you're uninformed. If you read the newspaper, you're mis-informed.
2	To look at the newspaper is to raise a seashell to one's ear and to be overwhelmed by the roar of humanity.

関連項目

関数:

- “PRXCHANGE 関数” (1019 ページ)
- PRXMATCH
- PRXPARSE
- “PRXPOSN 関数” (1034 ページ)

パッケージ:

- “DS2 PCRXFIND パッケージのメソッド、演算子、およびステートメント” (1893 ページ)
- “DS2 PCRXREPLACE パッケージのメソッド、演算子、およびステートメント” (1911 ページ)

PRXPARSE 関数

文字値のパターンマッチングに使用できる Perl 正規表現(PRX)をコンパイルします。

カテゴリ: 文字列マッチング

制限事項: この関数は、PCRX パッケージによって置き換えられました。
他の Perl 正規表現とともに使用します。

返されるデータの種類: INTEGER

注: SAS は、Unicode 文字列データに一致する正規表現を実装するために International Components for Unicode (ICU)を採用しています。詳細については、正規表現を参照してください。

構文

regular-expression-id=**PRXPARSE**(*perl-regular-expression*)

引数

regular-expression-id
PRXPARSE 関数によって返される数値のパターン識別子です。

データの種類 INTEGER

perl-regular-expression
Perl 正規表現を値とする文字、定数、変数、式のいずれかを指定します。

データの種類 CHAR

詳細

基本

PRXPARSE 関数は、他の Perl 関数でパターンをマッチングするために使用するパターン識別子番号を返します。正規表現の解析でエラーが発生すると、SAS は欠損値を返します。

PRXPARSE は、Perl 正規表現の構成にメタ文字を使用します。一般的なメタ文字の表については、"Perl 正規表現(PRX)のメタ文字テーブル" (*SAS 関数と CALL ルーチン: リファレンス*)を参照してください。

注: DS2 の PRX 関数には次の制限が適用されます。

- PRX 形式の/.../.../で使用できるのは **m**、**i**、**s**、**x** のみで、前にも後にも 1 文字を置くことができます。
- マッチングモード修飾子の **p**、**o**、**c**、**a**、**l** はサポートされていません。

パターンマッチングの詳細については、"Perl 正規表現(PRX)を使用したパターンマッチング " (*SAS 関数と CALL ルーチン: リファレンス*)を参照してください。

Perl 正規表現のコンパイル

perl-regular-expression が定数の場合、Perl 正規表現は 1 回のみコンパイルされます。後続の PRXPARSE の呼び出しでは再コンパイルされず、すでにコンパイル済みの正規表現の *regular-expression-id* が返されます。この動作では、初期化ブロック(IF _N_ =1)を使用して Perl 正規表現を初期化する必要がないため、コードが簡素化されます。

比較

SAS PRX*関数は、DATA ステップとの互換性のために DS2 で提供されています。
PCRX パッケージは、DS2 言語によって提供される文字マッチング機能です。

例

例 1: Perl 正規表現のコンパイル

次のプログラムは、PRXPARSE を使用して Perl 正規表現をコンパイルします。

```
proc ds2;
 data _null_;
  dcl double patternID position;
  method init();
    /* 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=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
position=7
```

例 2: PRXPARSE を使用して姓と名を逆にする

次のプログラムは、PRXPARSE を使用して姓と名を逆にします。

```
proc ds2;
 data _null_;
  dcl double patternID position;
  dcl char(32) names[4];
  dcl int i;
  method init();
     names := ('Jones, Fred   '
               'Kavich, Kate  '
               'Turley, Ron   '
               'Dulix, Yolanda');

     /* Reverse last and first names */
     do i = 1 to dim(names);
       names[i]=prxchange('s/(\w+), (\w+)/$2 $1/', -1, names[i]);
       put names[i];
     end;
  end;
enddata;
```

```
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
Fred Jones
Kate Kavich
Ron Turley
Yolanda Dulix
```

関連項目

関数:

- “PRXCHANGE 関数” (1019 ページ)
- “PRXMATCH 関数” (1023 ページ)
- “PRXPAREN 関数” (1027 ページ)
- “PRXPOSN 関数” (1034 ページ)

パッケージ:

- “DS2 PCRXFIND パッケージのメソッド、演算子、およびステートメント” (1893 ページ)
- “DS2 PCRXREPLACE パッケージのメソッド、演算子、およびステートメント” (1911 ページ)

PRXPOSN 関数

キャプチャバッファーの値が含まれる文字列を返します。

カテゴリ:	文字列マッチング
制限事項:	この関数は、PCRX パッケージによって置き換えられました。
返されるデータの種類:	CHAR
注:	SAS は、Unicode 文字列データに一致する正規表現を実装するために International Components for Unicode (ICU)を採用しています。詳細については、正規表現を参照してください。

構文

PRXPOSN(*regular-expression-id*, *capture-buffer*, *source*)

引数

regular-expression-id
PRXPARSE 関数によって返されるパターン識別子の値が含まれる数値変数を指定します。

データの種類 INTEGER

capture-buffer
値を取得するキャプチャバッファーを識別する数値定数、変数または式です。

- *capture-buffer* の値が 0 の場合、PRXPOSN は一致全体を返します。
- *capture-buffer* の値が 1 から正規表現の開始かっこ数までの間の場合、PRXPOSN はそのキャプチャバッファーの値を返します。
- *capture-buffer* の値が開始かっこの数よりも大きい場合、PRXPOSN は欠損値を返します。

データの種類 INTEGER

source
キャプチャバッファーを抽出するテキストを指定します。

データの種類 CHAR

詳細

PRXPOSN 関数は、PRXMATCH または PRXCHANGE の結果を使用してキャプチャバッファーを返します。PRXPOSN が有益な情報を返すには、これらの関数のいずれかで一致が見つかる必要があります。

注: DS2 の PRX 関数には次の制限が適用されます。

- PRX 形式の/.../.../で使用できるのは **m**、**i**、**s**、**x** のみで、前にも後にも 1 文字を置くことができます。
- マッチングモード修飾子の **p**、**o**、**c**、**a**、**l** はサポートされていません。

パターンマッチングの詳細については、“Perl 正規表現(PRX)を使用したパターンマッチング ” (*SAS 関数と CALL ルーチン: リファレンス*)を参照してください。

比較

SAS PRX*関数は、DATA ステップとの互換性のために DS2 で提供されています。PCRX パッケージは、DS2 言語によって提供される文字マッチング機能です。

例

例 1: 姓と名の抽出

次のプログラムでは、PRXPOSN を使用してテーブルから姓と名を抽出します。

```
proc ds2;
data ReversedNames;
  dcl char(32) name;
  method init();
    name='Jones, Fred'; output;
    name='Kavich, Kate'; output;
    name='Turley, Ron'; output;
    name='Dulix, Yolanda'; output;
  end;
enddata;
run;
quit;

proc ds2;
data FirstLastNames (overwrite=yes);
  dcl char(16) first last;
  dcl double re;
  keep first last;
  retain re;

  method init();
    dcl varchar(32) expression;
    expression = '/(\w+), (\w+)/';
    re=prxparse(expression);
    if missing( re ) then do;
      put 'ERROR: Invalid expression ' expression;
      stop;
    end;
  end;

  method run();
    set ReversedNames;
    if prxmatch(re, name) then
      do;
        last=prxposn(re, 1, name);
        first=prxposn(re, 2, name);
      end;
  end;
enddata;
run;
quit;

proc print data=FirstLastNames;
run;
```

アウトプット 7.18　PRXPOSN からの出力: 姓と名

The SAS System

Obs	first	last
1	Fred	Jones
2	Kate	Kavich
3	Ron	Turley
4	Yolanda	Dulix

例 2: 一部の名前が無効な場合の名前の抽出

次のプログラムでは、名前のリストを含むテーブルを作成します。名のみまたは姓のみを含む行は無効です。PRXPOSN はテーブルから有効な名前を抽出し、その名前をテーブル NEW に書き込みます。

```
proc ds2;
data old (overwrite=yes);
  dcl char(60) name;
  method init();
    name='Judith S Reaveley'; output;
    name='Ralph F. Morgan'; output;
    name='Jess Ennis'; output;
    name='Carol Echols'; output;
    name='Kelly Hansen Huff'; output;
    name='Judith'; output;
    name='Nick'; output;
    name='Jones'; output;
  end;
enddata;
run;
quit;

proc ds2;
data new (overwrite=yes);
  dcl char(40) first middle last;
  keep first middle last;

  method run();
    re=prxparse('/(\S+)\s+([^\s]+\s+)?(\S+)/i');
    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;
  end;
enddata;
run;
quit;
```

```
proc print data=new;
run;
```

アウトプット 7.19　有効な名前の出力

The SAS System

Obs	first	middle	last
1	Judith	S	Reaveley
2	Ralph	F.	Morgan
3	Jess		Ennis
4	Carol		Echols
5	Kelly	Hansen	Huff

関連項目

関数:

- “PRXCHANGE 関数” (1019 ページ)
- “PRXMATCH 関数” (1023 ページ)
- “PRXPAREN 関数” (1027 ページ)
- “PRXPARSE 関数” (1031 ページ)

パッケージ:

- “DS2 PCRXFIND パッケージのメソッド、演算子、およびステートメント” (1893 ページ)
- “DS2 PCRXREPLACE パッケージのメソッド、演算子、およびステートメント” (1911 ページ)

PUT 関数

指定した出力形式を使用して値を返します。

カテゴリ:	特殊
返されるデータの種類:	CHAR、NCHAR、NVARCHAR、VARCHAR

構文

PUT(*expression*, *format*)

必須引数

expression
任意の有効な式を指定します。

データの種類 DOUBLE、DATE、TIME、TIMESTAMP、CHAR、NCHAR、NVARCHAR、VARCHAR

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

format.
expression に適用する DS2 出力形式またはユーザー定義出力形式のいずれかを指定します。

デフォルトの配置をオーバーライドするには、配置指定を出力形式に追加します。

–L
値を左揃えにする

–C
値を中央揃えにする

–R
値を右揃えにする

詳細

出力形式の幅または小数点以下の桁数の指定に値が指定されていない場合、DS2 はその出力形式のデフォルト値を使用します。入力幅に対応した出力形式を使用する場合、幅、小数点以下の桁数の指定、またはその両方が入力フィールドの幅から計算されることに注意してください。詳細については、"DS2 でフォーマットされたデータを書き込む" (69 ページ)を参照してください。

expression が出力形式の型(数値または文字)に対して有効なデータ型でない場合、DS2 は、次の例外を除いて、*expression* を *format* に対して有効なデータ型に変換します。

- 日付と時間の式は、数値出力形式の場合は SAS 日付、時間、または日時の DOUBLE 値に変換され、文字列出力形式の場合は NCHAR に変換されます
- 出力形式が$BINARY、$HEX、または$OCTAL などのバイナリ文字出力形式の場合、データ型が DOUBLE の式は NCHAR に変換されます
- VARBINARY のデータ型を生成しない数値出力形式で VARBINARY のデータ型の式を使用すると、エラーが発行されます

DS2 が割り当てステートメントで式のデータ型を変換すると、結果は左揃えになります。

PUT 関数を使用して、数値を文字値に変換したり、日付、時間、またはタイムスタンプ値を SAS 日付/時間値に変換したりできます。

比較

PUT 関数と PUT ステートメントの動作は似ています。ただし、PUT ステートメントはその結果を SAS ログに送信しますが、PUT 関数はその引数のフォーマット結果を含む NCHAR 値を返します。

例

例 1: PUT による値のフォーマット

次のプログラムは、PUT 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl char x;
  method run();
    x=put(17180, date7.);
    put x;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
14JAN07
```

```
proc ds2;
data test(overwrite=yes);
  dcl double x;
  method run();
    x=put('AB', $binary.);
    put x;
    end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
100000101000010
```

```
proc ds2;
```

```
data test(overwrite=yes);
  dcl double a x;
  method run();
    a=35436745.3354;
    x=put(a, comma20.4);
    put x;
  end;
enddata;
run;quit;
```

SAS は次の出力をログに書き込みます。

```
35436745.3354
```

```
proc ds2;
data test(overwrite=yes);
  dcl double a x;
  method run();
    a=35436745.3354;
    x=put(a, comma20.4 -L);
    put x;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
35436745.3354
```

例 2: PROC FORMAT 出力形式の使用

次のプログラムは、ユーザー定義出力形式を作成し、その出力形式を PUT 関数で使用します。

```
proc format;
value abc 1="Yes" 2="No";
run;

proc ds2;
data test (overwrite=yes);
  dcl double d e;
  dcl char(3) x y;
  method run();
    d=1;
    x=put(d, abc.);
    put x=;
    e=2;
    y=put(e, abc.);
    put y=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
x=Yes
y=No
```

関連項目

関数:

- “INPUTC 関数” (712 ページ)
- “INPUTN 関数” (714 ページ)

PUTC 関数

実行時に文字の出力形式を指定できるようにします。

カテゴリ:	特殊
返されるデータの種類:	CHAR、NCHAR、NVARCHAR、VARCHAR

構文

PUTC(*value*, *format-specification* [, *w*])

必須引数

value
フォーマットされる文字値を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

format-specification
value に適用する文字出力形式です。

有効な出力形式の指定方法は次のようになります。

- *format-name*
- *format-name.*(出力形式名はピリオドで終わります)
- *format-namew.*

format-name を除き、*format-specification* に–L、–R、および–C を使用して、左揃え、右揃え、中央揃えで出力を配置できます。たとえば、第 2 引数 *format-specification* の値には'upcase.-c'を使用できます。

オプション引数

w
出力形式に適用する幅を指定する数値定数、変数または式です。

操作 ここで指定した幅は、出力形式での幅の指定より優先されます。

詳細

まだ長さが割り当てられていない変数に PUTC 関数から値が返される場合、その変数の長さは最初の引数の長さによって決定されます。

比較

PUTN 関数は、実行時に数値の出力形式を指定できるようにします。

PUT 関数は実行時ではなくコンパイル時に出力形式を指定できるため、PUTC よりも高速です。

PUT 関数は、DS2 文字セマンティクスを使用して出力形式幅を適用します。つまり、PUT 関数は *w* を、フォーマットされた文字列に含める文字数として解釈します。PUTC 関数は、SAS バイトセマンティクスを使用します。

注: マルチバイト文字セットで使用すると、PUTC は値を切り捨てることができます。

例

例 1: PUTC 関数の操作

PROC FORMAT ステップでは、このステップで作成する他の 3 つの出力形式のいずれかの名前で変数値 1、2 および 3 をフォーマットする、出力形式 TYPEFMT.を作成します。これら 3 つの出力形式では、質問の種類に応じて、異なる単語として "positive"、"negative"および"neutral"の応答を出力します。PROC FORMAT で出力形式を作成した後に、DATA ステップで質問と応答の種類を識別する番号で構成された生データを使用して、SAS データセットを作成します。レコードの読み取り後、DATA ステップで TYPE の値を使用して、変数 RESPFMT を作成します。この変数は、現在の質問の種類に適切な出力形式の値を含みます。また、応答に適切な単語の値が含まれる別の変数 WORD も DATA ステップで作成します。PUTC 関数は、質問の種類と適切な出力形式に基づいて WORD の値を割り当てます。

```
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;

proc ds2;
data input_data (overwrite=yes);
dcl double type;
dcl char(8) response;
method init();
 do type = 1 to 3;
   do response = 'positive', 'negative', 'neutral';
    output;    end;
   end;
end;
enddata;
run;

data answers (overwrite=yes);
  method run();
   set input_data;
   respfmt=put(type, typefmt.);
   word=putc(response, respfmt);
  end;
enddata;
run;
quit;

proc print data=answers;
  title 'Using the Third Argument as a Number or Cycle';
run;
```

アウトプット 7.20　PUTC 関数使用時の出力

Using the Third Argument as a Number or Cycle

Obs	type	response	respfmt	word
1	1	positive	$groupx	agree
2	1	negative	$groupx	disagree
3	1	neutral	$groupx	notsure
4	2	positive	$groupy	accept
5	2	negative	$groupy	reject
6	2	neutral	$groupy	possible
7	3	positive	$groupz	pass
8	3	negative	$groupz	fail
9	3	neutral	$groupz	retest

開始オブザベーションの変数 WORD の値は agree です。最終オブザベーションの変数 WORD の値は retest です。

例 2: 文字値の配置

この例では、A の値を配置する、出力形式と配置文字の使用方法を示します。

```
proc ds2;
data _null_;
  dcl char(20) a y;
  method init();
  a='experiment';
  y=putc(a,'upcase.-r',20);
  put '*' y $char20. '*';
  put '*' a $upcase20. '*';
  end;
 enddata;
run;
quit;
```

```
*       EXPERIMENT*
*EXPERIMENT       *
```

関連項目

関数:

- “INPUTC 関数” (712 ページ)
- “INPUTN 関数” (714 ページ)
- “PUT 関数” (1038 ページ)
- “PUTN 関数” (1045 ページ)

PUTN 関数

実行時に数値の出力形式を指定できるようにします。

カテゴリ:	特殊
返されるデータの種類:	CHAR、NCHAR、NVARCHAR、VARCHAR

構文

PUTN(*value*, *format-specification* [, *w* [, *d*]])

必須引数

value
フォーマットされる数値を指定します。

データの種類 BIGINT、DECIMAL、DOUBLE、FLOAT、SMALLINT、TINYINT

format-specification
value に適用する数値出力形式です。

有効な出力形式の指定方法は次のようになります。

- *format-name*
- *format-name.*(出力形式名はピリオドで終わります)
- *format-namew.*
- *format-namew.d*

format-name を除き、*format-specification* に–L、–R、および–C を使用して、左揃え、右揃え、中央揃えで出力を配置できます。たとえば、第 2 引数 *format-specification* の値には'weekdate.-c'を使用できます。

オプション引数

w
出力形式に適用する幅を指定する数値定数、変数または式です。

操作 ここで指定した幅は、出力形式での幅の指定より優先されます。

d
使用する小数点以下の桁数を指定する数値定数、変数または式です。

操作 ここで指定した桁数は、出力形式での小数点以下の桁数の指定より優先されます。

詳細

まだ長さが割り当てられていない変数に PUTN 関数から値が返される場合、変数にはデフォルトで長さ 200 バイトが割り当てられます。

比較

PUTC 関数は、実行時に文字の出力形式を指定できるようにします。

PUT 関数は実行時ではなくコンパイル時に出力形式を適用するため、PUTN よりも高速です。

PUT 関数は、DS2 文字セマンティクスを使用して出力形式幅を適用します。つまり、PUT 関数は *w* を、フォーマットされた文字列に含める文字数として解釈します。PUTN 関数は、SAS バイトセマンティクスを使用します。

注: マルチバイト文字セットで使用すると、PUTN は値を切り捨てることができます。

例

例 1: PUTN 関数での日付と出力形式の操作

PROC FORMAT ステップでは、SAS 日付出力形式の名前で変数値 1 および 2 をフォーマットする、出力形式 WRITFMT.を作成します。DATA ステップで、番号とキーで構成される生データを使用して SAS データセットを作成します。レコードの読み取り後、DATA ステップで KEY の値を使用して、適切な日付出力形式の値が含まれる変数 DATEFMT を作成します。また、フォーマットされた日付の値が含まれる新しい変数 DATE も DATA ステップで作成します。PUTN は、NUMBER の値と適切な出力形式の値に基づいて DATE の値を割り当てます。

```
proc format;
  value writfmt 1='date9.'
                2='mmddyy10.';
run;

proc ds2;
data input_data (overwrite=yes);
dcl double number key;
dcl char(15) datefmt newdate;
method init();
number = 15756;
key = 1;
output;
number = 14552;
key = 2;
output;
end;
enddata;
run;


data dates (overwrite=yes);
method run();
set input_data;
datefmt=put(key, writfmt.);
newdate=putn(number, datefmt);
end;
enddata;
run;
quit;

proc print data=dates;
  title 'Working with Dates and Formats';
run;
```

アウトプット 7.21　PUTN 関数使用時の出力

Working with Dates and Formats

Obs	number	key	datefmt	newdate
1	15756	1	date9.	20FEB2003
2	14552	2	mmddyy10.	11/04/1999

例 2: PUTN 関数による出力の配置

この例では、値を左揃えにする、出力形式と配置文字の使用方法を示します。

```
proc ds2;
data _null_;
  dcl char(30) y;
  method init();
  y=putn(today(),'weekdate.-l',30);
  put '*' y $char30. '*';
  end;
 enddata;
run;
```

例のコード 7.1　配置結果

```
*Monday, November 19, 2012    *
```

関連項目

関数:

- “INPUTC 関数” (712 ページ)
- “INPUTN 関数” (714 ページ)
- “PUT 関数” (1038 ページ)
- “PUTC 関数” (1042 ページ)

PVP 関数

満期時の元本払い戻しで、定期的なキャッシュフローストリーム(債権など)の現在価値を返します。

カテゴリ:	財務
返されるデータの種類:	DOUBLE

構文

PVP(*A*, *c*, *n*, *K*, k_0, *y*)

必須引数

A

額面価格を指定します。

範囲 $A > 0$

A > 0

データの種類 DOUBLE

c

1 年当たりの名目クーポン率を分数で指定します。

範囲 $0 \le c < 1$

0 ≤ *c* < 1

データの種類 DOUBLE

n

1 年当たりのクーポン数を指定します。

範囲 $n > 0$

n > 0

データの種類 DOUBLE

K

クーポンの残数を指定します。

範囲 $K > 0$

K > 0

データの種類 DOUBLE

k_0

現在の日付から最初のクーポン日までの時間を指定します。年数で表します。

範囲 $0 < k_0 \le \frac{1}{n}$

$0 < k_0 \le 1/n$

データの種類 DOUBLE

y

1 年当たりの名目最終利回りを指定します。分数で表します。

範囲	$y > 0$
	$y > 0$
データの種類	DOUBLE

詳細

PVP 関数は次の関係に基づきます。

$$P = \sum_{k=1}^{K} \frac{c(k)}{\left(1 + \frac{y}{n}\right)^{t_k}}$$

前述の式には次の関係が適用されます。

- $t_k = nk_0 + k - 1$
- $c(k) = \frac{c}{n}A \quad for\ k = 1, \ldots, K-1$
- $c(K) = \left(1 + \frac{c}{n}\right)A$

例

次のプログラムは、PVP 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double p;
  method run();
    p=pvp(1000,.01,4,14,.33/2,.10);
    put p;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
743.167613519067
```

QTR 関数

SAS 日付値から年の四半期を返します。

カテゴリ: 日付と時間

返されるデータの種類: DOUBLE

構文

QTR(*date*)

必須引数

date
SAS 日付値を表す有効な式を指定します。

データの種類 DOUBLE

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

詳細

QTR 関数は、SAS 日付値から、日付値が含まれる四半期を示す値 1、2、3、4 を返します。

DS2 での日付値と時間値の処理方法の詳細については、"DS2 式" (*SAS DS2 プログラマガイド*)を参照してください。

例

次のプログラムは、QTR 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double a b c;
  method run();
    a=17180;
    b=put(a, date7.);
```

```
    c=qtr(a);
    put c;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
1
```

関連項目

関数:

- “YYQ 関数” (1270 ページ)

QUANTILE 関数

左側確率(CDF)を指定するときに分布から分位点を返します。

カテゴリ:	分位点
返されるデータの種類:	DOUBLE
参照項目:	“CDF 関数” (359 ページ)

構文

QUANTILE(*'distribution'*, *probability*, *parameter-1, ..., parameter-k*)

必須引数

distribution
分布を特定する文字定数、変数または式です。

注: 各 QUANTILE 分布関数の引数は、対応する CDF 分布関数の引数と同じです。

有効な分布は次のとおりです。

分布	引数
Bernoulli	'BERNOULLI'

分布	引数
ベータ	'BETA'
二項	'BINOMIAL'
Cauchy	'CAUCHY'
カイ 2 乗	'CHISQUARE'
Conway-Maxwell-Poisson	'CONMAXPOI'
指数	'EXPONENTIAL'
F	'F'
ガンマ	'GAMMA'
一般化 Poisson	'GENPOISSON'
幾何	'GEOMETRIC'
超幾何	'HYPERGEOMETRIC'
Laplace	'LAPLACE'
ロジスティック	'LOGISTIC'
対数正規	'LOGNORMAL'
負の二項	'NEGBINOMIAL'
正規	'NORMAL'|'GAUSS'
正規混合	'NORMALMIX'
パレート	'PARETO'
Poisson	'POISSON'
T	'T'
Tweedie	'TWEEDIE'
一様	'UNIFORM'
Wald (逆ガウス)	'WALD'|'IGAUSS'

分布	引数
Weibull	'WEIBULL'

注: T、F および NORMALMIX を除き、最初の 4 文字で分布を最小限に識別できます。

probability
確率変数の値を指定する、数値定数、変数または式です。

parameter-1, ..., parameter-k
特定の分布に適した任意の *shape*、*location* または *scale* パラメーターです。

詳細

QUANTILE 関数は、指定された確率値に基づいて、指定された連続分布または離散分布から分位点を計算します。詳細については、前述の表に記載されている個々の分布を参照してください。

QUANTILE 関数の Conway-Maxwell-Poisson 分布は、CDF 値が *p* 以下の最大の整数であるカウント値 *y* を返します。QUANTILE 関数の Conway-Maxwell-Poisson 分布の構文は、次の形式です。

QUANTILE('CONMAXPOI',*p*,*λ*,*v*)

p
0 と 1 の間の実数(両端の値を含める)。

λ
Poisson 分布の場合と同様で、平均に似ています。

v
ばらつきのパラメーターです。

詳細については、PDF 関数の“Conway-Maxwell-Poisson” (930 ページ)を参照してください。

例

次のプログラムは、QUANTILE 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double a b c d e f g h i j k l m n o p q r x t;
  method init();
```

```
    a=quantile('BERN', .75, .25);
    b=quantile('BETA', 0.1,3,4);
    c=quantile('BINOM',.4, .5, 10);
    d=quantile('CAUCHY', .85);
    e=quantile('CHISQ', .6,11);
    f=quantile('CONMAXPOI',0.2,2.3,.4);
    g=quantile('EXPO', .6);
    h=quantile('F',.8,2,3);
    i=quantile('GAMMA', .4,3);
    j=quantile('GENPOI', .9, 1, .7);
    k=quantile('HYPER', .5, 200, 50, 10);
    l=quantile('LAPLACE', .8);
    m=quantile('LOGISTIC', .7);
    n=quantile('LOGNORMAL', .5);
    o=quantile('NEGB', .5, .5, 2);
    p=quantile('NORMAL', .975);
    q=quantile('NORMALMIX',0.5, 1, 0.2, 1.1,0.1);
    r=quantile('PARETO', .01,1);
    s=quantile('POISSON', .9, 1);
    t=quantile('T', .8, 5);
    u=quantile('TWEEDIE', .8, 5);
    v=quantile('UNIFORM', 0.25);
    w=quantile('WALD', .6, 2);
    x=quantile('WEIBULL', .6, 2);
    put a=;
    put b=;
    put c=;
    put c=;
    put e=;
    put f=;
    put g=;
    put h=;
    put i=;
    put j=;
    put k=;
    put l=;
    put m=;
    put n=;
    put o=;
    put p=;
    put q=;
    put r=;
    put s=;
    put t=;
    put u=;
    put v=;
    put w=;
    put x=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
a=0
b=0.2009088788569
c=5
d=1.96261050550515
e=11.5298338409688
f=5
g=0.91629073187415
h=2.88602660731929
i=2.28507690400338
j=9
k=2
l=0.91629073187415
m=0.8472978603872
n=1
o=1
p=1.95996398454005
q=1.1
r=1.01010101010101
s=2
t=0.91954378024082
u=1.26111981969517
v=0.25
w=0.95262099270959
x=0.95723076208099
```

関連項目

関数:

- “CDF 関数” (359 ページ)
- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “PDF 関数” (918 ページ)
- “SDF 関数” (1120 ページ)
- “SQUANTILE 関数” (1149 ページ)

QUOTE 関数

文字値に二重引用符を付加します。

カテゴリ:	文字
返されるデータの種類:	CHAR、NCHAR、NVARCHAR、VARCHAR

構文

QUOTE(*expression*)

必須引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

詳細

QUOTE 関数は、文字値にデフォルト文字である二重引用符を付加します。引数内から二重引用符が検出されると、さらに二重にして出力されます。

受け取る変数には、引数(末尾の空白も含む)、先頭および末尾の引用符、二重にされた埋め込み引用符を含む十分な長さが必要です。たとえば、引数が ABC でその末尾に 3 つの空白が続く場合、"ABC###"を保持するために受け取る変数の長さは 8 文字以上であることが必要です。(文字#は空白を表します)。受け取るフィールドが長すぎると、QUOTE 関数は空白の文字列を返し、無効な引数というメモを SAS ログに書き込みます。

二重引用符で囲まれた文字列は DS2 識別子であり、文字定数ではありません。二重引用符は識別子の一部になります。引用符で囲まれた識別子を使用して、出力テーブルに列名を作成することはできません。

例

次のプログラムは、QUOTE 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl varchar(6) a b c d;
  dcl varchar(30) e f;
  method run();
    a='A"B';
    b=quote(a);
    put b;
    c='A''B';
    d=quote(c);
    put d;
    e='Paul''s Catering Service    ';
```

```
      f=quote(trim(e));
      put f;
    end;
  enddata;
  run;
  quit;
```

SAS は次の出力をログに書き込みます。

```
"A""B"
"A'B"
"Paul's Catering Service"
```

RANBIN 関数

二項分布からランダム変量を返します。

注: この関数はサポートされなくなりました。かわりに RAND('BINOMIAL')関数 (1058 ページ)を使用してください。

RANCAU 関数

Cauchy 分布からランダム変量を返します。

注: この関数はサポートされなくなりました。かわりに RAND('CAUCHY')関数 (1058 ページ)を使用してください。

RAND 関数

指定する分布から乱数を生成します。

カテゴリ: 乱数

返されるデータの種類: DOUBLE

注: 2025.12 以降、この関数では、引数が DOUBLE の場合、すべての引数が DOUBLE に変換され、結果は DOUBLE になります。PROC DS2 ステートメントで BIGINTPROCESSING=YES が設定され、いずれかの引数が BIGINT の場合、すべての引数が BIGINT に変換され、結果は BIGINT になります。

構文

RAND('*distribution*', *parameter-1, ...parameter-k*)

必須引数

distribution
分布を特定する文字定数、変数または式です。有効な分布は、次のとおりです。

分布	関数呼び出し	パラメーター値
Bernoulli	RAND('BERNoulli', prob)	ここでは、(0 <= prob <= 1)
ベータ	RAND('BETA', alpha, beta)	ここでは、(0.01 <= alpha <= 1.5e6)および(0.20 <= beta <= 1.5e6)
二項	RAND('BINOmial', prob, trials)	ここでは、(0 <= prob <= 1)および(0 <= trials)
Cauchy	RAND('CAUChy')	
カイ 2 乗	RAND('CHISquare', df)	ここでは、0.03 <= df <= 4e9
Erlang	RAND('ERLAng', alpha)	ここでは、1 <= alpha <= 2e9
指数	RAND('EXPOnential', scale)	ここでは、0 <= scale <= 1e300
極値	RAND('EXTReme')	デフォルト m = 0、scale = 1、shape = 0
	RAND('EXTReme', mu)	ここでは、–1e300 <= mu <= 1e300
	RAND('EXTReme', mu, scale)	ここでは、scale > 0
	RAND('EXTReme', mu, scale, shape)	ここでは、(scale > 0)および(–0.5 <= shape <= 5)
F	RAND('F', numdf, dendf)	ここでは、(0.025 <= numdf <= 1e7)および(0.1 <= dendf <= 2e9)
ガンマ	RAND('GAMMa', alpha)	ここでは、0.015 <= alpha <= 2e9、デフォルト scale = 1
	RAND('GAMMa', alpha, scale)	ここでは、(1e–80 <= scale <= 1e295)または(scale = 0)
幾何	RAND('GEOMetric', prob)	ここでは、0 < prob <= 1
Gompertz	RAND('GOMPertz')	デフォルト shape = 1、scale = 1
	RAND('GOMPertz' shape)	ここでは、shape >= 1e–300
	RAND('GOMPertz' shape , scale)	ここでは、(shape >= 1e–300)および(scale >= 0)
Gumbel	RAND('GUMBel')	デフォルト mu = 0、scale = 1

分布	関数呼び出し	パラメーター値
	RAND('GUMBel', mu)	ここでは、–1e300 <= mu <= 1e300
	RAND('GUMBel', mu, scale)	ここでは、(-1e300 <= mu <= 1e300)および (scale >= 0)
超幾何	RAND('HYPErgeometric', pop_size, cat_size, samp_size)	ここでは、1 <= pop_size <= 9e15 および 0 <= cat_size <= pop_size および 1 <= samp_size <= pop_size
一様整数	RAND('INTEger', int)	ここでは、–2147483648 <= int <= 2147483647
Laplace	RAND('LAPLace')	デフォルト mu = 0、scale = 1
	RAND('LAPLace', mu)	ここでは、–1e300 <= mu <= 1e300
	RAND('LAPLace', mu, scale)	ここでは、(–1e300 <= mu <= 1e300)および (scale >= 0)
ロジスティック	RAND('LOGIstic')	デフォルト mu = 0、scale = 1
	RAND('LOGIstic', mu)	ここでは、–1e300 <= mu <= 1e300
	RAND('LOGIstic', mu, scale)	ここでは、(–1e300 <= mu <= 1e300)および (scale >= 0)
対数正規	RAND('LOGNormal')	デフォルト mu = 0、sigma = 1
	RAND('LOGNormal', mu)	ここでは、ABS(mu) <= 702
	RAND('LOGNormal', mu, sigma)	ここでは、(ABS(mu) + 7*sigma <= 709)および ((sigma >= max(1, ABS(mu)) * 1e–13)または (sigma = 0))
負の二項	RAND('NEGBinomial', prob, n)	ここでは、(0 < prob <= 1)および(n >= 1)
正規またはガウス	RAND('NORMal')	デフォルト mu = 0、sigma = 1
	RAND('NORMal', mu)	ここでは、ABS(mu) <= 1e14
	RAND('NORMal', mu, sigma)	ここでは、(ABS(mu) <= 1e14*sigma)または (sigma = 0)
パレート	RAND('PAREto')	デフォルト shape = 1、scale = 1
	RAND('PAREto', shape)	ここでは、shape .04 <= xi <= 1e10

分布	関数呼び出し	パラメーター値
	RAND('PAREto', shape, scale)	ここでは、(shape .04 <= xi <= 1e10)および(0 <= scale <=1e100)
Poisson	RAND('POISson', mean)	ここでは、0 <= mean <= 1e300
シフトされた Gompertz	RAND('SHGOmpertz')	デフォルト shape = 1、inverse_scale = 1
	RAND('SHGOmpertz', shape)	ここでは、shape >= 1e–300
	RAND('SHGOmpertz', shape, inverse_scale)	ここでは、(shape >= 1e–300)および (inverse_scale >= 1e-300)
T	RAND('T', df)	ここでは、df >= 0.05
テーブル	RAND('TABLe', p_1, ..., p_n)	ここでは、(0 <= pi <= 1)および(p_1 + ...+ p_n <= 1)
三角	RAND('TRIAngular', height)	ここでは、0 <= height <= 1
一様	RAND('UNIForm')	デフォルト a = 1、b = 0
	RAND('UNIForm', a)	(min(a,0) max(a,0))で一様
	RAND('UNIForm', a, b)	(min(a,b) max(a,b))で一様
Wald (逆ガウス)	RAND('WALD', shape)	ここでは、1e–18 <= shape <= 1e17、デフォルト mu = 1
	RAND('WALD', shape, mu)	ここでは、(shape >= 1e–18)および(mu >= 1e–10)および(1e–21 <= shape/mu <= 1e17)
Weibull	RAND('WEIBull', shape)	ここでは、0.02 <= shape <= 1e13、デフォルト scale = 1
	RAND('WEIBull', shape, scale)	ここでは、(0.02 <= shape <= 1e13)および(1e–100 <= scale <= 1e20)

データの種類　CHAR、NCHAR、NVARCHAR、VARCHAR

注　T および F 以外は、最初の 4 文字によってどのような分布でも最小限に識別できます。

parameter-1, ...parameter-k
特定の分布に適した*形状*、*位置*または*尺度*パラメーターの値を指定する数値定数、変数または式です(省略可能)。

参照項目: “詳細” (1062 ページ)

詳細

乱数の生成

RAND 関数は、さまざまな連続分布および離散分布から乱数を生成します。可能な場合、最も単純な形式の分布が使用されます。

RAND 関数は、Matsumoto and Nishimura (1998)によって開発された Mersenne-Twister 乱数ジェネレーター(RNG)を応用しています。その他の乱数ジェネレーターは、STREAMINIT 関数で指定できます。

RAND 関数は単一のシードで開始します。ただし、プロセスの状態を単一のシードで捕捉することはできません。ジェネレーターを停止し、停止時点から再開することはできません。

デフォルトの 32 ビット Mersenne-Twister の場合、初期シードが 8192 で割り切れる場合、RAND 関数は 2002 年の初期化アルゴリズム(Matsumoto and Nishimura, 2002)を使用します。それ以外の場合、RAND は以前のリリースとの互換性を確保するため 1998 年の初期化アルゴリズムを使用します。

乱数ストリームの再現

乱数の再現可能なストリームを作成する場合は、RAND 関数を呼び出す前に STREAMINIT 関数を使用して、乱数生成のシード値を指定します。STREAMINIT 関数は、データプログラムごとに 1 回使用します。詳細については、“STREAMINIT 関数” (1160 ページ)の例を参照してください。

次の FCMP 関数は、DS2 プログラムから呼び出された場合、DS2 プログラムで設定されたシード値を認識しません。

- COMPCOST
- COMPGED
- RAND
- STREAMINIT

DATA ステップでは、RAND からのすべての乱数はデフォルトで同じストリームから取得されますが、CALL STREAM ルーチンを使用して複数の独立したストリームを作成できます。

PROC DS2 では、FCMP パッケージ内で呼び出される RAND の乱数は、別のストリームから取得されます。再現可能な乱数を取得するには、RAND を呼び出す各 FCMP 関数内で CALL STREAMINIT ルーチンを使用します。独立したストリームを取得するには、RAND を呼び出す各 FCMP 関数内で CALL STREAM ルーチンも使用します。そうすると、DS2 からの結果は DATA ステップと一致します。例については、“FCMP パッケージを使用する際の考慮事項と制限事項” (*SAS DS2 プログラマガイド*)を参照してください。

重複値

RAND 関数で使用される RNG アルゴリズムには非常に長い周期が使用されますが、大きなランダムサンプルに重複する値がないことを暗示するものではありません。デフォルトの 32 ビット Mersenne-Twister アルゴリズムでは、RAND 関数は最大 2^{32} 個の異なる値を返します。サイズが 10^5 のランダム一様サンプルでは、少なくとも 1 つの重複を引き当てる確率が 50%を超えます。サイズ M のランダム一様サンプルの期待重複数は、M が 2^{32} よりかなり小さいとき、約 $M^2/2^{33}$ になります。たとえば、サイズ $M=10^6$ のランダム一様サンプルでは、約 115 個の重複が期待されます。これらの結果は、確率論における有名な“誕生日の一致問題”で説明できます。

64 ビット RNG の場合、RAND('UNIFORM')は少なくとも $2^{53}-1$ 個の固有値を返すことができます。10^8 のサンプルサイズでは、少なくとも 1 つの重複を引き当てる確率が 50%を超えます。10^9 のサンプルサイズでは、重複数の期待値は 55.55 程度です。

極端なパラメーター値

疑似乱数は、一様なランダム変量を変換する数値アルゴリズムを使用して生成されます。一部の分布では、パラメーター値が非常に小さいか非常に大きいと、理論上の分布に十分に従わない疑似ランダム変量が発生する可能性があります。他の分布の場合、極端なパラメーター値は、計算中に数値のアンダーフローまたはオーバーフローを引き起こす可能性があります。RAND 関数は、これらの問題につながる可能性のあるパラメーター空間の領域を識別しようとします。

RAND 関数は、極端なパラメーターが無効な疑似ランダム変量につながる可能性があると判断した場合、欠損値を返し、警告メッセージを SAS ログに書き込みます。たとえば、カイ 2 乗分布のパラメーターの値が小さい場合(d=0.01 など)、次の警告メッセージが生成され、RAND 関数が、欠損値を返す可能性があります。

```
WARNING: In the function call, RAND('CHISQUARE', 0.01), there is a risk
         of underflow that would cause the distribution of the random
         numbers to be wrong. It is recommended that you use
         RAND('CHISQUARE', df), where 0.03 <= df <= 4e9.
```

RAND 関数は、(分布が不十分である可能性がありますが)乱数を生成します。警告と原因を抑制するには、最初の引数で指定された分布の名前の末尾に疑問符を追加します。RAND('CHISQUARE?',.01)がその例です。

Bernoulli 分布

x=**RAND**('BERNOULLI', *p*)

引数

x

次の確率密度関数を使用した Bernoulli 分布からのオブザベーションです。

$$f(x)=\begin{cases}1 & p=0, x=0\\ p^x(1-p)^{1-x} & 0<p<1, x=0,1\\ 1 & p=1, x=1\end{cases}$$

範囲 *x*=0, 1

p
数値の成功率です。

範囲　$0 \le p \le 1$

ベータ分布

x = **RAND**('BETA', *a*, *b*)

引数

x
次の確率密度関数を使用したベータ分布からのオブザベーションです。

$$f(x) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} x^{a-1}(1-x)^{b-1}$$

範囲　$0 < x < 1$

a
数値の形状パラメーターです。

範囲　$a > 0$

b
数値の形状パラメーターです。

範囲　$b > 0$

二項分布

x = **RAND**('BINOMIAL', *p*, *n*)

引数

x
次の確率密度関数を使用した二項分布からの整数であるオブザベーションです。

$$f(x) = \begin{cases} 1 & p = 0, x = 0 \\ \binom{n}{x} p^x (1-p)^{n-x} & 0 < p < 1, x = 0, \ldots, n \\ 1 & p = 1, x = n \end{cases}$$

範囲　$x = 0, 1, \ldots, n$

p
数値の成功率です。

範囲　$0 \le p \le 1$

n
独立 Bernoulli 試行数を数えるパラメーターです。この引数は整数にする必要があります。

範囲　$n = 1, 2, \ldots$

Cauchy 分布

x = **RAND**('CAUCHY')

引数

x

次の確率密度関数を使用した Cauchy 分布からのオブザベーションです。

$$f(x) = \frac{1}{\pi\left(1 + x^2\right)}$$

範囲 $-\infty < x < \infty$

カイ 2 乗分布

x = **RAND**('CHISQUARE', *df*)

引数

x

次の確率密度関数を使用したカイ 2 乗分布からのオブザベーションです。

$$f(x) = \frac{2^{-df/2}}{\Gamma\left(\frac{df}{2}\right)} x^{df/2-1} e^{-x/2}$$

範囲 $x > 0$

df

数値の自由度パラメーターです。

範囲 $df > 0$

Erlang 分布

x = **RAND**('ERLANG', *a*)

引数

x

次の確率密度関数を使用した Erlang 分布からのオブザベーションです。

$$f(x) = \frac{1}{\Gamma(a)} x^{a-1} e^{-x}$$

範囲 $x > 0$

a

数値の形状パラメーターです。この引数は整数にする必要があります。

範囲 $a = 1, 2, \ldots$

指数分布

x = **RAND**('EXPONENTIAL' ,[σ])

引数

x
次の確率密度関数を使用した指数分布からのオブザベーションです。

$$f(x) = e^{-x}$$

範囲 *x* > 0

σ
尺度パラメーターです。

デフォルト 1

範囲 σ > 0

極値分布

x=**RAND**('EXTRVALUE', [*μ*, *σ*, *ξ*])

引数

x
次の累積確率分布を持つ極値分布からのオブザベーションです。

$$F(x) = \exp(t(x))$$
$$where: t(x) = (1 + \xi(x - \mu)/\sigma)^{-1/\xi} \; for \; \xi \neq 0 \; and \; t(x) = \exp(-(x - \mu)/\sigma) \; for \; \xi = 0$$

ξ=0 の場合、分布はタイプ 1 分布であり、ガンベル型分布と呼ばれることもあります。X のランダムな値は、間隔(–∞, ∞)にあります。ξ=0 の場合、RAND 関数で GUMBEL オプションを使用する方が効率的です。ξ>0 の場合、分布はタイプ 2 分布であり、フレシェ型分布と呼ばれることもあります。x のランダムな値は、間隔(μ– σ/ξ, ∞)にあります。ξ<0 の場合、分布はタイプ 3 分布であり、ワイブル型分布と呼ばれることもあります。X のランダムな値は、間隔(–∞, μ– σ/ξ)にあります。

μ
数値の位置パラメーターです。

デフォルト 0

σ
数値の尺度パラメーターです。

デフォルト 0

範囲 σ>0

ξ
数値の形状パラメーターです。

デフォルト 1

F 分布

x = **RAND**('F', *n*, *d*)

引数

x

次の確率密度関数を使用した F 分布からのオブザベーションです。

$$f(x) = \frac{\Gamma\left(\frac{n+d}{2}\right)}{\Gamma\left(\frac{n}{2}\right)\Gamma\left(\frac{d}{2}\right)} \frac{n^{n/2} d^{d/2} x^{n/2-1}}{(d+nx)^{(n+d)/2}}$$

範囲 $x > 0$

n

数値の分子の自由度パラメーターです。

範囲 $n > 0$

d

数値の分母の自由度パラメーターです。

範囲 $d > 0$

ガンマ分布

x = **RAND**('GAMMA', *α* [,λ])

引数

x

次の確率密度関数を使用したガンマ分布からのオブザベーションです。

$$f(x) = \frac{x^{a-1}}{\lambda a \Gamma(a)} \exp(-x/\lambda)$$

範囲 $x > 0$

α

形状パラメーターを指定する数値定数、変数または式です。

範囲 $\alpha > 0$

λ

尺度パラメーターを指定する数値定数、変数または式です。

デフォルト 1

範囲 λ>1

幾何分布

x = **RAND**('GEOMETRIC', *p*)

引数

x
1 回の成功を得るために必要な試行回数を示すカウントです。この引数は整数にする必要があります。*X* は、次の確率密度関数を使用した幾何分布からのオブザベーションです。

$$f(x) = \begin{cases} (1-p)^{x-1}p & 0 < p < 1, x = 1, 2, \ldots \\ 1 & p = 1, x = 1 \end{cases}$$

範囲　*x* = 1, 2, ...

p
数値の成功率です。

範囲　0 < *p* ≤ 1

ゴンペルツ分布

x=**RAND**('GOMPERTZ',[λ,σ])

引数

x
ゴンペルツ分布からのオブザベーションであり、次の累積確率分布を持ちます。

$$F(x) = 1 - \exp(-\lambda \exp(1/\sigma) - 1)$$

x のランダムな値は、間隔(00, ∞)にあります。

λ
形状パラメーターを指定する数値定数、変数または式です。

デフォルト　1

σ
尺度パラメーターを指定する数値定数、変数または式です。

デフォルト　1

範囲　σ>0

ガンベル分布

x=RAND('GUMBEL',[μ, σ])

x
ガンベル分布からのオブザベーションであり、次の累積確率分布を持ちます。

$$F(x) = \exp(-\exp(-(x-\mu)/\sigma))$$

x のランダムな値は、間隔(0, ∞)にあります。ガンベル分布は極値分布です。分布は右に偏っています。

μ
数値の位置パラメーターを指定する数値定数、変数または式です。

デフォルト 0

σ

数値の尺度パラメーターを指定する数値定数、変数または式です。

デフォルト 1

範囲 σ>0

超幾何分布

x = **RAND**('HYPER', *N*, *R*, *n*)

引数

x

次の確率密度関数を使用した超幾何分布からのオブザベーションです。この引数は整数にする必要があります。

$$f(x) = \frac{\binom{R}{x}\binom{N-R}{n-x}}{\binom{N}{n}}$$

範囲 *x* = max(0, (*n* – (*N* – *R*))), ..., min(*n*, *R*)

N

母集団サイズパラメーターです。この引数は整数にする必要があります。

範囲 *N* = 1, 2, ...

R

対象カテゴリの項目数です。この引数は整数にする必要があります。

範囲 *R* = 0, 1, ..., *N*

n

サンプルサイズパラメーターです。この引数は整数にする必要があります。

範囲 *n* = 1, 2, ..., *N*

超幾何分布は、*N* 個のボールが壺に入っており、そのうちの *R* 個が赤のときに、壺から *n* 個のボールを取り出す実験を数学的に形式化したものです。超幾何分布は、*n* 個のサンプル中の赤いボールの数の分布です。

整数分布

x=**RAND**('INTEGER',*a*,[*b*])

引数

x

整数の有限集合上の離散一様分布からのランダム値です。1 つの整数パラメーター *a* を指定すると、*x* が集合{1,2,..., a–1, a}からランダムに均一に描画されま

す。2 つの整数パラメーター *a* と *b* を a≤b で指定すると、*x* は集合{a,a+1,..., b-1, b}からランダムに均一に描画されます。

a
整数パラメーターです。数値パラメーターを 1 つだけ指定する場合、*a* はランダム値の上限です。2 つのパラメーターを指定する場合、 *a* は下限です。

b
ランダム値の上限を指定する整数パラメーターです。

Laplace 分布

x=**RAND**('LAPLACE',[*θ*,*λ*])

引数

x
Laplace 分布からのオブザベーションであり、次の確率密度関数を持ちます。

$$f\left(x\right)=\frac{1}{2\lambda}\exp\left(-\left|x-\theta\right|/\lambda\right)$$

θ
任意の位置パラメーターです。

デフォルト 0

λ
任意の尺度パラメーターです。

デフォルト 0

範囲 λ>0

ロジスティック分布

x=**RAND**('LOGISTIC',[*θ*,*λ*])

引数

x
ロジスティック分布からのオブザベーションであり、次の確率密度関数を持ちます。

$$f\left(x\right)=\frac{\exp(-(x-\theta)/\lambda)}{\lambda(1+\exp(-(x-\theta)/\lambda))^2}$$

θ
任意の尺度パラメーターです。

デフォルト 0

λ
任意の尺度パラメーターです。

デフォルト 0

範囲 λ>0

対数正規分布

x = **RAND**('LOGNORMAL',[*θ*,*λ*])

引数

x

次の確率密度関数を使用した対数正規分布からのオブザベーションです。

$$f(x) = \frac{1}{x\lambda\sqrt{2\,\pi}}\exp\left(\frac{-(\ln(x)-\theta)2}{2\lambda 2}\right)$$

確率変数 *x* が対数正規分布している場合、log(*x*)は平均 θ と標準偏差 λ で正規分布しています。

範囲 *x* > 0

θ

対数尺度パラメーターを指定する数値定数、変数または式です。

デフォルト 0

λ

形状パラメーターを指定する数値定数、変数または式です。

デフォルト 1

範囲 λ > 0

負の二項分布

x = **RAND**('NEGBINOMIAL', *p*, *k*)

引数

x

次の確率密度関数を使用した負の二項分布からのオブザベーションです。この引数は整数にする必要があります。

$$f(x) = \begin{cases} \binom{x+k-1}{k-1}(1-p)^x p^k & 0 < p < 1, x = 0, 1, \ldots \\ 1 & p = 1, x = 0 \end{cases}$$

範囲 *x* = 0, 1, ...

k

成功回数を示す数値パラメーターです。整数および非整数の *k* 値を使用できます。

範囲 *k* = 1, 2, ...

p

数値の成功率です。

範囲 0 < *p* ≤ 1

負の二項分布は、同じ成功確率 *p* で行う逐次独立試行で *k* 回成功するまでの失敗回数の分布です。

正規分布

x=**RAND**('NORMAL' [, *θ*, *λ*])

引数

x

次の確率密度関数を使用した平均 θ および標準偏差 λ の正規分布のオブザベーションです。

$$f(x) = \frac{1}{\lambda\sqrt{2\,\pi}}\ exp\left(-\frac{(x-\theta)^2}{2\lambda^2}\right)$$

範囲 $-\infty < x < \infty$

θ

平均パラメーターです。

デフォルト 0

λ

標準偏差パラメーターです。

デフォルト 1

範囲 λ > 0

パレート分布

x=**RAND**('PARETO', *a*,[*k*])

引数

x

パレート分布からのオブザベーションであり、次の確率密度関数を持ちます。

$$f\left(x\right) = \frac{a}{k}\left(\frac{k}{x}\right)a + 1$$

a

形状パラメーターです。

範囲 *a*>0

k

任意の尺度パラメーターです。

デフォルト 1

範囲 *k*>0

Poisson 分布

x = **RAND**('POISSON', *m*)

引数

x

次の確率密度関数を使用した分布からのオブザベーションです。この引数は整数にする必要があります。

$$f(x) = \frac{m^x e^{-m}}{x!}$$

範囲 *x* = 0, 1, ...

m

数値の平均パラメーターです。

範囲 *m* > 0

シフトされたゴンペルツ分布

x=**RAND**('SHGOMPERTZ', [*η*,*τ*])

引数

x

シフトされたゴンペルツ分布からのオブザベーションであり、次の累積確率分布を持ちます。

$$F(x) = (1 - \exp(-\tau x))\exp(-\eta \exp(-\tau x))$$

x のランダムな値は、間隔(0, ∞)にあります。η→0 として、シフトされたゴンペルツ分布は、形状パラメーター 1/τ で指数分布に近づきます。

η

形状パラメーターです。

デフォルト 1

範囲 *η*>0

τ

逆の尺度パラメーターです。

デフォルト 1

範囲 *τ*>0

T 分布

x = **RAND**('T', *df*)

引数

x

次の確率密度関数を使用した T 分布からのオブザベーションです。

$$f(x) = \frac{\Gamma\left(\frac{df+1}{2}\right)}{\sqrt{df\,\pi}\,\Gamma\left(\frac{df}{2}\right)}\left(1+\frac{x^2}{df}\right)^{-\frac{df+1}{2}}$$

範囲　$-\infty < x < \infty$

df
数値の自由度パラメーターです。

範囲　$df > 0$

テーブル分布

x = **RAND**('TABLE', *p1*, *p2*, ...)

引数

x
次のいずれかの分布からのオブザベーションです。この引数は整数にする必要があります。

$\sum_{i=1}^{n} p_i < 1$の場合、*x* は次の確率密度関数からのオブザベーションです。

$f(i) = p_i, \quad i = 1, 2, \ldots, n$

and

$f(n+1) = 1 - \sum_{i=1}^{n} p_i$

$\sum_{i=1}^{n} p_i \geq 1$がインデックス *n* で成り立つならば、*x* は次の確率密度関数からのオブザベーションです。

$f(i) = p_i, \quad i = 1, 2, \ldots, n-1$

and

$f(n) = 1 - \sum_{i=1}^{n-1} p_i$

p1, p2, ...
数値の確率値です。

範囲　$0 \leq p1, p2, \ldots \leq 1$

制限事項　確率パラメーターの最大数は動作環境によって異なりますが、パラメーターの最大数は少なくとも 32,767 です。

テーブル分布は、指定した確率で値 1、2、...、*n* を取ります。

注: FORMAT ステートメントを使用することにより、セット{1, 2, ..., *n*}を *n* 個以下の要素セットにマップできます。

三角分布

x = **RAND**('TRIANGLE', *h*)

引数

x

次の確率密度関数を使用した三角分布からのオブザベーションです。

$$f(x) = \begin{cases} \frac{2x}{h} & 0 \le x \le h \\ \frac{2(1-x)}{1-h} & h < x \le 1 \end{cases}$$

この式では、0≤*h*≤1 になります。

範囲　0 ≤ *x* ≤ 1

注　分布は簡単に移動および調整できます。

h

三角形の頂点の水平位置です。

範囲　0≤*h*≤1

一様分布

x = **RAND**('UNIFORM', [a,b])

引数

x

間隔(a,b)の連続一様分布からのオブザベーションです。a< b の場合、(a,b)の確率密度関数は次のようになります。

$$f(x) = \frac{1}{|b-a|}$$

パラメーターを指定しない場合は、間隔(0,1)が使用されます。1 つのパラメーターを指定すると、間隔(0,c)が使用されます。ここで、c は指定されたパラメーターです。2 つのパラメーターを指定すると、a < x < b になります。ここで、a と b は指定されたパラメーターです。

範囲　0 < *x* < 1

RAND 関数は、Matsumoto and Nishimura (1998)によって開発された Mersenne-Twister 乱数ジェネレーター(RNG)を応用しています。その他の乱数ジェネレーターは、STREAMINIT 関数で指定できます。

RAND 関数は単一のシードで開始します。ただし、プロセスの状態を単一のシードで捕捉することはできません。ジェネレーターを停止し、停止時点から再開することはできません。

デフォルトの 32 ビット Mersenne-Twister の場合、初期シードが 8192 で割り切れる場合、RAND 関数は 2002 年の初期化アルゴリズム(Matsumoto and Nishimura, 2002)を使用します。それ以外の場合、RAND は以前のリリースとの互換性を確保するため 1998 年の初期化アルゴリズムを使用します。

Wald (逆ガウス)分布

x=**RAND**('WALD', *λ*,[*θ*])

x=**RAND**('IGAUSS', *λ*,[*θ*])

引数

x

Wald 分布からのオブザベーションであり、次の確率密度関数を持ちます。

$$f\left(x\right)=\left(\frac{\lambda}{2\pi x^3}\right)^{1/2}\exp\left(-\frac{\lambda(x-\theta)2}{2\theta^2 x}\right)$$

x のランダムな値は間隔(0,∞)にあります。SAS/STAT ソフトウェアの MCMC プロシジャを含む多くの参考資料では、逆ガウス分布の最初のパラメーターとして θ がリストされています。ただし、θ はオプションのパラメーターであるため、RAND、PDF、CDF、および QUANTILE 関数の最後のパラメーターです。

λ

形状パラメーターです。

範囲 *λ*>0

θ

平均パラメーターです。

デフォルト 1

範囲 *θ*>0

Weibull 分布

x = **RAND**('WEIBULL', *a*, *b*)

引数

x

次の確率密度関数を使用した Weibull 分布からのオブザベーションです。

$$f(x)=\frac{a}{b^a}x^{a-1}e^{-\left(\frac{x}{b}\right)^a}$$

範囲 $x \geq 0$

a

数値の形状パラメーターです。

範囲 $a > 0$

b

数値の尺度パラメーターです。

範囲 $b > 0$

例

次のプログラムは、RAND 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double a b c d e f g h i j k l m n o p q r s t u v;
  dcl double w x y z aa bb cc;
  method run();
    a=rand('BERN', .75);
    b=rand('BETA', 3, 1);
    c=rand('BINOM', 0.75, 10);
    d=rand('CAUCHY');
    e=rand('CHISQ', 22);
    f=rand('ERLANG', 7);
    g=rand('EXTRVAL', 0, 1);
    h=rand('EXPO');
    i=rand('F', 12, 322);
    j=rand('GAMMA', 7.25);
    k=rand('GEOM', 0.02);
    l=rand('GOMP', 1, 0.5);
    m=rand('GUMBEL', 0, 2);
    n=rand('HYPER', 10, 3, 5);
    o=rand('INTEGER', 1, 10);
    p=rand('LAPLACE');
    q=rand('LOGIST');
    r=rand('LOGN');
    s=rand('NEGB', 0.8, 5);
    t=rand('NORMAL');
    u=rand('PARETO', 3, 1);
    v=rand('POISSON', 6.1);
    w=rand('SHGOMP', 0.5, 1.2);
    x=rand('T', 4);
    y=rand('TABLE', .2, .5);
    z=rand('TRIANGLE',0.7);
    aa=rand('UNIFORM');
    bb=rand('WALD', 0.25, 2.1);
    cc=rand('WEIB', 1, 2);
    put a;
    put b;
    put c;
    put d;
    put e;
    put f;
    put g;
    put h;
    put i;
    put j;
    put k;
    put l;
    put m;
```

```
      put n;
      put o;
      put p;
      put q;
      put r;
      put s;
      put t;
      put u;
      put v;
      put w;
      put x;
      put y;
      put z;
      put aa;
      put bb;
      put cc;
    end;
  enddata;
  run;
  quit;
```

SAS は次の出力をログに書き込みます。この関数は乱数ジェネレーターであるため、プログラムを実行するたびに結果が異なります。

```
0
0.74302422399001
8
-0.53507631686494
16.8091595104245
9.85226977170385
-0.23569561070954
3.91526159715128
0.70109395516368
4.49313340782904
43
0.13847585083914
4.71625869848403
2
8
-0.64278704678732
2.77758410988081
0.65580499717406
4
0.58012595905153
2.07213926731126
3
0.43334467603266
0.38949035606593
2
0.27075405231828
0.01891799294389
0.08049855030189
4.57316490321742
```

参考文献

Matsumoto, M., and T. Nishimura. “Mersenne Twister with Improved Initialization.” 2002. URI: Mersenne Twister.

Fishman, George S. 1996. *Monte Carlo: Concepts, Algorithms, and Applications*. New York, USA: Springer-Verlag.

Fushimi, M., and S. Tezuka.1983. . “The k-Distribution of Generalized Feedback Shift Register Pseudorandom Numbers.” *Communications of the ACM* 26: 516-523.

Gentle, James E. 1998. *Random Number Generation and Monte Carlo Methods*. New York, USA: Springer-Verlag.

Lewis, T. G., and W. H. Payne.1973 . “Generalized Feedback Shift Register Pseudorandom Number Algorithm.” *Journal of the ACM* 20: 456-468.

Matsumoto, M., and Y. Kurita.1992 . “Twisted GFSR Generators.” *ACM Transactions on Modeling and Computer Simulation* 2: 179-194.

Matsumoto, M., and Y. Kurita.1994 . “Twisted GFSR Generators II.” *ACM Transactions on Modeling and Computer Simulation* 4: 254-266.

Matsumoto, M., and T. Nishimura.1998 . “Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator.” *ACM Transactions on Modeling and Computer Simulation* 8: 3-30.

Ripley, Brian D. 1987. *Stochastic Simulation*. New York, USA: Wiley.

Robert, C. P., and G. Casella. 1999. *Monte Carlo Statistical Methods*. New York, USA: Springer-Verlag.

Ross, Sheldon M. 199. *Simulation*. San Diego, USA: Academic Press.

RANEXP 関数

指数分布からランダム変量を返します。

注: この関数はサポートされなくなりました。かわりに RAND('EXPONENTIAL')関数 (1058 ページ)を使用してください。

RANGAM 関数

ガンマ分布からランダム変量を返します。

注: この関数はサポートされなくなりました。かわりに RAND('GAMMA')関数 (1058 ページ)を使用してください。

RANGE 関数

最大値と最小値の差を返します。

カテゴリ: 記述統計

返されるデータの種類: DOUBLE

構文

RANGE(*expression* [, ...*expression*])

必須引数

expression
数値に評価される有効な式を指定します。

要件 少なくとも 1 つの非 null または非欠損の引数が必要です。それがない場合は、関数から null または欠損値が返されます。

データの種類 DOUBLE

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

詳細

RANGE 関数は、最大と最小の非 null または非欠損引数の差を返します。

例

次のプログラムは、RANGE 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double x0 x1 x2 x3 x4;
  method run();
    x0=range(., .);
    x1=range(-2, 6, 3);
    x2=range(2, 6 ,3, .);
    x3=range(1, 6, 3, 1);
    x4=range(of x1-x3);
    put x0= x1= x2= x3= x4=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
x0=. x1=8 x2=4 x3=5 x4=4
```

RANK 関数

ASCII 照合順序で文字の位置を返します。

カテゴリ: 文字

返されるデータの種類: DOUBLE

構文

RANK(*expression*)

必須引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

詳細

RANK 関数は、ASCII 照合順序で文字の位置を表す整数を返します。2 つ以上の文字が指定されている場合、RANK 関数は、最初の文字の ASCII 照合順序での位置を返します。

注: ASCII 127 より上の文字を指定して RANK 関数を使用するプログラムはいずれも移植性がありません。(16 進表記は**'7F'x** です。)プログラムに移植性がないのは、これらの文字が国別文字であり、国によって異なるためです。

例

次のプログラムは、RANK 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl varchar x;
  method run();
    x=rank('A');
    put x;
  end;
enddata;
run;
quit;
```

SAS は次の ASCII 出力をログに書き込みます。

```
65
```

関連項目

関数:

- “BYTE 関数” (343 ページ)

RANNOR 関数

正規分布からランダム変量を返します。

注: この関数はサポートされなくなりました。かわりに RAND('NORMAL')関数 (1058 ページ)を使用してください。

RANPOI 関数

Poisson 分布からランダム変量を返します。

注: この関数はサポートされなくなりました。かわりに RAND('POISSON')関数 (1058 ページ)を使用してください。

RANTBL 関数

テーブル形式の確率分布からランダム変量を返します。

注: この関数はサポートされなくなりました。かわりに RAND('TABLE')関数 (1058 ページ)を使用してください。

RANTRI 関数

三角分布からランダム変量を返します。

注: この関数はサポートされなくなりました。かわりに RAND('TRIANGLE')関数 (1058 ページ)を使用してください。

RANUNI 関数

一様分布からランダム変量を返します。

注: この関数はサポートされなくなりました。かわりに RAND('UNIFORM')関数 (1058 ページ)を使用してください。

REPEAT 関数

文字式を繰り返します。

カテゴリ: 文字

返されるデータの種類: CHAR、NCHAR、NVARCHAR、VARCHAR

構文

REPEAT(*expression*, *n*)

必須引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

n
expression を繰り返す回数を指定します。

制限事項 *n* は、0 以上である必要があります。

データの種類 BIGINT、DOUBLE

詳細

REPEAT 関数は、*n* 回繰り返された第 1 引数で構成される文字値を返します。つまり、第 1 引数が *n*+1 回、結果に表示されます。

例

次のプログラムは、REPEAT 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl varchar(10) x;
  method run();
    x=repeat('ONE', 2);
    put x;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
ONEONEONE
```

REVERSE 関数

文字式を逆にします。

カテゴリ:	文字
返されるデータの種類:	CHAR、NCHAR、NVARCHAR、VARCHAR

構文

REVERSE(*expression*)

必須引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

詳細

REVERSE 関数が返す文字値では、式の最後の文字が結果の最初の文字になり、式の最後から 2 番目の文字が結果の 2 番目の文字になります(以下同様です)。

注: 式の末尾の空白は結果の先頭の空白になります。

例

次のプログラムは、REVERSE 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl varchar backward;
  method run();
    backward=reverse('xyz ');
    put backward=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
backward= zyx
```

RIGHT 関数

文字式を右揃えにします。

カテゴリ: 文字

返されるデータの種類: CHAR、NCHAR、NVARCHAR、VARCHAR

構文

RIGHT(*expression*)

必須引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

詳細

RIGHT 関数は、末尾の空白を値の先頭に移動した引数を返します。引数の長さは変わりません。

例

次のプログラムは、RIGHT 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl char(10) a b c;
  method run();
    a='Due Date  ';
    b=put(a, $10.);
    c=put(right(a), $10.);
    put b;
    put c;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
Due Date
 Due Date
```

```
proc ds2;
data test(overwrite=yes);
  dcl char(12) a;
  dcl char(15) b;
```

```
  method run();
    a='Due Date  ';
    b='*'||right(a);
    put a= $12. b= $15.;
  end;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
a=Due Date    b=*   Due Date
```

関連項目

関数:

- “COMPRESS 関数” (452 ページ)
- “LEFT 関数” (806 ページ)
- “TRIM 関数” (1217 ページ)

RMS 関数

2 乗平均平方根を返します。

カテゴリ:	記述統計
返されるデータの種類:	DOUBLE

構文

RMS(*expression* [, ...*expression*])

必須引数

expression
数値に評価される有効な式を指定します。

データの種類	DOUBLE
参照項目:	“DS2 式” (*SAS DS2 プログラマガイド*)

詳細

2 乗平均平方根は、値の 2 乗の算術平均の平方根です。すべての引数が null または欠損値の場合、結果は null または欠損値になります。それ以外の場合、結果は非 null または非欠損値の 2 乗平均平方根になります。

nを非 null または非欠損値を持つ引数の数とし、$x_1, x_2, \ldots, x_n$をそれらの引数の値とします。2 乗平均平方根は次のように計算されます。

$$\sqrt{\frac{x_1^2 + x_2^2 + \ldots + x_n^2}{n}}$$

例

次のプログラムは、RMS 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;data test(overwrite=yes);
  dcl double x1 x2 x3;
  method run();
    x1=rms(1,7);
    x2=rms(.,1,5,11);
    x3=rms(of x1-x2);
    put x1=;
    put x2=;
    put x3=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
x1=5
x2=7
x3=6.0827625302982
```

ROUND 関数

第 1 引数を第 2 引数の最も近い倍数(第 2 引数が省略された場合は最も近い整数)に丸めます。

カテゴリ:	切り捨て
返されるデータの種類:	DOUBLE

構文

ROUND(*expression* [, *rounding-unit*])

必須引数

expression
数値に評価される丸め対象の有効な式を指定します。

データの種類 DOUBLE

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

オプション引数

rounding-unit
丸め単位を示す正の数値式を指定します。

データの種類 DOUBLE

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

詳細

基本概念

ROUND 関数は、第 1 引数を第 2 引数の倍数の近似値に丸めます。結果は第 2 引数の正確な倍数でない場合があります。

2 進算術演算と 10 進算術演算の差異

コンピューターは、有限精度の 2 進算術演算を使用します。通常、厳密な 2 進表現がない数値を扱う場合、コンピューターによって生成される結果は 10 進算術演算で生成される結果とわずかに異なります。

たとえば、10 進値の 0.1 および 0.3 には厳密な 2 進表現はありません。10 進算術演算の 3*0.1 は厳密に 0.3 になりますが、2 進算術演算ではこの等式は成り立ちません。

次の例が示すように、**a** が float で **b** が REAL の場合、2 つの値には差があります。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl float a diff;
  dcl real b;
  method run();
    a=0.3;
```

```
      b=3*0.1;
      diff=a-b;
      put a=;
      put b=;
      put diff=;
    end;
enddata;
run;
quit;
```

次の行が SAS ログに書き込まれます。

```
a=0.3
b=0.30000001192092;
diff=-1.192092896618E-8
```

丸めの影響

丸める値に最も近い丸め単位の厳密な倍数を見つけることが丸めの定義となります。たとえば、10 進算術演算の場合、0.33 を最も近い 0.1 の倍数に丸めると、3*0.1 つまり 0.3 になります。2 進算術演算では 0.3 が 0.1 の厳密な倍数ではないため、2 進算術演算の場合、0.33 を最も近い 0.1 の倍数に丸めると 3*0.1 になりますが、0.3 にはなりません。

ROUND 関数は、数学的に正しい厳密な結果でなくても 10 進算術演算に基づいて値を返します。ROUND(0.33,0.1)の例では、ROUND は 3*0.1 ではなく 0.3 を返します。

2 進値の表現

DS2 プログラムで文字"0.3"が定数として示されている場合、値は 3/10 として計算されます。ROUND(0.33,0.1)は、標準の入力形式にあわせて結果を 3/10 として計算します。次のステートメントでは期待される結果が生成されます。

```
if round(x,0.1) = 0.3 then
  ... more DS2 statements ...
```

ただし、次のステートメントのように定数 0.3 のかわりに変数 Y を使用する場合、Y の計算方法によっては予期しない結果が生じる可能性があります。

```
if round(x,0.1) = y then
  ... more DS2 statements ...
```

ROUND が標準の入力形式を使用して Y を文字"0.3"として読み込む場合、IF ステートメントに定数 0.3 が示されている場合と同じ結果になります。ROUND が別の入力形式を使用して Y を読み込む場合や、SAS 以外のプログラムが Y を読み込む場合、文字"0.3"では厳密な 3/10 の値が生成されない可能性があります。また、厳密な 2 進表現ではない数値が計算に含まれている場合や、浮動小数点表現が異なる別の動作環境にテーブルを移植する場合も、正確な結果を得られない可能性があります。

Y が小数第 1 位の 10 進数だとわかっているが、標準の入力形式で生成される値とまったく同じかどうかわからない場合、次のステートメントを使用することをお勧めします。

```
if round(x,0.1) = round(y,0.1) then
  ... more DS2 statements ...
```

近似等式のテスト

近似等式をテストする一般的な方法として ROUND 関数を使用しないでください。最下位ビットのみが異なる 2 つの数値を丸める場合、一方の数値が切り下げられてもう一方の数値が切り上げられると、異なる値に丸められる可能性があります。近似等式のテストは、数値の計算方法によって異なります。両方の数値が高い相対精度で計算されている場合、次の例のように ABS および MAX 関数を使用して近似等式をテストできます。

```
if abs(x-y) <= 1e-12 * max( abs(x), abs(y) ) then
  ... more DS2 statements ...
```

期待される結果の生成

通常、ROUND(expression, rounding-unit)は、結果の有効桁数が 9 以下で、次のどの条件にも該当しない場合、10 進算術演算で期待される結果を生成します。

- 丸め単位が整数である。
- 丸め単位が 1e-15 以上の 10 のべき乗である。 [1]
- 10 進算術演算で期待される結果が小数第 4 位以下である。

次に例を示します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data rounding(overwrite=yes);
  method run();
    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;
  end;
enddata;
run;
quit;

proc print data=rounding; run;
```

次の出力は結果を示しています。

1. 丸め単位が 1 よりも小さい場合、ROUND は、丸め単位の逆数が 10 のべき乗との違いが最下位から第 3 または第 4 ビットまでであれば、丸め単位を 10 のべき乗として扱います。

アウトプット 7.22　丸め単位の値に基づく丸めの結果

Obs	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10
1	0	0	0	0	0	0	0	0	0	0.012345

丸め単位が整数の逆数である場合

丸め単位が整数の逆数である場合 [1]ROUND 関数は整数で除算して結果を計算します。そのため、丸め単位の倍数ではなく 2 つの整数の比率を使用して ROUND の結果を比較できます。次に例を示します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data rounding2(overwrite=yes);
  drop pi unit;
  method run();
    pi = arcos(-1);
    unit=1/7.;
    d1=round(pi,unit) - 22/7.;
    d2=round(pi, unit) - 22*unit;
  end;
enddata;
run;
quit;

proc print data=rounding2;
run;
```

次の出力は結果を示しています。

アウトプット 7.23　整数の逆数による丸めの結果

Obs	d1	d2
1	0	1.3323E-15

特殊なケースの計算結果

ROUND 関数は、次の条件にすべて該当する場合に丸め単位で整数を乗算して結果を計算します。

- 丸め単位が整数でない。
- 丸め単位が 10 のべき乗でない。

1. ROUND は、丸め単位の逆数と整数の違いが最下位から第 3 または第 4 ビットまでであれば、丸め単位を整数の逆数として扱います。

- 丸め単位が整数の逆数でない。
- 10 進算術演算で期待される結果が小数第 4 位以下である。

次に例を示します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  method run();
    difference=round(1234.56789,.11111) - 11111*.11111;
    put difference=;
  end;
enddata;
run;
quit;
```

次の行が SAS ログに書き込まれます。

```
difference=0
```

値が丸め単位の倍数の中間にある場合の計算結果

丸める値が丸め単位の 2 つの倍数のほぼ中間にある場合、ROUND 関数は絶対値を切り上げ、符合を元に戻します。次に例を示します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  method run ();
    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;
  end;
enddata;
run;
quit;

proc print data=test;
run;
```

次の出力は結果を示しています。

アウトプット 7.24　値が丸め単位の倍数の中間にある場合の丸めの結果

The SAS System

Obs	I	VALUE	ROUND
1	8	0.50000	0
2	9	0.50000	0
3	10	0.50000	0
4	11	0.50000	0
5	12	0.50000	0
6	13	0.50000	1
7	14	0.50000	1
8	15	0.50000	1
9	16	0.50000	1
10	17	0.50000	1
11	8	-0.50000	0
12	9	-0.50000	0
13	10	-0.50000	0
14	11	-0.50000	0
15	12	-0.50000	0
16	13	-0.50000	-1
17	14	-0.50000	-1
18	15	-0.50000	-1
19	16	-0.50000	-1
20	17	-0.50000	-1

近似は丸める値のサイズに関連し、次の例のように計算されます。このコード例の結果と ROUND 関数の結果がまったく同じにはならない場合があります。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data testfile(overwrite=yes);
  method run();
```

```
    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;
  end;
enddata;
run;
quit;

proc print data=testfile noobs;
  format value 19.16;
run;
```

比較

ROUND 関数は ROUNDE 関数とほぼ同じですが、第 1 引数が第 2 引数の最も近い 2 つの倍数の中間にある場合に ROUNDE が偶数の倍数を返す点が異なります。ROUND は、絶対値が大きい方の倍数を返します。

ROUNDZ 関数は、10 進算術演算で計算された結果と結果を一致させようとせずに、丸め単位の倍数を返します。

例

次のプログラムは、ROUND 関数によって返される結果と ROUNDE 関数によって返される結果を比較します。出力は、Windows 動作環境で生成されています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data results(overwrite=yes);
  dcl double x rounde round;
  method run();
    do x=0 to 4 by .25;
      Rounde=rounde(x);
      Round=round(x);
      output;
    end;
  end;
enddata;
run;
quit;

proc print data=results;
run;
```

次の出力は結果を示しています。

アウトプット 7.25　ROUND 関数と ROUNDE 関数によって返される結果

Obs	x	rounde	round
1	0.00	0	0
2	0.25	0	0
3	0.50	0	1
4	0.75	1	1
5	1.00	1	1
6	1.25	1	1
7	1.50	2	2
8	1.75	2	2
9	2.00	2	2
10	2.25	2	2
11	2.50	2	3
12	2.75	3	3
13	3.00	3	3
14	3.25	3	3
15	3.50	4	4
16	3.75	4	4
17	4.00	4	4

関連項目

関数:

- “CEIL 関数” (410 ページ)
- “CEILZ 関数” (412 ページ)
- “FLOOR 関数” (653 ページ)
- “FLOORZ 関数” (655 ページ)
- “INT 関数” (717 ページ)
- “INTZ 関数” (781 ページ)
- “ROUNDE 関数” (1097 ページ)
- “ROUNDZ 関数” (1100 ページ)

ROUNDE 関数

第 1 引数を第 2 引数の最も近い倍数に丸め、第 1 引数が 2 つの最も近い倍数の中間にある場合には偶数の倍数を返します。

カテゴリ: 切り捨て

返されるデータの種類: DOUBLE

構文

ROUNDE(*expression* [, *rounding-unit*])

必須引数

expression
数値に評価される丸め対象の有効な式を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

オプション引数

rounding-unit
丸め単位を示す正の数値式です。

デフォルト 1

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

ROUNDE 関数は、第 1 引数を第 2 引数の最も近い倍数に丸めます。

比較

ROUNDE 関数は ROUND 関数とほぼ同じですが、第 1 引数が第 2 引数の最も近い 2 つの倍数の中間にある場合に ROUNDE が偶数の倍数を返す点が異なります。
ROUND は、絶対値が大きい方の倍数を返します。

例

次のプログラムは、ROUNDE 関数によって返される結果と ROUND 関数によって返される結果を比較します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data results(overwrite=yes);
  dcl double x rounde round;
  method run();
    do x=0 to 4 by .25;
      rounde=rounde(x);
      round=round(x);
      output;
    end;
  end;
enddata;
run;
quit;
```

次の出力は結果を示しています。

アウトプット 7.26　ROUNDE 関数と ROUND 関数によって返される結果

Obs	x	rounde	round
1	0.00	0	0
2	0.25	0	0
3	0.50	0	1
4	0.75	1	1
5	1.00	1	1
6	1.25	1	1
7	1.50	2	2
8	1.75	2	2
9	2.00	2	2
10	2.25	2	2
11	2.50	2	3
12	2.75	3	3
13	3.00	3	3
14	3.25	3	3
15	3.50	4	4
16	3.75	4	4
17	4.00	4	4

関連項目

関数:

- “CEIL 関数” (410 ページ)
- “CEILZ 関数” (412 ページ)
- “FLOOR 関数” (653 ページ)
- “FLOORZ 関数” (655 ページ)
- “INT 関数” (717 ページ)
- “INTZ 関数” (781 ページ)
- “ROUND 関数” (1088 ページ)
- “ROUNDZ 関数” (1100 ページ)

ROUNDZ 関数

第 1 引数を第 2 引数の最も近い倍数にゼロファジーを使用して丸めます。

カテゴリ: 切り捨て

返されるデータの種類: DOUBLE

構文

ROUNDZ(*expression* [, *rounding-unit*])

必須引数

expression
数値に評価される有効な式を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

オプション引数

rounding-unit
数値式に評価され、丸め単位を示す有効な式を指定します。

デフォルト 1

要件 正の値のみが有効です。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

ROUNDZ 関数は、第 1 引数を第 2 引数の最も近い倍数に丸めます。

比較

ROUNDZ 関数は、ROUND 関数と同じですが、次の点が異なります。

- ROUNDZ は、第 1 引数が第 2 引数の最も近い 2 つの倍数のちょうど中間にある場合には偶数の倍数を返します。ROUND は、第 1 引数が最も近い 2 つの倍数のほぼ中間にある場合には絶対値が大きい方の倍数を返します。
- 丸め単位が 1 よりも小さい場合や整数の逆数でない場合、ROUNDZ によって返される結果は 10 進算術演算の結果と完全には一致しない可能性があります。ROUNDZ は、結果をファジー処理しません。ROUND は、結果が 10 進算術演算に一致するようにファジーと呼ばれる追加の計算を実行します。

例

例 1: ROUNDZ 関数と ROUND 関数の結果の比較

次のプログラムでは、ROUNDZ 関数によって返される結果と ROUND 関数によって返される結果を比較します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double i value roundz round;
  method run();
    do i=10 to 17;
      Value=2.5 - 10**(-i);
      Roundz=roundz(value);
      Round=round(value);
      output;
    end;
    do i=16 to 12 by -1;
      value=2.5 + 10**(-i);
      roundz=roundz(value);
      round=round(value);
      output;
    end;
  end;
enddata;
run;
quit;

proc print data=test;
format value 19.16;
quit;
```

次の出力は結果を示しています。

アウトプット 7.27　ROUNDZ 関数と ROUND 関数によって返される結果

Obs	i	value	roundz	round
1	10	2.4999999999000000	2	2
2	11	2.4999999999900000	2	2
3	12	2.4999999999990000	2	3
4	13	2.4999999999999000	2	3
5	14	2.4999999999999900	2	3
6	15	2.4999999999999900	2	3
7	16	2.5000000000000000	2	3
8	17	2.5000000000000000	2	3
9	16	2.5000000000000000	2	3
10	15	2.5000000000000000	3	3
11	14	2.5000000000000100	3	3
12	13	2.5000000000001000	3	3
13	12	2.5000000000010000	3	3

例 2: ROUNDZ 関数の出力例

次のプログラムは、ROUNDZ 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double x a b c d e;
  method run();
    x=223.456;
    a=roundz(x,1);
    b=roundz(x, .01);
    c=roundz(x, 100);
    d=roundz(x);
    e=roundz(x, .3);
    put a= b= c= d= e=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
a=223 b=223.46 c=200 d=223 e=223.5
```

関連項目

関数:

- “ROUND 関数” (1088 ページ)
- “ROUNDE 関数” (1097 ページ)

SASLOGLINENUMBER 関数

SAS ログ内の現在のステートメントの行番号を返します。

カテゴリ:	特殊
返されるデータの種類:	INTEGER

構文

SASLOGLINENUMBER( )

引数なし

SASLOGLINENUMBER()関数には引数がありません。

詳細

SASLOGLINENUMBER 関数は、SAS ログ内の現在のステートメントの行番号を返します。SAS ログにリストされていないストアドパッケージまたはストアドスレッドの場合、SASLOGLINENUMBER は、ストアドパッケージの PACKAGE ステートメントまたはストアドスレッドの THREAD ステートメントからオフセットされた現在のステートメントの行番号を返します。

例

次のプログラムは、SASLOGLINENUMBER 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
resetline;
proc ds2;
thread testthread / overwrite=yes;
```

```
    dcl char a;
    dcl int threadlogline;
    method init();
        a = 'apple';
      threadlogline = sasloglinenumber();
      put threadlogline=;
    end;
  endthread;

  data _null_;
    dcl thread testthread t();
    dcl int datalogline;
    dcl char b;
    method init();
      b = 'banana';
      datalogline = sasloglinenumber();
      put datalogline=;
    end;

    method run();
      set from t threads = 1;
   end;
  enddata;
  run;
  quit;
```

SAS は次の出力をログに書き込みます。

```
78   resetline;
1    proc ds2;
2
3    thread testthread / overwrite=yes;
4     dcl char a;
5     dcl int threadlogline;
6     method init();
7    a = 'apple';
8       threadlogline = sasloglinenumber();
9    put threadlogline=;
10     end;
11   endthread;
12
13   data _null_;
14   dcl thread testthread t();
15   dcl int datalogline;
16   dcl char b;
17     method init();
18       b = 'banana';
19       datalogline = sasloglinenumber();
20     put datalogline=;
21     end;
22
23     method run();
24   set from t threads = 1;
25     end;
26   enddata;
27   run;
27 !      quit;
datalogline=19
threadlogline=8
```

関連項目

“PROGRAMBLOCKLINENUMBER 関数” (1013 ページ)

SAVING 関数

定期預金の将来価値を返します。

カテゴリ:	財務
返されるデータの種類:	DOUBLE

構文

SAVING(*f*, *p*, *r*, *n*)

必須引数

f
数値の将来総額です(*n* 期間の終了時)。

範囲	$f \geq 0$
データの種類	DOUBLE

p
数値の固定定期的支払いです。

範囲	$p \geq 0$
データの種類	DOUBLE

r
10 進で表される数値の定期利率です。

範囲	$r \geq 0$
データの種類	DOUBLE

n
整数の複利期間数です。

範囲	$n \geq 0$
データの種類	DOUBLE

詳細

SAVING 関数は、定期預金の 4 つの引数のリストの欠損引数を返します。これらの引数には次の式の関係があります。

$$f = \frac{p(1+r)\left((1+r)^n - 1\right)}{r}$$

引数は 1 つを欠損値とする必要があります。次に、残りの 3 つから計算されます。結果を変換して丸めた数字にする調整は行われません。

例

ある定期預金口座は 5%の名目年利が支払われ、毎月複利計算されます。毎月$100 を入金する場合、少なくとも累計が$12,000 になるために必要な支払い回数は、次のように表すことができます。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double a;
  method run();
    a=saving(12000, 100, .05/12, .);
    put  'a= ' a;
  end;
enddata;
run;
quit;
```

返される値は 97.18 か月です。第 4 引数には欠損値が設定されます。これは、支払い回数を計算することを示します。5%の名目年利は、0.05/12 の月利に変換されます。rate は分数(パーセントではない)で表す複利計算期間当たりの利率です。

関連項目

関数:

- “SAVINGS 関数” (1106 ページ)

SAVINGS 関数

変動金利を使用して定期預金の残高を返します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

SAVINGS(*base-date*, *initial-deposit-date*, *deposit-amount*, *deposit-number*, *deposit-interval*, *compounding-interval*, *date-1*, *rate-1*[, ...*date-n*, *rate-n*])

必須引数

base-date
返される値が基準日の預金残高になるように指定します。

要件 *base-date* は SAS 日付です。

データの種類 DOUBLE

initial-deposit-date
初回入金日を指定します。後続の入金間隔の開始時に次の入金が行われます。

要件 *initial-deposit-date* は SAS 日付です。

データの種類 DOUBLE

deposit-amount
各入金の値を指定します。すべての 入金は定数を前提としています。

データの種類 DOUBLE

deposit-number
入金の回数を指定します。

データの種類 DOUBLE

deposit-interval
入金の頻度を指定します。

要件 *deposit-interval* は SAS 間隔です。

データの種類 CHAR

compounding-interval
複利間隔を指定します。

要件 *compounding-interval* は SAS 間隔です。

データの種類 CHAR

date
rate が有効になるタイミングを指定します。各日付は利率とペアになります。

要件 *date* は SAS 日付です。

データの種類 DOUBLE

rate
date に開始する利率を数値のパーセンテージで指定します。各利率は日付とペアになります。

データの種類 DOUBLE

詳細

SAVINGS 関数には、次の詳細情報が適用されます。

- 利率の値は–99 から 120 までの値にする必要があります。
- *deposit-interval* は'CONTINUOUS'にすることはできません。
- 日付-利率ペアのリストは日付順にする必要はありません。
- 1 日に複数の利率が変更される場合、SAVINGS 関数は、その日付でリストされている最後の利率のみを適用します。
- 一部の期間には単利が適用されます。
- *initial-deposit-date* と *base-date* の両方の日付またはその前の日付となる有効な日付-利率ペアが必要になります。

例

- 年利 4%で年 4 回複利計算される口座に 2 年間毎月$300 を入金する場合、5 年後の口座残高は次のように表すことができます。

```
proc ds2;
data _null_;
  dcl double bd idd d amount_base1;
  method run();
    bd=to_double(date'2005-01-01');
    idd=to_double(date'2000-01-01');
    d=to_double(date'2000-01-01');
    amount_base1=savings(bd, idd, 300, 24, 'month', 'qtr', d, 4.00);
    put amount_base1;
  end;
enddata;
run;
quit;
```

次の行が SAS ログに書き込まれます。

```
8458.79415896917
```

- 金利が毎年 1/4 ポイント上昇すると、口座の残高は次のように表すことができます。

```
proc ds2;
data _null_;
  dcl double bd idd d1 d2 d3 d4 d5 amount_base2;
```

```
  method run();
    bd= to_double(date'2005-01-01');
    idd= to_double(date'2000-01-01');
    d1= to_double(date'2000-01-01');
    d2= to_double(date'2001-01-01');
    d3= to_double(date'2002-01-01');
    d4= to_double(date'2003-01-01');
    d4= to_double(date'2004-01-01');
    amount_base2 = savings(bd, idd, 300, 24, 'month', 'qtr', d1, 4.00, d2,
      4.25, d3, 4.50, d4, 4.75, d5, 5.0);
    put amount_base2;
  end;
enddata;
run;
quit;
```

次の行が SAS ログに書き込まれます。

```
8623.09024586998
```

- 1 年間の預金の後の残高を決定するために、次のプログラムは amount_base3 を目的の残高に設定します。

```
proc ds2;
data _null_;
  dcl double bd idd d amount_base3;
  method run();
    bd= to_double(date'2001-01-01');
    idd= to_double(date'2000-01-01');
    d= to_double(date'2000-01-01');
    amount_base3 = savings(bd, idd, 300, 24, 'month', 'qtr', d, 4);
    put amount_base3;
  end;
enddata;
run;
quit;
```

次の行が SAS ログに書き込まれます。

```
3978.69037121739
```

SAVINGS 関数は、基準日後の入金を無視するため、基準日後の入金は戻り値に影響しません。

関連項目

関数:

- “SAVING 関数” (1105 ページ)

SCAN 関数

文字式から *n* 番目の単語を返します。

カテゴリ: 文字

返されるデータの種類: CHAR、NCHAR、NVARCHAR、VARCHAR

構文

SCAN(*expression*, *n* [, *delimiters*[, *modifier*]])

必須引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

n
SCAN で選択する文字式の単語番号を指定するゼロ以外の数値式です。次のルールが適用されます。*n* が正の場合、SCAN は文字列内の単語を左から右にカウントします。*n* が負の場合、SCAN は文字列内の単語を右から左にカウントします。*n* が *expression* の単語数より大きい場合、SCAN は空白値を返します。

SCAN で選択する文字式の単語番号を指定するゼロ以外の数値式です。次の規則が適用されます。

- *n* が正の場合、SCAN は文字列内の単語を左から右にカウントします。
- *n* が負の場合、SCAN は文字列内の単語を右から左にカウントします。
- *n* が *expression* の単語数より大きい場合、SCAN は空白値を返します。

オプション引数

delimiters
文字列を評価するか文字列に強制変換できて、SCAN で式の単語区切りとして使用される有効な式を指定します。

デフォルト

要件 *delimiter* が定数の場合は、*delimiter* を単一引用符で囲みます。

操作 ASCII のデフォルトの区切り文字は、空白 ! $ % & ( ) * + , – . / ; < |です。^文字のない環境では、SCAN はかわりに~文字を使用します。

修飾子を指定すると、*delimiter* の文字を変更できます。たとえば、*modifier* 引数で K 修飾子を指定すると、*delimiter* に含まれていないすべての文字が区切り文字として使用されます。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

ヒント 他の修飾子を使用して、*delimiter* にさらに文字を追加できます。

参照項目: “ASCII 環境でデフォルトの区切り文字を使用する” (1113 ページ)

“DS2 式” (*SAS DS2 プログラマガイド*)

modifier
文字定数、変数または式を指定します。これらの空白以外の各文字で SCAN 関数のアクションを変更します。空白は無視され、大文字と小文字は区別されません。次の文字を修飾子として使用します。

A
文字のリストにアルファベット文字を追加します。

B
count 引数の符号に関係なく、左から右ではなく、右から左に逆方向にスキャンします。

C
文字のリストに制御文字を追加します。

D
文字のリストに数字を追加します。

F
文字のリストにアンダースコアと 英字を追加します。

G
文字のリストにグラフィカル文字を追加します。グラフィカル文字は、紙面に印刷するとイメージになる文字です。

H
文字のリストに水平タブを追加します。

I
大文字か小文字かは無視します。

K
文字のリストに含まれていないすべての 文字を区切り文字として扱うようにします。つまり、K が指定されている場合、文字のリストにある文字は、区切り文字であるために省略されるのではなく、返された値に保持されます。K を指定しない場合、文字のリストに含まれているすべての 文字が区切り文字として扱われます。

L
小文字を文字リストに追加します。

M
複数の連続する区切り文字、および *string* 引数の先頭または末尾の区切り文字が、長さがゼロの単語を参照するように指定します。M 修飾子を指定しない場合、複数の連続する区切り文字は 1 つの区切り文字として扱われ、*string* 引数の先頭または末尾の区切り文字は無視されます。

N
文字のリストに数字、アンダースコア、および英字を追加します。

O
charlist および *modifier* 引数を 1 回だけ処理します。SCAN 関数の呼び出し時毎には処理されません。DATA ステップ(WHERE 句以外)または SQL プロシジャで O 修飾子を使用すると、*character-list* 引数および *modifier* 引数に変更がないループで SCAN が呼び出されるとき、より迅速に実行できます。O 修飾子は SAS コードの SCAN 関数の各インスタンスに個別に適用され、SCAN 関数のすべてのインスタンスで 同じ区切り文字および修飾子が使用されるようには*なりません*。

P
文字のリストに句読点を追加します。

Q
引用符で囲まれた部分文字列内の区切り文字を無視します。*string* 引数の値に、対になっていない引用符が含まれている場合、左から右へのスキャンと、右から左へのスキャンとでは異なる単語が生成されます。

R
SCAN が返す単語から先頭および末尾の空白を削除します。Q 修飾子と R 修飾子を指定すると、SCAN 関数は単語から先頭および末尾の空白をまず削除します。次に、SCAN は単語が引用符で始まる場合には単語から引用符の層も 1 つ削除します。

S
文字のリストに空白文字(空白、水平タブ、垂直タブ、キャリッジリターン、ラインフィード、フォームフィード)を追加します。

T
string 引数と *charlist* 引数から末尾の空白をトリムします。両方の文字引数ではなく 1 つの文字引数からのみ末尾の空白を削除する場合、T 修飾子を指定して SCAN 関数を使用するかわりに TRIM 関数を使用します。

U
大文字を文字リストに追加します。

W
印刷可能文字を文字リストに追加します。

X
文字のリストに 16 進文字を追加します。

制限事項　この引数は、SAS Viya プラットフォームでのみサポートされます。

ヒント　*modifier* 引数が文字定数の場合は、引数を引用符で囲みます。一組の引用符で複数の修飾子を指定します。引数 *modifier* には、文字変数や文字式も指定できます。

詳細

"区切り文字"と"単語"の定義

*区切り文字*とは、単語を区切るために使用される複数の文字のどれかです。*delimiter* 引数と *modifier* 引数で区切り文字を指定できます。

Q 修飾子を指定すると、引用符で囲まれた部分文字列内部の区切り文字は無視されます。

SCAN 関数では、"単語"は、次の特徴をすべて備えた部分文字列を示します。

- 左側が区切り文字で囲まれているか、文字列の先頭である
- 右側が区切り文字で囲まれているか、文字列の末尾である
- 区切り文字が含まれていない

文字列の先頭または末尾に区切り文字がある場合、または文字列に 2 つ以上の連続する区切り文字が含まれている場合、単語の長さがゼロになることがあります。ただし、SCAN 関数では、M 修飾子を指定しなければ、長さがゼロの単語は無視されます。

注: ”単語”の定義は SCAN 関数と COUNTW 関数に共通します。

ASCII 環境でデフォルトの区切り文字を使用する

2 つの引数のみを指定して SCAN 関数を使用する場合、デフォルトの区切り文字は、コンピューターで使用している文字(ASCII または EBCDIC)によって異なります。

- お使いのコンピューターが ASCII 文字を使用している場合、デフォルトの区切り文字は次のとおりです。

  空白 ! $ % & ( ) * + , - ./ ; < ^ |

区切り文字とする文字を指定せずに *modifier* 引数を使用すると、使用される区切り文字は *modifier* 引数で定義される区切り文字のみになります。この場合、ASCII 環境のデフォルトの区切り文字のリストは使用されません。つまり、修飾子は、*delimiter* 引数によって明示的に指定された区切り文字のリストに追加されます。修飾子は、デフォルトの修飾子のリストには追加しません。

結果の長さ

式の最初の単語の前にある先頭の区切り文字は、SCAN に影響しません。連続する区切り文字が 2 つ以上ある場合、SCAN はそれらを 1 つとして扱います。

DS2 では、SCAN 関数が、まだ長さが指定されていない変数に値を返す場合、その変数には第 1 引数の長さが指定されます。SCAN 関数で第 1 引数の長さとは異なる値を変数に割り当てる必要がある場合は、SCAN 関数を使用するステートメントの前に、その変数の DECLARE ステートメントを使用します。

SCAN 関数によって返される単語の最小長は、M 修飾子が指定されているかどうかによって異なります。“M 修飾子ありで SCAN 関数を使用” (1113 ページ)を参照してください。“M 修飾子なしで SCAN 関数を使用” (1114 ページ)も参照してください。

M 修飾子ありで SCAN 関数を使用

M 修飾子を指定すると、文字列内の単語数は文字列内の区切り文字数に 1 を足した数になります。ただし、Q 修飾子を指定すると、引用符内の区切り文字は無視されます。

M 修飾子を指定すると、SCAN 関数は次のいずれかの条件に該当する場合に長さがゼロの単語を返します。

- 文字列の先頭が区切り文字であり、ユーザーが最初の単語を要求した場合
- 文字列の末尾が区切り文字であり、ユーザーが最後の単語を要求した場合
- 文字列が 2 つの連続する区切り文字を含んでおり、ユーザーがこれら 2 つの区切り文字間にある単語を要求した場合

M 修飾子なしで SCAN 関数を使用

M 修飾子を指定しない場合、文字列内の単語数は連続する非区切り文字の最大部分文字列数になります。ただし、Q 修飾子を指定すると、引用符内の区切り文字は無視されます。

M 修飾子を指定しないと、SCAN 関数は次のように動作します。

- 文字列の最初または最後の区切り文字は無視されます。
- 2 つ以上の連続する区切り文字を、単一の区切り文字のように扱います。

文字列に区切り文字以外の文字が含まれていない場合や、文字列の単語数よりも絶対値の大きい count を指定する場合、SCAN 関数は次のいずれかの項目を返します。

- 1 つの空白(DATA ステップから SCAN 関数を呼び出す場合)
- 長さがゼロの文字列(マクロプロセッサから SCAN 関数を呼び出す場合)

null 引数の使用

SCAN 関数では、文字引数を null に指定できます。null 引数は長さがゼロの文字列として扱われます。数値引数は null にできません。

SBCS および DBCS データの処理

SCAN 関数は、SBCS データを処理するように設計されていますが、DBCS データを処理することもできます。基準は次のとおりです。

- *expression* が VARCHAR として宣言されておらず、シングルバイトデータを処理している場合、SCAN は SBCS を処理します。
- *string* が VARCHAR として宣言されていて、マルチバイトデータを処理している場合、SCAN は DBCS を処理します。

例

例 1: 文字列内の最初と最後の単語を検索する

文字列内の最初と最後の単語をスキャンする例を次に示します。

- カウントを負に指定すると、SCAN 関数は右から左へスキャンします。
- M 修飾子が使用されていないため、先頭と末尾の区切り文字は無視されます。
- 最終オブザベーションでは、文字列のすべての文字が区切り文字となります。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl varchar(8) first_word last_word;
  dcl varchar(29) string1 string2 string3 string4;
  dcl double i;
  method run();
    string1='Jack and Jill';
    first_word=scan(string1, 1);
    last_word=scan(string1, -1);
    put first_word= last_word=;
    string2='& Bob & Carol & Ted & Alice &';
    first_word=scan(string2, 1);
    last_word=scan(string2, -1);
    put first_word= last_word=;
    string3='Leonardo';
    first_word=scan(string3, 1);
    last_word=scan(string3, -1);
    put first_word= last_word=;
    string4='! $ % & ( ) * , - . / ;';
    first_word=scan(string4, 1);
    last_word=scan(string4, -1);
    put first_word= last_word=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
first_word=Jack last_word=Jill
first_word=Bob last_word=Alice
first_word=Leonardo last_word=Leonardo
first_word=  last_word=
```

例 2: M 修飾子を使用せずに文字列内のすべての単語を検索する

この例では、戻される単語が空白になるまで文字列を左から右へスキャンします。M 修飾子が使用されていないため、SCAN 関数は長さがゼロの単語を返しません。デフォルトの区切り文字に空白が含まれているため、SCAN 関数はカウントが文字列の単語数を超えている場合にのみ空白の単語を返します。そのため、SCAN が空白の単語を返す場合、ループが停止する可能性があります。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(drop=(string) overwrite=yes);
  dcl varchar(47) string word;
  dcl double count;
  method run();
    string=' The quick brown fox jumps over the lazy dog.   ';
```

```
    do until(word=' ');
      count+1;
      word=scan(string, count);
      output;
    end;
  end;
enddata;
run;
quit;

proc print data=test;
run;
```

Obs	word	count
1	The	1
2	quick	2
3	brown	3
4	fox	4
5	jumps	5
6	over	6
7	the	7
8	lazy	8
9	dog	9
10		10

例 3: M 修飾子と O 修飾子を使用して文字列内のすべての単語を検索する

区切り文字としてカンマを指定した M 修飾子を使用した結果の例を次に示します。M 修飾子を使用すると、先頭、末尾および複数の連続する区切り文字がある場合、SCAN 関数は長さがゼロの単語を返します。したがって、空白単語のテストではループが終了しないようにします。かわりに、同じ修飾子と区切り文字の COUNTW 関数を使用して、文字列の単語を数えることができます。

区切り文字と修飾子は SCAN および COUNTW 関数の各呼び出しで同じであるため、効率を上げるために O 修飾子を使用します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data comma(keep=(count word) overwrite=yes);
  dcl varchar(45) string;
  dcl varchar(20) delim modif word;
  dcl double nwords count;
  method run();
    string=',leading, trailing,and multiple,,delimeters,,';
    delim=',';
    modif='mo';
    nwords=countw(string, delim, modif);
    put nwords=;
    do count=1 to nwords;
      word=scan(string, count, delim, modif);
```

```
      output;
    end;
  end;
enddata;
run;
quit;

proc print data=comma;
run;
```

Obs	word	count
1		1
2	leading	2
3	trailing	3
4	and multiple	4
5		5
6	delimeters	6
7		7
8		8

例 4: カンマ区切り値、引用符で囲まれた部分文字列、O 修飾子、R 修飾子の使用

この例では、O 修飾子とカンマ区切り文字および R 修飾子を指定した SCAN 関数と、O 修飾子、カンマ区切り文字のみを指定した SCAN 関数を使用します。

SCAN または COUNTW 関数の各呼び出しでは区切り文字と修飾子は変わらないため、効率を上げるために O 修飾子を使用します。O 修飾子は SCAN 関数の 2 つの各インスタンスに個別に適用されます。

- SCAN 関数の最初のインスタンスは、SCAN の各呼び出しで同じ区切り文字と修飾子を使用します。
- SCAN 関数の 2 番目のインスタンスは、SCAN の各呼び出しで同じ区切り文字と修飾子を使用します。
- SCAN 関数の最初のインスタンスは 2 番目のインスタンスと同じ修飾子を使用しませんが、O 修飾子を使用しても問題ありません。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(keep=(word word_r count) overwrite=yes);
  dcl varchar(40) string;
  dcl varchar(20) word word_r;
  dcl varchar delim modif;
  dcl double count nwords;
  method run();
    string='Hello, "She said, ""Hi!""", not "Bye!"';
```

```
    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;
  end;
enddata;
run;
quit;

proc print data=test;
run;
```

Obs	word	word_r	count
1	Hello	Hello	1
2	"She said, ""Hi!"""	She said, "Hi!"	2
3	not "Bye!"	not "Bye!"	3

例 5: D 修飾子と K 修飾子を使用して数字の部分文字列を検索する

この例では、数字の部分文字列を検索します。character-list 引数は null です。つまり、文字のリストの初期状態は空です。D 修飾子は文字のリストに数字を追加します。K 修飾子はリストに含まれていない文字をすべて区切り文字として扱います。したがって、数字以外の文字はすべて区切り文字になります。

注: このサンプルコードでは、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(keep=(count digits) overwrite=yes);
  dcl char(25) string digits;
  dcl double count;
  method run();
    string='Call (800) 555-1234 now!';
    do until(digits=' ');
      count+1;
      digits=scan(string, count, '', 'dko');
      output;
    end;

  end;
enddata;
run;
quit;

proc print data=test;
run;
```

Obs	digits	count
1	800	1
2	555	2
3	1234	3
4		4

例 6: M 修飾子と O 修飾子を使用して文字列内のすべての単語を検索する

区切り文字としてカンマを指定した M 修飾子を使用した結果の例を次に示します。M 修飾子を使用すると、先頭、末尾および複数の連続する区切り文字がある場合、SCAN 関数は長さがゼロの単語を返します。したがって、空白単語のテストではループが終了しないようにします。かわりに、同じ修飾子と区切り文字の COUNTW 関数を使用して、文字列の単語を数えることができます。

区切り文字と修飾子は SCAN および COUNTW 関数の各呼び出しで同じであるため、効率を上げるために O 修飾子を使用します。

注: この例は、SAS Viya プラットフォームでのみ有効です。

```
cas;

proc ds2 sessref=casauto;
data comma (keep= (count word) overwrite=yes);
  dcl char(45) string;
  dcl char(12) delim modif word;
  dcl double nwords count;
  method run();
    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;
  end;
enddata;
run;
quit;

libname mycas cas sessref=casauto;
proc print data=mycas.comma;
run;
```

The SAS System

Obs	word	count
1	leading	1
2		2
3	trailing	3
4	and	4
5	multiple	5
6		6
7	delimiters	7
8		8
9		9
10		10

SDF 関数

生存関数を返します。

カテゴリ: 確率

参照項目: “CDF 関数” (359 ページ)

構文

SDF(*'distribution'*, *quantile*, *parameter-1, ..., parameter-k*)

必須引数

distribution
分布を識別する文字列です。

注: 各 SDF 分布関数の引数は、対応する CDF 分布関数の引数と同じです。

有効な分布は次のとおりです。

分布	引数
Bernoulli	'BERNOULLI'
ベータ	'BETA'
二項	'BINOMIAL'
Cauchy	'CAUCHY'
カイ 2 乗	'CHISQUARE'

分布	引数
Conway-Maxwell-Poisson	'CONMAXPOI'
指数	'EXPONENTIAL'
F	'F'
ガンマ	'GAMMA'
一般化 Poisson	'GENPOISSON'
幾何	'GEOMETRIC'
超幾何	'HYPERGEOMETRIC'
Laplace	'LAPLACE'
ロジスティック	'LOGISTIC'
対数正規	'LOGNORMAL'
負の二項	'NEGBINOMIAL'
正規	'NORMAL'|'GAUSS'
正規混合	'NORMALMIX'
パレート	'PARETO'
Poisson	'POISSON'
T	'T'
Tweedie	'TWEEDIE'
一様	'UNIFORM'
Wald (逆ガウス)	'WALD'|'IGAUSS'
Weibull	'WEIBULL'

注 T、F および NORMALMIX を除き、最初の 4 文字で分布を最小限に識別できます。

quantile
確率変数の値を指定する、数値定数、変数または式です。

データの種類 DOUBLE

parameter-1, ..., parameter-k
特定の分布に適した任意の *shape*、*location* または *scale* パラメーターです。

データの種類 DOUBLE

詳細

SDF 関数は、さまざまな連続分布および離散分布の生存関数(上裾)を計算します。詳細については、前述の表にリストされている個々の分布を参照してください。

Conway-Maxwell-Poisson 分布の SDF 関数の形式は次のとおりです。

SDF('CONMAXPOI',*y*,*λ*,*ν*)

y
負でない整数でカウントデータを表します。

λ
Poisson 分布の場合と同様で、平均に似ています。

ν
ばらつきのパラメーターです。

SDF 関数は、カウント値が *y* より大きくなる確率を返します。

詳細については、PDF 関数の“Conway-Maxwell-Poisson” (930 ページ)を参照してください。

例

次のプログラムは、SDF 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null;
  dcl double y;
  method run();
    y=sdf('BERN', 0, .25);
    put 'Bern dist:     ' y;
    y=sdf('BETA', 0.2, 3,4);
    put 'Beta dist:     ' y;
    y=sdf('BINOM', 4, .5, 10);
    put 'Binom dist:    ' y;
    y=sdf('CAUCHY', 2);
    put 'Cauchy dist:   ' y;
    y=sdf('CHISQ', 11.264, 11);
    put 'Chisq dist:    ' y;
```

```
    y=sdf('CONMAXPOI', 12, 2.3, .4);
    put 'Conmaxpoi dist: ' y;
    y=sdf('EXPO', 1);
    put 'Expo dist:      ' y;
    y=sdf('F', 3.32, 2,3);
    put 'F dist:         ' y;
    y=sdf('GAMMA', 1, 3);
    put 'Gamma dist:     ' y;
    y=sdf('GENPOISSON', .9, 1, .7);
    put 'Genpoisson dist:' y;
    y=sdf('GEOMETRIC', .5, .3);
    put 'Geometric dist: ' y;
    y=sdf('HYPER', 2, 200, 50, 10);
    put 'Hyper dist:     ' y;
    y=sdf('LAPLACE',1);
    put 'Laplace dist:   ' y;
    y=sdf('LOGISTIC', 1);
    put 'Logistic dist:  ' y;
    y=sdf('LOGNORMAL', 1);
    put 'LogNormal dist: ' y;
    y=sdf('NEGB', 1, .5,2);
    put 'NegB dist:      ' y;
    y=sdf('NORMAL', 1.96);
    put 'Normal dist:    ' y;
    y=sdf('NORMALMIX', 2.3, 3, .33, .33, .34, .5, 1.5, 2.5, .79, 1.6, 4.3);
    put 'Normal mix dist:' y;
    y=sdf('PARETO', 1, 1);
    put 'Pareto dist:    ' y;
    y=sdf('POISSON', 2, 1);
    put 'Poisson dist:   ' y;
    y=sdf('T', .9, 5);
    put 'T dist:         ' y;
    y=sdf('TWEEDIE', .8, 5);
    put 'Tweedie dist:   ' y;
    y=sdf('UNIFORM', 0.25);
    put 'Uniform dist:   ' y;
    y=sdf('WALD', 1, 2);
    put 'Wald dist:      ' y;
    y=sdf('WEIBULL', 1, 2);
    put 'Weibull dist:   ' y;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
Bern dist:      0.25
Beta dist:      0.90112
Binom dist:     0.62304687499999
Cauchy dist:    0.14758361765043
Chisq dist:     0.42141867068269
Conmaxpoi dist: 0.19705138765343
Expo dist:      0.36787944117144
F dist:         0.17360663977568
Gamma dist:     0.9196986029286
Genpoisson dist: 0.63212055882855
Geometric dist: 0.7
Hyper dist:     0.47632659187826
Laplace dist:   0.18393972058572
Logistic dist:  0.26894142136999
LogNormal dist: 0.5
NegB dist:      0.5
Normal dist:    0.02499789514822
Normal mix dist: 0.28186912768685
Pareto dist:    1
Poisson dist:   0.08030139707139
T dist:         0.20468560017231
Tweedie dist:   0.40823708356802
Uniform dist:   0.75
Wald dist:      0.37230216184474
Weibull dist:   0.36787944117144
```

関連項目

関数:

- “CDF 関数” (359 ページ)
- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “PDF 関数” (918 ページ)

SEC 関数

正割を返します。

カテゴリ:	三角関数
返されるデータの種類:	DOUBLE

構文

SEC(*expression*)

必須引数

expression
数値に評価される有効な式をラジアンで表して指定します。

制限事項 *expression* は、PI/2 の奇数倍にはできません。

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

比較

SEC 関数は COS 関数と関係があります。

sec(x)=1/cos(x)

例

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
 proc ds2;
data _null_;
  dcl double x y z;
  method run();
    x=sec(0.5);
    y=sec(0);
    z=sec(3.14159/3);
    put x= y= z=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
x=1.13949392732454 y=1 z=1.99999693590391
```

関連項目

関数:

- “COS 関数” (468 ページ)
- “COT 関数” (470 ページ)
- “CSC 関数” (483 ページ)
- “SIN 関数” (1138 ページ)

- “TAN 関数” (1185 ページ)

SECOND 関数

SAS 時間または日時値から秒を返します。

カテゴリ: 日付と時間

返されるデータの種類: DOUBLE

構文

SECOND(*time* | *datetime*)

必須引数

time
SAS 時間値を表す有効な式を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

datetime
SAS 日時値を表す有効な式を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

SECOND 関数は、特定の秒部分を表す数値を生成します。結果は>= 0 かつ< 60 の数値になります。

例

例 1: 基本的な SECOND 関数の使用法

次のプログラムは、SECOND 関数を示しています。

注: このサンプルコードでは、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double a b c d e f;
  method run();
    a=hms(3,19,24);
    b=second(a);
    put b=;
    c=hms(6,25,65);
    d=second(c);
    put d=;
    e=hms(3,19,60);
    f=second(e);
    put f=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
b=24
d=5
f=0
```

例 2: 現在の日付と時間に関連付けられている分数の秒数の表示

このプログラムでは、日時に関連付けられている分数の秒数が示されます。ミリ秒数の最初の 3 桁の 876 が関連しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double x;
  method run();
    x=second(datetime());
    put x=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
x=18.8769989013671
```

関連項目

- “DS2 式” (*SAS DS2 プログラマガイド*)

関数:

- “HOUR 関数” (699 ページ)
- “MINUTE 関数” (854 ページ)

SETTHREADERRORSTATUS 関数

致命的ではないランタイムエラーに対してユーザー指定のエラーステータス値を設定し、オプションでユーザー指定のメッセージを SAS ログに記録します。

カテゴリ:	エラーチェック
要件	SETTHREADERRORSTATUS 関数は、GETTHREADERRORSTATUS 関数 (680 ページ) と組み合わせて使用されます。
返されるデータの種類:	INTEGER
注:	SETTHREADERRORSTATUS 関数によって生成されたエラーメッセージは、App.TableServices.DS2.Runtime.Log ロガーによって捕捉されます。このロガーの詳細については、“ランタイムロガー” (*SAS DS2 プログラマガイド*)を参照してください。

構文

SETTHREADERRORSTATUS(*status-value*[, '*string*'])

必須引数

status-value
任意の整数を指定します。

オプション引数

'*string*'
オプションのユーザー定義の ERROR メッセージを指定します。この関数は、指定されたテキストの先頭に単語 ERROR を付加するため、メッセージ文字列に単語 ERROR を含める必要はありません。

データの種類	CHARACTER、NCHAR、NVARCHAR、VARCHAR

注 DS2_OPTIONS ステートメントで MSGORDER=TEMPORAL が設定されていない限り、500 文字を超えるメッセージは切り捨てられます MSGORDER=TEMPORAL が設定されている場合、長さに関係なく、完全なエラーメッセージがログに報告されます。

SETTHREADERRORSTATUS でエラーステータス値 0 が指定されている場合でも、すべてのメッセージは ERROR 接頭辞付きでログに出力されます。

詳細

SETTHREADERRORSTATUS 関数を使用すると、GETTHREADERRORSTATUS 関数によってクエリされるエラーステータス変数に対して、ユーザー指定のステータス値とオプションのユーザー指定の ERROR メッセージを設定できます。この関数を使用して、カスタムエラーステータス値を指定したり、内部ステータス変数を明示的に 0 (ゼロ)にリセットしたりできます。

この関数は、エラーステータスの設定が成功したか、関数が失敗したかを示す整数を返します。値 0 は関数が成功したことを示し、値 1 はエラーが発生したことを示します。

SETTHREADERRORSTATUS 関数は、1 つのスレッドのみのエラーステータス値を設定します。複数のスレッドが DS2 THREAD ブロックと DS2 DATA ブロックを実行している場合、この関数は現在のスレッドのエラーステータス値のみを更新します。他のスレッドのエラーステータス値は影響を受けません。

自動変数_N_および_THREADID_をテキスト文字列で直接指定することはできません。ただし、CAT 関数 (344 ページ)を使用すると、それらを含むテキスト文字列を構築できます。

例: SETTHREADERRORSTATUS 関数による ERROR メッセージの指定

この例は、PUT ステートメントではなく SETTHREADERRORSTATUS 関数の 'STRING'引数でカスタムエラー条件のエラーメッセージを指定している点を除いて、“例 2: スレッドプログラムでの GETTHREADERRORSTATUS 関数の使用”と同じです。

```
proc ds2;
  data largeNumberSet (overwrite=yes);
  dcl int row;

  method run();
  dcl int x;
   do x = 1 to 10000;
     if (mod(x, 1000) = 0) then row = NULL;
     else row = x;
     output;
   end;
  end;
```

```
enddata;
run;
quit;

proc ds2;
  thread processDataset / overwrite=yes;
  dcl int partial_sum;
  dcl char(100) msg;
  retain partial_sum;

  /* Set the partial sum to 0 */
  method init();
  partial_sum = 0;
  end;

  /* Start processing each row */
  method run();
  set largeNumberSet;
  if (missing(row) or null(row)) then
    do;
    msg = cat('Null value in row ', _n_, ' thread ', _threadid_);
    setThreadErrorStatus(4, msg);
    end;
  if (getThreadErrorStatus() = 0) then partial_sum = partial_sum + row;
  end;

  method term();
  output;
  end;
endthread;

data _null_;
  dcl bigint summation;
  dcl thread processDataset t;
  retain summation;
/* Initialize summation to 0 */
method init();
summation = 0;
end;
method run();
set from t threads=2;
summation = summation + partial_sum;
end;
method term();
put summation=;
end;
enddata;
run;
quit;
```

ERROR メッセージのペアのかわりに、SETTHREADERRORSTATUS 関数でエラーメッセージテキストを定義すると、カスタムエラー条件を満たす行ごとに 1 つの ERROR メッセージがログに書き込まれます。

```
... [ snip] ...
summation=49950000
NOTE: Created thread processdataset in data set work.processdataset.
ERROR: Line 113: Null value in row 1000 thread 1
ERROR: Line 113: Null value in row 808 thread 2
ERROR: Line 113: Null value in row 2000 thread 1
ERROR: Line 113: Null value in row 1808 thread 2
ERROR: Line 113: Null value in row 3000 thread 1
ERROR: Line 113: Null value in row 4000 thread 1
ERROR: Line 113: Null value in row 5000 thread 1
ERROR: Line 113: Null value in row 6000 thread 1
ERROR: Line 113: Null value in row 7000 thread 1
ERROR: Line 113: Null value in row 8000 thread 1
```

SHA256 関数

指定した文字列のメッセージダイジェストの結果を返します。

カテゴリ: 文字

返されるデータの種類: BINARY

構文

SHA256('*string*')

必須引数

string
文字定数、変数または式を指定します。

データの種類 BINARY、CHAR

ヒント 文字のリテラル文字列は単一引用符で囲みます。

スカラー文字変数引数の場合、変数が SHA256 関数に渡される前に、DECLARE ステートメントで指定された初期文字セットエンコーディングを使用して変数がトランスコードされます。バイナリ引数の場合、バイナリ値は文字列として扱われ、値が SHA256 関数に渡される前にセッションエンコーディングを使用してトランスコードされます。

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

SHA256 関数は、SHA256 アルゴリズムに基づいてメッセージ文字列を 256 ビットのハッシュ値に変換します。

SHA256 関数では、出力形式は適用されません。$BINARY*w*. 出力形式または$HEX*w*. 出力形式を使用して、読み取り可能な結果を表示します。

SHA256 関数を使用して、データセット内の変更を追跡できます。SHA256 関数は、テーブルレコードの列の値セットのダイジェストを生成できます。このダイジェストをレコードの署名として扱い、レコードに加えられる変更を追跡するために使用できます。新しいレコードからのダイジェストが、テーブルレコードの既存のダイジェストと一致する場合、2 つのレコードは同一です。ダイジェストが異なる場合、レコードの列の値が変更されています。新たに変更されたレコードは、既存のキーのある値への変更を表す新しい代理キーとともにテーブルに追加されます。

SHA256 関数は、ソフトウェアのインストール、ファイル比較、ファイル破損および改ざん検出のためのシェルスクリプトを開発するときに便利な場合があります。

例: SHA256 関数を使用した結果の生成

このプログラムでは、SHA256 関数によって返される結果を生成する方法を示します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl binary(64) y z;
  method init();
    y=sha256('abc');
    z=sha256('access method');
    put y=$HEX64.;
    put z=$HEX64.;
  end;
enddata;
run;
quit;
```

ASCII システムの場合、SAS は次の出力をログに書き込みます。

```
y=BA7816BF8F01CFEA414140DE5DAE2223B00361A396177A9CB410FF61F20015AD
z=F2758E91725621F59F2F80D15DE8824560EDC471EBE40A83BA6D1259B1605915
```

SHA256HEX 関数

指定した文字列のメッセージダイジェストの結果を返し、文字列を 16 進表現に変換します。

カテゴリ: 文字

返されるデータの種類: CHAR、NCHAR、NVARCHAR、VARCHAR

注: SHA256HEX 関数は、メッセージのデータ統合性と認証を確認します。
エンコーディング間の一貫性を確保するために、SHA256HEX 関数の引数には UTF-8 テキストが推奨されます。

構文

SHA256HEX('*string*', *string_indicator*);

必須引数

string
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

string_indicator
引数 *string* が通常の文字であるか 16 進表現文字であるかを示します。0 は、引数 *string* の表現が通常の文字であることを示します。引数 *string* の表現が 16 進表現の文字であることを示します。

引数 *string* が通常の文字であるか 16 進表現文字であるかを示します。

0
引数 *string* の表現は通常の文字です。

1
引数 *string* の表現は 16 進表現の文字です。

注: 偶数の 16 進表現の文字が含まれ、すべての文字が 0–9、a–f、A–F の範囲内に存在する必要があります。16 進表現文字列の空白は無視されます。

データの種類 DOUBLE

詳細

基本

SHA256HEX 関数は、SHA256 アルゴリズムに基づいて文字列を 256 ビットのハッシュ値に変換します。次に、このデータを 16 進表現形式に変換します。

SHA256HEX 関数を使用する

SHA256HEX 関数を使用して、データセット内の変更を追跡できます。SHA256HEX 関数は、テーブルレコードの列の値セットのダイジェストを生成できます。このダイジェストをレコードの署名として扱い、レコードに加えられる変更を追跡するために使用できます。新しいレコードからのダイジェストが、テーブルレコードの既存のダイジェストと一致する場合、2 つのレコードは同一です。ダイジェストが異なる場合、レコードの列の値が変更されています。新たに変更されたレコードは、既存のキーのある値への変更を表す新しい代理キーとともにテーブルに追加されます。

SHA256HEX 関数は、ソフトウェアのインストール、ファイル比較、ファイル破損および改ざん検出のためのシェルスクリプトまたは Perl プログラムを開発するときに便利な場合があります。

比較

SHA256 関数はその出力をフォーマットしないため、$BINARYw.出力形式または $HEXw.出力形式を使用して読み取り可能な結果を表示する必要があります。SHA256HEX 関数はその出力をフォーマットするため、$BINARY 出力形式や$HEX 出力形式を使用する必要はありません。

例: SHA256HEX 関数を使用した結果の生成

この例では、SHA256HEX 関数によって返される結果を生成する方法を示します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl char(64) y z;
  method run();
    y=sha256hex('abc');
    z=sha256hex('access method');
    put y=;
    put z=;
  end;
enddata;
run;
quit;
```

ASCII システムの場合、SAS は次の出力をログに書き込みます。

```
y=BA7816BF8F01CFEA414140DE5DAE2223B00361A396177A9CB410FF61F20015AD
z=F2758E91725621F59F2F80D15DE8824560EDC471EBE40A83BA6D1259B1605915
```

関連項目

関数:

- “SHA256HMACHEX 関数” (1135 ページ)

SHA256HMACHEX 関数

HMAC (Hash-based Message Authentication)アルゴリズムを使用して、指定した文字列のメッセージダイジェストの結果を返します。

カテゴリ: 文字

返されるデータの種類: CHAR、NCHAR、NVARCHAR、VARCHAR

注: SHA256HMACHEX 関数は、メッセージのデータ統合性と認証を確認します。

ハッシュベースのメッセージ認証コード(HMAC)の詳細については、次の記事を参照してください。

エンコーディング間の一貫性を確保するために、SHA256HMACHEX 関数の引数には UTF-8 テキストが推奨されます。

構文

SHA256HMACHEX('*key*', '*message*' [*string-indicator*]);

必須引数

key
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

message
秘密キーを指定します。ハッシュ関数の入力ブロックサイズに合うように、右側にゼロを追加して埋め込みます。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

オプション引数

string_indicator
キーとメッセージが 16 進表現であるかを指定します。0 は、引数 *key* と引数 *message* が 16 進表現で表現されていないことを示します。1 は、引数 *message* が 16 進表現で表現されていることを示します。2 は、引数 *key* が 16 進表現で表

現されていることを示します。3 は、引数 *key* と引数 *message* が 16 進表現で表現されていることを示します。

キーとメッセージが 16 進表現であるかを指定します。

0
引数 *key* と引数 *message* は 16 進表現で表現されていません。

1
引数 *message* は 16 進表現で表現されています。

2
引数 *key* は 16 進表現で表現されています。

3
引数 *key* と引数 *message* は 16 進表現で表現されています。

注: この引数は、SHA256HMACHEX 関数を繰り返し呼び出し、結果を次の呼び出しでキーとして使用する際に役立ちます。次のコードはこの機能を示します。

```
length digest $64;
digest = sha256hmachex('mykey', 'mymessage', 0);
digest = sha256hmachex(digest, 'my new message', 2);
```

データの種類　DOUBLE

詳細

SHA256HMACHEX 関数は、SHA256 アルゴリズムに基づいて文字列を 256 ビットのハッシュ値に変換します。

ハッシュベースのメッセージ認証コード(HMAC)の詳細については、次の記事を参照してください。

例: SHA256HMACHEX 関数を使用した結果の生成

この例では、SHA256HMACHEX 関数によって返される結果を生成する方法を示します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl char(200) digest;
  method run();
    digest = SHA256HMACHEX('key',
    'The quick brown fox jumps over the lazy dog', 0);
    if digest=
```

```
        upcase('f7bc83f430538424b13298e6aa6fb143ef4d59a14946175997479dbc2d1a3cd8')
        then
          put 'matched';
        else
          put 'not matched';
    end;
enddata;
run;
quit;
```

```
matched
```

SIGN 関数

数値式の符号を示す数字を返します。

カテゴリ: 数学

返されるデータの種類: DOUBLE

注: 2025.12 以降、この関数では、引数が DOUBLE の場合、すべての引数が DOUBLE に変換され、結果は DOUBLE になります。PROC DS2 ステートメントで BIGINTPROCESSING=YES が設定され、いずれかの引数が BIGINT の場合、すべての引数が BIGINT に変換され、結果は BIGINT になります。

構文

SIGN(*expression*)

必須引数

expression
数値に評価される有効な式を指定します。

データの種類 すべての数値型

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

詳細

SIGN 関数は次の値を返します。

-1
expression < 0 の場合

0
expression = 0 の場合

1
expression > 0 の場合

例

次のプログラムは、SIGN 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double x y z;
  method run();
    x=sign(-5);
    y=sign(5);
    z=sign(0);
    put x= y= z=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
x=-1 y=1 z=0
```

SIN 関数

三角関数の正弦を返します。

カテゴリ:	三角関数
返されるデータの種類:	DOUBLE

構文

SIN(*expression*)

必須引数

expression
数値に評価される有効な式を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

例

次のプログラムは、SIN 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double x y z;
  method run();
    x=sin(0.5);
    y=sin(0);
    z=sin(3.14159/4);
  put x= y= z=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
x=0.4794255386042 y=0 z=0.70710631209355
```

SINH 関数

双曲線正弦を返します。

カテゴリ: 双曲線

返されるデータの種類: DOUBLE

構文

SINH(*expression*)

必須引数

expression
数値に評価される有効な式を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

SINH 関数は、次の式によって得られる引数の双曲線正弦を返します。

$$e^{argument} - e^{-argument}/2$$

例

次のプログラムは、SINH 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double x y z;
  method run();
    x=sinh(0);
    y=sinh(1);
    z=sinh(-1);
    put x= y= z=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
x=0 y=1.1752011936438 z=-1.1752011936438
```

SKEWNESS 関数

歪度を返します。

カテゴリ: 記述統計

返されるデータの種類: DOUBLE

構文

SKEWNESS(*expression-1, expression-2, expression-3* [, ...*expression-n*])

必須引数

expression
数値に評価される有効な式を指定します。

要件 少なくとも 3 つの非 null または非欠損の引数が必要です。それがない場合は、関数から null または欠損値が返されます。

データの種類 DOUBLE

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

詳細

すべての非 null または非欠損引数の値が等しい場合、歪度は数学的に定義されておらず、SKEWNESS 関数は null または欠損値を返します。

例

次のプログラムは、SKEWNESS 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double x1 x2 x3 x4;
  method run();
    x1=skewness(0, 1, 1);
    x2=skewness(2, 4, 6, 3, 1);
    x3=skewness(2, 0, 0);
    x4=skewness(of x1-x3);
    put x1=;
    put x2=;
    put x3=;
    put x4=;
  end;
enddata;
run;
```

```
quit;
```

SAS は次の出力をログに書き込みます。

```
x1=-1.73205080756887
x2=0.59012865638436
x3=1.73205080756887
x4=-0.9530977136649
```

SLEEP 関数

指定した期間、この関数を呼び出すプログラムの実行を中断します。

カテゴリ:	特殊
返されるデータの種類:	DOUBLE

構文

SLEEP(*number-of-time-units* [, *time-unit*])

必須引数

number-of-time-units
数値に評価され、プログラムの実行を中断する時間の単位数を示す有効な式を指定します。

範囲	$n \geq 0$
制限事項	負の値は無効です。
データの種類	DOUBLE
ヒント	*n* 引数に小数を使用する場合、1 秒の何分の 1 かの間実行を中断するときは、*unit* 引数が必要です。たとえば、SLEEP(.25);では中断されません。SLEEP(1.25);では、1 秒間中断されます。SLEEP(1.25, 1);では、1.25 秒間中断されます。SLEEP(.25,1)では、0.25 秒間中断されます。

オプション引数

time-unit
秒単位の時間単位が 10 の倍数であることを指定し、これが *number-of-time-units* に適用されます。たとえば、1 は 1 秒、0.001 はミリ秒、5 は 5 秒に相当します。

デフォルト	.001

データの種類 DOUBLE

例 このコードは、プログラムの実行を 10 秒間中断します。

```
x=sleep(10,1);
```

詳細

SLEEP 関数は、この関数を呼び出したプログラムの実行を指定した期間中断します。

SLEEP 関数は、プログラムの実行を一時停止し、最初の引数の値を返します。

SAS Studio などの GUI 環境を使用している場合、SLEEP 関数はデフォルトの *time-unit* 値を使用します。SAS がスリープ状態になる時間を示すウィンドウが表示されます。

SLEEP 関数を呼び出すプログラムをサブミットする場合、**SLEEP** ウィンドウが表示され、SAS のスリープ解除時間が示されます。スリープ期間が終わるまでは、SAS セッションは非アクティブの状態が維持されます。SLEEP 関数の呼び出しをキャンセルするには、Ctrl+Break アテンションシーケンスを使用します。

null のデータプログラムを使用して SLEEP 関数を呼び出し、このデータプログラムの後に残りの SAS プログラムを続けます。このように SLEEP 関数を使用すると、Ctrl+Break アテンションシーケンスを使用して SLEEP 関数を中断して、残りの SAS プログラムの実行を継続できます。

例

例 1: 指定した期間の実行中断

次のプログラムは、実行を 20 秒遅らせます。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data payroll;
  method init();
  time_slept=sleep(20,1);
 /*...more DS2 statements...*/
  end;
enddata;
run;
quit;
```

例 2: スリープ時間の計算に基づいた実行中断

次に、2011 年 6 月 15 日の午前 0 時まで実行を中断するよう SAS に指示するプログラムを示します。DS2 では、対象日とコードの実行開始日時に基づいて中断の長さが計算されます。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data budget;
  method init();
  sleeptime= dhms(mdy(06,15,2011),00,00,02) - dhms(mdy(06,15,2011),00,00,00);
  time_calc=sleep(sleeptime,1);
  /*...more DS2 statements...*/
  end;
enddata;
run;
quit;
```

SMALLEST 関数

k 番目に小さい非 null 値または非欠損値を返します。

カテゴリ: 記述統計

返されるデータの種類: DOUBLE

注: 2025.12 以降、この関数では、引数が DOUBLE の場合、すべての引数が DOUBLE に変換され、結果は DOUBLE になります。PROC DS2 ステートメントで BIGINTPROCESSING=YES が設定され、いずれかの引数が BIGINT の場合、すべての引数が BIGINT に変換され、結果は BIGINT になります。

構文

SMALLEST(*k*, *expression* [, ...*expression*])

必須引数

k
数値に評価されて返される有効な式を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

expression
数値に評価されて処理される有効な式を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

k が null または欠損値、0 未満、または値の数よりも大きい場合、結果は null または欠損値になります。

比較

SMALLEST 関数と ORDINAL 関数の違いは、SMALLEST は null 値と欠損値を無視しますが、ORDINAL は null 値と欠損値を数える点です。

例

この例では、SMALLEST 関数によって返される値と ORDINAL 関数によって返される値を比較します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data comparison;
  dcl double smallest_num having label 'SMALLEST Function';
  dcl double ordinal_num having label 'ORDINAL Function';
  dcl double k;
  method run();
    do k = 1 to 4;
      smallest_num = smallest(k, 456, 789, .Q, 123);
      ordinal_num  = ordinal (k, 456, 789, .Q, 123);
      output;
    end;
  end;
enddata;
run;
quit;
proc print data=comparison label;
  var k smallest_num ordinal_num;
  title 'Results From the SMALLEST and the ORDINAL Functions';
run;
```

アウトプット 7.28　値の比較: SMALLEST 関数と ORDINAL 関数

Results From the SMALLEST and the ORDINAL Functions

Obs	K	SMALLEST Function	ORDINAL Function
1	1	123	Q
2	2	456	123
3	3	789	456
4	4	.	789

関連項目

関数:

- “LARGEST 関数” (799 ページ)
- “ORDINAL 関数” (914 ページ)
- “PCTL 関数” (916 ページ)

SQLEXEC 関数

FedSQL ステートメントを実行して、テーブルを作成、削除、または更新するか、テーブルに行を挿入します。

カテゴリ: 特殊

制限事項: この関数は CAS サーバーではサポートされていません。

構文

SQLEXEC('*sql-text*')

必須引数

'*sql-text*'
テーブルの行の挿入、更新、作成、削除を行う有効な FedSQL ステートメントです。

注意

SQLEXEC 関数で使用できるのは FedSQL ステートメントのみです。DBMS 固有の SQL は使用できません。 詳細については、*SAS FedSQL 言語リファレンス*を参照してください。

要件 FedSQL ステートメントは、単一引用符(')で囲む必要があります。

注 ステートメントは、文字列リテラル、式から生成された文字列値、または変数に保存されている文字列値にすることができます。

詳細

SQLEXEC 関数には次の項目が適用されます。

- FedSQL ステートメントを 1 回だけ実行する場合は、SQLEXEC 関数を使用します。
- 割り当て、準備、実行、解放は実行時に実行されます。
- SQLEXEC 関数はパラメーターをサポートしていません。
- SQLEXEC 関数は、結果セットの戻りをサポートしていません。SELECT ステートメントでは使用できません。
- SQLEXEC は、SQL EXECUTE IMMEDIATE ステートメントまたは JDBC Statement.executeUpdate (*string*)メソッドに似ています。

SQLSTMT パッケージを使用すると、FedSQL ステートメントを複数回実行したり、パラメーターを使用したり、結果セットにアクセスしたりできます。詳細については、"SQLSTMT パッケージの使用" (*SAS DS2 プログラマガイド*)を参照してください。

例: SQLEXEC 関数を使用して空のテーブルを作成する

次のプログラムは、5 つのテーブルを作成します。各テーブルには 3 つの列があり、行はありません。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  declare int i;
  declare int rc;
  method init();
    do i = 1 to 5;
     rc = sqlexec('create table testdata' || i ||
       '(x double, y double, z double)');
     if (rc ne 0) then put 'TEST FAILED';
  end;
end;
```

```
enddata;
run;
quit;
```

関連項目

“SQLSTMT パッケージと SQLEXEC 関数の比較” (*SAS DS2 プログラマガイド*)

SQRT 関数

値の平方根を返します。

カテゴリ: 数学

返されるデータの種類: DOUBLE

構文

SQRT(*expression*)

必須引数

expression
負でない数値に評価される有効な式を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

例

次のプログラムは、SQRT 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double a b c;
  method run();
    a=sqrt(36);
    b=sqrt(25);
    c=sqrt(4.4);
```

```
    put a=;
    put b=;
    put c=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
a=6
b=5
c=2.0976176963403
```

SQUANTILE 関数

右側確率(SDF)の指定した場合に分布から分位点を返します。

カテゴリ:	分位点
返されるデータの種類:	DOUBLE
参照項目:	“SDF 関数” (1120 ページ)

構文

SQUANTILE(*'distribution'*, *probability*, *parameter-1, ..., parameter-k*)

必須引数

distribution
分布を特定する文字定数、変数または式です。

注: 各 SQUANTILE 分布関数の引数は、対応する CDF 分布関数の引数と同じです。

有効な分布は次のとおりです。

分布	引数
Bernoulli	'BERNOULLI'
ベータ	'BETA'
二項	'BINOMIAL'

分布	引数
Cauchy	'CAUCHY'
カイ 2 乗	'CHISQUARE'
Conway-Maxwell-Poisson	'CONMAXPOI'
指数	'EXPONENTIAL'
F	'F'
ガンマ	'GAMMA'
一般化 Poisson	'GENPOISSON'
幾何	'GEOMETRIC'
超幾何	'HYPERGEOMETRIC'
Laplace	'LAPLACE'
ロジスティック	'LOGISTIC'
対数正規	'LOGNORMAL'
負の二項	'NEGBINOMIAL'
正規	'NORMAL'|'GAUSS'
正規混合	'NORMALMIX'
パレート	'PARETO'
Poisson	'POISSON'
T	'T'
Tweedie	'TWEEDIE'
一様	'UNIFORM'
Wald (逆ガウス)	'WALD'|'IGAUSS'
Weibull	'WEIBULL'

注: T、F および NORMALMIX を除き、最初の 4 文字で分布を最小限に識別できます。

probability
確率変数の値を指定する、数値定数、変数または式です。

データの種類 DOUBLE

parameter-1, ..., parameter-k
特定の分布に対応する *shape*、*location* または *scale* パラメーターです(省略可能)。

データの種類 DOUBLE

詳細

SQUANTILE 関数は、指定された確率値に基づいて、指定された連続分布または離散分布から分位点を計算します。詳細については、前述の表に記載されている個々の分布を参照してください。

SQUANTILE 関数の Conway-Maxwell-Poisson 分布は、SDF 値が *p* 未満の最小の整数であるカウント値 *y* を返します。SQUANTILE 関数の Conway-Maxwell-Poisson 分布の構文は、次の形式です。

SQUANTILE('CONMAXPOI', *p*, *λ*, *ν*)

p
0 と 1 の間の実数(両端の値を含める)。

λ
Poisson 分布の場合と同様で、平均に似ています。

ν
ばらつきのパラメーターです。

詳細については、PDF 関数の“Conway-Maxwell-Poisson” (930 ページ)を参照してください。

例

例 1: LOGISTIC 分布の使用

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double p sdf;
  dcl varchar(10) dist;
```

```
    method init();
      dist='logistic';
      sdf=squantile(dist, 1.e-20);
      put sdf=;
      p=sdf(dist, sdf);
      put p= /* p will be 1.e-20 */;
    end;
enddata;
run;
quit;
```

SAS は次の結果をログに書き込みます。

```
sdf=46.0517018598809
p=1E-20
```

例 2: Conway-Maxwell-Poisson 分布の使用

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double y;
  method init();
    y=squantile('conmaxpoi',.2,2.3,.4);
    put y=;
  end;
enddata;
run;
quit;
```

SAS は次の結果をログに書き込みます。

```
y=12
```

関連項目

関数:

- “CDF 関数” (359 ページ)
- “LOGCDF 関数” (825 ページ)
- “LOGPDF 関数” (828 ページ)
- “LOGSDF 関数” (831 ページ)
- “PDF 関数” (918 ページ)
- “SDF 関数” (1120 ページ)
- “QUANTILE 関数” (1052 ページ)

STD 関数

標準偏差を返します。

カテゴリ: 記述統計

返されるデータの種類: DOUBLE

構文

STD(*expression-1, expression-2* [, ...*expression-n*])

必須引数

expression
数値に評価される有効な式を指定します。

要件 少なくとも 2 つの非 null または非欠損の引数が必要です。それがない場合は、関数から null または欠損値が返されます。

データの種類 DOUBLE

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

例

次のプログラムは、STD 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double val1 val2 val3 val4;
  method run();
    val1=2;
    val2=4;
    val3=6;
    val4=8;
    output;
  end;
enddata;
run;
```

```
data test2(overwrite=yes);
  dcl double x1 x2 x3 x4;
  method init();
    set test;
    x1=std(val1,val3);
    x2=std(val1,val3, .);
    x3=std(val1,val2,val3, 3, 1);
    x4=std(of x1-x3);
    put x1= x2= x3= x4=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
x1=2.82842712474619 x2=2.82842712474619 x3=1.92353840616713 x4=0.52243774525827
```

STDERR 関数

平均の標準誤差を返します。

カテゴリ:	記述統計
返されるデータの種類:	DOUBLE

構文

STDERR(*expression-1, expression-2* [, ...*expression-n*])

必須引数

expression
数値に評価される有効な式を指定します。

要件	少なくとも 2 つの非 null または非欠損の引数が必要です。それがない場合は、関数から null または欠損値が返されます。
データの種類	DOUBLE
参照項目:	“DS2 式” (*SAS DS2 プログラマガイド*)

例

次のプログラムは、STDERR 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double val1 val2 val3 val4;
  method run();
    val1=2;
    val2=4;
    val3=6;
    val4=8;
    output;
  end;
enddata;
run;


data test2(overwrite=yes);
  dcl double x1 x2 x3 x4;
  method init();
    set test;
    x1=stderr(val1,val3);
    x2=stderr(val1,val3, .);
    x3=stderr(val1,val2,val3, 3, 1);
    x4=stderr(of x1-x3);
    put x1=;
    put x2=;
    put x3=;
    put x4=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
x1=2
x2=2
x3=0.86023252670426
x4=0.37992249109857
```

STFIPS 関数

州郵便番号を FIPS 州コードに変換します。

カテゴリ:	州と ZIP コード
制限事項:	この関数は CAS サーバーではサポートされていません。
返されるデータの種類:	DOUBLE

構文

STFIPS(*postal-code*)

必須引数

postal-code
2 文字の標準州郵便番号を含む文字式を指定します。文字は大文字と小文字を混在させることができます。関数は末尾の空白を無視しますが、式に先頭の空白が含まれている場合はエラーが生成されます。

データの種類 CHAR()

詳細

STFIPS 関数は、2 文字の州郵便番号(または米国領の場合は世界の GSA 地理コード)を、対応する数値の FIPS (U.S. Federal Information Processing Standards)コードに変換します。

比較

STFIPS、STNAME、および STNAMEL 関数は同じ引数を取りますが、異なる値を返します。STFIPS は、数値の FIPS (U.S. Federal Information Processing Standards)コードを返します。STNAME は大文字の州名を返します。STNAMEL は、大文字と小文字が混在する州名を返します。

例

次の例は、ノースカロライナ州の 2 桁の郵便番号で STFIPS、STNAME、および STNAMEL を使用した場合の違いを示しています。

```
proc ds2;
 data _null_;
  method run();
  dcl double fips;
  dcl char(15) state;
  fips=stfips('NC');
  put fips=;
  state=stname('NC');
  put state=;
  state=stnamel('NC');
  put state=;
  end;
 enddata;
run;
quit;
```

これらのステートメントは、次の結果を生成します。

```
fips=37
state=NORTH CAROLINA
state=North Carolina
```

関連項目

関数:

- “FIPNAME 関数” (648 ページ)
- “FIPNAMEL 関数” (650 ページ)
- “FIPSTATE 関数” (651 ページ)
- “STNAME 関数” (1157 ページ)
- “STNAMEL 関数” (1158 ページ)

STNAME 関数

州郵便番号を大文字の州名に変換します。

カテゴリ:	州と ZIP コード
制限事項:	この関数は CAS サーバーではサポートされていません。
返されるデータの種類:	CHAR

構文

STNAME(*postal-code*)

必須引数

postal-code
2 文字の標準州郵便番号を含む文字式を指定します。文字は大文字と小文字を混在させることができます。関数は末尾の空白を無視しますが、式に先頭の空白が含まれている場合はエラーが生成されます。

詳細

STNAME 関数は、2 文字の州郵便番号(または米国領の場合は世界の GSA 地理コード)を大文字の対応する州名に変換します。

比較

STNAME 関数と STNAMEL 関数はどちらも、引数として 2 桁の郵便番号を取り、州名を返します。STNAME は大文字の州名を返します。STNAMEL は、大文字と小文字が混在する州名を返します。

例

この例では、ノースカロライナ州の 2 桁の郵便番号を大文字の州名に変換します。

```
proc ds2;
 data _null_;
  method run();
  dcl char(15) state;
  state=stname('NC');
  put state=;
  end;
 enddata;
run;
quit;
```

これらのステートメントは、次の結果を生成します。

```
state=NORTH CAROLINA
```

関連項目

関数:

- “FIPNAME 関数” (648 ページ)
- “FIPNAMEL 関数” (650 ページ)
- “FIPSTATE 関数” (651 ページ)
- “STNAMEL 関数” (1158 ページ)

STNAMEL 関数

州郵便番号を大文字と小文字が混在する州名に変換します。

カテゴリ:	州と ZIP コード
制限事項:	この関数は CAS サーバーではサポートされていません。
返されるデータの種類:	CHAR

構文

STNAMEL(*postal-code*)

必須引数

postal-code
2 文字の標準州郵便番号を含む文字式を指定します。文字は大文字と小文字を混在させることができます。関数は末尾の空白を無視しますが、式に先頭の空白が含まれている場合はエラーが生成されます。

詳細

まだ長さが割り当てられていない変数に STNAMEL 関数から値が返される場合、その変数にはデフォルトで長さ 20 が割り当てられます。

STNAMEL 関数は、2 文字の州郵便番号(または米国領の場合は世界の GSA 地理コード)を大文字と小文字が混在する対応する州名に変換します。

例

この例では、ノースカロライナ州の 2 桁の郵便番号を、大文字と小文字が混在する州名に変換します。

```
proc ds2;
 data _null_;
  method run();
  dcl char(15) state;
  state=stnamel('NC');
  put state=;
  end;
 enddata;
run;
quit;
```

これらのステートメントは、次の結果を生成します。

```
state=North Carolina
```

関連項目

関数:

- “FIPNAME 関数” (648 ページ)
- “FIPNAMEL 関数” (650 ページ)
- “FIPSTATE 関数” (651 ページ)

- “STFIPS 関数” (1155 ページ)
- “STNAME 関数” (1157 ページ)

STREAMINIT 関数

乱数を生成するための乱数ジェネレーター-とシード値を指定します。

カテゴリ: 乱数

返されるデータの種類: DOUBLE

注: 2025.12 以降、この関数では、引数が DOUBLE の場合、すべての引数が DOUBLE に変換され、結果は DOUBLE になります。PROC DS2 ステートメントで BIGINTPROCESSING=YES が設定され、いずれかの引数が BIGINT の場合、すべての引数が BIGINT に変換され、結果は BIGINT になります。

構文

形式 1: **STREAMINIT**([*seed*])

形式 2: **STREAMINIT**([*RNG*]

形式 3: **STREAMINIT**([*RNG*, *seed*]

オプション引数

seed
疑似乱数のストリームの初期化に使用される値を指定する数式。

範囲 シード値の範囲は RNG によって異なります。

データの種類 DOUBLE

注 スレッドごとに、STREAMINIT 関数の最初の呼び出しでシード値を使用してすべてのストリームが 初期化されます。同じスレッドからの STREAMINIT 関数への後続の呼び出しは無視されます。

RNG
RAND 関数への後続の呼び出しで乱数または疑似乱数を生成するための乱数ジェネレーター(RNG)を指定する文字式。この文字式は、大文字と小文字を区別しません。

次のジェネレーターがサポートされています。

RNG	説明
MTHybrid	ハイブリッド 1998/2002 32 ビット Mersenne Twister。デフォルトの方法。
MT1998	1998 32 ビット Mersenne Twister。非推奨。
MT32 | MT2002	2002 32 ビット Mersenne Twister。
MT64	64 ビット Mersenne Twister。
PCG | PCG64i	64 ビット PCG (permuted congruential generator)。
TF2 | THREEFRY2x64	Random123 ライブラリの Threefish 暗号化関数に基づく Threefry 2x64 ビットカウンターベースの RNG。
TF4 | THREEFRY4x64	Random123 ライブラリの Threefry メソッドに基づく Threefry 4x64 ビットカウンターベースの RNG。
HARDWARE	サポートされているハードウェアベースの乱数ジェネレーター。
RDRAND	Intel ハードウェアベース RdRand 命令。

乱数ジェネレーターの詳細については、“DATA ステップで乱数関数と乱数 CALL ルーチンの使用” (*SAS 関数と CALL ルーチン: リファレンス*)を参照してください。

制限事項 HARDWARE RNG は、RdRand 命令をサポートする Intel プロセッサ (Ivy Bridge とその後続プロセッサ)でのみ使用できます。

ヒント スレッドごとに、STREAMINIT 関数の最初の呼び出しでシード値を使用してすべてのストリームが初期化されます。同じスレッドからの STREAMINIT 関数への後続の呼び出しは無視されます。

RNG を指定しない場合、RAND 関数は MTHYBRID RNG を使用して(疑似)乱数のストリームを計算します。

PCG RNG は、統計的品質、速度、サイクル長、および複数の独立したストリームの優れた組み合わせを提供します。

詳細

基本

RNG によって生成されるランダムまたは疑似ランダム値のシーケンスは、*ストリーム*と呼ばれます。STREAMINIT 関数は、正のシードを使用して *RNG* を初期化します。各スレッド内で、STREAMINIT 関数への最初の呼び出しで *RNG* が初期化されます。同じスレッドでの後続の呼び出しでは、ストリームは再初期化されません。*RNG* は、DS2 プログラムの残りの部分では各スレッドで有効なままです。

RNG は、一様分布からランダムな値を生成します。RAND 関数は、*RNG* からの 1 つ以上の一様変量を使用して、確率分布から値を作成します。RAND 関数への各呼び出しは、ストリームからの少なくとも 1 つの値を使用します。この呼び出しにより、*RNG* の内部状態が更新されます。

疑似乱数値の再現可能なストリームを生成するには、正のシード値を使用して STREAMINIT 関数を呼び出し、任意の疑似乱数 *RNG* を指定します。RAND 関数の前に STREAMINIT 関数が呼び出された場合、結果として得られる乱数のストリームは再現可能です。DS2 プログラムを実行するたびに、同じストリームが生成されます。

ハードウェアベースの *RNG* を指定した場合、シード値は無視されます。ハードウェア *RNG* からの乱数は再現できません。ハードウェア *RNG* を複数回使用する SAS プログラムを実行すると、プログラムを実行するたびに異なる乱数が取得されます。

疑似乱数ジェネレーターを使用してこれらのルーチンまたは関数のいずれかを呼び出すと、SAS はデフォルトのシード値を計算します。

- 欠損しているゼロまたは負の *seed* 引数を指定して STREAMINIT を呼び出します。
- *seed* 引数なしで STREAMINIT 関数を呼び出します。
- 事前に STREAMINIT 関数を呼び出さずに、RAND 関数を呼び出します。

STREAMINIT 関数が 1 つまたは複数のストリームを初期化する場合、シードの値を返します。1 つまたは複数のストリームが STREAMINIT 関数または RAND 関数を呼び出してすでに初期化されている場合、STREAMINIT 関数は 0 を返します。

次の FCMP 関数は、DS2 プログラムから呼び出された場合、DS2 プログラムで設定されたシード値を認識しません。

COMPCOST
COMPGED
RAND
STREAMINIT

DATA ステップでは、RAND からのすべての乱数はデフォルトで同じストリームから取得されますが、CALL STREAM ルーチンを使用して複数の独立したストリームを作成できます。

PROC DS2 では、FCMP パッケージ内で呼び出される RAND の乱数は、別のストリームから取得されます。再現可能な乱数を取得するには、RAND を呼び出す各 FCMP 関数内で CALL STREAMINIT ルーチンを使用します。独立したストリームを取得するには、RAND を呼び出す各 FCMP 関数内で CALL STREAM ルーチンも使用します。そうすると、DS2 からの結果は DATA ステップと一致します。例について

は、"FCMP パッケージを使用する際の考慮事項と制限事項" (*SAS DS2 プログラマガイド*)を参照してください。

シード値の範囲

MTHybrid、MT32、および MT1998 の場合、シード値は 1〜2^{32} –1=4294967295 の整数にすることができます。

MT64、PCG、Threefry2x64、および Threefry4x64 の場合、シード値は 1〜2^{64} –1025=18446744073709549568 の整数にすることができます。一部の大きな整数は、倍精度浮動小数点数で正確に表現されていません。

シード引数が欠損しているか、またはゼロ以下の場合、SAS は"DATA ステップで乱数関数と乱数 CALL ルーチンの使用" (*SAS 関数と CALL ルーチン: リファレンス*)に記載されたようにシード値を生成します。

正のシード値が範囲外の場合、シード引数の仮数は範囲内の整数にスケーリングされます。

例

例 1: RAND 関数を使用して再現可能な疑似乱数のストリームを作成するためのシード値を指定する

次のプログラムは、STREAMINIT 関数でシード値を指定して、RAND 関数で再現可能な疑似乱数のストリームを作成する方法を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data random (overwrite=yes);
  dcl double i x1;
  method run();
    streaminit(123);
    do i=1 to 10;
      x1=rand('cauchy');
      output;
    end;
  end;
enddata;
run;
quit;

proc print data=random;
  id i;
run;
quit;
```

アウトプット 7.29　STREAMINIT 関数でシードされた数値文字列

The SAS System

i	x1
1	-0.17593
2	3.76106
3	1.23427
4	0.49095
5	-0.05094
6	0.72496
7	-0.51646
8	7.61304
9	0.89784
10	1.69348

例 2: 64 ビット Mersenne Twister 乱数ジェネレーターを使用する

次のプログラムは、MT64 RNG でシード値を指定して、RAND 関数で疑似乱数のストリームを作成する方法を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data random (overwrite=yes drop=(MT64));
  dcl double i x1;
  method run();
    streaminit('MT64', 123);
    do i=1 to 10;
      x1=rand('cauchy');
      output;
    end;
  end;
enddata;
run;
quit;

proc print data=random;
  id i;
run;
quit;
```

アウトプット 7.30　MT64 乱数ジェネレーターでシードされた数値文字列

The SAS System

i	x1
1	-3.3370
2	1.8546
3	1.0090
4	-2.1520
5	-27.0630
6	-2.6697
7	-7.6492
8	0.2939
9	3.7349
10	-0.5185

例 3: シードを指定して 64 ビット Mersenne Twister 乱数ジェネレーターを使用する

次のプログラムは、MT64 RNG でシード値を指定して、RAND 関数で疑似乱数のストリームを作成する方法を示しています。

注: このサンプルコードでは、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data random (overwrite=yes drop=(MT64));
  dcl double i x1;
  method run();
    streaminit('MT64', 128645);
    do i=1 to 10;
      x1=rand('cauchy');
      output;
    end;
  end;
enddata;
run;
quit;

proc print data=random;
  id i;
run;
quit;
```

アウトプット 7.31　シード番号と 64 ビット Mersenne Twister 乱数ジェネレータ
ーを含む数値文字列

The SAS System

i	x1
1	0.00658
2	0.55544
3	9.67566
4	0.54930
5	0.44583
6	-2.88661
7	1.90463
8	0.01001
9	-0.01532
10	0.72421

関連項目

関数:

- “RAND 関数” (1058 ページ)

STRIP 関数

先頭と末尾の空白を削除して文字列を返します。

カテゴリ: 文字

返されるデータの種類: CHAR、NCHAR、NVARCHAR、VARCHAR

構文

STRIP(*expression*)

必須引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

STRIP 関数は引数の先頭と末尾から空白をすべて削除して返します。引数が空白の場合、STRIP は長さがゼロの文字列を返します。

調整された値が受け取る変数の長さより短い場合、値の末尾に新しい空白が埋め込まれます。

注: 連結演算子では末尾の空白が削除されないため、STRIP 関数は連結を行う場合に便利です。

比較

STRIP 関数と TRIMN 関数の比較を次のリストに示します。

- 空白の文字列の場合、STRIP 関数と TRIM 関数はどちらも長さがゼロの文字列を返します。
- 先頭に空白がない文字列では、STRIP 関数と TRIM 関数は同一値を返します。

例

次のプログラムでは、STRIP 関数を使用して先頭と末尾の空白を削除します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data lengthn (overwrite=yes);
  dcl char(8) string;
  method init();
    string='abcd    '; output;
    string='  abcd  '; output;
    string='    abcd'; output;
    string='abcdefgh'; output;
    string='x y x'; output;
  end;
enddata;
run;

data stripstring (overwrite=yes);
  dcl varchar(10) original;
```

```
  dcl varchar(10)  stripped;
  method run();
    set lengthn;
    original = '*' || string || '*';
    stripped = '*' || strip(string) || '*';
  end;
enddata;
run;
quit;

proc print data=stripstring;
run;
```

アウトプット 7.32　STRIP 関数の結果

The SAS System

Obs	ORIGINAL	STRIPPED	STRING
1	*abcd *	*abcd*	abcd
2	* abcd *	*abcd*	abcd
3	* abcd*	*abcd*	abcd
4	*abcdefgh*	*abcdefgh*	abcdefgh
5	*x y x *	*x y x*	x y x

関連項目

関数:

- “CAT 関数” (344 ページ)
- “CATS 関数” (351 ページ)
- “CATT 関数” (353 ページ)
- “CATX 関数” (356 ページ)
- “LEFT 関数” (806 ページ)
- “TRIM 関数” (1217 ページ)

擬似 SUBSTR(割り当ての左辺に用いた場合) 関数

文字変数のコンテンツを置き換えます。

カテゴリ: 文字

構文

SUBSTR (*character-variable*, *position-expression* [, *length-expression*]) = *characters-to-replace*

必須引数

character-variable
文字変数を指定します。

制限事項 *character-variable* は、変数または配列参照でなければなりません。式またはリテラルにすることはできません。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

position-expression
置換される *character-variable* のコンテンツの最初の文字位置を指定します。

要件 *position-expression* は、1 以上である必要があります。

データの種類 BIGINT。他の型は BIGINIT に強制変換されます。

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

characters-to-replace
character-variable のコンテンツを置き換える文字定数、変数、または式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

注 文字のリテラル文字列を引用符で囲みます。

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

オプション引数

length-expression
置換される部分文字列の長さを指定します。

制限事項 *length-expression* は、*character-variable* で残されている、*position-expression* 以降の式の長さを超えることはできません。

データの種類 BIGINT。他の型は BIGINIT に強制変換されます。

注 *length-expression* を省略した場合、割り当てステートメントの右側のすべての文字を使用して、*character-variable* の値が置き換えられます。

length-expression は、*characters-to-replace* の実際の長さよりも長くすることができます。

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

注: 次の説明では、長さは、バイトベースの長さではなく、文字ベースの長さを指します。

割り当てステートメントの左側で SUBSTR 関数を使用すると、*character-variable* のコンテンツの部分文字列が等号(=)記号の右側の文字列値に置き換えられます。

SUBSTR は、*position-expression* で指定した文字位置から置換を開始します。SUBSTR は、*length-expression* で指定された文字数を置換文字から *character-variable* に挿入します。

SUBSTR 関数(割り当ての左辺に用いた場合は、次のいずれかに該当する場合、WARNING メッセージを返し、*character-variable* を変更しません。

- *position-expression* が欠損値、null 値、ゼロ未満、または *character-variable* の部分文字列の長さを超えている場合。
- *length-expression* が欠損値、null 値、ゼロ未満、または *character-variable* の部分文字列の長さを超えている場合。
- *length-expression* と *position-expression* を *character-variable* に適用した結果の文字列の長さが、宣言された *character-variable* の長さを超えている場合。

characters-to-replace の長さが *length-expression* よりも長い場合、文字列は、*character-variable* で部分文字列の置換として使用される前に、*length-expression* の文字数に切り捨てられます。*characters-to-replace* の長さが *length-expression* よりも短い場合、*characters-to-replace* は、*character-variable* で部分文字列の置換として使用される前に、*length-expression* の文字数まで空白で埋められます。

宣言されていない文字変数を指定して SUBSTR 関数(割り当ての左辺に用いた場合)を使用すると、予期しない結果が生じる可能性があります。

例

次のプログラムは、SUBSTR 関数(割り当ての左辺に用いた場合)を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
/*** replaces 3 characters starting at position 1 ***/
proc ds2;
data test (overwrite=yes);
  dcl char mystring;
  method run();
```

```
    mystring='kidnap';
    substr(mystring, 1, 3)='cat';
    put mystring=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
mystring=catnap
```

```
/*** When position 1 is specified ***/
/***  and length is omitted, all characters  ***/
/***  are replaced with characters-to-replace ***/
proc ds2;
data test (overwrite=yes);
  dcl char mystring;
  method run();
    mystring='kidnap';
    substr(mystring, 1)='cat';
    put mystring=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
mystring=cat
```

```
/*** When character-variable is undeclared, ***/
/*** length is ommitted, and position is 1, all characters ***/
/***  are replaced with the first letter of characters-to-replace. ***/
proc ds2;
data test (overwrite=yes);
  method run();
    mystring=' ';
    substr(mystring, 1)='cat';
    put mystring=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
c
WARNING: Line 83: No DECLARE for assigned-to variable mystring; creating it as a global variable of type
char(1).
```

```
/*** When position is a missing value,***/
/*** nothing is replaced ***/
proc ds2;
data test (overwrite=yes);
  dcl char mystring;
  method run();
```

```
    mystring='kidnap';
    substr(mystring,.)='cat';
    put mystring=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
mystring=kidnap
WARNING: Line 85: Argument 2 to function SUBSTR is invalid.
```

```
/*** characters-to-replace can be appended ***/
/*** to character-variable as long as the resulting ***/
/*** string does not exceed the variable's declared length ***/
proc ds2;
data test (overwrite=yes);
  dcl varchar(10) mystring;
  method run();
    mystring='cat';
    substr(mystring, 4, 3)='napper';
    put mystring=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
catnap
```

関連項目

関数:

- “SUBSTR (=の右) 関数” (1172 ページ)
- “SUBSTRN 関数” (1175 ページ)

SUBSTR (=の右) 関数

引数から部分文字列を抽出します。

カテゴリ:	文字
返されるデータの種類:	CHAR、NCHAR、NVARCHAR、VARCHAR

構文

SUBSTR(*character-expression*, *position-expression* [, *length-expression*])

必須引数

character-expression
文字定数、変数または式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

position-expression
開始文字位置を示す数値定数、変数または式を指定します。

要件 *position-expression* は、ゼロ以上である必要があります。

データの種類 BIGINT

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

オプション引数

length-expression
抽出する部分文字列の長さを示す数値定数、変数または式を指定します。

操作 *length-expression* を省略した場合、SUBSTR 関数は式の開始文字位置以降を抽出します。*length-expression* がゼロ、負の値または string 内で position より後に残った式の長さよりも長い場合、この関数は式の残りを抽出します。この関数は、また、引数が無効であることを示す WARNING もログに出力します。

データの種類 BIGINT

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

詳細

SUBSTR 関数は、式内の *character-expression* で指定された部分を抽出します。この部分は、*position-expression* で指定した文字位置から始まり、*length-expression* で指定した文字数になります。

比較

DATA ステップと同様の部分文字列抽出動作が必要な場合は、SUBSTR 関数を使用します。入力式が無効な場合、DATA ステップは_ERROR_を 1 に設定し、ログに注を出力します。DS2 は WARNING を返し、ログに注を出力します。

例

次のプログラムは、SUBSTR (=の右)関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
/**** selects all chars starting at position 5 to the end of the string ****/
proc ds2;
data test (overwrite=yes);
  dcl char(12) a b;
  method run();
    a='chsh234960b3';
    b=substr(a,5);
    put b;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
234960b3
```

```
/**** selects 6 chars starting at position 5 ****/
proc ds2;
data test (overwrite=yes);
  dcl char(12) a b;
  method run();
    a='chsh234960b3';
    b=substr(a,5, 6);
    put b;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
234960
```

```
/**** selects all chars starting at position 5 to the end of the string ****/
/**** because of the invalid third argument ****/
proc ds2;
data test (overwrite=yes);
```

```
  dcl char(12) a b;
  method run();
    a='chsh234960b3';
    b=substr(a, 5, 15);
    put b;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
234960b3
WARNING: Line 351: Argument 3 to function SUBSTR is invalid.
```

```
/**** selects all chars starting at position 5 to the end of the string ****/
/**** because of the invalid third argument ****/
proc ds2;
data test (overwrite=yes);
  dcl char(12) a b;
  method run();
    a='chsh234960b3';
    b=substr(a, 5, -5);
    put b;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
234960b3
WARNING: Line 362: Argument 3 to function SUBSTR is invalid.
```

関連項目

関数:

- “擬似 SUBSTR(割り当ての左辺に用いた場合) 関数” (1168 ページ)
- “SUBSTRN 関数” (1175 ページ)

SUBSTRN 関数

部分文字列を返します。長さがゼロの結果も返せます。

カテゴリ:	文字
返されるデータの種類:	CHAR、NCHAR、NVARCHAR、VARCHAR

構文

SUBSTRN(*expression*, *position-expression*[, *length-expression*])

必須引数

expression
文字列に評価されるか強制変換できる有効な式または数値を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR、DOUBLE

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

position-expression
BIGINT に評価され、部分文字列内の最初の文字の位置を指定する任意の有効な式を指定します。

データの種類 BIGINT

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

オプション引数

length-expression
BIGINT に評価され、部分文字列の長さを指定する任意の有効な式を指定します。

データの種類 BIGINT

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

詳細

基本

次の情報は SUBSTRN 関数についての説明です。

- *length-expression* が次の条件のいずれかを満たす場合、SUBSTRN 関数は *position-expression* から *character-expression* の末尾までの部分文字列を返します。
  - 指定されていない
  - *position-expression* から *expression* の末尾までの部分文字列の長さよりも大きい
- SUBSTRN 関数は、次の条件のいずれかに該当する場合、長さゼロの欠損値(SAS モード)または null 値(ANSI モード)を返します。
  - *position-expression* が欠損値(SAS モード)または null 値(ANSI モード)
  - *length-expression* が欠損値(SAS モード)または null 値(ANSI モード)

注: さらに、引数が無効であることを示すエラーがログに書き込まれます。

- SUBSTRN 関数は、次の条件のいずれかに該当する場合、長さゼロの欠損値(SAS モード)または空の文字列(ANSI モード)を返します。
  - *expression* が欠損値(SAS モード)または null 値(ANSI モード)
  - *length-expression* が 1 より小さい
  - *position-expression* が 1 より小さいか、*character-expression* の長さより大きい

比較

次の表に、SUBSTRN 関数と SUBSTR 関数の比較を示します。

条件	関数	結果
position-expression の値が正でない	SUBSTRN	長さ 0 の欠損値(SAS モード)または空の文字列(ANSI モード)を返す
position-expression の値が正でない	SUBSTR	長さ 0 の欠損値(SAS モード)または null 値(ANSI モード)を返す
length-expression の値が正でない	SUBSTRN	長さ 0 の欠損値(SAS モード)または空の文字列(ANSI モード)を返す
length-expression の値が正でない	SUBSTR	*position-expression* から *character-expression* の末尾までの部分文字列を返す
length-expression の値が欠損値(SAS モード)または null 値(ANSI モード)	SUBSTRN	長さゼロの欠損値または null 値を返し、第 3 引数が無効であるというエラーをログに書き込む
length-expression の値が欠損値(SAS モード)または null 値(ANSI モード)	SUBSTR	*position-expression* から *expression* の末尾までの部分文字列を返し、第 3 引数が無効であるという警告をログに書き込む

例

例 1: SUBSTRN 関数を使用した文字列の操作

次のプログラムでは、SUBSTRN 関数を使用して文字列を操作する方法を示します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl char(6) string result;
  dcl double position length;
  retain string 'abcd';
  drop string;
  method run();
    do position=-1 to 6;
      do length=max(-1,-position) to 7-position;
        result=substrn(string, position, length);
        output;
      end;
    end;
  end;
enddata;
run;
quit;

proc print data=test;
run;
```

アウトプット 7.33　SUBSTRN 関数からの部分出力

Obs	result	position	length
1		-1	1
2		-1	2
3	a	-1	3
4	ab	-1	4
5	abc	-1	5
6	abcd	-1	6
7	abcd	-1	7
8	abcd	-1	8
9		0	0
10		0	1
11	a	0	2
12	ab	0	3
13	abc	0	4
14	abcd	0	5
15	abcd	0	6
16	abcd	0	7
17		1	-1
18		1	0
19	a	1	1
20	'		2

例 2: SUBSTR 関数と SUBSTRN 関数の比較

次のプログラムでは、第 1 引数が数値のときに SUBSTR 関数と SUBSTRN 関数を使用した結果を比較しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl char substr_result substrn_result;
  method run();
    substr_result='*' || substr('   1234.5678',2,6) || '*';
    put substr_result=;
    substrn_result='*' || substrn('1234.5678',2,6) || '*';
    put substrn_result=;
  end;
```

```
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
substr_result=*  1234*
substrn_result=*234.56*
```

例 3: VARCHAR 引数での SUBSTRN の使用

次のプログラムでは、VARCHAR データ型の文字列で SUBSTRN 関数を使用しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
   dcl varchar(100000) inarg inarg2 inarg3;
   dcl double l;
   method init();
     inarg = 'x';
     inarg2 = repeat(inarg,30000-1)||'abc';
     l=length(inarg2);
     inarg3 = substrn(inarg2,29998,5);
     put l= inarg3=;
     inarg2 = repeat(inarg,90000-1)||'abc';
     l=length(inarg2);
     inarg3 = substrn(inarg2,89998,5);
     put l= inarg3= ' (should be xxxab)';
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
l=30003 inarg3=xxxab
l=90003 inarg3=xxxab  (should be xxxab)
```

関連項目

関数:

- “擬似 SUBSTR(割り当ての左辺に用いた場合) 関数” (1168 ページ)
- “SUBSTR (=の右) 関数” (1172 ページ)

SUM 関数

非 null または非欠損引数の合計を返します。

カテゴリ:	記述統計
返されるデータの種類:	BIGINT、DECIMAL、DOUBLE
注:	2025.12 以降、この関数では、引数が DOUBLE の場合、すべての引数が DOUBLE に変換され、結果は DOUBLE になります。PROC DS2 ステートメントで BIGINTPROCESSING=YES が設定され、いずれかの引数が BIGINT の場合、すべての引数が BIGINT に変換され、結果は BIGINT になります。

構文

SUM(*expression-1*, *expression-2* [, ...*expression-n*])

必須引数

expression
数値に評価される有効な式を指定します。

要件	少なくとも 2 つの引数が必要です。
データの種類	BIGINT、DECIMAL、DOUBLE
参照項目:	"DS2 式" (*SAS DS2 プログラマガイド*)

詳細

null 値と欠損値は無視され、計算には含まれません。すべての引数が欠損値の場合、結果は欠損値になります。すべての引数が null 値の場合、結果は null 値になります。

この関数の引数が数値以外の場合、その引数は DOUBLE に変換されます。いずれかの引数が DOUBLE または REAL の場合、すべての引数が(まだ変換されていなければ) DOUBLE に変換され、結果は DOUBLE になります。それ以外の場合で、いずれかの引数が DECIMAL のときは、すべての引数が(まだ変換されていなければ) DECIMAL に変換され、結果は DECIMAL になります。それ以外の場合は、すべての引数が BIGINT に変換され、結果は BIGINT になります。

例

次のプログラムは、SUM 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double u v w x y z;
  dcl double val1 val2 val3 val4;
  method run();
    val1=4;
    val2=3;
    val3=9;
    val4=8;
    u=sum(4,3,9,8);
    v=sum(val1, val2, val3, val4);
    w=sum(val1, val2, val3, val4, .);
    x=sum(of val1-val2, of val3-val4);
    y=sum(of val1-val2);
    z=sum(of val:);
    put u= v= w= x= y= z=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
u=24 v=24 w=24 x=24 y=7 z=24
```

SUMABS 関数

非欠損引数の絶対値の合計を返します。

カテゴリ:	記述統計
返されるデータの種類:	DOUBLE
注:	2025.12 以降、この関数では、引数が DOUBLE の場合、すべての引数が DOUBLE に変換され、結果は DOUBLE になります。PROC DS2 ステートメントで BIGINTPROCESSING=YES が設定され、いずれかの引数が BIGINT の場合、すべての引数が BIGINT に変換され、結果は BIGINT になります。

構文

SUMABS(*value-1*[, ...*value-n*])

必須引数

value
数値に評価される有効な式を指定します。

データの種類 DOUBLE

詳細

すべての引数が null または欠損値の場合、結果は null または欠損値になります。それ以外の場合は、結果は非欠損値の絶対値の合計です。

例

例 1: 絶対値の合計の計算

次のプログラムでは、非欠損引数の絶対値の合計を返します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double x;
  method run();
    x=sumabs(1,.,-2,0,3,.q,-4);
    put x=;
  end;
enddata;
run;
quit;
```

SAS は次の結果をログに書き込みます。

```
x=10
```

例 2: 変数リスト使用時の絶対値の合計の計算

次のプログラムでは、変数リストを使用して非欠損引数の絶対値の合計を返します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double x1 x2 x3 x4 x5 x;
  method run();
    x1 = 1;
    x2 = 3;
    x3 = 4;
    x4 = 3;
    x5 = 1;
    x = sumabs(x1, x2, x3, x4, x5);
  put x=;
  end;
enddata;
run;
quit;
```

SAS は次の結果をログに書き込みます。

```
x=12
```

SYSGET 関数

指定された動作環境変数の値を返します。

カテゴリ: 特殊

制限事項: この関数は、管理レベルの機能を持つユーザーに対してのみ CAS でサポートされます。

構文

SYSGET('*environment-variable*')

必須引数

environment-variable
Linux での環境変数名を値とする文字定数、変数または式です。

要件 この引数は、単一引用符で囲む必要があります。

詳細

概要情報

SYSGET 関数は、文字列として環境変数の値を返します。たとえば、次のステートメントは、Linux の HOME 環境変数の値を返します。

```
here=sysget('HOME');
```

まだ長さが割り当てられていない変数に SYSGET 関数から値が返される場合、その変数にはデフォルトで長さ 200 が割り当てられます。

動作環境変数の値が切り捨てられる場合、または動作環境で変数が定義されていない場合、SYSGET は SAS ログに警告メッセージを表示します。

注: *environment-variable* 引数に入力する値の大文字と小文字の区別は、Linux 動作環境に保存されている変数の大文字と小文字の区別に一致している必要があります。

TAN 関数

正接を返します。

カテゴリ: 三角関数

返されるデータの種類: DOUBLE

構文

TAN(*expression*)

必須引数

expression
ラジアン単位の数値に評価される有効な式を指定します。

制限事項 *expression* は、π/2 の奇数倍にはできません。

データの種類 DOUBLE

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

例

次のプログラムは、TAN 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
```

```
  dcl double x1 x2 x3;
  method run();
    x1=tan(0.5);
    x2=tan(0);
    x3=tan(3.14159/3);
    put x1= x2= x3=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
x1=0.54630248984379 x2=0 x3=1.73204726945457
```

関連項目

関数:

- “TANH 関数” (1186 ページ)

TANH 関数

双曲線正接を返します。

カテゴリ:	双曲線
返されるデータの種類:	DOUBLE

構文

TANH(*expression*)

必須引数

expression
数値に評価される有効な式を指定します。

制限事項	*expression* は、π/2 の奇数倍にはできません。
データの種類	DOUBLE
参照項目:	“DS2 式” (*SAS DS2 プログラマガイド*)

詳細

TANH 関数は、次の式によって得られる引数の双曲線正接を返します。

$$\frac{e^{argument} - e^{-argument}}{e^{argument} + e^{-argument}}$$

例

次のプログラムは、TANH 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl double a b c;
  method run();
    a=tanh(0);
    b=tanh(0.5);
    c=tanh(-0.5);
    put 'a= ' a;
    put 'b= ' b;
    put 'c= ' c;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
a= 0
b= 0.46211715726
c= -0.46211715726
```

関連項目

関数:

- “TAN 関数” (1185 ページ)

TIME 関数

数値の SAS 時間値として現在時刻を返します。

カテゴリ: 日付と時間

返されるデータの種類: DOUBLE

構文

TIME( )

詳細

TIME 関数は引数を取りません。SAS 時間値の形式で現在時刻を生成します。

例

次のプログラムは、TIME 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double current;
  method run();
    current=time();
    put current=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
current=40486.0569992065
```

```
proc ds2;
data test(overwrite=yes);
  dcl double current;
  method run();
    current=time();
    put current=time.;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
current=11:14:46
```

関連項目

- “DS2 式” (*SAS DS2 プログラマガイド*)

関数:

- “DATE 関数” (495 ページ)
- “DATETIME 関数” (499 ページ)

TIMEPART 関数

SAS 日時値から時間値を抽出します。

カテゴリ: 日付と時間

返されるデータの種類: DOUBLE

構文

TIMEPART(*datetime*)

必須引数

datetime
SAS 日時値を表す有効な式を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

例

次のプログラムは、TIMEPART 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double dttm tm;
  dcl char(10) x;
  method run();
    dttm=datetime();
```

```
    put dttm;
    x=put(timepart(dttm), time.);
    put x;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
1866291827.48899
14:23:47
```

関連項目

- “DS2 式” (*SAS DS2 プログラマガイド*)

関数:

- “DATEPART 関数” (498 ページ)

TIMEVALUE 関数

変動金利を使用して、基準日の参照額に相当する額を返します。

カテゴリ:	財務
返されるデータの種類:	DOUBLE

構文

TIMEVALUE(*base-date*, *reference-date*, *reference-amount*, *compounding-interval*, *date-1*, *rate-1*[, ...*date-n*, *rate-n*])

必須引数

base-date
base-date における *reference-amount* の時間値を指定します。

要件	*base-date* は SAS 日付です。
データの種類	DOUBLE

reference-date
reference-amount の日付を指定します。

要件 *reference-date* は SAS 日付です。

データの種類 DOUBLE

reference-amount
reference-date における金額を指定します。

データの種類 DOUBLE

compounding-interval
複利間隔を指定します。

要件 *compounding-interval* は SAS 間隔です。

データの種類 CHAR

date
rate が有効になるタイミングを指定します。各日付は利率とペアになります。

要件 *date* は SAS 日付です。

データの種類 DOUBLE

rate
date に開始する利率を数値のパーセンテージで指定します。各利率は日付とペアになります。

データの種類 DOUBLE

詳細

TIMEVALUE 関数には次の詳細が適用されます。

- 利率の値は–99 から 120 までの値にする必要があります。
- 日付-利率ペアのリストは、日付順に並べ替える必要はありません。
- 1 日に複数の利率が変更される場合、TIMEVALUE 関数は、その日付でリストされている最後の利率のみを適用します。
- 一部の期間には単利が適用されます。
- *reference-date* と *base-date* の両方の日付またはその前の日付となる有効な日付-利率ペアが必要になります。

例

- 名目金利が 10%で 1 年間毎月複利計算される$1,000 の投資の累計額は、次のように表すことができます。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double bd rd d amount_base1;
  method run();
    bd= to_double(date'2001-01-01');
    rd= to_double(date'2000-01-01');
    d= to_double(date'2000-01-01');
    amount_base1 = timevalue(bd, rd, 1000, 'month', d, 10);
    put amount_base1;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
1104.71306744129
```

- 半年後に利率が 20%に上昇した場合の計算は次のとおりです。

```
proc ds2;
data _null_;
  dcl double bd rd d1 d2 amount_bases;
  method run();
    bd= to_double(date'2001-01-01');
    rd= to_double(date'2000-01-01');
    d1= to_double(date'2000-01-01');
    d2= to_double(date'2000-07-01');
    amount_base2 = timevalue(bd, rd, 1000, 'month', d1, 10, d2, 20);
    put amount_base2;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
1160.63657783831
```

- 日付-利率ペアは、日付順に並べ替える必要はありません。この柔軟性があるため、amount_base2 と amount_base3 は同じ値になります。

```
proc ds2;
data _null_;
  dcl double bd rd d1 d2 amount_base3;
  method run();
    bd= to_double(date'2001-01-01');
    rd= to_double(date'2000-01-01');
    d1= to_double(date'2000-07-01');
    d2= to_double(date'2000-01-01');
    amount_base3 = timevalue(bd, rd, 1000, 'month', d1, 20, d2, 10);
    put amount_base3;
end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
1160.63657783831
```

TINV 関数

t 分布から分位点を返します。

カテゴリ:	分位点
返されるデータの種類:	DOUBLE

構文

TINV(*p*, *df*[, *nc*])

必須引数

p
数値の確率に評価される有効な式を指定します。

範囲	$0 < p < 1$
データの種類	DOUBLE
参照項目:	“DS2 式” (*SAS DS2 プログラマガイド*)

df
数値の自由度パラメーターに評価される有効な式を指定します。

範囲	$df > 0$
データの種類	DOUBLE
参照項目:	“DS2 式” (*SAS DS2 プログラマガイド*)

オプション引数

nc
数値の非心度パラメーターに評価される有効な式を指定します。

データの種類	DOUBLE
参照項目:	“DS2 式” (*SAS DS2 プログラマガイド*)

詳細

TINV 関数は、自由度が *df*、非心度パラメーターが *nc* のスチューデントの *t* 分布から *p*番目の分位点を返します。*t* 分布のオブザベーションが、返される分位点以下となる確率は *p* です。

TINV では、整数以外の自由度パラメーター *df* を指定できます。このパラメーター *nc* (省略可能)が指定されていないか 0 の場合、心度 *t* 分布から分位点が返されます。

注意
nc の値が大きいとアルゴリズムが失敗する場合があります。 その場合、欠損値が返されます。

比較

TINV は PROBT 関数の逆数です。

例

次のプログラムは、TINV 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
 proc ds2;
data _null_;
  dcl double x y;
  method run();
    x=tinv(.95, 2);
    y=tinv(.95, 2.5, 3);
    put x y;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
x=2.91998558035372
y=11.0338336251942
```

関連項目

関数:

- “PROBT 関数” (1011 ページ)

TNONCT 関数

スチューデントの *t* 分布から非心度パラメーターの値を返します。

カテゴリ: 数学

返されるデータの種類: DOUBLE

構文

TNONCT(*x*, *df*, *prob*)

必須引数

x
数値の確率変数です。

データの種類 DOUBLE

df
数値の自由度パラメーターです。

範囲 *df* > 0

データの種類 DOUBLE

prob
確率です。

範囲 0 < *prob* < 1

データの種類 DOUBLE

詳細

TNONCT 関数は、パラメーターが *x*、*df* および *nc* の非心度 *t* 分布から、負でない非心度パラメーターを返します。式の平方根 *nc* を求めるためにニュートン型のアルゴリズムが使用されます。

$$P_t(x|df, nc) - prob = 0$$

前述の式には次の関係が適用されます。

$$P_t(x|df,nc) = \frac{1}{\Gamma\left(\frac{df}{2}\right)} \int_0^{\infty} v^{\frac{df}{2}-1} \varepsilon^{-v} \int_{-\infty}^{x\sqrt{\frac{2v}{df}}} \varepsilon^{-\frac{(u-nc)^2}{2}} du dv$$

アルゴリズムで固定小数点を収束できない場合、欠損値が返されます。

例

次のプログラムでは、*t* 分布から非心度パラメーターを計算します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data work /overwrite=yes;
  dcl double x df nc prob ncc;
  method init();
    x=2;
    df=4;
    do nc=1 to 3 by .5;
      prob=probt(x, df, nc);
      ncc=tnonct(x, df, prob);
      output;
    end;
  end;
enddata;
run;
quit;

proc print data=work;
run;
```

アウトプット 7.34　*t* 分布の非心度パラメーターの計算出力結果

Obs	x	df	nc	prob	ncc
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

関連項目

関数:

- “CNONCT 関数” (422 ページ)
- “FNONCT 関数” (659 ページ)

TO_DATE 関数

SAS 日付値を指定する DOUBLE 値から DATE 値を返します。

カテゴリ: 日付と時間

返されるデータの種類: DATE

構文

TO_DATE(*date*)

必須引数

date
SAS 日付値を表す有効な式を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

SAS 日付値を指定する DOUBLE 値は、1960 年 1 月 1 日から指定日付までの日数を表します。SAS は、西暦 1582 年から西暦 19,900 年までの日付で計算を実行できます。1960 年 1 月 1 日より前の日付は負の数、1960 年 1 月 1 日以降の日付は正の数です。

- SAS 日付値は、2000 年のうるう年の日を含むすべてのうるう年の日を考慮しています。
- SAS 日付値は、カレンダーが数日削除されて調整された 1752 年 9 月までさかのぼって、特定の曜日が何日に当たるかを確実に知ることができます。SAS の曜日と時間の長さの計算は、西暦 19,900 年までの未来で正確です。

例

次のプログラムは、SAS 日付値を指定する DOUBLE 値を DATE 値に変換します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
/* SAS date to date */
proc ds2;
data _null_;
  dcl date da;
  dcl double d;
  method init();
    d = 22096;
    da = to_date(d);
    put d=YYMMDD10. da=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
d=2020-06-30 da=2020-06-30
```

関連項目

関数:

- “TO_DOUBLE 関数” (1198 ページ)
- “TO_TIME 関数” (1202 ページ)
- “TO_TIMESTAMP 関数” (1203 ページ)

TO_DOUBLE 関数

DATE、TIME、または TIMESTAMP 値から、SAS 日付、時間、または日時の値を指定する DOUBLE 値を返します。

カテゴリ:	日付と時間
返されるデータの種類:	DOUBLE

構文

TO_DOUBLE (*date* | *time* | *timestamp*)

必須引数

date
日付定数、変数または式です。

time
時間定数、変数または式です。

timestamp
低無スタンプ定数、変数または式です。

詳細

次のリストは、TO_DOUBLE 関数によって返される値を示しています。

- SAS 日付値は、1960 年 1 月 1 日から指定日付までの日数を表します。SAS は、西暦 1582 年から西暦 19,900 年までの日付で計算を実行できます。1960 年 1 月 1 日より前の日付は負の数、1960 年 1 月 1 日以降の日付は正の数です。
  - SAS 日付値は、2000 年のうるう年の日を含むすべてのうるう年の日を考慮しています。
  - SAS 日付値は、カレンダーが数日削除されて調整された 1752 年 9 月までさかのぼって、特定の曜日が何日に当たるかを確実に知ることができます。SAS の曜日と時間の長さの計算は、西暦 19,900 年までの未来で正確です。
- SAS 時間値は、現在日の午前 0 時からの秒数を表します。SAS 時間値は 0 - 86400 です。
- タイムスタンプは、特定のイベントが発生した日時の記録です。
- SAS 日時値は、1960 年 1 月 1 日から、指定日付内の時/分/秒までの秒数を表します。

例

例 1: TIMESTAMP 値から DOUBLE 値への変換

次のプログラムは、TIMESTAMP 値を、SAS 日時値を指定する DOUBLE 値に変換します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
/* Timestamp to SAS datetime */
proc ds2;
data _null_;
```

```
    dcl timestamp ts;
    dcl double d_asd;
    dcl date d;
    method init();
      ts = timestamp '2019-10-19 16:51:36.0625';
      d_asd = to_double(ts);
      put ts=;
      put d_asd;
      put d_asd=DATETIME28.9;
    end;
  enddata;
  run;
  quit;
```

SAS は次の出力をログに書き込みます。

```
ts=2019-10-19 16:51:36.062500000
1887123096.0625
d_asd=19OCT2019:16:51:36.062500000
```

例 2: 日付値から SAS 日付値への変換

次のプログラムは、DATE 値を、SAS 日付値を指定する DOUBLE 値に変換します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
/* Date to SAS date */
proc ds2;
data _null_;
  dcl date da;
  dcl double d;
  method init();
    da = date '2019-10-19';
    d = to_double(da);
    put d;
    put d=YYMMDD10.;
    put da=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
21841
d=2019-10-19
da=2019-10-19
```

例 3: 時間値から SAS 時間値への変換

次のプログラムは、TIME 値を SAS 時間値に変換します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
/* Time to SAS time */
proc ds2;
data _null_;
  dcl time t;
  dcl double d;
  method init();
    t = time '16:51:36.0625';
    d = to_double(t);
    put d=TIME18.9 t=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
d=16:51:36.062500000 t=16:51:36.062500000
```

例 4: NULL タイムスタンプから SAS 日時値への変換

次のプログラムは、NULL TIMESTAMP を SAS 日時値に変換します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
/* NULL timestamp to SAS datetime */
proc ds2;
data _null_;
  dcl timestamp ts;
  dcl double d;
  method init();
    ts = null;
    d = to_double(ts);
    put d=DATETIME28.9 ts=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
d=                    . ts=
```

関連項目

関数:

- “TO_DATE 関数” (1197 ページ)

- “TO_TIME 関数” (1202 ページ)
- “TO_TIMESTAMP 関数” (1203 ページ)

TO_TIME 関数

SAS 時間値を指定する DOUBLE 値から TIME 値を返します。

カテゴリ: 日付と時間

返されるデータの種類: TIME

構文

TO_TIME(*date*)

必須引数

date
SAS 時間値を表す有効な式を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

SAS 時間値は、現在日の午前 0 時からの秒数を表します。SAS 時間値は 0 - 86400 です。

例

次のプログラムは、SAS 時間値をフォーマットされた時間値に変換します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
/* SAS time to time */
proc ds2;
data _null_;
  dcl time t;
  dcl double d;
  method init();
```

```
    d = 45911.68;
    t = to_time(d);
    put d=TIME18.9 t=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
d=12:45:11.680000000 t=12:45:11.680000000
```

関連項目

関数:

- “TO_DATE 関数” (1197 ページ)
- “TO_DOUBLE 関数” (1198 ページ)
- “TO_TIMESTAMP 関数” (1203 ページ)

TO_TIMESTAMP 関数

SAS 時間値を指定する DOUBLE 値から TIMESTAMP 値を返します。

カテゴリ:	日付と時間
返されるデータの種類:	TIMESTAMP

構文

TO_TIMESTAMP(*date*)

必須引数

date
SAS 日時値を表す有効な式を指定します。

データの種類	DOUBLE
参照項目:	“DS2 式” (*SAS DS2 プログラマガイド*)

詳細

SAS 日時値は、1960 年 1 月 1 日から指定日付内の時/分/秒までの秒数を表す DOUBLE です。

例

例 1: SAS 日時値からタイムスタンプ値への変換

次のプログラムは、SAS 日時値をタイムスタンプ値に変換します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
/* SAS datetime to timestamp */
proc ds2;
data _null_;
  dcl timestamp ts;
  dcl double d;
  method init();
    d = 1845773470.3;
    ts = to_timestamp(d);
    put d=DATETIME28.9 ts=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
d=28JUN2018:02:51:10.299999952 ts=2018-06-28 02:51:10.299999952
```

例 2: 欠損している SAS 日時値の変換

次のプログラムは、欠損している SAS 日時値の変換を処理する方法を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
/* Missing SAS datetime to timestamp */
proc ds2;
data _null_;
  dcl timestamp ts;
  dcl double d;
  method init();
    d = .;
    ts = to_timestamp(d);
    put d=DATETIME28.9 ts=;
  end;
enddata;
```

```
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
d=                . ts=
```

関連項目

関数:

- “TO_DATE 関数” (1197 ページ)
- “TO_DOUBLE 関数” (1198 ページ)
- “TO_TIME 関数” (1202 ページ)

TODAY 関数

数値の SAS 日付値として現在の日付を返します。

カテゴリ:	日付と時間
返されるデータの種類:	DOUBLE

構文

TODAY( )

詳細

TODAY 関数は引数を取りません。SAS 日付値(1960 年 1 月 1 日からの日数)の形式で現在の日付を生成します。

DS2 が日付を処理する方法の詳細については、“DS2 式” (*SAS DS2 プログラマガイド*)を参照してください。

例

次のプログラムは、TODAY 関数を示しています。

```
proc ds2;
data test(overwrite=yes);
```

```
  dcl double td;
  dcl char(10) tday;
  method run();
    td=today();
    tday=put(td, date.);
    put td;
    put tday;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
21600
20FEB19
```

TRANSLATE 関数

文字式内の特定の文字を置き換えます。

カテゴリ: 文字

返されるデータの種類: CHAR、NCHAR、NVARCHAR、VARCHAR

構文

TRANSLATE(*expression*, *to-characters*, *from-characters*)

必須引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。*expression* には元の文字値が含まれます。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

to-characters
TRANSLATE で代替文字として使用する文字を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

from-characters
TRANSLATE で置換する文字を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

詳細

to-characters と *from-characters* の値は、個々の文字ごとに対応しています。たとえば、TRANSLATE では、*from-characters* の最初の文字が *to-characters* の最初の文字に変更されます。*to-characters* の文字数が *from-characters* の文字数より少ない場合、TRANSLATE は *from-characters* の余った文字を空白に変更します。*to-characters* の文字数が *from-characters* の文字数より多い場合、TRANSLATE は *to-characters* の余った文字を無視します。

比較

TRANSTRN 関数は、置換文字列にゼロの長さを使用できるという点で TRANWRD 関数と異なります。TRANWRD では、置換文字列の長さがゼロ場合はかわりに 1 つの空白を使用します。

TRANSLATE 関数は、ユーザーが指定したすべての文字を別の文字に変換します。TRANSLATE は 1 回の呼び出しで複数の文字をスキャンできます。ただし、このスキャンを実行すると、TRANSLATE は文字列に含まれる個々のすべての文字を検索します。つまり、ターゲット文字列内の文字がソース文字列内に見つかった場合、その文字は対応する置換文字列内の文字に置き換えられます。

TRANWRD 関数は、単語(または文字のパターン)をスキャンし、それらの単語を 2 番目の単語(または文字のパターン)で置き換える点で TRANSLATE とは異なります。

例

次のプログラムは、TRANSLATE 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl char(10) x y;
  method run();
    x=translate('XYZW', 'AB', 'VW');
    y='AABBAABABB';
    y=translate(y, '12', 'AB');
    put x;
    put y;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
XYZB
1122112122
```

関連項目

関数:

- “TRANSTRN 関数” (1208 ページ)
- “TRANWRD 関数” (1211 ページ)

TRANSTRN 関数

文字列に含まれる特定の部分文字列をすべて置き換えるか取り除きます。

カテゴリ: 文字

返されるデータの種類: CHAR、NCHAR、NVARCHAR、VARCHAR

構文

TRANSTRN(*source-expression*, *target-expression*, *replacement-expression*)

必須引数

source-expression
文字列に評価されるか強制変換できて、変換する文字を含む有効な式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

target-expression
文字列に評価されるか強制変換できて、その文字が *source-expression* で検索される有効な式を指定します。

要件 *target-expression* の長さにはゼロより大きい値を指定する必要があります。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

replacement-expression
文字列に評価されるか強制変換できて、*target-expression* を置換する有効な式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

TRANSTRN 関数は、文字列に含まれる特定の部分文字列をすべて置き換えるか取り除きます。TRANSTRN 関数では、*target-expression* 文字列と *replacement-expression* 文字列の末尾の空白は取り除きません。出現する *target* をすべて取り除くには、*replacement-expression* に TRIMN(' ')を指定します。

比較

TRANSTRN 関数は、置換文字列にゼロの長さを使用できるという点で TRANWRD 関数と異なります。TRANWRD では、置換文字列の長さがゼロ場合はかわりに 1 つの空白を使用します。

TRANSLATE 関数は、ユーザーが指定したすべての文字を別の文字に変換します。TRANSLATE は 1 回の呼び出しで複数の文字をスキャンできます。ただし、このスキャンを実行すると、TRANSLATE は文字列に含まれる個々のすべての文字を検索します。つまり、ターゲット文字列内の文字がソース文字列内に見つかった場合、その文字は対応する置換文字列内の文字に置き換えられます。

TRANSTRN 関数は、部分文字列をスキャンし、それらの部分文字列を 2 番目の部分文字列で置き換える点で TRANSLATE と異なります。

例

例 1: 単語のすべての出現箇所の置換

このプログラムでは、TRANSTRN 関数を使用して **Mrs.**と **Miss** を **Ms.**に置き換えています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl char(20) name;
  method run();
    name='Mrs.  Joan Smith';
    name=transtrn(name, 'Mrs.', 'Ms.');
    put name;
    name='Miss Alice Cooper';
    name=transtrn(name, 'Miss', 'Ms.');
    put name;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
Ms.  Joan Smith
Ms. Alice Cooper
```

例 2: 検索文字列からの空白の除外

このプログラムでは、空白を除外するために TRIM 関数を *target* とともに使用しています。TRIM 関数を含めなかった場合、ターゲット文字列に空白が含まれるため、TRANSTRN 関数はソース文字列を置き換えません。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test (overwrite=yes);
  dcl char(10) target;
  dcl char(3) replacement;
  dcl char(8) salelist salelist2;
  method run();
    salelist='CATFISH';
    target='FISH';
    replacement='NIP';
    salelist2=transtrn(salelist, trim(target), replacement);
    put salelist2;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
CATNIP
```

例 3: TRANSTRN 関数の第 3 引数の長さがゼロの場合

次のプログラムでは、第 3 引数 *replacement* の長さがゼロの場合の TRANSTRN 関数の結果を示しています。DS2 では、2 つの引用符とその間にある空白で構成される文字定数は、長さがゼロの文字列ではなく 1 つの空白を表します。次のプログラムでは、*string1* の結果は *string2* の結果とは異なります。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl char string1 string2;
  method run();
    string1='*' || transtrn('abcxabc', 'abc', trimn(' ')) || '*';
    put string1=;
    string2='*' || transtrn('abcxabc', 'abc', ' ') || '*';
    put string2=;
  end;
enddata;
run;
```

```
quit;
```

SAS は次の出力をログに書き込みます。

関連項目

関数:

- “TRANSLATE 関数” (*SAS 関数と CALL ルーチン: リファレンス*)
- “TRANWRD 関数” (1211 ページ)

TRANWRD 関数

文字列に含まれる特定の部分文字列をすべて置き換えます。

カテゴリ:	文字
返されるデータの種類:	CHAR、NCHAR、NVARCHAR、VARCHAR

構文

TRANWRD(*source-expression*, *target-expression*, *replacement-expression*)

必須引数

source-expression
置き換える文字を含む文字列に評価されるか強制変換できる有効な式を指定します。

データの種類	CHAR、NCHAR、NVARCHAR、VARCHAR
参照項目:	“DS2 式” (*SAS DS2 プログラマガイド*)

target-expression
文字列に評価されるか強制変換できて、*source-expression* で検索される有効な式を指定します。

要件	*target-expression* の長さにはゼロより大きい値を指定する必要があります。
データの種類	CHAR、NCHAR、NVARCHAR、VARCHAR
参照項目:	“DS2 式” (*SAS DS2 プログラマガイド*)

replacement-expression
文字列に評価されるか強制変換できて、*target-expression* を置換する有効な式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

TRANWRD 関数は、文字列に含まれる特定の部分文字列(または文字のパターン)の出現箇所をすべて置き換えます。TRANWRD 関数では、*target-expression* 文字列と *replacement-expression* 文字列の末尾の空白は取り除きません。

比較

TRANSTRN 関数は、置換文字列にゼロの長さを使用できるという点で TRANWRD 関数と異なります。TRANWRD では、置換文字列の長さがゼロ場合はかわりに 1 つの空白を使用します。

TRANSLATE 関数は、ユーザーが指定したすべての文字を別の文字に変換します。TRANSLATE は 1 回の呼び出しで複数の文字をスキャンできます。ただし、これを実行すると、TRANSLATE は文字列に含まれる個々のすべての文字を検索します。つまり、ターゲット文字列内の文字がソース文字列内に見つかった場合、その文字は対応する置換文字列内の文字に置き換えられます。

TRANWRD 関数は、部分文字列(または文字のパターン)をスキャンし、それらの部分文字列を 2 番目の部分文字列(または文字のパターン)で置き換える点で TRANSLATE とは異なります。

例

例 1: 単語のすべての出現箇所の置換

次のプログラムは、TRANWRD 関数を使用して、出現するすべての部分文字列を置換する方法を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl char(16) text1 text2 nametrn1 nametrn2;
  method run();
    text1='Mrs. Jane Doe';
    text2='Miss Joan Smith';
```

```
    nametrn1=tranwrd(text1, 'Mrs.', 'Ms.');
    nametrn2=tranwrd(text2, 'Miss', 'Ms.');
    put nametrn1;
    put nametrn2;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
Ms. Jane Doe
Ms. Joan Smith
```

例 2: データ型の変更による検索文字列からの空白の削除

このプログラムは、誤ったデータ型宣言を示しています。TARGET は char(10)として宣言されており、TRANWRD 関数は''pail'ではなく文字列'pail 'を検索するため、TRANWRD 関数はソース文字列を置換しません。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl char(100) text finaltext;
  dcl char(10) target;
  dcl varchar(10) rplc;
  method run();
    text='Believe and act as if it were impossible to pail.
       -Charles F. Kettering';
    target='pail';
    rplc='fail';
    finaltext=tranwrd(text, target, rplc);
    put finaltext=;
  end;
enddata;
run;
quit;
```

この行が SAS ログに書き込まれます。

```
finaltext=Believe and act as if it were impossible to pail.-Charles F. Kettering
```

データ型宣言を VARCHAR(10)に変更することにより、末尾の空白が検索から除外されます。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl char(100) text finaltext;
  dcl varchar(10) target;
```

```
  dcl varchar(10) rplc;
  method run();
    text='Believe and act as if it were impossible to pail.
      -Charles F. Kettering';
    target='pail';
    rplc='fail';
    finaltext=tranwrd(text, target, rplc);
    put finaltext=;
  end;
enddata;
run;
quit;
```

この行が SAS ログに書き込まれます。

```
finaltext=Believe and act as if it were impossible to fail.-Charles F. Kettering
```

例 3: TRIM を使用して検索文字列から空白を削除する

このプログラムでは、ターゲット文字列に空白が含まれるため、TRANWRD 関数はソース文字列を置き換えません。

注: このサンプルコードでは、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl char prod target replacement salelist;
  method run();
    prod='CATFISH';
    target='FISH';
    replacement='NIP';
    prod=tranwrd(prod, target, replacement);
    put prod;
  end;
enddata;
run;
quit;
```

DECLARE ステートメントは target を 8 の長さまで空白で埋め込むため、TRANWRD 関数は PROD 内で文字列'FISH 'を検索します。検索は失敗するため、次の行が SAS ログに書き込まれます: CATFISH

target または replacement 変数から末尾の空白を除外するには、TRIM 関数を使用します。target に TRIM 関数を使用します。

```
prod=tranwrd(prod, trim(target), replacement;
put prod;
```

SAS は次の出力をログに書き込みます。

```
CATNIP
```

例 4: カンマの繰り返しの除外

TRANWRD 関数を使用して、テキスト内のカンマの繰り返しを取り除いて 1 つのカンマに置き換えることができます。このプログラムでは、1 つ目の TRANWRD 関数で 3 つのカンマを 1 つのカンマに置き換え、2 つ目の TRANWRD 関数で末尾の 2 つのカンマをピリオドに置き換えます。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl char(70) mytxt newtext newtext2;
  method run();
    mytxt='If you exercise your power to vote,,,then your opinion
      will be heard,,';
    newtext=tranwrd(mytxt, ',,,', ',');
    newtext2=tranwrd(newtext, ',,' , '.');
    put mytxt=;
    put newtext=;
    put newtext2=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
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.
```

関連項目

関数:

- “TRANSLATE 関数” (1206 ページ)
- “TRANSTRN 関数” (1208 ページ)

TRIGAMMA 関数

トリガンマ関数の値を返します。

カテゴリ:	数学
返されるデータの種類:	DOUBLE

構文

TRIGAMMA(*expression*)

必須引数

expression
数値に評価される有効な式を指定します。

制限事項　正でない整数は無効です。

データの種類　DOUBLE

参照項目:　"DS2 式" (*SAS DS2 プログラマガイド*)

詳細

TRIGAMMA 関数は、ディガンマ関数の導関数を返します。*expression* > 0 の場合、TRIGAMMA 関数は lgamma 関数の第 2 導関数です。

例

次のプログラムは、TRIGAMMA 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double x;
  method run();
    x=trigamma(3);
    put x=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
x=0.39493406684822
```

関連項目

関数:

- “DIGAMMA 関数” (517 ページ)
- “LGAMMA 関数” (816 ページ)

TRIM 関数

文字式から末尾の空白を取り除きます。文字式が欠損値の場合は、長さがゼロの文字列を返します。

カテゴリ:	文字
別名:	TRIMN
返されるデータの種類:	CHAR、NCHAR、NVARCHAR、VARCHAR

構文

TRIM('*expression*')

必須引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類	CHAR、NCHAR、NVARCHAR、VARCHAR
参照項目:	“DS2 式” (*SAS DS2 プログラマガイド*)

詳細

TRIM 関数では、文字の引数をコピーし、末尾の空白を取り除いて、その結果調整された引数を返します。引数が空白の場合、TRIM は長さがゼロの文字列を返します。連結では末尾の空白は取り除かれないため、TRIM は連結後に便利です。

注: TRIM 関数は、Unicode 標準で定義されている空白とスペースの両方の文字を取り除きます。したがって、TRIM 関数は DBCS ブランクとシフトアウト/シフトイン(SO/SI)エスケープコードも処理します。Unicode スペース文字標準の詳細については、Wikipedia: Unicode character property を参照してください。

例

例 1: 末尾の空白の削除

次のプログラムは、TRIM 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl char(17) string result;
  method run();
    string='Testscore   ';
    result=trim(string)||'File.xls';
    put result;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
TestscoreFile.xls
```

例 2: 空白の文字式の連結

次のプログラムは、TRIM 関数を使用して空白文字を連結する方法を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data new(overwrite=yes);
  dcl char x z y;
  method run();
    x='A'||trim(' ')||'B';
    z=' ';
    y='>'||trim(z)||'<';
    put x;
    put y;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
AB
><
```

関連項目

関数:

- “COMPRESS 関数” (452 ページ)
- “LEFT 関数” (806 ページ)
- “STRIP 関数” (1166 ページ)

TRUNC 関数

数値を指定した長さに切り捨てます。

カテゴリ: 切り捨て

返されるデータの種類: DOUBLE

構文

TRUNC(*expression*, *length-expression*)

必須引数

expression
数値に評価される有効な式を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

length-expression
数値に評価される有効な式を指定します。

範囲 3-8

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

TRUNC 関数では、完全な長さの数値式(DOUBLE として保存)を *length-expression* で指定したより小さいバイト数に切り捨て、切り捨てられたバイトを 0 で埋め込みます。切り捨てとその後の展開は、最初に完全な長さより短い数を保存した結果を複製し、次にそれを読み込みます。

例

次のプログラムは、TRUNC 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double i x;
  method run();
    do i=3 to 8;
      x=trunc(3.1,i);
      put x;
    end;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
3.099609375
3.09999847412109
3.09999999403953
3.09999999997671
3.0999999999999
3.1
```

TSLVL 関数

リリース番号、ホットフィックス、メンテナンスイメージなど、SAS インストールイメージに関する情報を提供します。

カテゴリ: インストール情報

返されるデータの種類: VARCHAR(n)

構文

TSLVL(*binary-file*[,*option*[*option-n*]])

必須引数

binary-file
SAS インストールイメージからの SAS バイナリファイルの名前からファイル拡張子を除いたものを指定します。 DS2 の有効なファイル名には、tkeds、tkmk、

tkedsq、tkeds2、tkedspem が含まれます。tkeds は、DS2 コンパイラのバイナリファイルの名前です。オプションなしでバイナリファイル名を指定すると、SAS は、製品名、テクニカルサポートレベル、メンテナンスレベルなど、SAS インストールイメージに関する高レベルの情報を返します。この情報は、バイナリファイルに固有のものではありません。

データの種類 CHAR(n)

オプション引数

オプションでは大文字と小文字は区別されません。

A

追加のトラック情報を返します。

データの種類 CHAR(n)

B

<Script DVD>/script/<Script Name>.bld という形式でスクリプト情報を書き込みます。

データの種類 CHAR(n)

```
proc ds2;
data _null_;
 method init();
   dcl varchar(30) x;
   x = tslvl('tkeds', 'b');
   put x=;
  end;
enddata;
run;
quit;
```

```
x=tkeds/script/tkeds.bld
```

C

出力区切り文字をカンマからコロンに変更します。I オプションが指定されている場合は、イメージの説明もデコードします。

要件 C オプションは、D、P、T、I のうち少なくとも 1 つのオプションと一緒に指定する必要があります。

データの種類 CHAR(n)

```
proc ds2;
data _null_;
 method init();
  dcl varchar(128) x;
  x = tslvl('tkeds', 'cdpti');
  put x=;
 end;
enddata;
run;
quit;
```

```
x=:V.04.00M0P11012022:tktslang:day/mva-vbviya:DS2 extension
```

D
ポート日付を返します(例: V.04.00M0P10302022)。

データの種類 CHAR(n)

E
1970 年 1 月 1 日からの秒数である UNIX 日付として日付を返します。SAS 日付は、1960 年 1 月 1 日からの秒数に基づいています。

データの種類 CHAR(n)

H
ホットフィックス情報が含まれています。

データの種類 CHAR(n)

I
ファイルの説明が含まれています。Z オプションとともに使用して、フィールドをデコードします。

データの種類 CHAR(n)

M
メンテナンスリリース情報を返します。

データの種類 CHAR(n)

N
モジュールが見つからない場合、一部のエラーメッセージを抑制します。

データの種類 CHAR(n)

P
製品番号を返します。

データの種類 CHAR(n)

S
スクリプト DVD、スクリプト名、スクリプトの場所をこの順序で含むコロン区切りのリストを返します。

データの種類 CHAR(n)

注: Z オプションを使用して、フィールドをデコードします。

<Script DVD>:<Script Name>:<Script Location>

```
proc ds2;
data _null_;
 method init();
   dcl varchar(35) x;
   x = tslvl('sasxkern', 'sz');
   put x=;
 end;
enddata;
```

```
run;
quit;

x=sup:sasxkern:/sas/day/mva-vbviya,
```

T
SAS トラック情報を返します。

データの種類 CHAR(n)

Z
エンコードされたフィールドをデコードします。

データの種類 CHAR(n)

詳細

TSLVL 関数を指定する場合は、バイナリファイルの名前とオプションをカンマで区切ります。必要なすべてのオプションを 1 つの文字列で指定します。

TSLVL 関数の出力は、要求された情報項目ごとに 1 文字の文字列を返し、結果の解析を容易にするためにカンマで区切られます。かわりにコロンを使用するには、C オプションを指定します。

イメージの情報が使用できない場合、次のエラーが表示されることがあります。

- ImageNotLoaded は、要求されたイメージがロードされていないことを示します。
- VersionNotFound は、バージョン情報が見つからなかったことを示します。
- NotAvailable は、バージョン情報を取得する際のその他の問題を示します。

例

次の例では、TSLVL 関数を 2 回呼び出します。最初のインスタンスは、SAS Viya トラック、バージョン番号(.04)、テクニカルサポートレベル(0)、およびメンテナンスレベル(0)を返します。これは、SAS Viya プラットフォームの 4 番目のバージョンです。2 番目のインスタンスは、トラック、ポート日付、およびイメージがビルドされた UNIX 時間を要求します。このイメージは、2022 年 10 月 30 日に SAS Viya プラットフォームのメイン開発トラックでビルドされました。

```
proc ds2;
 data _null_;
 method init();
   dcl varchar(15) y;
   dcl varchar(50) a;
   y=tslvl('sasxkern');
   put y=;
   a=tslvl('sasxkern','tde');
   put a=;
  end;
enddata;
```

```
run;
quit;
```

```
y=V.04 TS0M0
a=day/mva-vbviya,V.04.00M0P10302022,1667167175
```

UNIFORM 関数

一様分布からランダム変量を返します。

注: この関数はサポートされなくなりました。かわりに RAND('UNIFORM')関数 (1058 ページ)を使用してください。

UPCASE 関数

引数内のすべての文字を大文字に変換します。

カテゴリ: 文字

別名: UPPER

返されるデータの種類: CHAR、NCHAR、NVARCHAR、VARCHAR

構文

UPCASE(*expression*)

必須引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

詳細

UPCASE 関数は、文字式をコピーし、すべての小文字を大文字に変換して、変更された値を結果として返します。

比較

LOWCASE 関数は、引数内のすべての文字を小文字に変換します。UPCASE 関数は、引数内のすべての文字を大文字に変換します。

例

次のプログラムは、UPCASE 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data names(overwrite=yes);
  dcl char(13) name;
  method run();
    name='SaraH'; output;
    name='Mylinda'; output;
    name='Ryan'; output;
    name='MaRk T. Smith'; output;
  end;
enddata;
run;
quit;

proc ds2;
data test(overwrite=yes);
  dcl char(13) ucname;
  method run();
    set names;
    ucname=upcase(name);
    put ucname;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
SARAH
MYLINDA
RYAN
MARK T. SMITH
```

関連項目

関数:

- “LOWCASE 関数” (833 ページ)

URLDECODE 関数

URL のエスケープ構文を使用してデコードされた文字列を返します。

カテゴリ: Web ツール

構文

URLDECODE(*expression*)

必須引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

詳細

基本

URL のエスケープ構文は、URL に使用されると意味を持つ可能性のある文字を非表示にするために使用されます。

URL のエスケープシーケンスは、次のいずれです。

- プラス記号。空白で置き換えられます。
- パーセント記号で始まり、その後に 2 つの 16 進文字が続く 3 つの文字のシーケンス。このパターンは、指定された 16 進数値を持つ単一の文字に置き換えられます。

expression は、SAS セッションエンコーディングまたは UTF-8 エンコーディングのいずれかを使用してデコードできます。

文字確認

URLDECODE および URLENCODE 関数では、エスケープシーケンスによって 生成されたバイトがエンコーディングに基づく有効な文字かどうかは確認されません。

返される変数の長さ

まだ長さが割り当てられていない変数に URLDECODE 関数から値が返される場合、その変数には引数の長さが割り当てられます。

例

次のプログラムは、3 つの URL 文字列をデコードします。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl varchar(15) x1 x2 x3;
  method run();
    x1=urldecode('abc+def');
    x2=urldecode('why%3F');
    x3=urldecode('%41%42%43%23%31');
    put x1= x2= x3=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
x1=abc def x2=why? x3=ABC#1
```

関連項目

関数:

- “URLENCODE 関数” (1227 ページ)

URLENCODE 関数

URL のエスケープ構文を使用してエンコードされた文字列を返します。

カテゴリ: Web ツール

構文

URLENCODE(*expression*)

必須引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類　CHAR、NCHAR、NVARCHAR、VARCHAR

詳細

基本

expression は、SAS セッションエンコーディングまたは UTF-8 エンコーディングのいずれかを使用してエンコードできます。

URLENCODE 関数は、URL に使用されると意味を持つ可能性のある文字をエンコードします。この関数は、次を除くすべての文字をエンコードします。

- すべての英数字
- ドル記号($)
- ハイフン(-)
- アンダースコア(_)
- アットマーク(@)
- ピリオド(.)
- 感嘆符(!)
- アスタリスク(*)
- 開始かっこ(( )と閉じかっこ( ))
- カンマ(,)

注: エンコードされた文字列は、元の文字列よりも長くなる可能性があります。この関数を使用する場合は、長さを追加することを検討する必要があります。

文字確認

URLDECODE および URLENCODE 関数では、エスケープシーケンスによって 生成されたバイトがエンコーディングに基づく有効な文字かどうかは確認されません。

例

次のプログラムは、URL 文字列をエンコードします。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl varchar(7) x1;
  dcl varchar(9) x2;
```

```
  method run();
    x1=urlencode('abc def');
    put x1=;
    x2=urlencode('abc def');
    put x2=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
x1=abc%20d
x2=abc%20def
```

関連項目

関数:

- “URLDECODE 関数” (1226 ページ)

USS 関数

無修正平方和を返します。

カテゴリ:	記述統計
返されるデータの種類:	DOUBLE

構文

USS(*expression* [, ...*expression-n*])

必須引数

expression
数値に評価される有効な式を指定します。

要件	少なくとも 1 つの非 null または非欠損の引数が必要です。それがない場合は、関数から null または欠損値が返されます。
データの種類	DOUBLE
参照項目:	“DS2 式” (*SAS DS2 プログラマガイド*)

例

次のプログラムは、USS 関数を示しています。

```
proc ds2;
data values(overwrite=yes);
  dcl double x1 x2 x3 x4;
  method run();
    x1=4;
    x2=2;
    x3=3.5;
    x4=6;
    output;
  end;
enddata;
run;
quit;

proc ds2;
data new (overwrite=yes);
  dcl double val1 val2 val3;
  method run();
    set values;
    val1=uss(of x1-x4);
    put val1;
    val2=uss(of x1-x4, .);
    put val2;
    val3=uss(of val1-val2);
    put val3;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
68.25
68.25
9316.125
```

関連項目

関数:

- “CSS 関数” (484 ページ)

UUIDGEN 関数

ユニバーサル一意識別子(UUID)の短い形式を返します。

カテゴリ:	特殊
返されるデータの種類:	CHAR、NCHAR、NVARCHAR、VARCHAR

構文

UUIDGEN( )

引数なし

UUIDGEN 関数には引数がありません。

詳細

UUIDGEN 関数は、各呼び出しに対して UUID (固有値)を返します。デフォルトの結果は、次のように 36 文字になります。

```
5ab6fa40–426b-4375–bb22–2d0291f43319
```

例

次のプログラムは UUID を返します。36 文字の変数宣言が必要であることに注意してください。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl char(36) x;
  method run();
    x=uuidgen();
    put x;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。各 UUID は一意です。

```
25C752D5-AFA1-4932-BEE6-39E4006C2AAB
```

関連項目

その他のリファレンス:

- “ユニバーサル一意識別子” (*SAS プログラマガイド: エッセンシャル*)

VAR 関数

分散を返します。

カテゴリ:	記述統計
返されるデータの種類:	DOUBLE

構文

VAR(*expression-1*, *expression-2* [,...*expression-n*])

必須引数

expression
数値に評価される有効な式を指定します。引数リストは変数リストで構成することができます。

要件	少なくとも 2 つの非 null または非欠損の引数が必要です。それがない場合は、関数から null または欠損値が返されます。
データの種類	DOUBLE
参照項目:	“DS2 式” (*SAS DS2 プログラマガイド*)

例

次のプログラムは、VAR 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data new (overwrite=yes);
  vararray double a[4];
  dcl double x1 x2 x3;

  method init();
```

```
    a:=(4, 2, 3.5, 6);
  end;

  method run();
    x1=var(of a1-a4);
    put x1;
    x2=var(of a1, a4, .);
    put x2;
    x3=var(of x1-x2);
    put x3;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
2.72916666666666
2
0.26584201388888
```

VERIFY 関数

式に固有の最初の文字の位置を返します。

カテゴリ: 文字

返されるデータの種類: DOUBLE

構文

VERIFY(*target-expression*, *search–expression*)

必須引数

target-expression
文字列に評価されるか強制変換できる検索先の有効な式を指定します。

要件 リテラル文字列は、単一引用符で囲む必要があります。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

search-expression
文字列に評価されるか強制変換できる有効な式を指定します。

要件 リテラル文字列は、単一引用符で囲む必要があります。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

VERIFY 関数は、*target-expression* における *search-expression* に存在しない最初の文字の位置を返します。*target-expression* に *search-expression* と異なる文字がない場合、VERIFY は 0 を返します。

比較

INDEX 関数は、*target-expression* に存在する *search-expression* の最初の出現位置を返し、VERIFY 関数は、*target-expression* における *search-expression* が含まれていない最初の文字の位置を返します。

例

次のプログラムは、VERIFY 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double z;
  dcl char x y;
  method run();
    x='abc';
    y='abcdef';
    z=verify(y,x);
    put z;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
4
```

```
proc ds2;
data test(overwrite=yes);
  dcl double z;
  dcl char x y;
  method run();
```

```
    x='abcdef';
    y='abcdef';
    z=verify(y,x);
    put z;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
0
```

関連項目

関数:

- “INDEX 関数” (705 ページ)

VFORMAT 関数

指定したグローバル変数に関連付けられた出力形式を返します。

カテゴリ:	変数情報
返されるデータの種類:	CHAR、NCHAR、NVARCHAR、VARCHAR
注:	構文規則の角かっこは、オプションの引数を示します。角かっこの前のエスケープ文字(\ )は、構文に角かっこが必要であることを示します。*variable-array* および *varlist* のインデックス要素は、角かっこ([ ])で囲む必要があります。
参照項目:	“添字演算子” (2119 ページ)

構文

VFORMAT(*variable* | *variable-array*\[*i* \] | *varlist*\[*i* \])

必須引数

variable
グローバルスカラー変数の名前を指定します。

制限事項	式は引数として使用できません。
データの種類	すべてのデータ型

variable-array
変数配列の名前を指定します。

参照項目: “変数配列” (*SAS DS2 プログラマガイド*)

varlist
varlist の名前を指定します。

参照項目: “DS2 varlist” (*SAS DS2 プログラマガイド*)

オプション引数

i
指定された varlist または変数配列内の変数の要素番号を指定します。

要件 *i* は角かっこ([ ])で囲む必要があります。

詳細

VFORMAT は完全な出力形式名を返します。この名前には、幅とピリオドが含まれます($CHAR20.など)。

例

次のプログラムは、文字変数の出力形式を返します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl varchar(7) str having format $char7.;
  dcl varchar a;
  method run();
    a = vformat(str);
    put a=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
a=$char7.
```

関連項目

関数:

- “VINARRAY 関数” (1239 ページ)
- “VINFORMAT 関数” (1241 ページ)
- “VGETVALUE 関数” (1237 ページ)
- “VLABEL 関数” (1243 ページ)
- “VLENGTH 関数” (1245 ページ)
- “VNAME 関数” (1247 ページ)
- “VSETVALUE 関数” (1252 ページ)
- “VTYPE 関数” (1253 ページ)

VGETVALUE 関数

指定された変数の値を取得し、それを別の変数に割り当てます。

カテゴリ: 変数の割り当て

返されるデータの種類: INTEGER

参照項目: “添字演算子” (2119 ページ)

構文

VGETVALUE (*source-variable*, *destination-variable*)

必須引数

source-variable
データ値を取得するグローバル変数を指定します。グローバルスカラー変数、変数配列の要素、または varlist の要素は有効な引数です。変数配列および varlist 要素は、添字演算子を使用して指定する必要があります。

制限事項 ローカル変数、ドット演算子でアクセスされるパッケージ属性、一時配列の要素、式は、*source-variable* としてサポートされていません。

データの種類 すべてのデータ型

destination-variable
取得したデータ値を書き込む変数を指定します。ローカル変数またはグローバル変数、一時配列または変数配列の要素、または varlist の要素は有効な引数で

す。配列要素および varlist 要素は、添字演算子を使用して指定する必要があります。

制限事項　式は引数として使用できません。

データの種類　すべてのデータ型

詳細

VGETVALUE 関数は、*source-variable* の値を取得し、その値を *destination-variable* の型に変換し、変換された値を *destination-variable* に割り当てます。*source-variable* と *destination-variable* が同じ型の場合、データ変換は行われません。

VGETVALUE によって取得された値は、varlist 型ではなくビルトインデータ型です。たとえば、varlist の最初の要素が DOUBLE 変数の場合、次の要求は、varlist の最初の要素から DOUBLE データ値を取得し、取得した値を変数 x の型に変換し、変換した値を変数 x に書き込みます。

```
VGETVALUE(varlist_name[1], x)
```

VGETVALU 関数はステータスインジケーターを返します。最初の引数の値が *destination-variable* に正常に割り当てられると、0 (ゼロ)が返されます。エラーが発生した場合、*destination-variable* 変数には SAS 欠損値または ANSI null が割り当てられ、ゼロ以外の値が返されます。

DS2 プログラムでの VGETVALUE 関数の使用例については、“例: varlist のデータ値の取得” (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

概念:

- “変数配列” (*SAS DS2 プログラマガイド*)
- “DS2 varlist” (*SAS DS2 プログラマガイド*)

関数:

- “VINARRAY 関数” (1239 ページ)
- “VFORMAT 関数” (1235 ページ)
- “VINFORMAT 関数” (1241 ページ)
- “VLABEL 関数” (1243 ページ)
- “VLENGTH 関数” (1245 ページ)
- “VSETVALUE 関数” (1252 ページ)

VINARRAY 関数

指定した変数が配列のメンバーかどうかを示す値を返します。

カテゴリ: 変数情報

返されるデータの種類: INTEGER

注: 構文規則の角かっこは、オプションの引数を示します。角かっこの前のエスケープ文字(\ )は、構文に角かっこが必要であることを示します。*variable-array* および *varlist* のインデックス要素は、角かっこ([ ])で囲む必要があります。

参照項目: “添字演算子” (2119 ページ)

構文

VINARRAY(*variable* | *variable-array*\[*i* \] | *varlist*\[*i* \])

必須引数

variable
グローバルスカラー変数の名前を指定します。

制限事項 式は引数として使用できません。

データの種類 すべてのデータ型

variable-array
変数配列の名前を指定します。

参照項目: “変数配列” (*SAS DS2 プログラマガイド*)

varlist
varlist の名前を指定します。

参照項目: “DS2 varlist” (*SAS DS2 プログラマガイド*)

オプション引数

i
指定された変数配列または名前付き varlist の要素番号を指定します。

注 *i* は角かっこ([ ])で囲む必要があります。

詳細

指定された変数が変数配列のメンバーである場合、VINARRAY は値 1 を返します。それ以外の場合、VINARRAY は値 0 を返します。

例

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

例 1: 変数テスト

次のプログラムは、指定された変数が配列のメンバーであるかどうかに応じて、値 (または 1)を返します。

```
proc ds2;
data _null_;
  dcl  int a b c d;
  vararray int x[3] a b c;
  method init();
    dcl int c_in;
    dcl int d_in;
    c_in=vinarray(c);  /* c is in array x */
    d_in=vinarray(d);  /* d is not in an array */
    put c_in=;
    put d_in=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
c_in=1
d_in=0
```

例 2: vrlist テスト

次のプログラムは、指定された変数が配列のメンバーであるかどうかに応じて、値 (または 1)を返します。

```
proc ds2;
data _null_;
  dcl double x y;
  varlist lst [x];
  method init();
  if (VINARRAY(x)) then put 'x in array';
    else put 'x not in array';

  if (VINARRAY(y)) then put 'y in array';
    else put 'y not in array';
```

```
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
x not in array
y not in array
```

関連項目

関数:

- “VGETVALUE 関数” (1237 ページ)
- “VFORMAT 関数” (1235 ページ)
- “VINFORMAT 関数” (1241 ページ)
- “VLABEL 関数” (1243 ページ)
- “VLENGTH 関数” (1245 ページ)
- “VNAME 関数” (1247 ページ)
- “VSETVALUE 関数” (1252 ページ)
- “VTYPE 関数” (1253 ページ)

VINFORMAT 関数

指定した変数に関連付けられた入力形式を返します。

カテゴリ:	変数情報
返されるデータの種類:	CHAR、NCHAR、NVARCHAR、VARCHAR
注:	構文規則の角かっこは、オプションの引数を示します。角かっこの前のエスケープ文字(\ )は、構文に角かっこが必要であることを示します。*variable-array* および *varlist* のインデックス要素は、角かっこ([ ])で囲む必要があります。
参照項目:	“添字演算子” (2119 ページ)

構文

VINFORMAT(*variable* | *variable-array*\[*i* \] | *varlist*\[*i* \])

必須引数

variable
グローバルスカラー変数の名前を指定します。

制限事項　式は引数として使用できません。

データの種類　すべてのデータ型

variable-array
変数配列の名前を指定します。

参照項目:　“変数配列” (*SAS DS2 プログラマガイド*)

varlist
varlist の名前を指定します。

参照項目:　“DS2 varlist” (*SAS DS2 プログラマガイド*)

オプション引数

i
指定された varlist または変数配列内の変数の要素番号を指定します。

要件　*i* は角かっこ([ ])で囲む必要があります。

詳細

VINFORMAT は完全な入力形式名を返します。この名前には、幅とピリオドが含まれます($CHAR20.など)。

例

次のプログラムは、指定した変数に関連付けられた入力形式を返します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl varchar(10) str having informat $char5.;
  method run();
    a=vinformat(str);
    put a=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
a=$char5.
```

関連項目

関数:

- “VINARRAY 関数” (1239 ページ)
- “VGETVALUE 関数” (1237 ページ)
- “VFORMAT 関数” (1235 ページ)
- “VLABEL 関数” (1243 ページ)
- “VLENGTH 関数” (1245 ページ)
- “VNAME 関数” (1247 ページ)
- “VTYPE 関数” (1253 ページ)

VLABEL 関数

指定した変数に関連付けられているラベルを返します。

カテゴリ:	変数情報
返されるデータの種類:	CHAR、NCHAR、NVARCHAR、VARCHAR
注:	構文規則の角かっこは、オプションの引数を示します。角かっこの前のエスケープ文字(\ )は、構文に角かっこが必要であることを示します。*variable-array* および *varlist* のインデックス要素は、角かっこ([ ])で囲む必要があります。
参照項目:	“添字演算子” (2119 ページ)

構文

VLABEL(*variable* | *variable-array*\[*i* \] | *varlist*\[*i* \])

必須引数

variable
グローバルスカラー変数の名前を指定します。

データの種類 すべてのデータ型

variable-array
変数配列の名前を指定します。

参照項目: “変数配列” (*SAS DS2 プログラマガイド*)

varlist
varlist の名前を指定します。

参照項目: “DS2 varlist” (*SAS DS2 プログラマガイド*)

オプション引数

i
指定された varlist または変数配列内の変数の要素番号を指定します。

要件 *i* は角かっこ([ ])で囲む必要があります。

詳細

VLABEL は、指定した変数に関連付けられたラベルを返します。ラベルが存在しない場合、VLABEL は変数名を返します。

例

次のプログラムは、指定された変数のラベルを返します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl varchar(10) fname having label 'First Name';
  method run();
    a=vlabel(fname);
    put a=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
a=First Name
```

関連項目

関数:

- “VINARRAY 関数” (1239 ページ)

- “VFORMAT 関数” (1235 ページ)
- “VGETVALUE 関数” (1237 ページ)
- “VINFORMAT 関数” (1241 ページ)
- “VLENGTH 関数” (1245 ページ)
- “VNAME 関数” (1247 ページ)
- “VSETVALUE 関数” (1252 ページ)
- “VTYPE 関数” (1253 ページ)

VLENGTH 関数

指定した変数のサイズを返します。

カテゴリ: 変数情報

返されるデータの種類: BIGINT

注: 構文規則の角かっこは、オプションの引数を示します。角かっこの前のエスケープ文字(\ )は、構文に角かっこが必要であることを示します。*variable-array* および *varlist* のインデックス要素は、角かっこ([ ])で囲む必要があります。

参照項目: “添字演算子” (2119 ページ)

構文

VLENGTH(*variable* | *variable-array*\[*i* \] | *varlist*\[*i* \])

必須引数

variable
グローバルスカラー変数の名前を指定します。

制限事項 式は引数として使用できません。

データの種類 すべてのデータ型

variable-array
変数配列の名前を指定します。

参照項目: “変数配列” (*SAS DS2 プログラマガイド*)

varlist
varlist の名前を指定します。

参照項目: “DS2 varlist” (*SAS DS2 プログラマガイド*)

オプション引数

i

指定された varlist または変数配列内の変数の要素番号を指定します。

要件　*i* は角かっこ([ ])で囲む必要があります。

詳細

数値データ型の長さは、その列のデータ型が使用する最大桁数、またはデータの精度で定義されます。文字型の場合はデータの文字数、バイナリデータ型の場合はデータのバイト数です。TIME、TIMESTAMP、およびすべての 間隔データ型の場合、これはこのデータの文字表現の文字数です。

例

次のプログラムは指定した変数の長さを返します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl char(7) str;
  dcl double a;
  method run();
    str='World';
    a=vlength (str);
    put a=;
  end;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
a=7
```

関連項目

関数:

- “VINARRAY 関数” (1239 ページ)
- “VFORMAT 関数” (1235 ページ)
- “VGETVALUE 関数” (1237 ページ)
- “VINFORMAT 関数” (1241 ページ)

- “VLABEL 関数” (1243 ページ)
- “VNAME 関数” (1247 ページ)
- “VSETVALUE 関数” (1252 ページ)
- “VTYPE 関数” (1253 ページ)

VNAME 関数

指定した変数の名前を返します。

カテゴリ: 変数情報

返されるデータの種類: CHAR、NCHAR、NVARCHAR、VARCHAR

注: 構文規則の角かっこは、オプションの引数を示します。角かっこの前のエスケープ文字(\ )は、構文に角かっこが必要であることを示します。*variable-array* および *varlist* のインデックス要素は、角かっこ([ ])で囲む必要があります。

参照項目: “添字演算子” (2119 ページ)

構文

VNAME(*variable* | *variable-array*\[*i* \] | *varlist*\[*i* \])

必須引数

variable
グローバルスカラー変数の名前を指定します。

制限事項 式は引数として使用できません。

データの種類 すべてのデータ型

variable-array
変数配列の名前を指定します。

参照項目: “変数配列” (*SAS DS2 プログラマガイド*)

varlist
varlist の名前を指定します。

参照項目: “DS2 varlist” (*SAS DS2 プログラマガイド*)

オプション引数

i
指定された varlist または変数配列内の変数の要素番号を指定します。

要件　*i* は角かっこ([ ])で囲む必要があります。

例

次のプログラムは指定した変数の名前を返します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  vararray int x[3] a b c;
  dcl char y;
  method run();
    y=vname(x[1]);
    put y=;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
y=a
```

関連項目

関数:

- “VINARRAY 関数” (1239 ページ)
- “VFORMAT 関数” (1235 ページ)
- “VGETVALUE 関数” (1237 ページ)
- “VINFORMAT 関数” (1241 ページ)
- “VLABEL 関数” (1243 ページ)
- “VLENGTH 関数” (1245 ページ)
- “VSETVALUE 関数” (1252 ページ)
- “VTYPE 関数” (1253 ページ)

VSETMISSING 関数

指定された変数に欠損値または null 値を割り当てます。

カテゴリ: 変数の割り当て

返されるデータの種類: INTEGER

参照項目: “存在しないデータの DS2 モード” (*SAS DS2 プログラマガイド*)

構文

VSETMISSING(*variable-name1*[,...*variable-nameN*])

必須引数

variable-name(s)
カンマ区切りの引数リストで 1 つ以上のグローバル変数の名前を指定します。グローバルスカラー変数、変数配列、変数配列または varlist の要素、および省略変数リストは有効な引数です。引数リストでは、異なる型の引数を混在させることができます。

変数配列および省略変数リストは、OF 演算子を使用して指定する必要があります。次に例を示します。

```
vsetmissing(of _all_)
```

詳細については、"OF 演算子" (2109 ページ)を参照してください。

制限事項 ローカル変数、ドット演算子でアクセスされるパッケージ属性、および一時配列は、引数としてサポートされていません。

データの種類 すべてのデータ型

注 型変数リストと、型変数リスト指定を含む変数配列が引数としてサポートされます。

詳細

VSETMISSING 関数は、DATA ステップ MISSING 呼び出しルーチンと似ていますが、VSETMISSING が DS2 データ型を認識し、SAS モードに加えて ANSI モードをサポートする点が異なります。

VSETMISSING 関数は、変数のデータ型とプログラムが実行されている DS2 モード (ANSI または SAS)に基づいて、引数リスト内の各変数に SAS 欠損値または ANSI null を割り当てます。

- ANSI モードでは、すべてのデータ型の変数に ANSI null が割り当てられます。
- SAS モードでは、DOUBLE 型または固定長 CHARACTER 型の変数には SAS 欠損が割り当てられます。他のすべてのデータ型の変数には、ANSI null が割り当てられます。

VSETMISSING 関数はステータスインジケーターを返します。引数リスト内のすべての関連引数への SAS 欠損または ANSI null の割り当てが成功すると、0 (ゼロ)が返

されます。エラーがある場合は、失敗した引数のエラーメッセージがログに出力され、ゼロ以外の値が返されます。

これらはすべて 有効な式です。

```
vsetmissing(b)
vsetmissing(b, c, d, e)
vsetmissing(z, of a[*], y)
vvsetmissing(y, of s:, z)
vsetmissing(of _all_)
vsetmissing(z, v[1], a[3], y)
vsetmissing(x, y, of double)
vsetmissing(OF x y z a[*])
vsetmissing(OF sum int)
```

注: 引数リスト内の OF 演算子の位置とカンマの使用が柔軟になりました。

例

DS2 プログラムで VSETMISSING 関数を使用する方法の例を次に示します。この例では、省略変数リストを指定した VSETMISSING 関数を使用して、指定されたテーブル列とすべてのテーブル 列から値をクリアします。

注: この例は、DS2 プロシジャを使用して SAS にサブミットされます。

```
proc ds2;
package annual_record;
 dcl double sales_q1 sales_q2 sales_q3 sales_q4 having format 8.2;
 dcl double profit_q1 profit_q2 profit_q3 profit_q4 having format 8.2;

 /* Set the sales package attributes to missing. */
 method clear_sales();
  vsetmissing(of sales:);
 end;

 /* Set the profit package attributes to missing. */
 method clear_profit();
  vsetmissing(of profit:);
 end;

 /* Set all package attributes to missing. */
 method clear_all();
  vsetmissing(of _all_);
 end;

 method print_report();
  put 'QUARTER    SALES    PROFIT';
  put '   1    ' sales_q1 profit_q1;
  put '   2    ' sales_q2 profit_q2;
  put '   3    ' sales_q3 profit_q3;
  put '   4    ' sales_q4 profit_q4;
  put;
```

```
  end;
endpackage;

data _null_;
 method init();
  dcl package annual_record record();

  record.sales_q1 = 1001.50;
  record.sales_q2 = 155.25;
  record.sales_q3 = 405.00;
  record.sales_q4 = 2000.50;

  record.profit_q1 = 24.00;
  record.profit_q2 = 75.00;
  record.profit_q3 = 26.00;
  record.profit_q4 = 100.00;

  record.print_report();
  record.clear_profit();
  record.print_report();
  record.clear_all();
  record.print_report();
 end;
enddata;
run;
quit;
```

SAS ログに書き込まれる出力は次のとおりです。

```
QUARTER   SALES   PROFIT
  1     1001.50   24.00
  2      155.25   75.00
  3      405.00   26.00
  4     2000.50   100.00

QUARTER   SALES   PROFIT
  1     1001.50     .
  2      155.25     .
  3      405.00     .
  4     2000.50     .

QUARTER   SALES   PROFIT
  1         .       .
  2         .       .
  3         .       .
  4         .       .
```

関連項目

概念:

- “変数配列” (*SAS DS2 プログラマガイド*)
- “DS2 変数リスト” (*SAS DS2 プログラマガイド*)
- “DS2 varlist” (*SAS DS2 プログラマガイド*)

関数:

- “VGETVALUE 関数” (1237 ページ)
- “VSETVALUE 関数” (1252 ページ)

VSETVALUE 関数

変数を指定された値に設定します。

カテゴリ:	変数の割り当て
返されるデータの種類:	INTEGER
ヒント:	変数値をクリアするには、VSETMISSING 関数を使用します。
参照項目:	“添字演算子” (2119 ページ)

構文

VSETVALUE (*destination-variable*, *value*)

必須引数

destination-variable
データ値を書き込むグローバル変数を指定します。グローバルスカラー変数、変数配列の要素、または varlist の要素は有効な引数です。変数配列および varlist 要素は、添字演算子を使用して指定する必要があります。

制限事項	ローカル変数、ドット演算子でアクセスされるパッケージ属性、一時配列の要素、式は、*destination-variable* としてサポートされていません。
データの種類	すべてのデータ型

value
データ値を指定します。スカラー値に評価される式が有効な式です。

データの種類	すべてのデータ型

詳細

VSETVALUE 関数は、*destination-variable* のデータ値を設定します。*destination-variable* は、スカラー変数の名前、変数配列の要素、または varlist の要素にすることができます。

value が *destination-variable* と同じ型ではない場合、スカラー値は *destination-variable* に割り当てられる前に *destination-variable* の型に変換されます。型変換については、“式オペランドの DS2 自動型変換” (*SAS DS2 プログラマガイド*)を参照してください。*value* と *destination-variable* が同じ型の場合、データ変換は行われません。

VGETVALU 関数はステータスインジケーターを返します。*destination-variable* への *value* の割り当てが成功すると、0 (ゼロ)が返されます。エラーが発生した場合、*destination-variable* 変数には SAS 欠損値または ANSI null が割り当てられ、ゼロ以外の値が返されます。

有効な式の例を次に示します。

```
vsetvalue(v[1], 42.42);
vsetvalue(v[2], y);
vsetvalue(v[3], x+y*m())
```

DS2 プログラムでの VSETVALUE 関数の使用例については、“例: varlist のデータ値の設定” (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

概念:

- “変数配列” (*SAS DS2 プログラマガイド*)
- “DS2 Varlist”

関数:

- “VINARRAY 関数” (1239 ページ)
- “VFORMAT 関数” (1235 ページ)
- “VGETVALUE 関数” (1237 ページ)
- “VINFORMAT 関数” (1241 ページ)
- “VLABEL 関数” (1243 ページ)
- “VLENGTH 関数” (1245 ページ)

VTYPE 関数

変数に関連付けられているデータ型のフルネームを返します。

カテゴリ:	変数情報
返されるデータの種類:	CHAR、NCHAR、NVARCHAR、VARCHAR
注:	構文規則の角かっこは、オプションの引数を示します。角かっこの前のエスケープ文字(\ )は、構文に角かっこが必要であることを示します。*variable-array* および *varlist* のインデックス要素は、角かっこ([ ])で囲む必要があります。

参照項目: “添字演算子” (2119 ページ)

構文

VTYPE(*variable* | *variable-array*\[*i* \] | *varlist*\[*i* \])

必須引数

variable
グローバルスカラー変数の名前を指定します。

制限事項 式は引数として使用できません。

データの種類 すべてのデータ型

variable-array
変数配列の名前を指定します。

参照項目: “変数配列” (*SAS DS2 プログラマガイド*)

varlist
varlist の名前を指定します。

参照項目: “DS2 varlist” (*SAS DS2 プログラマガイド*)

オプション引数

i
指定された varlist または変数配列内の変数の要素番号を指定します。

要件 *i* は角かっこ([ ])で囲む必要があります。

詳細

VTYPE は、変数のデータ型のデータ型名を返します。

例

次のプログラムは、指定された変数に関連付けられているデータ型の名前を返します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
```

```
data _null_;
 dcl double d;
 dcl timestamp t;
 dcl nvarchar(10) n;
 method run();
   declare char(20) a b c;
   a=vtype(d);
   put a=;
   b=vtype(t);
   put b=;
   c=vtype(n);
   put c=;
 end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
a=double
b=timestamp
c=nvarchar
```

関連項目

関数:

- “VINARRAY 関数” (1239 ページ)
- “VFORMAT 関数” (1235 ページ)
- “VINFORMAT 関数” (1241 ページ)
- “VLABEL 関数” (1243 ページ)
- “VLENGTH 関数” (1245 ページ)
- “VNAME 関数” (1247 ページ)

WEEK 関数

週番号の値を返します。

カテゴリ:	日付と時間
返されるデータの種類:	DOUBLE

構文

WEEK([*sas-date*], ['*descriptor*'])

オプション引数

sas-date
SAS 日付値を指定します。*sas-date* 引数を指定しない場合、WEEK 関数は現在の日付の週番号の値を返します。

データの種類 DOUBLE

descriptor
ディスクリプターの値を指定します。次のディスクリプターを大文字または小文字で指定できます。

U
1 年のうちの週番号を指定します。日曜日が週の第 1 日とみなされます。週番号値は、0 から 53 の範囲の 10 進数として表されます。第 53 週に特別な意味はありません。week('31dec2013'd, 'u')の値は 53 です。U はデフォルト値です。

ヒント U および W ディスクリプターは似ていますが、U ディスクリプターが日曜日を週の第 1 日とみなすのに対し、W ディスクリプターは月曜日を週の第 1 日とみなします。

参照項目: “U ディスクリプター” (1257 ページ)

V
1 から 53 の範囲の 10 進数として表される週番号値を指定します。月曜日が週の第 1 日とみなされ、年の第 1 週は、1 月 4 日とその年の最初の木曜日の両方を含む週です。1 月の最初の月曜日が 2 日、3 日または 4 日の場合、それより前の日は前年の最後の週に組み込まれます。

参照項目: “V ディスクリプター” (1257 ページ)

W
1 年のうちの週番号を指定します。月曜日が週の第 1 日とみなされます。週番号値は、0 から 53 の範囲の 10 進数として表されます。第 53 週に特別な意味はありません。week('31dec2013'd, 'w')の値は 53 です。

ヒント U および W ディスクリプターは似ていますが、U ディスクリプターが日曜日を週の第 1 日とみなすのに対し、W ディスクリプターは月曜日を週の第 1 日とみなします。

参照項目: “W ディスクリプター” (1257 ページ)

デフォルト U

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

注 U、V、W 以外のディスクリプターを指定した場合、引数が無効であることを示すメモが SAS ログに書き込まれます。この関数は NULL を返します。

詳細

基本

WEEK 関数は、SAS 日付値を読み取り、週番号を返します。WEEK 関数はロケールに依存せず、計算にはグレゴリオ暦のみを使用します。

U ディスクリプター

U ディスクリプターを使用する WEEK 関数は、SAS 日付値を読み取り、1 年のうちの週番号を返します。週番号値は、先頭に 0 を付けた 0 から 53 までの範囲の 10 進数として表され、最大値は 53 になります。第 0 週は、週の第 1 日が前年になるということを意味します。年の第 5 週は 05 と表されます。

日曜日が週の第 1 日とみなされます。たとえば、week('01jan2013'd, 'u')は 0 です。

V ディスクリプター

V ディスクリプターを使用する WEEK 関数は、SAS 日付値を読み取り、週番号を返します。週番号値は、01 から 53 の範囲の 10 進数として表されます。10 進数の先頭に 0 を使用し、最大値は 53 です。各週は月曜日から始まり、年の第 1 週は 1 月 4 日とその年の最初の木曜日の両方を含む週です。1 月の最初の月曜日が 2 日、3 日または 4 日の場合、それより前の日は前年の最後の週に組み込まれます。次の例では、01jan2014 と 31dec2013 は同じ週になります。この週の第 1 日(月曜日)は 30dec2013 です。したがって、week('01jan2014'd, 'v')と week('30dec2013'd, 'v')の両方は 53 を返します。つまり、両方の日付が 2013 年の第 53 週に含まれるという意味です。

W ディスクリプター

W ディスクリプターを使用する WEEK 関数は、SAS 日付値を読み取り、1 年のうちの週番号を返します。週番号値は、先頭に 0 を付けた 0 から 53 までの範囲の 10 進数として表され、最大値は 53 になります。第 0 週は、週の第 1 日が前年になるということを意味します。年の第 5 週は 05 と表されます。

月曜日が週の第 1 日とみなされます。したがって、week('30dec2013'd, 'w')の値は 1 です。

ディスクリプターの比較

U はデフォルトのディスクリプターです。範囲は 0 から 53 で、週の第 1 日は日曜日です。V ディスクリプターの範囲は 1 から 53 で、週の第 1 日は月曜日です。W ディスクリプターの範囲は 0 から 53 で、週の第 1 日は月曜日です。

ディスクリプターおよび関連付けられた週を次のリストに示します。

- 第 0 週:

U　現在のグレゴリオ暦年の第 1 週より前の日を示します。

V　該当しません。

W　現在のグレゴリオ暦年の第 1 週より前の日を示します。

- 第 1 週:

U　グレゴリオ暦年の最初の日曜日から始まります。

V　グレゴリオ暦による前年 12 月 29 日から当年 1 月 4 日までの間の月曜日から始まります。ISO 形式の最初の週は、グレゴリオ暦による前年と当年をまたぐことがあります。

W　グレゴリオ暦年の最初の月曜日から始まります。

- 年末の週:

U　1 年の最後の週(第 52 または 53)の日数を 7 日未満にできることを指定します。グレゴリオ暦による連続する 2 年をまたぐ日曜日から土曜日までの期間は、第 52 週と第 0 週、または第 53 週と第 0 週として指定されます。

V　ISO 形式の年の最後の週(第 52 または 53)の日数が 7 日であることを指定します。ただし、ISO 形式の年の最後の週は、グレゴリオ暦の当年と翌年をまたぐことがあります。

W　1 年の最後の週(第 52 または 53)の日数を 7 日未満にできることを指定します。グレゴリオ暦による連続する 2 年をまたぐ月曜日から日曜日までの期間は、第 52 と第 0 週、または第 53 週と第 0 週として指定されます。

例

次のプログラムは、2019 年 3 月 1 日の日付における U、V、W ディスクリプターの値を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double sasdate x y z;
  method run();
    sasdate=to_double(date'2019-03-01');
    x=week(sasdate, 'u');
    y=week(sasdate, 'v');
    z=week(sasdate, 'w');
    put x;
    put y;
    put z;
  end;
enddata;
run;
```

```
quit;
```

次の行が SAS ログに書き込まれます。

```
8
9
8
```

関連項目

関数:

- “INTNX 関数” (755 ページ)

出力形式:

- “WEEKDATEw. 出力形式” (205 ページ)
- “WEEKDATXw. 出力形式” (207 ページ)
- “WEEKDAYw. 出力形式” (209 ページ)

WEEKDAY 関数

SAS 日付値から、曜日に対応する整数を返します。

カテゴリ:	日付と時間
返されるデータの種類:	DOUBLE

構文

WEEKDAY(*expression*)

必須引数

expression
SAS 日付値を表す有効な式を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

WEEKDAY 関数は、曜日を表す整数を生成します(1=日曜日、2=月曜日、...、7=土曜日)。

DS2 での日付値と時間値の処理法については、"DS2 式" (*SAS DS2 プログラマガイド*) を参照してください。

例

次のプログラムは、現在日が日曜日の場合の WEEKDAY 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double sasdate x;
  method run();
    sasdate=to_double(date'2019-02-24');
    x=weekday(sasdate);
    put x;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
1
```

WHICHC 関数

文字列のリストから文字列の最初の位置を返します。

カテゴリ:	文字
返されるデータの種類:	DOUBLE

構文

WHICHC(*search-expression*, *expression-list-item-1*, *expression-list-item-2* [, ...*expression-list-item-n*])

必須引数

search-expression
文字列式のリストと比較される文字列に評価されるか強制変換できる有効な式を指定します。

要件 リテラル文字列は、単一引用符で囲む必要があります。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

expression-list-item
文字列に評価されるか強制変換できて、文字列式のリストのメンバーである有効な式を指定します。

要件 リテラル文字列は、単一引用符で囲む必要があります。

リストには少なくとも 2 つの式が必要です。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

WHICHC 関数は、文字式リストを左から右に検索して、検索式に一致する最初の式を探します。一致するものが見つかった場合、WHICHC は式リスト内のその位置を返します。いずれの式も検索式に一致しない場合、WHICHC は値 0 を返します。

例

次のプログラムでは、'Spain'がリストに 2 回出現します。WHICHC 関数は、リスト内の'Spain'の最初の位置を返します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double position;
  method run();
    position=whichc('Spain', 'Denmark', 'Germany', 'Austria','Spain',
      'China', 'Egypt', 'Spain', 'France');
    put position;
  end;
run;
quit;
```

SAS は次の出力をログに書き込みます。

4

関連項目

関数:

- “WHICHN 関数” (1262 ページ)

WHICHN 関数

数値のリストから数値の最初の位置を返します。

カテゴリ:	数学
返されるデータの種類:	DOUBLE

構文

WHICHN(*search-expression*, *expression-list-item-1*, *expression-list-item-2* [, ...*expression-list-item-n*])

必須引数

search-expression
数値に評価され、数値式のリストと比較される有効な式を指定します。

データの種類	DOUBLE
参照項目:	“DS2 式” (*SAS DS2 プログラマガイド*)

expression-list-item
数値に評価され、リストの一部である有効な式を指定します。

要件	リストには少なくとも 2 つの式が必要です。
データの種類	DOUBLE
参照項目:	“DS2 式” (*SAS DS2 プログラマガイド*)

詳細

WHICHN 関数は、数値式リストを左から右に検索して、検索式に一致する最初の式を探します。一致するものが見つかった場合、WHICHN は式リスト内のその位置を

返します。いずれの式も検索式に一致しない場合、WHICHN は値 0 を返します。WHICHN 関数の引数には、任意の数値データ型を指定できます。

例

次のプログラムでは、4.5 がリストに 2 回出現します。WHICHN 関数は、リスト内の 4.5 の最初の位置を返します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double position;
  method run();
    position=whichn(4.5,7.3, 8.6, 4.5, 4.5, 2.1, 6.4);
    put position;
  end;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
3
```

関連項目

関数:

- “WHICHC 関数” (1260 ページ)

YEAR 関数

SAS 日付値から年を返します。

カテゴリ:	日付と時間
返されるデータの種類:	DOUBLE

構文

YEAR(*date*)

必須引数

date
SAS 日付値を表す有効な式を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

YEAR 関数は、年を表す 4 桁の数値を生成します。

例

次のプログラムは、年が現在の年である場合の YEAR 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double x thdate;
  method run();
    thdate=today();
    x=year(thdate);
    put x;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
2019
```

関連項目

- “DS2 式” (*SAS DS2 プログラマガイド*)

関数:

- “DAY 関数” (500 ページ)
- “MONTH 関数” (863 ページ)

YIELDP 関数

定期的なキャッシュフローストリーム(債権など)の最終利回りを返します。

カテゴリ: 財務

返されるデータの種類: DOUBLE

構文

YIELDP(*A*, *c*, *n*, *K*, k_0, *p*)

必須引数

A
額面価格を指定します。

範囲 $A > 0$

A > 0

データの種類 DOUBLE

c
分数で表す、年間の名目クーポン率を指定します。

範囲 $0 \leq c < 1$

0 ≤ *c* < 1

データの種類 DOUBLE

n
1 年当たりのクーポン数を指定します。

範囲 $n > 0$

n > 0

データの種類 DOUBLE

K
決済日から満期までの残りのクーポン数を指定します。

範囲 $K > 0$

K > 0

データの種類 DOUBLE

k$_0$
決算日から次のクーポンまでの時間を、1 年ごとの分数で指定します。

範囲 $0 < k_0 \leq \frac{1}{n}$

$0 < k_0 \leq 1/n$

データの種類 DOUBLE

p
未払い利息込みの価格を指定します。

範囲 $p > 0$

$p > 0$

データの種類 DOUBLE

詳細

YIELDP 関数は次の関係に基づきます。

$$P = \sum_{k=1}^{K} c(k) \frac{1}{\left(1 + \frac{y}{n}\right)^{t_k}}$$

前述の式には次の関係が適用されます。

- $t_k = nk_0 + k - 1$
- $c(k) = \frac{c}{n}A \quad for\ k = 1, \ldots, K-1$
- $c(K) = \left(1 + \frac{c}{n}\right)A$

YIELDP 関数は、*y* に対して解決します。

例

次のプログラムでは、YIELDP 関数は、額面$1000、年間クーポン率 0.01、年間クーポン数 4、残りクーポン数 14 の債権の最終利回りを返します。決算日から次のクーポン日までの時間は 0.165、未払い利息込みの価格は 800 です。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
```

```
data _null_;
  dcl double y;
  method run();
    y=yieldp(1000,.01,4,14,.165,800);
    put y;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
0.0775031247735
```

YRDIF 関数

指定した日数計算規則に従って 2 つの日付の差を年単位で返します。人の年齢を返します。

カテゴリ: 日付と時間

返されるデータの種類: DOUBLE

構文

YRDIF(*start-date*, *end-date*[, *basis*])

必須引数

start-date
開始日を識別する SAS 日付値を指定します。

データの種類 DOUBLE

end-date
終了日を識別する SAS 日付値を指定します。

データの種類 DOUBLE

オプション引数

basis
SAS で日付の差(人の年齢)が計算される方法を記述する文字定数、変数または式を指定します。有効な文字列は、'30/360' (30 日の月と 360 日の年)、'ACT/ACT'、'ACT/360',、'ACT/365'、'AGE'です。たとえば、'ACT/360'では、年数の計算に日付間の実際の日数を使用します。この値は、各年の実際の日数に関係なく、日数を 360 で除算して計算されます。詳細については、SAS DS2 言語リファレンスを参照してください。

SAS で日付の差(人の年齢)が計算される方法を記述する文字定数、変数または式を指定します。有効な文字列は次のとおりです。

'30/360'
年数の計算で 30 日の月と 360 日の年を指定します。各月または年の実際の日数に関係なく、各月は 30 日、各年は 360 日とみなされます。

別名 '360'

ヒント いずれかの日付が月末になる場合、その日は 30 日の月の最終日として扱われます。

'ACT/ACT'
年数の計算には、日付間の実際の日数を使用します。この値は、365 日の年の日数を 365 で除算して得た値と、366 日の年の日数を 366 で除算して得た値を足して計算されます。

別名 'Actual'

'ACT/360'
年数の計算には、日付間の実際の日数を使用します。この値は、各年の実際の日数に関係なく、日数を 360 で除算して計算されます。

'ACT/365'
年数の計算には、日付間の実際の日数を使用します。この値は、各年の実際の日数に関係なく、日数を 365 で除算して計算されます。

'AGE'
人の年齢が計算されることを指定します。

第 3 引数を指定しない場合、AGE が *basis* のデフォルト値になります。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

詳細

会計アプリケーションでの YRDIF の使用

基本

YRDIF 関数は、第 3 引数 *basis* が存在する場合に、確定利付証券の利息を計算するために使用できます。指定した日数計算規則に従って 2 つの日付の差を返します。

ACT/ACT 基準を使用する計算

ACT/ACT 基準を使用する YRDIF 計算では、365 日の年と 366 日の年の両方が考慮されます。たとえば、n365 が 365 日の年の開始日から終了日までの日数と等しく、n366 が 366 日の年の開始日から終了日までの日数と等しい場合、YRDIF 計算は YRDIF=n365/365.0 + n366/366.0 として計算されます。この計算は、金融関係の文献に記されている一般に理解されている *ACT/ACT* 日数計算基準に対応します。
basis の値には、他に *30/360*、*ACT/360* および *ACT/365* があります。各基準には、

特定の金融商品の利息の支払いを計算するために従う必要がある明確な意味があります。

人の年齢の計算

YRDIF 関数で人の年齢を計算できます。最初の 2 つの引数 *start-date* と *end-date* は必須です。*basis* の値が AGE の場合、YRDIF は年齢を計算します。年齢の計算では、うるう年が考慮されます。*basis* のその他の値は、人の年齢の計算には無効です。

例

例 1: basis に基づいた年の差の計算

次のプログラムでは、YRDIF は *basis* の各オプションに基づいて 2 つの日付の差(年数)を返します。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data test(overwrite=yes);
  dcl double sdate edate y30360 yactact yact360 yact365;
  method run();
    sdate= to_double(date'1998-10-16');
    edate= to_double(date'2010-02-06');
    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=;
    put yactact=;
    put yact360=;
    put yact365=;
  end;
enddata;
run;
quit;
```

SAS は次の結果をログに書き込みます。

```
y30360=11.3055555555555
yactact=11.3095890410958
yact360=11.475
yact365=11.317808219178
```

例 2: 人の年齢の計算

YRDID 関数で 3 つの引数を使用して人の年齢を計算できます。第 3 引数 *basis* の値は AGE にする必要があります。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
data _null_;
  dcl double sdate edate age;
  method run();
    sdate= to_double(date'1998-10-16');
    edate= to_double(date'2010-02-16');
    age=yrdif(sdate, edate, 'AGE');
    put age= 'years';
  end;
enddata;
run;
quit;
```

SAS は次の結果をログに書き込みます。

```
age=11.3369863013698 years
```

関連項目

関数:

- “DATDIF 関数” (492 ページ)

参考文献

“Day count convention.” *Wikipedia*, 2010. URI: Day count convention. Accessed on May 1, 2013..

ISDA International Swaps and Derivatives Association, Inc “EMU and Market Conventions: Recent Developments.” 1998. ISDA. URI: EMU and Market Conventions: Recent Developments.

Mayle, Jan. 1994. *Standard Securities Calculation Methods – Fixed Income Securities Formulas for Analytic Measures*. 2 巻 New York, , New York: Securities Industry Association.

YYQ 関数

年と四半期の値から SAS 日付値を返します。

カテゴリ:	日付と時間
返されるデータの種類:	DOUBLE

構文

YYQ(*year*, *quarter*)

必須引数

year
年を表す 2 桁または 4 桁の整数に評価される有効な式を指定します。

操作 YEARCUTOFF=システムオプションは、2 桁の日付の年の値を定義します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

quarter
年の四半期(1、2、3、4)を指定します。

データの種類 DOUBLE

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

YYQ 関数は、指定した四半期の最初の日に対応する SAS 日付値を返します。*year* または *quarter* のいずれかが null または欠損している場合、または四半期の値が無効である場合、結果は null 値または欠損値になります。

例

次のプログラムは、YYQ 関数を示しています。

注: この例では、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
  data test(overwrite=yes);
  dcl double DateValue Date7Value Date9Value;
  method run();
    DateValue=yyq(2019,3);
    put DateValue;
    put DateValue date7.;
    put DateValue date9.;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
21731
01JUL19
01JUL2019
```

```
proc ds2;
data test(overwrite=yes);
  dcl double StartOfQuarter;
  method run();
    StartOfQuarter=yyq(2019,4);
    put StartOfQuarter;
    put StartOfQuarter date9.;
  end;
enddata;
run;
quit;
```

SAS は次の出力をログに書き込みます。

```
21823
01OCT2019
```

関連項目

概念:

- “DS2 の日付と時間” (*SAS DS2 プログラマガイド*)

関数:

- “QTR 関数” (1051 ページ)
- “YEAR 関数” (1263 ページ)

ZIPCITY 関数

ZIP コードに対応する都市名と 2 文字の郵便番号を返します。

カテゴリ:	州と ZIP コード
制限事項:	この関数は CAS サーバーではサポートされていません。
返されるデータの種類:	CHAR

構文

ZIPCITY(*ZIP-code*)

必須引数

ZIP-code
5 桁の ZIP コードを含む数値式または文字式を指定します。

データの種類 DOUBLE、CHAR(n)

ヒント *ZIP-code* の値が先頭のゼロで始まる場合、先頭のゼロなしで値を入力できます。たとえば、1040 と入力すると、ZIPCITY は値が 01040 であると想定します。

詳細

基本

ZIPCITY は、都市名と、5 桁の ZIP コード引数に対応する 2 文字の郵便番号を返します。ZIPCITY は、大文字と小文字が混在した文字値を返します。

ZIP コードが不明な場合、ZIPCITY は空白値を返します。まだ長さが割り当てられていない変数に ZIPCITY 関数から値が返される場合、その変数にはデフォルトで長さ 20 が割り当てられます。

注: この関数を使用するときは、Sashelp.Zipcode データセットが存在する必要があります。データセットを削除すると、ZIPCITY は予期しない結果を返します。

ZIP コードを州郵便番号に変換する方法

特定の ZIP コードに対応する州を特定するために、この関数は、各州の開始と終了の ZIP コードの値で構成されるゾーンテーブルを使用します。関数は、その ZIP コードの範囲に対応する州を見つけます。各州の開始と終了の ZIP コードの値では、例外が許可されます。

ごくわずかな例外を除いて、ゾーンは複数の州にまたがることはありません。例外はゾーンテーブルに含まれています。米国郵政公社によっていつでも新しいゾーンまたは新しい例外が追加される可能性があります。ただし、SAS ソフトウェアは、製品の新しいリリースごとにのみ更新されます。

州郵便番号テーブルが最後に更新された時期の特定

Sashelp.Zipcode データセットには、ZIPCITY およびその他の ZIP コード関数で使用される郵便番号情報が含まれています。このデータセットが最後に更新されたのはいつかを判断するには、PROC CONTENTS を実行します。

```
proc contents data=sashelp.zipcode;
run;
```

CONTENTS プロシジャからの出力は、Sashelp.Zipcode データセットのコンテンツとともに、最終更新の日付を提供します。

注: Sashelp.Zipcode ファイルの最新バージョンは、テクニカルサポート Web サイトからダウンロードできます。Name 列から **Zipcode Data Set** を選択して、ダウンロードプロセスを開始します。データセットをダウンロードして解凍した後、CIMPORT プロシジャを実行する必要があります。

無効な ZIP コード(Sashelp.Zipcode データセットにないもの)を使用すると、ZIPCITY 関数に無効な引数があることを示すメッセージが SAS から返されます。

比較

ZIPCITY、ZIPNAME、ZIPNAMEL、ZIPSTATE 関数は同じ引数を受け入れますが、異なる値を返します。

- ZIPCITY は、大文字と小文字が混在する都市名と、5 桁の ZIP コード引数に対応する 2 文字の郵便番号を返します。
- ZIPNAME は、5 桁の ZIP コード引数に対応する州または米国領の大文字の名前を返します。
- ZIPNAMEL は、5 桁の ZIP コード引数に対応する州または米国領の大文字と小文字が混在する名前を返します。
- ZIPSTATE は、5 桁の ZIP コード引数に対応する大文字の 2 文字の州郵便番号(または米国領の場合は世界の GSA 地理コード)を返します。

例

この例では、都市名と、ZIP コードに対応する 2 文字の州郵便番号が返されます。

```
proc ds2;
 data _null_;
  method run();
  dcl char(50) city;
  city=zipcity(27513);
  put city;
  end;
 enddata;
run;
quit;
```

これらのステートメントは、次の結果を生成します。

```
Cary, NC
```

関連項目

関数:

- “ZIPCITYDISTANCE 関数” (1275 ページ)

- “ZIPFIPS 関数” (1276 ページ)
- “ZIPNAME 関数” (1278 ページ)
- “ZIPNAMEL 関数” (1280 ページ)
- “ZIPSTATE 関数” (1281 ページ)

ZIPCITYDISTANCE 関数

2 つの ZIP コード位置間の測地距離を返します。

カテゴリ:	距離 州と ZIP コード
別名:	ZIPCITYDIST
制限事項:	この関数は CAS サーバーではサポートされていません。
返されるデータの種類:	DOUBLE

構文

ZIPCITYDISTANCE(*ZIP-code-1*, *ZIP-code-2*)

必須引数

ZIP-code
アメリカ合衆国内の場所の ZIP コードを含む数値式または文字式を指定します。

データの種類 CHAR(n)、DOUBLE

詳細

ZIPCITYDISTANCE 関数は、2 つの ZIP コードの場所の間の測地距離をマイル単位で返します。各 ZIP コードの重心が計算に使用されます。

この関数を使用するときは、Sashelp.Zipcode データセットが存在する必要があります。データセットを削除すると、ZIPCITYDISTANCE は予期しない結果を返します。

Sashelp.Zipcode データセットには、ZIPCITYDISTANCE およびその他の ZIP コード関数で使用される郵便番号情報が含まれています。このデータセットが最後に更新されたのはいつかを判断するには、PROC CONTENTS を実行します。

```
proc contents data=sashelp.zipcode;
run;
```

CONTENTS プロシジャからの出力は、SASHELP.ZIPCODE データセットのコンテンツとともに、最終更新の日付を提供します。

注: Sashelp.Zipcode ファイルの最新バージョンは、SAS 外部 Web サイトからダウンロードできます。このファイルは、テクニカルサポート Web サイトにあります。**Name** 列から **Zipcode Data Set** を選択して、ダウンロードプロセスを開始します。データセットをダウンロードして解凍した後、CIMPORT プロシジャを実行する必要があります。詳細については、Zipcode データセットの更新を参照してください。

無効な ZIP コード(Sashelp.Zipcode データセットにないもの)を使用すると、ZIPCITYDISTANCE 関数に無効な引数があることを示すメッセージが SAS から返されます。

例

次の例では、最初の ZIP コードは San Francisco, CA の場所を識別し、2 番目の ZIP コードは Bangor, ME の場所を識別します。ZIPCITYDISTANCE は、これら 2 つの場所の間の距離をマイル単位で返します。

```
proc ds2;
 data _null_;
  method run();
  dcl double dist;
  dist=zipcitydist(94103, 04401);
  put 'Miles between San Francisco, CA, to Bangor, ME: ' dist;
  end;
 enddata;
run;
quit;
```

これらのステートメントは、次の結果を生成します。

```
Miles between San Francisco, CA, to Bangor, ME:  2782.3
```

関連項目

関数:

- “ZIPCITY 関数” (1272 ページ)

ZIPFIPS 関数

ZIP コードを 2 桁の FIPS コードに変換します。

カテゴリ: 州と ZIP コード

制限事項: この関数は CAS サーバーではサポートされていません。

返されるデータの種類: DOUBLE

構文

ZIPFIPS(*ZIP-code*)

必須引数

ZIP-code
5 桁の ZIP コードを含む数値式または文字式を指定します。

データの種類 CHAR(n)、DOUBLE

ヒント *ZIP-code* の値が先頭のゼロで始まる場合、先頭のゼロなしで値を入力できます。たとえば、1040 と入力すると、ZIPFIPS は値が 01040 であると想定します。

詳細

基本

ZIPFIPS 関数は、5 桁の ZIP コード引数に対応する 2 桁の数値の FIPS (U.S. Federal Information Processing Standards)コードを返します。

例

この例では、Cary, NC の ZIP コードを 2 桁の FIPS コードに変換します。

```
proc ds2;
 data _null_;
  method run();
  dcl double statefips;
  statefips=zipfips(27511);
  put statefips=;
  end;
 enddata;
run;
quit;
```

これらのステートメントは、次の結果を生成します。

```
statefips=37
```

関連項目

関数:

- “ZIPCITY 関数” (1272 ページ)
- “ZIPNAME 関数” (1278 ページ)
- “ZIPNAMEL 関数” (1280 ページ)
- “ZIPSTATE 関数” (1281 ページ)

ZIPNAME 関数

ZIP コードを大文字の州名に変換します。

カテゴリ:	州と ZIP コード
制限事項:	この関数は CAS サーバーではサポートされていません。
返されるデータの種類:	CHAR

構文

ZIPNAME(*ZIP-code*)

必須引数

ZIP-code
5 桁の ZIP コードを含む数値式または文字式を指定します。

データの種類	CHAR(n)、DOUBLE
ヒント	*ZIP-code* の値が先頭のゼロで始まる場合、先頭のゼロなしで値を入力できます。たとえば、1040 と入力すると、ZIPNAME は値が 01040 であると想定します。

詳細

基本

ZIPNAME は、5 桁の ZIP コード引数に対応する州または米国領の名前を返します。ZIPNAME は、最大 20 文字の長さの文字値をすべて大文字で返します。

まだ長さが割り当てられていない変数に ZIPNAME 関数から値が返される場合、その変数にはデフォルトで長さ 20 が割り当てられます。

ZIP コードを州郵便番号に変換する方法

特定の ZIP コードに対応する州を特定するために、この関数は、各州の開始と終了の ZIP コードの値で構成されるゾーンテーブルを使用します。関数は、その ZIP コードの範囲に対応する州を見つけます。各州の開始と終了の ZIP コードの値では、例外が許可されます。

ごくわずかな例外を除いて、ゾーンは複数の州にまたがることはありません。例外はゾーンテーブルに含まれています。米国郵政公社によっていつでも新しいゾーンまたは新しい例外が追加される可能性があります。ただし、SAS ソフトウェアは、製品の新しいリリースごとにのみ更新されます。

比較

ZIPNAME 関数と ZIPNAMEL 関数はどちらも、郵便番号を受け取り、州名を返します。ZIPNAME は、5 桁の ZIP コード引数に対応する州または米国領の大文字の名前を返します。ZIPNAMEL は、5 桁の ZIP コード引数に対応する州または米国領の大文字と小文字が混在する名前を返します。

例

この例は、ZIPNAME と ZIPNAMEL の違いを示しています。

```
proc ds2;
 data _null_;
  method run();
  dcl char(15) state;
  state=zipname(27511);
  put state=;
  state=zipnamel(27511);
  put state=;
  end;
 enddata;
run;
quit;
```

これらのステートメントは、次の結果を生成します。

```
state=NORTH CAROLINA
state=North Carolina
```

関連項目

関数:

- “ZIPCITY 関数” (1272 ページ)
- “ZIPFIPS 関数” (1276 ページ)
- “ZIPNAMEL 関数” (1280 ページ)

■ “ZIPSTATE 関数” (1281 ページ)

ZIPNAMEL 関数

ZIP コードを大文字と小文字が混在する州名に変換します。

カテゴリ:	州と ZIP コード
制限事項:	この関数は CAS サーバーではサポートされていません。
返されるデータの種類:	CHAR

構文

ZIPNAMEL(*ZIP-code*)

必須引数

ZIP-code
5 桁の ZIP コードを含む数値式または文字式を指定します。

データの種類	CHAR(n)、DOUBLE
ヒント	*ZIP-code* の値が先頭のゼロで始まる場合、先頭のゼロなしで値を入力できます。たとえば、1040 と入力すると、ZIPNAMEL は値が 01040 であると想定します。

詳細

基本

まだ長さが割り当てられていない変数に ZIPNAMEL 関数から値が返される場合、その変数にはデフォルトで長さ 20 が割り当てられます。

ZIPNAMEL は、5 桁の ZIP コード引数に対応する州または米国領の名前を返します。ZIPNAMEL は、大文字と小文字が混在する最大 20 文字の値を返します。

ZIP コードを州郵便番号に変換する方法

特定の ZIP コードに対応する州を特定するために、この関数は、各州の開始と終了の ZIP コードの値で構成されるゾーンテーブルを使用します。関数は、その ZIP コードの範囲に対応する州を見つけます。各州の開始と終了の ZIP コードの値では、例外が許可されます。

ごくわずかな例外を除いて、ゾーンは複数の州にまたがることはありません。例外はゾーンテーブルに含まれています。米国郵政公社によっていつでも新しいゾーンまたは新しい例外が追加される可能性があります。ただし、SAS ソフトウェアは、製品の新しいリリースごとにのみ更新されます。

例

この例では、Cary, NC の郵便番号を大文字と小文字が混在する州名に変換します。

```
proc ds2;
 data _null_;
  method run();
  dcl char(15) state;
  state=zipnamel(27511);
  put state=;
  end;
 enddata;
run;
quit;
```

これらのステートメントは、次の結果を生成します。

```
state=North Carolina
```

関連項目

関数:

- “ZIPCITY 関数” (1272 ページ)
- “ZIPFIPS 関数” (1276 ページ)
- “ZIPNAME 関数” (1278 ページ)
- “ZIPSTATE 関数” (1281 ページ)

ZIPSTATE 関数

ZIP コードを 2 文字の州郵便番号に変換します。

カテゴリ:	州と ZIP コード
制限事項:	この関数は CAS サーバーではサポートされていません。
返されるデータの種類:	CHAR(2)

構文

ZIPSTATE(*ZIP-code*)

必須引数

ZIP-code
有効な 5 桁の ZIP コードを含む数値式または文字式を指定します。

ヒント *ZIP-code* の値が先頭のゼロで始まる場合、先頭のゼロなしで値を入力できます。たとえば、1040 と入力すると、ZIPSTATE は値が 01040 であると想定します。

詳細

基本

まだ長さが割り当てられていない変数に ZIPSTATE 関数から値が返される場合、その変数にはデフォルトで長さ 20 が割り当てられます。

ZIPSTATE は、5 桁の ZIP コード引数に対応する 2 文字の州郵便番号(または米国領の場合は世界の GSA 地理コード)を返します。ZIPSTATE は文字値を大文字で返します。

注: ZIPSTATE は ZIP コードを検証しません。

ZIP コードを州郵便番号に変換する方法

特定の ZIP コードに対応する州を特定するために、この関数は、各州の開始と終了の ZIP コードの値で構成されるゾーンテーブルを使用します。関数は、その ZIP コードの範囲に対応する州を見つけます。ゾーンテーブルは、ZIP コードの値を検証しません。

通常、ゾーンは複数の州にまたがることはありません。例外はゾーンテーブルに含まれています。米国郵政公社によっていつでも新しいゾーンまたは新しい例外が追加される可能性があります。ただし、SAS ソフトウェアは、製品の新しいリリースごとにのみ更新されます。

陸軍郵便局(APO)および艦隊郵便局(FPO)の郵便番号

ZIPSTATE 関数は、APO および FPO ZIP コードを認識します。これらの軍の ZIP コードは、米国内の出国基地に対応しています。ZIP コードは、Sashelp.Zipmil データセットに含まれています。このデータセットが最後に更新されたのはいつかを判断するには、PROC CONTENTS を実行します。

```
proc contents data=sashelp.zipmil;
run;
```

CONTENTS プロシジャからの出力は、最終更新の日付と Sashelp.Zipmil データセットのコンテンツを提供します。

注: Sashelp.Zipmil データセットの最新バージョンは、テクニカルサポート Web サイトからダウンロードできます。Name 列から **Zipcode Data Set** を選択して、ダウンロードプロセスを開始します。データセットをダウンロードして解凍した後、CIMPORT プロシジャを実行する必要があります。

州郵便番号テーブルが最後に更新された時期の特定

APO および FPO 住所を除き、Sashelp.Zipcode データセットには、ZIPSTATE およびその他の ZIP コード関数で使用される郵便番号情報が含まれています。このデータセットが最後に更新されたのはいつかを判断するには、PROC CONTENTS を実行します。

```
proc contents data=sashelp.zipcode;
run;
```

CONTENTS プロシジャからの出力は、最終更新の日付と Sashelp.Zipcode データセットのコンテンツを提供します。

注: Sashelp.Zipcode データセットの最新バージョンは、テクニカルサポート Web サイトからダウンロードできます。Name 列から **Zipcode Data Set** を選択して、ダウンロードプロセスを開始します。データセットをダウンロードして解凍した後、CIMPORT プロシジャを実行する必要があります。

比較

ZIPSTATE は、5 桁の ZIP コード引数に対応する大文字の 2 文字の州郵便番号(または米国領の場合は世界の GSA 地理コード)を返します。ZIPFIPS は、米国の ZIP コードに対応する 2 桁の FIPS コードを返します。

例: ZIPSTATE 関数

この例では、ノースカロライナ州の ZIP コードを州郵便番号に変換します。

```
proc ds2;
 data _null_;
  method run();
  dcl char(2) state;
  state=zipstate(27511);
  put state=;
  end;
 enddata;
run;
quit;
```

これらのステートメントは、次の結果を生成します。

```
state=NC
```

関連項目

関数:

- “ZIPCITY 関数” (1272 ページ)
- “ZIPNAME 関数” (1278 ページ)
- “ZIPNAMEL 関数” (1280 ページ)

8

DS2 入力形式

入力形式の概要 1285
一般的な入力形式構文 1285
DS2 での入力形式の使用方法 1286
DS2 で入力形式を指定する方法 1287
DS2 入力形式の検証 1287
DS2 入力形式の例 1287

入力形式の概要

入力形式は、値が列に読み込まれる方法を決定する命令です。たとえば、次の値にはドル記号とカンマが含まれています。

```
$1,000,000
```

数値 1000000 を列に保存する前にドル記号($)とカンマ(,)を削除するには、この値をCOMMA11.入力形式で読み取ります。

一般的な入力形式構文

DS2 入力形式は次のような形式になります。

[$] *informat* [*w*] . [*d*]

構文の説明は次のとおりです。

$
文字入力形式であることを表します。数値入力形式には使用されません。

informat
入力形式名です。入力形式には、SAS 入力形式、PROC FORMAT の INVALUE ステートメントを使用して定義されたユーザー定義の入力形式があります。ユーザー定義の入力形式の詳細については、*Base SAS プロシジャガイド*の PROC FORMAT を参照してください。

w
入力形式の幅です。大部分の入力形式では、入力データのカラム数となります。

d
オプションの小数点以下の桁数です(数値入力形式)。入力データを 10 の *d* 乗で除算します。

注: 小数点以下 32 桁まで読み込めるように数値入力形式を指定している場合でも、多くのコンピューターで使用されている 8 バイトの浮動小数点数表記の制限により、小数点以下 15 桁を超える浮動小数点数では精度が低下することがあります。

入力形式は、名前の一部として必ずピリオド(.)を含みます。入力形式の *w* 値と *d* 値を省略すると、デフォルト値が使用されます。データに小数点が含まれる場合、*d* 値は無視され、実際に入力データにある小数点以下桁数が読み込まれます。

入力形式の動作と入力形式の完全なリストの詳細については、*SAS 出力形式と入力形式: リファレンス*を参照してください。

DS2 での入力形式の使用方法

DS2 は、次のように SAS 入力形式をサポートします。

- SAS によって提供される入力形式とユーザー定義の入力形式の両方を列に関連付けることができます。SAS で独自の入力形式を作成する方法については、*Base SAS プロシジャガイド*の PROC FORMAT を参照してください。

  注: ユーザー定義の入力形式を作成してアクセスするには、SAS 入力形式の定義を保存する SAS カタログファイルにアクセスするために、SAS セッションが使用可能である必要があります。

- SAS データセットと SPD データセットのみが、列を含む入力形式の保存と取得をサポートします。
- 入力形式はすべてのデータ型に関連付けることができますが、すべてのデータ型は CHAR または DOUBLE のいずれかに変換されます。
- DS2 DECLARE ステートメントの HAVING 句を使用して、SAS 入力形式を列に関連付けることができます。詳細については、“DS2 で入力形式を指定する方法” (1287 ページ)を参照してください。

SAS が提供する入力形式の詳細と完全なリストについては、*SAS 出力形式と入力形式: リファレンス*の入力形式のセクションを参照してください。

DS2 で入力形式を指定する方法

DS2 では、DECLARE ステートメントの HAVING 句の属性として入力形式を指定します。たとえば、次のステートメントでは、列 *y* は IEEE8.2 出力形式および BITS5.2 入力形式で宣言されています。

```
dcl double y having format ieee8.2 informat bits5.2;
```

注: DS2 では、列の入力形式を変更または削除することはできません。

DS2 入力形式の検証

入力形式は、実行時までデータソースによって 検証されたり、列に適用されたりすることはありません。

列のメタデータが要求されると、検証なしで入力形式名が返されます。

DS2 入力形式の例

```
dcl char(10) y having label 'varchar' format $5. informat
$charzb4.3;

dcl double ssn having format best 10.4 informat comma10.4;

dcl double(6) salary having informat uscurrency.;

dcl char(12) site having informat $city.;
```

9

DS2 演算子

ディクショナリ **1289**
NEW 演算子 1289

ディクショナリ

NEW 演算子

パッケージのインスタンスを構築します。

注: 式の演算子については、“式の演算子 ”を参照してください。
かっこの前のエスケープ文字(\)は、構文にかっこが必要であることを示します。

参照項目: 事前定義された DS2 パッケージの_NEW_演算子は、各パッケージのリファレンスセクションに記載されています。

構文

package-variable=**_NEW_** *package-name* ([*constructor-arguments*]);

必須引数

package-variable
パッケージのインスタンスを参照できる名前を指定します。

package-name
パッケージの名前を指定します。

要件 *package-name* は、DS2 で事前定義されたパッケージタイプであるか、_NEW_演算子を実行する前に PACKAGE ステートメントで作成されている必要があります。

オプション引数

constructor-arguments
パッケージインスタンスのコンストラクターに渡されるコンストラクター引数を指定します。

詳細

DS2 パッケージは、特定のインスタンスを構築して他の DS2 プログラムで使用できる変数とメソッドのコレクションです。

PACKAGE ステートメントを使用してメソッドと変数をパッケージに保存した後、パッケージを宣言してインスタンス化することにより、それらにアクセスできます。_NEW_演算子を使用してパッケージをインスタンス化する場合は、最初に DECLARE PACKAGE ステートメントを使用してパッケージ変数を宣言する必要があります。

declare package *package-name variable-name*;

variable-name = _new_ *package-name*( );

たとえば、次のコード行で、DECLARE PACKAGE ステートメントは、C が COMPLEX パッケージ型の変数であることを SAS に通知します。_NEW_演算子は、COMPLEX パッケージのインスタンスを構築し、それをパッケージ変数 C に割り当てます。

```
declare package complex c;
c = _new_ complex();
```

パッケージデータを初期化する場合は、コンストラクターを使用できます。コンストラクターは、パッケージをインスタンス化し、パッケージデータを初期化するために使用されるメソッドです。たとえば、ハッシュおよびハッシュ反復子パッケージのコンストラクター構文のパラメーターを使用して、初期化データを提供できます。

```
declare package hash h();
h = _new_ hash(0, 'mytable', 'yes', 'replace', 'sumnum', 'y');
```

注: DECLARE PACKAGE ステートメントをコンストラクター引数とともに使用して、パッケージを 1 つのステップで宣言およびインスタンス化できます。前述の例は次のようになります。

```
declare package hash h(0, 'mytable', 'yes', 'replace', 'sumnum', 'y');
```

詳細については、“DS2 パッケージ” (*SAS DS2 プログラマガイド*)および“DECLARE PACKAGE ステートメント” (1323 ページ)を参照してください。

比較

DECLARE PACKAGE ステートメントと_NEW_演算子を使用するか、DECLARE PACKAGE ステートメントのみを使用して、パッケージのインスタンスを宣言およびインスタンス化できます。

例

“ユーザー定義パッケージ” (*SAS DS2 プログラマガイド*)および “事前定義された DS2 パッケージ” (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- “パッケージのコンストラクターとデストラクター” (*SAS DS2 プログラマガイド*)
- “パッケージ、スコープ、および有効期間” (*SAS DS2 プログラマガイド*)

演算子:

- “_NEW_ 演算子: FCMP パッケージ” (1642 ページ)
- “_NEW_ 演算子: ハッシュパッケージ” (1694 ページ)
- “_NEW_ 演算子: ハッシュ反復子パッケージ” (1698 ページ)
- “_NEW_ 演算子: ロガーパッケージ” (1814 ページ)
- “_NEW_ 演算子: 行列パッケージ” (1875 ページ)
- “_NEW_ 演算子: PCRXFIND パッケージ” (1908 ページ)
- “_NEW_ 演算子: PCRXREPLACE パッケージ” (1916 ページ)
- “_NEW_ 演算子: SQLSTMT パッケージ” (1991 ページ)

ステートメント:

- “DECLARE PACKAGE ステートメント” (1323 ページ)

10

DS2 ステートメント

ステートメントの概要 **1294**

ブロックステートメント **1295**
ブロックステートメントの概要 1295
プログラムブロックステートメント 1295
プログラムサブブロックステートメント 1295

グローバル宣言ステートメント **1297**

ローカルステートメント **1298**

DS2 プログラムにコメントを含める **1299**

ディクショナリ **1300**
ARRAY 割り当て ステートメント 1300
Assignment ステートメント 1301
BY ステートメント 1304
CONTINUE ステートメント 1309
DATA ステートメント 1310
DECLARE ステートメント 1313
DECLARE PACKAGE ステートメント 1323
DECLARE THREAD ステートメント 1335
DO ステートメント 1337
DROP ステートメント 1344
DROP PACKAGE ステートメント 1346
DROP THREAD ステートメント 1347
DS2_OPTIONS ステートメント 1348
ENDDATA ステートメント 1356
ENDPACKAGE ステートメント 1357
ENDTHREAD ステートメント 1357
FORWARD ステートメント 1358
GOTO ステートメント 1359
IF ステートメント: サブセット化 1360
IF-THEN/ELSE ステートメント 1362
KEEP ステートメント 1364
Labels ステートメント 1366
LEAVE ステートメント 1367
MERGE ステートメント 1370
METHOD ステートメント 1376
Null ステートメント 1385

OUTPUT ステートメント 1386
PACKAGE ステートメント 1392
PUT ステートメント 1397
RENAME ステートメント 1405
RESIZEARRAY ステートメント 1406
RETAIN ステートメント 1408
RETURN ステートメント 1411
SELECT ステートメント 1412
SET ステートメント 1416
SET FROM ステートメント 1422
STOP ステートメント 1427
Sum ステートメント 1427
THREAD ステートメント 1429
VARARRAY ステートメント 1434
VARLIST ステートメント 1439

ステートメントの概要

DS2 ステートメントは、キーワード、識別子、特殊文字、演算子を含めることができる一連のアイテムです。DS2 ステートメントは、次のアクションを実行できます。

- 変数の割り当ての実行
- プログラムコントロールへの影響
- データの入出力の実行
- メソッドの作成
- メソッドと関数の呼び出し
- DS2 プログラムの動作の指定または変更

すべての DS2 ステートメントはセミコロンで終わります。

ステートメントカテゴリは次のとおりです。

動作
: DS2 プログラムの動作を指定または変更するために使用します。

ブロック
: プログラミングブロックを作成するために使用します。詳細については、*"プログラミングブロック" (SAS DS2 プログラマガイド)*を参照してください。

ブロック制御
: プログラミングブロックのプログラム動作を変更するために使用します。

グローバル
: DATA、PACKAGE、または THREAD ステートメントによって作成されたプログラミングブロックのグローバルセクションの任意の場所で使用します。

ローカル
: DS2 メソッド内でのみ使用します。

一部のステートメントは複数のカテゴリに属していることに注意してください。たとえば、DECLARE ステートメントは、グローバルステートメントとローカルステートメントの両方にすることができます。

ブロックステートメント

ブロックステートメントの概要

プログラミングブロックで他のステートメントをグループ化するために使用されるという点で、他のステートメントとは異なる 5 つの DS2 ステートメントがあります。ブロックステートメントは次のとおりです。

- DO...END
- METHOD...END
- PACKAGE...ENDPACKAGE
- DATA...ENDDATA
- THREAD...ENDTHREAD

ブロックステートメントは、プログラムブロックを作成するステートメントとプログラムサブブロックを作成するステートメントの 2 つのカテゴリにさらに分類できます。プログラムブロックステートメントには、PACKAGE...ENDPACKGE、DATA...ENDDATA、および THREAD...ENDTHREAD が含まれます。プログラムサブブロックステートメントには、METHOD...END および DO...END が含まれます。

このドキュメントでは、これらの用語はプログラミングブロックに使用されます。

- データプログラミングブロックまたは**データプログラム**は、DATA...ENDDATA ステートメントで囲まれたコードを指します。
- パッケージプログラミングブロックまたは**パッケージ**は、PACKAGE... ENDPACKAGE ステートメントによって囲まれた変数およびメソッドの保存されたライブラリを指します。パッケージの変数とメソッドは、DS2 プログラム、スレッド、またはその他のパッケージで使用できます。
- スレッドプログラミングブロックまたは**スレッドプログラム**は、THREAD... ENDTHREAD ステートメントで囲まれたストアドプログラムを指します。スレッドプログラムは、DS2 プログラムまたはパッケージの SET FROM ステートメントによって呼び出すことができます。
- DO プログラミングブロックまたは **DO ループ**は、DO...END ステートメントで囲まれたプログラミングステートメントのサブブロックを指します。
- メソッドプログラミングブロックまたは**メソッドブロック**は、METHOD...END ステートメントで囲まれたプログラミングステートメントのサブブロックを指します。

各データプログラムには、1 つだけのプログラムブロックステートメントが必要です。データプログラムには、複数のサブブロックステートメントが含まれる場合が

あります。プログラミングブロックの詳細については、“プログラミングブロック” (*SAS DS2 プログラマガイド*)を参照してください。

プログラムブロックステートメント

プログラムブロックは、DATA、PACKAGE、および THREAD ステートメントを使用して作成されます。これらのステートメントとその最後の END ステートメントは、他のステートメントの外部で使用され、それらが定義するプログラムブロックには、他のステートメントを含めることができます。他のすべてのステートメントは、これらのブロックの 1 つで使用する必要があります。

データプログラム、パッケージ、またはスレッドプログラムには、2 つのセクションが含まれています。グローバル宣言のセクションとそれに続く METHOD ステートメントのセクションです。これはデータプログラムの例です。

```
data t;
  ...global declarations
  ...METHOD statement;
enddata;
```

すべてのグローバル宣言は、プログラムの最初の METHOD ステートメントの前に置く必要があります。プログラムの METHOD ステートメントの後にグローバル宣言が見つかった場合、構文エラーが発生します。

DATA ステートメントはデータプログラムを作成します。データプログラムは、DATA...ENDDATA ステートメントによって作成されたプログラム内に含まれるグローバル宣言リストと METHOD ステートメントリストで構成されます。データプログラムのコンパイルと実行については、“DS2 プロシジャ” (*Base SAS プロシジャガイド*)を参照してください。DATA ステートメントの詳細については、“DATA ステートメント” (1310 ページ)を参照してください。

スレッドプログラムは、THREAD...ENDTHREAD ステートメントによって作成されます。スレッドプログラムは、THREAD...ENDTHREAD ステートメント内に含まれるグローバル宣言リストと METHOD ステートメントリストで構成されます。スレッドプログラムの構造は基本的にデータプログラムの構造と同じですが、複数のスレッドを並列に実行するために使用されます。別のデータプログラムまたはパッケージで DECLARE ステートメントを使用してスレッドを参照すると、スレッドプログラムがメモリにロードされます。その後、後続のデータプログラムで SET FROM ステートメントを使用して、1 つ以上のオペレーティングシステムスレッドでプログラムを実行できます。詳細については、“THREAD ステートメント” (1429 ページ)を参照してください。

パッケージは、PACKAGE...ENDPACKAGE ステートメントによって定義されます。パッケージは、データプログラム、スレッドプログラム、または別のパッケージから呼び出すことができる変数とメソッドのコレクションです。パッケージは、PACKAGE...ENDPACKAGE ステートメントによって作成されたプログラミングブロック内に含まれるグローバル宣言リストと METHOD ステートメントリストで構成されます。パッケージは、後で使用するためにコンパイルおよび保存されます。DS2 プログラム、スレッドプログラム、または別のパッケージでパッケージを宣言すると、ストアドパッケージがメモリにロードされます。その後、パッケージ内のメソッドと変数にアクセスできます。詳細については、“PACKAGE ステートメント” (1392 ページ)を参照してください。

プログラムサブブロックステートメント

プログラムサブブロックステートメントは、METHOD または DO ステートメントを使用して作成されます。DS2 プログラムには通常、プログラミングステートメントのいくつかのサブブロックが含まれています。

各サブブロックには、2 つのセクションが含まれています。グローバル宣言ステートメントのセクションと、それに続く他のローカルステートメントのセクションです。これは、METHOD サブブロックの例です。

```
method m;
  ...global declarations
  ...local statements;
end;
```

すべてのグローバル宣言ステートメントは、サブブロック内の他のすべてのローカルステートメントの前に置く必要があります。グローバル宣言ステートメントがプログラミングサブブロック内の他のタイプのステートメントの後に配置されると、構文エラーが発生します。

グローバル宣言ステートメント

グローバル宣言ステートメントは、DATA、PACKAGE、または THREAD ステートメントによって作成されたプログラムの宣言セクションに含まれている必要があるステートメントです。グローバル宣言ステートメントは通常、DS2 プログラムの情報を提供するか、情報またはデータを要求します。通常、グローバル宣言ステートメントは実行可能ではありません。これらは、DS2 がプログラムステートメントをコンパイルするとすぐに有効になります。

次の表に、DS2 プログラムの宣言セクションで許可される DS2 ステートメントを示します。

注: DECLARE ステートメントは、グローバルとメソッド内の両方で使用できます。

- DECLARE
- DROP
- FORWARD
- KEEP
- RENAME
- RETAIN
- VARARRAY
- VARLIST

ローカルステートメント

ローカルステートメントは、METHOD ステートメントで作成されたプログラミングブロック内で使用できるステートメントです。

次の表に、DS2 メソッドステートメントを示します。

注: すべてのグローバル宣言ステートメントは、メソッドプログラミングブロック内の他のすべてのローカルステートメントの前に置く必要があります。

注: METHOD ステートメントはローカルステートメントではありません。したがって、METHOD ステートメントを別の METHOD ステートメント内にネストすることはできません。

- Assignment
- BY
- CONTINUE
- DECLARE
- DECLARE PACKAGE
- DECLARE THREAD
- DO
- GOTO
- IF, Subsetting
- IF-THEN/ELSE
- Labels
- LEAVE
- MERGE
- Null
- OUTPUT
- PUT
- RETURN
- SELECT
- SET
- SET FROM
- STOP

- Sum

DS2 プログラムにコメントを含める

プログラムの目的を文書化するために、DS2 プログラムにコメントを含めることをお勧めします。

DS2 言語は、DS2 コードにコメントを含めるために次の構文をサポートしています。

```
/*comment*/
```

DS2 コメントには次の特徴があります。

- これは、文字定数または区切られた識別子を除いて、単一の空白スペースが表示される可能性のある DS2 プログラムの任意の場所に追加できます。
- 任意の長さにすることができます。
- DS2 ステートメントおよび DS2 式に埋め込むことができます。
- 内部空白、改行、セミコロン、引用符(対になっていない引用符を含む)を含めることができます。
- DS2 を実行できるすべての環境でサポートされています。
- ネストすることは *できません*。

SAS マクロコメント構文*%*comment;*は、DS2 コンパイラではサポートされていません。SAS マクロコメントは、DS2 プロシジャでサブミットされる DS2 プログラムに埋め込むことができます。マクロ機能は、コードが DS2 プロシジャに渡されてさらに処理される前に、プログラムコードから SAS マクロコメントを削除します。SAS プログラミングランタイム環境の外部で実行される DS2 プログラムで使用される SAS マクロコメントは、構文エラーを引き起こし、コンパイルを停止します。

コメント用の DS2 コードでは、/.../および*...を使用できます。

このコメント例は、/*...を使用したテストライブラリに関するコメントを提供します。

```
/*----------------------------------------------------------
 *
 *            SAS  TEST  LIBRARY
 *
 *     NAME: ds2mod.sas
 *  SUPPORT: saswss
 *    TITLE: MOD function
 *     KEYS: mod function
```

```
*   SYSTEMS: all

*  COMMENTS: This test is self-validating. A message is printed

*           in the log if an incorrect value is obtained.

-----------------------------------------------------------/
```

このサンプルコメントは、eps 変数の詳細を提供します。*を使用しています...

```
eps=1.e-12;    *** put eps=;
```

Ds2 コメントの使用方法を示すその他の例を次に示します。

```
data _null_;
   dcl double x;
   dcl double y;  /* DS2 comments are preferred in PROC DS2; they required in other
programming environments */
   method init();
      /* multiline comment with
        an embedded semicolon ;
        an embedded single quotation mark '
        an embedded double quotation mark " */
      if (missing(x) or /* multiline comment embedded
                        in a DS2 expression */
         missing(y)) then do;
         put /* comment embedded in a DS2 statement */ 'Hello, World';
      end;
   end;
enddata;
run;
```

ディクショナリ

ARRAY 割り当て ステートメント

一時配列または定数リストのいずれかを一時配列に割り当てます。

構文

array-name:= *array-name*;

array-name:=(*constant-list*);

必須引数

array-name
配列の名前を指定します。

constant-list
配列要素を定義するリストを指定します。

詳細

一時配列または定数リストのいずれかを一時配列に割り当てることができます。

ある配列を別の配列に割り当てる場合、2 つの配列のデータ型は互換性がある(同じまたは変換可能)必要があります。次元の数は 2 つの配列で同じである必要はなく、各配列の要素の総数は同じである必要はありません。

例

次のステートメントは、配列の割り当ての例です。

```
ar1:=('sales', 'inv', 'profit');

ar2:=(3*3.14159, 2*'5', 2*(1,2), 99);

ar7:=ar2;
```

関連項目

"配列割り当て" (*SAS DS2 プログラマガイド*)

Assignment ステートメント

式を評価し、結果を変数に保存します。

カテゴリ: ローカル

構文

variable=*expression*;

必須引数

variable
新規または既存の変数の名前を指定します。

範囲 *variable* には、変数名、配列参照、または SUBSTR 関数を指定できます。

ヒント　割り当てステートメントによって作成された変数は自動的に保持されません。

expression
有効な DS2 式です。

ヒント　*expression* には、等号の左側で使用される変数を含めることができます。ステートメントの両側に変数が現れると、右側の元の値が式の評価に使用され、結果は等号の左側の変数に保存されます。

参照項目:　"DS2 式" (*SAS DS2 プログラマガイド*)

詳細

基本

割り当てステートメントでは、等号の右側にある式を評価し、等号の左側に指定されている変数に結果を保存します。

次の型変換は、割り当てステートメントで行われます。

- 変数にデータ型があり、式のデータ型が一致せず、変換できる場合、式は変数のデータ型に変換されます。式を変換できない場合、エラーが発生します。
- 変数にデータ型がない場合は、次のいずれかのアクションが発生します。
  - 式の値が null でなく、データ型が CHAR、BINARY、DATE、TIME のいずれかの場合、変数には式のデータ型が与えられます。
  - 式の値が null でなく、数値型の場合、変数には DOUBLE のデータ型が与えられます。式の値が null の場合、変数には DOUBLE のデータ型が与えられます。
  - 式の型が CHAR または BINARY の場合、変数には式の値の長さが与えられます。コンパイル時に式の値を決定できない場合(たとえば、VARCHAR 文字列の場合)、変数にはデフォルトの長さ 200 が与えられます。
  - 式の型が TIME または TIMESTAMP の場合、変数には式の精度が与えられます。
- 割り当てステートメントが a = b = 5 の場合、b = 5 は式です。b が 5 以外の値の場合、b = 5 は 0 と評価されます。したがって、a には値 0 が割り当てられます。最初の等号(=)は割り当て演算子であり、2 番目の等号は論理等値演算子です。詳細については、"例 2: 複数の等号を持つ式の使用" (1303 ページ)を参照してください。

注: DS2 は、等号だけでなく eq の使用もサポートしています。たとえば、x = y < z < w;は x = y < z & z < w;と同等です。別の例としては、a = b = c =d;は a = ((b = c) & (c = d));に等しいです。

例

例 1: さまざまなタイプの式

これらの割り当てステートメントは、さまざまなタイプの式を使用します。

- name='Nagasaki';
- FullName='Mr. '||name;
- price=price+markup;
- declare int i;
  declare double d;
  declare character(200) c;

  i = 5;
  d = 1.2345;
  d = d + i;
  c = 'abc';
  c = d;
  c = '123' || '456';
  i = c;

例 2: 複数の等号を持つ式の使用

これらの割り当てステートメントの結果は **a**=0 です。b、c、および d の値は変更されません。

```
data;
  dcl double a b c d;

  method init();
    a = b = c = d = 5;
    put _all_;
  end;
enddata;
run;
```

例 3: 角かっこで囲まれた長いリテラルの使用

次の例では、ネストされた角かっこを使用して、長いリテラル文字列を VARCHAR 変数 s に割り当てます。

```
data _null_;
  method init();
    dcl varchar(100) s;
    s = [===[line 1
          line 2
          line 3 contains a double square bracket [[
          line 4]===];
    put s;
  end;
enddata;
run;
```

次の行が SAS ログに書き込まれます。

```
line 1
line 2
line 3 contains a double square bracket [[
line 4
```

関連項目

- “DS2 定数” (*SAS DS2 プログラマガイド*)
- “式オペランドの DS2 型変換” (*SAS DS2 プログラマガイド*)

BY ステートメント

DS2 プログラムの MERGE または SET ステートメントの操作を制御し、特別なグループ化変数を設定します。

カテゴリ: ローカル

制限事項: BY ステートメントは、MERGE または SET ステートメントの直後に続く必要があります。SET ステートメントを使用する場合、BY ステートメントはオプションです。

ヒント: テーブルを SET または MERGE ステートメントと組み合わせる場合、末尾の空白は常に無視されます。

構文

BY [DESCENDING] *column*... [DESCENDING] *column*;

必須引数

column(s)
テーブルを並べ替えるための各列の名前を指定します。これらの列は BY 変数と呼ばれます。

ヒント テーブルは複数の列で並べ替えることができます。

オプション引数

DESCENDING
指定された変数によってテーブルが降順で並べ替えられることを指定します。DESCENDING とは、数値的には最大から最小へ、文字変数ではアルファベットの逆順となります。

制限事項 インデックスは常に昇順で保存されるため、インデックス付きのテーブルで DESCENDING オプションを使用することはできません。

詳細

DS2 が BY グループの開始と終了を示す方法

DS2 は、BY 変数ごとに FIRST.*variable* と LAST.*variable* の 2 つの一時変数を作成することにより、BY グループの開始と終了を示します。これらの変数の値は 0 または 1 です。DS2 は、BY グループの最初の行を読み取るときに FIRST.*variable* の値を 1 に設定し、BY グループの最後の行を読み取るときに LAST.*variable* の値を 1 に設定します。これらの一時変数は DS2 プログラミングで使用できますが、結果セットには追加されません。

SAS がグループ化されたデータを処理する方法とデータを準備する方法の完全な説明については、"テーブルの結合" (*SAS DS2 プログラマガイド*)を参照してください。

データプログラム

BY ステートメントは、データプログラムでその前にある SET または MERGE ステートメントにのみ適用され、データプログラムでこれらの各ステートメントに付随できる BY ステートメントは 1 つだけです。BY ステートメントがデータプログラムの他の場所に記述された場合、エラーが発生します。

注: BY ステートメントは、SORT プロシジャで SORTSEQ=LINGUISTIC オプションを使用して並べ替えられたデータの言語照合を認識します。

注: 文字データ型の列を持つテーブルがあると仮定します。DECLARE ステートメントを使用した入力時に列を数値データ型に変更した場合、結果の列の並べ替え順は数値ではありません。たとえば、次の文字列(CHAR)がテーブルに存在するとします: 9、10、500。SET および BY ステートメントを使用してテーブルを読み取るときに s を数値列(DOUBLE)として宣言すると、データは英数字順、つまり 10、500、9 で出力として生成されます。SET ステートメントは、文字列が数値データ型に変換される前に、行を英数字順に並べ替えます。

詳細については、"テーブルの結合" (*SAS DS2 プログラマガイド*)を参照してください。

BY グループの処理

SAS は次の値を FIRST.*variable* と LAST.*variable* に割り当てます。

- FIRST.*variable* の値は、次の条件下で 1 になります。
  - 現在の行がテーブルから読み取られる最初の行である場合。
  - 現在の行の BY 変数の値が、前の行のその BY 変数の値と異なる場合。
  - FIRST.*variable* は、BY ステートメントの先行する変数の値が 1 です。

    他のすべての場合、FIRST.*variable* の値は 0 です。
- LAST.*variable* の値は、次の条件下で 1 になります。
  - 現在の行がテーブルから読み取られる最後の行である場合。
  - 現在の行の BY 変数の値が、次の行のその BY 変数の値と異なる場合。

- LAST.*variable* は、BY ステートメントの先行する変数の値が 1 です。

  他のすべての場合、LAST.*variable* の値は 0 です。

例

例 1: 1 つ以上の BY 変数の指定

- オブザベーションは、変数 DEPT の昇順です。

  by dept;

- オブザベーションは、CITY のアルファベット順(昇順)であり、CITY の各値内では、ZIPCODE の昇順です。

  by city zipcode;

例 2: 並べ替え順序の指定

- オブザベーションは SALESREP の昇順であり、各 SALESREP 値内では、JANSALES の値の降順です。

  by salesrep descending jansales;

- オブザベーションは BEDROOMS の降順であり、BEDROOMS の各値内では、PRICE の降順です。

  by descending bedrooms descending price;

例 3: SET ステートメントでテーブルを結合するときに BY ステートメントを使用する

次の例では、2 つのテーブルを作成し、SET ステートメントで、**common** 列を使用してテーブルを結合します。

```
data mrg01a(overwrite=yes);
  dcl varchar(10) common animal;
  method init();
    common='a'; animal='Ant'; output;
    common='b'; animal='Bird'; output;
    common='c'; animal='Cat'; output;
    common='d'; animal='Dog';  output;
    common='e'; animal='Eagle'; output;
    common='f'; animal='Frog';  output;
  end;
enddata;
run;


data mrg01b(overwrite=yes);
  dcl varchar(10) common plant;
  method init();
    common='a'; plant='Apple'; output;
    common='b'; plant='Banana'; output;
    common='c'; plant='Coconut'; output;
    common='d'; plant='Dewberry';  output;
```

```
    common='e'; plant='Eggplant'; output;
    common='f'; plant='Fig';  output;
     common='g'; plant='Grapefruit'; output;
  end;
enddata;
run;


/*  set concatenates */
data;
  method run();
    set mrg01a mrg01b; by common;
  end;
 enddata;
run;
```

次のテーブルが生成されます。

common	animal	plant
a	Ant	
a		Apple
b	Bird	
b		Banana
c	Cat	
c		Coconut
d	Dog	
d		Dewberry
e	Eagle	
e		Eggplant
f	Frog	
f		Fig
g		Grapefruit

例 4: MERGE ステートメントでテーブルを結合するときに BY ステートメントを使用する

次の例では、2 つのテーブルを作成し、MERGE ステートメントで、**common** 列を使用してテーブルを結合します。

```
data mrg01a(overwrite=yes);
  dcl char(10) common animal;
  method init();
    common='a'; animal='Ant'; output;
    common='b'; animal='Bird'; output;
    common='c'; animal='Cat'; output;
    common='d'; animal='Dog';  output;
```

```
    common='e'; animal='Eagle'; output;
    common='f'; animal='Frog';  output;
  end;
enddata;
run;


data mrg01b(overwrite=yes);
  dcl char(10) common plant;
  method init();
    common='a'; plant='Apple'; output;
    common='b'; plant='Banana'; output;
    common='c'; plant='Coconut'; output;
    common='d'; plant='Dewberry';  output;
    common='e'; plant='Eggplant'; output;
    common='f'; plant='Fig';  output;
    common='g'; plant='Grapefruit'; output;
  end;
enddata;
run;


/*  match merge */
data;
  method run();
    merge mrg01a mrg01b; by common;
  end;
 enddata;
run;
```

次のテーブルが生成されます。

common	animal	plant
a	Ant	Apple
b	Bird	Banana
c	Cat	Coconut
d	Dog	Dewberry
e	Eagle	Eggplant
f	Frog	Fig
g		Grapefruit

関連項目

- *“テーブルの結合” (SAS DS2 プログラマガイド)*

ステートメント:

- “MERGE ステートメント” (1370 ページ)

- “SET ステートメント” (1416 ページ)

CONTINUE ステートメント

現在の DO ループの反復の処理を停止し、次の反復から再開します。

カテゴリ: ローカル

構文

CONTINUE;

引数なし

CONTINUE ステートメントに引数はありません。DO ループの次の反復でステートメントの処理を再開します。

詳細

CONTINUE ステートメントは、反復 DO ループのステートメントリストにのみ記述できます(例: DO *i*=、DO WHILE、または DO UNTIL)。

比較

- CONTINUE ステートメントは、DO ステートメントの現在の反復の処理を停止し、現在の DO ステートメントの次の反復でプログラムの実行を再開します。
- LEAVE ステートメントは、現在の DO ステートメントの処理を停止し、現在の DO ステートメントの外部でプログラムの実行を再開します。

例

この例は、CONTINUE ステートメントの使用法を示しています。DO ループが反復され、ctr の増分値が出力されます。ctr が 3 に等しくなると、CONTINUE ステートメントは、実行を DO ループの次の反復にジャンプさせ、ctr が出力されないようにします。

```
data _null_;
  dcl int ctr;
  method init();
    do ctr = 1 to 5;
      if ctr = 3 then continue;
```

```
      put ctr;
    end;
  end;
enddata;
```

次の行が SAS ログに書き込まれます。

```
1
2
4
5
```

関連項目

ステートメント:

- “DO ステートメント” (1337 ページ)
- “LEAVE ステートメント” (1367 ページ)

DATA ステートメント

DS2 プログラムを開始し、出力テーブルの名前を指定します。

カテゴリ: ブロック

別名: TABLE

構文

DATA [<table-expression>] [... <table-expression>] ;

... *program-body* ...

[ENDDATA ;]

<table-expression>::=

table (*table-options*)

| **_ROWSET_** (*table-options*)

| **_NULL_**

引数なし

DATA ステートメントでテーブル名を指定しない場合、DS2 プログラムはテーブル行をクライアントアプリケーションに返し、テーブルは作成されません。

必須引数

table
テーブルの名前を指定します。*table* は、次のいずれかの形式をとります。

- *catalog.schema.table-name*
- *schema.table-name*
- *catalog.table-name*
- *table-name*
- *caslib.table-name*

catalog	SQL カタログの ANSI SQL 規格が実装された、論理的に関連するスキーマをグループ化したデータコンテナーオブジェクトです。カタログは、 データ編成階層の第 1 レベル(最上位)のグループ化メカニズムです。 スキーマとともに使用され、名前を修飾する方法を提供します。
schema	SQL スキーマの ANSI SQL 規格が実装された、テーブルやビューなどのファイルや、ストアドプロシジャなどのデータソースで サポートされるその他のオブジェクトをグループ化するデータコンテナーオブジェクトです。スキーマは、catalog とともに使用されて、名前を修飾する方法を提供する グループ化オブジェクトを提供します。
table-name	テーブルの名前です。
caslib.table-name	CAS サーバーにあるテーブルの名前です。

注 テーブル名にドットが含まれていて、CAS テーブルにアクセスしている 場合は、テーブル名を二重引用符で囲む必要があります。次に例を示します。

```
data mycaslib "tdlibref.foo";
```

テーブル名とスキーマ名を引用符で囲まない場合、DS2 はテーブル名とスキーマ名を大文字で保存し、二重引用符を加えます。引用符で囲まれたテーブル名とスキーマ名は、そのまま使用されます。つまり、引用符で囲まれた元の大文字小文字が引用符で囲まれたままになります。たとえば、data mytable;ではテーブル名は"MYTABLE"として保存され、data "MyTable";ではテーブル名は"MyTable"として保存されます。これは、データソースのテーブル名とスキーマ名で大文字と小文字が区別される場合に重要です。

注意 **LIBNAME ステートメントで PRESERVE_TAB_NAMES=no オプションを使用すると、予期しない結果が生じる可能性があります。**

ROWSET
DATA ステートメントでテーブルを作成せず、代わりにテーブル行をクライアントアプリケーションに返すように指定します。

NULL
DATA ステートメントでテーブルを作成したり、クライアントアプリケーションに行を返したりしないように指定します。

table-options
DS2 プログラムで出力テーブルに行を書き込むときに適用するオプションの引数を指定します。テーブルオプションの詳細については、“DS2 テーブルオプション” (1457 ページ)を参照してください。

ヒント _NULL_は、PUT ステートメントを使用する場合のプログラムのデバッグに役立ちます。

詳細

DS2 プログラムは、DATA ステートメントで始まり、ENDDATA ステートメントで終わります。

DS2 プログラムは、入力データを処理し、出力データを生成します。DS2 プログラムは、プログラムまたはスレッドとして、2 つの異なる方法で実行できます。DS2 プログラムをプログラムとして実行した場合の結果を次に示します。

- 入力データには、データベーステーブルの行と DS2 プログラムスレッドの行の両方を含めることができます。
- 出力データは、クライアントアプリケーションに返されるデータベーステーブルまたは行のいずれかになります。

DS2 プログラムをスレッドとして実行した場合の結果を次に示します。

- 入力データには、データベーステーブルの行のみを含めることができ、他のスレッドの行は含めることができません。
- 出力データには、スレッドを開始した DS2 プログラムに返される行が含まれます。

DATA ステートメントでテーブル名を指定しない場合やキーワード_ROWSET_を指定する場合、DS2 プログラムはテーブル行をクライアントアプリケーションに返し、テーブルは作成されません。DATA ステートメントでテーブル名を指定しない場合は、少なくとも 1 つのグローバル変数が必要です。

_NULL_テーブル名に行が書き込まれることはありません。したがって、DATA ステートメントに存在する唯一のテーブル名が_NULL_である場合、DS2 プログラムでは行を返しません。

他のテーブル名が存在する場合、プログラムでは *table* ごとにテーブルを作成し、テーブル行がそれらのテーブルに書き込まれます。 詳細については、“OUTPUT ステートメント” (1386 ページ)を参照してください。

大文字と小文字を区別しないデータソースにサブミットされる、区切られた列名を持つテーブルは警告されます。大文字と小文字を区別しないデータソースでは、引用符が削除され、列名が区切られていないものとして扱われます。

注: MongoDB データソースには、テーブルを作成するための特別な要件があります。データに応じて、ルートテーブル、親テーブル、子テーブルを作成することをお勧めします。DS2 ではルートテーブルのみを作成できます。親テーブルと子テーブルを作成するには、FedSQL を使用する必要があります。*SAS/ACCESS for Relational Databases: リファレンス*の MongoDB に関する情報を参照してください。

比較

パッケージ、DS2 プログラム、スレッドの比較については、“ブロックステートメント” (1295 ページ)を参照してください。

例

例 1: 出力テーブルの作成

DATA ステートメントを使用して、1 つ以上の出力テーブルを作成します。テーブルオプションを使用して、出力テーブルをカスタマイズできます。次の DS2 プログラムでは、EXAMPLE1 と EXAMPLE2 の 2 つの出力テーブルを作成します。テーブルオプション DROP を使用して、列 IDNumber が EXAMPLE2 テーブルに書き込まれないようにします。

```
data example1 example2 (drop=(IDnumber));
  set sample;
  . . .more statements. . .
enddata;
```

例 2: テーブルを作成しない場合

通常、DATA ステートメントでは、出力テーブルを作成するために少なくとも 1 つのテーブル名を指定します。テーブル名としてキーワード_NULL_を使用すると、DS2 プログラムの実行時にテーブルに行が書き込まれません。この例では、各行の NAME と ID の値をログに書き込みます。出力テーブルは作成されません。

```
data _null_;
  set sample;
  put Name ID;
enddata;
```

関連項目

ステートメント:

- “OUTPUT ステートメント” (1386 ページ)
- “SET ステートメント” (1416 ページ)

DECLARE ステートメント

1 つ以上の DS2 変数または一時配列を宣言します。

カテゴリ: グローバル
ローカル

注: 構文規則の角かっこは、オプションの引数を示します。角かっこの前のエスケープ文字(\ )は、構文に角かっこが必要であることを示します。配列の境界は角かっこ([ ])で囲む必要があります。

構文

DECLARE [PRIVATE] { <data-type> <variable-list> [<having-clause>] } ;

< data-type >::=

<exact-numeric-type> | <approximate-numeric-type> | <binary-string-type>
| <string-type>
| <date-type>

<exact-numeric-type> ::=

{**INT** | **BIGINT** | **SMALLINT** | **TINYINT**
| **DECIMAL** [(*precision* [,*scale*])] | **NUMERIC** [(*precision* [,*scale*])] }

<approximate-numeric-type> ::=

{ **DOUBLE** | **DOUBLE PRECISION** | **FLOAT** | **REAL** }

<binary-string-type>::=

BINARY(*length*) | **VARBINARY**(*length*)

<string-type>::=

NCHAR [(*character-length*)]

| **NVARCHAR** [(*character-length*)]

| **CHAR** [(*character-length*)] [CHARACTER SET *character-set-identifier*]

| **VARCHAR** [(*character-length*)] [CHARACTER SET *character-set-identifier*]

<date-type>::=

{ **TIME** | **TIMESTAMP** } [(*precision*)] | **DATE**

<variable-list>::=

{ *variable* [...*variable*] }

| { *variable* <array-declaration> [*variable*...<array-declaration>] }

<array-declaration>::=

\[<array-bound> [, ...< array-bound>] \] | \[* \]

<array-bound>::=

{ [*dim-lower*:] *dim-upper*} | { [*dim-lower*:] {DIM (*a*[,*n*]) }

<having-clause>::=

HAVING <having-option> [... <having-option>]

<having-option>::=

LABEL '*string*' | n'*string*'

| **FORMAT** *format*

| **INFORMAT** *format*

必須引数

INT
整数変数または配列を指定します。

別名 INTEGER

参照項目: “DS2 データ型” (*SAS DS2 プログラマガイド*)

BIGINT
整数変数または配列を指定します。

参照項目: “DS2 データ型” (*SAS DS2 プログラマガイド*)

SMALLINT
整数変数または配列を指定します。

参照項目: “DS2 データ型” (*SAS DS2 プログラマガイド*)

TINYINT
整数変数または配列を指定します。

参照項目: “DS2 データ型” (*SAS DS2 プログラマガイド*)

DECIMAL [(*precision* [, *scale*])] | NUMERIC [(*precision* [, *scale*])]
正確な数値変数または配列を指定します。

precision
小数点の左側と右側の両方に保存できる最大合計桁を指定します

注 すべてのデータソースが 52 桁の精度をサポートできるわけではありません。

scale
小数点の右側に保存できる最大桁数を指定します

範囲 0–*precision*

注 *scale* は *precision* は以下です。

参照項目: “DS2 データ型” (*SAS DS2 プログラマガイド*)

DOUBLE | DOUBLE PRECISION | FLOAT | REAL
浮動小数点変数または配列を指定します。

参照項目: “DS2 データ型” (*SAS DS2 プログラマガイド*)

BINARY (*length*)
バイナリ変数または配列を指定します。

要件 BINARY を指定する場合は、変数または配列の *length* もバイト単位で指定する必要があります。

参照項目: “DS2 データ型” (*SAS DS2 プログラマガイド*)

VARBINARY (*length*)
可変長のバイナリ変数または配列を指定します。

別名 BINARY VARYING

要件 VARBINARY を指定する場合は、バイナリ変数または配列の *length* もバイト単位で指定する必要があります。

参照項目: "DS2 データ型" (*SAS DS2 プログラマガイド*)

NCHAR
文字変数または配列を指定します。

別名 NATIONAL CHARACTER

NATIONAL CHAR

参照項目: "DS2 データ型" (*SAS DS2 プログラマガイド*)

NVARCHAR
文字変数または配列を指定します。

別名 NATIONAL CHARACTER VARYING、

NATIONAL CHAR VARYING

参照項目: "DS2 データ型" (*SAS DS2 プログラマガイド*)

CHAR
文字変数または配列を指定します。

別名 CHARACTER

参照項目: "DS2 データ型" (*SAS DS2 プログラマガイド*)

VARCHAR
文字変数または配列を指定します。

別名 CHARACTER VARYING

参照項目: "DS2 データ型" (*SAS DS2 プログラマガイド*)

character-length
文字列が NCHAR、NVARCHAR、CHAR、VARCHAR データ型に対して保持できる最大文字数を指定します。

デフォルト 8

ヒント CHAR を使用して宣言された文字変数がストレージに使用するバイト数は、セッションエンコーディングによって異なります。NCHAR バリアントのいずれかを使用して宣言されたものは、ストレージが広く、シングルバイト文字ストレージでは不十分な文字セット(Unicode など)を表すために使用できます。セッションエンコーディングで文字ごとに複数のバ

イトが必要な場合(UTF-8 など)、CHAR と NCHAR は同じ型であり、どちらも NCHAR を使用します。

CHARACTER SET *character-set-identifier*
CHAR データ型と VARCHAR データ型の文字セットエンコーディング情報を指定します。

デフォルト　デフォルトのエンコーディングは、オペレーティングシステムとロケールによって異なります。

ヒント　文字セット名には、文字列リテラルまたは単純な文字列を使用できます。たとえば、"ibm-866"または'ibm-866'を指定できます。

参照項目:　文字セットエンコーディング値の完全なリストについては、*SAS 各国語サポート(NLS): リファレンスガイド*の"SAS 言語要素中のエンコーディング値"を参照してください。

TIME
時間変数または配列を指定します。

TIMESTAMP
日付と時間の両方の変数または配列を指定します。

precision
TIME データ型または TIMESTAMP データ型の精度を指定します。

デフォルト　6

注　SAS 以外のデータソースで TIME 値と TIMESTAMP 値を操作していて、精度を指定しない場合、デフォルトの精度は常に DS2 のデフォルトの精度である 6 になります。

DATE
日付変数または配列を指定します。

variable
スカラー変数または配列名を指定します。1 つ以上の変数または配列を指定できます。ただし、*variable* は *data-type* で指定された型のみにすることができます。同じ型のスカラー変数と配列変数を混在させることができます。

dim-lower and *dim-upper*
固定一時配列の配列境界の数とサイズを定義する正または負の整数を指定します。

ヒント　次元の下限が指定されていない場合、下限はデフォルトで 1 になります。

参照項目:　"一時配列変数の宣言" (1320 ページ)

\[* \]
動的境界を持つ一時配列を作成することを指定します。

参照項目:　"動的一時配列変数の宣言" (1320 ページ)

DIM(*a*[, *n*])
配列の上限のサイズが、DIM 関数呼び出しを使用して以前に宣言された配列の次元内の要素の数によって決定されることを指定します。

a
以前に宣言された配列の名前を指定します。

n
多次元配列の次元で、要素の数を知りたい次元を指定します。

ヒント *n* 値が指定されていない場合、DIM 関数は配列の最初の次元の要素の数を返します。

制限事項 DIM 関数は、配列の上限を指定するために使用できる唯一の関数です。DIM 関数は、次元の下限を指定するためには使用できません。

参照項目: “DIM 関数” (518 ページ)

LABEL '*string*' | n'*string*'
変数または配列に説明ラベルを割り当てます。ラベルは、CHAR リテラル(*string*)または NCHAR リテラル(n*string*)にすることができます。

参照項目: “DS2 データ型” (*SAS DS2 プログラマガイド*)

FORMAT *format*
有効な DS2 出力形式を変数または配列に関連付けます。

参照項目: “DS2 出力形式” (65 ページ)

INFORMAT *informat*
有効な SAS 入力形式を変数または配列に関連付けます。

参照項目: 8 章, “DS2 入力形式” (1285 ページ)

オプション引数

PRIVATE
パッケージ内からのみアクセスできる変数を指定します。

参照項目: “属性とメソッド” (*SAS DS2 プログラマガイド*)

詳細

DECLARE ステートメントの概要

DECLARE ステートメントを使用して、スカラー変数と一時配列を指定できます。固定境界または動的境界を持つ一時配列を指定できます。

DECLARE ステートメントでは、複数の変数と配列の型を指定できます。たとえば、次の DECLARE ステートメントは、**x** および **y** という名前の 2 つのスカラー変数と、**a**、**b**、および **d** という名前の 3 つの一時配列を指定します。配列 **a** と **b** は標準配列

です。それらは固定境界を指定します。配列 **d** は、動的境界を持つ一時配列です。配列境界としてアスタリスクを指定します。

```
declare double a[10] x y b[20] d[*];
```

DECLARE ステートメントは、データ型を変数リストまたは配列内の各変数に関連付けます。前の例では、**x**、**y**、**a**、**b**、**d** のデータ型はすべて DOUBLE です。

DECLARE ステートメントは、メソッドに対してローカルまたはプログラムブロックに対してグローバルなスカラー変数と一時配列を宣言できます。すべての変数宣言には、HAVING 句を含めることができます。

DS2 では、DECLARE ステートメントはパッケージおよびスレッドの宣言にも使用されます。パッケージとスレッドの宣言の詳細については、“DECLARE PACKAGE ステートメント” (1323 ページ)および “DECLARE THREAD ステートメント” (1335 ページ)を参照してください。

デフォルトでは、宣言されていない変数に対して警告が表示されます。変数を宣言せずに使用する場合、DS2 は変数にデータ型を割り当てます(暗黙の宣言)。割り当てステートメントの左側にある宣言されていない変数のデータ型は、割り当てステートメントの右側にある値のデータ型によって決定されます。ただし、DS2 プロシャで DS2SCOND システムオプションまたは SCOND オプションを使用して、DS2 が宣言されていない変数を処理する方法を制御することができます。これらのオプションを使用して、すべてのテーブル列と変数の宣言を要求できます。DS2SCOND=ERROR または SCOND=ERROR を指定する場合は、列または変数ごとに DECLARE ステートメントを使用する必要があります。割り当てによる宣言は発生しません。詳細については、“DS2SCOND= システムオプション” (1454 ページ)、“DS2 プロシジャ” (*Base SAS プロシジャガイド*)、および“変数宣言” (*SAS DS2 プログラマガイド*)を参照してください。

注: 型は、データプログラムとスレッドプログラムの間で完全に一致する必要があります。DECIMAL データ型の場合、精度とスケールの両方が一致する必要があります。文字列の場合、列サイズ、文字セット、および文字列が可変長であるか固定長であるかは、すべて一致する必要があります。

スカラー変数の宣言

スカラー宣言は、数値、文字、日付、時間のいずれかのデータ型に使用できます。文字列に含めることができる最大文字数を指定できます。次に例を示します。

```
declare char(200) s;
```

DS2 はこの数に制限を課していませんが、実際には、マシンの制限のためにいくつかの制約がある場合があります。特定のデータ型のデフォルトの長さまたは精度は、データソースによって異なります。

固定長文字変数の場合、最大長が文字列メモリの初期(および唯一の)割り当てとして使用されます。可変長文字列の場合、メモリは必要に応じて最大長まで割り当てられます。固定長文字変数のように操作後に空白パディングを必要としないため、可変長文字変数を使用することには実行時の利点がある可能性があります。

CHAR を使用して宣言された文字変数がストレージに使用するバイト数は、セッションエンコーディングによって異なります。NCHAR バリアントのいずれかを使用して宣言されたものは、ストレージが広く、シングルバイト文字ストレージでは不十分な文字セット(Unicode など)を表すために使用できます。セッションエンコー

ディングで文字ごとに複数のバイトが必要な場合(UTF-8 など)、CHAR と NCHAR は同じ型であり、どちらも NCHAR を使用します。

変数が同じスコープで複数回宣言され、その宣言が同一でない場合、エラーが発生します。

変数を宣言せずに使用すると、変数に割り当てられた値の型を受け取ります。値が割り当てられていない場合、DS2 は変数を DOUBLE 型として割り当て、値を欠損値または null 値として割り当てます。

一時配列変数の宣言

一時配列は、配列の要素が PDV に配置されていないため、結果テーブルに表示されないという点で一時的です。一時配列も、DS2 プログラムの期間中のみ存在します。

標準一時変数宣言

標準一時配列宣言は、スカラー宣言に似ています。ただし、データ型と名前に加えて、配列境界の数とサイズも指定します。配列の境界は、符号付き整数のペア[*l*:*h*]として指定されます。ここで、*l* は指定された境界の最小インデックスを表し、*h* は指定された境界の最大インデックスを表します。*h*<*l* の場合、エラーが返されます。1 つの整数のみで配列の境界を指定すると、その整数が最大インデックスとして解釈されます。デフォルトの最小インデックスは 1 です。

この例では、DOUBLE 型の一時配列 *a* を宣言し、5 つの要素に 1 から 5 までのインデックスを付けています。

```
declare double a[5];
```

一時配列の複数の境界(または次元)は、カンマ区切り文字を使用して指定されます。この例では、1 次元に 5 要素、2 次元に 10 要素、合計 50 要素の 2 次元文字配列 *b* を宣言します。

```
declare char b[5,10];
```

この例では、2 つの要素を持つ一時配列 *c* を宣言します。配列には、下限 0 と上限 1 でインデックスが付けられます。

```
declare int c[0:1];
```

動的一時配列変数の宣言

動的一時配列は、動的境界付きで定義される一時配列です。標準一時配列境界は、実行時に静的(固定)です。また、標準境界を持つ配列の要素数も、実行時に固定されています。動的境界は実行時に変更できるため、動的境界を持つ配列の要素の数は、DS2 プログラムの実行時に動的に増減できます。

動的一時配列は、アスタリスクを配列境界として宣言することによって作成されます。この例では、DOUBLE 型の動的一時配列 **d** を宣言しています。

```
declare double d[*];
```

アスタリスクは、1 次元、下限 1、初期上限 0 の配列を定義します。その結果、0 要素を含むように初期化された 1 次元配列になります。配列の上限が設定されており、実行時に RESIZEARRAY ステートメントを使用して変更できます。動的一時配列の次元数と下限は変更できません。動的一時配列は常に、下限が 1 の 1 次元配列です。

動的一時配列の要素を指定する方法については、“RESIZEARRAY ステートメント” (1406 ページ)を参照してください。

DS2 配列の詳細については、“DS2 配列” (*SAS DS2 プログラマガイド*) および“一時配列” (*SAS DS2 プログラマガイド*)を参照してください。

HAVING 句

ラベル、出力形式、および入力形式の属性を、1 つ以上のスカラー変数または配列に関連付けることができます。HAVING 句は、DATA ステップの FORMAT、INFORMAT、および LABEL ステートメントと同じように機能します。ただし、DS2 では、属性は変数または配列の宣言ステートメントで指定する必要があります。

DS2 が出力形式と入力形式を処理する方法の詳細については、“DS2 での出力形式の使用” (68 ページ)および“DS2 での入力形式の使用方法” (1286 ページ)を参照してください。

配列と HAVING 句の詳細については、“HAVING 句を使用した配列の宣言” (*SAS DS2 プログラマガイド*)を参照してください。

注: 変数がスレッドプログラムの HAVING 句で宣言され、その変数がデータプログラムで HAVING 句を使用して再宣言された場合、スレッドプログラムの HAVING 句の代わりにデータプログラムの HAVING 句が使用されます。データプログラムの DECLARE ステートメントに HAVING 句がない場合、スレッドプログラムの HAVING 句は使用されません。

例

例 1: 変数の宣言

次の例は、DECLARE ステートメントを示しています。

```
declare bigint b2 b3;
declare double d;
declare char(200) c1 c2;
declare varchar vc;
declare nchar(100) wc;
declare time(4) tm;
declare date dt;
declare varbinary(10) b;
dcl double x having label 'Amount' format ieee8.2;
dcl char(10) y having label 'varchar' format $quote.;
dcl private double x;
```

例 2: 標準一時配列の宣言

次の例は、一時配列の DECLARE ステートメントを示しています。

- declare double darr[-5:4];
- declare char carr[1:2, 0:3];
- declare int iarr[10] having format octal7.;

例 3: 標準一時配列の次元

次の表には、標準一時配列とそれらの配列の次元を指定するステートメントの例が含まれています。

ステートメント	次元の数	次元の範囲	要素の数
declare double a[100];	1	1:100	100
declare double a[10, 20, 30];	3	1:10 1:20 1:30	10x20x30 = 6000
declare double a[5:10];	1	05:10	6
declare double a[-3:3, 5, 7:9, 10];	4	-3:3 1:5 7:9 1:10	7x5x3x10 = 1050
declare double a[DIM(u)];	1	1:DIM(u)	DIM(u)
declare double a[DIM(u,1), 0:DIM(u,2)];	2	1:DIM(u,1) 0:DIM(u,2)	DIM(u,1)x(DIM(u,2)+1)

例 4: 動的境界を持つ一時配列の宣言

次の例は、動的境界のある一時配列の DECLARE ステートメントを示しています。

- declare bigint bi[*];
- declare binary(4) b[*] having format $hex.;
- declare char(10) c1[*];
- declare date dt[*];
- declare decimal(10,0) dc[*];
- declare double d[*];
- declare int i[*];
- declare nchar(10) nc[*];
- declare nvarchar(10) nvc[*];
- declare real r[*];
- declare smallint si[*];
- declare time tm[*];
- declare timestamp ts[*];
- declare tinyint ti[*];
- declare varbinary(4) vb[*] having format $hex.;
- declare varchar(10) vc[*];

関連項目

- “変数宣言” (*SAS DS2 プログラマガイド*)
- “DS2 配列” (*SAS DS2 プログラマガイド*)
- “一時配列” (*SAS DS2 プログラマガイド*)
- “HAVING 句を使用した配列の宣言” (*SAS DS2 プログラマガイド*)

ステートメント:

- “DECLARE PACKAGE ステートメント” (1323 ページ)
- “DECLARE THREAD ステートメント” (1335 ページ)
- “RESIZEARRAY ステートメント” (1406 ページ)
- “VARARRAY ステートメント” (1434 ページ)

システムオプション:

- “DS2SCOND= システムオプション” (1454 ページ)

DECLARE PACKAGE ステートメント

パッケージ変数を作成し、オプションでパッケージインスタンスを作成します。

該当要素: すべてのプログラムブロック

カテゴリ: グローバルとローカル

注: 構文規則の角かっこは、オプションの引数を示します。角かっこの前のエスケープ文字(\ )は、構文に角かっこが必要であることを示します。配列の境界は角かっこ([ ])で囲む必要があります。

参照項目: 事前定義された DS2 パッケージに固有の DECLARE PACKAGE ステートメントの情報は、各パッケージのリファレンスセクションに記載されています。

構文

DECLARE [PRIVATE] **PACKAGE** *package* [(*table-options*)]
<package-variable> [... <package-variable>];

<package-variable::=

{ *scalar-package-variable* [(*constructor-arguments*)] }

| { *array-package-variable*\[array-bounds \] }

| { *array-package-variable*\[array-bounds \] := \[(*constructor-arguments-1*), ... (*constructor-arguments-N*) \] }

<array-bounds>::=

{ \[<array-bound> [, ...<array-bound> \]]}

| { \[* \] }

<array-bound>::=

{ [*dim-lower*:] *dim-upper* }

{ *dim-lower* := *n* }

{ *dim-upper*: *n* } | { DIM (*array-name*[, *n*]) }

必須引数

package
パッケージ名を指定します。*package* は、次のいずれかの形式をとります。

- *catalog.schema.package*
- *schema.package*
- *catalog.package*
- *package*

catalog
SQL カタログの ANSI SQL 規格が実装された、論理的に関連するスキーマをグループ化したデータコンテナーオブジェクトです。カタログは、データ編成階層の第 1 レベル(最上位)のグループ化メカニズムです。スキーマとともに使用され、名前を修飾する方法を提供します。

schema
SQL スキーマの ANSI SQL 規格が実装された、テーブルやビューなどのファイルや、ストアドプロシジャなどのデータソースでサポートされるその他のオブジェクトをグループ化するデータコンテナーオブジェクトです。スキーマは、catalog とともに使用されて、名前を修飾する方法を提供するグループ化オブジェクトを提供します。

package
パッケージの名前です。

要件　パッケージ名は、PACKAGE ステートメントで作成されたパッケージの名前と一致するか、事前定義された DS2 パッケージである必要があります。そうでない場合は、エラーが発生します。

パッケージの命名規則は、データソースに基づいています。詳細については、データソースのドキュメントを参照してください。

参照項目:　“PACKAGE ステートメント” (1392 ページ)

package-variable
スカラーパッケージ変数を作成するか配列パッケージ変数を作成するかを指定します。DECLARE PACKAGE ステートメントを使用して、複数のパッケージ変数を作成するように指定できます。

オプション引数

PRIVATE
パッケージ変数にパッケージ内からのみアクセスできることを指定します。

参照項目: “属性とメソッド” (*SAS DS2 プログラマガイド*)

scalar-package-variable
パッケージのインスタンスを参照できる名前を指定します。

array-package-variable\[*array-bounds* \]
パッケージの複数のインスタンスを参照できる名前を指定します。

注 配列パッケージ変数は一時配列です。したがって、固定または動的境界を持つことができます。詳細については、“一時配列” (*SAS DS2 プログラマガイド*) を参照してください。

dim-lower and *dim-upper*
固定配列パッケージ変数の配列境界の数とサイズを定義する正または負の整数を指定します。

注 次元の下限が指定されていない場合、下限はデフォルトで 1 になります。

参照項目: “一時配列変数の宣言”

DIM(*array-name*[, n])
配列パッケージ変数の上限のサイズが、DIM 関数呼び出しを使用して以前に宣言された配列パッケージ変数の次元内の要素の数によって決定されることを指定します。

array-name 以前に宣言された配列パッケージ変数の名前を指定します。

n 要素の数を知りたい多次元配列の次元を指定します。

制限事項 DIM 関数は、配列の上限を指定するために使用できる唯一の関数です。DIM 関数は、次元の下限を指定するためには使用できません。

ヒント *n* 値が指定されていない場合、DIM 関数は配列の最初の次元の要素の数を返します。

参照項目: DIM 関数

\[* \]
動的境界を持つ配列パッケージ変数を作成することを指定します。

参照項目: “動的一時配列変数の宣言”

table-options
DS2 プログラムがパッケージを作成するときに適用するオプションの引数を指定します。テーブルオプションの詳細については、“DS2 テーブルオプション” (1457 ページ)を参照してください。

constructor-arguments
パッケージインスタンスのコンストラクターに渡されるコンストラクター引数を指定します。

スカラーパッケージ変数: パッケージインスタンスは、パッケージ変数名の後にかっこのセットを指定することによって作成され、割り当てられます。かっこには、空のパラメーターリストまたはカンマ区切りの値のリストを含めることができます。

```
s()
s('gvar1',gvar2)
```

配列/パッケージ変数: コンストラクター引数の前に配列割り当て記号(:=)を付け、角かっこで囲む必要がります。配列の要素ごとに個別のコンストラクター引数を指定する必要があります。有効なコンストラクター引数は、かっこ内の値のセット、値のないかっこのセット、または空白スペースです。各要素のコンストラクター引数はカンマで区切る必要があります。

```
arr1[3] := [(), ,('var1',gvar2)];
```

初期化ブロックには、配列の要素よりも少ないコンストラクターを含めることも、多くのコンストラクターを含めることもできます。初期化ブロックに含まれるコンストラクターの数が配列の要素数よりも少ない場合、初期化されていない要素は NULL 参照のままになります。初期化ブロックに要素よりも多くのコンストラクターが含まれている場合、余分なコンストラクターは無視されます。空白スペースで示されたパッケージインスタンスは、NULL 参照として作成されます。

constructor-arguments の値は、変数のパッケージタイプのコンストラクターメソッドの 1 つに対する有効な引数である必要があります。

制限事項　現在、配列の複数の要素または配列全体を同じコンストラクターパラメーターで初期化する方法はありません。

注　宣言された配列サイズと初期化ブロック配列サイズの違いを示すログは生成されません。

詳細

パッケージの変数またはメソッドを別のプログラムで使用するには、その前に、パッケージ変数を宣言し、パッケージのインスタンスを構築する必要があります。DECLARE PACKAGE ステートメントは、スカラーパッケージ変数または配列パッケージ変数の作成をサポートします。スカラーパッケージ変数は、指定されたパッケージの単一インスタンスを参照します。配列パッケージ変数は、複数のパッケージインスタンスをグループ化するためのデータ構造です。配列パッケージ変数には次の特性があります。

- 配列の各要素はパッケージのインスタンスを参照できます。
- 配列の 2 つ以上の要素が同じパッケージインスタンスを参照できます。
- 配列パッケージ変数を使用すると、単一の変数を使用して複数のパッケージインスタンスを参照できます。

コンストラクター引数がパッケージ変数宣言で提供されている場合、パッケージインスタンスが構築され、パッケージ変数または配列パッケージ変数の要素が DECLARE PACKAGE ステートメントによって構築されたパッケージインスタンスを参照するように設定されます。

コンストラクター引数を指定するかわりに、宣言後に_NEW_演算子を使用してパッケージインスタンスを構築し、スカラーパッケージ変数または配列パッケージ変数の要素に割り当てることもできます。

スカラーパッケージ変数または配列パッケージ変数の要素は、割り当てステートメントを使用して、別のスカラー変数または配列要素の別の要素から既存のパッケージインスタンスへの参照をコピーすることによっても 設定できます。このような割り当てにより、複数のスカラー変数または配列要素が同じパッケージインスタンスを参照することになります。

配列パッケージ変数は、他の DS2 一時配列変数と似ています。配列パッケージ変数の要素は、一時配列の要素を指定できる場所ならどこでも指定できます。詳細については、“DS2 配列” (*SAS DS2 プログラマガイド*)を参照してください。

次の例では、スカラーパッケージ変数と配列パッケージ変数がどのように宣言され、インスタンス化されるかを比較します。すべての例はこのパッケージを参照しています。

```
package dog /inline;
  dcl varchar(100) name;
  dcl varchar(100) breed;
  method dog(varchar(100) name, varchar(100) breed);
  this.name = name;
  this.breed = breed;
  method dog();
 end;
end;
endpackage;
```

スカラーパッケージ変数を宣言し、パッケージインスタンスを構築する DECLARE PACKAGE ステートメントの例を次に示します。

```
declare package dog d('Benji', 'Mixed Breed');
```

前述のステートメントは、次の 2 つのステートメントと同等です。

```
declare package dog d;  1
 d = _new_ dog('Benji', 'Mixed Breed');      2
```

1 null パッケージ参照であるパッケージ DOG に対して D という名前のスカラーパッケージ変数を作成します。

2 パッケージ DOG インスタンスを構築し、構築されたパッケージインスタンスを参照するようにスカラーパッケージ変数 D を設定します。

DECLARE PACKAGE ステートメントを使用して配列パッケージ変数を宣言し、パッケージインスタンスを構築する方法の例を次に示します。

```
dcl package dog dogs[4] := \[('Lassie','Collie'), ('Benji, 'Mixed Breed'), ('Beethoven', 'St.
Bernard'), ('Toto', 'Cairn Terrier')\];
```

配列パッケージ変数 DOGS の 4 つの要素すべてを 初期化して値を割り当てるコンストラクターパラメーターが定義されています。

ここでは、DECLARE PACKAGE ステートメントを使用して配列パッケージ変数を宣言し、パッケージインスタンスを別のステップとして構築する方法の例を示します。

```
dcl package dog dogs[4];                    1
dogs[1]= _new_ dog('Lassie','Collie');      2
dogs[2]= dogs[1];                           3
dogs[4]= _new_ dog('Toto', 'Cairn Terrier');    4
```

1 4 つの配列要素で構成される固定配列である DOGS という名前の配列パッケージ変数を作成します。DECLARE PACKAGE ステートメントは、配列要素を null 参照として初期化します。

2 パッケージ DOG の NAME 変数と BREED 変数の値'Lassie'と'Collie'を定義するパッケージ DOG のインスタンスを構築し、構築されたパッケージインスタンスを参照するように配列パッケージ変数 DOGS の要素 1 を設定します。

3 配列要素 1 と配列要素 2 はどちらも、割り当てステートメントの後で同じパッケージインスタンスを参照します。

4 _NEW_演算子は、配列パッケージ変数 DOGS の要素 4 のパッケージインスタンスを構築するためにも使用されます。

配列パッケージ変数 DOGS を動的一時配列として作成するために DECLARE PACKAGE ステートメントを変更する方法の例を次に示します。

```
dcl package dog dogs[*];
resizearray dog[4];
```

DECLARE PACKAGE ステートメントで配列境界としてアスタリスク(*)を指定すると、RESIZEARRAY ステートメントで配列境界を設定できるようになります。

単一の DECLARE PACKAGE ステートメントを使用して、コンストラクターを含む複数のパッケージ変数(スカラーまたは配列)を作成できます。

注: パッケージ変数は変数スコーピングルールの対象となりますが、パッケージインスタンスは変数スコーピングルールの対象ではありません。詳細については、“パッケージ、スコープ、および有効期間” (*SAS DS2 プログラマガイド*)を参照してください。

例

注: このセクションの例では、DS2 プロシジャを使用して DS2 プログラムを SAS Compute Server にサブミットします。

例 1: 配列パッケージ変数の宣言

この例では、配列パッケージ変数を作成し、_NEW_演算子を使用して配列の要素をインスタンス化します。

```
proc ds2;
package dog / inline;                    /* 1 */
  dcl varchar(100) name;
  dcl varchar(100) breed;

  method dog(varchar(100) name, varchar(100) breed);
   /* Must use THIS.to distinguish global */
   /* variables NAME and BREED from parameters named NAME */
   /* and BREED */
    this.name = name;
    this.breed = breed;
```

```
  end;

  method describe() returns varchar(100);
   return 'My name is ' || name ||
         ', and I am a ' || breed || '.';
  end;
endpackage;

data _null_;
  method init();
  dcl varchar(100) description;
  dcl int i;

  dcl package dog dogs[4];                    /* 2 */
  dogs[1]= _new_ dog('Lassie','Collie');      /* 3 */
  dogs[2]= dogs[1];                           /* 4 */
  dogs[3]= _new_ dog('Beethoven', 'St. Bernard');
  dogs[1]= dogs[3];
  dogs[4]= _new_ dog('Toto', 'Cairn Terrier');

  do i = 1 to dim(dogs);                      /* 5 */
    if (not null(dogs[i])) then do;
      description = dogs[i].describe();
      put 'dogs[' i ']:' description;
    end;
  end;
 end;
enddata;
run;
quit;
```

1 パッケージ DOG は、NAME と BREED の 2 つの変数と、DOG と DESCRIBE の 2 つのメソッドを定義します。

2 配列パッケージ変数 DOGS は、パッケージ DOG のパッケージ変数として宣言されています。配列パッケージ変数 DOGS は、4 つの配列要素で構成される固定配列です。DECLARE PACKAGE ステートメントは、配列要素を null 参照として初期化します。

3 _NEW_演算子は、パッケージ DOG の変数の値'Lassie'と'Collie'を定義するパッケージ DOG のインスタンスを構築し、構築されたパッケージインスタンスを参照するように配列パッケージ変数 DOGS の要素 1 を設定します。_NEW_演算子は、配列パッケージ変数の要素 3 および 4 のパッケージインスタンスを構築するためにも使用されます。

4 割り当てステートメントは、配列パッケージ変数 DOGS の要素 1 によって参照されるパッケージインスタンスを、配列パッケージ変数の要素 2 に設定します。2 番目の割り当てステートメントは、配列パッケージ変数の要素 1 によって参照されるパッケージインスタンスを要素 3 によって参照されるパッケージインスタンスに変更するために使用されます。

5 DO ループは、パッケージメソッド DESCRIBE()を呼び出して、null ではない配列パッケージ変数 DOGS の各要素に DOG インスタンスの説明を出力します。

プログラムからの出力は次のとおりです。

```
dogs[ 1 ]: My name is Beethoven, and I am a St. Bernard.
dogs[ 2 ]: My name is Lassie, and I am a Collie.
dogs[ 3 ]: My name is Beethoven, and I am a St. Bernard.
dogs[ 4 ]: My name is Toto, and I am a Cairn Terrier.
```

例 2: コンストラクターを使用した配列パッケージ変数の宣言

この例では、配列パッケージ変数を作成し、DECLARE PACKAGE ステートメント内でコンストラクターを使用して配列の要素を開始します。

```
proc ds2;
package dog /inline;
  dcl varchar(100) name;
  dcl varchar(100) breed;
  method dog(varchar(100) name, varchar(100) breed);
  this.name = name;
  this.breed = breed;
  end;

  method describe() returns varchar(100);
   return 'My name is ' || name ||
         ', and I am a ' || breed || '.';
  end;
endpackage;

data _null_;
  method init();
  dcl varchar(100) name breed description;
  dcl int i;

 dcl package dog dogs[4] := [('Lassie','Collie'), ('Benji', 'Mixed Breed'),
('Beethoven','St. Bernard'), ('Toto','Cairn Terrier')];

  do i = 1 to dim(dogs);
   if (not null(dogs[i])) then do;
     description = dogs[i].describe();
     put 'dogs[' i ']:' description;
   end;
  end;
 end;
enddata;
run;
quit;
```

```
dogs[ 1 ]: My name is Lassie, and I am a Collie.
dogs[ 2 ]: My name is Benji, and I am a Mixed Breed.
dogs[ 3 ]: My name is Beethoven, and I am a St. Bernard.
dogs[ 4 ]: My name is Toto, and I am a Cairn Terrier.
```

例 3: スカラーパッケージ変数と配列パッケージ変数間の相互作用

スカラーパッケージ変数と配列パッケージ変数の要素は交換可能です。配列の要素とスカラー変数は、同じパッケージインスタンスを参照できます。

```
proc ds2;
package dog / inline;
  dcl varchar(100) name;
  dcl varchar(100) breed;
  method dog(varchar(100) name, varchar(100) breed);
   this.name = name;
   this.breed = breed;
  end;
  method describe() returns varchar(100);
   return 'My name is ' || name ||
         ', and I am a ' || breed || '.';
  end;
endpackage;

data _null_;
  method init();
   dcl package dog var;                    /* 1 */
   dcl package dog arr[4];
   var = _new_ dog('Lassie', 'Collie');      /* 2 */
   arr[1] = var;                           /* 3 */
   put var.name= arr[1].name=;               /* 4 */

   arr[2] = _new_ dog('Toto', 'Cairn Terrier'); /* 5 */
   var = arr[2];   /* VARIABLE = ARRAY ELEMENT  /* 6 */
   put var.name= arr[1].name= arr[2].name=;    /* 7 */
  end;
enddata;
run;
quit;
```

1 DECLARE PACKAGE ステートメントは、VAR という名前のスカラーパッケージ変数と ARR という名前の配列パッケージ変数を宣言します。ARR は 4 つの配列要素を持つ固定配列です。

2 _NEW_演算子はパッケージ DOG のインスタンスを構築し、それをパッケージ変数 VAR に割り当てます。

3 割り当てステートメントは、パッケージ変数 VAR によって参照されるのと同じパッケージインスタンスを参照するように配列要素 ARR[1]を設定します。

4 PUT ステートメントは、ドット表記を使用して、パッケージ変数 VAR および ARR[1]によって参照されるパッケージインスタンスの NAME 属性の値を出力します。どちらの参照も"Lassie"という名前を返します。

5 _NEW_演算子はパッケージ DOG のインスタンスを構築し、それを配列パッケージ変数 ARR[2]に割り当てます。

6 割り当てステートメントは、配列要素 ARR[2]によって参照されるのと同じパッケージインスタンスを参照するようにスカラーパッケージ変数 VAR を設定します。

7 PUT ステートメントは、ドット表記を使用して、パッケージ変数 VAR、ARR[1]、および ARR[2]によって参照されるパッケージインスタンスの NAME 属性を出力します。

```
var.name=Lassie arr[1].name=Lassie
var.name=Toto arr[1].name=Lassie arr[2].name=Toto
```

例 4: DECLARE PACKAGE ステートメントでの複数のパッケージ変数の宣言

この例では、単一の DECLARE PACKAGE ステートメントで各タイプの複数のパッケージ変数を宣言します。

```
proc ds2;
 package dog /inline;
   dcl varchar(100) name;
   dcl varchar(100) breed;
   method dog(varchar(100) name, varchar(100) breed);
   this.name = name;
   this.breed = breed;
   end;

   method describe() returns varchar(100);
    return 'My name is ' || name ||
          ', and I am a ' || breed || '.';
   end;
 endpackage;

 data _null_;
   method init();
   dcl varchar(100) name breed description;
   dcl int i;
   dcl int addedone;

   dcl package dog genx[5] := [('Lassie','Collie'), ('Benji', 'Mixed Breed'), ('Toto','Cairn
Terrier'), ('Digby','Old English Sheepdog'), ('Hooch','Dogue de Bordeaux')]
                cartoon('Balto', 'Wolf Hybrid')
                millenials[10] := [('Shadow', 'Golden Retriever'), ('Chance', 'American Bulldog'),
('Beethoven','St. Bernard'), , ('Buddy', 'Golden Retriever'), ('Copper','Bloodhound')]
                boomers[5] := [('Old Yeller', 'Black Mouth Cur'), ('Lassie','Collie'), ('Charlie',
'Bearded Collie Mix'), ('Chinook', 'Alaskan Malamute'),('Toto','Cairn Terrier')];

   put 'Famous Dogs of Generation X:';
   do i = 1 to dim(genx);
    if (not null(genx[i])) then do;
      description = genx[i].describe();
      put '  [' i ']:' genx[ i ].name ',' genx[ i ].breed;
    end;
   end;

   put 'Famous Dogs of the Millenials Generation:';
   do i = 1 to dim(millenials);
    if (not null(millenials[i])) then do;
      description = millenials[i].describe();
```

```
      put '  [' i ']:' millenials[ i ].name ',' millenials[ i ].breed;
    end;
  end;

  put 'Famous Dogs of the Boomers Generation:';
  do i = 1 to dim(boomers);
    if (not null(boomers[i])) then do;
      description = boomers[i].describe();
      put '  [' i ']:' boomers[ i ].name ',' boomers[i].breed;
    end;
  end;

  put 'A Famous Dog of Cartoon:';
  put '  ' cartoon.name ',' cartoon.breed;

  put 'Adding Cartoon to Millenials';
  addedone = 0;
  do i = 1 to dim(millenials);
    if ((addedone < 1) AND null(millenials[i])) then do;
      millenials[ i ] = cartoon;
      addedone = 1;
    end;
  end;

  put 'Famous Dogs of the Millenials Generation:';
  do i = 1 to dim(millenials);
    if (not null(millenials[i])) then do;
      description = millenials[i].describe();
      put '  [' i ']:' millenials[ i ].name ',' millenials[ i ].breed;
    end;
  end;

end;
enddata;
run;
quit;
```

```
Famous Dogs of Generation X:
   [ 1 ]: Lassie , Collie
   [ 2 ]: Benji , Mixed Breed
   [ 3 ]: Toto , Cairn Terrier
   [ 4 ]: Digby , Old English Sheepdog
   [ 5 ]: Hooch , Dogue de Bordeaux
Famous Dogs of the Millenials Generation:
   [ 1 ]: Shadow , Golden Retriever
   [ 2 ]: Chance , American Bulldog
   [ 3 ]: Beethoven , St. Bernard
   [ 5 ]: Buddy , Golden Retriever
   [ 6 ]: Copper , Bloodhound
Famous Dogs of the Boomers Generation:
   [ 1 ]: Old Yeller , Black Mouth Cur
   [ 2 ]: Lassie , Collie
   [ 3 ]: Charlie , Bearded Collie Mix
   [ 4 ]: Chinook , Alaskan Malamute
   [ 5 ]: Toto , Cairn Terrier
A Famous Dog of Cartoon:
    Balto , Wolf Hybrid
Adding Cartoon to Millenials
Famous Dogs of the Millenials Generation:
   [ 1 ]: Shadow , Golden Retriever
   [ 2 ]: Chance , American Bulldog
   [ 3 ]: Beethoven , St. Bernard
   [ 4 ]: Balto , Wolf Hybrid
   [ 5 ]: Buddy , Golden Retriever
   [ 6 ]: Copper , Bloodhound
```

関連項目

例

- "Hello World!"プログラム – ユーザー定義パッケージの場合 (27 ページ)
- "Hello World!"プログラム – 暗黙的なループの場合 (29 ページ)
- "Hello World!"プログラム – 複数のパッケージインスタンスの場合 (31 ページ)

概念

- "DS2 パッケージ内のドット表記" (*SAS DS2 プログラマガイド*)
- "ユーザー定義パッケージ" (*SAS DS2 プログラマガイド*)
- "パッケージのコンストラクターとデストラクター" (*SAS DS2 プログラマガイド*)

演算子:

- "_NEW_ 演算子" (1289 ページ)

ステートメント:

- "DECLARE PACKAGE ステートメント: FCMP パッケージ" (1637 ページ)
- "DECLARE PACKAGE ステートメント: ハッシュパッケージ" (1656 ページ)
- "DECLARE PACKAGE ステートメント: ハッシュ反復子パッケージ" (1665 ページ)
- "DECLARE PACKAGE ステートメント: ロガーパッケージ" (1807 ページ)

- “DECLARE PACKAGE ステートメント: 行列パッケージ” (1845 ページ)
- “DECLARE PACKAGE ステートメント: SQLSTMT パッケージ” (1957 ページ)
- “PACKAGE ステートメント” (1392 ページ)

DECLARE THREAD ステートメント

スレッドのインスタンスを作成します。

カテゴリ: ローカル

構文

DECLARE THREAD *thread* [(*table-options*)] *instance*(*argument*) [... *instance* (*argument*)];

必須引数

thread
スレッド名を指定します。*thread* は、次のいずれかの形式をとります。

- *catalog.schema.thread*
- *schema.thread*
- *catalog.thread*
- *thread*

catalog
SQL カタログの ANSI SQL 規格の実装です。SQL カタログは、論理的に関連するスキーマをグループ化するデータコンテナーオブジェクトです。カタログは、データ編成階層の第 1 レベル(最上位)のグループ化メカニズムです。スキーマとともに使用され、名前を修飾する方法を提供します。

schema
SQL スキーマの ANSI SQL 規格の実装です。SQL スキーマは、テーブルやビューなどのファイルや、ストアドプロシジャなどのデータソースでサポートされるその他のオブジェクトをグループ化するデータコンテナーオブジェクトです。スキーマは、catalog とともに使用されて、名前を修飾する方法を提供するグループ化オブジェクトを提供します。

thread
スレッドの名前です。

要件 スレッド名は、THREAD ステートメントで作成されたスレッドの名前と一致する必要があります。一致しない場合、エラーが発生します。

スレッドの命名規則は、データソースに基づきます。詳細については、データソースのドキュメントを参照してください。

参照項目: “スレッド処理の概要” (*SAS DS2 プログラマガイド*)

instance
スレッドのインスタンスを識別する名前を指定します。

argument
instance で使用される引数を指定します。

オプション引数

table-options
DS2 プログラムがスレッドを作成するときに適用するオプションの引数を指定します。テーブルオプションの詳細については、“DS2 テーブルオプション” (1457 ページ)を参照してください。

詳細

スレッドが宣言されると、スレッドを表す変数はスレッドのインスタンスと見なすことができます。これは、2 つの異なるスレッド変数がスレッドの 2 つの完全に別個のコピーを表すことを意味します。

スレッド変数はグローバルスコープにのみ表示できます。それ以外の場合はエラーが返されます。DECLARE ステートメントで指定されたスレッドが存在しない場合、エラーが返されます。

この例では、スレッド work.t がインスタンス化され、thread_name という名前が付けられています。

```
declare thread work.t thread_name;
```

スレッドのインスタンスが作成されると、それを SET FROM ステートメントで使用できます。詳細については、“THREAD ステートメント” (1429 ページ)を参照してください。 スレッドの詳細については、“スレッド処理の概要” (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- “スレッド処理” (*SAS DS2 プログラマガイド*)

ステートメント:

- “SET FROM ステートメント” (1422 ページ)
- “THREAD ステートメント” (1429 ページ)

DO ステートメント

1 単位として実行されるステートメントのグループを指定します。

カテゴリ: ローカル

構文

DO [<numeric-data-type>][*index-variable* = <index-variable-clause>]
[<conditional-clause>]
[, ...[<index-variable-clause>] [<conditional-clause>]];

...*statement-list*...

END [*end-label*] ;

< numeric-data-type >::=

<exact-numeric-type> | <approximate-numeric-type>

<exact-numeric-type> ::=

{**INT** | **BIGINT** | **SMALLINT** | **TINYINT**
| **DECIMAL** [(*precision* [,*scale*])] | **NUMERIC** [(*precision* [,*scale*])] }

<approximate-numeric-type> ::=

{ **DOUBLE** | **DOUBLE PRECISION** | **FLOAT** | **REAL** }

<index-variable-clause>::=

start [TO *stop* [BY *increment*]]

<conditional-clause>::=

WHILE (*expression*) | **UNTIL** (*expression*)

引数なし

インデックス変数の引数と句を含まない DO ステートメントは、DO グループ処理の最も単純な形式です。DO ステートメントと END ステートメントの間のステートメントは、**DO グループ**と呼ばれます。単純な DO ループでは、DO グループ内のステートメントは 1 回だけ実行されます。DO グループ内に DO ステートメントをネストできます。単純な DO ステートメントは、IF-THEN/ELSE ステートメント内でよく使用され、IF 条件が真であるか偽であるかに応じて実行されるステートメントのグループを指定します。

必須引数

statement-list
有効な DS2 ステートメントを指定します。

オプション引数

INT | BIGINT | SMALLINT | TINYINT
整数変数または配列を指定します。

別名　INT の場合は INTEGER

参照項目: “DS2 データ型” (*SAS DS2 プログラマガイド*)

DECIMAL[(*precision* [, *scale*])] | NUMERIC[(*precision* [, *scale*])]
正確な数値変数または配列を指定します。

precision
小数点の左側と右側の両方に保存できる最大合計桁を指定します

注　すべてのデータソースが 52 桁の精度をサポートできるわけではありません。

scale
小数点の右側に保存できる最大桁数を指定します

範囲　0–*precision*

注　*scale* は *precision* は以下です。

参照項目: “DS2 データ型” (*SAS DS2 プログラマガイド*)

DOUBLE | DOUBLE PRECISION | FLOAT | REAL
浮動小数点変数または配列を指定します。

参照項目: “DS2 データ型” (*SAS DS2 プログラマガイド*)

index-variable
ループのインデックスカウンターとして使用される変数を指定します。

注意
DO グループ内のインデックス変数を変更しないでください。 反復 DO グループ内のインデックス変数を変更すると、無限ループが発生する可能性があります。

要件　変数は数値に解決される必要があります。

ヒント　変数がローカル変数として宣言されていない場合、明示的に削除されない限り、作成中のテーブルに保存されます。ローカル変数の詳細については、“DS2 識別子のスコープ” (*SAS DS2 プログラマガイド*)を参照してください。

インデックス変数は THIS 式にすることができます。 詳細については、“THIS 式” (2115 ページ)を参照してください。

end-label
END ステートメントは DO ループを閉じます。オプションの *end-label* 引数は、識別子を指定します。Labels ステートメントを使用して作成されたこのラベルは、DO ステートメントの直前のラベルと一致する必要があります。一致しない

場合、エラーが発生します。詳細については、“Labels ステートメント” (1366 ページ)を参照してください。

句

< index-variable-clause >
DO グループが実行される回数を決定する数値スカラー式または一連の式を指定します。

start
インデックス変数の初期値を指定します。 *start* は、数値に解決される任意の式にすることができます。

TO *stop* なしで使用する場合、*start* の値は、次の形式で表される一連のアイテムになります。

item-1 <, ...item-n>

アイテムは、数値または数値を生成する式にすることができます。DO グループは、リスト内の値ごとに 1 回ずつ実行されます。WHILE 条件が追加されると、その直後のアイテムにのみ適用されます。

要件 TO *stop* とともに使用する場合、*start* は数値または数値を生成する式である必要があります。

注 DO グループは、最初に *start* に等しい *index-variable* で実行されます。ループの最初の実行の前に、*start* の値が評価されます。

index-variable が整数型で、*start* が浮動小数点型の場合、*start* の値は INTEGER 型に変換され、精度が低下する可能性があります。

TO *stop*
インデックス変数の終了値を指定します。*stop* は、数値に解決される任意の式にすることができます。

注 *index-variable* の値が *stop* の値を超えるか、DO ステートメントで指定された WHILE または UNTIL 句が満たされるか、あるいは DO グループ内のステートメントがループ外の実行を指示するまで、*increment* の値に基づいて実行が継続されます。ループの最初の実行の前に、*stop* の値が評価されます。

index-variable が整数型で、*stop* が浮動小数点型の場合、*stop* の値は INTEGER 型に変換され、精度が低下する可能性があります。DO グループ内で行われた *stop* の変更は、反復回数には影響しません。

BY *increment*
index-variable の増分または減分を制御する正または負の値を指定します。*increment* は、数値に解決される任意の式にすることができます。

注 *increment* の値は、ループの実行前に評価されます。*increment* が正数の場合、*start* はループの下限、*stop* はループの上限にする必要があります。*increment* が負数の場合、*start* はループの上限、*stop* はループの下限にする必要があります。増分が指定されていない場合、インデックス変数は 1 ずつ増分されます。

index-variable が整数型で、*increment* が浮動小数点型の場合、*increment* の値は INTEGER 型に変換され、精度が低下する可能性があります。DO グループ内で行われた増分の変更は、反復回数には影響しません。

<conditional-clause>
真または偽を返す句を指定します。

WHILE (*expression*)
条件が真である間、DO グループステートメントを繰り返し実行します。

注 WHILE 句はループの実行前に毎回評価されるため、グループ内のステートメントは、式が真である間は繰り返し実行されます。式が最初に評価されたときに偽の場合、DO ループは 1 回も繰り返されません。

参照項目: 付録 2, “DS2 式” (2095 ページ)

UNTIL (*expression*)
条件が真になるまで、DO グループステートメントを繰り返し実行します。

注 UNTIL 句はループの実行後に毎回評価されるため、グループ内のステートメントは、式が真になるまで繰り返し実行されます。DO ループは常に少なくとも 1 回は反復されます。

参照項目: 付録 2, “DS2 式” (2095 ページ)

詳細

DO ステートメントを使用すると、ステートメントのグループを 1 単位として実行できます。反復句または条件句が指定されている場合、このステートメントのグループは複数回実行できます。

整数インデックス変数を使用した DO ループの反復は、INTEGER 演算を使用して実行されます。それ以外の場合、DO ループの反復は DOUBLE 演算を使用して実行されます。DOUBLE 型の表現は 2 進数であるため、不正確な数値が累積されると、予想される回数の DO ループの実行を妨げるのに十分な誤差が発生する可能性があります。たとえば、このループは予想される回数実行されない場合があります。

```
do i = 0.001 to 10 by 0.001;
```

詳細については、“数値精度” (*SAS プログラマガイド: エッセンシャル*)を参照してください。

DO ステートメントはスコープブロックを定義し、DO ステートメントで宣言された変数が DO ステートメントのスコープに対してローカルなスコープを持つようにします。

DO ステートメントには次の 3 つの形式があります。

- 句のない DO ステートメントは、DO グループ処理の最も単純な形式です。この形式では、ステートメントのグループが 1 単位として実行され、通常は IF-THEN/ELSE ステートメントの一部として実行されます。

- 反復 DO ステートメントは、インデックス変数の値に基づいて、DO ステートメントと END ステートメントの間のステートメントを繰り返し実行できます。
- DO ステートメントは、条件句が真である間、DO ループ内のステートメントを繰り返し実行し、DO ループの反復前に毎回条件をチェックできます。DO UNTIL 形式では、ループの末尾の条件が評価され、DO WHILE 形式では、ループの先頭の条件が評価されます。

  ループの最終値は、最大でも指定された TO 値です。たとえば、do i = 1 to 10 では、ループが 10 回実行されます。これは他の言語とは異なります(たとえば、C 言語では、最後の反復が TO 値より小さい)。

CONTINUE ステートメントを DO グループ内で使用すると、DO ステートメントの次の反復で実行をすぐに継続できます。

LEAVE ステートメントを DO ステートメント内で使用して、指定されたターゲット DO ステートメントの直後のステートメントまたは現在の DO ステートメントのいずれかに実行を転送できます。

DO ステートメント内でループカウンターを宣言できます。次に例を示します。

```
do double i=0 to 10;
```

変数宣言のスコープは、ループの反復に対してローカルです。

注: 数値データ型のみが許可されます。

例

例 1: 句のない DO ステートメント

```
do;
  dcl int i j;
    i = 2;
    j = i + 5
end;
```

例 2: インデックス変数句のある DO

- ```
  do j = 1 to 10 by 2;
    i + j;
  end;
  ```
- ```
  do k = 11 to 0 by -3;
    i + k;
  end;
  ```
- ```
  x = -2;
  y = -1;
  do k = 11 to 0 by x + y;
    dcl int i;
    if k < 5 then i = k;
    else
    i = k - 1;
  end;
  ```
  ```

```
dcl double i;
do i = 0 to 5 by 0.5;
  put i=;
  /* the values output are */
  /* 0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5   */
end;
```

例 3: 連続値を含むインデックス変数句のある DO

```
dcl int i;
do i = 0 to 10 by 3, 6, 7, 8;
   put i=;
   /*  the values output are*/
   /*  0, 3, 6, 9, 6, 7, 8       */
end;
```

例 4: WHILE 句のある DO ステートメント

```
k = 1;
do while(k <= 5);
  k = k + 1;
end;
```

例 5: インデックス変数と WHILE 句の両方がある DO ステートメント

```
j = -2;
do k = 1 to 10 by 2 while (j <= 0);
 j = j + 1;
end;
```

例 6: UNTIL 句のある DO ステートメント

```
k = 1;
do until(k > 6);
  k = k + 1;
end;
```

例 7: インライン宣言のある DO ステートメント

この例では、カウンター i を DOUBLE として宣言しています。

```
data _null_;
  method init();
    do double i = 1.1 to 3.1;
      put i=;
    end;
  end;
enddata;
run;
```

次の行が SAS ログに書き込まれます。

```
1.1
2.1
3.1
```

例 8: 逆方向にカウントするインライン宣言のある DO ステートメント

この例では、カウンター i を DOUBLE として宣言しています。

```
data _null_;
  method init();
    do double i = 1 to '-3' by -1;
      put i=;
    end;
  end;
enddata;
run;
```

次の行が SAS ログに書き込まれます。

```
1
0
-1
-2
-3
```

例 9: インライン宣言と非数値引数のある DO ステートメント

次の例では、非数値が DOUBLE ループインデックスに割り当てられています。その結果、欠損値がログに書き込まれます。

```
data _null_;
  method init();
    do double i = 'abc', 123, 'def', 'ghi';
      put i=;
    end;
  end;
enddata;
run;
```

次の行が SAS ログに書き込まれます。

```
.
123
.
.
```

関連項目

ステートメント:

- “CONTINUE ステートメント” (1309 ページ)
- “Labels ステートメント” (1366 ページ)

- “LEAVE ステートメント” (1367 ページ)

DROP ステートメント

出力テーブルから列を除外します。

カテゴリ: グローバル

注: このステートメントは、メソッド内では使用できません。

構文

DROP *column-list* | *vararray*;

必須引数

column-list
出力テーブルから省略する 1 つ以上の列の名前を指定します。

制限事項 *col1–coln* 形式の番号付き範囲リストおよび *col:*形式の名前接頭辞リストはサポートされていません。

vararray
変数配列の名前を指定します。

参照項目: “VARARRAY ステートメント” (1434 ページ)

詳細

DROP ステートメントは、同じ DS2 プログラム内で作成されたすべてのテーブルに適用され、データ、スレッド、またはパッケージプログラムのグローバルステートメントセクションにのみ記述できます。DROP ステートメントの列は、DS2 プログラムで処理できます。DROP または KEEP ステートメントが記述されない場合、DS2 プログラムで作成されるすべてのテーブルにすべての 列が含まれます。同じ DS2 プログラム内で DROP ステートメントと KEEP ステートメントの両方を使用しないでください。

比較

- DROP ステートメントは、DATA ステートメントで指定されているすべての出力テーブルに適用されます。一部のテーブルから列を除外し、他のテーブルからは除外しないようにするには、DATA ステートメントで DROP=テーブルオプションを使用します。

- KEEP ステートメントは、出力テーブルに書き込む列のリストを指定する並列ステートメントです。含める列の数が省略する列の数よりも大幅に少ない場合は、DROP ステートメントのかわりに KEEP ステートメントを使用します。
- KEEP および DROP ステートメントは、出力テーブルに含める列または出力テーブルから除外する列を選択します。サブセット化 IF ステートメントは行を選択します。

例

- これらの例は、DROP ステートメントを使用して列をリストするための正しい構文を示しています。
  - drop time shift batchnum;
  - drop grade1 grade2 grade3 grade4;
- この例では、列 PURCHASE および REPAIR が処理に使用されますが、出力テーブル INVENTRY には書き込まれません。

```
data inventry;
  drop purchase repair;
  method run();
    set table-specification;
    totcost=sum(purchase,repair);
  end;
enddata;
```

- この例では、変数配列 x の最初の 3 つの変数が出力から削除されます。

```
/* drop x1, x2, x3 */
proc ds2;
data mytest;
  vararray double x[10];
  drop x1-x3;
  method init();
    do i = 1 to 10;
      x[i] = i;
    end;
  output;
  end;
enddata;
run;
quit;
```

関連項目

ステートメント:

- “KEEP ステートメント” (1364 ページ)

テーブルオプション:

- “DROP= テーブルオプション” (1483 ページ)

DROP PACKAGE ステートメント

DS2 パッケージを削除します。

カテゴリ: ブロック制御

制限事項: このステートメントは、他のプログラミングブロックの外部で使用する必要があります。プログラミングブロックは、DATA...ENDDATA、THREAD...ENDTHREAD、または PACKAGE...ENDPACKAGE ステートメントで区切られます。

構文

DROP PACKAGE *package* [(*table-options*)];
RUN;

必須引数

package
削除するパッケージの名前を指定します。

オプション引数

table-options
DS2 プログラムがパッケージを削除するときに適用するオプションの引数を指定します。テーブルオプションの詳細については、“DS2 テーブルオプション” (1457 ページ)を参照してください。

詳細

DROP PACKAGE ステートメントは、指定された DS2 パッケージのコードを含むテーブルを削除します。テーブルは、PACKAGE ステートメントを使用して事前に作成されている必要があります。

注: RUN ステートメントは、DROP PACKAGE ステートメントの後に必要です。

例

これらの例は、DS2 パッケージを削除するための構文を示しています。

```
drop package mypkg;
run;
```

```
drop package mypkg (pw='n1234');
run;
```

関連項目

ステートメント:

- “PACKAGE ステートメント” (1392 ページ)

DROP THREAD ステートメント

DS2 スレッドプログラムを削除します。

カテゴリ: ブロック制御

制限事項: このステートメントは、他のプログラミングブロックの外部で使用する必要があります。プログラミングブロックは、DATA...ENDDATA、THREAD...ENDTHREAD、または PACKAGE...ENDPACKAGE ステートメントで区切られます。

構文

DROP THREAD *thread* [(*table-options*)];

必須引数

thread
削除するスレッドの名前を指定します。

オプション引数

table-options
DS2 プログラムがスレッドを削除するときに適用するオプションの引数を指定します。テーブルオプションの詳細については、“DS2 テーブルオプション” (1457 ページ)を参照してください。

詳細

DROP THREAD ステートメントは、指定された DS2 スレッドのコードを含むテーブルを削除します。テーブルは、THREAD ステートメントを使用して事前に作成されている必要があります。

例

これらの例は、DS2 プログラムスレッドを削除するための構文を示しています。

```
drop thread mythread;
drop thread mythread (pw='n1234');
```

関連項目

ステートメント:

- “THREAD ステートメント” (1429 ページ)

DS2_OPTIONS ステートメント

DS2 プログラムのデフォルトの動作の指定または変更

カテゴリ: 動作

要件 特に指定がない限り、DS2_OPTIONS ステートメントは、DATA、PACKAGE、または THREAD ステートメントの直前に記述する必要があります。DS2_OPTIONS ステートメントは、次の DATA、PACKAGE、または THREAD プログラムブロックにのみ適用されます。これにより、同じ DS2 プログラム内のプログラムブロックが異なるオプション値を持つことができます。

操作: 保存された THREAD または PACKAGE プログラムブロックが別のプログラムブロックによってロードされ、ロードされたプログラムブロックが DS2_OPTIONS ステートメントでオプションを明示的に設定しない場合、ロードされたプログラムブロックは、それをロードしたプログラムブロックからオプション設定を継承します。詳細については、“オプションの継承” (1354 ページ)を参照してください。

注: DS2_OPTIONS オプションのいくつかは、DS2 プロシジャオプションまたは DS2 CAS アクションパラメーターとして設定することもできます。DS2 プロシジャまたは CAS アクション、およびこのステートメントでオプションが指定されている場合、このステートメントの設定が優先されます。ただし、すべてのオプションが同じように動作するわけではありません。詳細については、“オプション範囲”を参照してください。

構文

DS2_OPTIONS *options*;

必須引数

options
1 つ以上の DS2 オプションを指定します。*options* は、次のいずれかの値になります。

BIGINTPROCESSING=YES | NO

BIGINT 処理を使用するかどうかを指定します。

BIGINTPROCESSING=NO は、BIGINIT 処理を無効にします。無効にすると、値は組み込み関数と出力形式に渡される前に自動的に DOUBLE に強制されます。15 桁または 16 桁を超える大きな値の場合、精度が失われる可能性があります。BIGINTPROCESSING=YES は、BIGINT 処理を有効にします。これにより、DOUBLE への自動的な強制が削除され、BIGINT 値が完全な精度で保持されます。

デフォルト NO

注 BIGINTPROCESSING は 2025.12 から新しく追加されました。

DS2 プログラム内のすべてのパッケージ、スレッド、データブロックで、従来の BIGINT の動作または新しい BIGINT の動作を使用する必要があります。パッケージまたはスレッドが指定された動作から逸脱した場合、不一致があるという警告が発行され、データブロックから継承された設定が選択されます。

例
```
ds2_options bigintprocessing=yes;
data _null_;
  dcl package mypkg1 p();
enddata;
run;
```

重要 整数値(BIGINT、INTEGER、SMALLINT、TINYINT)が組み込み関数や出力形式に渡される前に、DOUBLE への暗黙的な変換に依存している場合、BIGINTPROCESSING=YES を指定すると、以前のリリースとは動作が異なる可能性があります。

従来の CAS の動作を前提とする既存のプログラムまたはワークフローとの互換性を維持するには、BIGINTPROCESSING=NO を使用する必要があります。

DIVBYZERO=ERROR | IGNORE

DS2 がゼロ除算演算を処理する方法を指定します。

ERROR 行の処理を停止し、SAS ログにエラーを書き込みます。

IGNORE 欠損値または null 値を結果セットに書き込みます。SAS ログにメッセージは書き込まれません。

デフォルト ERROR

LOGICALEXPR=STANDARD | OPTIMIZED

次のプログラムブロックの論理 AND 式と OR 式の評価方法を指定します。

STANDARD DS2 が結果を返す前に論理 AND または OR 式の両方のオペランドを評価できることを指定します。

OPTIMIZED　DS2 が、可能な場合は最初のオペランドに基づいて結果を返すことにより、論理 AND および OR 式のオペランドの評価を最適化することを指定します。

デフォルト　STANDARD

操作

注　このオプションは、DS2 プロシジャまたは DS2 CAS アクションでも設定できます。

ヒント　2 番目のオペランドの評価をバイパスすると、そうでなければ無効になる計算を回避できます。

参照項目:　“DS2 での短絡評価” (2125 ページ)

MISSING_NOTE
無効な関数引数が欠損値を生成すると、SAS ログにメモを書き込みます。

デフォルト　無効な関数引数が欠損値を生成すると、エラーメッセージが SAS ログに書き込まれます。

MODE=ANSI | SAS
固定文字列と DOUBL 型の列で、存在しない値の処理方法を指定します。

ANSI　存在しない値を NULL 値として扱います。

SAS　存在しない値を SAS 欠損値として扱います。

デフォルト　SAS

注　このオプションは、DS2 プロシジャまたは DS2 CAS アクションでも設定できます。

参照項目:　“DS2 による null および SAS 欠損値の処理方法” (*SAS DS2 プログラマガイド*)

MSGLIMIT=n | MIN | MAX
DS2 プログラムの実行中に SAS ログに書き込むことができるエラー、警告、およびメモメッセージの最大数を指定します。

n　DS2 プログラムの実行中に生成されたエラー、警告、およびメモメッセージがログに書き込まれる数を示す整数です。メッセージの制限に達すると、DS2 は制限に達したことを示すエラーメッセージを SAS ログに書き込みます。プログラムは完了まで実行を継続します。ただし、その後のエラー、警告、およびメモのメッセージは SAS ログに書き込まれません。*n* は、1 (*n*)、1,024 (*n*K)、1,048,576 (*n*M)、1,073,741,824 (*n*G)のいずれかで指定できます。たとえば、値 8 は 8 メッセージを指定し、値 8K は 8,192 メッセージを指定します。

MIN　DS2 プログラムの実行中に生成されるエラー、警告、およびメモのメッセージがログに書き込まれないことを指定します。MIN を指定することは、MSGLIMIT=0 を指定することと同じです。

MAX　DS2 プログラムの実行中に生成されるすべてのエラー、警告、およびメモのメッセージがログに書き込まれることを指定します。

デフォルト　プログラムの実行ごとに 1,024 メッセージ。

制限事項　MSGLIMIT=オプションは、MSGORDER=TEMPORAL が設定されている場合にのみ有効です。MSGORDER=STANDARD は、プログラムの実行ごとに最大 1,024 個のメッセージを出力します。この数には、プログラムのコンパイルと実行のメッセージが含まれます。

注　MSGLIMIT=オプションは、DS2 プログラムの実行中に生成される診断メッセージに適用されます。このオプションは、DS2 プログラムのコンパイル中に生成されるエラーメッセージ、PUT ステートメントによって生成されるメッセージ、および SAS ログ機能に書き込まれるメッセージには影響しません。

このオプションは、DS2 プロシジャまたは DS2 CAS アクションでも設定できます。

ヒント　MSGLIMIT=によって抑制されたメッセージは、App.TableServices.DS2.Runtime.Log ロガーを有効にすることでキャプチャできます。プログラムを実行する前に、レベル INFO でロガーを有効にしてください。

重要　MSGORDER=および MSGLIMIT=オプションは、DS2 プログラム全体に適用されます。これらの設定は、プログラムブロックによって 変わることはありません。

MSGORDER=STANDARD | TEMPORAL

DS2 がエラー、警告、およびメモのメッセージを生成時に SAS ログに書き込むか、DS2 プログラムの完了後に書き込むかを指定します。

STANDARD　DS2 プログラムの完了後に DS2 がメッセージを書き込むことを指定します。PUT ステートメントメッセージは、ログの診断メッセージの前にあります。

TEMPORAL　DS2 がメッセージを生成時に書き込むことを指定します。PUT ステートメントメッセージは、ログ内の診断メッセージに散在しています。

デフォルト　STANDARD

制限事項　MSGORDER=および MSGLIMIT=オプションは、DS2 プログラム全体に適用されます。これらの設定は、プログラムブロックによって 変わることはありません。

注 このオプションは、DS2 プロシジャまたは DS2 CAS アクションでも設定できます。

ヒント メッセージの時間的な順序付けは、PUT ステートメントメッセージとエラー、警告、およびメモのメッセージを生成された順序で報告することにより、DS2 プログラムのデバッグを支援できます。DS2 プログラムのロジックフローの戦略的ポイントに PUT ステートメントを挿入して、エラー、警告、またはメモメッセージを生成するエラー状態の原因を診断するのに役立てることができます。

重要 MSGORDER=および MSGLIMIT=オプションを指定する DS2_OPTIONS ステートメントは、オプションが認識されるために、DS2 プログラムのルートプログラミングブロックの前に置く必要があります。DS2 プログラムに DATA プログラミングブロックが含まれている場合、ルートプログラミングブロックは DATA プログラミングブロックです。DS2 プログラムに PACKAGE プログラミングブロックのみが含まれている場合、最後のパッケージプログラミングブロックがルートと見なされます。

REPORTLINE=YES | NO

次のプログラムブロックのログ行番号を DS2 プログラム実行メッセージで報告するかどうかを指定します。ログ行番号は、エラー、警告、およびメモのメッセージに含まれ、メッセージが生成されたときに実行されていた DS2 ステートメントの場所を示します。ストアドパッケージおよびスレッドについて報告される行番号は、SAS ログの行番号ではなく、パッケージまたはスレッドブロックの先頭からオフセットされた行番号です。

YES
DS2 プログラム実行メッセージの行番号を生成します。

NO
ログ行番号の生成を抑制します。

デフォルト YES

注 REPORTLINE=は、DS2 プログラムのステートメントの実行中に DS2 サービスによって生成されるメッセージを制御します。プログラムの前処理と後処理の間、または SAS/ACCESS サービスによって生成されるメッセージには影響しません。REPORTLINE=NO は、DS2 プログラムのコンパイルメッセージで報告される行番号には影響しません。

このオプションは、DS2 プロシジャまたは DS2 CAS アクションでも設定できます。

SAS

固定文字列と DOUBLE 型の列で、存在しない値を SAS 欠損値として処理することを指定します。

デフォルト DS2 は、存在しない値を SAS の欠損値として処理します。

参照項目: “DS2 による null および SAS 欠損値の処理方法” (*SAS DS2 プログラマガイド*)

重要 SAS オプションは MODE=オプションに置き換えられます。SAS オプションは下位互換性のためにサポートされています。

SCOND

DS2 変数宣言 strict モードの SAS ログに表示されるメッセージのレベルを指定します。strict モードでは、すべての変数を DS2 プログラムで宣言する必要があります。DS2 変数宣言の strict モードの詳細については、“変数宣言” (*SAS DS2 プログラマガイド*)を参照してください。

WARNING 警告メッセージを SAS ログに書き込みます。

NONE SAS ログにメッセージは書き込まれません。

NOTE SAS ログにメモを書き込みます。

ERROR エラーメッセージを SAS ログに書き込みます。

デフォルト デフォルトは、DS2SCOND=システムオプションによって決定されます。DS2SCOND=のデフォルトは WARNING です。詳細については、“DS2SCOND= システムオプション” (1454 ページ)を参照してください。

注 このオプションは、DS2 プロシジャまたは DS2 CAS アクションでも設定できます。

TRACEVARIABLES

DS2 プログラム実行中の変数値の変更のトレースを有効にします。

要件 DS2 App.TableServices.DS2.Runtime.TraceVariables ロガーも構成し、TRACE または DEBUG に設定する必要があります。変数の変更がトレースされるステートメントと関数のリストについては、“App.TableServices.DS2.Runtime.TraceVariables” (*SAS DS2 プログラマガイド*)を参照してください。

操作 このオプションは、DS2_OPTIONS ステートメントのすぐ下にあるプログラムブロックのトレースを有効にします。複数のプログラムブロックの変数トレースを有効にするには、トレースする各プログラムブロックの前に DS2_OPTIONS ステートメントを指定する必要があります。

例 “例: トレース変数のログ” (*SAS DS2 プログラマガイド*)

TREEALGORITHM=ITERATIVE | RECURSIVE

DS2 コンパイラが構文ツリーの構築に使用するデフォルトのアルゴリズムを変更します。RECURSIVE オプションを使用していて、スタックオーバーフローエラーが発生した場合、アルゴリズムを変更すると、スタックオーバーフローエラーが解決されることがあります。

デフォルト SAS プログラミングランタイム環境: RECURSIVE

CAS: ITERATIVE

注　このオプションは、DS2 プロシジャまたは DS2 CAS アクションでも設定できます。

TYPEWARN
暗黙的な型変換が発生すると、SAS ログに警告を出力します。

詳細

オプションの継承

次のログフラグメントは、保存された THREAD または PACKAGE プログラムブロックが別のプログラムブロックによってロードされ、ロードされたプログラムブロックが DS2_OPTIONS ステートメントでオプションを明示的に設定しない場合に発生する継承を示しています。ストアドパッケージ Compute が最初の DATA プログラムブロックによって 13 行目にロードされると、パッケージは DIVBYZERO オプションの IGNORE 設定を継承します。ストアドパッケージ Compute が 2 番目の DATA プログラムブロックによって 25 行目にロードされると、パッケージは DIVBYZERO オプションの ERROR 設定を継承します。

```
1    proc ds2;
2    package compute / overwrite=yes;
3     method div(double x, double y) returns double;
4      return x/y;
5     end;
6    endpackage;
7    run;
7   !  quit;
NOTE: Created package compute in data set work.compute.
NOTE: Execution succeeded. No rows affected.
...
8
9    proc ds2;
10    ds2_options divbyzero=ignore;
11    data _null_;
12     method init();
13      dcl package compute p();
14      dcl double d;
15      d = p.div(10, 0);      /* divide by zero ignored */
16      put d=;
17     end;
18    enddata;
19    run;
19   !  quit;
d=.
NOTE: Execution succeeded. No rows affected.
...
20
21    proc ds2;
22    ds2_options divbyzero=error;
23    data _null_;
24     method init();
25      dcl package compute p();
```

```
26      dcl double d;
27      d = p.div(10, 0);      /* divide by zero halts processing */
28      put d=;
29     end;
30    enddata;
31    run;
31   !  quit;
ERROR: Line 3: Float divide by zero
ERROR: General error
NOTE: PROC DS2 has set option NOEXEC and will continue to prepare statements.

NOTE: The SAS System stopped processing this step because of errors.
```

オプション範囲

DS2 プロシジャまたは CAS アクション、およびこのステートメントでオプションが指定されている場合、このステートメントの設定が優先されます。次の表は、このステートメントで DS2_OPTIONS オプションが DS2 プロシジャオプションまたは DS2 CAS アクションパラメーターとして 設定されている場合の動作を要約したものです。

表 10.1　オプション範囲の要約

DS2_OPTIONS オプション	指定されたプログラムブロックのプロシジャ設定をオーバーライドする [1]	DS2 プログラム全体のプロシジャ設定をオーバーライドする [2]
LOGICALEXPR=	*	
MODE=	*	
MSGLIMIT=		*
MSGORDER=		*
REPORTLINE=	*	
SAS	*	
SCOND=	*	
TREEALGORITHM=	*	

1 プログラムブロックがロードプログラムから 継承するオプション値も、プロシジャまたはアクションの設定よりも優先されます。プロシジャまたはアクションの値は、DS2 プログラムの残りのプログラムブロックに対して強制されます。

2 これらの値は、DS2 プログラムのすべてのプログラムブロックで 一貫している必要があります。プロシジャまたはアクションの値は、ルートプログラムブロックの DS2_OPTIONS ステートメントに明示的な設定がない場合にのみ適用されます。

例: DS2_OPTIONS オプションの指定

DS2_OPTIONS オプションを指定するコードフラグメントは次のとおりです。

```
ds2_options scond=error;
ds2_options divbyzero=ignore;
ds2_options msgorder=temporal msglimit=2K;
ds2_options mode=ansi;
```

ENDDATA ステートメント

DATA ステートメントの終わりを示します。

カテゴリ: ブロック

別名: ENDTABLE

構文

ENDDATA;

詳細

DS2 プログラムには、複数のパッケージサブプログラムとそれに続くオプションのデータプログラムを含めることができます。次の制限が適用されます。

- データプログラムは 1 つしか存在できず、データプログラムは最後のサブプログラムである必要があります。
- ENDPACKAGE、ENDTHREAD、または ENDDATA ステートメントは、DS2 プログラムの最後のサブプログラムではオプションです。これらのステートメントは、他のすべてのサブプログラムでは必須です。

関連項目

ステートメント:

- “DATA ステートメント” (1310 ページ)

ENDPACKAGE ステートメント

PACKAGE ステートメントの終わりを示します。

カテゴリ: ブロック

構文

ENDPACKAGE;

詳細

DS2 プログラムには、複数のサブプログラムとそれに続くオプションのデータプログラムを含めることができます。次の制限が適用されます。

- データプログラムは 1 つしか存在できず、データプログラムは最後のサブプログラムである必要があります。
- ENDPACKAGE、ENDTHREAD、または ENDDATA ステートメントは、DS2 プログラムの最後のサブプログラムではオプションです。これらのステートメントは、他のすべてのサブプログラムでは 必須です。

関連項目

ステートメント:

- “PACKAGE ステートメント” (1392 ページ)

ENDTHREAD ステートメント

THREAD ステートメントの終わりを示します。

カテゴリ: ブロック

構文

ENDTHREAD;

詳細

DS2 プログラムには、複数のスレッドサブプログラムとそれに続くオプションのデータプログラムを含めることができます。次の制限が適用されます。

- データプログラムは 1 つしか存在できず、データプログラムは最後のサブプログラムである必要があります。
- ENDPACKAGE、ENDTHREAD、または ENDDATA ステートメントは、DS2 プログラムの最後のサブプログラムではオプションです。これらのステートメントは、他のすべてのサブプログラムでは必須です。

関連項目

ステートメント:

- “THREAD ステートメント” (1429 ページ)

FORWARD ステートメント

メソッド定義がメソッド式に続いていることを示します。

カテゴリ: グローバル

構文

FORWARD *method* [...*method*];

必須引数

method
定義するメソッドの名前を指定します。

詳細

メソッド定義がそれを参照するメソッド式の後に記述される場合、メソッドの FORWARD ステートメントをメソッド式の前に宣言する必要があります。そうしないと、DS2 コンパイラは、メソッド式がメソッドを参照しているかどうかを判別できません。

例

- この例では、D メソッドは RUN メソッド内で呼び出されます。D メソッドは呼び出された後に定義されるため、D メソッドを呼び出す前に FORWARD ステートメントを指定する必要があります。

```
forward d;
method run();
  d = d();
  d = d(100);
end;
method d() returns double;
  return 99;
end;
method d(int y) returns int;
  return 100 + y;
end;
```

- この例では、システム関数 SIN をマスクするユーザー定義メソッド SIN を作成します。ユーザーメソッドは、システム関数 SIN を呼び出します。

```
forward sin;
method run()
  d = sin(3.14159/2);
  put 'd= ' d;
end;
method sin(double x) returns double;
  return system.sin(x);
end;
```

GOTO ステートメント

実行をラベル付きステートメントにすぐに送ります。

カテゴリ: ローカル

構文

GOTO *label*;

必須引数

label
GOTO 移動先を識別するステートメントラベルを指定します。

詳細

GOTO ステートメントの移動先ラベルは、同じ DS2 メソッド内にある必要があります。*label* 引数の指定が必要で、指定しないとエラーが発生します。ステートメントラベルは、ラベルステートメントを使用して定義します。

比較

GOTO ステートメントは、多くの場合、DO-END や IF-THEN/ELSE のプログラミングロジックで置き換えることができます。

例

この例では、x=2 のとき、プログラムの実行は DONE ラベルに移行します。

```
method run();
  x = 1;
  do;
    if x=2 then goto done;
    put x;
    x+1;
  end;
  done:
  put x;
end;
```

関連項目

ステートメント:

- “DO ステートメント” (1337 ページ)
- “Labels ステートメント” (1366 ページ)
- “LEAVE ステートメント” (1367 ページ)

IF ステートメント: サブセット化

条件を満たす行のみの処理を続行します。

カテゴリ: ローカル

構文

IF *expression*;

必須引数

expression
真または偽と評価される有効な式です。

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

詳細

サブセット化 IF ステートメントの式が評価され、ゼロ以外の値またはゼロのいずれかの結果が生成されます。ゼロ以外の値を指定すると、式は真になり、結果がゼロの場合、式は偽になります。

サブセット化 IF ステートメントは、次の IF-THEN ステートメントと同等です。

```
if not (expression)
  then return;
```

expression が真の場合、DS2 はプログラム内のステートメントを実行し続けます。

注: 論理演算では、サブセット化 IF ステートメントなどの式では、欠損値は SAS モードで偽と評価されます。null 値は、ANSI モードでは真にも偽にも評価されません。

注: 入力テーブルから保持されていない変数は、RUN メソッドの SET または MERGE ステートメントの前に使用された場合、欠損または null にリセットされます。

比較

IF-THEN/ELSE ステートメントを使用して、真と偽の両方の条件が存在する場合、または値が生成される前にさらに処理が必要な場合にステートメントを処理します。

関連項目

ステートメント:

- "IF-THEN/ELSE ステートメント" (1362 ページ)

IF-THEN/ELSE ステートメント

特定の条件を満たす行に対してステートメントを実行します。

カテゴリ: ローカル

構文

IF *expression* **THEN** *statement*;
[ELSE *statement* ;]

必須引数

expression
真または偽と評価され、必須引数である有効な式です。

参照項目: "DS2 式" (*SAS DS2 プログラマガイド*)

statement
任意の実行可能ステートメントまたは DO グループにすることができます。

詳細

IF-THEN ステートメントの式が評価され、ゼロ以外の値またはゼロのいずれかの結果が生成されます。ゼロ以外の値を指定すると、式は真になり、結果がゼロの場合、式は偽になります。

注: IF-THEN/ELSE ステートメントを含む論理演算では、SAS 欠損値と null 値はゼロ(または偽)と評価されます。IF-THEN-ELSE ステートメントで null 値をチェックするには、この例に示すように NULL 関数を使用する必要があります。

```
method init ();
  x=null;
  if (null(x)) then
    put 'null';
  else
    put 'not null';
end;
```

IF 句で指定された条件が満たされると、IF-THEN ステートメントがステートメントを実行します。オプションの ELSE ステートメントは、THEN 句が実行されない場合に代替アクションを提供します。ELSE ステートメントを使用する場合は、IF-THEN ステートメントの直後に続ける必要があります。

ELSE ステートメント *なし*で IF-THEN ステートメントを使用すると、すべての IF-THEN ステートメントが評価されます。IF 句が真の場合、THEN の後のステートメントが実行されます。それ以外の場合、ELSE の後のステートメントが実行されます。

注: 効率を高めるには、確率を下げるように IF-THEN/ELSE ステートメントを作成します。

注: IF 式を使用すると、条件式の評価が真か偽かに基づいて 2 つの値から選択することができます。さらに、IF 式をネストして、多方向の決定のために多くの値から選択することができます。詳細については、"IF 式" (2101 ページ)を参照してください。

比較

- 相互に排他的な条件が長く連続する場合は、IF-THEN ステートメントを連続で使用するかわりに SELECT 式を使用します。ネストされた IF-THEN/ELSE ステートメントが多数あると、内部エラーが発生する可能性があります。対照的に、SELECT 式は 1 回だけ評価されるため、パフォーマンスが向上する可能性があります。
- THEN 句を指定せずにサブセット化 IF ステートメントを使用して、処理が不要で出力が生成されないことを条件が示している場合に、ゼロ以外の値と評価される式のみの処理を続行します。

例

これらの例は、IF-THEN/ELSE ステートメントを示しています。

- ```
  if a = b then
    d = e;
  else
    d = f;
  ```
- ```
  if status='OK' and type=3 then count+1;
  ```
- ```
  if x=0 then
    if y ne 0 then put 'X ZERO, Y NONZERO';
    else put 'X ZERO, Y ZERO';
  else put 'X NONZERO';
  ```
- ```
  if answer=9 then
    do;
      answer=.;
      put 'INVALID ANSWER FOR ' id=;
    end;
  else
    do;
      answer=answer10;
      valid+1;
  ```

```
    end;
```

- ```
  xx = if
    if x then y
    else z then b
  else c;
  ```

関連項目

- “IF 式” (2101 ページ)

ステートメント:

- “IF ステートメント: サブセット化” (1360 ページ)

KEEP ステートメント

出力テーブルに列を含めます。

カテゴリ: グローバル

注: このステートメントは、メソッド内では使用できません。

構文

KEEP *column-list* | *vararray*;

必須引数

column-list
出力テーブルに書き込む 1 つ以上の列の名前を指定します。

制限事項 *col1–coln* 形式の番号付き範囲リストはサポートされていません。

vararray
変数配列の名前を指定します。

参照項目: “VARARRAY ステートメント” (1434 ページ)

詳細

KEEP ステートメントは、列リストのすべての列が出力行の作成に含まれるように指定します。KEEP ステートメントが指定されている場合、KEEP ステートメントに含まれていないすべての列が出力行から削除されます。DROP または KEEP ステートメントが記述されない場合、DS2 プログラムで作成されるすべてのテーブルにすべ
  ```

ての列が含まれます。同じ DS2 プログラム内で DROP ステートメントと KEEP ステートメントの両方を使用しないでください。

比較

- KEEP ステートメントは、DATA ステートメントで指定されているすべての出力テーブルに適用されます。異なるテーブルに異なる列を書き込むには、KEEP=テーブルオプションを使用する必要があります。
- DROP ステートメントは、出力テーブルから省略する列を指定する並列ステートメントです。
- KEEP および DROP ステートメントは、出力テーブルに含める列または出力テーブルから除外する列を選択します。サブセット化 IF ステートメントは行を選択します。
- KEEP ステートメントと RETAIN ステートメントを混同しないでください。RETAIN ステートメントは、DS2 RUN メソッドの 1 回の反復から次の反復まで、行値を列に保持します。KEEP ステートメントは行値には影響しませんが、出力テーブルに含める列のみを指定します。

例

例 1

この例では、KEEP ステートメントを使用して、出力テーブルに NAME 列と AVG 列のみを含めます。

```
data;
  keep name avg;
  method run();
    set table-specification;
  end;
enddata;
```

例 2

この例では、変数配列 x の最初の 3 つの変数が出力に保持されます。

```
/* keep x1, x2, x3 */
proc ds2;
data mytest;
  vararray double x[10];
  keep x1-x3;
  method init();
    do i = 1 to 10;
      x[i] = i;
```

```
    end;
    output;
  end;
enddata;
run;
quit;
```

関連項目

ステートメント:

- “DROP ステートメント” (1344 ページ)

テーブルオプション:

- “KEEP= テーブルオプション” (1502 ページ)

Labels ステートメント

別のステートメントによって参照されているステートメントを識別します。

カテゴリ: ローカル

構文

label: *statement*; [... *statement*];

必須引数

label
任意の識別子を指定し、その後にコロン(:)が続きます。label 引数を指定する必要があります。

範囲 1–255 文字

制限事項 ラベルに非ラテン文字が含まれている場合は、二重引用符で囲む必要があります。

statement
null ステートメント(;)を含む任意の実行可能なステートメントを指定します。statement 引数を指定する必要があります。

詳細

ラベルは、特定のステートメントに識別子を関連付け、GOTO や LEAVE などの他のステートメントからそのステートメントを参照できるようにします。ステートメントには複数のラベルを設定できます。

比較

DECLARE ステートメントの HAVING 句の LABEL 属性は、説明ラベルを列に割り当てます。ステートメントラベルは、GOTO や LEAVE などの別のステートメントによって同じ DS2 プログラムで参照されるステートメントまたはステートメントのグループを識別します。

例

- ```
  restock:
    if x > 1 then
      y = 3;
    else
      y = 5;
  ```
- ```
  label1:
    label2:
      do i = 1 to 3;
        j = j * i;
      end;
  ```

関連項目

ステートメント:

- “GOTO ステートメント” (1359 ページ)
- “LEAVE ステートメント” (1367 ページ)

LEAVE ステートメント

現在の DO ループの処理を停止し、実行を現在の DO ステートメントに続くステートメント、または現在の DO ステートメントを囲むラベル付き DO ステートメントのいずれかに転送します。

カテゴリ: ローカル

構文

LEAVE [*identifier*] ;

引数なし

LEAVE ステートメントは、現在の DO ステートメントの処理を停止し、現在の DO ステートメントに続く次のステートメントから処理を再開します。

オプション引数

identifier
ターゲット DO ステートメントに関連付けられたラベル。

詳細

LEAVE ステートメントを使用すると、DO ループを途中で終了させることができます。LEAVE ステートメントは、単独で使用することも、条件に基づいて(たとえば、IF ステートメントと組み合わせて)使用することもできます。LEAVE ステートメントの後に識別子が続かない場合、ターゲットは LEAVE ステートメントを囲んでいる DO ステートメントです。LEAVE ステートメントの後に識別子が続く場合、ターゲットは識別子で指定されたラベルを持つ DO ステートメントです。ターゲット DO ステートメントは、現在の DO ステートメントを囲む必要があります。識別子が DO ステートメント以外のステートメント、または現在の DO ステートメントを含まない DO ステートメントを指定している場合、エラーが発生します。指定されたラベルが存在しない場合にもエラーが発生します。

比較

- LEAVE ステートメントは、現在の DO ステートメントの処理を停止し、現在の DO ステートメントの外部でプログラムの実行を再開します。
- CONTINUE ステートメントは、DO ステートメントの現在の反復の処理を停止し、現在の DO ステートメントの次の反復でプログラムの実行を再開します。

例

例 1: 識別子のない LEAVE ステートメント

この例は、識別子のない LEAVE ステートメントを示しています。

```
data _null_;
  dcl int i;
  method init();
    do i = 1 to 10;
      if i > 4 then leave;
```

```
      put i;
    end;
  end;
enddata;
```

次の行が SAS ログに書き込まれます。

```
1
2
3
4
```

例 2: 識別子のある LEAVE ステートメント

この例は、識別子のある LEAVE ステートメントを示しています。

```
data _null_;
  dcl int sum i j k ;
  method init();
    lab1: do i = 1 to 5;
      sum + 1;
    lab2: do j = 1 to 3;
      sum + 1;
      do k = 1 to 3;
        put sum i j k;
        sum + 1;
        if j = 2 then
          leave lab2;
        if i = 2 then
          leave lab1;
        if k = 2 then
          leave ;
        end;
      end;
      end;
  end;
enddata;
```

次の行が SAS ログに書き込まれます。

```
2 1 1 1
3 1 1 2
5 1 2 1
8 2 1 1
```

関連項目

ステートメント:

- “CONTINUE ステートメント” (1309 ページ)
- “DO ステートメント” (1337 ページ)

MERGE ステートメント

2 つ以上のテーブルの行を 1 つの行に結合します。

カテゴリ: ローカル

制限事項: SPD Engine または HDMD 形式のテーブルは、MERGE ステートメントをサポートしていません。

要件 MERGE ステートメントの後に BY ステートメントを配置する必要があります。

注: MERGE ステートメントを使用して読み取られる変数は、欠損値または null 値のいずれかに設定されます。

ヒント: 結果の列の幅は、単一の SET ステートメント内のすべてのテーブルにわたる 最大幅によって決定されます。末尾の空白は、MERGE ステートメントとは無関係です。

構文

MERGE <table-reference> <table-reference> [... <table-reference>] [/RETAIN];

<table-reference>::=

{ *table* [(*table-options*)] }

必須引数

table
テーブルの名前を指定します。*table* は、次のいずれかの形式をとります。

- *catalog.schema.table-name*
- *schema.table-name*
- *catalog.table-name*
- *table-name*
- *caslib.table-name*

catalog SQL カタログの ANSI SQL 規格が実装された、論理的に関連するスキーマをグループ化したデータコンテナーオブジェクトです。カタログは、 データ編成階層の第 1 レベル(最上位)のグループ化メカニズムです。 スキーマとともに使用され、名前を修飾する方法を提供します。

schema SQL スキーマの ANSI SQL 規格が実装された、テーブルやビューなどのファイルや、ストアドプロシジャなどのデータソースで サポートされるその他のオブジェクトをグループ化するデータコンテナーオブジェクトです。スキーマは、catalog とともに使用されて、名前を修飾する方法を提供する グループ化オブジェクトを提供します。

table-name テーブルの名前です。

caslib.table-name CAS サーバーにあるテーブルの名前です。

注 テーブル名にドットが含まれていて、CAS テーブルにアクセスしている場合は、テーブル名を二重引用符で囲む必要があります。次に例を示します。

```
data mycaslib "tdlibref.foo";
```

テーブル名とスキーマ名を引用符で囲まない場合、DS2 はテーブル名とスキーマ名を大文字で保存し、二重引用符を加えます。引用符で囲まれたテーブル名とスキーマ名は、そのまま使用されます。つまり、引用符で囲まれた元の大文字小文字が引用符で囲まれたままになります。たとえば、data mytable;では、テーブル名は"MYTABLE"として保存され、data "MyTable";では、テーブル名は"MyTable"として保存されます。これは、データソースのテーブル名とスキーマ名で大文字と小文字が区別される場合に重要です。

注意 **LIBNAME ステートメントで PRESERVE_TAB_NAMES=no オプションを使用すると、予期しない結果が生じる可能性があります。**

オプション引数

(*table-options*)
DS2 プログラムで出力テーブルに行を書き込むときに適用するオプションの引数を指定します。

注 テーブルオプションの詳細については、“DS2 テーブルオプション” (1457 ページ)を参照してください。

/RETAIN
特定の BY グループのデータセットの最後の行が、寄与しているデータセットのいずれにも行がなくなるまで繰り返し使用されることを指定します。

ヒント この引数を使用して、DATA ステップマージのようなマージ動作を実現します。

詳細

MERGE ステートメントは柔軟性があり、DS2 プログラミングでさまざまな用途があります。このセクションでは、MERGE の基本的な使用法について説明します。その他の用途としては、複数の BY 変数の使用、3 つ以上のテーブルのマージ、およびいくつかの行と別のテーブルのすべての 行のマージなどがあります。

マッチマージは、共通変数の値に従って、2 つ以上のテーブルの行を新しいテーブルの 1 つの行に結合します。新しいテーブルの行数は、すべてのテーブルの 各 BY グループの最大行数の合計です。マッチマージを実行するには、MERGE ステートメントの直後に BY ステートメントを使用します。BY ステートメントの変数は、すべてのテーブルに共通である必要があります。データプログラムの各 MERGE ステートメントに付随できる BY ステートメントは 1 つだけです。

詳細については、“マッチマージ” (*SAS DS2 プログラマガイド*)を参照してください。

注意
DECIMAL または NUMERIC データ型を持つ DS2 マージの BY 変数は、DOUBLE データ型に変換されます。 一致する DECIMAL 列が BY 変数でない場合、DECIMAL 列は DECIMAL データ型のままになります。

注意
テーブル間で DECIMAL または NUMERIC データ型の列間に型、スケール、または精度の不一致がある場合、その列は DOUBLE データ型に変換されます。

注: MERGE ステートメント内のデータセットの順序は、マッチングに影響を与える可能性があります。

DATA ステップと DS2 のマージ動作の違いに注意してください。MERGE ステートメントは、多対多のマッチマージでデカルト積を生成しません。かわりに、少なくとも 1 つのテーブルに BY グループの行があるときに、疎な 1 対 1 マージを実行します。DATA ステップのマージと比較すると、DS2 MERGE ステートメントの結果はデカルト積のサブセットです。対照的に、DATA ステップマージの結果は、PDV を埋めるために遅延保持戦略を使用するため、サブセットではありません。

これがコードです。

```
merge T1 (in=inT1) T2 (in=inT2); by K;
```

次の例は、共通変数 K によるテーブル T1 と T2 のマージ中の DATA ステップ保持戦略を示しています。テーブル T1 の 2 行目のマッチマージが発生すると、列 C の値は BY 変数の前回の一致から保持されます。変数 inT2 は **1** に設定され、テーブル T2 が最終的なテーブル結果に寄与したことを示します。

図 10.1 DATA ステップマージ

T1

K	A	B
101	Cat	Apple
101	Pig	Banana
102	Duck	Fig

T2

K	A	C
101	*Cow*	*Red*
102	.	*Yellow*
103	.	*Blue*

K	inT1	inT2	A	B	C
101	1	1	*Cow*	Apple	*Red*
101	1	1	Pig	Banana	*Red*
102	1	1	.	Fig	*Yellow*
103	0	1	.	.	*Blue*

DATA ステップマージとは対照的に、DS2 マージは BY グループ処理間の PDV をクリアします。テーブル T1 の 2 行目にはテーブル T2 に対応する一致がないため、列 C は空のままで、変数 inT2 は **0** に設定され、テーブル T2 が最終的なテーブル結果に寄与しなかったことを示します。

図 10.2 DS2 マージ

T1

K	A	B
101	Cat	Apple
101	Pig	Banana
102	Duck	Fig

T2

K	A	C
101	*Cow*	*Red*
102	.	*Yellow*
103	.	*Blue*

K	inT1	inT2	A	B	C
101	1	1	*Cow*	Apple	*Red*
101	1	0	Pig	Banana	.
102	1	1	.	Fig	*Yellow*
103	0	1	.	.	*Blue*

MERGE ステートメントで RETAIN 引数を使用して、DATA ステップマージと同様の多対多のマッチマージを生成できます。前述と同じ DS2 プログラムが DATA ステッププログラムと同様の結果を生成することを確認するには、“例 2: 保持された行を持つテーブルのマージ” (1375 ページ)を参照してください。

テーブルの行間に 1 対 1 の行マッピングが含まれていないテーブルをマッチマージすると、予期しない結果が生じる可能性があります。特定の BY グループ内では、BY グループ内のオブザベーションは、マッチマージ中にデータセットに表示される順序で必ずしも選択されるとは限りません。

注: MongoDB データソースには、テーブルを作成するための特別な要件があります。データに応じて、ルートテーブル、親テーブル、子テーブルを作成することをお勧めします。DS2 ではルートテーブルのみを作成できます。親テーブルと子テーブルを作成するには、FedSQL を使用する必要があります。*SAS/ACCESS for Relational Databases: リファレンス*の MongoDB に関する情報を参照してください。

比較

SET ステートメントを指定すると、行数が等しくない場合、SAS はすべてのテーブルからすべての行が読み取られる前に処理を停止します。対照的に、SAS は、MERGE ステートメントで指定されたすべてのテーブルのすべての 行を処理し続けます。

例

例 1: 1 つの列を使用した 2 つのテーブルのマージ

次の例では、2 つのテーブルを作成し、MERGE ステートメントで、**common** 列を使用してテーブルを結合します。

```
data mrg01a(overwrite=yes);
  dcl varchar(10) common animal;
```

```
  method init();
    common='a'; animal='Ant'; output;
    common='b'; animal='Bird'; output;
    common='c'; animal='Cat'; output;
    common='d'; animal='Dog';  output;
    common='e'; animal='Eagle'; output;
    common='f'; animal='Frog';  output;
  end;
enddata;
run;


data mrg01b(overwrite=yes);
  dcl varchar(10) common plant;
  method init();
    common='a'; plant='Apple'; output;
    common='b'; plant='Banana'; output;
    common='c'; plant='Coconut'; output;
    common='d'; plant='Dewberry';  output;
    common='e'; plant='Eggplant'; output;
    common='f'; plant='Fig';  output;
    common='g'; plant='Grapefruit'; output;
  end;
enddata;
run;


/*  match merge */
data;
  method run();
    merge mrg01a mrg01b; by common;
  end;
 enddata;
run;
```

次のテーブルが生成されます。

common	animal	plant
a	Ant	Apple
b	Bird	Banana
c	Cat	Coconut
d	Dog	Dewberry
e	Eagle	Eggplant
f	Frog	Fig
g		Grapefruit

例 2: 保持された行を持つテーブルのマージ

次の例では、詳細セクションに示されているデータセットを再作成し、RETAIN 引数を使用してそれらをマージして、DATA ステップで生成されたデカルト積と同様の結果を生成します。

```
/********** create data sets ***********/
proc ds2;
data t1 (overwrite=yes);
  dcl double k;
  dcl char a b;
  method init();
    k=101; a='Cat'; b='Apple'; output;
    k=101; a='Pig'; b='Banana'; output;
    k=102; a='Duck'; b='Fig'; output;
  end;
enddata;
run;
quit;

proc ds2;
data t2 (overwrite=yes);
  dcl double k;
  dcl char a c;
  method init();
    k=101; a='Cow'; c='Red'; output;
    k=102; a='' ; c='Yellow'; output;
    k=103; a='' ; c='Blue';  output;
  end;
enddata;
run;
quit;

/*************Merge with Retain argument **********/
proc ds2;
data ds2merge (overwrite=yes);
  dcl double k;
  dcl char a b c;
  method run();
    merge t1 t2 /retain;
    by k;
  end;
enddata;
run;
quit;

proc print data=ds2merge;
run;
quit;
```

次のテーブルが生成されます。これは、“詳細” (1371 ページ)に示されている DATA ステップマージコードによって生成されたテーブルと同じです。

Obs	k	a	b	c
1	101	Cow	Apple	Red
2	101	Cow	Banana	Red
3	102		Fig	Yellow
4	103			Blue

関連項目

- *“マッチマージ” (SAS DS2 プログラマガイド)*

ステートメント:

- “BY ステートメント” (1304 ページ)
- “SET ステートメント” (1416 ページ)

METHOD ステートメント

複数回呼び出して実行できるコードのブロックを定義します。

カテゴリ: ブロック

構文

[PRIVATE] **METHOD** *method* ([[IN_OUT] <parameter> [, ... [IN_OUT]< parameter>]])

[RETURNS *data-type*];
...*method-body* ...
END;

<parameter>::=

<data-type> *variable*

< data-type >::=

<exact-numeric-type> | <approximate-numeric-type> | <binary-string-type>
| <string-type>
| <date-type>

<exact-numeric-type> ::=

{**INT** | **BIGINT** | **SMALLINT** | **TINYINT**
| **DECIMAL** [(*precision* [,*scale*])] | **NUMERIC** [(*precision* [,*scale*])] }

<approximate-numeric-type> ::=

{ **DOUBLE** | **DOUBLE PRECISION** | **FLOAT** | **REAL** }

<binary-string-type>::=

BINARY(*length*) | **VARBINARY**(*length*)

<string-type>::=

NCHAR [(*character-length*)]

| **NVARCHAR** [(*character-length*)]

| **CHAR** [(*character-length*)] [CHARACTER SET *character-set-identifier*]

| **VARCHAR** [(*character-length*)] [CHARACTER SET *character-set-identifier*]

<date-type>::=

{ **TIME** | **TIMESTAMP** } [(*precision*)] | **DATE**

必須引数

method
メソッドの名前を指定します。メソッド名にはグローバルスコープがあります。

method-body
メソッドが呼び出されたときに実行される変数宣言と実行可能 DS2 コードで構成されます。メソッド本体で宣言されているすべての変数は、メソッドに対してローカルです。

END
メソッドの終わりを示します。

オプション引数

PRIVATE
パッケージ内からのみアクセスできるメソッドを指定します。

参照項目: “属性とメソッド” (*SAS DS2 プログラマガイド*)

IN_OUT
引数が値ではなく参照によって操作されることを指定します。IN_OUT パラメーターは、引数のコピーではなく、引数を操作します。

要件 IN_OUT パラメーターに渡される引数は、式ではなく、識別子などの変更可能な値である必要があります。

引数のデータ型は、IN_OUT パラメーターと同じデータ型である必要があります。

注 メソッド宣言で IN_OUT パラメーターの長さが指定されている場合、警告が発行され、長さは無視されます。指定された引数の長さが常に使用されます。

ヒント DS2 メソッドでは、数千もの引数をサポートできます。DS2 メソッドでサポートされる引数の数は、引数のデータ型と DS2 プログラムが実行されるシステムによって異なります。サポートされている引数の数を超える DS2 メソッドは、コンパイルエラーを生成します。

参照項目: “IN_OUT パラメーターを使用する場合の注意点” (1382 ページ)

“メソッドに文字値を渡す” (1381 ページ)

parameter
メソッドに渡されるパラメーターを指定します。型は、任意の有効な文字、数値、または日付型、あるいはパッケージにすることができます。パラメーターには、メソッドに対してローカルなスコープがあり、デフォルトでは、パラメーターは値によって渡されます。パラメーターは、IN_OUT パラメーターで指定されているか、パッケージまたは配列である場合を除き、パラメーターに指定されている引数の値のコピーになるように初期化されます。IN_OUT キーワードが指定されていない場合でも、パッケージ型変数と配列は常に参照によって渡されます。

操作 IN_OUT パラメーターまたはパッケージパラメーターが指定されている場合、メソッドが戻るときにパラメーターに渡される引数の値が変更されている可能性があります。

ヒント DS2 メソッドでは、数千もの引数をサポートできます。DS2 メソッドでサポートされる引数の数は、引数のデータ型と DS2 プログラムが実行されるシステムによって異なります。サポートされている引数の数を超える DS2 メソッドは、コンパイルエラーを生成します。

INT | BIGINT | SMALLINT | TINYINT
整数変数または配列を指定します。

別名 INT の場合は INTEGER

参照項目: “DS2 データ型” (*SAS DS2 プログラマガイド*)

DECIMAL[(*precision* [, *scale*])] | NUMERIC[(*precision* [, *scale*])]
正確な数値変数または配列を指定します。

precision
小数点の左側と右側の両方に保存できる最大合計桁を指定します

注 すべてのデータソースが 52 桁の精度をサポートできるわけではありません。

scale
小数点の右側に保存できる最大桁数を指定します

範囲 0–*precision*

注 *scale* は *precision* は以下です。

参照項目: “DS2 データ型” (*SAS DS2 プログラマガイド*)

DOUBLE | DOUBLE PRECISION | FLOAT | REAL
浮動小数点変数または配列を指定します。

参照項目: “DS2 データ型” (*SAS DS2 プログラマガイド*)

BINARY (*length*)
バイナリ変数または配列を指定します。

要件 BINARY を指定する場合は、変数または配列の *length* もバイト単位で指定する必要があります。

参照項目: “DS2 データ型” (*SAS DS2 プログラマガイド*)

VARBINARY (*length*)
可変長のバイナリ変数または配列を指定します。

別名 BINARY VARYING

要件 VARBINARY を指定する場合は、バイナリ変数または配列の *length* もバイト単位で指定する必要があります。

参照項目: “DS2 データ型” (*SAS DS2 プログラマガイド*)

NCHAR | NVARCHAR | CHAR | VARCHAR
文字変数または配列を指定します。

別名 NCHAR の場合は NATIONAL CHARACTER、NATIONAL CHAR

NVARCHAR の場合は NATIONAL CHARACTER VARYING、NATIONAL CHAR VARYING

CHAR の場合は CHARACTER

VARCHAR の場合は CHARACTER VARYING

参照項目: “DS2 データ型” (*SAS DS2 プログラマガイド*)

character-length
文字列が NCHAR、NVARCHAR、CHAR、VARCHAR データ型に対して保持できる最大文字数を指定します。

デフォルト 8

ヒント CHAR を使用して宣言された文字変数がストレージに使用するバイト数は、セッションエンコーディングによって異なります。NCHAR バリアントのいずれかを使用して宣言されたものは、ストレージが広く、シングルバイト文字ストレージでは不十分な文字セット(Unicode など)を表すために使用できます。セッションエンコーディングで文字ごとに複数のバイトが必要な場合(UTF-8 など)、CHAR と NCHAR は同じ型であり、どちらも NCHAR を使用します。

CHARACTER SET *character-set-identifier*
CHAR データ型と VARCHAR データ型の文字セットエンコーディング情報を指定します。

デフォルト デフォルトのエンコーディングは、オペレーティングシステムとロケールによって異なります。

ヒント　文字セット名には、文字列リテラルまたは単純な文字列を使用できます。たとえば、"ibm-866"また'ibm-866'を指定できます。

参照項目:　文字セットエンコーディング値の完全なリストについては、*SAS 各国語サポート(NLS): リファレンスガイド*の“SAS 言語要素中のエンコーディング値”を参照してください。

TIME
時間変数または配列を指定します。

TIMESTAMP
日付と時間の両方の変数または配列を指定します。

precision
TIME データ型または TIMESTAMP データ型の精度を指定します。

デフォルト　6

注　SAS 以外のデータソースで TIME 値と TIMESTAMP 値を操作していて、精度を指定しない場合、デフォルトの精度は常に DS2 のデフォルトの精度である 6 になります。

DATE
日付変数または配列を指定します。

variable
スカラー変数または配列名を指定します。1 つ以上の変数または配列を指定できます。ただし、*variable* は *data-type* で指定された型のみにすることができます。同じ型のスカラー変数と配列変数を混在させることができます。

RETURNS *data-type*
メソッドが返す値のデータ型を指定します。型は、任意の有効な文字、数値、または日付型にすることができます。

詳細

メソッドの基本

DS2 には、*システム*メソッドと*ユーザー定義*メソッドの 2 種類のメソッドがあります。METHOD ステートメントを使用すると、独自のユーザー定義メソッドを作成できます。システムメソッドについては、“DS2 システムメソッド” (1443 ページ)を参照してください。

メソッドが値を返す場合、メソッドに記述される各 RETURN ステートメントには、関連付けられた戻り式が必要です。詳細については、“RETURN ステートメント” (1411 ページ)を参照してください。

メソッド定義がそれを参照するメソッド式の後に記述される場合、メソッドの FORWARD ステートメントは、メソッド式の前にメソッドに対して記述される必要があります。DS2 INIT、RUN、または TERM メソッドの前に定義されていないユーザー定義メソッドは、FORWARD ステートメントで宣言する必要があります。詳細については、“FORWARD ステートメント” (1358 ページ)を参照してください。

注: TINYINT および SMALLINT メソッドパラメーターは自動的に INTEGER にプロモートされ、REAL メソッドパラメーターは自動的に DOUBLE にプロモートされます。警告メッセージが表示されます。詳細については、"式オペランドの DS2 型変換" (*SAS DS2 プログラマガイド*)を参照してください。

メソッドに文字値を渡す

DS2 のデフォルトのパラメーター渡しセマンティクスは、値渡しです。これは、パラメーターのコピーが作成され、元のパラメーターではなく、そのコピーがメソッドに渡されることを意味します。メソッドによるパラメーターの変更は、元のパラメーターではなく、コピーに対して行われます。これにより、メソッドの呼び出し元に対して元のパラメーターの値が維持されます。

文字変数の潜在的なサイズを考えると、メモリと時間の両方の観点から、コピーを作成するコストは非常に大きくなる可能性があります。このコストを回避するには、IN_OUT パラメーターの使用を検討してください。これにより、パラメーターが参照渡しされます。これは、パラメーターの場所またはアドレスがメソッドに渡され、パラメーターの 2 番目のコピーが作成されないことを意味します。メソッドによってパラメーターの値に意図的または非意図的に加えられた変更は、メソッドが戻った後、呼び出し元によるパラメーターへの参照に反映されます。

この構文は、2 種類のパラメーター渡しを示しています。

```
/* pass-by-value */
method copy_made(char(256) x);
  ...
end;

/* pass-by-reference */
method no_copy(in_out char x);
  ...
end;
```

メソッドから文字値を返す

メソッドによって文字列値が返されると、DS2 は戻り値を保持するための一時的な文字列を作成します。戻り値を一時的な文字列からターゲットの場所にコピーするにはコストがかかります。コピーのコストを回避するために、IN_OUT パラメーターで戻り値のターゲットの場所を指定します。

```
method copy_made() returns char(1024);
  return 'very long string';
end;

method no_copy(in_out char return_val);
  return_val = 'very long string';
end;
```

IN_OUT パラメーターでターゲットの場所を渡す場合の利点の 1 つは、メソッドがコンパイル時に返される文字列値のサイズを指定する必要がないことです。

IN_OUT パラメーターを使用する場合の注意点

ほとんどの DS2 演算子では、サブ式が評価される順序は未定義です。したがって、式で変数を複数回使用すると、式がその変数を IN_OUT パラメーターとしてメソッド呼び出しに渡すことで変更したときに、予期しない結果が生じる可能性があります。次に例を示します。

```
method addValue( in_out integer i) returns integer;
  i = i + 100;
  return i;
end;

method init();
  X = 0;
  Y = addValue(X) + X;
end;
```

式 addValue(X) + X で X を評価する場合、評価の順序は定義されていません。右端の X 値をキャプチャする前に addValue(X)メソッド呼び出しが評価された場合、結果は Y=200 になります。メソッド呼び出しの前に右端の X 値がキャプチャされた場合、結果は Y=100 になります。

この問題を回避する最も簡単な方法は、メソッド呼び出しが X を変更するサブ式を、X を含む他のサブ式とは別のステートメントに配置することです。例を次に示します。

- addValue(X)を最初に評価する必要がある場合は、前述のコードで Y=addValue(X) + X;を変更します。改訂された式は次のとおりです。

  ```
  Y = addValue(X);
  Y = Y + X;
  ```

- addValue(X)を最後に評価する必要がある場合は、Y=addValue(X) + X;を変更します。改訂された式は次のとおりです。

  ```
  Y = X;
  Y = addValue(X) + Y;
  ```

例

例 1: ユーザー定義のメソッド

これらの 3 つの例では、M、CONCAT、および ADD はユーザー定義のメソッドです。

- ```
  method m(int x, int y) returns int;
    return x + y;
  end;
  ```

- ```
  method concat(char(100) x, char(100) y) returns char(200);
    return trim(x) || y;
  end;
  ```

- ```
  method add(double x, double y);
    this.x = this.x + x;
    this.y = this.y + y;
  end;
  ```
  ```

例 2: オーバーロードされたメソッド

次の例では、D メソッドと CONCAT メソッドがオーバーロードされています。2 つのメソッド定義の名前が同じで、型シグネチャが異なる場合(つまり、メソッドの型が異なる場合)、そのメソッドはオーバーロードされます。

```
method d(double x, double y) returns double;
  dcl double temp;
  temp = x * 99;
  return x + y + temp;
end;
method d(double x, int y) returns double;
  return x + y
end;
method d(int x);
  put x;
end;
method concat(char(100) x, char(100) y) returns char(200);
  return "pre" || trim(x) || y;
end;
method concat(char(100) x, char(100) y, char(100) z) returns char(200);
  return trim(x) || trim(y) || z;
end;
```

例 3: IN_OUT 引数の使用

次の例では、メソッド *swapper* が引数値を交換します。IN_OUT 引数を使用すると、メソッドが呼び出された場所で値を変更できます。

```
package xyzzy;
  method swapper(in_out double a, in_out double b);
    declare double x;
    x=a; a=b; b=x;
  end;
endpackage;
run;

data _null_;
  method init();
    dcl package xyzzy x();
    a=10; b=42;
    put 'before: ' a= b=;
    x.swapper(a,b);
    put 'after:  ' a= b=;
  end;
enddata;
run;
```

次の行が SAS ログに書き込まれます。

```
before:  a=10 b=42
after:  a=42 b=10
```

例 4: メソッド引数としてパッケージを渡す

次の例では、*person_list* パッケージに *addP1* と *addP2* の 2 つのメソッドが含まれています。*addP1* メソッドと *addP2* メソッドはそれぞれ、入力パラメーターとして *person* パッケージを取ります。

```
proc ds2;
 package work.person / overwrite=yes;
  declare varchar(32) lastname;
  declare varchar(32) firstname;
  method setNames(varchar(32) lastname_p, varchar(32) firstname_p);
   lastname = lastname_p;
   firstname = firstname_p;
  end;
  method getFullname() returns varchar(66);
   return (catx(', ', lastname, firstname));
  end;
 endpackage;
 package work.person_list / overwrite=yes;
  declare package work.person pl1;
  declare package work.person pl2;
  method addP1( package work.person pers_p);
   pl1 = pers_p;
  end;
  method addP2( package work.person pers_p);
   pl2 = pers_p;
  end;
  method getPersonList() returns varchar(256);
   return ( catx(':', pl1.getFullname(), pl2.getFullname()) );
  end;
 endpackage;
run;

 data new (overwrite=yes);
  declare char(66) myname;
  declare char(256) allNames;
   method init();
   declare package work.person pers1 ();
   declare package work.person pers2 ();
   declare package work.person_list pl ();
   pers1.setNames('Miller', 'Brad');
   pers2.setNames('Smith', 'Colin');
   myname = pers1.getFullname();
   pl.addP1(pers1);
   pl.addP2(pers2);
   allNames = pl.getPersonList();
   put myname=;
   put allNames=;
  end;
 enddata;
run;
quit;
```

次の行が SAS ログに書き込まれます。

```
myname=Miller, Brad
allNames=Miller, Brad:Smith, Colin
```

関連項目

- “メソッド” (*SAS DS2 プログラマガイド*)

ステートメント:

- “FORWARD ステートメント” (1358 ページ)
- “RETURN ステートメント” (1411 ページ)

メソッド:

- “INIT メソッド” (1445 ページ)
- “REGISTEROUTPARAMETER メソッド” (1995 ページ)
- “RUN メソッド” (1446 ページ)
- “TERM メソッド” (1450 ページ)

Null ステートメント

空のステートメントを作成します。

カテゴリ: ローカル

構文

;

詳細

Null ステートメントは、セミコロンのみで構成されています。空のステートメントが作成されます。

例

この例は、Null ステートメントを使用してステートメントを実行しない方法を示しています。

```
method init();
  x = 1;
  y = 0;
  z = 1;
  if x & y | not z then
    ;
```

```
  else
    put 'else';
end;
```

OUTPUT ステートメント

現在の行をテーブルに書き込みます。

カテゴリ: ローカル

構文

OUTPUT [{ *table* [... *table*] } | _ROWSET_ | _NULL_];

引数なし

引数なしで OUTPUT を使用すると、現在の行は、DATA ステートメントまたはスレッドプログラムに指定されたすべてのテーブルに 書き込まれます。DATA ステートメントまたはスレッドプログラムにテーブルが指定されていない場合、行はクライアントアプリケーションに書き込まれます。

オプション引数

table
行を書きこむテーブルの名前を指定します。*table* は、次のいずれかの形式をとります。

- *catalog.schema.table-name*
- *schema.table-name*
- *catalog.table-name*
- *table-name*
- *caslib.table-name*

catalog SQL カタログの ANSI SQL 規格が実装された、論理的に関連するスキーマをグループ化したデータコンテナーオブジェクトです。カタログは、 データ編成階層の第 1 レベル(最上位)のグループ化メカニズムです。 スキーマとともに使用され、名前を修飾する方法を提供します。

schema SQL スキーマの ANSI SQL 規格が実装された、テーブルやビューなどのファイルや、ストアドプロシジャなどのデータソースで サポートされるその他のオブジェクトをグループ化するデータコンテナーオブジェクトです。スキーマは、catalog とともに使用されて、名前を修飾する方法を提供する グループ化オブジェクトを提供します。

table-name テーブルの名前です。

caslib.table-name CAS サーバーにあるテーブルの名前です。

制限事項 OUTPUT ステートメントで指定された名前はすべて DATA ステートメントにも記述されている必要があります。

注 テーブル名にドットが含まれていて、CAS テーブルにアクセスしている場合は、テーブル名を二重引用符で囲む必要があります。次に例を示します。

```
data mycaslib "tdlibref.foo";
```

テーブル名とスキーマ名を引用符で囲まない場合、DS2 はテーブル名とスキーマ名を大文字で保存し、二重引用符を加えます。引用符で囲まれたテーブル名とスキーマ名は、そのまま使用されます。つまり、引用符で囲まれた元の大文字小文字が引用符で囲まれたままになります。たとえば、data mytable;ではテーブル名は"MYTABLE"として保存され、data "MyTable";ではテーブル名は"MyTable"として保存されます。これは、データソースのテーブル名とスキーマ名で大文字と小文字が区別される場合に重要です。

ヒント OUTPUT ステートメントでは、その DS2 プログラムの DATA ステートメントで指定した数だけテーブルを指定できます。

注意 **LIBNAME ステートメントで PRESERVE_TAB_NAMES=no オプションを使用すると、予期しない結果が生じる可能性があります。**

ROWSET
OUTPUT ステートメントで行をテーブルに書き込まず、かわりにテーブル行をクライアントアプリケーションに返すように指定します。

NULL
OUTPUT ステートメントで、テーブルとクライアントアプリケーションのいずれにも行を書き込まないように指定します。

詳細

OUTPUT ステートメントが行を書き込むタイミングと場所

OUTPUT ステートメントは、OUTPUT ステートメントの実行時にグローバル変数に含まれる行の値を使用して、出力行を作成します。OUTPUT ステートメントは、DS2 プログラムの最後ではなく、すぐに現在の行をテーブルに書き込みます。OUTPUT ステートメントでテーブル名が指定されていない場合、行は DATA ステートメントにリストされている 1 つまたは複数のテーブルに書き込まれます。

DS2 は、値が既存のテーブルから読み取られたかプログラムで作成さたかに関係なく、コンパイラが DS2 プログラム内で検出した順序で値を追跡します。

注: MongoDB データソースには、テーブルを作成するための特別な要件があります。データに応じて、ルートテーブル、親テーブル、子テーブルを作成することをお勧めします。DS2 ではルートテーブルのみを作成できます。親テーブルと子テー

ブルを作成するには、FedSQL を使用する必要があります。*SAS/ACCESS for Relational Databases: リファレンス*の MongoDB に関する情報を参照してくださ。

暗黙的出力と明示的出力

OUTPUT ステートメントを指定しなかった場合、DS2 は RUN メソッドの最後に暗黙的に 1 つ追加し、作成中の 1 つまたは複数のテーブルに行を書き込みます。

DS2 プログラムに明示的な OUTPUT ステートメントを配置すると、自動出力がオーバーライドされ、明示的な OUTPUT ステートメントが実行された場合にのみテーブルに行が追加されます。ただし、OUTPUT ステートメントを使用して任意の 1 つのテーブルに行を書き込むと、RUN メソッドの最後に暗黙的な OUTPUT ステートメントは存在しなくなります。この状況では、DS2 プログラムは、明示的な OUTPUT ステートメントが実行された場合にのみテーブルに行を書き込みます。OUTPUT ステートメントは、単独で使用することも、IF-THEN/ELSE または SELECT ステートメントの一部として使用することも、DO ループ処理で使用することもできます。

DS2 プログラムスレッドでの OUTPUT ステートメントの使用

スレッドプログラムの OUTPUT ステートメントにテーブル名を含めることはできません。各出力行は、スレッドを開始したスレッドプログラムに返されます。

比較

- OUTPUT ステートメントは、行をテーブルまたはクライアントアプリケーションに書き込みます。PUT ステートメントは、変数値またはテキスト文字列を SAS ログに書き込みます。
- テーブルに行が書き込まれるタイミングを制御するには、OUTPUT ステートメントを使用します。テーブルに書き込まれる列を制御するには、DATA ステートメントで KEEP=または DROP=テーブルオプションを使用するか、KEEP または DROP ステートメントを使用します。

例

例 1: OUTPUT の使用例

- このコード行では、現在の行がテーブルに書き込まれます。

```
output;
```

- このコード行では、指定された条件が真の場合に、現在の行がテーブルに書き込まれます。

```
if deptcode gt 2000 then output;
```

- このコード行では、PHONE 値が欠損している場合に、MARKUP テーブルに行が書き込まれます。

```
if phone=. then output markup;
```

例 2: 入力の各行から複数の行を作成する

入力データの各行から 2 つ以上の行を作成できます。この DS2 プログラムは、SULFA テーブルの各行に対して RESPONSE テーブルに 3 つの行を作成します。

```
data response(drop= (time1 time2 time3));
  method run();
    set sulfa;
    time=time1;
    output;
    time=time2;
    output;
    time=time3;
    output;
   end;
enddata;
```

例 3: 単一の入力テーブルから複数のテーブルを作成する

1 つの入力テーブルから複数のテーブルを作成できます。この例では、OUTPUT は EASTERN と WESTERN の 2 つのテーブルに行を書き込みます。

```
data eastern western;
  method run();
    set cities;
    if location = 'east' then output eastern;
    else output western;
  end;
enddata;
```

例 4: 複数の入力行から 1 つの行を作成する

複数の入力行を結合して 1 つの行にすることができます。この例では、OUTPUT は、入力テーブルの最初の 10 行の DEFECTS の値を合計した 1 つの行を作成します。

```
data discards;
 drop defects;
  method run();
    set gadgets;
        reps+1;
    if reps=1 then total=0;
    total+defects;
    if reps=10 then do;
      output;
      stop;
     end;
  end;
enddata;
```

例 5: スレッドを使用した出力

次の例では、20 個の乱数の結果セットを生成します。データプログラムは 4 つのスレッドを開始します。各スレッドは、変数 **x** で 5 個の乱数を生成し、書き込みます。データプログラムは、SET ステートメントで各スレッドの出力を読み取り、入力された乱数をデータセット **random_data** に書き込みます。

```
/* Thread generates and outputs 5 random numbers */
```

```
thread thread_pgm;
  declare double x;
  method init();
    declare int i;
    streaminit(_threadid_);
    do i = 1 to 5;
      x = rand('uniform', 1);
      output;             /* output variable x */
    end;
  end;
endthread;

 data random_data;
   dcl thread thread_pgm t;
   method run();
     /* Start 4 threads and read the output of each thread. */
     set from t threads=4;
   end;
enddata;
run;
```

次の出力が生成されます。

The SAS System

Obs	x
1	0.58602
2	0.78108
3	0.03955
4	0.31368
5	0.73902
6	0.45233
7	0.98123
8	0.06654
9	0.10132
10	0.62904
11	0.51530
12	0.88424
13	0.35661
14	0.11297
15	0.16502
16	0.79072
17	0.90079
18	0.79053
19	0.26467
20	0.22305

関連項目

ステートメント:

- “PUT ステートメント” (1397 ページ)
- “DATA ステートメント” (1310 ページ)

PACKAGE ステートメント

DS2 パッケージを作成します。

カテゴリ: ブロック

構文

形式 1:

PACKAGE *package* [/ENCRYPT=SAS | AES] [/*table-options*];
... package-body ...
ENDPACKAGE;

形式 2:

PACKAGE *fcmp-package-name* [/ENCRYPT=SAS | AES] [/*table-options*]
LANGUAGE=[']FCMP['] TABLE='*library-name*';
... package-body ...
ENDPACKAGE;

必須引数

package
パッケージ名を指定します。*package* は、次のいずれかの形式をとります。

- *catalog.schema.package*
- *schema.package*
- *catalog.package*
- *package*

catalog
SQL カタログの ANSI SQL 規格が実装された、論理的に関連するスキーマをグループ化したデータコンテナーオブジェクトです。カタログは、データ編成階層の第 1 レベル(最上位)のグループ化メカニズムです。スキーマとともに使用され、名前を修飾する方法を提供します。

schema
SQL スキーマの ANSI SQL 規格が実装された、テーブルやビューなどのファイルや、ストアドプロシジャなどのデータソースでサポートされるその他のオブジェクトをグループ化するデータコンテナーオブジェクトです。スキーマは、catalog とともに使用されて、名前を修飾する方法を提供するグループ化オブジェクトを提供します。

package
パッケージの名前です。

要件 パッケージの命名規則は、データソースに基づいています。詳細については、データソースのドキュメントを参照してください。

package-body
パッケージの宣言とメソッドが含まれています。

fcmp-package-name
FCMP パッケージの名前を指定します。

制限事項 この引数は CAS サーバーではサポートされていません。

参照項目: “FCMP パッケージの使用” (*SAS DS2 プログラマガイド*)

'*library-name*'
FCMP 関数が存在するライブラリの名前を指定します。

制限事項 この引数は CAS サーバーではサポートされていません。

要件 この場所は、PROC FCMP ステートメントの OUTLIB オプションで指定された場所の library.dataset 部分である必要があります。

参照項目: *Base SAS プロシジャガイド*の FCMP プロシジャ

オプション引数

/ENCRYPT=SAS | AES
暗号化アルゴリズムを指定します。SAS は、SAS Proprietary アルゴリズムを指定します。AES は、Advanced Encryption Standard アルゴリズムを指定します。

デフォルト SAS

操作 PACKAGE ステートメントの ENCRYPT オプションは、ENCRYPT=テーブルオプションとは異なり、値も異なります。ENCRYPT=テーブルオプションは、SAS 出力データセットにのみ影響します。詳細については、“ENCRYPT= テーブルオプション” (1484 ページ)を参照してください。

/table-options
DS2 プログラムがパッケージを作成するときに適用するオプションの引数を指定します。テーブルオプションの詳細については、“DS2 テーブルオプション” (1457 ページ)を参照してください。

注 DS2 で認識されないオプションは、基になるデータソースにエラーなしで渡されます。

詳細

パッケージの基本

パッケージは DS2 プログラムに似ています。パッケージの本体は、一連のグローバル宣言とメソッドのリストで構成されます。主な構文上の違いは、PACKAGE ステー

トメントと ENDPACKAGE ステートメントです。これらのステートメントは、グローバルスコープを持つブロックを定義します。スコープの詳細については、“DS2 識別子のスコープ” (*SAS DS2 プログラマガイド*)を参照してください。

パッケージのメソッドライブラリは、別のパッケージを含む他の DS2 プログラムから呼び出すことができます。したがって、INIT、RUN、および TERM メソッドは、パッケージ内で特別な意味を持ちません。

パッケージのコンパイルが成功すると、パッケージのソースコードのコピーが、パッケージ名で識別されるカタログエントリに保存されます。

PACKAGE ステートメントは、すべてのユーザー定義パッケージおよび SAS によって提供される FCMP パッケージに必要です。ハッシュ、ハッシュ反復子、HTTP、JSON、ロガー、行列、PCRXFIND、PCRXREPLACE、SQLSTMT パッケージには、PACKAGE ステートメントは必要ありません。これらのパッケージも SAS から提供されます。詳細については、“DS2 パッケージの概要” (*SAS DS2 プログラマガイド*)を参照してください。

パッケージ変数は、DECLARE PACKAGE ステートメントを使用して宣言されます。パッケージ変数を宣言しても、パッケージのインスタンスは自動的に構築されません。パッケージインスタンスは、パッケージ変数が宣言されたときにコンストラクター引数を提供するか、_NEW_演算子を呼び出すことによって構築されます。

デフォルトでは、再利用のためにテーブルに保存される DS2 パッケージは、SAS 暗号化で暗号化されます。PACKAGE ステートメントで ENCRYPT テーブルオプションを使用すると、このデフォルトを上書きして AES 暗号化を指定できます。SAS Proprietary は、SAS ソフトウェアに含まれている固定エンコーディングアルゴリズムです。詳細については、*SAS Viya プラットフォーム暗号化: 保存データ*を参照してください。

テーブルオプションは、PACKAGE ステートメントで指定できます。パッケージ名の後に指定され、スラッシュが前に付きます。

FCMP パッケージ

SAS は、DS2 言語内からの FCMP 関数およびサブルーチンの呼び出しをサポートする FCMP パッケージを提供します。詳細については、“FCMP パッケージの使用” (*SAS DS2 プログラマガイド*)を参照してください。

比較

パッケージ、DS2 プログラム、スレッドの比較については、“ブロックステートメント” (1295 ページ)を参照してください。

例

例 1: 複素数パッケージの作成

この例では、非常に単純な複素数パッケージを作成します。2 つのグローバル変数 X と Y は、複素数を構成する実数の順序対を表します。この一連のメソッドは、加算や乗算など、複素数に対するさまざまな操作を実行します。

```
package complex;
  dcl double x y;
 method init(double x, double y);
    this.x = x;
    this.y = y;
 end;
 method add(double x, double y);
    this.x = this.x + x;
    this.y = this.y + y;
 end;
 method mult(double x, double y);
    this.x = this.x * x - this.y * y;
    this.y = this.x * y + x * this.y;
 end;
 method norm() returns double;
    return sqrt(x ** 2 + y ** 2);
 end;
 method print();
    put 'x = ' x ' y= ' y;
 end;
endpackage;
run;
```

次の DS2 プログラムは、前の PACKAGE ステートメントで定義されたメソッドをインスタンス化して呼び出します。

```
data _null_;
  method init();
    dcl package complex c();
    dcl package complex c2();
    c.x=3;
    c.y=4;
    d = c.norm();
    put 'd= ' d;
    c.add(5, 6);
    c.print();
    c2.x=7;
    c2.y=24;
    d = c2.norm();
    put 'd= ' d;
    c2.print();
  end;
enddata;
run;
```

次の行が SAS ログに書き込まれます。

```
d= 5
x = 8 y= 10
d= 25
x = 7 y= 24
```

例 2: FCMP パッケージの作成

この例では、FCMP で square ルーチンを作成し、DS2 プログラムでそのルーチンを使用します。現在のディレクトリは、FCMP パッケージの"ライブラリ"として使用されます。

```
libname base '.';

* fcmp defines a function, square;
proc fcmp outlib = base.fcmpsubs.package1;
  function square(a);
    return (a*a);
  endsub;
run;

* define the ds2 package thru which the fcmp functions will be called;
proc ds2;
package pkg /
  overwrite=yes
  language='fcmp'
  table='base.fcmpsubs';
run;

* call fcmp thru the ds2 wrapper package;
data;
  dcl package pkg p();
  dcl double a b;
  method init();
    do a = 10 to 20;
      b=p.square(a);
      put a= b=;
    end;
  end;
enddata;
run;
quit;
```

次の行が SAS ログに書き込まれます。

```
a=10 b=100
a=11 b=121
a=12 b=144
a=13 b=169
a=14 b=196
a=15 b=225
a=16 b=256
a=17 b=289
a=18 b=324
a=19 b=361
a=20 b=400
```

関連項目

- “ユーザー定義パッケージ” (*SAS DS2 プログラマガイド*)
- “FCMP パッケージの使用” (*SAS DS2 プログラマガイド*)

ステートメント:

- “DECLARE PACKAGE ステートメント” (1323 ページ)
- “DECLARE PACKAGE ステートメント: FCMP パッケージ” (1637 ページ)

PUT ステートメント

プログラム変数、配列、および定数の値を SAS ログに出力します。

カテゴリ: ローカル

例:
“例 1: PUT ステートメント” (1401 ページ)
“例 2: PUT を使用した配列の出力” (1402 ページ)
“例 3: _ALL_引数の使用” (1403 ページ)
“例 4: PUT を使用したパッケージ属性の出力” (1403 ページ)

構文

PUT < put-list > [... <put-list>] ;

<put-list>::=
ALL
| '*character-string*'
| ['*character-string*']<eq-expression> [=] [[:] *format* [-L | -C | -R]]

<eq-expression>::=
identifier
| *array-reference*
| *this-expression*
| *package-attribute*

<package-attribute>::=
package-variable.*package-attribute*
| *package-variable*...[.*package-variableN*].*package-attribute*

引数なし

引数なしの PUT は、SAS ログに空白行を出力します。

オプション引数

ALL
事前定義された変数を含むすべての変数の値を SAS ログに出力します。

'*character-string*'
SAS ログに書き込まれるテキストの文字列を指定します。

identifier
値が SAS ログに書き込まれる変数を指定します。

array-reference
配列要素を指定します。添え字は、PUT ステートメントの実行時に整数値に解決される任意の SAS 式にすることができます。配列のすべての要素を出力するには、配列の添え字アスタリスク(*)を使用します。

this-expression
THIS 式を指定します。

参照項目: “THIS 式” (2115 ページ)

package-attribute
package-variable によって参照されるパッケージの属性をドット演算子を使用して指定します。この属性は、パッケージのスカラー属性、配列属性、または配列属性の要素にすることができます。

制限事項 配列のすべての要素を出力するために、配列添字のアスタリスク(*)と組み合わせてドット表記を使用することはサポートされていません。

package-variable
パッケージのインスタンスを指定します。パッケージインスタンスの名前の後には、ドット演算子と package-attribute が続く必要があります。

.
(ドット演算子)は、左側のオペランドから右側のオペランドにアクセスするための呼び出しを実行します。

要件 PUT ステートメントでは、package-variable をドット演算子の左側のオペランドとして指定する必要があります。package-attribute は、右側のオペランドとして指定する必要があります。

操作 ドット表記を使用すると、パッケージの外部にあるコードからパッケージ属性の値を出力できます。ネストされたドット表記を使用すると、パッケージの階層内にネストされたパッケージからパッケージ属性を出力できます。

参照項目: “DOT 演算子式” (2100 ページ)

“DS2 パッケージ内のドット表記” (*SAS DS2 プログラマガイド*)

=
変数または配列要素の後に等号が追加された場合、変数または配列要素の前に、変数または配列要素の名前と等号が出力されます。

:
PUT ステートメントが変数値の出力に使用する出力形式を指定できます。先頭と末尾の空白はすべて削除され、各値の後に 1 つの空白が続きます。

制限事項 ":"を使用する場合は、出力形式を指定する必要があります。

format
データ値が SAS ログに書き込まれるときに使用する出力形式を指定します。出力形式名にコロン修飾子(:)を使用すると、先頭と末尾の空白がすべて削除され、各値の後に 1 つの空白が続きます。デフォルトの配置をオーバーライドするには、配置指定を出力形式に追加します。

−L 値を左揃えにします。

−C 値を中央揃えにします。

−R 値を右揃えにします。

ヒント 出力形式幅によって、値と、出力形式に含まれるカンマ、ドル記号、小数点、その他の特殊文字を出力するのに十分なスペースが確保されるようにしてください。

詳細

PUT ステートメントの使用方法

PUT ステートメントは、キーワード PUT と、それに続く変数と文字定数のリストで構成されます。値を書き込む変数の名前をリストするか、引用符で文字列を指定します。出力形式を指定しない場合、PUT ステートメントはデフォルトの出力形式で変数値を出力し、単一の空白を挿入してから、次の値を出力します。出力形式を指定すると、出力は出力形式幅を使用して書き込まれます。文字値はフィールド内で左揃えになり、先頭と末尾の空白は削除されます。数値はフィールド内で右揃えになります。

存在しないデータ(SAS 欠損値または null 値)がどのように書き込まれるかは、ANSI モードか SAS モードかによって異なります。デフォルトの出力形式である BEST32. が DOUBLE に関連付けられている場合、DOUBLE の欠損ドット値と null 値に対してピリオドが生成されます。特殊欠損値(.a-.z および._)の場合、BEST32.を使用すると、ピリオドの後の文字が生成されます。たとえば、変数が値.a を保持している場合、A が生成されます。INTEGER およびその他の DOUBLE 以外の数値型を欠損にすることはできません。ただし、null にすることはできます。その場合、値が null のとき PUT ステートメントでは何も生成されません。詳細については、“DS2 による null および SAS 欠損値の処理方法” (*SAS DS2 プログラマガイド*)を参照してください。

リスト出力の間隔

リスト出力では、変数値と文字列を出力するときに異なる間隔調整方法を使用します。変数がリスト出力で書き込まれると、SAS は自動的に空白スペースを 挿入します。出力ポインターは、変数値に続く 2 カラム目で停止します。

```
dcl int a b;
```

```
dcl varchar c d;
area = 9924;
city = 1001;
ctry1='Peru';
ctry2='Bolivia';
   put area city ctry1 ctry2;
```

これらの行が SAS ログに書き込まれます。

```
----+----1----+----2
9924 1001 Peru Bolivia
```

ただし、文字列を書き込む場合は、SAS が自動的に空白スペースを 挿入することはありません。出力ポインターは、文字列の最後の文字の直後のカラムで停止します。

文字列と変数値の両方を書き込むときの間隔の問題を回避するために、文字列の最後の文字として空白スペースを 使用することをお勧めします。

出力形式名にコロン修飾子(:)を使用すると、SAS は指定された出力形式で値を出力し、空白スペースを 挿入して、ポインターを次のカラムに移動します。

```
proc ds2;
data _null_;
  dcl int area POP;
  dcl varchar ctry1 ctry2;

  method init();
  area=9924;
  POP=1000;
  ctry1='Perú';
  ctry2='Bolivia';
  put area : octal10. pop : comma8.2 ctry1 ctry2;
  end;
 enddata;
run;
quit;
```

これらの行が SAS ログに書き込まれます。

```
----+----1----+----2----+----3----+
0000023304 1,000.00 Perú Bolivia
```

フォーマットされた出力

出力形式を使用して、SAS が変数値を出力する方法を制御できます。PUT ステートメントは、変数名に続く出力形式を使用して各値を出力します。フォーマットされた出力では、SAS は値の間に空白を自動的に追加しません。フォーマットされた出力では、値がその長さに満たない場合でも、ポインターが出力形式の長さだけ移動します。ポインターが次のカラムに移動します。間にある空白は挿入されません。値で指定よりも少ないカラムが使用されている場合、出力形式幅で指定されたフィールドで、文字値は左揃え、数値は右揃えになります。

```
dcl int a b;
dcl varchar c d;
pop = 1000
area = 9924;
ctry1='Peru';
ctry2='Bolivia';
put area octal10. pop comma8.2 ctry1 ctry2;
```

これらの行が SAS ログに書き込まれます。

```
----+----1----+----2----+----3----+
000000233041,000.00Peru Bolivia
```

出力形式が指定されていない場合、変数のデフォルト出力形式が使用されます。DECLARE ステートメントで HAVING 句を使用して、出力形式を列に関連付けることができます。 詳細については、“DECLARE ステートメント” (1313 ページ)を参照してください。

注: DECLARE ステートメントの HAVING 句を使用してパッケージ属性に割り当てられたデフォルトの出力形式は、PUT ステートメントがドット表記を使用して取得されたパッケージ属性値を書き込むときには適用されません。詳細については、“ドット表記を使用した PUT ステートメントの機能” (*SAS DS2 プログラマガイド*)を参照してください。

比較

PUT ステートメントと PUT 関数の動作は似ています。ただし、PUT ステートメントはその結果を SAS ログに送信しますが、PUT 関数はその引数のフォーマット結果を含む NCHAR 値を返します。

例

注: これらの例の DS2 ステートメントは、SAS DS2 プロシジャを使用して SAS Compute Server で実行されます。

例 1: PUT ステートメント

この例には、複数の PUT ステートメントが含まれています。

```
proc ds2;
 data _null_;
  dcl double x y z a[2];
  dcl varchar(3) s;

  method init();
  x = 1;
  y = 2;
  z = 3;
  s = 'abc';
  a[2] = 99;
  put 'x =' x;         /* 1 */
  put y=
  put z s a[2];        /* 2 */
 end;
enddata;
run;
quit;
```

1 この PUT ステートメントは、変数 x の変数名と値を SAS ログに出力するように指定します。追加の PUT ステートメントは、変数 y と z について同じ情報を要求します。個別の PUT ステートメントが使用されるため、情報は SAS ログの個別の行に書き込まれます。

2 この PUT ステートメントは、変数 z と s の値、および配列 a の 2 番目の要素を、先行する文字列なしで出力することを指定します。すべての変数値は、SAS ログの同じ行に出力されます。

次の行が SAS ログに書き込まれます。

```
x =  1
y=2
3 abc 99
```

注: 変数または配列要素の後に等号が追加されると、変数または配列要素の前に、変数または配列要素の名前と等号が出力されます。たとえば、これら 2 つのコード行は同等です。

```
put x= y= a[2]=;

put 'x='x ' y='y ' a[2]='a[2];
```

例 2: PUT を使用した配列の出力

この例では、一時配列と変数配列の両方のコンテンツを出力します。

```
proc ds2;
data _null_;
  declare  double a[6] having format 4.1;
  vararray double b[2, 3];
  declare  double c[0:1, 2:4, 5:5];

  method init();
    a := (3 6 9 12 15 18);
    b := (3 6 9 12 15 18);
    c := (3 6 9 12 15 18);
    put a[*]=;    /* 1 */
    put b[*]=;
    put c[*]=;
    put 'array a with default format:' a[*];  /* 2 */
    put 'array a with best3.  format:' a[*] best3.;
  end;
enddata;
run;
quit;
```

1 アスタリスク添え字を使用して PUT ステートメントを指定すると、配列のすべての要素の値が SAS ログに出力されます。

2 PUT ステートメントで変数または配列の前にある文字列には、任意のテキストを含めることができます。PUT ステートメントの変数または配列名の後の引用符で囲まれていないテキストは、出力形式として適用されます。

次の行が SAS ログに書き込まれます。

```
a[1]=3.00 a[2]=6.00 a[3]=9.00 a[4]=12.0 a[5]=15.0 a[6]=18.0
b[1,1]=3 b[1,2]=6 b[1,3]=9 b[2,1]=12 b[2,2]=15 b[2,3]=18
c[0,2,5]=3 c[0,3,5]=6 c[0,4,5]=9 c[1,2,5]=12 c[1,3,5]=15 c[1,4,5]=18
array a with default format: 3.00 6.00 9.00 12.0 15.0 18.0
array a with best3.  format:   3  6  9 12 15 18
```

例 3: _ALL_引数の使用

この例では、PUT ステートメントの_ALL_引数を使用して、事前定義された_N_変数を含むすべての 変数の値を SAS ログに出力します。

```
proc ds2;
data _null_;
 dcl double a b c;
 method init();
   a = 116;
   b = 220;
   c = 37;
   put _all_;
 end;
enddata;
run;
quit;
```

次の行が SAS ログに書き込まれます。

```
a=116 b=220 c=37 _n_=1
```

例 4: PUT を使用したパッケージ属性の出力

このプログラム例では、ドット表記を使用してパッケージ属性に関するデータを出力する方法を示しています。

```
proc ds2;
/* The package named information declares global variable age
 *  and constructor method information. */
package information / inline;
   declare int age;
   method information(int age);
     this.age = age;
   end;
endpackage;

/* The package named person declares a package-variable named
 * info, a global array variable named address, and
 * a constructor method named person. Package-variable info
 * is initialized to a null package reference. */
package person / inline;
   declare package information info;
   declare char(20) address[2];
   method person(int age);
     info = _new_ information(age);
   end;
endpackage;

data _null_;
  method run();
```

```
    declare package person a(15);      /* 1 */

    a.address[1] = '123 SAS Campus';   /* 2 */
    a.address[2] = '1 Main Street';

    put a.address[1]=;                 /* 3 */
    put a.address[2]=;

    put a.info.age=;
  end;
enddata;
run;
quit;
```

1 データブロックの RUN メソッドは、パッケージ person をインスタンス化し、ローカル変数 a を割り当てて、パッケージ person インスタンスを参照します。パッケージ person のインスタンス化中に、次の暗黙の操作が実行されます。

- パッケージ person のパッケージメソッド person は、引数値 15 で呼び出されます。
- パッケージメソッド person は、_NEW_演算子を使用してパッケージ information をインスタンス化します。パッケージ person の info 属性は、パッケージ information インスタンスを参照するために割り当てられます。
- パッケージ information のインスタンス化中に、パッケージメソッド person のパラメーター age の値は、パッケージメソッド information に渡されます。そこでは、パッケージ information の age 属性に割り当てられます。

2 ドット表記を使用すると、各割り当てステートメントは次のようになります。

- RUN メソッドのローカル変数 a で参照されるパッケージ person インスタンスを取得します。
- person パッケージインスタンスから配列属性 address を取得します。
- 配列属性 address の要素に値を割り当てます。

3 ドット表記を使用すると、PUT ステートメントは次のようになります。

- RUN メソッドのローカル変数 a で参照されるパッケージ person インスタンスを取得します。
- パッケージ person インスタンスのパッケージ属性 info で参照されるパッケージ information インスタンスを取得します。
- パッケージ information インスタンスからパッケージ属性 age の値を取得します。
- 値を SAS ログに書き込みます。

SAS ログに書き込まれる出力は次のとおりです。

```
a.address[1]=123 SAS Campus
a.address[2]=1 Main Street
a.info.age=15
```

関連項目

- “配列の内容を書き込む方法” (*SAS DS2 プログラマガイド*)
- “例: ドット表記を使用した配列属性へのアクセス” (*SAS DS2 プログラマガイド*)
- “ドット表記を使用した PUT ステートメントの機能” (*SAS DS2 プログラマガイド*)

関数:

- “PUT 関数” (1038 ページ)

RENAME ステートメント

出力テーブルの列の新しい名前を指定します。

カテゴリ: グローバル

構文

RENAME *old-name* {= | AS} *new-name* [... *old-name* {= | AS} *new-name*];

必須引数

old-name
入力テーブル、または新規作成列の場合は現在の DS2 プログラムに表示される列の名前を指定します。

new-name
出力テーブルで使用する名前を指定します。

詳細

RENAME ステートメントを使用すると、1 つ以上の列の名前を変更できます。新しい列名は、出力テーブルにのみ書き込まれます。現在の DS2 プログラムのプログラミングステートメントでは古い列名を使用します。RENAME は、すべての出力テーブルに適用されます。RENAME ステートメントは、列の名前を変更するだけでなく、列のラベルも変更します。

比較

- RENAME=テーブルオプションを使用すると、入力テーブルまたは出力テーブルごとに名前を変更する列を指定できます。入力テーブルで使用して、処理前に列の名前を変更します。

- 出力テーブルで RENAME=テーブルオプションを使用する場合、現在の DS2 プログラムのプログラミングステートメントでは引き続き古い列名を使用する必要があります。RENAME=テーブルオプションは、その出力テーブルにのみ影響します。RENAME ステートメントは、すべての出力テーブルに影響します。
- RENAME ステートメントと RENAME=出力テーブルオプションの両方を使用する場合、RENAME ステートメントが優先されます。RENAME ステートメントで X が Y に名前変更され、RENAME=テーブルオプションで X が Z に名前変更された場合、RENAME ステートメントが優先され、X は Y に名前変更されます。
- SET ステートメントの RENAME=テーブルオプションは、入力テーブルの列の名前を変更します。現在の DS2 プログラムのプログラミングステートメントでは、新しい名前を使用する必要があります。

例

次の例は、RENAME ステートメントを示しています。

- rename prod=ProductName;
- rename street as Address cit as City st as State;
- rename oldsalary=newsalary;

関連項目

テーブルオプション

- “RENAME= テーブルオプション” (1519 ページ)

RESIZEARRAY ステートメント

動的境界を持つ一時配列の要素数を設定または変更します。

カテゴリ: グローバル
ローカル

注: 構文規則の角かっこは、オプションの引数を示します。角かっこの前のエスケープ文字(\ )は、構文に角かっこが必要であることを示します。配列の境界は角かっこ([ ])で囲む必要があります。

構文

RESIZEARRAY *array-name* \[*n* \];

必須引数

array-name
動的境界を持つ既存の一時配列の名前を指定します。

\[*n*\]
配列の上限を指定します。*n* 引数は、整数定数または整数式にすることができます。たとえば、次のいずれかが有効です。

```
RESIZE values[100]

RESIZEARRAY values[top+100];
```

注 動的境界のある一時配列の次元と下限は、RESIZEARRAY ステートメントでは変更できません。

詳細

RESIZEARRAY ステートメントは、動的境界のある一時配列の要素数を増減して、配列が *n* 個の要素を持つようにします。サイズ変更後、配列の下限は 1 で、配列の上限は *n* になります。

配列の境界のサイズを複数回変更した場合、配列要素の値は、新しい要素数と古い要素数で少ない方の数までは変更されません。要素数が増やされた場合、新しい要素は欠損または null に初期化されます。

グローバル配列の場合、RESIZEARRAY ステートメントは、グローバル配列を宣言したパッケージ、スレッド、またはデータブロックのメソッドから配列の境界のサイズを変更する場合にのみ呼び出すことができます。

ローカル配列の場合、RESIZEARRAY ステートメントは、ローカル配列を宣言したメソッドから配列の境界のサイズを変更する場合にのみ呼び出すことができます。

RESIZEARRAY ステートメントは、固定境界付きで、宣言された配列引数、変数配列、または一時配列の境界を変更できません。

例: RESIZEARRAY ステートメントの指定

DECLARE ステートメントは、バイナリ型の b 次の名前の動的境界を持つグローバル一時配列を定義します。RESIZEARRAY ステートメントは、配列 b に 10 要素の上限を割り当てます。

```
1         proc ds2;
2         data _null_;
3           dcl binary(2) b[*] having format $hex.;
4           method init();
5             resizearray b[10];
6             b[1] = x'ABCD';
7             put b[1]=;
8           end;
9         enddata;
10          run;
11      !     quit;
b[1]=ABCD
```

関連項目

概念:

- “一時配列” (*SAS DS2 プログラマガイド*)

ステートメント:

- “DECLARE ステートメント” (1313 ページ)

RETAIN ステートメント

RUN メソッドの実行間に、すべての列または列リスト内のすべての列の値が保持されるように指定します。

カテゴリ: グローバル

構文

形式 1: **RETAIN**;

形式 2: **RETAIN** *column-list*;

形式 3: **RETAIN** *column-list* < constant-value >;

形式 4: **RETAIN** *column-list* (< constant-value > ... < constant-value >);

< constant-value >::=

bit_constant
| *hex_constant*
| *floating_constant*
| *decimal_constant*
| *sas_missing_value*
| *integer_constant*
| *string_constant*
| *null*
| **DATE** *character_constant*
| **TIME** *character_constant*
| **TIMESTAMP** *character_constant*

形式 5: **RETAIN** *vararray*;

引数なし

(形式 1)引数を指定しない場合、RETAIN ステートメントにより、RUN メソッドのある実行から次の実行まで、すべての列の値が保持されます。

必須引数

column-list
値を保持する列名を指定します。

vararray
変数配列の名前を指定します。

参照項目: “VARARRAY ステートメント” (1434 ページ)

詳細

列の動作(形式 2)

RETAIN ステートメントは、RUN メソッドの各実行中に、列リスト内のすべての列がその値を保持するように指定します。通常、PDV の列は、RUN メソッドが実行される前に欠損値または null 値のいずれかに設定されます。

初期値の割り当て(形式 3 および 4)

RETAIN ステートメントを使用して、個々の列、列のリスト、または配列のメンバーの初期値を指定します。RETAIN ステートメントに値が記述される場合、リスト内でその値の前に表示される列は、最初はその値に設定されます。

(形式 3)すべての列に 1 つの値を割り当てることができます。たとえば、次のステートメントは、列 A、B、および C に値 100 を割り当てます。

```
retain a b c 100;
```

(形式 4)各列に異なる値を割り当てることができます。たとえば、次のステートメントは、値'Vancouver'、'BC'、および'Canada'を列 CITY、PROVINCE、および COUNTRY に割り当てます。

```
retain city province country ('Vancouver', 'BC', 'Canada');
```

反復子を使用してすべての列に 1 つの値を割り当てられることにも注意してください。たとえば、次のステートメントは、列 A、B、および C に値 100 を割り当てます。

```
retain a b c (3* 100);
```

(形式 5)配列変数に初期値を割り当てることができます。RETAIN 初期化子は、配列の境界を越えて機能します。また、初期リストが短い場合、残りの値は欠損値に設定されます。この例では、ns[3]の値が欠損値です。

```
proc ds2;
data y /overwrite=yes;
  vararray double c[5];
  vararray double sums[dim(c)];
  vararray double ns[dim(c)];

  retain sums ns (1 2 3 4 5 6 7);
  method init();
    sums[4] = 99;
  end;
```

```
  method run();
    put sums[1]=;
    put sums[2]=;
    put sums[3]=;
    put sums[4]=;
    put sums[5]=;
    put ns[1]=;
    put ns[2]=;
    put ns[3]=;
  end;
enddata;
run;
quit;
```

冗長性

これらのアイテムは、DS2 プログラムのある反復から次の反復まで自動的に値が保持されるため、RETAIN ステートメントで名前を付けるのは冗長です。

- SET ステートメントで読み取られる列

  注: 複数のテーブルが SET ステートメントに関連付けられている場合は、冗長ではない可能性があります。テーブル **A** が変数 **X** を定義しているが、テーブル **B** は定義していないと仮定します。RETAIN ステートメントがない場合、レコードがテーブル **B** から読み取られるときに、変数 **X** は欠損に設定されます。RETAIN ステートメントがある場合、変数 **X** には、テーブル **A** または DS2 プログラムロジックによって最後に割り当てられた値が含まれます。

- Sum ステートメントで値が割り当てられている列
- 配列で指定されたデータ要素

比較

RETAIN ステートメントは、値が保持される列を指定します。KEEP ステートメントは、作成されるすべてのテーブルに 含まれる列を指定します。

関連項目

ステートメント:

- “KEEP ステートメント” (1364 ページ)

RETURN ステートメント

メソッドからメソッド呼び出し元に実行を返します。

カテゴリ: ローカル

構文

RETURN [*expression*];

引数なし

RETURN ステートメントに *expression* がない場合、制御は RETURN ステートメントが配置されているメソッドの呼び出し元に戻されます。メソッドの呼び出し元に値は返されません。

オプション引数

expression
: 単一の値を返す有効な式を指定します。式の型が評価され、必要に応じて、METHOD ステートメントの RETURNS 句で指定された型に変換されます。次に、*expression* の値がメソッドの呼び出し元に返されます。

詳細

暗黙的なループで RETURN ステートメントが実行されると、暗黙的なループの次の反復が実行されます。RETURN ステートメントは制御を移し、*expression* が存在する場合は、メソッドの呼び出し元に値を返します。値を返す(つまり、METHOD ステートメントに RETURNS 句がある)メソッドには、METHOD 本体の最後のステートメントとして RETUR *expression* ステートメントが必要です。それ以外の場合は、エラーが発生します。メソッドに RETURN *expression* ステートメントがあり、RETURNS 句がない場合、警告が発生します。

STOP ステートメントを使用して、RUN メソッドを終了させることができます。

例

この例では、CONCAT メソッドは連結文字列を返します。RETURN ステートメントの型は、RETURNS 句の型に変換されます。この例では、戻り型は CHAR です。

```
method concat(char(100) x, char(100) y) returns char(200);
  return trim(x) || y;
end;
```

関連項目

ステートメント:

- “METHOD ステートメント” (1376 ページ)

SELECT ステートメント

複数のステートメントのいずれか、またはステートメントグループを実行します。

カテゴリ: ローカル

構文

SELECT [(*select-expression*)];
[< when-list > [...< when-list>]] ;
[OTHERWISE *statement-list*] ;

END [*end-label*];

<when-list>::=
WHEN (*when-expression*) [*statement-list*]

オプション引数

select-expression
VARBINARY 以外の任意の型の単一の値に評価される式を指定します。

end-label
END ステートメントは SELECT ステートメントを閉じます。オプションの *end-label* 引数は、識別子を指定します。Labels ステートメントを使用して作成されたこのラベルは、SELECT ステートメントの直前のラベルと一致する必要があります。一致しない場合、エラーが発生します。

when-expression
任意の式を指定します。

要件 *when-expression* を少なくとも 1 つ指定する必要があります。

statement-list
1 つまたは複数の任意の実行可能ステートメントにすることができます。

詳細

SELECT グループでの WHEN ステートメントの使用

この SELECT ステートメントは、SELECT グループを開始します。SELECT グループには、特定の条件が真のときに実行される DS2 ステートメントを識別する WHEN ステートメントが含まれます。SELECT グループで少なくとも 1 つの WHEN ステートメントを使用します。オプションの OTHERWISE ステートメントでは、WHEN 条件が満たされない場合に実行されるステートメントを指定します。END ステートメントで、SELECT グループを終了します。

WHEN ステートメントで使用される Null ステートメントは、条件が真の場合にそれ以上のアクションを実行しません。

注: SELECT ステートメントはネストできます。

注: SELECT 式を使用して、他の式の値に基づいて複数の式から選択できます。詳細については、“SELECT 式” (2116 ページ)を参照してください。

注: 論理演算では、WHEN ステートメントを含むすべての式で、欠損値は SAS モードで偽と評価されます。null 値は、ANSI モードでは真にも偽にも評価されません。

select-expression が含まれている場合の *when-expression* の評価

select-expression が存在する場合、*select-expression* と *when-expression* の両方が評価されます。2 つが等しいかどうかが比較され、真または偽の値が返されます。比較が真の場合、*statement-list* が実行され、SELECT ステートメントの処理が終了します。その他の条件はテストされません。

比較が偽の場合、実行は次の WHEN ステートメントに進みます。WHEN ステートメントが残っていない場合は、OTHERWISE ステートメントがあればそちらに実行が進みます。すべての SELECT-WHEN 比較の結果が偽であり、OTHERWISE ステートメントが存在しない場合、エラーは発生せず、DS2 プログラムは実行を継続します。

select-expression が含まれていない場合の *when-expression* の評価

select-expression が存在しない場合、*when-expression* が評価され、真または偽の結果が生成されます。結果が真の場合、*statement-list* が実行され、SELECT ステートメントの処理が終了します。その他の条件はテストされません。

結果が偽の場合、実行は次の WHEN ステートメントに進むか、OTHERWISE ステートメントがある場合はそちらに進みます。(つまり、最初の真の WHEN ステートメントで示されるアクションが実行されます。)すべての *when-expressions* の結果が偽で、OTHERWISE ステートメントも存在しない場合、エラーメッセージが発行されます。複数の WHEN ステートメントに真の *when-expression* がある場合、最初の WHEN ステートメントのみが使用されます。一度 *when-expression* が真になると、その他の *when-expression* は評価されません。

statement-list が含まれていない場合の *when-expression* の評価

when-expression が *statement-list* なしで記述された場合、次の *when-expression* の *statement-list* が使用されます。次の例では、10 の出力が生成されます。

```
a = 10;
  select(a);
  when(10)
    when(20) put a=;
```

ただし、*statement-list* のない *when-expression* では、この動作は中断されます。この例では、出力は生成されません。

```
a = 10;
  select(a);
  when(10) ;
  when(20) put a=;
```

注: これは、OTHERWISE の前にある *when-expression* には当てはまりません。この場合、ステートメントのない *when-expression* は、空のステートメント(;)があるかのように扱われます。

比較

ステートメントが少ないプログラムには、IF-THEN/ELSE ステートメントを使用します。THEN 句なしでサブセット化 IF ステートメントを使用すると、IF 句で指定された条件を満たす行のみ処理が続行されます。

例

例 1: select-expression を使用した SELECT

この例は、SELECT ステートメントを *select-expression* と一緒に使用する方法を示しています。

```
select (a);
  when (1) x=x*10;
  when (2);
  when (3) x=x*100;
  when (4) x=x*100;
  when (5) x=x*100;
  otherwise;
end;
```

例 2: select-expression を使用しない SELECT

これは、*select-expression* を使用しない SELECT ステートメントの例です。

```
select;
  when (mon in ('JUN', 'JUL', 'AUG')
  and temp>70) put 'SUMMER ' mon=;
```

```
  when (mon in ('MAR', 'APR', 'MAY'))
  put 'SPRING ' mon=;
  otherwise put 'FALL OR WINTER ' mon=;
end;
```

例 3: IF 式を使用しない SELECT

この例では、SELECT ステートメントを使用しています。IF 式を使用することもできます。IF 式を使用すると、計算をよりコンパクトに表現できます。

```
temp3=.;
select(age);
  when(12) temp3=0;
  when(13) temp3=-14.2769145744513;
  when(14) temp3=-27.3577925153955;
  when(15) temp3=-21.9485551871151;
  when(16) temp3=-13.1764092846992;
  when(17) temp3=-0.63898626243485;
end;
temp4=.;
select(sex);
  when('F') temp4=-1.47186167693037;
  when('M') temp4=1.47186167693037;
end;
temp5=.;
select(age);
  when(12) DO;
    select(sex);
      when('F') temp5=0;
      when('M') temp5=0;
    end;
  end;
  when(13) DO;
    select(sex);
      when('F') temp5=7.47012474340756;
      when('M') temp5=-7.47012474340756;
    end;
  end;
  when(14) DO;
    select(sex);
      when('F') temp5=4.35656087162482;
      when('M') temp5=-4.35656087162482;
    end;
  end;
  when(15) DO;
    select(sex);
      when('F') temp5=2.40720037896732;
      when('M') temp5=-2.40720037896732;
    end;
  end;
  when(16) DO;
    select(sex);
      when('F') temp5=7.56020843202274;
      when('M') temp5=-7.56020843202274;
    end;
  end;
```

```
  when(17) DO;
    select(sex);
      when('F') temp5=-0.0520606347702515;
      when('M') temp5=0.0520606347702515;
    end;
  end;
end;

predictedWeight = -182.556181904311 + temp3 + temp4 +
  4.85030791094268*height + temp5;
```

関連項目

- “SELECT 式” (2116 ページ)

ステートメント:

- “IF ステートメント: サブセット化” (1360 ページ)
- “IF-THEN/ELSE ステートメント” (1362 ページ)

SET ステートメント

1 つ以上のテーブルから行を読み取ります。

カテゴリ: ローカル

注: 構文規則の中かっこは、構文のグループ化を示します。中かっこの前のエスケープ文字(\)は、構文に中かっこが必要であることを示します。*sql-text* は中かっこ({ })で囲む必要があります。

ヒント: スレッドプログラムの SET ステートメントは、その SET ステートメントの単一のリーダーを共有します。SET ステートメントが実行されるたびに、指定された入力テーブルから 1 行が読み取られます。ただし、すべてのスレッドが単一のリーダーを共有します。入力テーブルの各行は、正確に 1 つのスレッドに送信されます。INIT メソッドで SET ステートメントを使用する場合、最初のスレッドがすべての行を読み取ります。他のスレッドはファイルの終わりに到達し、関連する RUN メソッドに進まずに処理を終了します。したがって、スレッドプログラムの INIT または TERM メソッドでは、SET ステートメントを使用しないことをお勧めします。

結果の列の幅は、単一の SET ステートメントのすべてのテーブルにわたる 最大幅によって決定されます。末尾の空白は、SET ステートメントとは無関係です。

構文

SET < table-reference > [... < table-reference >] [INDSNAME=*variable*] ;

[BY [DESCENDING] *column* [... [DESCENDING] *column*]] ;

< table-reference>::=

{ *table* [(*table-options*)] }

| \{ *sql-text* \}

必須引数

table

入力テーブルの名前を指定します。*table* は、次のいずれかの形式をとります。

- *catalog.schema.table-name*
- *schema.table-name*
- *catalog.table-name*
- *table-name*
- *caslib.table-name*

catalog	SQL カタログの ANSI SQL 規格が実装された、論理的に関連するスキーマをグループ化したデータコンテナーオブジェクトです。カタログは、 データ編成階層の第 1 レベル(最上位)のグループ化メカニズムです。 スキーマとともに使用され、名前を修飾する方法を提供します。
schema	SQL スキーマの ANSI SQL 規格が実装された、テーブルやビューなどのファイルや、ストアドプロシジャなどのデータソースで サポートされるその他のオブジェクトをグループ化するデータコンテナーオブジェクトです。スキーマは、catalog とともに使用されて、名前を修飾する方法を提供する グループ化オブジェクトを提供します。
table-name	テーブルの名前です。
caslib.table-name	CAS サーバーにあるテーブルの名前です。

制限事項

DS2 には WHERE テーブルオプションがありません。SET ステートメントで WHERE 句を使用する場合は、{*sql-text*}を使用します。“例 3: SET ステートメントでの WHERE 句の指定” (1420 ページ)を参照してください。

DS2 には、FIRSTOBS=および OBS=テーブルオプションがありません。SET ステートメントで行制御オプションを指定する場合は、{*sql-text*}を使用します。“例 4: 処理される行数の指定” (1420 ページ)を参照してください。

注

テーブル名にドットが含まれている場合は、テーブル名を二重引用符で囲む必要があります。次に例を示します。

```
set "my.schema"."my.table";
```

テーブル名とスキーマ名を引用符で囲まない場合、DS2 はテーブル名とスキーマ名を大文字で保存し、二重引用符を加えます。引用符で囲まれたテーブル名とスキーマ名は、そのまま使用されます。つまり、引用符で囲まれた元の大文字小文字が引用符で囲まれたままになります。たとえば、data mytable;では、テーブル名は"MYTABLE"として保存され、data "MyTable";で

は、テーブル名は"MyTable"として保存されます。これは、データソースのテーブル名とスキーマ名で大文字と小文字が区別される場合に重要です。

注意 **LIBNAME ステートメントで PRESERVE_TAB_NAMES=no オプションを使用すると、予期しない結果が生じる可能性があります。**

{*sql-text*}
テーブル行のセットに解決される有効な FedSQL コードです。

制限事項 SET ステートメントの SQL は、列がデータソーステーブルに含まれている場合にのみ列を参照できます。

要件 FedSQL クエリは中かっこ({ })で囲む必要があります。

ヒント *sql-text* を使用すると、1 つ以上のテーブルの行をマージできます。

SET ステートメントの SQL は、コンパイル時に静的に評価されます。

参照項目: *SAS FedSQL 言語リファレンス*の SELECT ステートメント

オプション引数

INDSNAME=*variable*
現在の行が読み取られるテーブルの名前を保存する変数を作成して名前を付けます。保存される名前は、テーブル名または物理名にすることができます。物理名は、動作環境がファイルを認識するための名前です。

データの種類 NCHAR

ヒント INDSNAME 変数は自動的に NCHAR として宣言されますが、明示的に CHAR として宣言することができます。

事前に定義されていない限り、変数の長さは 41 文字に設定されます。変数が特定の長さの CHAR として宣言されている場合、その長さは変更されません。INDSNAME 変数に入れられた値がその長さより長い場合、値は切り捨てられます。

column
テーブルを並べ替えるための各列の名前を指定します。

ヒント テーブルは複数の列で並べ替えることができます。

table-options
DS2 プログラムで出力テーブルに行を書き込むときに適用するオプションの引数を指定します。テーブルオプションの詳細については、“DS2 テーブルオプション” (1457 ページ)を参照してください。

DESCENDING
指定された列によってテーブルが降順で並べ替えられることを指定します。DESCENDING は、数値列の場合は最大から最小へ、文字列の場合はアルファベットの逆順になります。

詳細

SET の機能

SET ステートメントは柔軟性があり、DS2 プログラミングでさまざまな用途があります。

- DS2 プログラムでさらに処理するために、既存のテーブルから行と列を読み取る
- テーブルの連結とインターリーブ、およびテーブルの 1 対 1 の読み取りの実行

これらの使用法は、SET ステートメントで使用するオプションとステートメントによって決まります。

SET ステートメントが実行されるたびに、1 行がプログラムデータベクトルに読み込まれます。SET は、特に指定しない限り、入力テーブルから一度に 1 行ずつすべての列を読み取ります。SET ステートメントには複数のテーブルを含めることができ、DS2 プログラムには、複数の SET ステートメントを含めることができます。

注: スレッドプログラムの SET ステートメントは、その SET ステートメントの単一のリーダーを共有します。入力テーブルの各行は、正確に 1 つのスレッドに送信されます。

注: SET ステートメントは、RUN メソッドの暗黙的なループ機能を利用できるように、RUN メソッドでの使用が適しています。

例

例 1: テーブルの読み取り

この例では、テーブル NC.MEMBERS の各行がプログラムデータベクトルに読み込まれます。CITY の値が Raleigh である行のみが、新しいテーブル RALEIGH.MEMBERS に書き込まれます。

```
data raleigh.members;
  method run();
    set nc.members;
    if city='Raleigh';
  end;
enddata;
run;
```

例 2: テーブルの連結

SET ステートメントに複数のテーブル名が記述されている場合、結果の出力テーブルは、リストされているすべてのテーブルを 連結したものになります。SAS は、最初のテーブルからすべての 行を読み取り、次に 2 番目のテーブルからすべての 行を読み取り、以下同様に、すべてのテーブルのすべての 行が読み取られるまで続けます。この例では、3 つのテーブルを FITNESS という名前の 1 つの出力テーブルに連結します。

```
data fitness;
  method run();
    set health exercise well;
  end;
enddata;
run;
```

例 3: SET ステートメントでの WHERE 句の指定

WHERE 式をサブミットするには、SET ステートメントで FedSQL SELECT ステートメントを指定する必要があります。

```
data investment_history;
  method run();
    set {select * from investment where investment_year > 2010};
  end;
enddata;
run;
```

例 4: 処理される行数の指定

入力テーブルから指定された数の行を要求するには、FedSQL SELECT ステートメントで OFFSET 句と LIMIT 句を指定します。OFFSET は、SELECT ステートメントが行を返し始める前にスキップする行数を指定します。LIMIT は、SELECT ステートメントが返すことができる行数を指定します。

```
data test;
    method run();
      set {select * from employees offset 15 limit 10};
    end;
  enddata;
  run;
```

この例では、FedSQL は 15 行目から始まる 10 行を返します。

例 5: テーブルのインターリーブ

2 つ以上のテーブルをインターリーブするには、SET ステートメントの後に BY ステートメントを使用します。

```
data april;
  method run();
    set payable recvable;
    by account;
  end;
enddata;
run;
```

例 6: 単一行をテーブル内のすべての行と結合する

あるテーブルの単一行を別のテーブルのすべての行と結合するには、INIT()メソッドで SET ステートメントを使用して単一行を選択します。INIT メソッドの SET ステートメントは、AVGSALES テーブルから 1 行のデータを読み取るために 1 回呼び出されます。2 番目のテーブルからすべての行を取得するには、RUN()メソッドで SET ステートメントを使用します。RUN()メソッドには、暗黙的なループ機能があります。

```
data national;
  method init ();
    set avgsales;
  end;
  method run();
    set totsales;
  end;
enddata;
run;
```

結果のテーブル NATIONAL には、AVGSALES テーブルの 1 つの行からのデータと、TOTSALES テーブルのすべての 行からのデータが含まれます。

例 7: 同じ SAS データセットから複数回読み取る

この例では、SAS は各 SET ステートメントを個別に処理します。つまり、2 つの別々の SAS データセットから読み取るかのように、1 つの SAS データセットから読み取ります。最初の SET ステートメントで LOCKTABLE=テーブルオプションを使用して、同じ SAS データセット(この場合は trial5)を同時に開くことができるようにしています。

```
data drugxyz;
  method run();
    set trial5(locktable=share keep=(sample));
    if sample>2;
      set trial5;
  end;
enddata;
run;
```

DS2 プログラムの反復ごとに、最初の SET ステートメントは 1 行を読み取ります。次に最初の SET ステートメントが実行されると、次の行が読み取られます。各 SET ステートメントは、DS2 プログラムの同じ反復で異なる行を読み取ることができます。

例 8: 埋め込み SQL ステートメントでのテーブルオプションの指定

FedSQL を使用して同じテーブルから複数回読み取る場合は、両方のテーブル参照で LOCKTABLE=テーブルステートメントを指定する必要があります。FedSQL 言語では、テーブルオプションを指定するための構文が DS2 とは異なることに注意してください。

```
data drugxyz;
  method run();
    set trial5 (locktable=share){select sample from trial5 {options locktable=share}
      where sample>2};
  end;
enddata;
run;
```

関連項目

- “SET ステートメントを使用したデータの読み取り” (*SAS DS2 プログラマガイド*)
- “DS2 テーブルの結合: 方法” (*SAS DS2 プログラマガイド*)

ステートメント:

- “BY ステートメント” (1304 ページ)
- “DATA ステートメント” (1310 ページ)
- “DECLARE PACKAGE ステートメント: 行列パッケージ” (1845 ページ)
- “MERGE ステートメント” (1370 ページ)

テーブルオプション:

- “IN= テーブルオプション” (1497 ページ)
- “LOCKTABLE= テーブルオプション” (1505 ページ)

SET FROM ステートメント

DS2 プログラムを 1 つ以上のスレッドとして実行します。

カテゴリ: ローカル

制限事項: データプログラムでは、複数の SET FROM ステートメントは使用できません。それ以外の場合は、エラーが発生します。

構文

SET FROM *thread* [THREADS = *threads*];

必須引数

thread
SET ステートメントによって実行されるスレッド名を指定します。*thread* は、次の形式の 1 つにすることができます。

- *catalog.schema.thread*
- *schema.thread*
- *thread*

catalog
SQL カタログの ANSI SQL 規格が実装された、論理的に関連するスキーマをグループ化したデータコンテナーオブジェクトです。カタログは、データ編成階層の第 1 レベル(最上位)のグループ化メカニズムです。スキーマとともに使用され、名前を修飾する方法を提供します。

schema
SQL スキーマの ANSI SQL 規格が実装された、テーブルやビューなどのファイルや、ストアドプロシジャなどのデータソースでサポートされるその他のオブジェクトをグループ化するデータコンテナーオブジェクトです。スキーマは、catalog とともに使用されて、名前を修飾する方法を提供するグループ化オブジェクトを提供します。

thread
スレッドの名前です。

要件 スレッド名は、THREAD ステートメントで作成されたスレッドの名前と一致する必要があります。また、スレッドは、SET FROM ステートメントが実行される前に作成される必要があります。そうでない場合、エラーが発生します。

スレッドの命名規則は、データソースに基づきます。詳細については、データソースのドキュメントを参照してください。

参照項目: *"スレッド処理" (SAS DS2 プログラマガイド)*

オプション引数

THREADS= *threads*
thread に対して実行されるスレッドの数を指定します。

制限事項 FCMP パッケージを使用している場合、*threads* の最大値は 100 です。それ以外の場合、*threads* の値に制限はありません。

要件 *threads* は、整数値である必要があります。

ヒント *threads* が存在しない場合、スレッドは単一のスレッドとして実行されます。

詳細

基本

SET FROM ステートメントを使用すると、DS2 プログラムを単一のスレッドまたは複数のスレッドとして実行できます。SET FROM ステートメントで指定されたスレッド名は、THREAD ステートメントによって作成された DS2 プログラムスレッドを参照します。

注: SET FROM ステートメントは、RUN メソッドの暗黙的なループ機能を利用できるように、RUN メソッドでの使用が適しています。

注: MongoDB データソースには、テーブルを作成するための特別な要件があります。データに応じて、ルートテーブル、親テーブル、子テーブルを作成することをお勧めします。DS2 ではルートテーブルのみを作成できます。親テーブルと子テー

ブルを作成するには、FedSQL を使用する必要があります。*SAS/ACCESS for Relational Databases: リファレンス*の MongoDB に関する情報を参照してください。

CAS での THREADS=引数と DS2 スレッドプログラム

SAS Cloud Analytic Services (CAS)の分析の基本的な計算モデルは、分散コンピューティングとマルチスレッドコンピューティングを組み合わせたものです。データを渡す CAS アクションは通常、複数のスレッドによって行われます。各スレッドはデータの一部を受け取り、入力テーブルの各レコードは 1 つのスレッドによってのみ消費されます。これらのスレッドは、実行の同時性を強調するために *ワーカースレッド*と呼ばれます。使用できる同時ワーカースレッドの最大数は、ソフトウェアライセンスによって決まります。

許可されている最大数よりも多いスレッドを DS2 プログラムで指定している場合、DS2 はスレッド数をその最大数に減らします。DS2 プログラムで 0 スレッドが指定されているか、スレッド数が指定されていない場合、DS2 は、ソフトウェアライセンスで決定されている、許可されている最大スレッド数を使用します。

比較

SET FROM で指定されたスレッドが実行を開始した後、SET FROM ステートメントは SET ステートメントと同じように実行されます。

注意すべき類似点は次のとおりです。

- スレッドの PDV 情報は、SET FROM ステートメントが指定された DS2 プログラムによって読み取られるため、スレッドからのすべての出力変数は、DS2 プログラムで正しい型で自動的に宣言されています。
- SET FROM と SET はどちらも、読み取る行がなくなるまで入力をループします。

注意すべき相違点は次のとおりです。

- SET FROM ステートメントは、テーブルから読み取るのではなく、各スレッドからの出力を読み取ります。
- 一般に、SET FROM ステートメントの入力は、実行中のすべてのスレッドが、スレッドの OUTPUT ステートメントを介して生成する出力のストリームで構成されます。スレッドの実行順序は予測できないため、SET ステートメントがテーブルを読み取るように、入力が順番に読み取られることはありません。スレッドにテーブルから行を読み取る SET ステートメントが含まれている場合、行はスレッド間で非同期に分割されます。スレッドが SET ステートメントを使用してデータを読み取っていない場合、SET FROM ステートメントの入力は、テーブルの 1 つ以上のコピーを読み込むことと似ていますが、読み込まれる行の順序は指定されていません。

例

例 1: 単一のスレッドの実行

この例では、スレッド X と Y は、DS2 プログラムで適切な型で自動的に宣言されています。SET FROM ステートメントが実行されると、T は単一のスレッドとして開始され、その行が読み取られ、それぞれに対して簡単な計算が実行されます。

```
thread work.t;
  dcl int x;
  dcl double y;
  method init();
    dcl int i;
    do i = 1 to 5;
      x = i;
      y = i * 2.5;
      output;
    end;
  end;
endthread;
data;
  dcl thread work.t t;
  method run();
    set from t;
    sum = x + y;
    put ' x= ' x ' y= ' y ' sum= ' sum;
  end;
enddata;
```

次の行が SAS ログに書き込まれます。

```
x= 1 y= 2.5  sum= 3.5
x= 2 y= 5    sum= 7
x= 3 y= 7.5  sum= 10.5
x= 4 y= 10   sum= 14
x= 5 y= 12.5 sum= 17.5
```

例 2: 複数のスレッドの実行

この例では、SET FROM ステートメントに THREADS オプションを追加することにより、スレッド T を変更して複数のスレッドを実行します。

```
thread work.t;
  dcl int x;
  dcl double y;
  method init();
    dcl int i;
    do i = 1 to 5;
      x = i;
      y = i * 2.5;
      output;
    end;
  end;
endthread;
data;
  dcl thread work.t t;
```

```
  method run();
    set from t threads=2;
    sum = x + y;
    put ' x= ' x ' y= ' y ' sum= ' sum;
  end;
enddata;
```

これにより、T に対して 2 つのスレッドが実行されます。次の行が SAS ログに書き込まれます。

```
x= 1 y= 2.5 sum= 3.5
x= 2 y= 5 sum= 7
x= 3 y= 7.5 sum= 10.5
x= 4 y= 10 sum= 14
x= 5 y= 12.5 sum= 17.5
x= 1 y= 2.5 sum= 3.5
x= 2 y= 5 sum= 7
x= 3 y= 7.5 sum= 10.5
x= 4 y= 10 sum= 14
x= 5 y= 12.5 sum= 17.5
```

この場合、出力はシーケンシャルですが、一貫して発生する保証はありません。

例 3: スレッド値の累計

この例では、スレッド T は値 1 を生成します。DS2 プログラムでは、4 つのスレッドが開始され、すべての出力値が合計され、TERM メソッドで出力されます。

```
thread t;
  dcl int x;
  method init();
    x = 1;
    output;
  end;
endthread;
data;
  dcl thread t t;
  dcl int sum;
  method init();
    sum = 0;
  end;
  method run();
    set from t threads=4;
    sum + x;
  end;
  method term();
    put 'sum= ' sum;
  end;
enddata;
```

次の行が SAS ログに書き込まれます。

```
sum= 4
```

関連項目

- “スレッド処理” (*SAS DS2 プログラマガイド*)

ステートメント:

- “SET ステートメント” (1416 ページ)
- “THREAD ステートメント” (1429 ページ)

STOP ステートメント

現在の DS2 プログラムの実行を停止します。

カテゴリ: ローカル

構文

STOP;

引数なし

STOP ステートメントを使用すると、現在の DS2 プログラムの処理がすぐに停止し、現在の DS2 プログラムの終了後にステートメントの処理が再開されます。

詳細

DS2 がテーブルを生成する場合、STOP の実行時に処理される行は追加されません。STOP ステートメントは、単独で使用することも、IF-THEN/ELSE ステートメントまたは SELECT グループで使用することもできます。

TERM メソッドは、STOP ステートメントが実行されるメソッドに関係なく、常に実行されます。TERM メソッドで STOP ステートメントを使用すると、TERM メソッドは STOP ステートメントが実行された時点で停止します。

STOP ステートメントが INIT メソッドまたは INIT メソッドから呼び出されたメソッドで実行された場合、RUN メソッドは実行されません。

Sum ステートメント

式の結果をアキュムレータ変数に加算または減算します。

カテゴリ: ローカル

注: Sum ステートメントには、グローバル変数のみ使用できます。ローカル変数を Sum ステートメントで使用する場合、値は保持されません。

構文

variable + *expression*;

variable – *expression*;

必須引数

variable
数値を含むアキュムレータ変数の名前を指定します。

制限事項 Sum ステートメントで宣言されていない配列を使用すると、エラーが発生し、メッセージが SAS ログに書き込まれます。

変数名を“x”にすることはできません。

ヒント DS2 が最初の行を読み取る前に、変数は自動的に 0 に設定されます。変数の値は、RETAIN ステートメントに指定されたかのように、ある反復から次の反復まで保持されます。

合計変数を 0 以外の値に初期化するには、初期値として RETAIN ステートメントに含めます。

expression
有効な DS2 式です。

ヒント DS2 では、欠損値または null 値を生成する式をゼロとして扱います。

参照項目: “DS2 式” (*SAS DS2 プログラマガイド*)

詳細

expression が評価され、結果がアキュムレータ変数に追加されます。プラス記号(+)を使用すると、結果がアキュムレータ変数に追加されます。マイナス記号(–)を使用すると、負の結果がアキュムレータ変数に加算される、つまり減算されます。

比較

この例に示すように、Sum ステートメントは SUM 関数と RETAIN ステートメントを使用することと同じことになります。

```
retain variable 0;
variable=sum(variable,expression);
```

例

例 1:

Sum ステートメントの例を次に示します。

```
i + 2;
balance - debit;
numvalid + (not missing(x));
```

例 2: Sum ステートメントでのグローバル変数とローカル変数の使用

次の例では、Sum ステートメントでグローバル変数とローカル変数の両方を使用しています。ローカル変数 **x** の値は、変更されません。

```
data _null_;
 dcl int y;
  method m();
   dcl int x;
   x = 1;
   x + 2;
   y + 4;
   put x= y=;
  end;

  method init();
   do i = 1 to 3;
    m();
   end;
  end;
enddata;
run;
```

次の行が SAS ログに書き込まれます。

```
x=3 y=4
x=3 y=8
x=3 y=12
```

THREAD ステートメント

DS2 プログラムスレッドを作成します。

カテゴリ: ブロック

制限事項: THREAD ステートメントと SETPARMS メソッドのスレッドパラメーターは、CAS サーバーではサポートされません。

パッケージの種類は、THREAD ステートメントのパラメーターとしてサポートされません。DS2 スレッドは、スレッドプログラム内でパッケージのローカルインスタンスをインスタンス化できます。パッケージインスタンスは他の DS2 スレッドか

らアクセスできず、データはスレッド間で共有されません。DS2 プログラムが 10 個の DS2 スレッドを生成する場合、メモリには 10 個のパッケージインスタンスが存在することになります。

構文

THREAD *thread* [(*data-type variable* [, ... *data-type variable*])]
[/ENCRYPT=SAS | AES] [/*table-options*];

... *thread-body* ...

ENDTHREAD;

必須引数

thread
スレッド名を指定します。*thread* は、次のいずれかの形式をとります。

- *catalog.schema.thread*
- *schema.thread*
- *catalog.thread*
- *thread*

catalog
SQL カタログの ANSI SQL 規格が実装された、論理的に関連するスキーマをグループ化したデータコンテナーオブジェクトです。カタログは、データ編成階層の第 1 レベル(最上位)のグループ化メカニズムです。スキーマとともに使用され、名前を修飾する方法を提供します。

schema
SQL スキーマの ANSI SQL 規格が実装された、テーブルやビューなどのファイルや、ストアドプロシジャなどのデータソースでサポートされるその他のオブジェクトをグループ化するデータコンテナーオブジェクトです。スキーマは、catalog とともに使用されて、名前を修飾する方法を提供するグループ化オブジェクトを提供します。

thread
スレッドの名前です。

要件 スレッドの命名規則は、データソースに基づきます。詳細については、データソースのドキュメントを参照してください。

thread-body
スレッドの宣言とメソッドが含まれています。

オプション引数

data-type
オプションのデータ型宣言です。詳細については、“DS2 データ型” (*SAS DS2 プログラマガイド*)を参照してください。

variable
パラメーターを識別するオプションの変数を指定します。

制限事項 THREAD ステートメントのスレッドパラメーターは、CAS サーバーではサポートされません。

/ENCRYPT=SAS | AES
暗号化アルゴリズムを指定します。SAS は、SAS Proprietary アルゴリズムを指定します。AES は、Advanced Encryption Standard アルゴリズムを指定します。

デフォルト SAS

操作 THREAD ステートメントの ENCRYPT オプションは、ENCRYPT=テーブルオプションとは異なり、値も異なります。ENCRYPT=テーブルオプションは、SAS 出力データセットにのみ影響します。詳細については、“ENCRYPT= テーブルオプション” (1484 ページ)を参照してください。

/table-options
DS2 プログラムがスレッドを作成するときに適用するオプションの引数を指定します。テーブルオプションの詳細については、“DS2 テーブルオプション” (1457 ページ)を参照してください。

詳細

DS2 スレッドは、THREAD ステートメントで始まり、ENDTHREAD ステートメントで終わります。これらのステートメントは、グローバルスコープを持つブロックを定義します。グローバルスコープの詳細については、“DS2 識別子のスコープ” (*SAS DS2 プログラマガイド*)を参照してください。

スレッドの本体は、一連のグローバル宣言とメソッドのリストで構成されます。SET FROM ステートメントを使用して、スレッドプログラムが使用するスレッド数を指定できます。

DS2 プログラムは、入力データを処理し、出力データを生成します。DS2 プログラムは、プログラムまたはスレッドとして、2 つの異なる方法で実行できます。DS2 プログラムをプログラムとして実行した場合の結果を次に示します。

- 入力データには、データベーステーブルの行と DS2 プログラムスレッドの行の両方を含めることができます。
- 出力データは、クライアントアプリケーションに返されるデータベーステーブルまたは行のいずれかになります。

DS2 プログラムをスレッドとして実行した場合の結果を次に示します。

- 入力データには、データベーステーブルの行のみを含めることができ、他のスレッドの行は含めることができません。
- 出力データには、スレッドを開始した DS2 プログラムに返される行が含まれます。

スレッドの詳細については、“スレッド処理の概要” (*SAS DS2 プログラマガイド*)を参照してください。

DS2 スレッドには、名前を付ける必要があります。この名前は、スレッドが正常にコンパイルされた後にスレッドのソースコードが保存されるカタログエントリを特定します。他の DS2 プログラムとスレッドは、SET FROM ステートメントを使用して、スレッドを読み取って実行できます。

スレッドが宣言されると、スレッドの名前を持つテーブルが作成されます。テーブルが作成されたことと、作成された場所(通常は Work ライブラリ)を示す情報が SAS ログに書き込まれます。ほとんどの場合、SAS セッションが終了した後、単一レベル名のテーブルは保持されません。しかし、一部の単一レベル名のテーブルは保持されます。複数レベル名のテーブルは、SAS セッション後も常に保持されます。スレッドが保持される場合は、スレッドを再宣言しなくても、複数回実行できます。プログラムに DROP THREAD ステートメントを追加して、不要なテーブルをクリーンアップできます。

スレッドは、DECLARE THREAD ステートメントを使用して、DS2 プログラムでの使用が宣言されます。スレッドを宣言すると、スレッドを表す変数はスレッドのインスタンスとみなされます。スレッド変数は、グローバルスコープにのみ置かれます。それ以外の場合は、エラーが発生します。スレッドのインスタンス化の詳細については、DECLARE THREAD ステートメントを参照してください。

注: 変数がスレッドプログラムの HAVING 句で宣言され、その変数がデータプログラムで HAVING 句を使用して再宣言された場合、スレッドプログラムの HAVING 句の代わりにデータプログラムの HAVING 句が使用されます。データプログラムの DECLARE ステートメントに HAVING 句がない場合、スレッドプログラムの HAVING 句は使用されません。

次の例のように、スレッドはパラメーターを持つことができます。

```
thread work.t (double d, char (100) sp);
```

パラメーター化されたスレッドを使用している場合、パラメーター名とその種類は THREAD ステートメントで指定されます。スレッドを呼び出す DS2 プログラムは、SETPARMS メソッドを呼び出してスレッドのパラメーターを初期化する必要があります。この例では、パラメーター D は値 *99* で初期化され、パラメーター SP は値*'ijk'* で初期化されます。

```
t.setparms(99, 'ijk');
```

デフォルトでは、DS2 スレッドは SAS 暗号化で暗号化されます。THREAD ステートメントで ENCRYPT テーブルオプションを使用すると、このデフォルトを上書きして AES 暗号化を指定できます。SAS Proprietary は、SAS ソフトウェアに含まれている固定エンコーディングアルゴリズムです。詳細については、*SAS Viya プラットフォームの暗号化: 移動中のデータ*を参照してください。

テーブルオプションは、THREAD ステートメントで指定できます。パッケージ名の後に指定され、スラッシュが前に付きます。

比較

パッケージ、DS2 プログラム、スレッドの比較については、“ブロックステートメント” (1295 ページ)を参照してください。

例

例 1: 単純なスレッド

この例では、THREAD ステートメントを使用して単一のスレッドを作成します。

```
thread t;
  dcl int x;
  method init();
    x = 99;
    output;
  end;
endthread;
```

例 2: 複数のスレッドの実行

この例では、SET FROM ステートメントに THREADS オプションを追加することにより、スレッド T を変更して複数のスレッドを実行します。

```
thread work.t;
  dcl int x;
  dcl double y;
  method init();
    dcl int i;
    do i = 1 to 5;
      x = i;
      y = i * 2.5;
      output;
    end;
  end;
endthread;
data;
  dcl thread work.t t;
  method run();
    set from t threads=2;
    sum = x + y;
    put ' x= ' x ' y= ' y ' sum= ' sum;
  end;
enddata;
```

これにより、T に対して 2 つのスレッドが実行されます。次の行が SAS ログに書き込まれます。

```
x= 1 y= 2.5  sum= 3.5
x= 2 y= 5    sum= 7
x= 3 y= 7.5  sum= 10.5
x= 4 y= 10   sum= 14
x= 5 y= 12.5 sum= 17.5
x= 1 y= 2.5  sum= 3.5
x= 2 y= 5    sum= 7
x= 3 y= 7.5  sum= 10.5
x= 4 y= 10   sum= 14
x= 5 y= 12.5 sum= 17.5
```

この場合、出力はシーケンシャルですが、一貫して発生する保証はありません。

関連項目

- “スレッド処理” (*SAS DS2 プログラマガイド*)

メソッド:

- “SETPARMS メソッド” (1448 ページ)

ステートメント:

- “DECLARE THREAD ステートメント” (1335 ページ)
- “SET FROM ステートメント” (1422 ページ)

VARARRAY ステートメント

1 つ以上の DS2 変数配列を宣言します。

注: 構文規則の角かっこは、オプションの引数を示します。角かっこの前のエスケープ文字(\ )は、構文に角かっこが必要であることを示します。配列の境界は角かっこ([ ])で囲む必要があります。

構文

VARARRAY <data-type> *array-name* <array-declaration> [<variable-list>] [<having-clause>];

< data-type >::=

<exact-numeric-type> | <approximate-numeric-type> | <binary-string-type>
| <string-type>
| <date-type>

<exact-numeric-type> ::=

{**INT** | **BIGINT** | **SMALLINT** | **TINYINT**
| **DECIMAL** [(*precision* [,*scale*])] | **NUMERIC** [(*precision* [,*scale*])] }

<approximate-numeric-type> ::=

{ **DOUBLE** | **DOUBLE PRECISION** | **FLOAT** | **REAL** }

<binary-string-type>::=

BINARY(*length*) | **VARBINARY**(*length*)

<string-type>::=

NCHAR [(*character-length*)]

| **NVARCHAR** [(*character-length*)]

| **CHAR** [(*character-length*)] [**CHARACTER SET** *character-set-identifier*]

| **VARCHAR** [(*character-length*)] [**CHARACTER SET** *character-set-identifier*]

<date-type>::=

{ **TIME** | **TIMESTAMP** } [(*precision*)] | **DATE**

<array-declaration::=\[array-bound>[, ...<array-bound>]\

<array-bound>::= {[*dim-lower*:]*dim-upper*} | {[*dim-lower*:] {DIM(*a*[, *n*]) | *} }

<variable-list>::=

name-varlist |
| *numbered-range-varlist*
| *name-range-list*
| *name-prefix-list*
| *type-varlist*
| *special-name-list*

<having-clause>::=

HAVING <having-option> [... <having-option>]

<having-option>::=

LABEL '*string*' | n'*string*'

| **FORMAT** *format*

| **INFORMAT** *format*

必須引数

INT | BIGINT | SMALLINT | TINYINT
整数配列を指定します。

別名 INT の場合は INTEGER

参照項目: "DS2 データ型" (*SAS DS2 プログラマガイド*)

| DECIMAL[(*precision* [, *scale*])] | NUMERIC[(*precision* [, *scale*])]
正確な数値変数または配列を指定します。

precision
小数点の左側と右側の両方に保存できる最大合計桁を指定します

注 すべてのデータソースが 52 桁の精度をサポートできるわけではありません。

scale
小数点の右側に保存できる最大桁数を指定します

範囲 0–*precision*

注 *scale* は *precision* は以下です。

参照項目: "DS2 データ型" (*SAS DS2 プログラマガイド*)

DOUBLE | DOUBLE PRECISION | FLOAT | REAL
浮動小数点配列を指定します。

参照項目: "DS2 データ型" (*SAS DS2 プログラマガイド*)

BINARY (*length*)
バイナリ変数または配列を指定します。

要件 BINARY を指定する場合は、変数または配列の *length* もバイト単位で指定する必要があります。

参照項目: “DS2 データ型” (*SAS DS2 プログラマガイド*)

VARBINARY (*length*) | BINARY (*length*)
固定長または可変長のバイナリ配列を指定します。

別名 BINARY VARYING

参照項目: “DS2 データ型” (*SAS DS2 プログラマガイド*)

NCHAR | NVARCHAR | CHAR | VARCHAR
文字配列を指定します。

別名 NCHAR の場合は NATIONAL CHARACTER、NATIONAL CHAR

NVARCHAR の場合は NATIONAL CHARACTER VARYING、NATIONAL CHAR VARYING

CHAR の場合は CHARACTER

VARCHAR の場合は CHARACTER VARYING

参照項目: “DS2 データ型” (*SAS DS2 プログラマガイド*)

TIME
時間配列を指定します。

TIMESTAMP
日付と時間の両方の配列を指定します。

DIM(*a*[, *n*])
配列の上限のサイズが、DIM 関数呼び出しを使用して以前に宣言された配列の次元内の要素の数によって決定されることを指定します。

a
以前に宣言された配列の名前を指定します。

n
多次元配列の次元で、要素の数を知りたい次元を指定します。

ヒント *n* 値が指定されていない場合、DIM 関数は配列の最初の次元の要素の数を返します。

制限事項 DIM 関数は、配列の上限を指定するために使用できる唯一の関数です。DIM 関数は、次元の下限を指定するためには使用できません。

参照項目: “DIM 関数” (518 ページ)

*
下限が 1 で、上限が変数リストに含まれる変数の数である 1 次元配列を指定します。

要件 変数リストに少なくとも 1 つの *variable* を指定する必要があります。

オプション引数

character-length
文字列が NCHAR、NVARCHAR、CHAR、VARCHAR データ型に対して保持できる最大文字数を指定します。

デフォルト 8

要件 文字変数配列に含まれるすべての文字変数は、同じ長さである必要があります。

CHARACTER SET *character-set-identifier*
CHAR データ型と VARCHAR データ型の文字セットエンコーディング情報を指定します。

デフォルト デフォルトのエンコーディングは、オペレーティングシステムとロケールによって異なります。

要件 文字変数配列に含まれるすべての文字変数は、同じエンコーディングである必要があります。

ヒント 文字セット名には、文字列リテラルまたは単純な文字列を使用できます。たとえば、"ibm-866"また'ibm-866'を指定できます。

参照項目: 文字セットエンコーディング値の完全なリストについては、*SAS 各国語サポート(NLS): リファレンスガイド*の“SAS 言語要素中のエンコーディング値”を参照してください。

precision
TIME データ型または TIMESTAMP データ型の精度を指定します。

デフォルト 6

注 SAS 以外のデータソースで TIME 値と TIMESTAMP 値を操作していて、精度を指定しない場合、デフォルトの精度は常に DS2 のデフォルトの精度である 6 になります。

DATE
日付配列を指定します。

dim-lower and *dim-upper*
配列の境界の数とサイズを定義するために使用される正または負の整数を指定します。

ヒント 次元の下限が指定されていない場合、下限はデフォルトで 1 になります。

参照項目: “変数配列宣言” (*SAS DS2 プログラマガイド*)

<variable-list>
配列の要素によって参照される変数の名前を指定します。

要件　*variable* は、*data-type* で指定された型と同じである必要があります。

ヒント　1 つ以上の変数を指定できます。

参照項目:　“DS2 変数リスト” (*SAS DS2 プログラマガイド*)

LABEL | '*string*' | n'*string*'
変数配列に説明ラベルを割り当てます。ラベルは、CHAR リテラル(*string*)または NCHAR リテラル(n*string*)にすることができます。

参照項目:　“DS2 データ型” (*SAS DS2 プログラマガイド*)

FORMAT *format*
有効な DS2 出力形式を変数または配列に関連付けます。

参照項目:　“DS2 出力形式” (65 ページ)

INFORMAT *informat*
有効な SAS 入力形式を変数または配列に関連付けます。

参照項目:　8 章, “DS2 入力形式” (1285 ページ)

詳細

VARARRAY ステートメントを使用して、変数配列を作成します。変数配列は、グローバル変数の一時的なグループです。VARARRAY ステートメントで指定できる配列は 1 つのみです。

変数配列は、DS2 プログラムの期間中のみ存在します。

1 つの変数リスト指定の中に、異なる形式の変数リストを混在させることができます。たとえば、vararray double a[*] u x1–x3 u:;は有効なステートメントです。

前述の変数リストは、u x1 x2 x3 u u1 u2 に展開されます。したがって、7 要素の変数変配列が作成されます。単一の変数が、変数配列の複数の要素によって参照されています。

詳細については、“DS2 配列” (*SAS DS2 プログラマガイド*) と“変数配列” (*SAS DS2 プログラマガイド*)を参照してください。

一時配列を作成する方法については、“DECLARE ステートメント” (1313 ページ)を参照してください。

例

次の表には、変数配列とそれらの配列の次元を指定するステートメントの例が含まれています。

ステートメント	次元の数	次元の範囲	要素の数	参照変数
vararray double a[100];	1	1:100	100	a1...a100
vararray double a[2, 2];	2	1:2 1:2	4	a1 a2 a3 a4
vararray double a[-3:3, 5, 7:9, 10];	4	-3:3 1:5 7:9 1:10	7x5x3x10 = 1050	a1 ... a1050
vararray double a[3] x y z;	1	1:3	3	x y z
vararray double a[3] c3-c1;	1	1:3	3	c3 c2 c1
vararray double a[2, 2] t u v w;	2	1:2 1:2	4	t u v w
vararray double a[2, 2, 2] u v2-v4 w1-w3 x;	3	1:2 1:2 1:2	8	u v2 v3 v4 w1 w2 w3 x
vararray double a[*] x y z;	1	1:3	13	x y z
vararray double a[*] a1-a10;	1	01:10	10	a1...a10

関連項目

- “DS2 配列” (*SAS DS2 プログラマガイド*)
- “変数配列” (*SAS DS2 プログラマガイド*)

ステートメント:

- “DECLARE ステートメント” (1313 ページ)

VARLIST ステートメント

名前付き変数リストを作成します。

注: 構文規則の角かっこは、オプションの引数を示します。角かっこの前のエスケープ文字(\ )は、構文に角かっこが必要であることを示します。配列の境界は角かっこ([ ])で囲む必要があります。

構文

VARLIST *list-name* \[*variable-list*\];

必須引数

list-name
変数リストの名前を指定します。

[*variable-list*]
リストによって参照される変数を指定します。

要件 *variable-list* は角かっこ([])で囲む必要があります。

詳細

VARLIST ステートメントは、DS2 パッケージまたはプログラムのグローバルスコープに限定されていることに注意します。VARLIST ステートメントを使用してローカル変数リストを作成することはできません。

例

例 1: PDV 内のすべての 変数を含む変数リスト

この例では、VARLIST ステートメントは **allvars** という名前の変数リストを作成します。このリストには、DS2 プログラム内のすべての PDV 変数が含まれています。

```
varlist allvars [_all_];
```

例 2: 変数リストのデータ値の初期化

この例では、変数リスト c3 が宣言されてから、INIT メソッドで初期化されます。

```
data;
  declare int cost1 cost2 cost3 cost4 cost5;
  varlist c3 [cost1-cost5];

  method init();
    cost1 = 1;
    cost2 = 2;
    cost3 = 3;
    cost4 = 4;
    cost5 = 5;
  end;
enddata;
run;
```

例 3: VARLIST を使用してローカル変数リストを作成するとエラーが発生する

この例では、VARLIST ステートメントが原因で、次のエラーが表示されます: Invalid variable list. vars is not a global scalar variable. VARLIST ステートメントを使用してローカル変数リストを作成することはできません。

```
data _null_;
  declare double a;
  varlist vars [a];
  vararray double y[1] vars;  /* Error: varlist variable */
enddata;
run;
```

例 4: 変数リスト引数を受け入れる関数に変数リストを渡す

次の例では、変数リスト v を含むメソッド printNames を作成します。変数リスト v は、VNAME 関数と VTYPE 関数に渡されます。

```
data _null_;
  vararray double x[5];
  dcl bigint xyz;
  dcl date xanadu;

  method printNames(varlist v);
    dcl varchar(100) name type;
    dcl bigint i;
    do i = 1 to dim(v);
      name = vname(v[i]);
      type = vtype(v[i]);
      put name= type=;
    end;
  end;

  method init();
    printNames([x:]);
  end;
enddata;
run;
```

次の行が SAS ログに書き込まれます。

```
name=x1 type=double
name=x2 type=double
name=x3 type=double
name=x4 type=double
name=x5 type=double
name=xyz type=bigint
name=xanadu type=date
```

関連項目

“例: 名前付き varlist の作成” (*SAS DS2 プログラマガイド*)

11

DS2 システムメソッド

システムメソッドの概要 **1443**
ディクショナリ **1445**
INIT メソッド 1445
RUN メソッド 1446
SETPARMS メソッド 1448
TERM メソッド 1450

システムメソッドの概要

メソッドは基本的なプログラム実行ユニットです。メソッドはスコープブロックを定義するため、メソッド本体のパラメーターと変数宣言はメソッドに対してローカルです。DS2 では、すべてのプログラムコードがなんらかのメソッドに 存在する必要があります。

DS2 では、システムメソッドには事前設定された意味があります。システムメソッドには、INIT、RUN、および TERM の 3 つがあります。これらのメソッドはオーバーロードできません。スレッドでのみ使用されるオプションのシステムメソッドが 1 つあります。それは、SETPARMS です。

次の表は、各 DS2 システムメソッドの目的をリストにしてまとめたものです。

システムメソッド	実行の詳細	目的
INIT( )	プログラムの最初のメソッドとして、1 回自動的に実行されます。	名前が示すように、INIT()はグローバルプログラム変数を初期化するのに適した場所です。ほとんどのグローバル変数は、システムによって 初期化されません。ただし、システムは、_N や_N_などの事前定義された変数、および Sum ステートメントと RETAIN ステートメントで使用される変数を初期化します。

システムメソッド	実行の詳細	目的
		プログラムが他のシステムメソッドの機能を必要としない場合は、プログラム全体を INIT( )メソッドでコーディングできます。以下を含むがこれらに限定されない DS2 ステートメントを追加するだけです。 ■ メソッドスコープのローカル変数を作成するための DECLARE ステートメント ■ 1 つ以上のユーザー定義メソッドの呼び出し ■ 変数の割り当てを実行したり、DS2 関数を呼び出したり、ループやその他のロジックを実行したりする DS2 ステートメント 詳細については、“INIT メソッド” (1445 ページ)を参照してください。
RUN( )	INIT( )の完了後に自動的に実行されます。	RUN( )メソッドは、DATA ステップと機能的に同等です。つまり、RUN( )メソッドに SET ステートメントが含まれている場合、メソッドは暗黙的なループとして実行されます。RUN( )を使用して、SET FROM ステートメントでスレッドプログラムから行を読み取ることもできます。 **注:** 暗黙的なループ機能を活用するコードを含める必要はありません。 必要に応じて、INIT( )で説明されているように、プログラム全体を RUN( )メソッドでコーディングできます。 詳細については、“RUN メソッド” (1446 ページ)を参照してください。
TERM( )	プログラムの最後のメソッドとして、1 回自動的に実行されます。	名前が示すように、TERM( )は、プログラムが終了する前に最終処理が行われる場所です。 TERM( )は、次の例外を除いて、グローバル変数を初期化されていない値に自動的にリセットします。 ■ _N や_N_などの事前定義された変数 ■ Sum ステートメントで使用されたアキュムレータ変数 ■ RETAIN ステートメントで使用された変数 ■ パッケージ変数 必要に応じて、INIT( )で説明されているように、プログラム全体を TERM( )メソッドでコーディングできます。 詳細については、“TERM メソッド” (1450 ページ)を参照してください。
SETPARMS( )	データプログラムから呼び出されたときに 1 回実行されて、パラメーター化されたスレッド	SETPARMS( )は、パラメーター化されたスレッドの値を初期化します。パラメーター化されたスレッドプログラムのみがこれを必要とするため、SETPARMS( )は呼び出す必要がある唯一のシステムメソッドです。

システムメソッド	実行の詳細	目的
	の値を初期化します。	**注:** スレッドプログラムに SETPARMS( )メソッドを記述しないでください。システムがメソッドを提供します。 **注:** SETPARMS( )を複数回呼び出さないでください。初期化は初回のみ機能します。 詳細については、“SETPARMS メソッド” (1448 ページ)を参照してください。

DS2 でメソッドがどのように機能するかについての完全な情報については、“メソッド” (*SAS DS2 プログラマガイド*)を参照してください。

ディクショナリ

INIT メソッド

プログラムの初期化を実行できる DS2 システムメソッドを呼び出します。

構文

METHOD INIT( );
END;

引数なし

METHOD INIT ステートメントには、引数がありません。引数を渡そうとすると、エラーが発生します。

詳細

通常、INIT メソッドには、変数の初期化やテーブルを開く処理などの初期化コードが含まれています。INIT メソッドのコードは、DS2 プログラムの最初に一度だけ実行されます。

すべての DS2 プログラムには、暗黙的または明示的に、INIT、RUN、TERM のメソッドが含まれています。METHOD INIT ステートメントを指定しない場合、DS2 では自動的にこのステートメントを提供します。

INIT メソッドと DS2 プログラムの動作の詳細については、*“メソッド” (SAS DS2 プログラマガイド)*を参照してください。

例

```
method init();
  dcl int i;
  dcl double d;
  d = 99;
  do i = 1 to 3;
    d = d + i;
    output d;
  end;
end;
```

関連項目

- *“メソッド” (SAS DS2 プログラマガイド)*

メソッド:

- “RUN メソッド” (1446 ページ)
- “TERM メソッド” (1450 ページ)

RUN メソッド

DS2 プログラムコードを暗黙的なループで実行できる DS2 システムメソッドを呼び出します。

構文

```
METHOD RUN( );
END;
```

引数なし

METHOD RUN ステートメントには、引数がありません。引数を渡そうとすると、エラーが発生します。

詳細

通常、RUN メソッドにはメインの DS2 プログラムコードが含まれます。RUN メソッドには、SAS DATA ステップと同じように、自動の暗黙的なループ機能があります。RUN メソッドが 1 回実行された後、RUN メソッドがもう一度実行されるか、制御が TERM メソッドに渡されます。

すべての DS2 プログラムには、暗黙的または明示的に、INIT、RUN、TERM のメソッドが含まれます。METHOD INIT ステートメントや METHOD TERM ステートメントを指定しない場合、DS2 では自動的にこのステートメントを提供します。

INIT メソッドが実行されてから RUN メソッドが実行されるまでに、(RETAIN ステートメントを使用して)保持されていないプログラムデータベクトルの変数は、SAS モードか ANSI モードかによって、SAS 欠損値か null 値のいずれかに設定されます。

RUN メソッドのローカル変数は、暗黙的なループでの RUN の呼び出しと呼び出しの間は完全に存在しなくなります。RUN メソッドを呼び出すたびに、すべてのローカル変数がメソッドの実行の開始時に作成され、メソッドの実行の終了時に破棄されます。SET ステートメントによって読み取られる列変数を除くすべてのグローバル変数は、暗黙的なループの各反復(RUN メソッドの呼び出し)の間は欠損または null に設定されます。暗黙的なループで状態データを保持するには、グローバル変数を作成し、RETAIN ステートメントを使用して RUN メソッドの実行中は常にこの変数の値を保持するように指定する必要があります。RUN メソッドは、*x* 行のテーブルに対して、*x*+1 回実行されます。SET ステートメントが実行され、それ以上行が見つからない場合、RUN メソッドの暗黙的なループは停止します。

SAS モードと ANSI モードの詳細については、“DS2 による null および SAS 欠損値の処理方法” (*SAS DS2 プログラマガイド*)を参照してください。

RUN メソッドと DS2 プログラムの動作の詳細については、“メソッド” (*SAS DS2 プログラマガイド*)を参照してください。

比較

DATA ステッププログラミングでは、DATA ステップ全体が暗黙的なループを表します。DS2 言語では、暗黙的なループは RUN メソッドで表されます。

例

DS2 の実行フローでは、INIT メソッドを 1 回呼び出し、次に入力テーブルが完全に読み取られるまで RUN メソッドを呼び出し、次に TERM メソッドを呼び出します。RUN メソッドは、暗黙的なループが存在する場所です。テーブル内の最小値を検索する次のプログラムを使用して、この制御フローを説明します。INIT メソッドが現在の最小値を見つけるために使用される列を初期化し、RUN メソッドが入力値を現在の最小値と比較し、TERM メソッドが最小値を出力テーブルに書き込みます。

```
data xy_data;
  dcl double x y;
  method init();
    do x = 1 to 5;
```

```
      y = 2*x;
      output;
    end;
  end;
enddata;
run;
/* Find the minimum value for x and y */
data xy_mins;
  dcl double min_x min_y;
  retain min_x min_y;
  keep min_x min_y;
  method init();
    min_x = 999999;
    min_y = 999999;
  end;
  method run();
    set xy_data;
    if x < min_x then min_x = x;
    if y < min_y then min_y = y;
  end;
  method term();
    output;
  end;
enddata;
run;
/* Send result table of minimums to output */
data;
   method run();
     set xy_mins;
   end;
enddata;
run;
```

関連項目

- *“メソッド” (SAS DS2 プログラマガイド)*

メソッド:

- “INIT メソッド” (1445 ページ)
- “TERM メソッド” (1450 ページ)

SETPARMS メソッド

DS2 スレッドのインスタンスのパラメーターを初期化します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

構文

thread.**SETPARMS**(*parameter-value*[, ...*parameter-value*]);

必須引数

thread
スレッドのインスタンスを指定します。

parameter-value
スレッドパラメーターの初期値を指定します。

詳細

パラメーター化されたスレッドを使用している場合、パラメーター名とその種類は THREAD ステートメントで指定されます。スレッドを呼び出す DS2 プログラムは、SETPARMS メソッドを呼び出してスレッドのパラメーターを初期化する必要があります。この例では、INV と PROD の 2 つのパラメーターを受け取るスレッド T のインスタンスがあるとします。

```
thread work.t (double inv, char (30) prod);
```

SETPARMS メソッドを使用して、パラメーター INV は値 *38824* で初期化され、パラメーター PROD は値*'rice'*で初期化されます。

```
 t.setparms(38824, 'rice');
```

各引数は、値によって対応するスレッドパラメーターに渡されます。すべての引数は、必要に応じて、対応するパラメーターのデータ型に変換されます。パラメーターのないスレッドに対して SETPARMS メソッドが呼び出されると、エラーが発生します。

スレッドの SET FROM ステートメントを実行する前に、SETPARMS メソッドを呼び出してスレッドのパラメーターを初期化するか、SAS モードまたは ANSI モードのどちらであるかに応じて、パラメーターを SAS 欠損値または null 値で初期化する必要があります。詳細については、“DS2 による null および SAS 欠損値の処理方法” (*SAS DS2 プログラマガイド*)を参照してください。

例

この例は、パラメーターを使用してスレッドを使用する方法を示しています。

```
thread work.t (double d, char (100) sp);
  dcl int x;
  dcl double y;
  dcl nchar(20) s;
  dcl char(30) c;
  method init();
    dcl int i;
    s = 'abc' || sp;
    c = 'uvwxyz' || sp;
```

```
      do i = 1 to 100;
        x = i;
        y = i * 2.5 + d;
        output;
      end;
    end;
endthread;
run;
data;
  dcl thread work.t t;
  method init();
    t.setparms(99, 'ijk');
  end;
  method run();
    set from t;
    answer = x + y;
    put 's= ' s ' x= ' x ' y= ' y ' c= ' c ' answer= ' answer;
  end;
enddata;
run;
```

SAS ログに書き込まれる行の部分的なリストを次に示します。

```
s= abcijk    x= 1  y= 101.5  c= uvwxyzijk  answer= 102.5
s= abcijk    x= 2  y= 104    c= uvwxyzijk  answer= 106
s= abcijk    x= 3  y= 106.5  c= uvwxyzijk  answer= 109.5
s= abcijk    x= 4  y= 109    c= uvwxyzijk  answer= 113
s= abcijk    x= 5  y= 111.5  c= uvwxyzijk  answer= 116.5
```

関連項目

ステートメント:

- “SET FROM ステートメント” (1422 ページ)
- “THREAD ステートメント” (1429 ページ)

TERM メソッド

プログラムのファイナライズを実行できる DS2 システムメソッドを呼び出します。

構文

METHOD TERM ( );
END;

引数なし

METHOD TERM ステートメントには、引数がありません。引数を渡そうとすると、エラーが発生します。

詳細

通常、TERM メソッドには、SAS ログへのデータの書き込みなどのファイナライズコードが含まれます。TERM メソッドのコードは、DS2 プログラムの最後に一度だけ実行されます。

すべての DS2 プログラムには、暗黙的または明示的に、INIT、RUN、TERM のメソッドが含まれています。METHOD TERM ステートメントを指定しない場合、DS2 では自動的にこのステートメントを提供します。

TERM メソッドと DS2 プログラムの動作の詳細については、*“メソッド” (SAS DS2 プログラマガイド)*を参照してください。

関連項目

- *“メソッド” (SAS DS2 プログラマガイド)*

メソッド:

- “INIT メソッド” (1445 ページ)
- “RUN メソッド” (1446 ページ)

12

DS2 システムオプション

システムオプションの概要 **1453**
ディクショナリ **1454**
DS2SCOND= システムオプション 1454

システムオプションの概要

システムオプションは、オプションが指定されてから変更されるまで、SAS プログラム全体または対話型 SAS セッションの処理に影響を与える命令です。

OPTIONS ステートメントでシステムオプションを指定するための構文は次のとおりです。

OPTIONS *options(s)*;

構文の説明は次のとおりです。

option
変更する 1 つ以上の DS2 システムオプションを指定します。

次の例は、OPTIONS ステートメントでシステムオプション DS2SCOND を使用する方法を示しています。

```
options ds2scond=none;
```

注: DS2 は、ほとんどの SAS システムオプションをサポートしていません。

ディクショナリ

DS2SCOND= システムオプション

DS2 変数宣言の strict モードで、PROC DS2 が SAS ログに表示するメッセージのレベルを指定します。strict モードでは、すべての変数を DS2 プログラムで宣言する必要があります。

該当要素: 構成ファイル、SAS 起動、OPTIONS ステートメント、**SAS システムオプション**ウィンドウ

PROC OPTIONS GROUP= ErrorHandling

注: このオプションは、サイト管理者が制限できます。詳細については、*"制限されたオプション" (SAS システムオプション: リファレンス)*を参照してください。

構文

DS2SCOND=ERROR | NONE | NOTE | WARNING

必須引数

ERROR
エラーメッセージを SAS ログに書き込みます。

別名 ERR

NONE
SAS ログにメッセージは書き込まれません。

NOTE
SAS ログにメモを書き込みます。

WARNING
警告メッセージを SAS ログに書き込みます。これがデフォルトです。

別名 WARN

詳細

PROC DS2 ステートメントで SCOND=オプションを指定することにより、DS2SCOND システムオプションをオーバーライドできます。

関連項目

- “変数宣言” (*SAS DS2 プログラマガイド*)

プロシジャ:

- “DS2 プロシジャ” (*Base SAS プロシジャガイド*)

13

DS2 テーブルオプション

テーブルオプションの概要 **1458**
DS2 でのテーブルオプションの使用 **1458**
DS2 でテーブルオプションを指定する方法 **1460**
DS2 テーブルオプションの例 **1460**
DS2 では無効な SAS データセットオプション **1461**
データソース別の DS2 ステートメントテーブルオプション **1461**
ディクショナリ **1469**
ALTER= テーブルオプション 1469
ASYNCINDEX= テーブルオプション 1470
BL_DELETE_DATAFILE= テーブルオプション 1471
BUFNO= テーブルオプション 1472
BUFSIZE= テーブルオプション 1473
BULKLOAD= テーブルオプション 1474
COMPRESS= テーブルオプション 1477
DBCREATE_INDEX_OPTS= テーブルオプション 1479
DBCREATE_TABLE_OPTS= テーブルオプション 1480
DROP= テーブルオプション 1483
ENCRYPT= テーブルオプション 1484
ENCRYPTKEY= テーブルオプション 1487
ENDOBS= テーブルオプション 1490
EXTENDOBSCOUNTER= テーブルオプション 1491
FETCH_NUMERIC_TYPE= テーブルオプション 1492
GP_DISTRIBUTED_BY= テーブルオプション 1493
IDXNAME= テーブルオプション 1494
IDXWHERE= テーブルオプション 1495
IN= テーブルオプション 1497
INLINE テーブルオプション 1499
IOBLOCKSIZE= テーブルオプション 1501
KEEP= テーブルオプション 1502
LABEL= テーブルオプション 1504
LOCKTABLE= テーブルオプション 1505
ORHINTS= テーブルオプション 1506
ORNUMERIC= テーブルオプション 1507
OVERWRITE= テーブルオプション 1508

PADCOMPRESS= テーブルオプション 1510
PARTSIZE= テーブルオプション 1511
POINTOBS= テーブルオプション 1512
POST_TABLE_OPTS= テーブルオプション 1513
PRE_TABLE_OPTS= テーブルオプション 1515
PW= テーブルオプション 1516
READ= テーブルオプション 1517
READMODE= テーブルオプション 1518
RENAME= テーブルオプション 1519
REUSE= テーブルオプション 1521
STARTOBS= テーブルオプション 1522
SCANSTRINGCOLUMNS= テーブルオプション 1523
TABLE_TYPE= テーブルオプション 1524
THREADNUM= テーブルオプション 1525
TYPE= テーブルオプション 1526
UNIQUESAVE= テーブルオプション 1527
USER= テーブルオプション 1528
WHERENOINDEX= テーブルオプション 1529
WRITE= テーブルオプション 1529

テーブルオプションの概要

テーブルオプションは、DATA ステッププログラミングのデータセットオプションに類似しています。

テーブルオプションは、一緒に記述されるテーブルにのみ適用されるアクションを指定します。これらは、テーブルオプションで実行できる操作の一部です。

- 変数の名前を変更する
- パスワードを指定する
- データのバルクロードのオプションを指定する
- 処理または結果テーブルから変数を削除する

注: 一部のテーブルオプションは、すべてのデータソースに適用されます。その他はデータソース固有です。DS2 によって認識されないテーブルオプションは、基になるテーブルドライバーにエラーなしで渡されます。

DS2 でのテーブルオプションの使用

テーブルオプションは、次の DS2 ステートメントで使用できます。

- DECLARE PACKAGE

- DECLARE THREAD
- DROP PACKAGE
- DROP THREAD
- PACKAGE
- SET
- DATA
- THREAD

一部のテーブルオプションは、パッケージとスレッドに適用できます。

ほとんどのテーブルオプションは、入力テーブルまたは出力テーブルのいずれかに適用できます。テーブルオプションが入力テーブルに関連付けられている場合、アクションは読み取られているテーブルに適用されます。オプションが DATA ステートメントに記述される場合、SAS はアクションを出力テーブルに適用します。DS2 では、出力テーブルのテーブルオプションは、存在する可能性のある OUTPUT ステートメントではなく、DATA ステートメントに記述する必要があります。

COMPRESS=などの一部のテーブルオプションは、データセットの期間中に存在する属性を設定するため、SAS データセットを作成する場合にのみ意味があります。ほとんどのテーブルオプションを変更またはキャンセルするには、テーブルを再作成する必要があります。

テーブルオプションが同じ DS2 プログラムの入力テーブルと出力テーブルの両方に記述される場合、まず SAS はテーブルオプションを入力テーブルに適用します。次に、SAS はプログラミングステートメントを評価するか、テーブルオプションを出力テーブルに適用します。同様に、作成中のテーブルに指定されたテーブルオプションは、プログラミングステートメントが処理された後に適用されます。たとえば、RENAME=テーブルオプションを使用する場合、DS2 プログラムがコンパイルされるまで、新しい名前は列に関連付けられません。

場合によっては、テーブルオプションが同じステートメントで使用されると、競合します。たとえば、同じステートメントで同じ変数に DROP=テーブルオプションと KEEP=テーブルオプションの両方を指定することはできません。タイミングも問題になる場合があります。たとえば、SET ステートメントで指定されたテーブルで KEEP=および RENAME=を使用している場合、KEEP=は元の列名を使用する必要があります。SAS は、テーブルが読み取られる前に KEEP=を処理します。RENAME=で指定された新しい名前は、SET ステートメントに続くプログラミングステートメントに適用されます。

テーブルオプションは、データを含むテーブルを読み書きするときにいつでも適用できます。したがって、パッケージとスレッドはテーブルに保存されるため、テーブルオプションは DS2 パッケージとスレッドで機能します。

DS2 でテーブルオプションを指定する方法

テーブルオプションは、使用されているステートメントに応じて、かっこで囲むか、スラッシュ(/)を前に付けます。テーブルオプションは、前にスラッシュが付いている場合は、ステートメントの最後に配置する必要があります。

これらのステートメントで使用される場合、テーブルオプションはかっこで囲まれます。

- DECLARE PACKAGE
- DECLARE THREAD
- DROP PACKAGE
- DROP THREAD
- SET
- DATA

テーブルオプションがかっこで囲まれ、オプション値がスペースで区切られた複数のアイテムである可能性がある場合、オプション値もかっこで囲まれます。例については、“DS2 テーブルオプションの例” (1460 ページ)を参照してください。

次のステートメントで使用する場合、テーブルオプションの前にはスラッシュ(/)が付きます。

- PACKAGE
- THREAD

DS2 テーブルオプションの例

かっこで囲まれたテーブルオプションの例を次に示します。

```
data a (bufno=10);

data prod (drop=(price sales));

declare package sales (pw=lk34890f);

drop thread complex (write=24klj);
```

スラッシュ(/)が前に付いているテーブルオプションの例を次に示します。

```
package invent /overwrite=yes;
```

```
thread work.t (double d, char (100) sp) /read=44kl7;
```

DS2 では無効な SAS データセットオプション

DS2 テーブルオプションは、SAS データセットオプションで使用できるのと同じ変数コントロール機能を提供します。ただし、次の SAS データセットオプションは DS2 では無効です。

CNTLLEV=	PWREQ=
DLDMGACTION=	REPEMPTY=
FILECLOSE=	REPLACE=
FIRSTOBS=	ROLE=
GENMAX=	SORTEDBY=
GENNUM=	SPILL=
INDEX=	TOBSNO=
MAXTABLEMEM=	USEDIRECTIO=
OBSBUF=	WHERE=
OBS=	WHEREUP=
OUTREP=	

DS2 では、埋め込み FedSQL ステートメントを使用して、WHERE 句の指定や、処理する連続行のセットの指定などのタスクを実行できます。詳細については、“SET ステートメント” (1416 ページ)を参照してください。

注: テーブルオプションは、FedSQL ステートメントでもサポートされています。FedSQL ステートメントで指定されるテーブルオプションは、FedSQL テーブルオプション構文を使用する必要があります。

データソース別の DS2 ステートメントテーブルオプション

次の表は、DS2 プログラムでサポートされているテーブルオプションを、それをサポートするデータソース別にリストにしたものです。オプションは、データソースごとにアルファベット順にリストされています。

表 13.1　サポートされているテーブルオプションのデータソース別リスト

データソース	言語要素	カテゴリ	説明
すべて	DROP=テーブルオプション	テーブルコントロール	入力テーブルの場合、指定された列を処理から除外します。出力テーブルの場合、指定された列がテーブルに書き込まれるのを除外します。
	IN=テーブルオプション	テーブルコントロール	テーブルが現在の行にデータを提供したかどうかを示す整数変数を作成します。
	INLINE テーブルオプション	テーブルコントロール	パッケージまたはスレッドのソースコードが再利用のためにテーブルに保存されず、データプログラムによってロードされた場合にのみ検証およびコンパイルされることを指定します。
	KEEP=テーブルオプション	テーブルコントロール	入力テーブルの場合、処理する列を指定します。出力テーブルの場合、テーブルに書き込む列を指定します。
	OVERWRITE=テーブルオプション	テーブルコントロール	テーブルの場合、置換出力テーブルに行が入力される前に出力テーブルを削除します。パッケージとスレッドの場合、同じ名前のパッケージまたはスレッドが存在する場合は、既存のパッケージまたはスレッドを削除します。
	RENAME=テーブルオプション	テーブルコントロール	列の名前を変更します。
Amazon Redshift	DBCREATE_TABLE_OPTS=テーブルオプション	テーブルコントロール	DBMS 固有の構文を DATA ステートメントに追加できるようにします。
DB2	DBCREATE_TABLE_OPTS=テーブルオプション	テーブルコントロール	DBMS 固有の構文を DATA ステートメントに追加できるようにします。
Google BigQuery	BULKLOAD=テーブルオプション	バルクロード	SAS が DBMS 機能を使用して DBMS テーブルにデータを挿入するかどうかを決定します。
	DBCREATE_TABLE_OPTS=テーブルオプション	テーブルコントロール	DBMS 固有のオプションを DATA ステートメントに追加できるようにします。

データソース	言語要素	カテゴリ	説明
	FETCH_NUMERIC_TYPE=テーブルオプション	テーブルコントロール	Google BigQuery からの NUMERIC データの処理方法を指定します。
	READMODE=テーブルオプション	テーブルアクセス	Google BigQuery から SAS にデータを移動するときに使用する方法を指定します。
	SCAN_COLUMN_STRINGS=テーブルオプション	テーブルアクセス	データベーステーブルまたはクエリの文字列の最大長を決定するかどうかを指定します。
Greenplum	DBCREATE_TABLE_OPTS=テーブルオプション	テーブルコントロール	DBMS 固有の構文を DATA ステートメントに追加できるようにします。
	GP_DISTRIBUTED_BY=テーブルオプション	テーブルコントロール	作成するテーブルの配布キーを指定します。
Hive	DBCREATE_TABLE_OPTS=テーブルオプション	テーブルコントロール	データベース固有のオプションを DATA ステートメントの後に配置できるようにします。
	POST_TABLE_OPTS=テーブルオプション	テーブルコントロール	DATA ステートメントのテーブル名の後にデータベース固有のオプションを配置できるようにします。
	PRE_TABLE_OPTS=テーブルオプション	テーブルコントロール	DATA ステートメントのテーブル名の前にデータベース固有のオプションを配置できるようにします。
MDS	BULKLOAD=テーブルオプション	バルクロード	SAS が DBMS 機能を使用して DBMS テーブルにデータを挿入するかどうかを決定します。
Microsoft SQL Server	BULKLOAD=テーブルオプション	バルクロード	SAS が DBMS 機能を使用して DBMS テーブルにデータを挿入するかどうかを決定します。
	DBCREATE_INDEX_OPTS=テーブルオプション	テーブルコントロール	DBMS 固有の構文を CREATE INDEX ステートメントに追加できるようにします。
	DBCREATE_TABLE_OPTS=テーブルオプション	テーブルコントロール	DBMS 固有の構文を DATA ステートメントに追加できるようにします。
MySQL	BULKLOAD=テーブルオプション	バルクロード	SAS が DBMS 機能を使用して DBMS テーブルにデータを挿入するかどうかを決定します。

データソース	言語要素	カテゴリ	説明
	DBCREATE_TABLE_OPTS=テーブルオプション	テーブルコントロール	DBMS 固有の構文を DATA ステートメントに追加できるようにします。
Netezza	DBCREATE_TABLE_OPTS=テーブルオプション	テーブルコントロール	追加の DBMS 固有の構文を DATA ステートメントに追加できるようにします。
ODBC	DBCREATE_INDEX_OPTS=テーブルオプション	インデックスコントロール	DBMS 固有の構文を CREATE INDEX ステートメントに追加できるようにします。
	DBCREATE_TABLE_OPTS=テーブルオプション	テーブルコントロール	DBMS 固有の構文を DATA ステートメントに追加できるようにします。
Oracle	DBCREATE_TABLE_OPTS=テーブルオプション	テーブルコントロール	DBMS 固有の構文を DATA ステートメントに追加できるようにします。
	ORHINTS=テーブルオプション	データコントロール	FedSQL から Oracle に渡す Oracle ヒントを指定します。
	ORNUMERIC=テーブルオプション	テーブルコントロール	Oracle NUMBER 列から読み取られた数値または Oracle NUMBER 列に挿入された数値の処理方法を指定します。
PostgreSQL	BULKLOAD=テーブルオプション	バルクロード	SAS が DBMS 機能を使用して DBMS テーブルにデータを挿入するかどうかを決定します。
SAP HANA	DBCREATE_TABLE_OPTS=テーブルオプション	テーブルコントロール	DBMS 固有の構文を DATA ステートメントに追加できるようにします。
	TABLE_TYPE=テーブルオプション	テーブルアクセス	SAP HANA でテーブルを作成するときに FedSQL が使用するテーブルストレージの種類を指定します。
SAS データセット	ALTER=テーブルオプション	テーブルコントロール	SAS データセットに ALTER パスワードを割り当て、ユーザーがファイルを置換または削除できないようにしたり、読み取りまたは書き込み保護されたファイルへのアクセスを有効にしたりします。
	BUFNO=テーブルオプション	テーブルコントロール	SAS データセットを処理するために割り当てられるバッファーの数を指定します。

データソース	言語要素	カテゴリ	説明
	BUFSIZE=テーブルオプション	テーブルコントロール	出力 SAS データセットの永久バッファーページのサイズを指定します。
	COMPRESS=テーブルオプション	テーブルコントロール	新しい出力データセットで行を圧縮する方法を指定します。
	ENCRYPT=テーブルオプション	テーブルコントロール	出力 SAS データセットを暗号化するかどうかを指定します。
	ENCRYPTKEY=テーブルオプション	テーブルコントロール	AES 暗号化のキー値を指定します。
	EXTENDOBSCOUNTER=テーブルオプション	テーブルコントロール	新しい出力 SAS データファイルの最大オブザベーション数を拡張するかどうかを指定します。
	IDXNAME=テーブルオプション	インデックス使用のユーザーコントロール	WHERE 句の条件に一致する特定のインデックスを使用するように SAS に指示します。
	IDXWHERE=テーブルオプション	インデックス使用のユーザーコントロール	SAS が WHERE 句の条件に一致するためにインデックス検索と順次検索のどちらを使用するかを指定します。
	LABEL=テーブルオプション	オブザベーションコントロール	SAS データセットのラベルを指定します。
	LOCKTABLE=テーブルオプション	テーブルコントロール	テーブルに共有ロックまたは排他ロックをかけます。
	POINTOBS=テーブルオプション	テーブルコントロール	SAS が圧縮データセットを作成し、そのオブザベーションがランダムアクセス可能か、順次アクセス可能かを指定します。
	PW=テーブルオプション	テーブルコントロール	READ、WRITE、および ALTER パスワードを SAS ファイルに割り当て、パスワードで保護されたファイルへのアクセスを有効にします。
	READ=テーブルオプション	テーブルコントロール	READ パスワードを SAS ファイルに割り当てて、ユーザーがパスワードを入力しない限り、ファイルを読み取れないようにします。

データソース	言語要素	カテゴリ	説明
	REUSE=テーブルオプション	テーブルコントロール	圧縮された SAS データセットの空き領域に新しい行を書き込むことができるかどうかを指定します。
	TYPE=テーブルオプション	テーブルコントロール	特別に構造化された SAS データセットのデータセットタイプを指定します。
	WRITE=テーブルオプション	テーブルコントロール	SAS ファイルに WRITE パスワードを割り当てて、ユーザーがファイルに書き込めないようにするか、書き込み保護されたファイルにアクセスできるようにします。
SingleStore	BULKLOAD=テーブルオプション	バルクロード	SAS が DBMS 機能を使用して DBMS テーブルにデータを挿入するかどうかを決定します。
	DBCREATE_TABLE_OPTS=テーブルオプション	テーブルコントロール	DBMS 固有の構文を DATA ステートメントに追加できるようにします。
SPD Engine データセット	ALTER=テーブルオプション	テーブルコントロール	SPD Engine データセットに ALTER パスワードを割り当て、ユーザーがファイルを置換または削除できないようにしたり、読み取りまたは書き込み保護されたファイルへのアクセスを有効にしたりします。
	ASYNCINDEX=テーブルオプション	インデックス使用のユーザーコントロール	SPD Engine データセットに複数のインデックスを作成するときに、インデックスを並行して作成するように指定します。
	COMPRESS=テーブルオプション	テーブルコントロール	作成時にディスク上の SPD Engine データセットを圧縮するように指定します。
	ENCRYPT=テーブルオプション	テーブルコントロール	出力 SPD Engine データセットを暗号化するかどうかを指定します。
	ENCRYPTKEY=テーブルオプション	テーブルコントロール	AES 暗号化のキー値を指定します。
	ENDOBS=テーブルオプション	オブザベーションコントロール	処理するオブザベーションのユーザー定義範囲内の終了オブザベーション番号を指定します。

データソース	言語要素	カテゴリ	説明
	IDXWHERE=テーブルオプション	インデックス使用のユーザーコントロール	SPD Engine で WHERE 式を処理するときにインデックスを使用するように指定します。
	IOBLOCKSIZE=テーブルオプション	テーブルコントロール	I/O 操作で使用されるオブザベーションブロックのサイズをバイト単位で指定します。
	LABEL=テーブルオプション	オブザベーションコントロール	SPD Engine データセットのラベルを指定します。
	PADCOMPRESS=テーブルオプション	テーブルコントロール	OUTPUT または UPDATE 用に開かれたデータセットの圧縮ブロックに追加するバイト数を指定します。
	PARTSIZE=テーブルオプション	テーブルコントロール	SPD Engine データセット内のデータコンポーネントパーティションのサイズを指定します。
	PW=テーブルオプション	テーブルコントロール	READ、WRITE、および ALTER パスワードを SAS ファイルに割り当て、パスワードで保護されたファイルへのアクセスを有効にします。
	READ=テーブルオプション	テーブルコントロール	READ パスワードを SAS ファイルに割り当てて、ユーザーがパスワードを入力しない限り、ファイルを読み取れないようにします。
	STARTOBS=テーブルオプション	オブザベーションコントロール	処理するオブザベーションのユーザー定義範囲内の開始オブザベーション番号を指定します。
	THREADNUM=テーブルオプション	テーブルコントロール	SPD Engine データセットを処理するために SPD Engin が起動できる I/O スレッドの最大数を指定します。
	TYPE=テーブルオプション	テーブルコントロール	特別に構造化された SPD Engine データセットのデータセットタイプを指定します。
	UNIQUESAVE=テーブルオプション	インデックス使用のユーザーコントロール	一意のインデックスを持つデータセットにオブザベーションを追加または挿入するときに、一意でないキー値を持つオブザベーション(拒否されたオブザベーション)を別のデータセットに保存するように指定します。

データソース	言語要素	カテゴリ	説明
	WHERENOINDEX=テーブルオプション	インデックス使用のユーザーコントロール	WHERE 式の評価を行うときに除外するインデックスのリストを指定します。
	WRITE=テーブルオプション	テーブルコントロール	SAS ファイルに WRITE パスワードを割り当てて、ユーザーがファイルに書き込めないようにするか、書き込み保護されたファイルにアクセスできるようにします。
SPD Server テーブル	COMPRESS=テーブルオプション	テーブルコントロール	作成時にディスク上の SPD Server テーブルを圧縮するように指定します。
	ENCRYPT=テーブルオプション	テーブルアクセス	出力 SPD Server テーブルを暗号化するかどうかを指定します。
	ENCRYPTKEY=テーブルオプション	テーブルアクセス	AES 暗号化のキーを指定します。
	ENDOBS=テーブルオプション	オブザベーションコントロール	処理するオブザベーションのユーザー定義範囲内の終了オブザベーション番号を指定します。
	IOBLOCKSIZE=テーブルオプション	テーブルコントロール	I/O 操作で使用される行ブロックのサイズをバイト単位で指定します。
	PARTSIZE=テーブルオプション	テーブルコントロール	SPD Server テーブル内のデータコンポーネントパーティションのサイズを指定します。
	PW=テーブルオプション	テーブルアクセス	SAS Proprietary 暗号化によって保護されている SPD Server テーブルにアクセスできるようにします。
	STARTOBS=テーブルオプション	オブザベーションコントロール	処理するオブザベーションのユーザー定義範囲内の開始オブザベーション番号を指定します。
Spark	DBCREATE_TABLE_OPTS=テーブルオプション	テーブルコントロール	データベース固有のオプションを DATA ステートメントの後に配置できるようにします。
	POST_TABLE_OPTS=テーブルオプション	テーブルコントロール	DATA ステートメントのテーブル名の後にデータベース固有のオプションを配置できるようにします。

データソース	言語要素	カテゴリ	説明
	PRE_TABLE_OPTS=テーブルオプション	テーブルコントロール	DATA ステートメントのテーブル名の前にデータベース固有のオプションを配置できるようにします。
Teradata	DBCREATE_TABLE_OPTS=テーブルオプション	テーブルコントロール	DBMS 固有の構文を DATA ステートメントに追加できるようにします。

ディクショナリ

ALTER= テーブルオプション

データセットに ALTER パスワードを割り当て、ユーザーがファイルを置換または削除できないようにしたり、読み取りまおよび書き込み保護されたファイルへのアクセスを有効にしたりします。

制限事項: このテーブルオプションは、CAS サーバーではサポートされていません。

データソース: SAS データセット、SPD Engine データセット

注: この操作の後にログをチェックして、パスワード値が表示されていないことを確認してください。詳細については、*"ログからパスワードと暗号化キーの値を消去する" (SAS プログラマガイド: エッセンシャル)*の"パスワード値と暗号化キー値の消去"を参照してください。

構文

ALTER==*alter-password*

必須引数

alter-password
有効な SAS 名である必要があります。

詳細

ALTER=オプションは、SAS データセットにのみ適用されます。このオプションを使用して、パスワードを割り当てたり、読み取り保護、書き込み保護、または変更保護されたファイルにアクセスしたりできます。ALTER パスワードで保護されている

データセットを置換すると、新しいデータセットには ALTER パスワードが継承されます。

コードが SAS ログに書き込まれると、パスワードは消去されます。次に例を示します。

```
set a(alter=XXXXXXX);
```

注: SAS パスワードは、SAS 以外の SAS ファイルへのアクセスを制御しません。SAS 外の SAS ファイルへのアクセスを制御するには、オペレーティングシステムが提供するユーティリティとファイルシステムのセキュリティ 制御を使用する必要があります。

ASYNCINDEX= テーブルオプション

SPD Engine データセットに複数のインデックスを作成するときに、インデックスを並行して作成するように指定します。

該当要素: CREATE INDEX ステートメント

カテゴリ: インデックス使用のユーザーコントロール

制限事項: このテーブルオプションは、CAS サーバーではサポートされていません。

データソース: SPD Engine データセット

構文

ASYNCINDEX= YES | NO

引数

YES
インデックスを並行して(非同期に)作成します。

NO
一度に 1 つのインデックスを(同期的に)作成します。これがデフォルト値です。

詳細

SPD Engine は、データセットの複数のインデックスを同時に作成できます。SPD Engine は、作成されたインデックスごとに 1 つのスレッドを生成し、スレッドを同時に処理します。インデックスを並行して作成する方が、一度に 1 つのインデックスを作成するよりもはるかに高速ですが、このオプションのデフォルトは NO です。並列作成には、追加のユーティリティワークスペースと 追加のメモリが必要ですが、使用できない場合があります。リソースが不足しているためにインデックスの作成に失敗した場合は、次のいずれかを実行できます。

- SAS システムオプションを MEMSIZE=0 に設定する
- SPDEUTILLOC=システムオプションを使用して、ユーティリティファイル領域のサイズを増やす

SPDEINDEXSORTSIZE=システムオプションを使用して、インデックスの並べ替えに使用されるメモリ領域を増やします。インデックスを並行して作成するように指定する場合は、SPDEUTILLOC=システムオプションを使用して十分な大きさの領域を指定します。

BL_DELETE_DATAFILE= テーブルオプション

バルクロード機能によって作成された中間ファイルを削除するかどうかを指定します。

カテゴリ: バルクロード

デフォルト: YES

制限事項: このテーブルオプションは、CAS サーバーではサポートされていません。

データソース: Snowflake

構文

BL_DELETE_DATAFILE= YES | NO

引数

YES
SAS/ACCESS エンジンが DBMS バルクロード機能用に作成するすべての(データ、コントロール、およびログ)ファイルを削除します。

NO
ファイルを保持します。

詳細

このオプションは“BULKOPTS= テーブルオプション” (*SAS FedSQL 言語リファレンス*)内で指定する必要があります。

関連項目

テーブルオプション:

- “BULKLOAD= テーブルオプション” (1474 ページ)

BUFNO= テーブルオプション

SAS データセットを処理するために割り当てられるバッファーの数を指定します。

カテゴリ: テーブルコントロール

制限事項: このテーブルオプションは、CAS サーバーではサポートされていません。

データソース: SAS データセット

構文

BUFNO= *n* | *n*K | *hex*X | MIN | MAX

引数

n | *n*K
バッファーの数を 1 (バイト)または 1,024 (KB)の倍数で指定します。たとえば、値 **8** は 8 個のバッファーを指定し、値 **1K** は 1024 個のバッファーを指定します。

要件 *K* は大文字でなければなりません。

*hex*X
バッファーの数を 16 進値で指定します。数値(0-9)で始まり、その後に X が続く値を指定する必要があります。たとえば、値 **2dX** は、バッファーの数を 45 バッファーに設定します。

要件 *X* は大文字でなければなりません。

MIN
バッファーの最小数を 0 に設定します。これにより、SAS は動作環境に最適な最小値を使用します。これがデフォルト値です。

MAX
バッファーの数を、動作環境で可能な最大数(最大の 4 バイトの符号付き整数まで)に設定します。最大の 4 バイトの符号付き整数は 2^{31}-1、つまり約 20 億です。

詳細

バッファー数は、データセットの永続的な属性ではありません。現在の操作に対してのみ有効です。

BUFNO=テーブルオプションは、入力、出力、または更新用に開かれた SAS データセットに適用されます。

バッファーの数を増やすと、特定の SAS データセットに必要な入出力(I/O)操作の数が制限されるため、実行時間が短縮されます。ただし、実行時間の向上は、メモリ消費量の増加という犠牲を伴います。

小さなデータセットの I/O 操作を減らし、実行時間を短縮するには、処理するデータのページごとに 1 つのバッファーを割り当てます。この手法は、処理中に同じオブザベーションを数回読み取る場合に最も効果的です。

BUFSIZE= テーブルオプション

出力 SAS データセットの永久バッファーページのサイズを指定します。

カテゴリ: テーブルコントロール

制限事項: このテーブルオプションは、CAS サーバーではサポートされていません。
出力データセットでのみ使用します。

データソース: SAS データセット

構文

BUFSIZE=*n* | *n*K | *n*M | *n*G | *hex*X | MIN | MAX

引数

n | *n*K | *n*M | *n*G
ページサイズを 1 (バイト)、1,024 (KB)、1,048,576 (MB)、1,073,741,824 (GB)のいずれかの倍数で指定します。たとえば、値 **8** はページサイズ 8 バイトを指定し、値 **4K** はページサイズ 4096 バイトを指定します。

要件 *K*、*M*、および *G* は大文字にする必要があります。

*hex*X
ページサイズを 16 進値で指定します。数値(0-9)で始まり、その後に X が続く値を指定する必要があります。たとえば、値 **2dX** は、ページサイズを 45 バイトに設定します。

要件 *X* は大文字でなければなりません。

MIN
バッファーの最小数を 0 に設定します。これにより、SAS は動作環境に最適な最小値を使用します。

MAX
ページサイズを、動作環境で可能な最大数(最大の 4 バイトの符号付き整数まで)に設定します。最大の 4 バイトの符号付き整数は 2^{31}-1、つまり約 20 億バイトです。

詳細

ページサイズは、1 回の I/O 操作で 1 つのバッファーに転送できるデータの量です。ページサイズは、データセットの永続的な属性であり、データセットの処理時に使用されます。

ページサイズを大きくすると、SAS が記憶媒体から読み書きする回数が減るため、実行時間が短縮されます。ただし、実行時間の向上は、メモリ消費量の増加という代償を伴います。

ページサイズを変更するには、データセットをコピーし、新しいページを指定するか、SAS のデフォルトを使用します。ページサイズを動作環境のデフォルト値にリセットするには、BUFSIZE=0 を指定します。

動作環境の情報: BUFSIZE=テーブルオプションのデフォルト値は、動作環境によって決まり、順次アクセスを最適化するように設定されています。直接(ランダム)アクセスのパフォーマンスを向上させるには、BUFSIZE=の値を変更する必要があります。

BULKLOAD= テーブルオプション

SAS が DBMS 機能を使用して DBMS テーブルにデータを挿入するかどうかを決定します。

カテゴリ:	バルクロード
デフォルト:	NO
制限事項:	このテーブルオプションは、CAS サーバーではサポートされていません。
要件	ターゲットデータソースに適切な SAS/ACCESS ソフトウェアを構成する必要があります。
データソース:	MDS、Microsoft SQL Server、MySQL、PostgreSQL、SingleStore
注:	Amazon Redshift および Microsoft SQL Server でのバルクロードのサポートは、2024.07 から開始されます。 MySQL および SingleStore でのバルクロードのサポートは、2024.10 から開始されます。 PostgreSQL でのバルクロードのサポートは、2024.12 から開始されます。
ヒント:	サードパーティの DBMS への DS2 バルクロードは、DS2 SQLSTMT パッケージでサブミットされた FedSQL ステートメントで使用できます。詳細については、*“SQLSTMT パッケージの使用” (SAS DS2 プログラマガイド)*を参照してください。
参照項目:	バルクロードテーブルオプションについて

構文

BULKLOAD= YES | NO

引数

YES
DBMS テーブルに行を挿入または追加するために、DBMS 固有のバルクロード機能を呼び出します。

NO
DBMS 固有のバルクロード機能を呼び出しません。

詳細

概要

バルクロードを使用するのが、DBMS テーブルに行を挿入する最も速い方法です。SAS/ACCESS 接続のバルクロードを有効にするには、BULKLOAD=YES を指定します。

LIBNAME ステートメントで BULKLOAD=YES が指定されている場合、一部の SAS/ACCESS エンジンでは、PROC DS2 のバルクロードが有効になります。これらのエンジンは、プロシジャ用に生成する接続文字列で内部接続オプションを渡します。これらのデータソースでは、BULKLOAD=テーブルオプションを使用して、LIBNAME ステートメントの設定をオーバーライドできます。

その他のデータソースでは、BULKLOAD=テーブルオプションがバルクロードを有効にする方法です。これらのデータソースのほとんどでは、BULKLOAD=テーブルオプションを YES に設定するだけでバルクロードが有効になります。

BULKLOAD=NO の場合、単純な複数行挿入 SQL スキームを使用してデータ行が挿入されます。

使用上の注意

Microsoft SQL Server

テーブルオプション BULKLOAD=YES は、SQL Server データベースのバルクロードを有効にします。このオプションは、基になる DataDirect ドライバーで属性 SQL_ATTR_BULK_LOAD_ENABLED=1 を設定します。Microsoft SQL Server へのバルクロードには、データソース固有のテーブルオプションはありません。

MDS

テーブルオプション BULKLOAD=YES は、MDS のバルクロードを有効にします。BULKLOAD=テーブルオプションが YES に設定されている場合、新しく挿入された行はすぐにコミットされ、既存のトランザクションで表示されるようになります。BULKLOAD=テーブルオプションが NO に設定されるか、設定されていない場合、新しく挿入された行は、既存のトランザクションがコミットまたはロールバックされるまで表示されません。MDS へのバルクロードには、データソース固有のテーブルオプションはありません。

MySQL

バルクロードは、MYSQL エンジンの LIBNAME ステートメントで、または BULKLOAD=テーブルオプションで YES を指定することによって有効にできます。MySQL へのバルクロードには、データソース固有のテーブルオプションはありません。

バルクロード用の中間ファイルは、UTILLOC=システムオプションで指定された場所に作成されます。

PostgreSQL

バルクロードは、POSTGRES エンジンの LIBNAME ステートメントで、または BULKLOAD=テーブルオプションで YES を指定することによって有効にできます。PostgreSQL でバルクロードを有効にするために必要なデータソース固有のテーブルオプションはありません。

SingleStore

バルクロードは、SSTORE エンジンの LIBNAME ステートメントで、または BULKLOAD=テーブルオプションで YES を指定することによって有効にできます。SingleStore へのバルクロードには、データソース固有のテーブルオプションはありません。

バルクロード用の中間ファイルは、UTILLOC=システムオプションで指定された場所に作成されます。

例:

```
data accesslib.myexample (BULKLOAD=YES);
  dcl float       f;
  method init();
    f=11.01;
    output;
  end;
enddata;
run;
```

関連項目

概念:

- “データソース別の DS2 ステートメントテーブルオプション” (1461 ページ)

テーブルオプション:

- “BULKOPTS= テーブルオプション” (*SAS FedSQL 言語リファレンス*)

COMPRESS= テーブルオプション

新しい出力データセットまたはテーブルで行を圧縮する方法を指定します。

カテゴリ: テーブルコントロール

制限事項: このテーブルオプションは、CAS サーバーではサポートされていません。
出力テーブルのみで使用します。

データソース: SAS データセット、SPD Engine データセット、SPD Server テーブル

構文

COMPRESS= NO | YES | CHAR | BINARY

引数

CHAR | YES
新しく作成されたテーブルの行が圧縮されることを指定します(可変長レコード)。SAS データセットは、ランレングスエンコーディング(RLE)を使用して圧縮されています。RLE は、繰り返される連続する文字(空白を含む)を 2 バイトまたは 3 バイトの表現に減らすことによって、行を圧縮します。SPD Engine データセットは、ランレングス圧縮アルゴリズム SPDSRLC2 を使用して圧縮されます。SPD Server テーブルは、ランレングス圧縮アルゴリズム SPDSRLLC を使用して圧縮されます。

別名 ON (SPD Engine データセットのみ)

ヒント この圧縮アルゴリズムを文字データに使用します。

BINARY
新しく作成されたデータセットまたはテーブルの行が、ロスデータ圧縮(RDC)を使用して SAS によって圧縮されることを指定します。RDC は、ランレングスエンコーディングとスライディングウィンドウ圧縮を組み合わせてファイルを圧縮します。SPD Engine データセットは、SPDSRDC を使用して SPD Engine によって圧縮されます。

ヒント この方法は、中規模から大規模(数百バイト以上)のバイナリデータ(数値変数)ブロックを圧縮する場合に非常に効果的です。圧縮機能は一度に 1 つのレコードに対して動作するため、効果的な圧縮にはレコード長が数百バイト以上である必要があります。

NO
新しく作成されたテーブルの行が非圧縮であることを指定します(固定長レコード)。

詳細

テーブルの圧縮は、各行を表すために必要なバイト数を減らす処理です。テーブルを圧縮する利点としては、テーブルのストレージ要件が削減され、処理中にデータの読み書きに必要な I/O 操作が少なくなることが挙げられます。ただし、圧縮されたテーブルを読み取るには、より多くの CPU リソースが必要になります(各行の圧縮を解除するオーバーヘッドのため)。また、結果のファイルサイズが減少せずに増加する場合もあります。

個々のテーブルを圧縮するには、COMPRESS=テーブルオプションを使用します。出力テーブルのみ、つまり CREATE TABLE ステートメントのテーブル名に対してオプションを指定します。

注: SPD Engine データセットと SPD Server テーブルでは、暗号化と圧縮は相互に排他的です。圧縮された SPD Engine データセットまたは SPD Server テーブルの暗号化を指定することはできません。暗号化された SPD Engine データセットまたは SPD Server テーブルは圧縮できません。

テーブルが圧縮されると、設定はテーブルの永続的な属性になります。つまり、設定を変更するには、テーブルを再作成する必要があります。つまり、テーブルを圧縮解除するには、DROP TABLE ステートメントを使用してテーブルを削除し、CREATE TABLE ステートメントと COMPRESS=NO オプションを使用してテーブルを再作成する必要があります。

SPD Engine データセットまたは SPD Server テーブルを圧縮するために COMPRESS=を指定すると、テーブルドライバーは、作成時にテーブルをブロックごとに圧縮します。圧縮ブロックのサイズを指定するには、“IOBLOCKSIZE= テーブルオプション” (1501 ページ)を使用します。

SPD Engine テーブルドライバーは、SPD Engine データセットを作成または更新するときに、PADCOMPRESS=テーブルオプションもサポートします。PADCOMPRESS=オプションを使用すると、新しく圧縮されたブロックにパディングを追加できます。“PADCOMPRESS= テーブルオプション” (1510 ページ)を参照してください。

比較

COMPRESS=テーブルオプションは、COMPRESS=接続文字列オプションをオーバーライドします。

SAS data sets only: 圧縮テーブルを作成する場合、スペースを追跡して再利用するために REUSE=YES テーブルオプションを指定することもできます。REUSE=YES を指定すると、他の行が更新または削除されたときに解放されたスペースに新しい行が挿入されます。デフォルトの REUSE=NO が有効な場合、新しい行が既存のテーブルに追加されます。

関連項目

テーブルオプション:

- “ENCRYPT= テーブルオプション” (1484 ページ)
- “POINTOBS= テーブルオプション” (1512 ページ)
- “IOBLOCKSIZE= テーブルオプション” (1501 ページ)
- “PADCOMPRESS= テーブルオプション” (1510 ページ)
- “REUSE= テーブルオプション” (1521 ページ)

DBCREATE_INDEX_OPTS= テーブルオプション

DBMS 固有のオプションを CREATE INDEX ステートメントに追加できるようにします。

カテゴリ: インデックスコントロール

制限事項: このテーブルオプションは、CAS サーバーではサポートされていません。

データソース: Microsoft SQL Server、ODBC

構文

DBCREATE_INDEX_OPTS= '*DBMS-SQL-clauses*'

引数

DBMS-SQL-clauses
SQL CREATE INDEX ステートメントの末尾に追加できる 1 つ以上の DBMS 固有の句を指定します。

詳細

このオプションを使用すると、DBMS 固有の句を CREATE INDEX ステートメントの末尾に追加できます。インターフェイスは CREATE INDEX ステートメントとその句を DBMS に渡し、DBMS はステートメントを実行して DBMS インデックスを作成します。

例

次の例では、DBCREATE_INDEX_OPTS= “as 'compact' with deferred rebuild”オプションの値を CREATE INDEX ステートメントに追加して、Hive インデックスが作成されます。

```
create index "COL1_1A03" on "TKTS002_1A03"{options DBCREATE_INDEX_OPTS="as 'compact'
with deferred rebuild"}("COL1")
```

インデックスを作成するために、次の CREATE INDEX ステートメントが DBMS に渡されます。

```
create index 'COL1_1A03' on table 'TKTS002_1A03'('COL1') as 'compact' with
deferred rebuild
```

DBCREATE_TABLE_OPTS= テーブルオプション

DATA ステートメントに追加する DBMS 固有のオプションをできるようにします。

制限事項: このテーブルオプションは、CAS サーバーではサポートされていません。

データソース: Amazon Redshift、UNIX および PC での DB2、Google BigQuery、Greenplum、Hive、Microsoft SQL Server、MySQL、Netezza、Oracle、SAP HANA、Spark、Teradata

構文

DBCREATE_TABLE_OPTS= '*DBMS-options*'

必須引数

DBMS-options
1 つ以上の有効な DBMS 固有のオプションを指定します。複数のオプションを指定する場合は、DBMS でオプションを区切るのと同じ方法でオプションを区切る必要があります。

詳細

基本

このオプションを使用すると、DBMS 固有のオプションを DATA ステートメントに追加できます。インターフェイスは DATA ステートメントとその句を DBMS に渡し、DBMS はステートメントを実行して DBMS テーブルを作成します。この例として、COLUMN-DELIMITER=オプションを使用して、Hadoop ファイルの列区切り文字を指定します。

注: SAS/ACCESS DBCREATE_TABLE_OPTS LIBNAME オプションと DS2 DBCREATE_TABLE_OPTS テーブルオプションが使用されている場合、DS2 テーブルオプションの設定が優先されます。

DBMS オプションの引用符

DBMS-options は、単一引用符で囲む必要があります。引用符で囲まれた *DBMS-options* 内の要素は、要素を二重引用符で囲む必要があります。たとえば、単一引用符を使用する場合は、2 つの単一引用符を使用します。二重引用符を使用する場合は、2 セットの二重引用符を使用します。次に例を示します。

```
/* Using double quotes around DBMS-option element causes an error */
DBCREATE_TABLE_OPTS="FIELDS TERMINATED BY '\012' "

/* Using single quotes around DBMS-option element works */
DBCREATE_TABLE_OPTS='FIELDS TERMINATED BY "\012" '

/* You can double the quote character to insert one into a quoted value*/

/* doubled single quote passes single quote to database */
/* trailing space after last set of single quotes added for emphasis */
DBCREATE_TABLE_OPTS='FIELDS TERMINATED BY ''\012'' '

/* doubled double quotes passes double quotes to database */
/* trailing space after last set of double quotes added for emphasis */
DBCREATE_TABLE_OPTS='FIELDS TERMINATED BY ""\012"" '
```

注: プログラムを Spark で実行する場合は、*DBMS-options* 値内で 2 つの単一引用符または二重引用符を使用しないでください。Spark が出力テーブルを作成するとき、引用符のセットの 1 つは削除されません。次に例を示します。

```
/* This value with doubled single quotes */
DBCREATE_TABLE_OPTS='FIELDS TERMINATED BY ''\012'' '
/* gets translated to this for any data source except Spark */
FIELDS TERMINATED BY '\012'
/* and gets translated like this for Spark jobs which causes an error */
FIELDS TERMINATED BY ''\012''
```

例

例 1

次の例では、Teradata テーブル Teralib は、DATA ステートメントに追加された primary index ()オプションの値を使用して作成されます。

```
libname teralib teradata server=terasvr database=model user=myid password=xxxx;

proc delete data=teralib.outtable;
proc ds2;
```

```
data teralib.outtable(overwrite=yes dbcreate_table_opts='primary index(i)');
  retain x 0;
  method run();
    do i=1 to 10 by 1 ;
      x=i;
      output;
    end;
  end;
enddata;
run;
quit;
```

例 2

次の例では、Hive テーブル **Hivelib.Dzoutput** が作成され、partition by オプションが DATA ステートメントに追加されます。この例では、**Col1** は SET ステートメントのテーブル **Hivelib.Dzpt** に存在しますが、これは必須ではありません。

```
options set=SAS_HADOOP_JAR_PATH="jar-path";
options set=SAS_HADOOP_CONFIG_PATH="config-path";
libname hivelib hadoop server="hostname" user=username password=xxxx
schema=ds2ip
dbmax_text=300;

proc ds2;
data hivelib.dzoutput (dbcreate_table_opts="partitioned by (col1 int)"
overwrite=yes);
  method run();
    set hivelib.dzpt;
      x+1;
      output; output;
  end;
enddata;
run;
quit;
```

例 3

DB2 テーブルを作成する場合、IBM Integrated Analytics System (IIAS)は、ORGANIZED BY COLUMN テーブルで更新可能なカーソルを開くことを許可しません。回避策は、次のようにテーブルを作成するときに、オプション DBCREATE_TABLE_OPTS="organize by row"を設定することです。

```
data test (DBCREATE_TABLE_OPTS="organize by row");
   dcl int x;
```

DROP= テーブルオプション

入力テーブルの場合、指定された列を処理から除外します。出力テーブルの場合、指定された列がテーブルに書き込まれるのを除外します。

データソース: すべて

構文

DROP= (*column-list*)

必須引数

column-list
出力テーブルから省略する列の名前を指定します。

制限事項 *col1–col5* 形式の番号付き範囲リストおよび *col:*形式の名前接頭辞リストはサポートされていません。

詳細

DROP=テーブルオプションは、*column-list* のすべての列が出力行の作成に含まれないように指定します。通常、プログラムデータベクトルのすべての列は出力行に含まれます。drop 属性が指定されている場合、drop ステートメントに含まれていないすべての列が、出力行に列を作成するために使用されます。

DROP=テーブルオプションが入力テーブルに関連付けられている場合、プログラムの実行中に列を処理することはできません。

比較

DROP=テーブルオプションは、次の点で DROP ステートメントとは異なります。

- DS2 プログラムでは、DROP=テーブルオプションを入力テーブルと出力テーブルの両方に適用できます。DROP ステートメントは、出力テーブルにのみ適用されます。
- DS2 プログラムで、複数の出力テーブルを作成する場合は、DROP=テーブルオプションを使用して、異なる列を異なるテーブルに書き込みます。DROP ステートメントは、すべての出力テーブルに適用されます。
- KEEP=テーブルオプションは、処理に含まれるか、または出力テーブルに書き込まれる列のリストを指定します。

例

例 1: 出力テーブルからの列の除外

この例では、変数 SALARY と GENDER は処理に含まれず、どちらの出力テーブルにも書き込まれません。

```
data plan1 plan2;
  method run ();
    set payroll(drop=(salary gender));
    if hired<'01jan07'd then output plan1;
      else output plan2;
  end;
enddata;
```

DROP=は SET ステートメントが PAYROLL からそれらを読み取ることを防ぐため、DS2 プログラムのどのロジックでも SALARY または GENDER を使用することはできません。

例 2: 出力テーブルに列を書き込まずに列を処理する

この例では、SALARY と GENDER は PLAN2 に書き込まれていませんが、PLAN1 に書き込まれています。

```
data plan1 plan2(drop=(salary gender));
  method run ();
    set payroll;
    if hired<'01jan07'd then output plan1;
      else output plan2;
  end;
enddata;
```

関連項目

ステートメント:

- “DROP ステートメント” (1344 ページ)

テーブルオプション:

- “KEEP= テーブルオプション” (1502 ページ)

ENCRYPT= テーブルオプション

出力 SAS データセットを暗号化するかどうかを指定します。

制限事項: 出力データセットでのみ使用します。
このテーブルオプションは、CAS サーバーではサポートされていません。

SPD Engine データセットでは、このテーブルオプションを COMPRESS=と一緒に使用することはできません。

データソース: SAS データセット、SPD Engine データセット、SPD Server テーブル

構文

ENCRYPT==AES | AES2 | NO | YES

必須引数

AES | AES2 | NO | YES
出力 SAS データセットを暗号化するかどうかを指定します。

AES または AES2
AES (高度暗号化標準)アルゴリズムを使用してファイルを暗号化します。AES は強化された暗号化を提供します。AES2 は、より強力なキー生成アルゴリズムを使用することを指定しています。ENCRYPT=AES または ENCRYPT=AES2 を使用する場合は、ENCRYPTKEY=テーブルオプションを指定する必要があります。詳細については、“ENCRYPTKEY= テーブルオプション” (1487 ページ)を参照してください。

制限事項 SPD Engine と SPD Server は AES2 暗号化をサポートしていません。

注意 **ENCRYPT=AES または ENCRYPT=AES2 を使用している場合は、すべての ENCRYPTKEY=値を記録してください。** ENCRYPTKEY=値を記録するのを忘れると、データが失われます。SAS は、ENCRYPTKEY=値の回復を支援することはできません。次の注がログに書き込まれます。

```
Note: If you lose or forget the ENCRYPTKEY= value, there will
be no way to open the file or recover the data.
```

NO
データセットを暗号化しません。

YES
SAS Proprietary アルゴリズムを使用してデータセットを暗号化します。この暗号化では、データセットに保存されているパスワードを使用します。
ENCRYPT=YES を指定すると同時に、READ=テーブルオプションまたは PW=テーブルオプションを指定する必要があります。詳細については、“READ= テーブルオプション” (1517 ページ)および“PW= テーブルオプション” (1516 ページ)を参照してください。

操作 暗号化法ではパスワードを使用するため、テーブルを再作成しないと暗号化ファイルの*いずれの*パスワードも変更することはできません。

注意 **ENCRYPT=YES を使用している場合は、すべてのパスワードを記録してください。** パスワードを忘れた場合、SAS の支援なしにパスワードをリセットすることはできません。これは、時間とリソースを大量に消費するプロセスです。

詳細

ENCRYPT=YES を使用する場合、次のルールが適用されます。

- データファイルが暗号化されている場合、関連するすべてのインデックスも 暗号化されます(SPD Server テーブルを除く)。SPD Server はテーブルインデックスまたはメタデータを暗号化しません。テーブル行データのみが暗号化されます。
- 暗号化されたデータファイルをコピーするには、出力エンジンが暗号化をサポートしている必要があります。それ以外の場合、データファイルはコピーされません。
- 暗号化には、圧縮とほぼ同じ量の CPU リソースが必要です。
- SAS Proprietary の暗号化された SAS ファイルで PROC CPORT を使用することはできません。

ENCRYPT=AES または ENCRYPT=AES2 を使用する場合、次のルールが適用されます。

- テーブルを暗号化するときは、ENCRYPTKEY=テーブルオプションを使用する必要があります。
- 復号化を有効にするには、ENCRYPTKEY=テーブルオプションを使用する必要があります。
- AES 暗号化で暗号化された SAS データセットデータセットは、AES 暗号化をサポートするエンジンによってのみ復号化できます。
- データファイルを再作成せずに、AES 暗号化データファイルの ENCRYPTKEY=値を変更することはできません。
- 参照一貫性制約のあるデータファイルは、AES 暗号化を使用できます。すべての主キーと外部キーのデータファイルは、すべての参照外部キーと主キーのデータファイルを開くのと同じ暗号化キーを使用する必要があります。

AES2 暗号化を使用した SAS データセットをコピーして新しい SPD Engine データセットまたは SPD Server テーブルを作成すると、暗号化は AES に変換されます。警告がログに書き込まれます。

注: PACKAGE および THREAD ステートメントで ENCRYPT 引数を使用して、暗号化された DS2 パッケージまたはスレッドプログラムを作成できます。

SAS が提供する暗号化の詳細については、*SAS Viya プラットフォームの暗号化: 移動中のデータ*を参照してください。

例

この例では、暗号化された SAS データセットを作成します。

```
proc ds2;
 data myfiles.mytable (encrypt=yes pw=mine);
    declare double j j2;
    method run();
      do j = 1 to 1000;
```

```
        j2 = 2*j;
        output;
      end;
    end;
 enddata;
run;
quit;
```

この例では、暗号化された SAS データセットを読み取ります。

```
proc print data=myfiles.mytable (pw=mine obs=10);
run;
```

関連項目

ステートメント:

- “PACKAGE ステートメント” (1392 ページ)
- “THREAD ステートメント” (1429 ページ)

テーブルオプション:

- “ENCRYPTKEY= テーブルオプション” (1487 ページ)

ENCRYPTKEY= テーブルオプション

AES 暗号化のキー値を指定します。

制限事項: AES および AES2 暗号化データファイルでのみ使用してください。
このテーブルオプションは、CAS サーバーではサポートされていません。

データソース: SAS データセット、SPD Engine データセット、SPD Server テーブル

注: この操作の後にログをチェックして、暗号化キー値が表示されていないことを確認してください。詳細については、*SAS 言語リファレンス: 概念*の"パスワード値と暗号化キー値の消去"を参照してください。

構文

ENCRYPTKEY=*key-value*

必須引数

key-value
暗号化キー値を割り当てます。ENCRYPT=AES または ENCRYPT=AES2 を使用する場合は、ENCRYPTKEY=テーブルオプションを指定する必要があります。キー値の長さは最大 64 バイトです。

要件 ENCRYPTKEY=キー値は、引用符付きまたは引用符なしで作成できます。詳細については、“ENCRYPTKEY 値を作成するときに引用符を使用するためのルール” (1489 ページ)を参照してください。

注 ENCRYPTKEY=キー値が DBCS 文字を使用する場合、64 バイトの制限は、UTF-8 エンコーディングにトランスコードされた後の文字列に適用されます。次の DATA ステップを使用して、DBCS のキー値の長さをバイト単位で計算できます。

```
data _null_;
   key=length(unicodec('key-value','UTF8'));
   put 'key length=' key;
run;
```

詳細

基本

注意
キー値を記録します。 ENCRYPTKEY=キー値を忘れると、データが失われます。キー値はデータセットに保存されていないため、SAS は ENCRYPTKEY=キー値の回復を支援できません。次の警告がログに書き込まれます。

```
WARNING: If you lose or forget the ENCRYPTKEY= value, there
will be not be any way to open the file or recover the data.
```

ENCRYPTKEY=オプションは、ファイルのコンテンツへのアクセスのみを防止します。ENCRYPTKEY=テーブルオプションは、テーブルを削除または置換から保護しません。暗号化されたデータセットは、ENCRYPTKEY=キー値を指定せずに、次のシナリオのいずれかを使用して削除できます。

- PROC DATASETS の KILL オプション
- PROC SQL の DROP ステートメント
- PROC DATASETS の DELETE ステートメントまたは DELETE プロシジャ

ファイルを削除または置換から保護するには、ファイルに ALTER=パスワードも含まれている必要があります。

AES で暗号化されたデータファイル内の ENCRYPTKEY=値を変更することはできません。暗号化キーの値を変更するには、データセットを再作成する必要があります。OVERWRITE=テーブルオプションは、暗号化キーで暗号化されたデータセットを再作成するために使用することはできません。PROC FEDSQL を使用してテーブルを削除することはできません。DS2 を使用してデータセットを再作成する前に、PROC DELETE または PROC SQL DROP TABLE ステートメントを使用してデータセットを削除します。

ENCRYPTKEY=キー値としてマクロ変数を使用することができます。ENCRYPTKEY=キー値にマクロ変数を指定する場合は、マクロ変数を二重引用符で囲む必要があります。二重引用符を使用しないと、予期しない結果が発生する可能性があります。次の例では、マクロ変数を定義し、マクロ変数を ENCRYPTKEY=キー値として使用します。

```
%let secret=myvalue;
data my.dsname(encrypt=aes encryptkey="&secret");
```

注: DS2 が SAS の外部(SAS Federation Server グリッドコンピューティング環境など)で実行されている場合、SAS マクロ機能は使用できず、マクロを含む DS2 プログラムはコンパイルに失敗します。

ENCRYPTKEY 値を作成するときに引用符を使用するためのルール

暗号化キーの値が引用符なしで作成される場合、値は文字で始まる必要があり、英数字とアンダースコアのみを含めることができます。キー値に空白を含めることはできません。文字は大文字、小文字、または大文字と小文字の混合にすることができます。許可される例を次に示します。

```
encryptkey=G_rEeN1
```

キー値の評価では大文字と小文字は区別されません。次のキー値を使用して、前述のキーで作成されたデータセットを読み取ることができます。

```
encryptkey=G_REEN1
```

単一引用符で囲んで指定する暗号化キー値には、英数字、特殊文字、DBCS 文字を含めることができます。空白は許可されますが、キー値をすべて空白で構成することはできません。文字列の評価では大文字と小文字が区別されます。有効なキー値の例を次に示します。データセットを読み取るには、データセットの暗号化に使用したのとまったく同じ値を指定する必要があります。

```
encryptkey='1Gr*#E7e.N 1'
```

二重引用符で囲んで指定される暗号化キー値には、単一引用符で囲んで指定されるキー値と同じ要件があります。さらに、暗号化キー値をマクロ解決に使用できます。

例

この例では、ENCRYPT=AES オプションを使用しています。

```
proc ds2;
data salary (encrypt=aes encryptkey='1Gr*#E7e.N 1');
  dcl char(8) name;
  dcl double yrsal;
  dcl double bonuspct;
  method init();
    name='Muriel'; yrsal=34567; bonuspct=3.2;
      name='Bjorn'; yrsal=74644; bonuspct=2.5;
    name='Freda'; yrsal=38755; bonuspct=4.1;
      name='Benny'; yrsal=29855; bonuspct=3.5;
      name='Agnetha'; yrsal=70998; bonuspct=4.1;
  end;
run;
quit;
```

CONTENTS プロシジャを実行するには、次のように暗号化キーの値を指定します。

```
proc contents data=salary (encryptkey='1Gr*#E7e.N 1');
```

```
run;
```

関連項目

テーブルオプション:

- “ENCRYPT= テーブルオプション” (1484 ページ)
- “暗号化” (*SAS プログラマガイド: エッセンシャル*)

ENDOBS= テーブルオプション

処理するオブザベーションのユーザー定義範囲内の終了オブザベーション番号を指定します。

該当要素: SELECT ステートメント

カテゴリ: オブザベーションコントロール

制限事項: このテーブルオプションは、CAS サーバーではサポートされていません。
入力データセットでのみ使用します。

操作: STARTOBS=テーブルオプションと組み合わせて使用します。

データソース: SPD Engine データセット、SPD Server テーブル

構文

ENDOBS= *n*

引数

n
終了オブザベーションの番号です。

詳細

STARTOBS=または ENDOBS=オプションでオブザベーションの範囲を指定しない限り、ソフトウェアはデータセット全体のすべてのオブザベーションを処理します。ENDOBS=オプションを指定せずに STARTOBS=オプションを使用すると、ENDOBS=の暗黙の値はデータセットの終わりになります。両方のオプションを一緒に使用する場合、ENDOBS=の値は STARTOBS=の値より大きくなければなりません。

SAS ソフトウェアオプションの FIRSTOBS=および OBS=とは対照的に、STARTOBS=および ENDOBS= SPD Server オプションは、テーブル入力操作に加えて、WHERE 句の処理に使用できます。ENDOBS=が WHERE 式で使用されている場合、

ENDOBS=値は、返されるオブザベーションの数ではなく、処理する最後のオブザベーションを表します。

例

次のコードは、ENDOBS=テーブルオプションが FedSQL SELECT ステートメントでどのように指定されているかを示しています。

```
select * from spdslib.table1(startobs=3 endobs=4);
```

関連項目

テーブルオプション:

- “STARTOBS= テーブルオプション” (1522 ページ)

EXTENDOBSCOUNTER= テーブルオプション

新しい出力 SAS データセットの最大オブザベーション数を拡張するかどうかを指定します。

該当要素: CREATE TABLE ステートメント

カテゴリ: テーブルコントロール

別名: EOC=

デフォルト: YES

制限事項: このテーブルオプションは、CAS サーバーではサポートされていません。

データソース: SAS データセット

構文

EXTENDOBSCOUNTER=YES | NO

引数

YES
32 ビットの制限を超えてオブザベーションをカウントする、新しく作成された SAS データセット内の拡張ファイル形式を要求します。この SAS データセットは、オブザベーション数を 32 ビット整数で格納する動作環境用に作成されていますが、データセットはカウンターに関しては 64 ビットファイルのように動作します。これがデフォルト値です。

制限事項 EXTENDOBSCOUNTER=YES は、内部データ表現がオブザベーション数を 32 ビット整数として格納する出力 SAS データセットに対してのみ有効です。EXTENDOBSCOUNTER=は、64 ビット整数の SAS データセットでは無視されます。

NO
新しく作成された SAS データファイルの最大オブザベーション数が、動作環境の長整数サイズによって決定されることを指定します。32 ビット整数の動作環境では、最大数は 2**31-1 または約 20 億のオブザベーション(2,147,483,647)です。64 ビット整数の動作環境では、最大数は 2**63-1 または約 920 京のオブザベーションです。

詳細

歴史的に、SAS には、最大約 20 億のオブザベーションをカウントでき、32 ビット長整数の動作環境で完全にサポートされるという制限がありました。EXTENDOBSCOUNTER=テーブルオプションは、制限を拡張して、64 ビット長整数の動作環境の制限に一致させます。EXTENDOBSCOUNTER=NO は下位互換性のために提供されています。

FETCH_NUMERIC_TYPE= テーブルオプション

Google BigQuery からの NUMERIC データの処理方法を指定します。

カテゴリ: テーブルコントロール

デフォルト: FLOAT64

データソース: Google BigQuery

注: このオプションのサポートは 2024.09 から開始されます。

構文

FETCH_NUMERIC_TYPE=FLOAT64 | NUMERIC

必須引数

FLOAT64
Google BigQuery から取得された NUMERIC(p,s)および BIGNUMERIC (p,s)値が DOUBLE として扱われることを指定します。(DS2 DOUBLE データ型は、Google BigQuery FLOAT64 データ型にマッピングされます。)

NUMERIC
Google BigQuery から取得された NUMERIC(p,s)および BIGNUMERIC (p,s)値が NUMERIC として扱われることを指定します。

詳細

デフォルト値を使用すると、読み取りパフォーマンスが向上します。このテーブルオプションを使用すると、NUMERIC 動作に戻すことができます。

データ定義は影響を受けません。

GP_DISTRIBUTED_BY= テーブルオプション

作成するテーブルの分散キーを指定します。

カテゴリ: テーブルコントロール

別名: DISTRIBUTED_BY=

制限事項: このテーブルオプションは、CAS サーバーではサポートされていません。

データソース: Greenplum

構文

GP_DISTRIBUTED_BY= 'DISTRIBUTED BY (*column*[, ...*column*])
| DISTRIBUTED RANDOMLY'

引数

DISTRIBUTED BY (*column*, [***...column***])
分散キーとして使用する 1 つ以上の DBMS 列名を指定します。

DISTRIBUTED RANDOMLY
テーブルの行をデータベースセグメントに分散するために使用する列または列のセットを決定することを指定します。これは、ラウンドロビン分散として知られています。

詳細

DISTRIBUTED BY は、分散キーとして宣言された 1 つ以上の列でハッシュ分散を使用します。最も均一なデータ分散の場合、分散キーはテーブルの主キーまたは一意の列(または列のセット)にする必要があります。それが不可能な場合は、DISTRIBUTED RANDOMLY を選択して、データをラウンドロビンでセグメントインスタンスに送信します。値が指定されていない場合は、主キー(テーブルにある場合)またはテーブルの最初の適格な列を分散キーとして使用して、ハッシュ分散が選択されます。

DISTRIBUTED_BY は、前述のように、または DBCREATE_TABLE_OPTS=テーブルオプション内でサブミットできます。DBCREATE_TABLE_OPTS=での指定方法の例を次に示します。

```
dbcreate_table_opts='distributed by ("b")'
```

関連項目

テーブルオプション:

- “DBCREATE_TABLE_OPTS= テーブルオプション” (1480 ページ)

IDXNAME= テーブルオプション

WHERE 句の条件に一致する特定のインデックスを使用するように SAS に指示します。

カテゴリ: SAS インデックス使用のユーザーコントロール

制限事項: このテーブルオプションは、CAS サーバーではサポートされていません。
入力データセットでのみ使用します。
IDXWHERE=テーブルオプションでは使用できません

データソース: SAS データセット

構文

IDXNAME= *index-name*

引数

index-name
SAS データセットの単純インデックスまたは複合インデックスの名前(最大 32 文字)を指定します。SAS は、指定されたインデックスが最適かどうか、または順次検索の方がリソース効率が高いかどうかを判断しようとしません。

操作 指定は、データセットの永続的な属性ではなく、データセットの現在の使用に対してのみ有効です。

詳細

デフォルトでは、インデックス付き SAS データセットの WHERE 句の条件を満たすために、SAS は WHERE 句の最適化に使用できる 0 個以上の候補インデックスを識別します。候補インデックスのリストから、SAS は最高のパフォーマンスを提供すると判断したものを選択するか、データの順次パスがより効率的であると予想される場合はすべてのインデックスを 拒否します。

SAS が選択するインデックスが常に最良の最適化を提供するとは限らないため、IDXNAME=テーブルオプションを指定して、候補インデックスの 1 つを使用するよ

う SAS に指示できます。SAS が候補インデックスとして識別しないインデックスを指定すると、IDXNAME=テーブルオプションは要求を処理しません。つまり、IDXNAME=では、誤った結果を生成するインデックスを指定することはできません。

比較

IDXWHERE=テーブルオプションを使用すると、インデックスを使用するかどうかに関する SAS の決定をオーバーライドできます。

例

この例では、IDXNAME=テーブルオプションを使用して、特定のインデックスを使用して WHERE 句を最適化するよう SAS に指示します。その後、SAS は、データセットの順次検索がリソース効率を高める可能性を無視し、指定されたインデックスが最適かどうかを判断しようとしません。(EMPNUM インデックスは NOMISS オプションを使用して作成されていないことに注意してください。)

```
create table mydata.empnew
  as select * from mydata.employee {option idxname=empnum}
  where empnum < 2000;
```

関連項目

テーブルオプション

- “IDXWHERE= テーブルオプション” (1495 ページ)

IDXWHERE= テーブルオプション

SAS が WHERE 句の条件に一致するためにインデックス検索と順次検索のどちらを使用するかを指定します。

カテゴリ: SAS インデックス使用のユーザーコントロール

制限事項: このテーブルオプションは、CAS サーバーではサポートされていません。
入力データセットでのみ使用します。
SAS データセット: IDXNAME=では使用できません。
SPD Engine データセット: IDXWHERE=NO は WHERENOINDEX=では使用できません。

データソース: SAS データセット、SPD Engine データセット

構文

IDXWHERE= YES | NO

引数

YES
最適なインデックスを選択して WHERE 句を最適化し、データセットの順次検索の方がリソース効率が高い可能性を無視するよう SAS に指示します。これがデフォルト値です。

NO
すべてのインデックスを 無視し、データセットの順次検索で WHERE 句の条件を満たすように SAS に指示します。

注 IDXWHERE=テーブルオプションを使用して、BY ステートメントを処理するためのインデックスの使用をオーバーライドすることはできません。

IDXWHERE=NO が使用されている場合、WHERENOINDEX=テーブルオプションは使用できません。

詳細

デフォルトでは、インデックス付きデータセットの WHERE 句の条件を満たすために、インデックスを使用するか、データセットを順次読み取るかがソフトウェアによって決定されます。ソフトウェアは相対的な効率を推定し、より効率的な方法を選択します。

IDXWHERE=テーブルオプションを指定して、ソフトウェアの決定をオーバーライドする必要がある場合があります。決定は、最良の結果が得られない場合がある一般的な規則に基づいているためです。つまり、IDXWHERE=テーブルオプションを指定することで、処理方法を決定することができます。

注: 指定は、データセットの永続的な属性ではなく、データセットの現在の使用に対してのみ有効です。

比較

IDXNAME=テーブルオプション(SAS データセットでのみサポート)を使用すると、SAS に特定のインデックスを使用するように指示できます。

WHERENOINDEX=テーブルオプションを使用すると、WHERE 式の評価を行うときに除外するインデックスのリストを指定できます。

例

例 1: インデックス使用の指定

この例では、IDXWHERE=テーブルオプションを使用して、WHERE 句を最適化するのに最適なインデックスを決定するよう SAS に指示します。次に、SAS は、データセットの順次検索の方がリソース効率が高い可能性を無視します。

```
create table mydata.empnew
  as select * from mydata.employee {option idxwhere=yes}
  where empnum < 2000;
```

例 2: インデックス使用なしの指定

この例では、IDXWHERE=テーブルオプションを使用して、インデックスを無視し、データセットの順次検索で WHERE 句の条件を満たすように SAS に指示します。

```
create table mydata.empnew
  as select * from mydata.employee {option idxwhere=no}
  where empnum < 2000;
```

関連項目

テーブルオプション:

- “IDXNAME= テーブルオプション” (1494 ページ)
- “WHERENOINDEX= テーブルオプション” (1529 ページ)

IN= テーブルオプション

テーブルが現在の行にデータを提供したかどうかを示す整数変数を作成します。

制限事項: SET および MERGE ステートメントでのみ使用します。

データソース: すべて

構文

IN=*variable*

必須引数

variable
その入力テーブルが現在の行にデータを提供したかどうかを示す値を持つ新しい変数に名前を付けます。DS2 プログラム内では、変数の値は、テーブルが現在の行に寄与している場合は 1、それ以外の場合は 0 です。

操作	変数が明示的に宣言されていない場合、SET または MERGE ステートメントのローカルスコープで INTEGER として自動的に宣言されます。
データの種類	BIGINT、INTEGER、SMALLINT、TINYINT

詳細

SET または MERGE ステートメントのテーブル名の後にかっこで囲んで IN=テーブルオプションを指定します。IN=変数の値は、DS2 プログラムの実行中にプログラムステートメントで使用できますが、作成中のテーブルにその変数は含まれません。作成中のテーブルで IN=変数の値を取得するには、別の変数を割り当てます。

BY グループ処理で IN=を使用する場合、そのテーブルが現在の行に寄与していれば、IN=変数は 1 に設定されます。テーブルが現在の行に寄与しなかった場合、IN=変数は 0 に設定されます。IN=変数は、SET または MERGE ステートメントによって常に 1 または 0 に設定されますが、後続のプログラミングロジックによって変更できます。

注: IN=変数が SET または MERGE ステートメントの前に設定されている場合、その値は SET または MERGE ステートメントの実行中に失われます。

例

次の例は、IN=テーブルオプションを示しています。

```
data _null_;
  dcl int gro;
  method run();
    dcl smallint gpo;
    dcl tinyint gpf;
    set gas_price_option (in=gpo) gas_rbid_option (in=gro)
      gas_price_forward (in=gpf);
    put gpo= gro= gpf=;
  end;
enddata;
```

関連項目

ステートメント:

- “DATA ステートメント” (1310 ページ)
- “SET ステートメント” (1416 ページ)

INLINE テーブルオプション

パッケージまたはスレッドのソースコードが再利用のためにテーブルに保存されず、データプログラムによってロードされた場合にのみ検証およびコンパイルされることを指定します。

制限事項: PACKAGE および THREAD ステートメントでのみ使用します。

データソース: すべて

構文

INLINE

詳細

デフォルトでは、DS2 はパッケージとスレッドのソースコードを検証およびコンパイルして、テーブルに保存しようとします。INLINE テーブルオプションを指定すると、次のアクションが発生します。

- パッケージとスレッドのソースコードはメモリに保存されます。
- パッケージとスレッドのソースコードは、将来使用するためにテーブルに書き込まれません。
- パッケージとスレッドのソースコードは、データプログラムによってロードされていない場合、検証およびコンパイルされません。

PROC DS2 で実行している場合、インライン化されたパッケージとスレッドのソースコードは、定義された時点から現在の RUN ブロックの最後までしか使用できません。RUN ブロックの最後で、インライン化されたパッケージとスレッドのソースコードが削除されます。

次の例を考えてみましょう。データプログラムがパッケージ **mypkg** と同じ RUN ブロックで定義されているため、**mypkg** 変数 **p1** の宣言は成功します。パッケージ **mypkg** がこの RUN ブロックで定義されていないため、**mypkg** 変数 **p2** の宣言は失敗します。

```
package mypkg / inline;
method mypkg();
  put 'constructing mypkg instance';
end;
endpackage;

data _null_;
  dcl package mypkg p1();  /* OK */
enddata;

run;
```

```
data _null_;
  dcl package mypkg p2();  /* ERROR */
enddata;

run;
```

例

次の例では、INLINE テーブルオプションを使用して、パッケージとスレッドのソースコードの保存を抑制しています。

```
package pkg1 / inline;
 method pkg1();
   put 'hello from pkg1';
 end;
endpackage;

package pkg2 / inline;
 method pkg2();
   dcl package pkg1 p1();
   put 'hello from pkg2';
 end;
endpackage;

package pkg3 / inline;
 method pkg3();
   dcl package pkg1 p1();
   dcl package pkg2 p2();
   put 'hello from pkg3';
 end;
endpackage;

thread thd1 / inline;
 dcl double x;
 method init();
   put 'BEGIN EXPECTED OUTPUT';
   put 'hello from thd1 ';
   put 'hello from pkg1 ';
   put 'hello from pkg1 ';
   put 'hello from pkg2 ';
   put 'hello from pkg3 ';
   put 'hello from thd1';
   put 'END EXPECTED OUTPUT';
   put;
   put 'BEGIN TEST OUTPUT';
 end;

 method run();
   dcl package pkg3 p3();
 end;

 method term();
   put 'END TEST OUTPUT';
 end;
```

```
endthread;

data _null_;
 dcl thread thd1 t();

 method run();
  set from t threads=1;
 end;
enddata;
run;
```

次の行が SAS ログに書き込まれます。

```
BEGIN EXPECTED OUTPUT
hello from thd1
hello from pkg1
hello from pkg1
hello from pkg2
hello from pkg3
hello from thd1
END EXPECTED OUTPUT

BEGIN TEST OUTPUT
hello from pkg1
hello from pkg1
hello from pkg2
hello from pkg3
END TEST OUTPUT
```

INLINE テーブルオプションが使用されなかった場合、パッケージとスレッドのコードは一時テーブルに保存され、SAS ログに次の行が書き込まれることもわかります。

```
NOTE: Created package pkg1 in data set work.pkg1.
NOTE: Created package pkg2 in data set work.pkg2.
NOTE: Created package pkg3 in data set work.pkg3.
NOTE: Created thread thd1 in data set work.thd1.
```

IOBLOCKSIZE= テーブルオプション

I/O 操作で使用されるブロック内の行数を指定します。

カテゴリ: テーブルコントロール

制限事項: このテーブルオプションは、CAS サーバーではサポートされていません。

データソース: SPD Engine データセット、SPD Server テーブル

構文

IOBLOCKSIZE= *n*

引数

n
ブロックのサイズをバイト単位で指定します。デフォルト値は、データソースでサポートされる最小のブロックサイズです。

詳細

ソフトウェアは、テーブル内の行をブロック単位で読み取り、格納します。IOBLOCKSIZE=は、圧縮または暗号化されたテーブルで役立ちます。ソフトウェアは、非圧縮テーブルまたは非暗号化テーブルでは IOBLOCKSIZE=を使用しません。

圧縮または暗号化するテーブルの場合、IOBLOCKSIZE=の指定によって、ブロックに含める行数が決まります。この指定は、ブロック圧縮と、ディスクとの間のデータ I/O に適用されます。IOBLOCKSIZE=値は、ディスク上のテーブルの編成に影響します。

圧縮または暗号化を使用する場合は、データへのアクセス方法(順次またはランダム)を補足する IOBLOCKSIZE=値を指定します。フルテーブルスキャンを必要とする順次アクセスまたは操作では、大きなブロックサイズが優先されます。対照的に、ランダムアクセスではブロックサイズが小さい方が優先されます。SPD Engine では、大きなブロックサイズは 131,072 バイトになります。許可される最小値は 32,768 バイトです。SPD Server の場合、大きなブロックサイズは 64,000 バイトです。許可される最小値は 8,000 バイトです。

関連項目

テーブルオプション:

- “COMPRESS= テーブルオプション” (1477 ページ)
- “ENCRYPT= テーブルオプション” (1484 ページ)
- “PADCOMPRESS= テーブルオプション” (1510 ページ)

KEEP= テーブルオプション

入力テーブルの場合、処理する列を指定します。出力テーブルの場合、テーブルに書き込む列を指定します。

データソース: すべて

構文

KEEP=(*column-list*)

必須引数

column-list
出力テーブルに保持する列の名前を指定します。

制限事項 *col1–col5* 形式の番号付き範囲リストおよび *col:*形式の名前接頭辞リストはサポートされていません。

詳細

KEEP=テーブルオプションは、*column-list* のすべての列が出力行の作成に含まれるように指定します。通常、プログラムデータベクトルのすべての列は出力行に含まれます。KEEP=テーブルオプションが指定されている場合、KEEP=テーブルオプションに含まれていないすべての列が出力行から削除されます。

KEEP=テーブルオプションが入力テーブルに関連付けられている場合、プログラム実行中の処理に使用できるのは、KEEP=テーブルオプションで指定された列のみです。

デフォルトでは、グローバル変数のみが出力に含まれます。プログラムループとインデックスに使用されるローカル変数は、出力から明示的に削除する必要はありません。

比較

KEEP=テーブルオプションは、次の点で KEEP ステートメントとは異なります。

- DS2 プログラムでは、KEEP=テーブルオプションを入力テーブルと出力テーブルの両方に適用できます。KEEP ステートメントは、出力テーブルにのみ適用されます。
- DS2 プログラムで、複数の出力テーブルを作成する場合は、KEEP=テーブルオプションを使用して、異なる列を異なるテーブルに書き込みます。KEEP ステートメントは、すべての出力テーブルに適用されます。
- DROP=テーブルオプションは、処理中に省略する列、または出力テーブルから省略する列を指定します。

例

この例では、IDNUM と SALARY のみが PAYROLL から読み取られ、これらは PAYROLL で処理に使用できる唯一の変数です。

```
data bonus;
  method run();
    set payroll(keep=(idnum salary));
    bonus=salary*1.1;
  end;
enddata;
```

関連項目

テーブルオプション:

- “DROP= テーブルオプション” (1483 ページ)

LABEL= テーブルオプション

テーブルのラベルを指定します。

制限事項: このテーブルオプションは、CAS サーバーではサポートされていません。

データソース: SAS データセット、SPD Engine データセット

構文

LABEL='*label*'

必須引数

'*label*'
最大 256 文字のテキスト文字列を指定します。ラベルテキストに単一引用符が含まれている場合は、ラベルを二重引用符で囲むか、ラベルテキストで 2 つの単一引用符を使用して、文字列を単一引用符で囲みます。テーブルからラベルを削除するには、引用符で囲まれた空白に等しいラベルを割り当てます。

詳細

入力テーブルと出力テーブルの両方で LABEL=オプションを使用できます。入力テーブルで LABEL=を使用すると、DS2 プログラムの期間中テーブルにラベルが割り当てられます。出力テーブルに指定すると、ラベルはそのテーブルの永久的な部分になります。

テーブルに割り当てられたラベルは、その場でテーブルを更新しても、そのテーブルに関連付けられたままになります。ただし、以前にラベルが割り当てられたテーブルを使用して DS2 プログラムで新しいテーブルを作成すると、ラベルは失われます。たとえば、新しい出力テーブル ONE を作成すると、以前にテーブル ONE に割り当てられていたラベルが失われます。

```
data one;
  set one;
enddata;
```

比較

DECLARE ステートメントの HAVING 句の LABEL=オプションを使用すると、変数にラベルを割り当てることもできます。

例

これらの例では、テーブルにラベルを割り当てます。

```
data w2 (label = '2009 W2 Info, Hourly);
data new (label = 'Sales'' list');
data acct (label = "Hillside''s Daily Account");
data sales (label = 'May (South)');
```

LOCKTABLE= テーブルオプション

テーブルに共有ロックまたは排他ロックをかけます。

制限事項: このテーブルオプションは、CAS サーバーではサポートされていません。

データソース: SAS データセット

構文

LOCKTABLE=SHARE | EXCLUSIVE

必須引数

SHARE | EXCLUSIVE
テーブルを共有モードでロックするか、排他的にロックするかを指定します。

注: 共有モードでは、他のユーザーまたはプロセスはテーブルからデータを読み取ることができますが、データを更新することはできません。テーブルが排他的にロックされている場合、他のユーザーは自分が開いているテーブルにアクセスできません。

詳細

テーブルをロックできるのは、自分が所有者であるか、必要な権限が付与されている場合のみです。

PROC DS2 を使用する場合、LOCKTABLE オプションのデフォルト値は EXCLUSIVE です。

埋め込み FedSQL ステートメントで LOCKTABLE=テーブルオプションを指定する場合は、FedSQL テーブルオプション構文を使用してください。例については、“SET ステートメント” (1416 ページ)を参照してください。

同じテーブルが複数回参照される場合は、テーブルを使用するたびに LOCKTABLE を指定する必要があります。

ORHINTS= テーブルオプション

FedSQL から Oracle に渡す Oracle ヒントを指定します。

該当要素: DELETE ステートメント、INSERT ステートメント、SELECT ステートメント、UPDATE ステートメント

カテゴリ: データコントロール

制限事項: このテーブルオプションは、CAS サーバーではサポートされていません。

データソース: Oracle

構文

ORHINTS='/* *Oracle-hint* */'

引数

Oracle-hint
FedSQL 言語プロセッサがクエリの一部として DBMS に渡す Oracle ヒントを指定します。ヒントを指定しない場合、FedSQL は Oracle にヒントを送信しません。詳細については、*SAS Federation Server: 管理者ガイド*を参照してください。

詳細

ORHINTS=テーブルオプションは、DRIVER_TRACE=および DRIVER_TRACEFILE=データソース接続オプションと組み合わせて使用されます。PROC FEDSQL および PROC DS2 は、データソース接続オプションの使用をサポートしていません。

例: ORHINTS=オプションの使用

次の例は、ORHINTS=を指定する方法を示しています。

```
select * from test{options orhints='/* dummy hint */'} ;

update test{options orhints='/* dummy hint */'}  set c1=1;

insert into test{options orhints='/* dummy hint */'}  values (100);
```

```
delete from test{options orhints='/* dummy hint */'} ;
```

ORNUMERIC= テーブルオプション

Oracle NUMBER 列から読み取られた数値または Oracle NUMBER 列に挿入された数値の処理方法を指定します。

カテゴリ: テーブルコントロール

デフォルト: 2025.08 以降:BULKLOAD=YES が有効な場合は NO です。それ以外の場合は、YES です。
以前のリリースの場合: YES

制限事項: このテーブルオプションは、CAS サーバーではサポートされていません。

データソース: Oracle

データの種類: DECIMAL、NUMERIC

構文

ORNUMERIC=YES | NO

引数

NO
数値が TKTS_DOUBLE 値として扱われることを示します。精度が 14 桁を超えない場合があります。

YES
明示的な精度を持つ非整数値が TKTS_NUMERIC 値として扱われることを示します。

詳細

このオプションは、接続オプションとテーブルオプションの両方として指定できます。接続オプションとテーブルオプションの両方を指定すると、テーブルオプションの値が接続オプションをオーバーライドします。2025.07 以前のリリースでは、このオプションのデフォルトは YES であるため、精度またはスケールのある NUMBER 列は TKTS_NUMERIC として記述されます。2025.08 以降では、BULKLOAD=YES が指定されている場合、このオプションのデフォルトは NO になります。それ以外の場合は、デフォルトで YES になります。

OVERWRITE= テーブルオプション

テーブルの場合、置換出力テーブルに行が入力される前に出力テーブルを削除します。パッケージとスレッドの場合、同じ名前のパッケージまたはスレッドが存在する場合は、既存のパッケージまたはスレッドを削除します。

制限事項: このテーブルオプションは、CAS サーバーではサポートされていません。デフォルトでは、CAS のテーブルは常に上書きされます。

データソース: すべて

構文

OVERWRITE=YES | NO

必須引数

YES | NO
置換出力テーブルが作成される前に出力テーブルが削除されるかどうか、またはパッケージまたはスレッドが削除されるかどうかを指定します。

注意
テーブルの場合、OVERWRITE=YES ステートメントは、バックアップされたデータまたは再構築可能なデータでのみ使用してください。 出力テーブルが最初に削除されるため、出力テーブルの書き込み中に障害が発生すると、データが失われます。

デフォルト NO

詳細

テーブルの詳細

デフォルトでは、DS2 では、OVERWRITE=テーブルオプションが YES に設定されていない限り、テーブルは上書きされません。出力テーブルが存在し、OVERWRITE=テーブルオプションが NO に設定されている場合、既存のテーブルが上書きされないため、エラーが発生します。

パッケージの詳細

PACKAGE ステートメントで OVERWRITE=YES を設定し、DS2 スレッドまたは通常のテーブルが作成中のパッケージと同じ名前で存在する場合、そのテーブルは削除されません。パッケージのみが削除されます。

スレッドの詳細

THREAD ステートメントで OVERWRITE=YES を設定し、DS2 パッケージまたは通常のテーブルが作成中のスレッドと同じ名前で存在する場合、そのテーブルは削除されません。スレッドのみが削除されます。

例

例 1: DROP、KEEP、および OVERWRITE の使用

次の例では、テーブル **a** および **b** に DROP=、KEEP=、および OVERWRITE=テーブルオプションを使用しています。

```
data
  a(keep=x overwrite=yes)
  _rowset_(drop=(x z))
  b(keep=z overwrite=yes);
  dcl double x y z;
  method init();
    do x = 1 to 10;
      y = 2*x;
      z = 3*x;
      output;
    end;
  end;
enddata;
run;
data;
  method run();
    set a;
  end;
enddata;
run;
data;
  method run();
    set b;
  end;
enddata;
run;
```

例 2: テーブルの上書き

次の例では、テーブルを作成してから、OVERWRITE=YES を指定せずにテーブルを上書きしようとします。

```
data a;
  method init();
    do x = 1 to 10;
      y = 2*x;
      z = 3*x;
      output;
    end;
  end;
enddata;
```

```
run;
/* This program fails because it is impossible to overwrite table A */
data a;
  method run();
    x = y + z;
    output;
  end;
enddata;
run;
/* This program deletes (drops) table A before attempting to create it, */
/*  so the program executes without error.  If there is an error during
/* execution, the old version of A is lost.  */
data a(overwrite=yes);   method run();
    x = y + z;
    output;
  end;
enddata;
run;
```

PADCOMPRESS= テーブルオプション

OUTPUT または UPDATE 用に開かれた SPD Engine データセットの圧縮ブロックに追加するバイト数を指定します。

カテゴリ: テーブルコントロール

制限事項: このテーブルオプションは、CAS サーバーではサポートされていません。

データソース: SPD Engine データセット

構文

PADCOMPRESS= *n*

引数

n
追加するバイト数を指定します。デフォルトの数は 0 (ゼロ)です。

詳細

圧縮された SPD Engine データセットは、ディスク上の領域のブロックを占有します。ブロックのサイズは、データセットの作成時に指定された IOBLOCKSIZE=テーブルオプションから取得されます。データセットが更新されると、更新を保持するために新しいブロックフラグメントの作成が必要になる場合があります。さらに更新すると新しいフラグメントが作成される可能性があり、その結果、データセットの読み取りに必要な I/O 操作の数が増加します。

データセットへの更新が多数予想される特定の状況でブロックパディングを 増やすことにより、断片化を最小限に抑えることができます。ただし、データセットを更新しない場合、パディングを追加すると領域が浪費される可能性があります。

すべての圧縮ブロックをパディングするコストと、一部の圧縮ブロックが断片化する可能性があるコストを比較検討する必要があります。

データセットを作成または更新するときに PADCOMPRESS=テーブルオプションを指定すると、ブロックがディスクに書き戻されるときにすべてのブロックに領域が追加されます。PADCOMPRESS=設定は、データセットのメタデータに保持されません。

関連項目

テーブルオプション:

- “COMPRESS= テーブルオプション” (1477 ページ)
- “IOBLOCKSIZE= テーブルオプション” (1501 ページ)

PARTSIZE= テーブルオプション

テーブルパーティションのサイズを 指定します。

該当要素: CREATE TABLE ステートメント

カテゴリ: テーブルコントロール

制限事項: このテーブルオプションは、CAS サーバーではサポートされていません。

データソース: SPD Engine データセット、SPD Server テーブル

構文

PARTSIZE= *n*

引数

n
パーティションのサイズを指定します。

要件 サイズは、MB、GB、および TB で指定できます。*n* が M、G、または T なしで指定された場合、デフォルトは MB です。たとえば、PARTSIZE=128 は PARTSIZE=128M と同じです。SPD Engine データセットの場合、デフォルト値は 128MB です。最大値は 8,796,093,022,207MB です。SPD Server テーブルの場合、デフォルト値は MINPARTSIZE=サーバーオプションの設定です。MINPARTSIZE=サーバーオプションが設定されていない場合、デフォルトは 16MB です。

SPD Engine データセットを作成するときに、行の長さが 65K を超えると、指定した PARTSIZE=と実際のパーティションサイズが一致しないことがあります。これらの数値を一致させるには、32 の倍数と行の長さである PARTSIZE=を指定します。

詳細

詳細については、*SAS Scalable Performance Data Engine: リファレンス*の"PARTSIZE=データセットオプション"および *SAS Scalable Performance Data Server: ユーザーガイド*の"PARTSIZE=テーブルオプション"を適宜参照してください。

POINTOBS= テーブルオプション

SAS が圧縮データセットを作成し、そのオブザベーションがランダムアクセス可能か、順次アクセス可能かを指定します。

カテゴリ: テーブルコントロール

制限事項: このテーブルオプションは、CAS サーバーではサポートされていません。
出力テーブルのみで使用します。

操作: COMPRESS=テーブルオプションで使用

データソース: SAS データセット

構文

POINTOBS= YES | NO

構文の説明

YES
FedSQL 言語は、オブザベーション番号によってランダムにアクセスされる 可能性のある圧縮データセットを生成します。これがデフォルトです。

オブザベーション番号でデータに直接アクセスする例を次に示します。

- DATA ステップの MODIFY および SET ステートメントの POINT=オプション
- PROC FSEDIT を使用して特定のオブザベーション番号に直接移動

> ヒント　POINTOBS=YES を指定しても、データセットから情報を取得する効率には影響しません。圧縮データセットを作成するとき、および情報を更新または追加するときに、CPU 使用率が約 10%増加します。

NO
オブザベーション番号によって圧縮されたデータセット内のオブザベーションにランダムにアクセスする機能を抑制します。

> ヒント　POINTOBS=NO を指定することは、圧縮データセット内の番号によってオブザベーションを直接指す機能が重要でないアプリケーションに適しています。オブザベーション番号でデータにアクセスする必要がない場合は、POINTOBS=NO を指定することで、パフォーマンスを約10%向上させることができます。
>
> - 圧縮データセットの作成時
> - オブザベーションの更新または追加時

詳細

REUSE=YES は POINTOBS=YES よりも優先されることに注意してください。次に例を示します。

```
create table test{options compress=yes pointobs=yes reuse=yes};
```

このコード例では、POINTOBS=NO のデータセットが生成されます。圧縮を使用する場合は POINTOBS=YES がデフォルトであるため、REUSE=YES によって POINTOBS=が NO に変更されます。

関連項目

テーブルオプション:

- “COMPRESS= テーブルオプション” (1477 ページ)
- “REUSE= テーブルオプション” (1521 ページ)

POST_TABLE_OPTS= テーブルオプション

DATA ステートメントのテーブル名の後にデータベース固有のオプションを配置できるようにします。

カテゴリ:	テーブルコントロール
デフォルト:	なし
制限事項:	このテーブルオプションは、CAS サーバーではサポートされていません。
データソース:	Hive、Spark
参照項目:	“DBCREATE_TABLE_OPTS= テーブルオプション” (1480 ページ)

“PRE_TABLE_OPTS= テーブルオプション” (1515 ページ)

構文

POST_TABLE_OPTS="*DBMS-SQL-options*"

必須引数

"*DBMS-SQL-options*"
DATA ステートメントのテーブル名の後に配置する追加のデータベース固有のオプションを指定します。DBMS 値を単一引用符または二重引用符で囲むことができます。

詳細

POST_TABLE_OPTS=テーブルオプションを単独で使用することも、関連するテーブルオプション PRE_TABLE_OPTS=および POST_STMT_OPTS=と一緒に使用することもできます。POST_STMT_OPTS=は、DBCREATE_TABLE_OPTS=のエイリアスです。たとえば、次のテンプレートに従ってデータベースオプションを 指定できます。

```
data mylib.myexample (pre_table_opts='external'
                post_table_opts='stored as orc');
        dcl float f;
        method init();
          f=11.01; output;
        end;
      enddata;
      run;

data mylib.myexample (pre_table_opts='external'
                post_table_opts='stored as orc')
            post_stmt_options="orc"} col1(int);
        dcl float f;
        method init();
          f=11.01; output;
        end;
      enddata;
      run;
```

3 つのテーブルオプションをすべて 一緒に指定すると、次のように処理されます。

```
DATA [PRE_TABLE_OPTIONS] mylib.myexample (col1 int) [POST_TABLE_OPTS]
[POST_STMT_OPTS/DBCREATE_TABLE_OPTS]
```

PRE_TABLE_OPTS= テーブルオプション

DATA ステートメントのテーブル名の前にデータベース固有のオプションを配置できるようにします。

カテゴリ: テーブルコントロール

デフォルト: なし

制限事項: このテーブルオプションは、CAS サーバーではサポートされていません。

データソース: Hive、Spark

参照項目: “DBCREATE_TABLE_OPTS= テーブルオプション” (1480 ページ)
“POST_TABLE_OPTS= テーブルオプション” (1513 ページ)

構文

PRE_TABLE_OPTS="*DBMS-SQL-options*"

必須引数

"*DBMS-SQL-options*"
DATA ステートメントのテーブル名の前に配置する追加のデータベース固有のオプションを指定します。DBMS 値を単一引用符または二重引用符で囲むことができます。

詳細

PRE_TABLE_OPTS=テーブルオプションを単独で使用することも、関連するテーブルオプション POST_TABLE_OPTS=および POST_STMT_OPTS=と一緒に使用することもできます。POST_STMT_OPTS=は、DBCREATE_TABLE_OPTS=のエイリアスです。たとえば、次のテンプレートに従ってデータベースオプションを指定できます。

```
data mylib.myexample (pre_table_opts='external'
                post_table_opts='stored as orc');
       dcl float f;
       method init();
         f=11.01; output;
       end;
      enddata;
      run;


data mylib.myexample (pre_table_opts='external'
                post_table_opts='stored as orc')
                post_stmt_options="orc"} col1(int);
       dcl float f;
```

```
  method init();
   f=11.01; output;
  end;
enddata;
run;
```

3 つのテーブルオプションをすべて 一緒に指定すると、次のように処理されます。

```
DATA [PRE_TABLE_OPTIONS] mylib.myexample (col1 int) [POST_TABLE_OPTS]
[POST_STMT_OPTS/DBCREATE_TABLE_OPTS]
```

PW= テーブルオプション

READ、WRITE、および ALTER パスワードを SAS データセットまたは SPD Engine データセットに割り当て、パスワードで保護されたファイルへのアクセスを有効にします。暗号化された SPD Server テーブルにアクセスするためのキー値を指定します。

カテゴリ: テーブルコントロール

制限事項: このテーブルオプションは、CAS サーバーではサポートされていません。

データソース: SAS データセット、SPD Engine データセット、SPD Server テーブル

注: この操作の後にログをチェックして、パスワード値が表示されていないことを確認してください。詳細については、*SAS プログラマガイド: エッセンシャル*の"パスワードと暗号化キーの値を消去する"を参照してください。

構文

PW= *password* | *key-value*

引数

password
有効な SAS 名である必要があります。

key-value
有効な SAS 名である必要があります。

詳細

PW=テーブルオプションは、SAS データセット、SPD Engine データセット、および SPD Server テーブルに適用されます。SAS データセットおよび SPD Engine データセットの場合、このオプションを使用して、データセットにパスワードを割り当てたり、パスワードで保護されたデータセットにアクセスしたりできます。パスワードで保護されているデータセットが置き換えられると、新しいデータセットはパスワードを継承します。

注: SAS パスワードは、SAS システム以外の SAS ファイルへのアクセスを制御しません。SAS 外の SAS ファイルへのアクセスを制御するには、オペレーティングシステムが提供するユーティリティとファイルシステムのセキュリティ 制御を使用する必要があります。

SPD Server テーブルの場合、PW=オプションを使用して、すでに暗号化されている SPD Server テーブルにアクセスできます。PW=オプションを使用すると、SAS 独自の暗号化アルゴリズムによって保護されているテーブルファイルにアクセスできます。

例

例 1: PW=によるパスワードまたは暗号化キーの割り当て

```
create table myfiles.mytable {options pw=green}(column1 double);
```

例 2: PW=によるパスワードまたは暗号化キーの指定

```
select * from myfiles.mytable {option pw=green};
```

READ= テーブルオプション

READ パスワードを SAS ファイルに割り当てて、ユーザーがパスワードを入力しない限り、ファイルを読み取れないようにします。

制限事項: このテーブルオプションは、CAS サーバーではサポートされていません。

データソース: SAS データセット、SPD Engine データセット

注: この操作の後にログをチェックして、パスワード値が表示されていないことを確認してください。詳細については、"ログからパスワードと暗号化キーの値を消去する" (*SAS プログラマガイド: エッセンシャル*)の"パスワード値と暗号化キー値の消去"を参照してください。

構文

READ=*read-password*

必須引数

read-password
有効な SAS 名である必要があります。

詳細

READ=オプションは、カタログを除くすべてのタイプの SAS ファイルに適用されます。このオプションを使用して、SAS ファイルにパスワードを割り当てたり、読み取り保護された SAS ファイルにアクセスしたりできます。コードが SAS ログに書き込まれると、パスワードは消去されます。次に例を示します。

```
declare package sales (read=XXXXXXX);
```

注: SAS パスワードは、SAS 以外の SAS ファイルへのアクセスを制御しません。SAS 外の SAS ファイルへのアクセスを制御するには、オペレーティングシステムが提供するユーティリティとファイルシステムのセキュリティ 制御を使用する必要があります。

関連項目

テーブルオプション:

- “ENCRYPT= テーブルオプション” (1484 ページ)
- “PW= テーブルオプション” (1516 ページ)
- “WRITE= テーブルオプション” (1529 ページ)

READMODE= テーブルオプション

Google BigQuery から SAS にデータを移動するときに使用する方法を指定します。

カテゴリ:	テーブルアクセス
デフォルト:	STANDARD
制限事項:	このテーブルオプションは、CAS サーバーではサポートされていません。
データソース:	Google BigQuery

構文

READMODE=STANDARD | STORAGE

オプション引数

STANDARD
SAS/ACCESS 方法を使用して、データを SAS に移動します。

STORAGE
Storage API for Google を使用して、データを SAS に移動します。

詳細

READMODE=STORAGE は、READMODE=STANDARD より高速です。

RENAME= テーブルオプション

列の名前を変更します。

データソース: すべて

構文

RENAME=(*old-name* { = | **AS** } *new-name* [...*old-name* { = | AS } *new-name*])

必須引数

old-name
名前を変更する列。

new-name
列の新しい名前。データソースの有効な名前である必要があります。

詳細

RENAME=テーブルオプションを使用すると、1 つ以上の列の名前を変更できます。

テーブルを作成するときに RENAME=を使用すると、新しい列名が出力テーブルに含まれます。入力テーブルで RENAME=を使用すると、DS2 プログラミングステートメントで新しい名前が使用されます。

同じ DS2 プログラム内で RENAME=を DROP=または KEEP=テーブルオプションと一緒に使用すると、DROP=および KEEP=テーブルオプションが RENAME=の前に適用されます。DROP=および KEEP=テーブルオプションでは古い名前を使用する必要があります。同じステートメントで同じ列を削除して名前を変更することはできません。

RENAME=は、列の名前を変更するだけでなく、列のラベルも変更します。

比較

- RENAME=テーブルオプションは、次の点で RENAME ステートメントとは異なります。

- RENAME ステートメントは、すべての出力テーブルに適用されます。異なるテーブルの異なる列の名前を変更する場合は、RENAME=テーブルオプションを使用する必要があります。
- RENAME=テーブルオプションを使用すると、入力テーブルまたは出力テーブルごとに名前を変更する列を指定できます。入力テーブルで使用して、処理前に列の名前を変更します。
- RENAME ステートメントと RENAME=出力テーブルオプションの両方を使用する場合、RENAME ステートメントが優先されます。RENAME ステートメントで X が Y に名前変更され、RENAME=テーブルオプションで X が Z に名前変更された場合、RENAME ステートメントが優先され、X は Y に名前変更されます。

- プログラムロジックで、同じ名前の列を持つ 2 つの入力テーブルなどの列の名前を変更する必要がある場合は、RENAME ステートメントまたは RENAME=テーブルオプションを使用します。

例

例 1: 出力時に列の名前を変更する

この例では、DATA ステートメントで RENAME=を使用して、出力テーブルに書き込まれるときに列の名前が変更されることを示しています。DS2 処理中、列は元の名前 X を保持します。

```
data two(rename=(x=keys))
  method run();
    set one;
    z=x+y;
  run;
enddata;
```

例 2: 入力時に列の名前を変更する

この例では、SET ステートメントで列 X を KEYS という名前の列に名前変更します。これは、DS2 処理前の名前変更です。DS2 処理で使用する列の名前は、X ではなく KEYS です。

```
data three;
  method run();
    set one(rename=(x AS keys));
    z=keys+y;
  run;
enddata;
```

関連項目

ステートメント:

- “RENAME ステートメント” (1405 ページ)

REUSE= テーブルオプション

圧縮された SAS データセットの空き領域に新しい行を書き込むことができるかどうかを指定します。

カテゴリ: テーブルコントロール

制限事項: このテーブルオプションは、CAS サーバーではサポートされていません。
出力データセットでのみ使用します。

データソース: SAS データセット

構文

REUSE= NO | YES

引数

NO
圧縮された SAS データセット内の領域を追跡および再利用しません。新しい行が既存のデータセットに追加されます。NO を指定すると、データセット内の多くの行を削除または更新する場合、データストレージの効率が低下します。

YES
圧縮された SAS データセット内の領域を追跡して再利用します。新しい行は、他の行が更新または削除されたときに解放された領域に挿入されます。

圧縮データセットの最後に行を追加する操作を使用する場合は、REUSE=NO を使用します。REUSE=YES を指定すると、必ずしもファイルの最後ではなく、ファイル内に領域があればどこでも新しい行が追加されます。

詳細

デフォルトでは、新しい行は既存の圧縮データセットに追加されます。他の行を削除または更新して空き領域を追跡および再利用する場合は、圧縮された SAS データセットを作成するときに REUSE=テーブルオプションを使用します。

REUSE=テーブルオプションは、COMPRESS=YES オプションを指定して新しいデータセットを作成する場合にのみ意味があります。既存の SAS データセットにアクセスしているときに REUSE=テーブルオプションを使用しても効果はありません。

関連項目

テーブルオプション:

- “COMPRESS= テーブルオプション” (1477 ページ)

- “POINTOBS= テーブルオプション” (1512 ページ)

STARTOBS= テーブルオプション

処理するユーザー定義の行範囲の開始行番号を指定します。

該当要素: SELECT ステートメント

カテゴリ: オブザベーションコントロール

制限事項: このテーブルオプションは、CAS サーバーではサポートされていません。
入力データセットでのみ使用します。

データソース: SPD Engine データセット、SPD Server テーブル

構文

STARTOBS= *n*

引数

n
開始行の番号です。

詳細

STARTOBS=および ENDOBS=オプションで行の範囲を指定しない限り、ソフトウェアはデータセット全体を処理します。ENDOBS=オプションを STARTOBS=オプションなしで使用すると、STARTOBS=の暗黙の値は 1 になります。両方のオプションを一緒に使用する場合、STARTOBS=の値は、ENDOBS=の値より小さくなければなりません。

SAS ソフトウェアオプションの FIRSTOBS=および OBS=とは対照的に、STARTOBS=および ENDOBS=オプションは、テーブル入力操作に加えて、WHERE 句の処理に使用できます。STARTOBS=が WHERE 式で使用されている場合、STARTOBS=値は WHERE 式を適用する最初の行を表します。

関連項目

テーブルオプション:

- “ENDOBS= テーブルオプション” (1490 ページ)

SCANSTRINGCOLUMNS= テーブルオプション

データベーステーブルまたはクエリの文字列の最大長を決定するかどうかを指定します。

カテゴリ: テーブルアクセス

別名: SCAN_STRINGS=
SCAN_STRING_COLUMNS=
SCANSTRINGCOLUMNS=

デフォルト: NO

データソース: Google BigQuery

構文

SCANSTRINGCOLUMNS=NO | YES

必須引数

NO
データをロードする前に文字列がスキャンされないように指定します。

YES
データをロードする前に、文字列をスキャンして実際の最大長を判別することを指定します。

詳細

Google BigQuery の文字データは、固定長ではない STRING 列に格納されます。MAX_CHAR_LEN= LIBNAME オプションは、BigQuery テーブルの列の最大長を指定するために使用されます。MAX_CHAR_LEN=に値が指定されていない場合、デフォルト値の 16,384 文字が使用されます。デフォルト設定は、切り捨てを避けるために大きくなっています。

SCANSTRINGCOLUMNS=テーブルオプションを使用すると、テーブルの MAX_CHAR_LEN= LIBNAME オプションの設定をオーバーライドできます。SCANSTRINGCOLUMNS=は、選択した列の実際の最大長を取得します。このオプションを使用すると、結果のテーブルのサイズを縮小し、ロードプロセスを高速化できます。このオプションは任意のテーブルに指定できますが、多くの文字列を持つ大きなテーブルで最適なパフォーマンスの向上が得られます。

関連項目

"SCANSTRINGCOLUMNS= LIBNAME Statement Option" (*SAS/ACCESS for Relational Databases: Reference*)

TABLE_TYPE= テーブルオプション

SAP HANA でテーブルを作成するときに FedSQL が使用するテーブルストレージの種類を指定します。

該当要素: CREATE TABLE ステートメント

カテゴリ: データアクセス

制限事項: このテーブルオプションは、CAS サーバーではサポートされていません。

操作: GLOBAL と GLOBAL TEMPORARY は同じ動作をする
LOCAL と LOCAL TEMPORARY は同じ動作をする

データソース: SAP HANA

構文

TABLE_TYPE= [COLUMN | GLOBAL | GLOBAL TEMPORARY | LOCAL | LOCAL TEMPORARY | ROW]

引数

COLUMN
SAP HANA で列ベースのストレージを使用してテーブルを作成します。

GLOBAL | GLOBAL TEMPORARY
SAP HANA にグローバルな一時テーブルを作成します。テーブルはグローバルに使用できます。ただし、データは現在のセッションでのみ表示されます。

LOCAL | LOCAL TEMPORARY
SAP HANA にローカルの一時テーブルを作成します。テーブルの定義とデータは、現在のセッションでのみ表示されます。

ROW
SAP HANA で行ベースのストレージを使用してテーブルを作成します。

詳細

SAP HANA TABLE_TYPE=オプションは、接続オプションおよびテーブルオプションとして使用できます。どちらも指定されていない場合、SAP HANA で作成されたテーブルは、行ストアまたは列ストアの SAP HANA デフォルトに従います。

関連項目

テーブルオプション:

- “DBCREATE_TABLE_OPTS= テーブルオプション” (1480 ページ)

THREADNUM= テーブルオプション

SPD Engine データセットを処理するために SPD Engin が起動できる I/O スレッドの最大数を指定します。

カテゴリ: テーブルコントロール

制限事項: このテーブルオプションは、CAS サーバーではサポートされていません。

データソース: SPD Engine データセット

構文

THREADNUM= *n*

引数

n
スレッド数を指定します。SPDEMAXTHREADS=システムオプションが設定されている場合、デフォルトはその値です。それ以外の場合、デフォルトはコンピューターの CPU 数の 2 倍です。

詳細

THREADNUM=を使用すると、SPD Engine データセットを処理するために SPD Engine が生成する I/O スレッドの最大数を指定できます。THREADNUM=値は、次の SPD Engine I/O 処理のいずれかに適用されます。

- WHERE 式の処理
- 並列インデックス作成
- スレッド対応アプリケーションによって要求された I/O

THREADNUM=を調整すると、システム管理者は SPD Engine が任意のプロセスに使用できる CPU リソースのレベルを調整できます。たとえば、64 ビットプロセッサシステムで THREADNUM=4 を設定すると、プロセスが最大で 4 つの CPU に制限されるため、他のユーザーやアプリケーションのスループットが向上します。

THREADNUM=が 1 より大きい場合、並列処理が発生する可能性があります。したがって、物理的な順序が出力で保持されない場合があります。THREADNUM=1 の設定は、並列処理が発生しないことを意味します。

このオプションを使用して、WHERE 式の評価のスケーラビリティを調べることもできます。

構成可能なシステムオプションである SPDEMAXTHREADS=は、システムリソースの消費に上限を課すため、THREADNUM=値を制約します。

TYPE= テーブルオプション

特別に構造化された SAS データセットのデータセットタイプを指定します。

制限事項: このテーブルオプションは、CAS サーバーではサポートされていません。

データソース: SAS データセット、SPD Engine データセット

構文

TYPE=*data-set-type*

必須引数

data-set-type
データセットの特殊なタイプを指定します。

詳細

DS2 プログラムで TYPE=テーブルオプションを使用して、適切な出力形式で特殊な SAS データセットを作成するか、プロシジャステートメントで特別なタイプの SAS データセットを識別します。

CONTENTS プロシジャを使用して、データセットのタイプを判別することができます。

ほとんどの SAS データセットには指定されたタイプがありません。ただし、一部の SAS/STAT プロシジャで使用される特別に構造化された SAS データセットがいくつかあります。これらの SAS データセットには特殊な変数とオブザベーションが含まれており、通常は SAS 統計プロシジャによって作成されます。

他の値は他の SAS ソフトウェア製品で使用可能であり、適切なドキュメントで説明されています。

注: DS2 プログラムで SET ステートメントでを使用して、特殊な SAS データセットを変更する場合は、DATA ステートメントで TYPE=オプションを指定する必要があります。*data-set-type* は、作成されたデータセットに自動的にコピーされません。

関連項目

ステートメント:

- “SET ステートメント” (1416 ページ)

UNIQUESAVE= テーブルオプション

一意のインデックスを持つデータセットにオブザベーションを挿入するときに、一意でないキー値を持つオブザベーション(拒否されたオブザベーション)を別のデータセットに保存するように指定します。

該当要素: INSERT ステートメント

カテゴリ: SAS インデックス使用のユーザーコントロール

制限事項: このテーブルオプションは、CAS サーバーではサポートされていません。

操作: SPDSUSDS=自動マクロ変数と組み合わせて使用

データソース: SPD Engine データセット

構文

UNIQUESAVE= YES | NO

引数

YES
拒否されたオブザベーションを、マクロ変数 SPDSUSDS=への参照によってアクセスできる別のシステム作成テーブルに書き込みます。

NO
拒否されたオブザベーションを別のテーブルに書き込みません(つまり、一意でないキー値を無視します)。

詳細

一意のインデックスを持つデータセットにオブザベーションが挿入されると、拒否されたオブザベーションは無視されます。UNIQUESAVE=YES を指定すると、拒否されたオブザベーションは別のデータセットに保存され、その名前はマクロ変数 SPDSUSDS に保存されます。データセット名のかわりにマクロ変数を指定して、拒否されたオブザベーションを識別することができます。

USER= テーブルオプション

HDFS ユーザー名を指定します。

カテゴリ:	バルクロード
別名:	USERID=、UID=
制限事項:	このテーブルオプションは、CAS サーバーではサポートされていません。
要件	“BULKOPTS= テーブルオプション” (*SAS FedSQL 言語リファレンス*)内で指定する必要があります。
操作:	(オプション)“PASSWORD= テーブルオプション” (*SAS FedSQL 言語リファレンス*)とともに使用します。
データソース:	Impala

構文

USER=*HDFS-user*

引数

HDFS-user
HDFS ユーザーを表す名前を指定します。名前にスペースが含まれていない限り、値を囲む引用符はオプションです。

例

BULKLOAD=YES BULKOPTS=(USER=HADOOP PWD="*hdfs-password*" CFG="*config-path*")

関連項目

テーブルオプション:

- “BULKLOAD= テーブルオプション” (1474 ページ)
- “PASSWORD= テーブルオプション” (*SAS FedSQL 言語リファレンス*)

WHERENOINDEX= テーブルオプション

WHERE 式の評価を行うときに除外するインデックスのリストを指定します。

カテゴリ: SAS インデックス使用のユーザーコントロール

制限事項: このテーブルオプションは、CAS サーバーではサポートされていません。
IDXWHERE=NO では使用できません。

データソース: SPD Engine データセット

構文

WHERENOINDEX= (*name1 name2*...)

引数

(*name1 name2*...)
使用から除外するインデックス名のリストを指定します。

関連項目

テーブルオプション:

- “IDXWHERE= テーブルオプション” (1495 ページ)

WRITE= テーブルオプション

WRITE パスワードを SAS ファイルに割り当てて、ユーザーがパスワードを入力しない限り、ファイルに書き込めないようにします。

制限事項: このテーブルオプションは、CAS サーバーではサポートされていません。

データソース: SAS データセット、SPD Engine データセット

注: この操作の後にログをチェックして、パスワード値が表示されていないことを確認してください。詳細については、“ログからパスワードと暗号化キーの値を消去する” (*SAS プログラマガイド: エッセンシャル*)の"パスワード値と暗号化キー値の消去"を参照してください。

構文

WRITE=*write-password*

必須引数

write-password
有効な SAS 名である必要があります。

詳細

WRITE=オプションは、カタログを除くすべてのタイプの SAS ファイルに適用されます。このオプションを使用して、SAS ファイルにパスワードを割り当てたり、書き込み保護された SAS ファイルにアクセスしたりできます。コードが SAS ログに書き込まれると、パスワードは消去されます。次に例を示します。

```
drop thread job2a (write=XXXXXXX);
```

注: SAS パスワードは、SAS 以外の SAS ファイルへのアクセスを制御しません。SAS 外の SAS ファイルへのアクセスを制御するには、オペレーティングシステムが提供するユーティリティとファイルシステムのセキュリティ 制御を使用する必要があります。

関連項目

テーブルオプション:

- “ENCRYPT= テーブルオプション” (1484 ページ)
- “READ= テーブルオプション” (1517 ページ)

14

DS2 データグリッドパッケージのメソッド、演算子、およびステートメント

ディクショナリ **1533**
APPEND メソッド 1533
BOTTOMN メソッド 1537
CLEAR メソッド: データグリッドパッケージ 1538
CLEARDATA メソッド 1538
COLUMNADD メソッド 1539
COLUMNADDCHARTYPE メソッド 1539
COLUMNADDNUMTYPE メソッド 1540
COLUMNCLEAR メソッド 1540
COLUMNCORR メソッド 1541
COLUMNCOUNT メソッド 1542
COLUMNCOUNTMATCH メソッド 1543
COLUMNCOUNTMATCHBYFILTER メソッド 1543
COLUMNCOUNTMATCHIN メソッド 1544
COLUMNCOUNTMISSING メソッド 1545
COLUMNCOUNTVALID メソッド 1546
COLUMNDELETE メソッド 1547
COLUMNEXISTS メソッド 1547
COLUMNFREQ 演算子 1548
COLUMNINDEX メソッド 1548
COLUMNISNUMERIC メソッド 1549
COLUMNMAX メソッド 1550
COLUMNMAXBOOL メソッド 1550
COLUMNMAXINT メソッド 1551
COLUMNMAXLONG 演算子 1552
COLUMNMAXSTRING メソッド 1552
COLUMNMEAN メソッド 1553
COLUMNMEDIAN メソッド 1554
COLUMNMIN メソッド 1554
COLUMNMINBOOL メソッド 1555
COLUMNMININT メソッド 1556

COLUMNMINLONG メソッド ... 1556
COLUMNMINSTRING メソッド ... 1557
COLUMNNAME メソッド ... 1558
COLUMNRENAME メソッド ... 1558
COLUMNSTDDEV メソッド ... 1559
COLUMNSUM メソッド ... 1559
COLUMNSUMLONG メソッド ... 1560
COLUMNTYPENAME メソッド ... 1561
COPY メソッド: データグリッドパッケージ ... 1562
DECLARE PACKAGE ステートメント: データグリッドパッケージ ... 1562
DESERIALIZE メソッド ... 1564
DISTINCTCOUNT メソッド ... 1565
FILTEREDCOMPARE メソッド ... 1565
FILTEREDGET メソッド ... 1566
FILTEREDGETBYFILTER メソッド ... 1567
FILTEREDGETIN メソッド ... 1568
FILTEREDGETINDEX メソッド ... 1569
FILTEREDGETINDEXBYFILTER メソッド ... 1570
FILTEREDGETINDEXIN メソッド ... 1571
FILTEREDSET メソッド ... 1572
FILTEREDSETBYFILTER メソッド ... 1573
FILTEREDSETALL メソッド ... 1574
FILTEREDSETALLBYFILTER メソッド ... 1575
FULLJOIN メソッド ... 1576
FULLJOINBYFILTER メソッド ... 1581
FULLJOINCOUNT メソッド ... 1587
FULLJOINTCOUNTBYFILTER メソッド ... 1588
GETBOOL メソッド ... 1589
GETDOUBLE メソッド ... 1590
GETINT メソッド ... 1590
GETLONG メソッド ... 1591
GETSTRING メソッド ... 1592
GETVALUE メソッド ... 1592
INNERJOIN メソッド ... 1593
INNERJOINBYFILTER メソッド ... 1595
INNERJOINCOUNT メソッド ... 1598
INNERJOINCOUNTBYFILTER メソッド ... 1599
LEFTJOIN メソッド ... 1600
LEFTJOINBYFILTER メソッド ... 1602
LEFTJOINCOUNT メソッド ... 1606
LEFTJOINCOUNTBYFILTER メソッド ... 1607
NAME メソッド ... 1608
NEW 演算子: データグリッドパッケージ ... 1608
RIGHTJOIN メソッド ... 1609
RIGHTJOINBYFILTER メソッド ... 1612
RIGHTJOINCOUNT メソッド ... 1616
RIGHTJOINCOUNTBYFILTER メソッド ... 1617
ROWADD メソッド ... 1617
ROWCOUNT メソッド ... 1618
ROWDELETE メソッド ... 1618
SERIALIZE メソッド ... 1619
SERIALIZEDATA メソッド ... 1620
SERIALIZEMETADATA メソッド ... 1621
SETALL メソッド ... 1622

SETBOOL メソッド ... 1623
SETDOUBLE メソッド ... 1623
SETINT メソッド ... 1624
SETLONG メソッド ... 1625
SETNAME メソッド ... 1625
SETSTRING メソッド ... 1626
SETVALUE メソッド ... 1626
SORT メソッド ... 1627
SUBSET メソッド ... 1630
SUBSETBYFILTER メソッド ... 1632
TOPN メソッド ... 1634
VERSION メソッド ... 1635

ディクショナリ

APPEND メソッド

指定されたソースデータグリッドを指定されたターゲットデータグリッドに追加します。

適用対象: データグリッドパッケージ

返されるデータの種類: INTEGER

注: 2 つのデータグリッドの列が一致している必要はありません。このメソッドは、結果のターゲットデータグリッドの行数を返します。エラーが発生した場合、このメソッドは-1 を返します。

ヘッドレスデータグリッドで append メソッドを使用する場合、列のデータ型を考慮することが重要です。2 つのデータグリッド間で列のデータ型が一致しない場合、ソース列の型は、ターゲットデータグリッドの対応する列の型によって推測されます。詳細については、例 3 を参照してください。

構文

package.**APPEND**(*dgSource* | *JSON_string*);

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

dgSource
ソースデータグリッドの変数名を指定します。

JSON_string
データグリッドのデータを含む JSON 文字列を指定します。単一引用符で囲まれたリテラル値を指定するか、JSON 文字列に評価される変数を指定できます。データグリッド JSON 形式の詳細については、“データグリッドの JSON 表現について” (*SAS DS2 プログラマガイド*)を参照してください。

例

例 1: 追加

この例では、dataGrid_A がターゲットで、dataGrid_B がソースです。各データグリッドのコンテンツは次のとおりです。

表 14.1 dataGrid_A

a	b	c
Aa1	Ab1	Ac1
Aa2	Ab2	Ac2

表 14.2 dataGrid_B

e	d	c	a
Be1	Bd1	Bc1	Ba1
Be2	Bd2	Bc2	Ba2
Be3	Bd3	Bc3	Ba3

次の操作が実行されます。

```
dataGrid_A.append(dataGrid_B)
```

結果のデータグリッドは次のとおりです。

表 14.3 dataGrid_A

a	b	c	e	d
Aa1	Ab1	Ac1	null	null
Aa2	Ab2	Ac2	null	null
Ba1	null	Bc1	Be1	Bd1
Ba2	null	Bc2	Be2	Bd2

a	b	c	e	d
Ba3	null	Bc3	Be3	Bd3

注: この例では、データグリッドオブジェクトを使用します。結果は、シリアル化された JSON 文字列を使用して定義されたデータグリッドの場合と同じです。

例 2: ヘッドレスデータグリッド

この例では、表 14.16 に示されているデータグリッドには、行データのみが含まれています。列ヘッダーのメタデータは含まれません。これは、*ヘッドレス*データグリッドとして知られています。

JSON 形式のヘッドレスデータグリッドとして表示された dataGrid_B は次のとおりです。

```
[{"data":[ ["Be1","Bd1","Bc1","Ba1"], ["Be2","Bd2","Bc2","Ba2"],
["Be3","Bd3","Bc3","Ba3"] ]}]
```

テーブル形式のヘッドレスデータグリッドとして表示された dataGrid_B は次のとおりです。

表 14.4 ヘッドレス dataGrid_B テーブル

COL_1	COL_2	COL_3	COL_4
Be1	Bd1	Bc1	Ba1
Be2	Bd2	Bc2	Ba2
Be3	Bd3	Bc3	Ba3

ヘッドレスデータグリッドを追加すると、append メソッドは、位置に基づいて宛先列をソース列にマッピングします。したがって、ヘッドレス dataGrid_B が表 14.15 に追加されると、次のマッピングが発生します。

dataGrid_A.a <-- dataGrid_B.COL_1

dataGrid_A.b <-- dataGrid_B.COL_2

dataGrid_A.c <-- dataGrid_B.COL_3

dataGrid_A.COL_4 <-- dataGrid_B.COL_4

結果の dataGrid_A JSON 文字列は次のとおりです。

```
[
 {"metadata":[{"A":"string"},{"B":"string"},{"C":"string"},{"COL_4":"string"}]},
 {"data":[["Aa1","Ab1","Ac1",null],["Aa2","Ab2","Ac2",null],
       ["Be1","Bd1","Bc1","Ba1"],["Be2","Bd2","Bc2","Ba2"],["Be3","Bd3","Bc3","Ba3"]]}
]
```

結果の dataGrid_A テーブルは次のとおりです。

表 14.5　結果の dataGrid_A テーブル

A	B	C	COL_4
Aa1	Ab1	Ac1	null
Aa2	Ab2	Ac2	null
Be1	Bd1	Bc1	Ba1
Be2	Bd2	Bc2	Ba2
Be3	Bd3	Bc3	Ba3

例 3: 推測されたソース列の型

この例は、APPEND メソッドをヘッドレスデータグリッドに適用すると、ソース列の型がターゲットデータグリッドの対応する列の型によって推測されることを示しています。この例で使用されるデータグリッドの JSON 文字列は次のとおりです。

dataGrid_A_json:

```
[
{"metadata":[{"A":"string"},{"B":"double"}]},
{"data":[["A1",1.11],["A2",2.22],["A3",3.33]]}
]
```

dataGrid_B_json:

```
[{"data":[["B1",10],["B2",20],["B3",30]}]
```

逆シリアル化メソッドを dataGrid_B_json に適用するとします。

```
dgB.deserialize(dataGrid_B_json)
```

このデータグリッド(*dgB*)が生成されます。COL_2 は long 型であることに注意してください。

表 14.6　逆シリアル化操作後の dgB

COL_1 (*string*)	COL_2 (*long*)
B1	10
B2	20
B3	30

dataGrid_A_json が *dgA* というデータグリッドに逆シリアル化されているとします。次に、append メソッドを使用して dataGrid_B_json を dgA に追加するとします。

```
dgA.append(dataGrid_B_json)
```

このデータグリッドが生成されます。dataGrid_B_json の long 値が double 値に変換されていることに注意してください。

表 14.7　追加操作後の dgA

A (*string*)	B (*double*)
A1	1.11
A2	2.22
A3	3.33
B1	10.00
B2	20.00
B3	30.00

BOTTOMN メソッド

ターゲットデータグリッドを指定された列で昇順に並べ替え、最初の *N* 行に切り捨てます。

適用対象: データグリッドパッケージ

返されるデータの種類: INTEGER

注: このメソッドは、結果のターゲットデータグリッドの行数を返します。エラーが発生した場合、このメソッドは-1 を返します。

構文

package.**BOTTOMN**(*dgSource*, *col*, *N*);

パッケージ変数

package
　データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

dgSource
　ソースデータグリッドの変数名を指定します。

col
名前またはインデックス値を使用して列を指定します。

N
返される行の最大数を指定します。

CLEAR メソッド: データグリッドパッケージ

指定されたデータグリッドからすべての行とすべての列のメタデータを削除します。

返されるデータの種類: INTEGER

注: このメソッドは、成功した場合、0 (ゼロ)を返します。成功しなかった場合、このメソッドはゼロ以外の値を返します。

構文

package.**CLEAR**();

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

CLEARDATA メソッド

指定されたデータグリッドからすべての行を削除しますが、列のメタデータは削除しません。

返されるデータの種類: INTEGER

注: このメソッドは、成功した場合は 0 (ゼロ)を返し、成功しなかった場合はゼロ以外の値を返します。

構文

package.**CLEARDATA**();

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

COLUMNADD メソッド

指定された型の列をターゲットデータグリッドに追加します。

返されるデータの種類: INTEGER

注: 列が追加されると、データグリッド内の列の更新された合計数が返されます。それ以外の場合は、-1 が返されます。

構文

package.**COLUMNADD**(*colName*, *typeName*);

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

colName
列の名前を指定します。

typeName
型名の値を指定します。有効な型は、[string, int | integer, long, decimal | double, Boolean]です。

COLUMNADDCHARTYPE メソッド

指定されたデータグリッドに文字列を追加します。

返されるデータの種類: INTEGER

注: 列が追加されると、データグリッド内の列の更新された合計数が返されます。それ以外の場合は、-1 が返されます。

構文

package.**COLUMNADDCHARTYPE**(*colName*);

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

colName
列の名前を指定します。

COLUMNADDNUMTYPE メソッド

指定されたデータグリッドに decimal 型の数値列を追加します。

返されるデータの種類: INTEGER

注: 列が追加されると、データグリッド内の列の更新された合計数が返されます。それ以外の場合は、-1 が返されます。

構文

package.**COLUMNADDNUMTYPE**(*colName*);

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

colName
列の名前を指定します。

COLUMNCLEAR メソッド

データグリッド列のすべての行をクリアします。

返されるデータの種類: INTEGER

注: 行インデックス値が指定されている場合、それらの行のみがクリアされます。
このメソッドは、列がクリアされた場合は 0 (ゼロ)を返し、エラーが発生した場合はゼロ以外の値を返します。

構文

package.**COLUMNCLEAR**(*col*, [*rowStartIndex*, *rowEndIndex*]);

パッケージ変数

package
　データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

col
　名前またはインデックス値を使用して列を指定します。

オプション引数

rowStartIndex
　操作を開始する行番号を指定します。

rowEndIndex
　操作を終了する行番号を指定します。

COLUMNCORR メソッド

指定された 2 つの列の Pearson 相関係数を返します。

返されるデータの種類: DECIMAL

注: エラーが発生した場合、戻り値は **Missing-Value** です。

構文

package.**COLUMNCORR**(*col1*, *col2* [, *rowStartIndex*, *rowEndIndex*]);

パッケージ変数

package
　データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

col1
col2
　相関係数を計算する数値データグリッド列を指定します。名前またはインデックス値を使用して列を指定できます。列名を指定するには、文字列値を使用しま

す。列インデックスを指定するには、数値を使用します。インデックス値は、正の整数(または正の整数に評価される)である必要があります。

オプション引数

rowStartIndex
操作を開始する行番号を指定します。

rowEndIndex
操作を終了する行番号を指定します。

詳細

Pearson 積率相関は、2 つの変数の関連性のパラメトリックな尺度です。線形関係の強さと方向の両方を測定します。1 つの変数 X が別の変数 Y の正確な線形関数である場合、相関が 1 の場合は正の関係が存在し、相関が-1 の場合は負の関係が存在します。2 つの変数間に線形の予測可能性がない場合、相関は 0 です。2 つの変数がともにゼロ相関の正規分布に従う場合、2 つの変数は独立しています。ただし、場合によっては根本的な因果関係が存在しない可能性があるため、相関は因果関係を意味するものではありません。

ρ_{xy}で表される Pearson 積率相関の式は次のとおりです。

$$\rho_{xy} = \frac{\mathrm{Cov}(x,y)}{\sqrt{\mathrm{V}(x)\mathrm{V}(y)}} = \frac{\mathrm{E}((x-\mathrm{E}(x))(y-\mathrm{E}(y)))}{\sqrt{\mathrm{E}(x-\mathrm{E}(x))^2\mathrm{E}(y-\mathrm{E}(y))^2}}$$

COLUMNCOUNT メソッド

指定されたデータグリッドの列数を返します。

返されるデータの種類: INTEGER

注: エラーが発生した場合、戻り値は **Missing-Value** です。

構文

package.**COLUMNCOUNT**();

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

COLUMNCOUNTMATCH メソッド

指定された条件に一致する行数を返します。

返されるデータの種類: INTEGER

注: エラーが発生した場合、戻り値は **Missing-Value** です。

構文

package.**COLUMNCOUNTMATCH**(*cmpCol*, *operator*, *cmpValue*);

パッケージ変数

package
データタグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

cmpCol
名前またはインデックス値を使用して列を指定し、その値を *cmpValue* と比較します。

operator
"比較演算子" (*SAS DS2 プログラマガイド*)に示されている演算子の 1 つを指定します。これらの演算子のいずれかに評価される文字変数の名前を指定するか、単一引用符で囲んだ演算子を指定できます。

cmpValue
cmpCol の値と比較する値を指定します。数値、単一引用符で囲まれた文字列、変数の名前、または式の名前を指定できます。指定する値は cmpCol であるか、または *cmpCol* と同じデータ型に評価される必要があります。

null 値と欠損値を検索するには、次の点に注意してください。

- 文字列の場合、null は null 値のみに一致します。欠損(空の文字列またはすべての空白文字を含む文字列)は、欠損値に一致します。
- 他のすべての型では、null 値は存在しません。cmpValue に null を指定した場合、比較演算で欠損値として扱われます。

COLUMNCOUNTMATCHBYFILTER メソッド

指定された条件に一致する行数を返します。この基準はフィルターによって定義されます。

返されるデータの種類: INTEGER

注: エラーが発生した場合、戻り値は **Missing-Value** です。
このメソッドサポートは、2025.02 で開始されます。

構文

形式 1: *package*.**COLUMNCOUNTMATCHBYFILTER**(*filter* [, *varList*]);

形式 2: *package*.**COLUMNCOUNTMATCHBYFILTER**(*filter*, *rowStartIndex*, *rowEndIndex* [, *varList*]);

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

filter
行の選択基準を指定します。詳細については、“フィルター構文について” (*SAS DS2 プログラマガイド*)を参照してください。

rowStartIndex
操作を開始する行番号を指定します。

rowEndIndex
操作を終了する行番号を指定します。

オプション引数

varList
フィルター内のパラメーター化された値を解決するために使用する DS2 変数のセットを含む DS2 varlist を指定します。パラメーター解決コードでは、DS2 varList 変数のデータ型が、関連付けられた列のデータ型と一致していると想定されます。

COLUMNCOUNTMATCHIN メソッド

比較演算が真と評価された、指定された列内の一致する行の数を返します。複数の値が演算の比較子として使用されます。これらの値は、*inDatagrid* 引数に指定されたデータグリッドの最初の列で定義されます。

返されるデータの種類: INTEGER

注: エラーが発生した場合、戻り値は **Missing-Value** です。

構文

package.**COLUMNCOUNTMATCHIN**(*cmpColumn*, *cmpOperator*, *inDatagrid* [, *rowStartIndex*, *rowEndIndex*]);

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

cmpColumn
名前またはインデックス値を使用して、比較演算を実行する列を指定します。

cmpOperator
比較演算で使用する演算子を指定します。EQ (等しい)または NEQ (等しくない)のいずれかです。

inDatagrid
比較演算で使用する比較子値が最初の列に含まれるデータグリッドを指定します。

オプション引数

rowStartIndex
操作を開始する行番号を指定します。

rowEndIndex
操作を終了する行番号を指定します。

COLUMNCOUNTMISSING メソッド

指定された条件の欠損値の数を返します。

返されるデータの種類: INTEGER

注: エラーが発生した場合、戻り値は **Missing-Value** です。

構文

package.**COLUMNCOUNTMISSING**(*col* [, *rowStartIndex*, *rowEndIndex*]);

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

col
名前またはインデックス値を使用して列を指定します。

オプション引数

rowStartIndex
操作を開始する行番号を指定します。

rowEndIndex
操作を終了する行番号を指定します。

COLUMNCOUNTVALID メソッド

指定された条件の有効な非欠損数値の数を返します。

返されるデータの種類: INTEGER

注: エラーが発生した場合、戻り値は **Missing-Value** です。

構文

package.**COLUMNCOUNTVALID**(*col* [, *rowStartIndex*, *rowEndIndex*]);

パッケージ変数

package
データグリッドパッケージ変数の名前を指定します。

必須引数

col
名前またはインデックス値を使用して列を指定します。

オプション引数

rowStartIndex
操作を開始する行番号を指定します。

rowEndIndex
操作を終了する行番号を指定します。

COLUMNDELETE メソッド

指定されたデータグリッドから指定された列を削除します。

カテゴリ: 作成、更新、または削除

返されるデータの種類: INTEGER

注: このメソッドは、列が削除された場合は 0 (ゼロ)を返し、エラーが発生した場合はゼロ以外の値を返します。

構文

package.**COLUMNDELETE**(*col*);

パッケージ変数

package
データグリッドパッケージ変数の名前を指定します。

必須引数

col
名前またはインデックス値を使用して列を指定します。

COLUMNEXISTS メソッド

指定された列が存在する場合は 1 (真)を返します。それ以外の場合は、0 (偽)を返します。

カテゴリ: 作成、更新、または削除

返されるデータの種類: INTEGER

注: このメソッドは、列が削除された場合は 0 (ゼロ)を返し、エラーが発生した場合はゼロ以外の値を返します。

構文

package.**COLUMNEXISTS**(*colName*);

パッケージ変数

package
データグリッドパッケージ変数の名前を指定します。

必須引数

colName
列の名前を指定します。

COLUMNFREQ 演算子

指定された列の重複しない値の数を返します。

カテゴリ:	統計
返されるデータの種類:	INTEGER

構文

package.**COLUMNFREQ**(*col* [, *rowStartIndex*, *rowEndIndex*]);

パッケージ変数

package
データグリッドパッケージ変数の名前を指定します。

必須引数

col
名前またはインデックス値を使用して列を指定します。

オプション引数

rowStartIndex
操作を開始する行番号を指定します。

rowEndIndex
操作を終了する行番号を指定します。

COLUMNINDEX メソッド

指定された列名の順序位置を返します。

返されるデータの種類: INTEGER

注: エラーが発生した場合、または列名が存在しない場合、戻り値は **Missing-Value** です。

構文

package.**COLUMNIDEX**(*colName*);

パッケージ変数

package
データグリッドパッケージ変数の名前を指定します。

必須引数

colName
列の名前を指定します。

COLUMNISNUMERIC メソッド

指定された列が数値データ型の場合、1 (真)を返します。それ以外の場合は、0 (偽)を返します。

返されるデータの種類: INTEGER

構文

package.**COLUMNISNUMERIC**(*col*);

パッケージ変数

package
データグリッドパッケージ変数の名前を指定します。

必須引数

col
名前またはインデックス値を使用して、列の名前を指定します。

COLUMNMAX メソッド

指定された条件の最大値を返します。

返されるデータの種類: DECIMAL

注: エラーが発生した場合、戻り値は **Missing-Value** です。

構文

package.**COLUMNMAX**(*col* [, *rowStartIndex*, *rowEndIndex*]);

パッケージ変数

package
データグリッドパッケージ変数の名前を指定します。

必須引数

col
名前またはインデックス値を使用して、列の名前を指定します。

オプション引数

rowStartIndex
操作を開始する行番号を指定します。

rowEndIndex
操作を終了する行番号を指定します。

COLUMNMAXBOOL メソッド

指定された条件の最大ブール値を返します。

返されるデータの種類: BOOLEAN

注: エラーが発生した場合、戻り値は **Missing-Value** です。

構文

package.**COLUMNMAXBOOL**(*col* [, *rowStartIndex*, *rowEndIndex*]);

パッケージ変数

package
　データグリッドパッケージ変数の名前を指定します。

必須引数

col
　名前またはインデックス値を使用して、列の名前を指定します。

オプション引数

rowStartIndex
　操作を開始する行番号を指定します。

rowEndIndex
　操作を終了する行番号を指定します。

COLUMNMAXINT メソッド

指定された条件の最大整数値を返します。

返されるデータの種類: INTEGER

注: エラーが発生した場合、戻り値は **Missing-Value** です。

構文

package.**COLUMNMAXINT**(*col* [, *rowStartIndex*, *rowEndIndex*]);

パッケージ変数

package
　データグリッドパッケージ変数の名前を指定します。

必須引数

col
　名前またはインデックス値を使用して、列の名前を指定します。

オプション引数

rowStartIndex
　操作を開始する行番号を指定します。

rowEndIndex
　操作を終了する行番号を指定します。

COLUMNMAXLONG 演算子

指定された条件の最大 long 値を返します。

返されるデータの種類: 64-BIT INTEGER

注: エラーが発生した場合、戻り値は **Missing-Value** です。

構文

package.**COLUMNMAXLONG**(*col* [, *rowStartIndex*, *rowEndIndex*]);

パッケージ変数

package
データグリッドパッケージ変数の名前を指定します。

必須引数

col
名前またはインデックス値を使用して、列の名前を指定します。

オプション引数

rowStartIndex
操作を開始する行番号を指定します。

rowEndIndex
操作を終了する行番号を指定します。

COLUMNMAXSTRING メソッド

指定された条件の最大文字列値を返します。

返されるデータの種類: STRING

注: エラーが発生した場合、戻り値は **Missing-Value** です。

構文

package.**COLUMNMAXSTRING**(*col* [, *rowStartIndex*, *rowEndIndex*]);

パッケージ変数

package
　データグリッドパッケージ変数の名前を指定します。

必須引数

col
　名前またはインデックス値を使用して、列の名前を指定します。

オプション引数

rowStartIndex
　操作を開始する行番号を指定します。

rowEndIndex
　操作を終了する行番号を指定します。

COLUMNMEAN メソッド

指定された条件の平均値を返します。

返されるデータの種類: DECIMAL

注: エラーが発生した場合、戻り値は **Missing-Value** です。

構文

package.**COLUMNMEAN**(*col* [, *rowStartIndex*, *rowEndIndex*]);

パッケージ変数

package
　データグリッドパッケージ変数の名前を指定します。

必須引数

col
　名前またはインデックス値を使用して、列の名前を指定します。

オプション引数

rowStartIndex
　操作を開始する行番号を指定します。

rowEndIndex
　操作を終了する行番号を指定します。

COLUMNMEDIAN メソッド

指定された条件の中央値を返します。

返されるデータの種類: DECIMAL

注: エラーが発生した場合、戻り値は **Missing-Value** です。

構文

package.**COLUMNMEDIAN**(*col* [, *rowStartIndex*, *rowEndIndex*]);

パッケージ変数

package
データグリッドパッケージ変数の名前を指定します。

必須引数

col
名前またはインデックス値を使用して、列の名前を指定します。

オプション引数

rowStartIndex
操作を開始する行番号を指定します。

rowEndIndex
操作を終了する行番号を指定します。

COLUMNMIN メソッド

指定された条件の最小値を返します。

返されるデータの種類: DECIMAL

注: エラーが発生した場合、戻り値は **Missing-Value** です。

構文

package.**COLUMNMIN**(*col* [, *rowStartIndex*, *rowEndIndex*]);

パッケージ変数

package
データグリッドパッケージ変数の名前を指定します。

必須引数

col
名前またはインデックス値を使用して、列の名前を指定します。

オプション引数

rowStartIndex
操作を開始する行番号を指定します。

rowEndIndex
操作を終了する行番号を指定します。

COLUMNMINBOOL メソッド

指定された条件の最小ブール値を返します。

返されるデータの種類: BOOLEAN

注: エラーが発生した場合、戻り値は **Missing-Value** です。

構文

package.**COLUMNMINBOOL**(*col* [, *rowStartIndex*, *rowEndIndex*]);

パッケージ変数

package
データグリッドパッケージ変数の名前を指定します。

必須引数

col
名前またはインデックス値を使用して、列の名前を指定します。

オプション引数

rowStartIndex
操作を開始する行番号を指定します。

rowEndIndex
操作を終了する行番号を指定します。

COLUMNMININT メソッド

指定された条件の最小整数値を返します。

返されるデータの種類: INTEGER

注: エラーが発生した場合、戻り値は **Missing-Value** です。

構文

package.**COLUMNMININT**(*col* [, *rowStartIndex*, *rowEndIndex*]);

パッケージ変数

package
データグリッドパッケージ変数の名前を指定します。

必須引数

col
名前またはインデックス値を使用して、列の名前を指定します。

オプション引数

rowStartIndex
操作を開始する行番号を指定します。

rowEndIndex
操作を終了する行番号を指定します。

COLUMNMINLONG メソッド

指定された条件の最小 long 値を返します。

返されるデータの種類: 64-BIT INTEGER

注: エラーが発生した場合、戻り値は **Missing-Value** です。

構文

package.**COLUMNMINLONG**(*col* [, *rowStartIndex*, *rowEndIndex*]);

パッケージ変数

package
　データグリッドパッケージ変数の名前を指定します。

必須引数

col
　名前またはインデックス値を使用して、列の名前を指定します。

オプション引数

rowStartIndex
　操作を開始する行番号を指定します。

rowEndIndex
　操作を終了する行番号を指定します。

COLUMNMINSTRING メソッド

指定された条件の最小文字列値を返します。

返されるデータの種類: STRING

注: エラーが発生した場合、戻り値は **Missing-Value** です。

構文

package.**COLUMNMINSTRING**(*col* [, *rowStartIndex*, *rowEndIndex*]);

パッケージ変数

package
　データグリッドパッケージ変数の名前を指定します。

必須引数

col
　名前またはインデックス値を使用して、列の名前を指定します。

オプション引数

rowStartIndex
　操作を開始する行番号を指定します。

rowEndIndex
　操作を終了する行番号を指定します。

COLUMNNAME メソッド

1 から始まる順序位置にある列の名前を返します。

返されるデータの種類: VARCHAR

注: 指定された順序位置が存在しない場合は、null DS2 文字列が返されます。

構文

package.**COLUMNNAME**(*colIndex*);

パッケージ変数

package
データグリッドパッケージ変数の名前を指定します。

必須引数

colIndex
データグリッド内の列のインデックス値を指定します。

COLUMNRENAME メソッド

指定された列の名前を変更します。

返されるデータの種類: INTEGER

注: このメソッドは、列の名前が変更された場合は 0 (ゼロ)を返し、エラーが発生した場合はゼロ以外の値を返します。

構文

package.**COLUMNRENAME**(*col*, *newName*)

パッケージ変数

package
データグリッドパッケージ変数の名前を指定します。

必須引数

col
　名前またはインデックス値を使用して列を指定します。

newName
　新しい名前を指定します。

COLUMNSTDDEV メソッド

指定された列の値の標準偏差を返します。

返されるデータの種類: DECIMAL

注: エラーが発生した場合、戻り値は **Missing-Value** です。

構文

package.**COLUMNSTDDEV**(*col* [, *rowStartIndex*, *rowEndIndex*]);

パッケージ変数

package
　データグリッドパッケージ変数の名前を指定します。

必須引数

col
　名前またはインデックス値を使用して、列の名前を指定します。

オプション引数

rowStartIndex
　操作を開始する行番号を指定します。

rowEndIndex
　操作を終了する行番号を指定します。

COLUMNSUM メソッド

指定された列の DOUBLE 値の合計を返します。

返されるデータの種類: DECIMAL

注: エラーが発生した場合、戻り値は **Missing-Value** です。

構文

package.**COLUMNSUM**(*col*);

パッケージ変数

package
データグリッドパッケージ変数の名前を指定します。

必須引数

col
名前またはインデックス値を使用して、DOUBLE 値を含む列を指定します。

例

この例では、dataGrid_A には 2 つの列があります。一方の列 col_a には DOUBLE 値が含まれています。もう一方の列 col_b には STRING 値が含まれます。したがって、この例では、col_a で COLUMNSUM を使用できます。

表 14.8 dataGrid_A

col_a	col_b
-11.5	city
10.1	name
19.3	address

次の操作が実行されます。

列名の使用:

```
double_sum = dataGrid_A.COLUMNSUM('col_a');
```

列インデックスの使用:

```
double_sum = dataGrid_A.COLUMNSUM('1');
```

COLUMNSUMLONG メソッド

指定された列の LONG 値の合計を返します。

返されるデータの種類: 64-BIT INTEGER

注: エラーが発生した場合、戻り値は **Missing-Value** です。

構文

package.**COLUMNSUMLONG**(*col*);

パッケージ変数

package
データグリッドパッケージ変数の名前を指定します。

必須引数

col
名前またはインデックス値を使用して、LONG 値を含む列を指定します。

例

この例では、dataGrid_A には 2 つの列があります。一方の列 col_a には LONG 値が含まれています。もう一方の列 col_b には STRING 値が含まれます。したがって、この例では、col_a で COLUMNSUMLONG を使用できます。

表 14.9 dataGrid_A

col_a	col_b
1586422	city
65246574	name
946672166	address

次の操作が実行されます。

列名の使用:

```
long_sum = dataGrid_A.COLUMNSUMLONG('col_a');
```

列インデックスの使用:

```
long_sum = dataGrid_A.COLUMNSUMLONG('1');
```

COLUMNTYPENAME メソッド

指定された列名の型名を返します。

返されるデータの種類: VARCHAR

注: 指定された列名が存在しない場合は、null DS2 文字列が返されます。

構文

package.**COLUMNTYPENAME**(*colName*);

パッケージ変数

package
データグリッドパッケージ変数の名前を指定します。

必須引数

colName
列の名前を指定します。

COPY メソッド: データグリッドパッケージ

ソースデータグリッドの複製をターゲットデータグリッドに作成します。

返されるデータの種類: INTEGER

注: このメソッドは、ターゲットデータグリッドにコピーされた行数を返します。エラーが発生した場合、このメソッドは-1 を返します。

構文

package.**COPY**(*dgSource*);

パッケージ変数

package
データグリッドパッケージ変数の名前を指定します。

必須引数

dgSource
ソースデータグリッドの変数名を指定します。

DECLARE PACKAGE ステートメント: データグリッドパッケージ

パッケージ変数を作成し、データグリッドパッケージのインスタンスを作成できるようにします。

構文

形式 1: **DECLARE PACKAGE DATAGRID** *variable*();

形式 2: **DECLARE PACKAGE DATAGRID** *variable*(*name*);

パッケージ変数

variable
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

name
インスタンスの参照に使用できる内部名を指定します。

詳細

DS2 パッケージは、特定のインスタンスを構築して他の DS2 プログラムで使用できる変数とメソッドのコレクションです。

ネイティブデータグリッドオブジェクトを作成または管理するには、データグリッドパッケージを使用します。データグリッドパッケージは、DS2 プログラム用に事前定義されています。

パッケージが宣言されると、パッケージのインスタンスを参照できる変数が作成されます。複数のパッケージ変数を作成でき、単一の DECLARE PACKAGE ステートメントで複数のパッケージインスタンスを構築できます。各パッケージインスタンスは、パッケージの完全に別個のコピーを表します。

データグリッドパッケージのインスタンスを構築する方法は 2 つあります。

- DECLARE PACKAGE ステートメントを_NEW_演算子と一緒に使用します。

```
declare package datagrid dg;
dg = _new_ datagrid();
```

- DECLARE PACKAGE ステートメントをそのコンストラクター構文と一緒に使用します。

```
declare package datagrid dg();
```

関連項目

- “パッケージのコンストラクターとデストラクター” (*SAS DS2 プログラマガイド*)
- “データグリッドパッケージの使用” (*SAS DS2 プログラマガイド*)

演算子:

- “_NEW_ 演算子: データグリッドパッケージ” (1608 ページ)

DESERIALIZE メソッド

指定された JSON 文字列からデータグリッドを作成します。

返されるデータの種類: INTEGER

注:
このメソッドは、データグリッドが作成されている場合は 0 (ゼロ)を返し、作成されていない場合はゼロ以外の値を返します。
メタデータが JSON 文字列で提供されていない場合、列のデータ型は、JSON 処理で識別された最初の型(string、Boolean、long、decimal)から推測されます。列のすべてのアイテムが null の場合、型は decimal に変換されます。

構文

package.**DESERIALIZE**(*JSON_string*);

パッケージ変数

package
データグリッドパッケージ変数の名前を指定します。

必須引数

JSON_string
データグリッドのデータを含む JSON 文字列を指定します。単一引用符で囲まれたリテラル値を指定するか、JSON 文字列に評価される変数を指定できます。データグリッド JSON 形式の詳細については、"データグリッドの JSON 表現について" (*SAS DS2 プログラマガイド*)を参照してください。

データグリッドを表す JSON 文字列には、次のいずれかを含めることができます。

- メタデータとデータの両方
- メタデータのみ
- データのみ

column1
column2
データグリッドの各列の列名を指定します。

row1_data
row2_data
データグリッドの各行のデータを指定します。行の各列の値はカンマ(,)で区切ります。文字値は二重引用符で囲みます。

例

null 以外の最初のデータ行によって、列の型が決まります。次の例では、最初の行はすべて null であり、型は推測されません。2 行目では、3 列目を除くすべての列の型が推測されます。列 3 のすべてのデータが null であるため、この列の型は 10 進数と推測されます。

次に、すべての null 列を含む JSON コンテンツを示します。

```
json:: [{"data":[[null,null,null,null,null],["Str_01",true,null,15,1.01],
["Str_02",false,null,77,2.02],["Str_03",false,null,93,3.03],
["Str_04",true,null,15,4.04],["Str_05",false,null,null,5.05]]}]
```

結果のデータグリッドは次のとおりです。

```
row[0000]: string(mv) boolean(mv) decimal(mv) long(m1v) decimal(mv)
row[0001]: string(Str_01) boolean(true) decimal(mv) long(15) decimal(1.01)
row[0002]: string(Str_02) boolean(false) decimal(mv) long(77) decimal(2.02)
row[0003]: string(Str_03) boolean(false) decimal(mv) long(93) decimal(3.03)
row[0004]: string(Str_04) boolean(true) decimal(mv) long(15) decimal(4.04)
row[0005]: string(Str_05) boolean(false) decimal(mv) long(mv) decimal(5.05)
```

DISTINCTCOUNT メソッド

データグリッド内の一意の行数を返します。

返されるデータの種類: INTEGER

注: エラーが発生した場合、戻り値は **Missing-Value** です。

構文

package.**DISTINCTCOUNT**();

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

FILTEREDCOMPARE メソッド

データグリッドの特定の列と行の値の比較結果を返します。

返されるデータの種類: Integer

注: エラーが発生した場合、戻り値は Missing-Value です。

構文

package.**FILTEREDCOMPARE**(*cmpColName*, *operator*, *cmpValue*, *cmpRowIndex*);

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

cmpColName
列名を指定します。

operator
"比較演算子" (*SAS DS2 プログラマガイド*)に示されている演算子の 1 つを指定します。これらの演算子のいずれかに評価される文字変数の名前を指定するか、単一引用符で囲んだ演算子を指定できます。

cmpValue
cmpColName と *cmpRowIndex* の交差の値と比較する値を指定します。

数値、単一引用符で囲まれた文字列、変数の名前、または式の名前を指定できます。指定する値は、*cmpColName* と *cmpRowIndex* の交差の値と同じデータ型として評価される必要があります。

null 値と欠損値を検索するには、次の点に注意してください。

- 文字列の場合、null は null 値のみに一致します。欠損(空の文字列またはすべての空白文字を含む文字列)は、欠損値に一致します。
- 他のすべての型では、null 値は存在しません。*cmpValue* に null を指定した場合、比較演算で欠損値として扱われます。

cmpRowIndex
行のインデックス値を指定します。インデックス値は、正の整数(または正の整数に評価される)である必要があります。

FILTEREDGET メソッド

比較演算が真と評価されたソース列の最初の行の値を返します。

返されるデータの種類: INTEGER、DECIMAL

注: エラーが発生した場合、戻り値は Missing-Value です。

構文

package.**FILTEREDGET**(*cmpCol*, *operator*, *cmpValue*, *srcCol* [, *rowStartIndex*, *rowEndIndex*]);

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

cmpCol
名前またはインデックス値を使用して列を指定し、その値を *cmpValue* と比較します。

operator
"比較演算子" (*SAS DS2 プログラマガイド*)に示されている演算子の 1 つを指定します。これらの演算子のいずれかに評価される文字変数の名前を指定するか、単一引用符で囲んだ演算子を指定できます。

cmpValue
cmpCol の値と比較する値を指定します。数値、単一引用符で囲まれた文字列、変数の名前、または式の名前を指定できます。指定する値は *cmpCol* であるか、または *cmpCol* と同じデータ型に評価される必要があります。

null 値と欠損値を検索するには、次の点に注意してください。

- 文字列の場合、null は null 値のみに一致します。欠損(空の文字列またはすべての空白文字を含む文字列)は、欠損値に一致します。
- 他のすべての型では、null 値は存在しません。*cmpValue* に null を指定した場合、比較演算で欠損値として扱われます。

srcCol
返す値を含む列の名前またはインデックス値を指定します。列名を指定するには、文字列値を使用します。列インデックスを指定するには、数値を使用します。インデックス値は、正の整数(または正の整数に評価される)である必要があります。

オプション引数

rowStartIndex
操作を開始する行番号を指定します。

rowEndIndex
操作を終了する行番号を指定します。

FILTEREDGETBYFILTER メソッド

比較演算が真と評価されたソース列の最初の行の値を返します。比較演算子は、フィルターを使用して定義されます。

返されるデータの種類: INTEGER、DECIMAL

注: エラーが発生した場合、戻り値は Missing-Value です。
このメソッドサポートは、2025.02 で開始されます。

構文

形式 1: *package*.**FILTEREDGETBYFILTER**(*filter*, *srcColName* [, *varList*]);

形式 2: *package*.**FILTEREDGETBYFILTER**(*filter*, *srcColName*, *rowStartIndex*, *rowEndIndex* [, *varList*]);

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

filter
行の選択基準を指定します。詳細については、“フィルター構文について” (*SAS DS2 プログラマガイド*)を参照してください。

srcColName
返す値を含む列の名前を指定します。

rowStartIndex
操作を開始する行番号を指定します。

rowEndIndex
操作を終了する行番号を指定します。

オプション引数

varList
フィルター内のパラメーター化された値を解決するために使用する DS2 変数のセットを含む DS2 varlist を指定します。パラメーター解決コードでは、DS2 varList 変数のデータ型が、関連付けられた列のデータ型と一致していると想定されます。

FILTEREDGETIN メソッド

比較演算が真と評価されたソース列の最初の行の値を返します。複数の値が演算の比較子として使用されます。これらの値は、*inDatagrid* 引数に指定されたデータグリッドの最初の列で定義されます。

返されるデータの種類: STRING、DECIMAL

注: エラーが発生した場合、戻り値は **Missing-Value** です。

構文

package.**FILTEREDGETIN**(*cmpColumn*, *cmpOperator*, *inDatagrid*, *srcCol* [, *rowStartIndex*, *rowEndIndex*]);

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

cmpColumn
名前またはインデックス値を使用して、比較演算を実行する列を指定します。

cmpOperator
比較演算で使用する演算子を指定します。EQ (等しい)または NEQ (等しくない)のいずれかです。

inDatagrid
比較演算で使用する比較子値が最初の列に含まれるデータグリッドを指定します。

scrCol
値を知りたい列の名前を指定します。

オプション引数

rowStartIndex
操作を開始する行番号を指定します。

rowEndIndex
操作を終了する行番号を指定します。

FILTEREDGETINDEX メソッド

比較演算が真と評価された最初の行のインデックスを返します。

返されるデータの種類: INTEGER

注: 条件が見つからない場合、このメソッドは 0 (ゼロ)を返します。

構文

package.**FILTEREDGETINDEX**(*cmpCol*, *operator*, *cmpValue* [, *rowStartIndex*, *rowEndIndex*]);

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

cmpCol
名前またはインデックス値を使用して列を指定し、その値を *cmpValue* と比較します。

operator
"比較演算子" (*SAS DS2 プログラマガイド*)に示されている演算子の 1 つを指定します。これらの演算子のいずれかに評価される文字変数の名前を指定するか、単一引用符で囲んだ演算子を指定できます。

cmpValue
cmpCol の値と比較する値を指定します。数値、単一引用符で囲まれた文字列、変数の名前、または式の名前を指定できます。指定する値は cmpCol であるか、または *cmpCol* と同じデータ型に評価される必要があります。

null 値と欠損値を検索するには、次の点に注意してください。

- 文字列の場合、null は null 値のみに一致します。欠損(空の文字列またはすべての空白文字を含む文字列)は、欠損値に一致します。
- 他のすべての型では、null 値は存在しません。*cmpValue* に null を指定した場合、比較演算で欠損値として扱われます。

filter
行の選択基準を指定します。詳細については、"フィルター構文について" (*SAS DS2 プログラマガイド*)を参照してください。

オプション引数

rowStartIndex
操作を開始する行番号を指定します。

rowEndIndex
操作を終了する行番号を指定します。

FILTEREDGETINDEXBYFILTER メソッド

比較演算が真と評価された最初の行のインデックスを返します。比較演算子は、フィルターを使用して定義されます。

返されるデータの種類: INTEGER

注: 条件が見つからない場合、このメソッドは 0 (ゼロ)を返します。
このメソッドサポートは、2025.02 で開始されます。

構文

形式 1: *package*.**FILTEREDGETINDEXBYFILTER**(*filter* [, *varList*]);

形式 2: *package*.**FILTEREDGETINDEXBYFILTER**(*filter*, *rowStartIndex*, *rowEndIndex* [, *varList*]);

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

filter
行の選択基準を指定します。詳細については、“フィルター構文について” (*SAS DS2 プログラマガイド*)を参照してください。

rowStartIndex
操作を開始する行番号を指定します。

rowEndIndex
操作を終了する行番号を指定します。

オプション引数

varList
フィルター内のパラメーター化された値を解決するために使用する DS2 変数のセットを含む DS2 varlist を指定します。パラメーター解決コードでは、DS2 varList 変数のデータ型が、関連付けられた列のデータ型と一致していると想定されます。

FILTEREDGETINDEXIN メソッド

比較演算が真と評価された最初の行のインデックスを返します。複数の値が演算の比較子として使用されます。これらの値は、*inDatagrid* 引数に指定されたデータグリッドの最初の列で定義されます。

返されるデータの種類: STRING、DECIMAL

注: 条件が見つからない場合、このメソッドは 0 (ゼロ)を返します。

構文

package.**FILTEREDGETINDEXIN**(*cmpColumn*, *cmpOperator*, *inDatagrid* [, *rowStartIndex*, *rowEndIndex*]);

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

cmpColumn
名前またはインデックス値を使用して、比較演算を実行する列を指定します。

cmpOperator
比較演算で使用する演算子を指定します。EQ (等しい)または NEQ (等しくない)のいずれかです。

inDatagrid
比較演算で使用する比較子値が最初の列に含まれるデータグリッドを指定します。

オプション引数

rowStartIndex
操作を開始する行番号を指定します。

rowEndIndex
操作を終了する行番号を指定します。

FILTEREDSET メソッド

比較が真と評価された場合、指定された列の値を指定された値に設定します。

返されるデータの種類: INTEGER

注: このメソッドは、指定された値が設定されている場合は 0 (ゼロ)を返し、指定された値が設定されていない場合はゼロ以外の値を返します。

構文

package.**FILTEREDSET**(*cmpCol*, *operator*, *cmpValue*, *rowIndex*, *setCol*, *setValue*);

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

cmpCol
名前またはインデックス値を使用して列を指定し、その値を *cmpValue* と比較します。

operator
"比較演算子" (*SAS DS2 プログラマガイド*)に示されている演算子の 1 つを指定します。これらの演算子のいずれかに評価される文字変数の名前を指定するか、単一引用符で囲んだ演算子を指定できます。

cmpValue
cmpCol の値と比較する値を指定します。数値、単一引用符で囲まれた文字列、変数の名前、または式の名前を指定できます。指定する値は cmpCol であるか、または *cmpCol* と同じデータ型に評価される必要があります。

null 値と欠損値を検索するには、次の点に注意してください。

- 文字列の場合、null は null 値のみに一致します。欠損(空の文字列またはすべての空白文字を含む文字列)は、欠損値に一致します。
- 他のすべての型では、null 値は存在しません。cmpValue に null を指定した場合、比較演算で欠損値として扱われます。

rowIndex
データグリッドの行番号を指定します。

setCol
指定された値が割り当てられる列の名前またはインデックス値を指定します。

setValue
割り当てる値を指定します。

FILTEREDSETBYFILTER メソッド

比較が真と評価された場合、指定された列の値を指定された値に設定します。比較演算子は、フィルターを使用して定義されます。

返されるデータの種類: INTEGER

注: このメソッドは、指定された値が設定されている場合は 0 (ゼロ)を返し、指定された値が設定されていない場合はゼロ以外の値を返します。
このメソッドサポートは、2025.02 で開始されます。

構文

package.**FILTEREDSETBYFILTER**(*filter*, *rowIndex*, *setColName*, *setValue* [, *varList*]);

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

filter
行の選択基準を指定します。詳細については、“フィルター構文について” (*SAS DS2 プログラマガイド*)を参照してください。

rowIndex
データグリッドの行番号を指定します。

setColName
指定された値が割り当てられる列の名前を指定します。

setValue
割り当てる値を指定します。

オプション引数

varList
フィルター内のパラメーター化された値を解決するために使用する DS2 変数のセットを含む DS2 varlist を指定します。パラメーター解決コードでは、DS2 varList 変数のデータ型が、関連付けられた列のデータ型と一致していると想定されます。

FILTEREDSETALL メソッド

指定された比較が真と評価された場合、指定された列のセルを指定された値に設定します。

注: このメソッドは、データグリッドで変更された行数を返します。エラーが発生した場合、戻り値は **Missing-Value** です。

構文

package.**FILTEREDSETALL**(*cmpCol*, *operator*, *cmpValue*, *setCol*, *setValue*);

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

cmpCol
名前またはインデックス値を使用して列を指定し、その値を *cmpValue* と比較します。

operator
"比較演算子" (*SAS DS2 プログラマガイド*)に示されている演算子の 1 つを指定します。これらの演算子のいずれかに評価される文字変数の名前を指定するか、単一引用符で囲んだ演算子を指定できます。

cmpValue
cmpCol の値と比較する値を指定します。数値、単一引用符で囲まれた文字列、変数の名前、または式の名前を指定できます。指定する値は *cmpCol* であるか、または *cmpCol* と同じデータ型に評価される必要があります。

null 値と欠損値を検索するには、次の点に注意してください。

- 文字列の場合、null は null 値のみに一致します。欠損(空の文字列またはすべての空白文字を含む文字列)は、欠損値に一致します。

- 他のすべての型では、null 値は存在しません。*cmpValue* に null を指定した場合、比較演算で欠損値として扱われます。

setCol
指定された値が割り当てられる列の名前またはインデックス値を指定します。

注: 形式 2 は列インデックス値の指定をサポートしていません。

setValue
割り当てる値を指定します。

FILTEREDSETALLBYFILTER メソッド

指定された比較が真と評価された場合、指定された列のセルを指定された値に設定します。比較演算子は、フィルターを使用して定義されます。

注: このメソッドは、データグリッドで変更された行数を返します。エラーが発生した場合、戻り値は **Missing-Value** です。
このメソッドサポートは、2025.02 で開始されます。

構文

package.**FILTEREDSETALLBYFILTER**(*filter*, *setColName*, *setValue* [, *varList*]);

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

filter
行の選択基準を指定します。詳細については、“フィルター構文について” (*SAS DS2 プログラマガイド*)を参照してください。

setColName
指定された値が割り当てられる列の名前を指定します。

setValue
割り当てる値を指定します。

オプション引数

varList
フィルター内のパラメーター化された値を解決するために使用する DS2 変数のセットを含む DS2 varlist を指定します。パラメーター解決コードでは、DS2 varList 変数のデータ型が、関連付けられた列のデータ型と一致していると想定されます。

FULLJOIN メソッド

2 つのデータグリッドの完全結合を実行し、結合の結果をターゲットデータグリッドに入力します。

返されるデータの種類: INTEGER

注:
右テーブルと左テーブルの行ごとに、1 つの行がターゲットデータグリッドに書き込まれます。これは、テーブル間に一致がない場合でも当てはまります。左右のテーブルで一致する列名は、ターゲットデータグリッドで *columnName*_R に名前変更されます。

このメソッドは、結合が成功した場合は 0 (ゼロ)を返し、成功しなかった場合はゼロ以外の値を返します。

形式 2 のサポートは、2023.03 で開始されます。

構文

形式 1: *package*.**FULLJOIN**(*dgLeft*, *colLeft*, *dgRight*, *colRight*);

形式 2: *package*.**FULLJOIN**(*dgLeft*, *dgRight*, *dgJoin*);

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

dgLeft
左データグリッドの変数名を指定します。

colLeft
名前またはインデックス値を使用して、*dgLeft* のキー列を指定します。

dgRight
右データグリッドの変数名を指定します。

colRight
名前またはインデックス値を使用して、*dgRight* のキー列を指定します。

dgJoin
複数列の結合を定義するデータグリッドを指定します。*dgJoin* は、(string, string)または(long, long)の形式である必要があります。

最初の列 *dgJoin.col1* は、名前またはインデックス値のいずれかを使用して、*dgLeft* のキー列を指定します。

2 番目の列 *dgJoin.col1* は、名前またはインデックス値のいずれかを使用して、*dgRight* のキー列を指定します。

例

注: 完全結合を実行すると、データの並べ替え順序が、これらの例で示した並べ替え順序と異なる場合があります。

例 1: 完全結合

完全結合の例を次に示します。操作で使用されるソーステーブルは次のとおりです。

表 14.10　価格テーブル(price)

PRODUCT	PRICE	MATCH
Potatoes	1.001	l_1
Avocados	2.002	l_2
Kiwis	3.003	l_3
Onions	4.004	l_4
Melons	5.005	l_5
Oranges	6.006	l_6
Tomatoes	7.007	l_7

表 14.11　数量テーブル(quantity)

MATCH	PROD	QUANTITY
r_1	Melons	11
r_2	Squash	22
r_3	Kiwis	33
r_4	Potatoes	44
r_8	Broccoli	55
r_6	Avocados	66
r_8	Broccoli	77
r_5	Avocados	88

MATCH	PROD	QUANTITY
r_7	Onions	99
r_5	Pickles	100

この例では、次の操作が実行されます。

```
dataGrid_A.fullJoin(price, 'PRODUCT', quantity, 'PROD');
```

結果のデータグリッドは次のとおりです。

表 14.12　dataGrid_A

PRODUCT	PRICE	MATCH	MATCH_R	PROD	QUANTITY
Potatoes	1.001	l_1	r_4	Potatoes	44
Avocados	2.002	l_2	r_6	Avocados	66
Avocados	2.002	l_2	r_5	Avocados	88
Kiwis	3.003	l_3	r_3	Kiwis	33
Onions	4.004	l_4	r_7	Onions	99
Melons	5.005	l_5	r_1	Melons	11
Oranges	6.006	l_6	null	null	null
Tomatoes	7.007	l_7	null	null	null
null	null	null	r_2	Squash	22
null	null	null	r_8	Broccoli	55
null	null	null	r_8	Broccoli	77
null	null	null	r_5	Pickles	100

例 2: 複数列の完全結合

複数列の完全結合の例を次に示します。操作で使用されるソーステーブルは次のとおりです。

表 14.13　コストテーブル(cost)

PRODUCT	TYPE	COST	MATCH
Potatoes	Russet	1.001	l_1

PRODUCT	TYPE	COST	MATCH
Potatoes	Sweet	1.011	l_1.1
Avocados	Hass	2.002	l_2
Avocados	Reed	2.022	l_2.2
Kiwis	Golden	3.003	l_3
Kiwis	Purple	3.033	l_3.3
Onions	Vidalia	4.004	l_4
Onions	Yellow	4.044	l_4.4
Melons	Honey Dew	5.005	l_5
Melons	Watermelon	5.055	l_5.5
Oranges	Blood	6.006	l_6
Oranges	Navel	6.066	l_6.6
Tomatoes	Heirloom	7.007	l_7
Tomatoes	Beefsteak	7.077	l_7.7

表 14.14　金額テーブル(amount)

MATCH	PROD	TYPE	AMOUNT
r_1	Melons	Watermelon	11
r_2	Squash	Butternut	22
r_3	Kiwis	Purple	33
r_4	Potatoes	Russet	44
r_8	Broccoli	Blue Wind	55
r_6	Avocados	Reed	66
r_8	Broccoli	Eastern Magic	77
r_5	Avocados	Hass	88

MATCH	PROD	TYPE	AMOUNT
r_7	Onions	Vidalia	99
r_5	Pickles	Dill	100

表 14.15　結合列テーブル(dgJoinKeys)

leftColumnName	rightColumnName
PRODUCT	PROD
TYPE	TYPE

この例では、次の操作が実行されます。

```
dataGrid_B.fullJoin(cost, amount, dgJoinKeys);
```

結果のデータグリッドは次のとおりです。

表 14.16　dataGrid_B

PRODUCT	TYPE	COST	MATCH	MATCH_R	PROD	TYPE_R	AMOUNT
Potatoes	Russet	1.001	l_1	r_4	Potatoes	Russet	44
Potatoes	Sweet	1.011	l_1.1	null	null	null	null
Avocados	Hass	2.002	l_2	r_5	Avocados	Hass	88
Avocados	Reed	2.022	l_2.2	r_6	Avocados	Reed	66
Kiwis	Golden	3.003	l_3	null	null	null	null
Kiwis	Purple	3.033	l_3.3	r_3	Kiwis	Purple	33
Onions	Vidalia	4.004	l_4	r_7	Onions	Vidalia	99
Onions	Yellow	4.044	l_4.4	null	null	null	null
Melons	Honey Dew	5.005	l_5	null	null	null	null
Melons	Watermelon	5.055	l_5.5	r_1	Melons	Watermelon	11

PRODUCT	TYPE	COST	MATCH	MATCH_R	PROD	TYPE_R	AMOUNT
Oranges	Blood	6.006	l_6	null	null	null	null
Oranges	Navel	6.066	l_6.6	null	null	null	null
Tomatoes	Heirloom	7.007	l_7	null	null	null	null
Tomatoes	Beefsteak	7.077	l_7.7	null	null	null	null
null	null	null	null	r_2	Squash	Butternut	22
null	null	null	null	r_8	Broccoli	Blue Wind	55
null	null	null	null	r_8	Broccoli	Eastern Magic	77
null	null	null	null	r_5	Pickles	Dill	100

FULLJOINBYFILTER メソッド

2 つのデータグリッドの完全結合を実行し、結合の結果をターゲットデータグリッドに入力します。ターゲット列とソース行の基準は、列保持句と行フィルター定義を使って指定します。

返されるデータの種類: INTEGER

注:

右テーブルと左テーブルの行ごとに、1 つの行がターゲットデータグリッドに書き込まれます。これは、テーブル間に一致がない場合でも当てはまります。左右のテーブルで一致する列名は、ターゲットデータグリッドで *columnName*_R に名前変更されます。

このメソッドは、結合が成功した場合は 0 (ゼロ)を返し、成功しなかった場合はゼロ以外の値を返します。

このメソッドサポートは、2025.02 で開始されます。

構文

package.**FULLJOINBYFILTER**(*dgLeft*, *dgRight*, *keepColumns*, *filter* [, *varList*]);

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

dgLeft
左データグリッドの変数名を指定します。

dgRight
右データグリッドの変数名を指定します。

keepColumns
結果データグリッドに配置する列のリスト。詳細については、“列保持構文について” (*SAS DS2 プログラマガイド*)を参照してください。

filter
行の選択基準を指定します。詳細については、“フィルター構文について” (*SAS DS2 プログラマガイド*)を参照してください。

オプション引数

varList
フィルター内のパラメーター化された値を解決するために使用する DS2 変数のセットを含む DS2 varlist を指定します。パラメーター解決コードでは、DS2 varList 変数のデータ型が、関連付けられた列のデータ型と一致していると想定されます。

例

注: 完全結合を実行すると、データの並べ替え順序が、これらの例で示した並べ替え順序と異なる場合があります。

例 1: 完全結合

完全結合の例を次に示します。操作で使用されるソーステーブルは次のとおりです。

表 14.17　価格テーブル(price)

PRODUCT	PRICE	MATCH
Potatoes	1.001	l_1
Avocados	2.002	l_2
Kiwis	3.003	l_3

PRODUCT	PRICE	MATCH
Onions	4.004	l_4
Melons	5.005	l_5
Oranges	6.006	l_6
Tomatoes	7.007	l_7

表 14.18　数量テーブル(quantity)

MATCH	PROD	QUANTITY
r_1	Melons	11
r_2	Squash	22
r_3	Kiwis	33
r_4	Potatoes	44
r_8	Broccoli	55
r_6	Avocados	66
r_8	Broccoli	77
r_5	Avocados	88
r_7	Onions	99
r_5	Pickles	100

この例では、次の操作が実行されます。

```
dataGrid_A.fullJoinByFilter(price, quantity, '*', 'L.PRODUCT eq R.PROD');
```

結果のデータグリッドは次のとおりです。

表 14.19　dataGrid_A

PRODUCT	PRICE	MATCH	MATCH_R	PROD	QUANTITY
Potatoes	1.001	l_1	r_4	Potatoes	44
Avocados	2.002	l_2	r_6	Avocados	66

PRODUCT	PRICE	MATCH	MATCH_R	PROD	QUANTITY
Avocados	2.002	l_2	r_5	Avocados	88
Kiwis	3.003	l_3	r_3	Kiwis	33
Onions	4.004	l_4	r_7	Onions	99
Melons	5.005	l_5	r_1	Melons	11
Oranges	6.006	l_6	null	null	null
Tomatoes	7.007	l_7	null	null	null
null	null	null	r_2	Squash	22
null	null	null	r_8	Broccoli	55
null	null	null	r_8	Broccoli	77
null	null	null	r_5	Pickles	100

この例では、keepColumns に***** (アスタリスク)が指定されています。これにより、一致しない列名が結果のデータグリッドに含まれます。また、varList には **product** という変数があり、値 **Potatoes** が割り当てられています。

```
dataGrid_A.fullJoinByFilter(price, quantity, '*', 'L.PRODUCT eq R.PROD & L.PRODUCT eq ?',
[product]);
```

結果のデータグリッドは次のとおりです。

表 14.20　dataGrid_A

PRODUCT	PRICE	MATCH	MATCH_R	PROD	QUANTITY
Potatoes	1.001	l_1	r_4	Potatoes	44
Avocados	2.002	l_2	null	null	null
Kiwis	3.003	l_3	null	null	null
Onions	4.004	l_4	null	null	null
Melons	5.005	l_5	null	null	null
Oranges	6.006	l_6	null	null	null
Tomatoes	7.007	l_7	null	null	null
null	null	null	r_1	Melons	11
null	null	null	r_2	Squash	22
null	null	null	r_3	Kiwis	33

PRODUCT	PRICE	MATCH	MATCH_R	PROD	QUANTITY
null	null	null	r_8	Broccoli	55
null	null	null	r_6	Avocados	66
null	null	null	r_8	Broccoli	77
null	null	null	r_5	Avocados	88
null	null	null	r_7	Onions	99
null	null	null	r_5	Pickles	100

例 2: 複数列の完全結合

複数列の完全結合の例を次に示します。操作で使用されるソーステーブルは次のとおりです。

表 14.21　コストテーブル(cost)

PRODUCT	TYPE	COST	MATCH
Potatoes	Russet	1.001	l_1
Potatoes	Sweet	1.011	l_1.1
Avocados	Hass	2.002	l_2
Avocados	Reed	2.022	l_2.2
Kiwis	Golden	3.003	l_3
Kiwis	Purple	3.033	l_3.3
Onions	Vidalia	4.004	l_4
Onions	Yellow	4.044	l_4.4
Melons	Honey Dew	5.005	l_5
Melons	Watermelon	5.055	l_5.5
Oranges	Blood	6.006	l_6
Oranges	Navel	6.066	l_6.6
Tomatoes	Heirloom	7.007	l_7

PRODUCT	TYPE	COST	MATCH
Tomatoes	Beefsteak	7.077	l_7.7

表 14.22　金額テーブル(amount)

MATCH	PROD	TYPE	AMOUNT
r_1	Melons	Watermelon	11
r_2	Squash	Butternut	22
r_3	Kiwis	Purple	33
r_4	Potatoes	Russet	44
r_8	Broccoli	Blue Wind	55
r_6	Avocados	Reed	66
r_8	Broccoli	Eastern Magic	77
r_5	Avocados	Hass	88
r_7	Onions	Vidalia	99
r_5	Pickles	Dill	100

この例では、次の操作が実行されます。

```
dataGrid_B.fullJoinByFilter(cost, amount, '*', 'L.PRODUCT eq R.PROD & L.TYPE eq R.TYPE');
```

結果のデータグリッドは次のとおりです。

表 14.23　dataGrid_B

PRODUCT	TYPE	COST	MATCH	MATCH_R	PROD	TYPE_R	AMOUNT
Potatoes	Russet	1.001	l_1	r_4	Potatoes	Russet	44
Potatoes	Sweet	1.011	l_1.1	null	null	null	null
Avocados	Hass	2.002	l_2	r_5	Avocados	Hass	88

PRODUCT	TYPE	COST	MATCH	MATCH_R	PROD	TYPE_R	AMOUNT
Avocados	Reed	2.022	l_2.2	r_6	Avocados	Reed	66
Kiwis	Golden	3.003	l_3	null	null	null	null
Kiwis	Purple	3.033	l_3.3	r_3	Kiwis	Purple	33
Onions	Vidalia	4.004	l_4	r_7	Onions	Vidalia	99
Onions	Yellow	4.044	l_4.4	null	null	null	null
Melons	Honey Dew	5.005	l_5	null	null	null	null
Melons	Watermelon	5.055	l_5.5	r_1	Melons	Watermelon	11
Oranges	Blood	6.006	l_6	null	null	null	null
Oranges	Navel	6.066	l_6.6	null	null	null	null
Tomatoes	Heirloom	7.007	l_7	null	null	null	null
Tomatoes	Beefsteak	7.077	l_7.7	null	null	null	null
null	null	null	null	r_2	Squash	Butternut	22
null	null	null	null	r_8	Broccoli	Blue Wind	55
null	null	null	null	r_8	Broccoli	Eastern Magic	77
null	null	null	null	r_5	Pickles	Dill	100

FULLJOINCOUNT メソッド

完全結合操作によって生成される行数を返します。この操作の影響を受けるデータグリッドはありません。

返されるデータの種類: INTEGER

注: エラーが発生した場合、このメソッドは-1 を返します。
形式 2 のサポートは、2023.03 で開始されます。

構文

形式 1: *dgLeft*.**FULLJOINCOUNT**(*colLeft*, *dgRight*, *colRight*);

形式 2: *dgLeft*.**FULLJOINCOUNT**(*dgRight*, *dgJoin*);

パッケージ変数

dgLeft
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

colLeft
名前またはインデックス値を使用して、dgLeft のキー列を指定します。

dgRight
右データグリッドの変数名を指定します。

colRight
名前またはインデックス値を使用して、dgRight のキー列を指定します。

dgJoin
複数列の結合を定義するデータグリッドを指定します。*dgJoin* は、(string, string)または(long, long)の形式である必要があります。

最初の列 *dgJoin.col1* は、名前またはインデックス値のいずれかを使用して、*dgLeft* のキー列を指定します。

2 番目の列 *dgJoin.col1* は、名前またはインデックス値のいずれかを使用して、*dgRight* のキー列を指定します。

FULLJOINTCOUNTBYFILTER メソッド

完全結合操作によって生成される行数を返します。ソース行の基準は、行フィルター定義を使用して指定します。この操作の影響を受けるデータグリッドはありません。

返されるデータの種類: INTEGER

注: エラーが発生した場合、このメソッドは-1 を返します。
このメソッドサポートは、2025.02 で開始されます。

構文

dgLeft.**FULLJOINCOUNTBYFILTER**(*dgRight*, *filter* [, *varList*]);

パッケージ変数

dgLeft
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

dgRight
右データグリッドの変数名を指定します。

filter
行の選択基準を指定します。詳細については、“フィルター構文について” (*SAS DS2 プログラマガイド*)を参照してください。

オプション引数

varList
フィルター内のパラメーター化された値を解決するために使用する DS2 変数のセットを含む DS2 varlist を指定します。パラメーター解決コードでは、DS2 varList 変数のデータ型が、関連付けられた列のデータ型と一致していると想定されます。

GETBOOL メソッド

指定された行と列のセルのブール値を返します。

返されるデータの種類: INTEGER

注: エラーが発生した場合、戻り値は **Missing-Value** です。

構文

package.**GETBOOL**(*col*, *rowIndex*);

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

col
名前またはインデックス値を使用して列を指定します。

rowIndex
データグリッドの行番号を指定します。

GETDOUBLE メソッド

指定された行と列のセルの 10 進数値を返します。

返されるデータの種類: DECIMAL

注: エラーが発生した場合、戻り値は **Missing-Value** です。

構文

package.**GETDOUBLE**(*col*, *rowIndex*);

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

col
名前またはインデックス値を使用して列を指定します。

rowIndex
データグリッドの行番号を指定します。

GETINT メソッド

指定された行と列のセルの整数値を返します。

返されるデータの種類: INTEGER

注: エラーが発生した場合、戻り値は **Missing-Value** です。

構文

package.**GETINT**(*col*, *rowIndex*);

パッケージ変数

package
　データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

col
　名前またはインデックス値を使用して列を指定します。

rowIndex
　データグリッドの行番号を指定します。

GETLONG メソッド

指定された行と列のセルの long 値を返します。

返されるデータの種類: 64-BIT INTEGER

注: エラーが発生した場合、戻り値は **Missing-Value** です。

構文

package.**GETLONG**(*col*, *rowIndex*);

パッケージ変数

package
　データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

col
　名前またはインデックス値を使用して列を指定します。

rowIndex
　データグリッドの行番号を指定します。

GETSTRING メソッド

指定された行と列のセルの文字列値を返します。

返されるデータの種類: STRING

注: エラーが発生した場合、戻り値は **Missing-Value** です。

構文

package.**GETSTRING**(*col*, *rowIndex*);

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

col
名前またはインデックス値を使用して列を指定します。

rowIndex
データグリッドの行番号を指定します。

GETVALUE メソッド

指定された行と列のセルの値を返します。

返されるデータの種類: STRING

注: 受け取る変数の型が double または decimal の場合、DS2 は返された文字列値を 10 進数に変換します。
エラーが発生した場合、戻り値は **Missing-Value** です。

構文

package.**GETVALUE**(*col*, *rowIndex*);

パッケージ変数

package
　データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

col
　名前またはインデックス値を使用して列を指定します。

rowIndex
　データグリッドの行番号を指定します。

INNERJOIN メソッド

2 つのデータグリッドの内部結合を実行し、結合の結果をターゲットデータグリッドに入力します。

注:　結合基準に一致する行のみがターゲットデータグリッドに追加されます。
　このメソッドは、結合が成功した場合は 0 (ゼロ)を返し、成功しなかった場合はゼロ以外の値を返します。
　形式 2 のサポートは、2023.03 で開始されます。

構文

形式 1:　*package*.**INNERJOIN**(*dgLeft*, *colLeft*, *dgRight*, *colRight*);

形式 2:　*package*.**INNERJOIN**(*dgLeft*, *dgRight*, *dgJoin*);

パッケージ変数

package
　データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

dgLeft
　左データグリッドの変数名を指定します。

colLeft
　名前またはインデックス値を使用して、*dgLeft* のキー列を指定します。

dgRight
　右データグリッドの変数名を指定します。

colRight
　名前またはインデックス値を使用して、*dgRight* のキー列を指定します。

dgJoin
　複数列の結合を定義するデータグリッドを指定します。*dgJoin* は、(string, string)または(long, long)の形式である必要があります。

最初の列 *dgJoin.col1* は、名前またはインデックス値のいずれかを使用して、*dgLeft* のキー列を指定します。

2 番目の列 *dgJoin.col1* は、名前またはインデックス値のいずれかを使用して、*dgRight* のキー列を指定します。

例

例 1: 内部結合

内部結合の例を次に示します。この例では、表 14.24 と表 14.25 (1577 ページ)に示されているソーステーブルを 使用します。

この例では、次の操作が実行されます。

```
dataGrid_A.innerJoin(price, 'PRODUCT', quantity, 'PROD');
```

結果のデータグリッドは次のとおりです。

表 14.24　dataGrid_A

PRODUCT	PRICE	MATCH	QUANTITY
Potatoes	1.001	l_1	44
Avocados	2.002	l_2	66
Avocados	2.002	l_2	88
Kiwis	3.003	l_3	33
Onions	4.004	l_4	99
Melons	5.005	l_5	11

例 2: 複数列の内部結合

複数列の内部結合の例を次に示します。この例では、表 14.27 (1578 ページ)、表 14.28 (1579 ページ)および表 14.29 (1580 ページ)に示されているソーステーブルを使用します。

この例では、次の操作が実行されます。

```
dataGrid_B.innerJoin(cost, amount, dgJoinKeys);
```

結果のデータグリッドは次のとおりです。

表 14.25 dataGrid_B

PRODUCT	TYPE	COST	MATCH	AMOUNT
Potatoes	Russet	1.001	l_1	44
Avocados	Hass	2.002	l_2	88
Avocados	Reed	2.022	l_2.2	66
Kiwis	Purple	3.033	l_3.3	33
Onions	Vidilia	4.004	l_4	99
Melons	Watermelon	5.055	l_5.5	11

INNERJOINBYFILTER メソッド

2 つのデータグリッドの内部結合を実行し、結合の結果をターゲットデータグリッドに入力します。ターゲット列とソース行の基準は、列保持句と行フィルター定義を使って指定します。

注: 結合基準に一致する行のみがターゲットデータグリッドに追加されます。

このメソッドは、結合が成功した場合は 0 (ゼロ)を返し、成功しなかった場合はゼロ以外の値を返します。

このメソッドサポートは、2025.02 で開始されます。

構文

package.**INNERJOINBYFILTER**(*dgLeft*, *dgRight*, *keepColumns*, *filter* [, *varList*]);

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

dgLeft
左データグリッドの変数名を指定します。

dgRight
右データグリッドの変数名を指定します。

keepColumns
結果データグリッドに配置する列のリスト。詳細については、*“列保持構文について” (SAS DS2 プログラマガイド)*を参照してください。

filter
行の選択基準を指定します。詳細については、“フィルター構文について” (*SAS DS2 プログラマガイド*)を参照してください。

オプション引数

varList
フィルター内のパラメーター化された値を解決するために使用する DS2 変数のセットを含む DS2 varlist を指定します。パラメーター解決コードでは、DS2 varList 変数のデータ型が、関連付けられた列のデータ型と一致していると想定されます。

例

例 1: 内部結合

内部結合の例を次に示します。この例では、表 14.24 と表 14.25 (1577 ページ)に示されているソーステーブルを 使用します。

この例では、次の操作が実行されます。

```
dataGrid_A.innerJoinByFilter(price, quantity, 'L.PRODUCT, L.PRICE, L.MATCH, R.QUANTITY',
'L.PRODUCT eq R.PROD');
```

結果のデータグリッドは次のとおりです。

表 14.26　dataGrid_A

PRODUCT	PRICE	MATCH	QUANTITY
Potatoes	1.001	l_1	44
Avocados	2.002	l_2	66
Avocados	2.002	l_2	88
Kiwis	3.003	l_3	33
Onions	4.004	l_4	99
Melons	5.005	l_5	11

この例では、keepColumns に* (アスタリスク)が指定されています。これにより、一致しない列名が結果のデータグリッドに含まれます。

```
dataGrid_A.innerJoinByFilter(price, quantity, '*', 'L.PRODUCT eq R.PROD');
```

結果のデータグリッドは次のとおりです。

表 14.27 dataGrid_A

PRODUCT	PRICE	MATCH	PROD	QUANTITY
Potatoes	1.001	l_1	Potatoes	44
Avocados	2.002	l_2	Avocados	66
Avocados	2.002	l_2	Avocados	88
Kiwis	3.003	l_3	Kiwis	33
Onions	4.004	l_4	Onions	99
Melons	5.005	l_5	Melons	11

この例では、varlist 引数には **product** という変数が割り当てられ、この変数には値 **Potatoes** が割り当てられています。

```
dataGrid_A.innerJoinByFilter(price, quantity, '*', 'L.PRODUCT eq R.PROD & L.PRODUCT
eq ?', [product]);
```

結果のデータグリッドは次のとおりです。

表 14.28 dataGrid_A

PRODUCT	PRICE	MATCH	PROD	QUANTITY
Potatoes	1.001	l_1	Potatoes	44

例 2: 複数列の内部結合

複数列の内部結合の例を次に示します。この例では、表 14.27 (1578 ページ)、表 14.28 (1579 ページ)および表 14.29 (1580 ページ)に示されているソーステーブルを使用します。

この例では、次の操作が実行されます。

```
dataGrid_B.innerJoinByFilter(cost, amount, 'L.PRODUCT, L.TYPE, L.COST, L.MATCH,
R.AMOUNT', 'L.PRODUCT eq R.PROD & L.TYPE eq R.TYPE');
```

結果のデータグリッドは次のとおりです。

表 14.29 dataGrid_B

PRODUCT	TYPE	COST	MATCH	AMOUNT
Potatoes	Russet	1.001	l_1	44
Avocados	Hass	2.002	l_2	88
Avocados	Reed	2.022	l_2.2	66

PRODUCT	TYPE	COST	MATCH	AMOUNT
Kiwis	Purple	3.033	l_3.3	33
Onions	Vidilia	4.004	l_4	99
Melons	Watermelon	5.055	l_5.5	11

次の例では、keepColumns に* (アスタリスク)が指定されています。これにより、一致しない列名が結果のデータグリッドに含まれます。

```
dataGrid_B.innerJoinByFilter(cost, amount, '*', 'L.PRODUCT eq R.PROD & L.TYPE eq R.TYPE');
```

表 14.30　dataGrid_B

PRODUCT	TYPE	COST	MATCH	PROD	AMOUNT
Potatoes	Russet	1.001	l_1	Potatoes	44
Avocados	Hass	2.002	l_2	Avocados	88
Avocados	Reed	2.022	l_2.2	Avocados	66
Kiwis	Purple	3.033	l_3.3	Kiwis	33
Onions	Vidilia	4.004	l_4	Onions	99
Melons	Watermelon	5.055	l_5.5	Melons	11

INNERJOINCOUNT メソッド

内部結合操作によって生成される行数を返します。データグリッドは影響を受けません。

返されるデータの種類: INTEGER

注: エラーが発生した場合、このメソッドは-1 を返します。
形式 2 のサポートは、2023.03 で開始されます。

構文

形式 1: *dgLeft*.**INNERJOINCOUNT**(*colLeft*, *dgRight*, *colRight*);

形式 2: *dgLeft*.**INNERJOINCOUNT**(*dgRight*, *dgJoin*);

パッケージ変数

dgLeft
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

colLeft
名前またはインデックス値を使用して、*dgLeft* のキー列を指定します。

dgRight
右データグリッドの変数名を指定します。

colRight
名前またはインデックス値を使用して、*dgRight* のキー列を指定します。

dgJoin
複数列の結合を定義するデータグリッドを指定します。*dgJoin* は、(string, string)または(long, long)の形式である必要があります。

最初の列 *dgJoin.col1* は、名前またはインデックス値のいずれかを使用して、*dgLeft* のキー列を指定します。

2 番目の列 *dgJoin.col1* は、名前またはインデックス値のいずれかを使用して、*dgRight* のキー列を指定します。

オプション引数

INNERJOINCOUNTBYFILTER メソッド

内部結合操作によって生成される行数を返します。ソース行の基準は、行フィルター定義を使用して指定します。この操作の影響を受けるデータグリッドはありません。

返されるデータの種類: INTEGER

注: エラーが発生した場合、このメソッドは-1 を返します。
このメソッドサポートは、2025.02 で開始されます。

構文

dgLeft.**INNERJOINCOUNTBYFILTER**(*dgRight*, *filter* [, *varList*]);

パッケージ変数

dgLeft
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

dgRight
右データグリッドの変数名を指定します。

filter
行の選択基準を指定します。詳細については、“フィルター構文について” (*SAS DS2 プログラマガイド*)を参照してください。

オプション引数

varList
フィルター内のパラメーター化された値を解決するために使用する DS2 変数のセットを含む DS2 varlist を指定します。パラメーター解決コードでは、DS2 varList 変数のデータ型が、関連付けられた列のデータ型と一致していると想定されます。

LEFTJOIN メソッド

2 つのデータグリッドの左結合を実行し、結合の結果をターゲットデータグリッドに入力します。

返されるデータの種類: INTEGER

注:
- 左テーブルのすべての 行は、一致する結合基準に関係なく、ターゲットデータグリッドに保持されます。
- このメソッドは、結合が成功した場合は 0 (ゼロ)を返し、成功しなかった場合はゼロ以外の値を返します。
- 形式 2 のサポートは、2023.03 で開始されます。

構文

形式 1: *package*.**LEFTJOIN**(*dgLeft*, *colLeft*, *dgRight*, *colRight*);

形式 2: *package*.**LEFTJOIN**(*dgLeft*, *dgRight*, *dgJoin*);

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

dgLeft
左データグリッドの変数名を指定します。

colLeft
名前またはインデックス値を使用して、*dgLeft* のキー列を指定します。

dgRight
右データグリッドの変数名を指定します。

colRight
名前またはインデックス値を使用して、*dgRight* のキー列を指定します。

dgJoin
複数列の結合を定義するデータグリッドを指定します。*dgJoin* は、(string, string)または(long, long)の形式である必要があります。

最初の列 *dgJoin.col1* は、名前またはインデックス値のいずれかを使用して、*dgLeft* のキー列を指定します。

2 番目の列 *dgJoin.col1* は、名前またはインデックス値のいずれかを使用して、*dgRight* のキー列を指定します。

例

例 1: 左結合

左結合の例を次に示します。この例では、表 14.24 と表 14.25 (1577 ページ)に示されているソーステーブルを 使用します。

この例では、次の操作が実行されます。

```
dataGrid_A.leftJoin(price, 'PRODUCT', quantity, 'PROD');
```

結果のデータグリッドは次のとおりです。

表 14.31 dataGrid_A

PRODUCT	PRICE	MATCH	QUANTITY
Potatoes	1.001	l_1	44
Avocados	2.002	l_2	66
Avocados	2.002	l_2	88
Kiwis	3.003	l_3	33
Onions	4.004	l_4	99
Melons	5.005	l_5	11
Oranges	6.006	l_6	null
Tomatoes	7.007	l_7	null

例 2: 複数列の左結合

複数列の左結合の例を次に示します。この例では、表 14.27 (1578 ページ)、表 14.28 (1579 ページ)および表 14.29 (1580 ページ)に示されているソーステーブルを 使用します。

この例では、次の操作が実行されます。

```
dataGrid_B.leftJoin(cost, amount, dgJoinKeys);
```

結果のデータグリッドは次のとおりです。

表 14.32　dataGrid_B

PRODUCT	TYPE	COST	MATCH	AMOUNT
Potatoes	Russet	1.001	l_1	44
Potatoes	Sweet	1.011	l_1.1	null
Avocados	Hass	2.002	l_2	88
Avocados	Reed	2.022	l_2.2	66
Kiwis	Golden	3.003	l_3	null
Kiwis	Purple	3.033	l_3.3	33
Onions	Vidalia	4.004	l_4	99
Onions	Yellow	4.044	l_4.4	null
Melons	Honey Dew	5.005	l_5	null
Melons	Watermelon	5.055	l_5.5	11
Oranges	Blood	6.006	l_6	null
Oranges	Navel	6.066	l_6.6	null
Tomatoes	Heirloom	7.007	l_7	null
Tomatoes	Beefsteak	7.077	l_7.7	null

LEFTJOINBYFILTER メソッド

2 つのデータグリッドの左結合を実行し、結合の結果をターゲットデータグリッドに入力します。ターゲット列とソース行の基準は、列保持句と行フィルター定義を使って指定します。

返されるデータの種類:　INTEGER

注: 左テーブルのすべての行は、一致する結合基準に関係なく、ターゲットデータグリッドに保持されます。

このメソッドは、結合が成功した場合は 0 (ゼロ)を返し、成功しなかった場合はゼロ以外の値を返します。

このメソッドサポートは、2025.02 で開始されます。

構文

package.**LEFTJOINBYFILTER**(*dgLeft*, *dgRight*, *keepColumns*, *filter* [, *varList*]);

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

dgLeft
左データグリッドの変数名を指定します。

dgRight
右データグリッドの変数名を指定します。

keepColumns
結果データグリッドに配置する列のリスト。詳細については、“列保持構文について” (*SAS DS2 プログラマガイド*)を参照してください。

filter
行の選択基準を指定します。詳細については、“フィルター構文について” (*SAS DS2 プログラマガイド*)を参照してください。

オプション引数

varList
フィルター内のパラメーター化された値を解決するために使用する DS2 変数のセットを含む DS2 varlist を指定します。パラメーター解決コードでは、DS2 varList 変数のデータ型が、関連付けられた列のデータ型と一致していると想定されます。

例

例 1: 左結合

左結合の例を次に示します。この例では、表 14.24 と表 14.25 (1577 ページ)に示されているソーステーブルを 使用します。

この例では、次の操作が実行されます。

```
dataGrid_A.leftJoinByFilter(price, quantity, 'L.PRODUCT, L.PRICE, L.MATCH, R.QUANTITY',
'L.PRODUCT eq R.PROD');
```

結果のデータグリッドは次のとおりです。

表 14.33　dataGrid_A

PRODUCT	PRICE	MATCH	QUANTITY
Potatoes	1.001	l_1	44
Avocados	2.002	l_2	66
Avocados	2.002	l_2	88
Kiwis	3.003	l_3	33
Onions	4.004	l_4	99
Melons	5.005	l_5	11
Oranges	6.006	l_6	null
Tomatoes	7.007	l_7	null

この例では、keepColumns に***** (アスタリスク)が指定されています。これにより、一致しない列名が結果のデータグリッドに含まれます。また、varList には **product** という変数があり、値 **Potatoes** が割り当てられています。

```
dataGrid_A.leftJoinByFilter(price, quantity, '*', 'L.PRODUCT eq R.PROD & L.PRODUCT eq ?',
[product]);
```

結果のデータグリッドは次のとおりです。

表 14.34　dataGrid_A

PRODUCT	PRICE	MATCH	PROD	QUANTITY
Potatoes	1.001	l_1	Potatoes	44
Avocados	2.002	l_2	null	null
Kiwis	3.003	l_3	null	null
Onions	4.004	l_4	null	null
Melons	5.005	l_5	null	null
Oranges	6.006	l_6	null	null
Tomatoes	7.007	l_7	null	null

この例では、keepColumns に* (アスタリスク)が指定されているため、結果のデータグリッドに一致しない列名が含まれます。ただし、前の例とは対照的に、varList が指定されていません。

```
dataGrid_A.leftJoinByFilter(price, quantity, '*', 'L.PRODUCT eq R.PROD');
```

結果のデータグリッドは次のとおりです。

表 14.35 dataGrid_A

PRODUCT	PRICE	MATCH	PROD	QUANTITY
Potatoes	1.001	l_1	Potatoes	44
Avocados	2.002	l_2	Avocados	66
Avocados	2.002	l_2	Avocados	88
Kiwis	3.003	l_3	Kiwis	33
Onions	4.004	l_4	Onions	99
Melons	5.005	l_5	Melons	11
Oranges	6.006	l_6	null	null
Tomatoes	7.007	l_7	null	null

例 2: 複数列の左結合

複数列の左結合の例を次に示します。この例では、表 14.27 (1578 ページ)、表 14.28 (1579 ページ)および表 14.29 (1580 ページ)に示されているソーステーブルを 使用します。

この例では、次の操作が実行されます。

```
dataGrid_B.leftJoinByFilter(cost, amount, 'L.PRODUCT, L.TYPE, L.COST, L.MATCH,
R.AMOUNT', 'L.PRODUCT eq R.PROD & L.TYPE eq R.TYPE');
```

結果のデータグリッドは次のとおりです。

表 14.36 dataGrid_B

PRODUCT	TYPE	COST	MATCH	AMOUNT
Potatoes	Russet	1.001	l_1	44
Potatoes	Sweet	1.011	l_1.1	null
Avocados	Hass	2.002	l_2	88
Avocados	Reed	2.022	l_2.2	66
Kiwis	Golden	3.003	l_3	null

PRODUCT	TYPE	COST	MATCH	AMOUNT
Kiwis	Purple	3.033	l_3.3	33
Onions	Vidalia	4.004	l_4	99
Onions	Yellow	4.044	l_4.4	null
Melons	Honey Dew	5.005	l_5	null
Melons	Watermelon	5.055	l_5.5	11
Oranges	Blood	6.006	l_6	null
Oranges	Navel	6.066	l_6.6	null
Tomatoes	Heirloom	7.007	l_7	null
Tomatoes	Beefsteak	7.077	l_7.7	null

LEFTJOINCOUNT メソッド

左結合操作によって生成される行数を返します。データグリッドは影響を受けません。

返されるデータの種類: INTEGER

注: エラーが発生した場合、このメソッドは-1 を返します。
形式 2 のサポートは、2023.03 で開始されます。

構文

形式 1: *dgLeft*.**LEFTJOINCOUNT**(*colLeft*, *dgRight*, *colRight*);

形式 2: *dgLeft*.**LEFTJOINCOUNT**(*dgRight*, *dgJoin*);

パッケージ変数

dgLeft
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

colLeft
名前またはインデックス値を使用して、*dgLeft* のキー列を指定します。

dgRight
右データグリッドの変数名を指定します。

colRight
名前またはインデックス値を使用して、*dgRight* のキー列を指定します。

dgJoin
複数列の結合を定義するデータグリッドを指定します。*dgJoin* は、(string, string)または(long, long)の形式である必要があります。

最初の列 *dgJoin.col1* は、名前またはインデックス値のいずれかを使用して、*dgLeft* のキー列を指定します。

2 番目の列 *dgJoin.col1* は、名前またはインデックス値のいずれかを使用して、*dgRight* のキー列を指定します。

LEFTJOINCOUNTBYFILTER メソッド

左結合操作によって生成される行数を返します。ソース行の基準は、行フィルター定義を使用して指定します。この操作の影響を受けるデータグリッドはありません。

返されるデータの種類: INTEGER

注:
エラーが発生した場合、このメソッドは-1 を返します。
このメソッドサポートは、2025.02 で開始されます。

構文

dgLeft.**LEFTJOINCOUNTBYFILTER**(*dgRight*, *filter*[, *varList*]);

パッケージ変数

dgLeft
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

dgRight
右データグリッドの変数名を指定します。

filter
行の選択基準を指定します。詳細については、“フィルター構文について” (*SAS DS2 プログラマガイド*)を参照してください。

オプション引数

varList
フィルター内のパラメーター化された値を解決するために使用する DS2 変数のセットを含む DS2 varlist を指定します。パラメーター解決コードでは、DS2

varList 変数のデータ型が、関連付けられた列のデータ型と一致していると想定されます。

NAME メソッド

データグリッドインスタンスの宣言で指定された名前を返します。

返されるデータの種類: STRING

構文

package.**NAME**();

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

NEW 演算子: データグリッドパッケージ

データグリッドパッケージのインスタンスを構築します。

構文

形式 1: *package*=**_NEW_DATAGRID**();

形式 2: *package*=**_NEW_DATAGRID**(*name*);

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

name
インスタンスの参照に使用できる内部名を指定します。

詳細

DS2 パッケージは、特定のインスタンスを構築して他の DS2 プログラムで使用できる変数とメソッドのコレクションです。

DECLARE PACKAGE ステートメントを使用して、データグリッドパッケージ変数を宣言します。データグリッドパッケージ変数を宣言した後、_NEW_演算子を使用してデータグリッドパッケージのインスタンスをインスタンス化し、新しいパッケージインスタンスをデータグリッドパッケージ変数に割り当てます。

```
declare package datagrid dg;
dg = _new_ datagrid();
```

DECLARE PACKAGE と_NEW_演算子を使用してデータグリッドパッケージを宣言およびインスタンス化する 2 ステップのプロセスのかわりに、コンストラクターメソッドとして DECLARE PACKAGE ステートメントを使用することにより、パッケージを 1 ステップで宣言およびインスタンス化できます。これは、DECLARE PACKAGE ステートメントのみを使用した同じ例です。

```
declare package datagrid dg();
```

注: パッケージ変数は、すべての変数スコーピングルールの対象となります。詳細については、“パッケージ、スコープ、および有効期間” (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- “パッケージのコンストラクターとデストラクター” (*SAS DS2 プログラマガイド*)
- “データグリッドパッケージの使用” (*SAS DS2 プログラマガイド*)

ステートメント:

- “DECLARE PACKAGE ステートメント: データグリッドパッケージ” (1562 ページ)

RIGHTJOIN メソッド

2 つのデータグリッドの右結合を実行し、結合の結果をターゲットデータグリッドに入力します。

返されるデータの種類: INTEGER

注:
- 右テーブルのすべての行は、一致する結合基準に関係なく、ターゲットデータグリッドに保持されます。
- このメソッドは、結合が成功した場合は 0 (ゼロ)を返し、成功しなかった場合はゼロ以外の値を返します。
- 形式 2 のサポートは、2023.03 で開始されます。

構文

形式 1: *package*.**RIGHTJOIN**(*dgLeft*, *colLeft*, *dgRight*, *colRight*);

形式 2: *package*.**RIGHTJOIN**(*dgLeft*, *dgRight*, *dgJoin*);

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

dgLeft
左データグリッドの変数名を指定します。

colLeft
名前またはインデックス値を使用して、*dgLeft* のキー列を指定します。

dgRight
右データグリッドの変数名を指定します。

colRight
名前またはインデックス値を使用して、*dgRight* のキー列を指定します。

dgJoin
複数列の結合を定義するデータグリッドを指定します。*dgJoin* は、(string, string)または(long, long)の形式である必要があります。

最初の列 *dgJoin.col1* は、名前またはインデックス値のいずれかを使用して、*dgLeft* のキー列を指定します。

2 番目の列 *dgJoin.col1* は、名前またはインデックス値のいずれかを使用して、*dgRight* のキー列を指定します。

例

例 1: 右結合

右結合の例を次に示します。この例では、表 14.24 と表 14.25 (1577 ページ)に示されているソーステーブルを 使用します。

この例では、次の操作が実行されます。

```
dataGrid_A.rightJoin(price, 'PRODUCT', quantity, 'PROD');
```

結果のデータグリッドは次のとおりです。

表 14.37 dataGrid_A

PRICE	MATCH	PROD	QUANTITY
5.005	r_1	Melons	11

PRICE	MATCH	PROD	QUANTITY
null	r_2	Squash	22
3.003	r_3	Kiwis	33
1.001	r_4	Potatoes	44
null	r_8	Broccoli	55
2.002	r_6	Avocados	66
null	r_8	Broccoli	77
2.002	r_5	Avocados	88
4.004	r_7	Onions	99
null	r_5	Pickles	100

この例では、keepColumns に***** (アスタリスク)が指定されています。これにより、一致しない列名が結果のデータグリッドに含まれます。また、varList には **product** という変数が含まれています: product = **Potatoes**。

```
dataGrid_A.rightJoin(price, quantity, '*', 'L.PRODUCT eq R.PROD & L.PRODUCT eq ?',
[product]);
```

結果のデータグリッドは次のとおりです。

例 2: 複数列の右結合

複数列の右結合の例を次に示します。この例では、表 14.27 (1578 ページ)、表 14.28 (1579 ページ)および表 14.29 (1580 ページ)に示されているソーステーブルを 使用します。

この例では、次の操作が実行されます。

```
dataGrid_B.rightJoin(cost, amount, dgJoinKeys);
```

結果のデータグリッドは次のとおりです。

表 14.38　dataGrid_B

COST	MATCH	PROD	TYPE	AMOUNT
5.055	r_1	Melons	Watermelon	11
null	r_2	Squash	Butternut	22
3.033	r_3	Kiwis	Purple	33
1.001	r_4	Potatoes	Russet	44

COST	MATCH	PROD	TYPE	AMOUNT
null	r_8	Broccoli	Blue Wind	55
2.022	r_6	Avocados	Reed	66
null	r_8	Broccoli	Eastern Magic	77
2.002	r_5	Avocados	Hass	88
4.004	r_7	Onions	Vidalia	99
null	r_5	Pickles	Dill	100

RIGHTJOINBYFILTER メソッド

2 つのデータグリッドの右結合を実行し、結合の結果をターゲットデータグリッドに入力します。ターゲット列とソース行の基準は、列保持句と行フィルター定義を使って指定します。

返されるデータの種類: INTEGER

注:
- 右テーブルのすべての 行は、一致する結合基準に関係なく、ターゲットデータグリッドに保持されます。
- このメソッドは、結合が成功した場合は 0 (ゼロ)を返し、成功しなかった場合はゼロ以外の値を返します。
- このメソッドサポートは、2025.02 で開始されます。

構文

package.**RIGHTJOINBYFILTER**(*dgLeft*, *dgRight*, *keepColumns*, *filter* [, *varList*]);

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

dgLeft
左データグリッドの変数名を指定します。

dgRight
右データグリッドの変数名を指定します。

keepColumns
結果データグリッドに配置する列のリスト。詳細については、"列保持構文について" (*SAS DS2 プログラマガイド*)を参照してください。

filter
行の選択基準を指定します。詳細については、"フィルター構文について" (*SAS DS2 プログラマガイド*)を参照してください。

オプション引数

varList
フィルター内のパラメーター化された値を解決するために使用する DS2 変数のセットを含む DS2 varlist を指定します。パラメーター解決コードでは、DS2 varList 変数のデータ型が、関連付けられた列のデータ型と一致していると想定されます。

例

例 1: 右結合

右結合の例を次に示します。この例では、表 14.24 と表 14.25 (1577 ページ)に示されているソーステーブルを 使用します。

この例では、次の操作が実行されます。

```
dataGrid_A.rightJoinByFilter(price, quantity, 'L.PRICE, R.MATCH, R.PROD, R.QUANTITY',
'L.PRODUCT eq R.PROD');
```

結果のデータグリッドは次のとおりです。

表 14.39 dataGrid_A

PRICE	MATCH	PROD	QUANTITY
5.005	r_1	Melons	11
null	r_2	Squash	22
3.003	r_3	Kiwis	33
1.001	r_4	Potatoes	44
null	r_8	Broccoli	55
2.002	r_6	Avocados	66
null	r_8	Broccoli	77
2.002	r_5	Avocados	88

PRICE	MATCH	PROD	QUANTITY
4.004	r_7	Onions	99
null	r_5	Pickles	100

この例では、keepColumns に* (アスタリスク)が指定されています。これにより、一致しない列名が結果のデータグリッドに含まれます。

```
dataGrid_A.rightJoinByFilter(price, quantity, '*', 'L.PRODUCT eq R.PROD');
```

結果のデータグリッドは次のとおりです。

表 14.40　dataGrid_A

PRODUCT	PRICE	MATCH	PROD	QUANTITY
Melons	5.005	r_1	Melons	11
null	null	r_2	Squash	22
Kiwis	3.003	r_3	Kiwis	33
Potatoes	1.001	r_4	Potatoes	44
null	null	r_8	Broccoli	55
Avocados	2.002	r_6	Avocados	66
null	null	r_8	Broccoli	77
Avocados	2.002	r_5	Avocados	88
Onions	4.004	r_7	Onions	99
null	null	r_5	Pickles	100

この例では、keepColumns に* (アスタリスク)が指定されています。これにより、一致しない列名が結果のデータグリッドに含まれます。また、varList には **product** という変数があり、値 **Potatoes** が割り当てられています。

```
dataGrid_A.rightJoinByFilter(price, quantity, '*', 'L.PRODUCT eq R.PROD & L.PRODUCT eq ?',
[product]);
```

結果のデータグリッドは次のとおりです。

表 14.41　dataGrid_A

PRODUCT	PRICE	MATCH	PROD	QUANTITY
null	null	r_1	Melons	11
null	null	r_2	Squash	22

PRODUCT	PRICE	MATCH	PROD	QUANTITY
null	null	r_3	Kiwis	33
Potatoes	1.001	r_4	Potatoes	44
null	null	r_8	Broccoli	55
null	null	r_6	Avocados	66
null	null	r_8	Broccoli	77
null	null	r_5	Avocados	88
null	null	r_7	Onions	99
null	null	r_5	Pickles	100

例 2: 複数列の右結合

複数列の右結合の例を次に示します。この例では、表 14.27 (1578 ページ)、表 14.28 (1579 ページ)および表 14.29 (1580 ページ)に示されているソーステーブルを 使用します。

この例では、次の操作が実行されます。

```
dataGrid_B.rightJoinByFilter(cost, amount, 'L.COST, R.MATCH, R.PROD, R.TYPE, R.AMOUNT',
'L.PRODUCT eq R.PROD & L.TYPE eq R.TYPE);
```

結果のデータグリッドは次のとおりです。

表 14.42　dataGrid_B

COST	MATCH	PROD	TYPE	AMOUNT
5.055	r_1	Melons	Watermelon	11
null	r_2	Squash	Butternut	22
3.033	r_3	Kiwis	Purple	33
1.001	r_4	Potatoes	Russet	44
null	r_8	Broccoli	Blue Wind	55
2.022	r_6	Avocados	Reed	66
null	r_8	Broccoli	Eastern Magic	77
2.002	r_5	Avocados	Hass	88
4.004	r_7	Onions	Vidalia	99

COST	MATCH	PROD	TYPE	AMOUNT
null	r_5	Pickles	Dill	100

RIGHTJOINCOUNT メソッド

右結合操作によって生成される行数を返します。データグリッドは影響を受けません。

返されるデータの種類: INTEGER

注: エラーが発生した場合、このメソッドは-1 を返します。
形式 2 のサポートは、2023.03 で開始されます。

構文

形式 1: *dgLeft*.**RIGHTJOINCOUNT**(*colLeft*, *dgRight*, *colRight*);

形式 2: *dgLeft*.**RIGHTJOINCOUNT**(*dgRight*, *dgJoin*);

パッケージ変数

dgLeft
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

colLeft
名前またはインデックス値を使用して、*dgLeft* のキー列を指定します。

dgRight
右データグリッドの変数名を指定します。

colRight
名前またはインデックス値を使用して、*dgRight* のキー列を指定します。

dgJoin
複数列の結合を定義するデータグリッドを指定します。*dgJoin* は、(string, string)または(long, long)の形式である必要があります。

最初の列 *dgJoin.col1* は、名前またはインデックス値のいずれかを使用して、*dgLeft* のキー列を指定します。

2 番目の列 *dgJoin.col1* は、名前またはインデックス値のいずれかを使用して、*dgRight* のキー列を指定します。

RIGHTJOINCOUNTBYFILTER メソッド

右結合操作によって生成される行数を返します。ソース行の基準は、行フィルター定義を使用して指定します。この操作の影響を受けるデータグリッドはありません。

返されるデータの種類: INTEGER

注: エラーが発生した場合、このメソッドは-1 を返します。
このメソッドサポートは、2025.02 で開始されます。

構文

dgLeft.**RIGHTJOINCOUNTBYFILTER**(*dgRight*, *filter* [, *varList*]);

パッケージ変数

dgLeft
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

dgRight
右データグリッドの変数名を指定します。

filter
行の選択基準を指定します。詳細については、“フィルター構文について” (*SAS DS2 プログラマガイド*)を参照してください。

オプション引数

varList
フィルター内のパラメーター化された値を解決するために使用する DS2 変数のセットを含む DS2 varlist を指定します。パラメーター解決コードでは、DS2 varList 変数のデータ型が、関連付けられた列のデータ型と一致していると想定されます。

ROWADD メソッド

指定された数の行を指定されたデータグリッドに追加します。

返されるデータの種類: INTEGER

注: このメソッドは、行が追加されている場合は 0 (ゼロ)を返し、行が追加されていない場合はゼロ以外の値を返します。

構文

package.**ROWADD**(*n*);

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

n
データグリッドに追加する行数を指定します。

ROWCOUNT メソッド

データグリッドの行数を返します。

返されるデータの種類: INTEGER

構文

package.**ROWCOUNT**();

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

ROWDELETE メソッド

指定されたデータグリッドから指定された行を削除します。

返されるデータの種類: INTEGER

注: 1 行だけを削除するには、そのインデックス値を *rowStartIndex* に指定します。
このメソッドは、行が削除されている場合は 0 (ゼロ)を返し、行が削除されていない場合はゼロ以外の値を返します。

構文

package.**ROWDELETE**(*rowStartIndex* <, *rowEndIndex*>);

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

rowStartIndex
操作を開始する行番号を指定します。

オプション引数

rowEndIndex
操作を終了する行番号を指定します。

SERIALIZE メソッド

指定されたデータグリッドの JSON 文字列を返します。

返されるデータの種類: 形式 1: VARCHAR 形式 2: INTEGER

注:
- データグリッドは、行定義の空の配列を持つ JSON 文字列を受け入れます。
- 形式 2 のシグネチャのサポートは、2024.09 で開始されます。
- エラーが発生した場合、形式 1 のシグネチャは **Missing-Value** を返し、形式 2 のシグネチャはゼロ以外の値を返します。詳細については、"詳細"を参照してください。

参照項目: "データグリッドの JSON 表現について" (*SAS DS2 プログラマガイド*)

構文

形式 1: *package*.**SERIALIZE**();

形式 2: *package*.**SERIALIZE**(*length*, *jsonString*);

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

length
JSON 文字列の長さで更新される変数を指定します。これは IN_OUT 引数です。

データの種類 INTEGER

jsonString
JSON 形式のシリアル化されたデータグリッドで更新される変数を指定します。これは IN_OUT 引数です。

データの種類 VARCHAR

詳細

戻りデータ型は、使用するシグネチャによって異なります。

- dg.serialize()はシリアル化された JSON 文字列を返します。(VARCHAR)

  注: 空のデータグリッドは、null の DS2 文字列を返します。

- dg.seriazlize(length, json)は、メソッドが成功した場合、0 (ゼロ)を返します。成功しなかった場合は、ゼロ以外の値を返します。(INTEGER)

SERIALIZEDATA メソッド

指定されたデータグリッドの行データの JSON 文字列を返します。列メタデータは含まれません。

返されるデータの種類: 形式 1: VARCHAR 形式 2: INTEGER

注: エラーが発生した場合、形式 1 のシグネチャは **Missing-Value** を返し、形式 2 のシグネチャはゼロ以外の値を返します。詳細については、"詳細"を参照してください。
形式 2 のシグネチャのサポートは、2024.09 で開始されます。

参照項目: "データグリッドの JSON 表現について" (*SAS DS2 プログラマガイド*)

構文

形式 1: *package*.**SERIALIZEDATA**();

形式 2: *package*.**SERIALIZEDATA**(*length*, *jsonString*);

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

length
JSON 文字列の長さで更新される変数を指定します。これは IN_OUT 引数です。

データの種類 INTEGER

jsonString
JSON 形式のシリアル化されたデータグリッドで更新される変数を指定します。これは IN_OUT 引数です。

データの種類 INTEGER

詳細

戻りデータ型は、使用するシグネチャによって異なります。

- dg.serializedata()はシリアル化された JSON 文字列を返します。 (VARCHAR)
- dg.seriazlizedata(length, json)は、メソッドが成功した場合、0 (ゼロ)を返します。成功しなかった場合は、ゼロ以外の値を返します。 (INTEGER)

SERIALIZEMETADATA メソッド

指定されたデータグリッドの列メタデータの JSON 文字列を返します。行データは含まれません。

返されるデータの種類: 形式 1: VARCHAR 形式 2: INTEGER

注: 形式 2 のシグネチャのサポートは、2024.09 で開始されます。
エラーが発生した場合、形式 1 のシグネチャは **Missing-Value** を返し、形式 2 のシグネチャはゼロ以外の値を返します。詳細については、"詳細"を参照してください。

参照項目: "データグリッドの JSON 表現について" (*SAS DS2 プログラマガイド*)

構文

形式 1: *package*.**SERIALIZEMETADATA**();

形式 2: *package*.**SERIALIZEMETADATA**(*length*, *jsonString*);

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

length
JSON 文字列の長さで更新される変数を指定します。これは IN_OUT 引数です。

データの種類　INTEGER

jsonString
JSON 形式のシリアル化されたデータグリッドで更新される変数を指定します。これは IN_OUT 引数です。

データの種類　VARCHAR

詳細

戻りデータ型は、使用するシグネチャによって異なります。

- dg.serializemetadata()はシリアル化された JSON 文字列を返します。(VARCHAR)
- dg.seriazlizemetadata(length, json)は、メソッドが成功した場合、0 (ゼロ)を返します。成功しなかった場合は、ゼロ以外の値を返します。(INTEGER)

SETALL メソッド

列のすべての値を指定された値に設定します。

返されるデータの種類:　INTEGER

注:　このメソッドは、データグリッドで変更された行数を返します。エラーが発生した場合、戻り値は **Missing-Value** です。

構文

package.**SETALL**(*col*, *setValue*);

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

col
名前またはインデックス値を使用して列を指定します。

setValue
割り当てる値を指定します。

SETBOOL メソッド

指定された行と列のセルに、指定されたブール値を設定します。

返されるデータの種類: INTEGER

注: このメソッドは、指定された値が設定されている場合は 0 (ゼロ)を返し、指定された値が設定されていない場合はゼロ以外の値を返します。

構文

package.**SETBOOL**(*col*, *rowIndex*, *setValue*);

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

col
名前またはインデックス値を使用して列を指定します。

rowIndex
データグリッドの行番号を指定します。

setValue
割り当てる値を指定します。

SETDOUBLE メソッド

指定された行と列のセルに、指定された 10 進数値を設定します。

返されるデータの種類: INTEGER

注: このメソッドは、指定された値が設定されている場合は 0 (ゼロ)を返し、指定された値が設定されていない場合はゼロ以外の値を返します。

構文

package.**SETDOUBLE**(*col*, *rowIndex*, *setValue*);

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

col
名前またはインデックス値を使用して列を指定します。

rowIndex
データグリッドの行番号を指定します。

setValue
割り当てる値を指定します。

SETINT メソッド

指定された行と列のセルに、指定された整数値を設定します。

返されるデータの種類: INTEGER

注: このメソッドは、指定された値が設定されている場合は 0 (ゼロ)を返し、指定された値が設定されていない場合はゼロ以外の値を返します。

構文

package.**SETINT**(*col*, *rowIndex*, *setValue*);

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

col
名前またはインデックス値を使用して列を指定します。

rowIndex
データグリッドの行番号を指定します。

setValue
割り当てる値を指定します。

SETLONG メソッド

指定された行と列のセルに、指定された long 値(64 ビット整数)を設定します。

返されるデータの種類: INTEGER

注: このメソッドは、指定された値が設定されている場合は 0 (ゼロ)を返し、指定された値が設定されていない場合はゼロ以外の値を返します。

構文

package.**SETLONG**(*col*, *rowIndex*, *setValue*);

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

col
名前またはインデックス値を使用して列を指定します。

rowIndex
データグリッドの行番号を指定します。

setValue
割り当てる値を指定します。

SETNAME メソッド

指定されたデータグリッドの名前を変更します。

注: このメソッドは、データグリッド名が変更された場合は 0 (ゼロ)を返し、エラーが発生した場合はゼロ以外の値を返します。

null 値を指定すると、このメソッドはデータグリッド名をデフォルト名_**dg** に変更します。

構文

package.**SETNAME**(*newName*);

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

newName
新しい名前を指定します。

SETSTRING メソッド

指定された行と列のセルに、指定された文字列値を設定します。

返されるデータの種類: INTEGER

注: このメソッドは、指定された値が設定されている場合は 0 (ゼロ)を返し、指定された値が設定されていない場合はゼロ以外の値を返します。

構文

package.**SETSTRING**(*col*, *rowIndex*, *setValue*);

パッケージ変数

package
データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

col
名前またはインデックス値を使用して列を指定します。

rowIndex
データグリッドの行番号を指定します。

setValue
割り当てる値を指定します。

SETVALUE メソッド

指定された行と列のセルに、指定された値を設定します。これは、列の型 DECIMAL と STRING でのみサポートされます。

返されるデータの種類: INTEGER

注: このメソッドは、指定された値が設定されている場合は 0 (ゼロ)を返し、指定された値が設定されていない場合はゼロ以外の値を返します。

構文

package.**SETVALUE**(*col*, *rowIndex*, *setValue*);

パッケージ変数

package
データタグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

col
名前またはインデックス値を使用して列を指定します。

rowIndex
データタグリッドの行番号を指定します。

setValue
割り当てる値を指定します。

SORT メソッド

ターゲットデータグリッドを並べ替えます。

返されるデータの種類: INTEGER

注: このメソッドは、ターゲットデータグリッドを並べ替えた場合は 0 (ゼロ)を返し、エラーが発生した場合はゼロ以外の値を返します。

構文

package.**SORT**(*sortcol*, *isAscending* | *dgSort*);

パッケージ変数

package
データタグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

sortCol
ターゲットデータグリッドの行を並べ替えるために値が使用される列の名前またはインデックス番号を指定します。

isAscending
ターゲットデータグリッドを昇順または降順のどちらで並べ替えるかを指定します。降順の場合は 0 (ゼロ)を指定し、昇順の場合はその他の値を指定します。sortCol の値が空の文字列の場合、関数は null 値を返します。

dgSort
データグリッドを渡して複数列の並べ替えを定義できるようにします。*dgSort* は、(string, num)または(num, num)の形式である必要があります。

最初の列 dgSort.col1 は、columnName または columnIndex です。

dgSort.col2 は、*isAscending* の値です。

package()は、row1、row2、そして *N* までの *dgSort* の基準に従って並べ替えられます。

例

複数列の並べ替えの例を次に示します。

この例では、Demographics がターゲットデータグリッドであり、dgSort が並べ替え基準を含むデータグリッドです。各データグリッドのコンテンツは次のとおりです。

表 14.43 ターゲットデータグリッド: Demographics

NAME	AGE
Charles	10
John	12
Matthew	14
Michael	16
Charles	20
John	22
Matthew	24
Michael	26
Charles	30

NAME	AGE
John	32
Matthew	34
Michael	36

表 14.44 並べ替え基準のデータグリッド: dgSort

COL_NAME	ISASCENDING
NAME	1
AGE	0

dgSort に示すように、まず NAME 列が昇順で並べ替えられます。一致する場合は、AGE 列を使用して降順で並べ替えます。

次の操作が実行されます。

```
Demographics.sort(dgSort)
```

結果のデータグリッドは次のとおりです。

表 14.45 Demographics

NAME	AGE
Charles	30
Charles	20
Charles	10
John	32
John	22
John	12
Matthew	34
Matthew	24
Matthew	14
Michael	36

NAME	AGE
Michael	26
Michael	16

SUBSET メソッド

データグリッドをサブセットして、比較列と値が true と評価される行のみを含めます。

返されるデータの種類: INTEGER

注: このメソッドは、サブセット化された行数を返します。エラーが発生した場合、このメソッドは-1 を返します。
形式 2 のサポートは、2023.03 で開始されます。

構文

形式 1: *package*.**SUBSET**(*dgSource*, *cmpCol*, *operator*, *cmpValue*);

形式 2: *package*.**SUBSET**(*dgSource*, *dgCriteria*);

パッケージ変数

package
サブセットの結果を受け取るデータグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

dgSource
ソースデータグリッドの変数名を指定します。

cmpCol
名前またはインデックス値を使用して列を指定し、その値を *cmpValue* と比較します。

operator
"比較演算子" (*SAS DS2 プログラマガイド*)に示されている演算子の 1 つを指定します。これらの演算子のいずれかに評価される文字変数の名前を指定するか、単一引用符で囲んだ演算子を指定できます。

cmpValue
cmpCol の値と比較する値を指定します。数値、単一引用符で囲まれた文字列、変数の名前、または式の名前を指定できます。指定する値は *cmpCol* と同じデータ型であるか、または *cmpCol* と同じデータ型に評価される必要があります。

null 値と欠損値を検索するには、次の点に注意してください。

- 文字列の場合、null は null 値のみに一致します。欠損(空の文字列またはすべての空白文字を含む文字列)は、欠損値に一致します。
- 他のすべての型では、null 値は存在しません。cmpValue に null を指定した場合、比較演算で欠損値として扱われます。

dgCriteria
渡すデータグリッドを指定します。このデータグリッドは、複数列のサブセットを定義します。*dgCriteria* は(string, string, string)または(long, string, string)の形式である必要があります。

- 最初の列 *dgCriteria.col1* は、columnName または columnIndex です。
- 2 番目の列 *dgCriteria.col2* には、"比較演算子" (*SAS DS2 プログラマガイド*)に示す演算子の 1 つを指定します。これらの演算子のいずれかに評価される文字変数の名前を指定するか、単一引用符で囲んだ演算子を指定できます。
- 3 番目の列 *dgCriteria.col3* には、*dgCriteria.col1* で指定された比較列と比較する値の文字列表現を指定します。

dgSource()は *dgCriteria* の基準によってサブセット化され、結果は *package()*に配置されます。*package*()に行を追加するには、*dgCriteria* のすべてのサブセット基準が真と評価される必要があります。

例

SUBSET メソッドの操作例をいくつか次に示します。これらのサンプル操作は、表 14.24 と表 14.25 (1577 ページ)ソーステーブルに示すデータを使用します。

例 1: 単一の値のサブセット

この例では、次の操作が実行されます。

```
dataGrid_A.subset(price, 'PRICE', 'le', 4.004);
```

結果のデータグリッドは次のとおりです。

表 14.46 dataGrid_A

PRODUCT	PRICE	MATCH
Potatoes	1.001	l_1
Avocados	2.002	l_2
Kiwis	3.003	l_3
Onions	4.004	l_4

例 2: 複数の値のサブセット

次のサンプル操作では、dgCriteria 引数を使用しています。**criteria_def** データグリッドのコンテンツは次のとおりです。

表 14.47　データグリッドのサブセット基準(dgCriteria)

列名	演算子	値
MATCH	ge	r_4
QUANTITY	le	77

操作は次のとおりです。

```
dataGrid_A.subset(quantity, criteria_def);
```

結果のデータグリッドは次のとおりです。

表 14.48　dataGrid_A

MATCH	PROD	QUANTITY
r_4	Potatoes	44
r_8	Broccoli	55
r_6	Avocados	66
r_8	Broccoli	77

SUBSETBYFILTER メソッド

行フィルターが true と評価される行のみを含むようにデータグリッドをサブセットします。ターゲット列とソース行の基準は、列保持句と行フィルター定義を使って指定します。

返されるデータの種類: INTEGER

注: このメソッドは、サブセット化された行数を返します。エラーが発生した場合、このメソッドは-1 を返します。

このメソッドサポートは、2025.02 で開始されます。

構文

package.**SUBSETBYFILTER**(*dgSource*, *keepColumns*, *filter* [, *varList*]);

パッケージ変数

package
サブセットの結果を受け取るデータグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

dgSource
ソースデータグリッドの変数名を指定します。

keepColumns
結果データグリッドに配置する列のリスト。詳細については、“列保持構文について” (*SAS DS2 プログラマガイド*)を参照してください。

filter
行の選択基準を指定します。詳細については、“フィルター構文について” (*SAS DS2 プログラマガイド*)を参照してください。

オプション引数

varList
フィルター内のパラメーター化された値を解決するために使用する DS2 変数のセットを含む DS2 varlist を指定します。パラメーター解決コードでは、DS2 varList 変数のデータ型が、関連付けられた列のデータ型と一致していると想定されます。

例

SUBSETBYFILTER メソッドの操作例をいくつか次に示します。これらのサンプル操作は、表 14.24 と表 14.25 (1577 ページ)ソーステーブルに示すデータを使用します。

例 1: 単一の値のサブセット

これらのサンプル操作は、単一の値に一致するサブセットを実行するためのさまざまな方法を示します。

この操作では、filter 引数は次のように定義されています。

```
dataGrid_A.subsetByFilter(price, '*', 'PRICE le 4.004');
```

この操作では、varList には **d** という変数があり、値 **4.004** が割り当てられています。

```
dataGrid_A.subsetByFilter(price, '*', 'PRICE le ?', [d]);
```

これらの操作ごとに得られるデータグリッドを次に示します。

表 14.49 dataGrid_A

PRODUCT	PRICE	MATCH
Potatoes	1.001	l_1
Avocados	2.002	l_2
Kiwis	3.003	l_3
Onions	4.004	l_4

例 2: 複数の値のサブセット

このサンプル操作は、'&' 論理演算子を持つフィルターを使用して、2 つの値に一致するサブセットを実行します。

```
dataGrid_A.subsetByFilter(quantity, '*', 'MATCH ge "r_4" & QUANTITY le 77');
```

このサンプル操作では、varList に match と count という変数が含まれています。変数 match には値 **r_4** が割り当てられ、count には値 **77** が割り当てられています。

```
dataGrid_A.subsetByFilter(quantity, '*', 'MATCH ge ? & QUANTITY le ?', [match count]);
```

これらの操作ごとに得られるデータグリッドを次に示します。

表 14.50 dataGrid_A

MATCH	PROD	QUANTITY
r_4	Potatoes	44
r_8	Broccoli	55
r_6	Avocados	66
r_8	Broccoli	77

TOPN メソッド

ターゲットデータグリッドを指定された列で降順に並べ替え、最初の *N* 行に切り捨てます。

返されるデータの種類: INTEGER

注: このメソッドは、結果のターゲットデータグリッドの行数を返します。エラーが発生した場合、このメソッドは-1 を返します。

構文

package.**TOPN** (*dgSource*, *col*, *N*);

パッケージ変数

package
　データグリッドパッケージインスタンスを参照する名前を指定します。

必須引数

dgSource
　ソースデータグリッドの変数名を指定します。

col
　名前またはインデックス値を使用して列を指定します。

N
　返される行の最大数を指定します。

VERSION メソッド

データグリッドインターフェイスのバージョンを返します。

返されるデータの種類: STRING

構文

package.**VERSION**();

パッケージ変数

package
　データグリッドパッケージインスタンスを参照する名前を指定します。

15

DS2 FCMP パッケージのメソッド、演算子、およびステートメント

ディクショナリ **1637**
DECLARE PACKAGE ステートメント: FCMP パッケージ 1637
DELETE メソッド: FCMP パッケージ 1641
NEW 演算子: FCMP パッケージ 1642

ディクショナリ

DECLARE PACKAGE ステートメント: FCMP パッケージ

パッケージ変数を作成し、FCMP パッケージのインスタンスを作成するオプションを提供します。

カテゴリ:	ローカル
制限事項:	このステートメントは CAS サーバーではサポートされていません。
要件	DECLARE PACKAGE ステートメントを使用する前に、PACKAGE ステートメントが必要です。
注:	DS2 は、PROC FCMP SUBROUTINE ステートメントのコードブロックを使用した PROC FCMP OUTARGS ステートメント引数をサポートします。DS2 は、PROC FCMP FUNCTION ステートメントのコードブロックを使用した OUTARGS ステートメント引数をサポートしていません。

構文

DECLARE PACKAGE *fcmp-package-name variable* ( );

必須引数

fcmp-package-name
FCMP パッケージの名前を指定します。

要件 パッケージ名は、PACKAGE ステートメントで作成されたパッケージの名前と一致する必要があります。一致しない場合、エラーが発生します。

参照項目: “PACKAGE ステートメント” (1392 ページ)

variable
パッケージのインスタンスを参照できる名前を指定します。

詳細

DS2 パッケージは、特定のインスタンスを構築して他の DS2 プログラムで使用できる変数とメソッドのコレクションです。

FCMP パッケージを使用して、FCMP プロシジャで使用可能または作成された関数およびサブルーチンへの呼び出しをサポートします。FCMP パッケージは、DS2 プログラム用に事前定義されています。

パッケージが宣言されると、パッケージのインスタンスを参照できる変数が作成されます。コンストラクター引数がパッケージ変数宣言で提供されている場合、パッケージインスタンスが構築され、構築されたパッケージインスタンスを参照するようにパッケージ変数が設定されます。複数のパッケージ変数を作成でき、単一の DECLARE PACKAGE ステートメントで複数のパッケージインスタンスを構築できます。各パッケージインスタンスは、パッケージの完全に別個のコピーを表します。

FCMP パッケージは、DECLARE PACKAGE ステートメントを使用して宣言します。これにより、FCMP パッケージが FCMP 名に関連付けられます。新しい FCMP パッケージを宣言した後、FCMP プロシジャで作成された関数とサブルーチンを呼び出すことができます。

FCMP パッケージのインスタンスを構築する方法は 2 つあります。

- DECLARE PACKAGE ステートメントを_NEW_演算子と一緒に使用します。

```
declare package fcmp pharma;
pharma = _new_ fcmp();
```

- DECLARE PACKAGE ステートメントをそのコンストラクター構文と一緒に使用します。

```
declare package fcmp pharma();
```

注: パッケージ変数は、すべての変数スコーピングルールの対象となります。詳細については、"パッケージ、スコープ、および有効期間" (*SAS DS2 プログラマガイド*)を参照してください。

FCMP パッケージの詳細については、"FCMP パッケージの使用" (*SAS DS2 プログラマガイド*)を参照してください。

例

例 1: FCMP OUTARGS パラメーターの使用

次の例では、OUTARGS パラメーターを使用する **package1** という名前の FCMP サブルーチンを作成します。FCMP プロシジャの OUTARGS パラメーターは、METHOD ステートメントでは IN_OUT パラメーターとして扱われます。

```
libname base '.';
proc fcmp outlib = base.fcmpsubs.package1;
  subroutine swapper(a,b);
    outargs a,b;
    t1 = b; b = a; a = t1;
  endsub;
run;
quit;

proc ds2;
  package pkg / overwrite=yes language='fcmp' table='base.fcmpsubs';
run;

data _null_;
  dcl package pkg p();
  method init();
    dcl double x y;
    x=10;
    y=42;
    put 'before:' x= y=;
    p.swapper(x,y);
    put 'after:' x= y=;
  end;
enddata;
run;
quit;
```

次の行が SAS ログに書き込まれます。

```
before: x=10 y=42
after: x=42 y=10
```

例 2: DOUBLE 引数を使用した FCMP パッケージ

この例では、FCMP で square ルーチンを作成し、DS2 プログラムからそのルーチンを使用する方法について説明します。現在のディレクトリは、FCMP パッケージの"ライブラリ"として使用されます。

```
libname base '.';

* fcmp defines a function, square;
proc fcmp outlib = base.fcmpsubs.package1;
  function square(a);
    return (a*a);
  endsub;
run;
quit;

* define the ds2 package thru which the fcmp functions will be called;
proc ds2;
  package pkg /overwrite=yes language='fcmp' table='base.fcmpsubs';
run;

* demonstration of calling fcmp thru the ds2 wrapper package;
data _null_;
  dcl package pkg p();
  dcl double a b;
  method init();
    do a = 10 to 20;
      b=p.square(a);
      put a= b=;
    end;
  end;
enddata;
run;
quit;
```

次の行が SAS ログに書き込まれます。

```
a=10 b=100
a=11 b=121
a=12 b=144
a=13 b=169
a=14 b=196
a=15 b=225
a=16 b=256
a=17 b=289
a=18 b=324
a=19 b=361
a=20 b=400
```

例 3: 文字引数付きの FCMP パッケージ

```
libname base '.';

proc fcmp outlib = base.fcmpsubs.package1;
  function f(a $) $ 10;
    return (trim(a) !! trim(a));
  endsub;
run;

proc ds2;
  package pkg /overwrite=yes language='fcmp' table='base.fcmpsubs';
run;

data _null_;
```

```
  dcl package pkg p();
  method runone(double arg, double expected);
    dcl double actual;
    actual=p.f(arg);
    if (actual ~= expected) then put 'ERROR:' arg= expected= actual=;
  end;
  method init();
    runone(5, 55);
    runone(345, 345345);
    runone(10, 1010);
    runone(4.2, .);    * can't convert back to double w/ two '.' chars;
    runone(., .);
  end;
enddata;
run;
quit;
```

関連項目

- “FCMP パッケージの使用” (*SAS DS2 プログラマガイド*)
- “パッケージのコンストラクターとデストラクター” (*SAS DS2 プログラマガイド*)

演算子:

- “_NEW_ 演算子: FCMP パッケージ” (1642 ページ)

ステートメント:

- “PACKAGE ステートメント” (1392 ページ)

DELETE メソッド: FCMP パッケージ

FCMP パッケージを削除します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

注: DELETE メソッドは必須ではありません。FCMP パッケージインスタンスを参照する FCMP パッケージ変数がない場合、パッケージインスタンスは自動的に削除されます。

構文

package.**DELETE**( );

必須引数

package
FCMP パッケージ変数の名前を指定します。

詳細

FCMP パッケージが不要になった場合は、DELETE メソッドを使用して削除します。FCMP パッケージを削除した後に使用しようとすると、ログにエラーが書き込まれます。

関連項目

"パッケージのコンストラクターとデストラクター" (SAS DS2 プログラマガイド)

NEW 演算子: FCMP パッケージ

FCMP パッケージのインスタンスを構築します。

制限事項: この演算子は CAS サーバーではサポートされていません。

注: かっこの前のエスケープ文字(\)は、構文にかっこが必要であることを示します。

構文

package-variable = **_NEW_** *fcmp-package-name*( );

必須引数

package-variable
パッケージのインスタンスを参照できる名前を指定します。

fcmp-package-name
パッケージの名前を指定します。

要件 *fcmp-package-name* は、PACKAGE ステートメントで作成された事前定義済み FCMP パッケージである必要があります。

詳細

DS2 パッケージは、特定のインスタンスを構築して他の DS2 プログラムで使用できる変数とメソッドのコレクションです。

PACKAGE ステートメントを使用して FCMP パッケージを作成し、次に DECLARE PACKAGE ステートメントを使用して FCMP パッケージを宣言およびインスタンス化します。これにより、FCMP パッケージが FCMP パッケージ変数名に関連付けられます。新しい FCMP パッケージを宣言した後、FCMP プロシジャで作成された関数とサブルーチンを呼び出すことができます。

FCMP パッケージのインスタンスを構築する方法は 2 つあります。

- DECLARE PACKAGE ステートメントを_NEW_演算子と一緒に使用します。

```
declare package pharma pkg;
pkg = _new_ pharma();
```

- DECLARE PACKAGE ステートメントをそのコンストラクター構文と一緒に使用します。

```
declare package pharma pkg();
```

注: パッケージ変数は、すべての変数スコーピングルールの対象となります。詳細については、“パッケージ、スコープ、および有効期間” (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- “FCMP パッケージの使用” (*SAS DS2 プログラマガイド*)
- “パッケージのコンストラクターとデストラクター” (*SAS DS2 プログラマガイド*)

ステートメント:

- “DECLARE PACKAGE ステートメント: FCMP パッケージ” (1637 ページ)
- “PACKAGE ステートメント” (1392 ページ)

16

DS2 ハッシュおよびハッシュ反復子パッケージの属性、メソッド、演算子、ステートメント

ディクショナリ **1646**
ADD メソッド: ハッシュパッケージ 1646
CHECK メソッド 1649
CLEAR メソッド 1651
DATA メソッド 1652
DATASET メソッド 1654
DECLARE PACKAGE ステートメント: ハッシュパッケージ 1656
DECLARE PACKAGE ステートメント: ハッシュ反復子パッケージ 1665
DEFINEDATA メソッド 1667
DEFINEDONE メソッド 1669
DEFINEKEY メソッド 1670
DELETE メソッド: ハッシュ, およびハッシュ反復子パッケージ 1672
DUPLICATE メソッド 1672
FIND メソッド 1674
FIND_NEXT メソッド 1676
FIND_PREV メソッド 1679
FIRST メソッド 1680
HASHEXP メソッド 1684
HAS_NEXT メソッド 1685
HAS_PREV メソッド 1687
ITEM_SIZE 属性 1688
KEYS メソッド 1689
LAST メソッド 1691
MULTIDATA メソッド 1692
NEW 演算子: ハッシュパッケージ 1694
NEW 演算子: ハッシュ反復子パッケージ 1698
NEXT メソッド 1700
NUM_ITEMS 属性 1701
ORDERED メソッド 1703
OUTPUT メソッド 1705
PREV メソッド 1707

REF メソッド 1708
REMOVE メソッド 1710
REMOVEALL メソッド 1712
REMOVEDUP メソッド 1714
REPLACE メソッド 1716
REPLACEDUP メソッド 1718
SETCUR メソッド 1721
SUM メソッド 1722
SUMDUP メソッド 1724
SUMINC メソッド 1727

ディクショナリ

ADD メソッド: ハッシュパッケージ

キー値、データ値、またはその両方をハッシュパッケージに追加します。

適用対象: ハッシュパッケージ

注: かっこの前のエスケープ文字(\)は、構文にかっこが必要であることを示します。

構文

形式 1: *package*.**ADD**();

形式 2: *package*.**ADD**(\[*keys*\], \[*data*\]);

形式 3: *package*.**ADD**(\[*keys*\]);

必須引数

package
ハッシュパッケージ変数のインスタンスを指定します。

オプション引数

[*keys*]
変数リストを使用してキー値を指定します。

制限事項 キーのみを指定した場合、ADD メソッドはキーのみのハッシュパッケージに対してのみ機能します。

参照項目: "DS2 変数リスト" (*SAS DS2 プログラマガイド*)

[*data*]
ハッシュデータを追加する変数を指定します。

参照項目: “DS2 変数リスト” (*SAS DS2 プログラマガイド*)

詳細

ADD メソッドを使用して、キーとデータの値をハッシュパッケージに保存できます。

キーとデータ値を ADD メソッドに渡す方法は 2 つあります。

- 暗黙の変数メソッド(形式 1)

  キー変数とデータ変数は ADD メソッドの呼び出しで暗黙的に指定されるため、指定する必要はありません。

- 変数リストメソッド(形式 2 および 3)

  指定されたキー変数とデータ変数は、ADD メソッドに明示的に渡されます。ハッシュパッケージにキーのみが含まれている場合は、形式 3 を使用します。

注:

- すでにハッシュパッケージに含まれているキーを追加すると、ADD メソッドはゼロ以外の値を返し、そのキーがすでにハッシュパッケージに含まれていることを示します。指定されたキーに関連付けられているデータを新しいデータに置き換えるには、REPLACE メソッドを使用します。ただし、ハッシュパッケージの作成時に DUPLICATE コンストラクターパラメーターまたはメソッドを ADD に設定すると、ADD メソッドはゼロを返します。
- DEFINEDATA メソッドでデータ変数を指定しない場合、データ変数は自動的にキーと同じであると見なされます。
- ADD メソッドは、データ変数の値をデータアイテムの値に設定しません。ハッシュパッケージの値のみを設定します。

例

例 1: 暗黙の変数メソッドの使用

次の例では、暗黙の変数メソッドを使用してキーとデータアイテムを 定義します。

```
data _null_;
  declare char(20) d;
  declare char(20) k;
  declare double rc;
  declare package hash h(4);
  method init();
    /* Define constant value for key and data */
    rc = h.defineKey ('k');
    rc = h.defineData ('d');
    rc = h.defineDone ();
    /* Define constant value for key and data */
```

```
    k='Homer';
    d ='Odyssey';
    /* Use the ADD method to add the key and data to the hash package */
    rc = h.add();
    /* Define constant value for key and data */
    k='Joyce';
    d='Ulysses';
    /* Use the ADD method to add the key and data to the hash package */
    rc = h.add();
  end;
enddata;
run;
```

例 2: 変数リストメソッドを使用したキー値とデータ値の追加

次の例では、暗黙の変数メソッドを使用してキーとデータアイテムを 定義します。

```
data _null_;
  declare char(20) d;
  declare char(20) k;
  declare double rc;
  method init();
    declare package hash h([k], [d]);
    /* Define constant value for key and data */
    k='Homer';
    d ='Odyssey';
    /* Use the ADD method to add the key and data to the hash package */
    rc = h.add([k], [d]);
    /* Define constant value for key and data */
    k='Joyce';
    d='Ulysses';
    /* Use the ADD method to add the key and data to the hash package */
    rc = h.add([k], [d]);
    h.output('hashdata2');
  end;
enddata;
run;
```

例 3: ADD メソッドと FIND メソッドの使用

次の例では、ADD メソッドを使用してデータをハッシュパッケージに保存し、データをキーに関連付けます。次に、FIND メソッドを使用して、キー値'Homer'に関連付けられているデータを取得します。

```
data _null_;
  declare char(20) d;
  declare char(20) k;
  declare double rc;
  method init();
    declare package hash h([k], [d], 4);
    /* Define constant value for key and data */
    k='Homer';
    put k=;
    d='Odyssey';
    put d=;
    /* Use the ADD method to add the key and data to the hash package */
```

```
    rc = h.add([k], [d]);
    /* Define constant value for key and data */
    k='Joyce';
    d='Ulysses';
    /* Use the ADD method to add the key and data to the hash package */
    rc = h.add([k], [d]);
    k='Homer';
    /* Use the FIND method to retrieve the data associated with 'Homer' key */
    if (h.find([k],[d]) = 0) then
   put d=;
   else
   put 'Key Homer not found.';
  end;
enddata;
run;
```

FIND メソッドは、データ値'Odyssey'を変数 D に割り当てます。'Odyssey'はキー値'Homer'に関連付けられます。

次の行が SAS ログに書き込まれます。

```
k=Homer
d=Odyssey
d=Odyssey
```

関連項目

- “暗黙の変数と変数リストのメソッド” (*SAS DS2 プログラマガイド*)

メソッド:

- “DEFINEDATA メソッド” (1667 ページ)
- “DEFINEKEY メソッド” (1670 ページ)
- “REF メソッド” (1708 ページ)

CHECK メソッド

指定されたキーがハッシュパッケージに保存されているかどうかを確認します。

適用対象: ハッシュパッケージ

注: かっこの前のエスケープ文字(\)は、構文にかっこが必要であることを示します。

構文

形式 1: *package*.**CHECK**( );

形式 2: *package*.**CHECK**(\[*keys*\]);

必須引数

package
ハッシュパッケージ変数のインスタンスを指定します。

オプション引数

[*keys*]
変数リストを使用してキー値を指定します。

参照項目: "DS2 変数リスト" (*SAS DS2 プログラマガイド*)

詳細

CHECK メソッドを使用して、キーがハッシュテーブルに存在するかどうかを判断しますが、データ変数は更新されません。CHECK メソッドは、キーがハッシュテーブルで見つかった場合はゼロ値を返し、キーが見つからなかった場合はゼロ以外の値を返します。

キーとデータ変数を CHECK メソッドに渡す方法は 2 つあります。

- 暗黙の変数メソッド(形式 1)

  キー変数は CHECK メソッドの呼び出しで暗示され、指定する必要はありません。
- 変数リストメソッド(形式 2)

  指定されたキー変数は、CHECK メソッドに明示的に渡されます。

比較

CHECK メソッドは、キーがハッシュパッケージにあるかどうかを示す値のみを返します。キーに関連付けられているデータ変数は更新されません。FIND メソッドも、キーがハッシュパッケージにあるかどうかを示す値を返します。ただし、キーがハッシュパッケージに含まれている場合、FIND メソッドはデータ変数をデータアイテムの値に設定し、メソッド呼び出し後に使用できるようにします。

例

次の例では、CHECK メソッドを使用して、データがキー値'Homer'に関連付けられているかどうかを判断します。

```
data _null_;
  declare char(20) d;
  declare char(20) k;
  declare double rc;
  method init();
    declare package hash h([k], [d]);
    /* Define constant value for key and data */
```

```
    k='Homer';
    put k=;
    d='Odyssey';
    put d=;
    /* Use the ADD method to add the key and data to the hash package */
    rc = h.add([k], [d]);
    /* Define constant value for key and data */
    k='Joyce';
    d='Ulysses';
    /* Use the ADD method to add the key and data to the hash package */
    rc = h.add([k], [d]);
    k='Homer';
    /* Use the CHECK method to verify the data associated with 'Homer' key */
    if (h.check([k]) = 0) then
      put 'Key Homer is found.';
    else
       put 'Key Homer not found.';
  end;
enddata;
run;
```

次の行が SAS ログに書き込まれます。

```
k=Homer
d=Odyssey
Key Homer is found
```

関連項目

- “暗黙の変数と変数リストのメソッド” (*SAS DS2 プログラマガイド*)

メソッド:

- “DEFINEKEY メソッド” (1670 ページ)
- “FIND メソッド” (1674 ページ)
- “KEYS メソッド” (1689 ページ)

CLEAR メソッド

ハッシュパッケージインスタンスを削除せずに、ハッシュパッケージからすべてのアイテムを 削除します。

適用対象: ハッシュパッケージ

構文

package.**CLEAR**( );

必須引数

package
ハッシュパッケージ変数のインスタンスを指定します。

詳細

CLEAR メソッドは、ハッシュパッケージ内からアイテムを削除しますが、ハッシュパッケージインスタンスは再利用できるように残します。ハッシュパッケージを完全に削除するには、DELETE メソッドを使用します。

ハッシュパッケージ MYHASH からすべてのアイテムをクリアするには、次のコードを使用します。

```
rc = myhash.clear();
```

関連項目

メソッド:

- “DELETE メソッド: ハッシュ, およびハッシュ反復子パッケージ” (1672 ページ)

DATA メソッド

変数リストを使用して、ハッシュパッケージに保存するデータ変数を指定します。

適用対象: ハッシュパッケージ

注: かっこの前のエスケープ文字(\)は、構文にかっこが必要であることを示します。
ハッシュインスタンスに渡されるすべての変数は、グローバル変数である必要があります。

構文

package.**DATA**(\[*data*\]);

必須引数

package
ハッシュパッケージ変数のインスタンスを指定します。

[*data*]
変数リストを使用してデータ変数の名前を指定します。

参照項目: "DS2 変数リスト" (*SAS DS2 プログラマガイド*)

詳細

ハッシュパッケージは、一意のルックアップキーを使用してデータを保存および取得します。キーとデータは変数であり、ドット表記メソッド呼び出しを使用してハッシュパッケージを初期化するために使用します。

変数リストを使用して、DATA メソッドでハッシュパッケージのデータ変数を指定します。すべてのキー変数とデータ変数を定義したら、ハッシュパッケージの初期化を完了するために DEFINEDONE メソッドを呼び出す必要があります。

注: または、DEFINEDATA メソッドや DECLARE PACKAGE ステートメントのコンストラクターを使用して、データ変数を指定することもできます。

キーとデータは、任意の数の文字変数または数値変数で構成されます。

例

この例では、2 つのデータ変数と 1 つのキー変数を含むハッシュパッケージを作成します。出力は降順で並べ替えられます。

```
data _null_;
  dcl double x;
  dcl double rc;
  dcl date d;
  /* The output will be in descending order. */
  dcl package hash h(8, '', 'descending', '', '');
  dcl package hiter hi('h');

  method init();
    rc = h.keys([d]);
    rc = h.data([d]);
    rc = h.data([x]);
    rc = h.definedone();
    d = date '1929-08-24'; x = 1; h.add();
    d = date '1930-09-25'; x = 2; h.add();
    d = date '1930-10-26'; x = 3; h.add();
    d = date '1930-10-27'; x = 4; h.add();
    d = date '1933-12-28'; x = 5; h.add();
    d = date '1999-12-01'; x = 999;
    do while (hi.next() = 0);
      put d= x=;
      d = date '1999-12-01'; x = 999;
    end;
  end;
enddata;
```

```
run;
```

次の行が SAS ログに書き込まれます。

```
d=1933-12-28 x=5
d=1930-10-27 x=4
d=1930-10-26 x=3
d=1930-09-25 x=2
d=1929-08-24 x=1
```

関連項目

- “キー変数とデータ変数の定義” (*SAS DS2 プログラマガイド*)
- “暗黙の変数と変数リストのメソッド” (*SAS DS2 プログラマガイド*)

メソッド:

- “DEFINEDATA メソッド” (1667 ページ)
- “DEFINEDONE メソッド” (1669 ページ)
- “KEYS メソッド” (1689 ページ)

ステートメント:

- “DECLARE PACKAGE ステートメント: ハッシュパッケージ” (1656 ページ)

DATASET メソッド

ハッシュパッケージにロードするテーブルの名前を指定します。

適用対象: ハッシュパッケージ

制限事項: このメソッドは CAS サーバーではサポートされていません。

注: 構文規則の中かっこは、構文のグループ化を示します。中かっこの前のエスケープ文字(\)は、構文に中かっこが必要であることを示します。*sql-text* は中かっこ({ })で囲む必要があります。

ハッシュインスタンスに渡されるすべての変数は、グローバル変数である必要があります。

構文

package.**DATASET**(['']*data-source*[''] | '\{*sql-text*\}');

必須引数

package
ハッシュパッケージ変数のインスタンスを指定します。

data-source | {*sql-text*}
データソースまたは FedSQL クエリとしてハッシュパッケージにロードするテーブルの名前を指定します。

data-source
テーブルの名前を指定します。

ヒント テーブルの名前は、文字列リテラルまたは文字変数にすることができます。リテラルを使用する場合は、テーブル名を単一引用符で囲む必要があります。

'{*sql-text*}'
テーブル行のセットに解決される有効な FedSQL コードを含む文字列です。

制限事項 この引数は CAS サーバーではサポートされていません。

要件 FedSQL クエリは中かっこ({ })で囲む必要があります。

注 FedSQL クエリは、{SELECT <select-list> FROM <table-specification>;}の形式で指定されます。詳細については、*SAS FedSQL 言語リファレンス*の SELECT ステートメントを参照してください。

詳細

DATASET メソッドを使用して、ハッシュパッケージにロードするテーブルを指定できます。

注: または、DECLARE PACKAGE ステートメントの *datasource* パラメーターまたは _NEW_演算子を使用して、テーブルを指定することもできます。

関連項目

- “データの格納と取得” (*SAS DS2 プログラマガイド*)
- “ハッシュパッケージの初期化データの提供” (*SAS DS2 プログラマガイド*)

演算子:

- “_NEW_ 演算子: ハッシュパッケージ” (1694 ページ)

ステートメント:

- “DECLARE PACKAGE ステートメント: ハッシュパッケージ” (1656 ページ)

DECLARE PACKAGE ステートメント: ハッシュパッケージ

パッケージ変数を作成し、ハッシュパッケージのインスタンスを作成するオプションを提供します。

カテゴリ: ローカル

注: 構文規則の中かっこは、構文のグループ化を示します。中かっこの前のエスケープ文字(\)は、構文に中かっこが必要であることを示します。*sql-text* は中かっこ({ })で囲む必要があります。

かっこの前のエスケープ文字(\)は、構文にかっこが必要であることを示します。

ハッシュインスタンスに渡されるすべての変数は、グローバル変数である必要があります。

ヒント: ハッシュパッケージには PACKAGE ステートメントは必要ありません。

構文

形式 1: **DECLARE PACKAGE HASH** *variable* ( );

形式 2: **DECLARE PACKAGE HASH** *variable* (*hashexp*, {'*datasource*' | '\{*sql-text*\}'}, '*ordered*', '*duplicate* ', '*suminc*', '*multidata*');

形式 3: **DECLARE PACKAGE HASH** *variable* (\[*keys*\], \[*data*\], [*hashexp*, {'*datasource*' | '\{*sql-text*\}'}, '*ordered*', '*duplicate* ', '*suminc*', '*multidata*']);

形式 4: **DECLARE PACKAGE HASH** *variable* (\[*keys*\], [*hashexp*, {'*datasource*' | '\{*sql-text*\}'}, '*ordered*', '*duplicate* ', '*suminc*', '*multidata*']);

引数

variable
ハッシュパッケージのインスタンスを参照できる変数を指定します。

[*keys*]
変数リストを使用してキー値を指定します。

参照項目: “DS2 変数リスト” (*SAS DS2 プログラマガイド*)

[*data*]
変数リストを使用してデータ変数を指定し、それらを指定されたキーに関連付けます。

参照項目: “DS2 変数リスト” (*SAS DS2 プログラマガイド*)

hashexp
ハッシュパッケージの内部テーブルサイズです。ここで、ハッシュテーブルのサイズは 2^n です。

hashexp の値は、ハッシュテーブルサイズを作成するための 2 の累乗の指数として使用されます。たとえば、hashexp の値が 4 の場合、ハッシュテーブルサイズは 2^4、つまり 16 に相当します。hashexp の最大値は 16 で、これはハッシュテーブルサイズ 2^{16} に相当します。

ハッシュテーブルサイズは、保存できるアイテムの数と同じではありません。ハッシュテーブルをコンテナーの配列と考えてください。16 のハッシュテーブルサイズには 16 個のコンテナーがあります。各コンテナーは、無限の数のアイテムを保持できます。ハッシュテーブルの効率は、アイテムをコンテナーにマッピングしたり、コンテナーからアイテムを取得したりするハッシュ関数の機能にあります。

ハッシュパッケージルックアップルーチンの効率を最大化するには、ハッシュパッケージ内のデータ量に応じてハッシュテーブルのサイズを設定する必要があります。最良の結果が得られるまで、さまざまな hashexp 値を試してください。たとえば、ハッシュパッケージに 100 万個のアイテムが含まれている場合、16 のハッシュテーブルサイズ(hashexp = 4)はあまり効率的ではありません。512 または 1024 (hashexp = 9 または 10)のハッシュテーブルサイズを使用すると、パフォーマンスが向上します。

範囲 0-20

要件 *hashexp* の値を入力する必要があります。0 未満の値を入力すると、デフォルト値の 8 が使用されます。これは、ハッシュテーブルサイズが 2^8 または 256 に相当します。20 より大きい値を入力すると、デフォルト値の 20 が使用されます。

データの種類 INTEGER

'*datasource*'
ハッシュパッケージにロードするテーブルの名前です。

テーブルの名前は、リテラルまたは文字変数にすることができます。テーブル名は、単一引用符で囲む必要があります。

制限事項 datasource 引数が null でない場合、DECLARE PACKAGE ステートメントは CAS サーバーでサポートされていません。

要件 引用符で囲まれた空の文字列のプレースホルダー('')または *datasource* の値を入力する必要があります。プレースホルダーが入力されている場合、ハッシュはデータソースからロードされません。

{*sql-text*}
テーブル行のセットに解決される有効な FedSQL SELECT ステートメントです。

制限事項 この引数は CAS サーバーではサポートされていません。

要件 FedSQL クエリは中かっこ({ })で囲む必要があります。

注 FedSQL クエリは、{SELECT <select-list> FROM <table-specification>;}の形式で指定されます。詳細については、*SAS FedSQL 言語リファレンス*の SELECT ステートメントを参照してください。

'*ordered*'
ハッシュ反復子パッケージでハッシュパッケージを使用する場合、またはハッシュパッケージの OUTPUT メソッドを使用する場合に、データがキーと値の順序で返されるかどうか、またはどのように返されるかを指定します。

ordered には次のいずれかの値を指定できます。

'ASCENDING'
データはキーと値の昇順で返されます。'**ASCENDING**'を指定することは、'**YES**'を指定することと同じです。

別名 'A'

'DESCENDING'
データはキーと値の降順で返されます。

別名 'D'

'YES'
データはキーと値の昇順で返されます。'**YES**'を指定することは、'**ASCENDING**'を指定することと同じです。

'NO'
データは未定義の順序で返されます。

要件 引用符で囲まれた空の文字列のプレースホルダー('')または *ordered* の値を入力する必要があります。プレースホルダーを入力すると、デフォルトの順序である'NO'が使用されます。

'*duplicate*'
テーブルをハッシュパッケージにロードするときに重複キーを無視するかどうかを決定します。デフォルトでは、最初のキーを保存し、後続のすべての重複を無視します。

duplicate には次のいずれかの値を指定できます。

'REPLACE'
最後の重複キーレコードを保存します。

'ERROR'
重複キーが見つかった場合、ログにエラーを報告します。

'ADD'
見つかった最初のキーレコードを保存し、重複は保存しません。

要件 引用符で囲まれた空の文字列のプレースホルダー('')または *duplicate* の値を入力する必要があります。プレースホルダーを入力すると、デフォルトの'ADD'が使用されます。

参照項目: “例 3: ハッシュパッケージでの Duplicate パラメーターの使用” (1662 ページ)

'*suminc*'
ハッシュパッケージキーの要約カウントを維持する変数を指定します。

要件 引用符で囲まれた空の文字列のプレースホルダー('')または *suminc* の値を入力する必要があります。

データの種類 INTEGER

注 この変数は、合計増分、つまり、キーへの各参照のキー要約に追加する量を保持します。*suminc* 値は、SUM 関数のように、欠損値または null 値をゼロとして扱います。たとえば、キー要約は、変数の現在の値を使用して変更されます。

'*multidata*'
キーごとに複数のデータアイテムを許可するかどうかを指定します。

multidata には次のいずれかの値を指定できます。

'YES'
キーごとに複数のデータアイテムを許可します。

別名 'Y'または'MULTIDATA'

'NO'
キーごとに 1 つのデータアイテムのみを許可します。

別名 'N'または'SINGLEDATA'

要件 引用符で囲まれた空の文字列のプレースホルダー('')または *multidata* の値を入力する必要があります。プレースホルダーを入力すると、デフォルトの'NO'が使用されます。

詳細

DS2 パッケージは、特定のインスタンスを構築して他の DS2 プログラムで使用できる変数とメソッドのコレクションです。

特定のハッシュパッケージインスタンスは、一連のキー変数、一連のデータ変数、およびオプションの初期化データによって定義されます。ハッシュパッケージインスタンスは、構築時に完全に定義することも、構築時と後続の一連のメソッド呼び出しで定義することもできます。ハッシュパッケージインスタンスの構築時にキー変数とデータ変数が指定されている場合(形式 3 および 4)、ハッシュインスタンスは完全な定義として構築されます。ハッシュパッケージインスタンスの構築時にキー変数とデータ変数が提供されない場合(形式 2)、ハッシュインスタンスの追加の定義は、後続の一連のメソッド呼び出しとそれに続く単一の DEFINEDONE メソッド呼び出しで指定できます。DEFINEDONE メソッドは、キー変数、データ変数、およびその他の初期化データの指定が完了したことを示します。変数または初期化データが提供されていない場合、ハッシュパッケージインスタンスは構築されません(形式 1)。この場合、ハッシュインスタンスは、_NEW_演算子または追加のメソッド呼び出しと、それに続く単一の DEFINEDONE メソッド呼び出しで構築されます。

次の例では、ハッシュ **h1** とハッシュ **h2** の定義は同等です。キー変数とデータ変数がコンストラクター引数として提供されるため、ハッシュ **h1** はハッシュが構築されるときに完全に定義されます。ハッシュ **h2** は、ハッシュが構築されるときに部分的にのみ定義され、ハッシュ定義は、一連のメソッド呼び出しとそれに続く単一の DEFINEDONE メソッド呼び出しによって完了します。

```
declare package hash h1([key1], [data1 data2 data3],
   0, 'testdata', '', '', '', 'multidata');
declare package hash h2();

h2.keys([key1]);
h2.data([data1 data2 data3]);
h2.dataset('testdata');
h2.multidata();
h2.defineDone();
```

DECLARE PACKAGE ステートメントは、null パッケージ参照であるハッシュパッケージ変数を作成できます。次に、ハッシュパッケージ変数を設定して、後続の_NEW_演算子の呼び出しによって構築されたハッシュパッケージインスタンスを参照できます。

```
declare package hash hashgnp;
hashgnp = _new_ hash(10, 'testpkg', 'yes');
```

(オプション) DECLARE PACKAGE ステートメントは、ハッシュパッケージ変数の作成とハッシュパッケージインスタンスの構築の両方を行うことができます。

```
declare package hash hashgnp(10, 'testpkg', 'yes');
```

注: パッケージ変数は、すべての変数スコーピングルールの対象となります。詳細については、*"パッケージ、スコープ、および有効期間" (SAS DS2 プログラマガイド)*を参照してください。

ハッシュパッケージの詳細については、*"ハッシュパッケージの使用" (SAS DS2 プログラマガイド)*を参照してください。

例

例 1: ハッシュパッケージによるデータの保存と取得

次の例では、H という名前のハッシュパッケージを宣言します。このパッケージは、いくつかのキーと値のペアを保存し、反復子を使用して、並べ替えられた順序でキーを書き込みます。

```
data _null_;
   dcl double x;
   dcl date d;
/* The output will be in descending order. */
   dcl package hash h(8, '', 'descending', '', '');
   dcl package hiter hi('h');
   method init();
    rc = h.defineKey('d');
    rc = h.defineData('d');
    rc = h.defineData('x');
    rc = h.defineDone();

    d = date '1929-08-24'; x = 1; h.add();
    d = date '1930-09-25'; x = 2; h.add();
    d = date '1930-10-26'; x = 3; h.add();
    d = date '1930-10-27'; x = 4; h.add();
```

```
    d = date '1933-12-28'; x = 5; h.add();

    d = date '1999-12-01'; x = 999;
    do while (hi.next() = 0);
      put d= x=;
      d = date '1999-12-01'; x = 999;
    end;
  end;
enddata;
```

次の行が SAS ログに書き込まれます。

```
d=1933-12-28 x=5
d=1930-10-27 x=4
d=1930-10-26 x=3
d=1930-09-25 x=2
d=1929-08-24 x=1
```

例 2: ハッシュパッケージへのテーブルのロード

テーブル SMALL に 2 つの数値変数 K (キー)と S (データ)が含まれ、別のテーブル LARGE に対応するキー変数 K が含まれていると仮定します。次のコードは、SMALL テーブルをハッシュパッケージにロードし、LARGE テーブルの変数 K でキーが一致するものをハッシュパッケージで検索します。

```
/* create small table */
data small(overwrite=yes);
  dcl char(8) k s;
  method init();
    dcl integer i;
    do i = 1 to 10;
      k = put(i, BEST8.);
      s = put(2*i, BEST8.);
      output;
    end;
  end;
enddata;
run;
/* create large table */
data large(overwrite=yes);
  dcl char(8) k;
  method init();
    dcl integer i;
    do i = -20 to 20;
      k = put(i, BEST8.);
      output;
    end;
  end;
enddata;
run;

/*load SMALL table into the hash package */
data myhash(overwrite=yes);
  declare char(8) k;
  declare char(8) s;
  declare package hash h(8,'small');
```

```
 /* define SMALL table variable K as key and S as value */

  method init();
    rc = h.defineKey('k');
    rc = h.defineData('s');
    rc = h.defineDone();
  end;

/* use the SET statement to iterate over the LARGE table using */
/* keys in the LARGE table to match keys in the hash package */

  method run();
    set large;
    if (h.find() = 0) then output;
  end;

enddata;
run;
```

例 3: ハッシュパッケージでの Duplicate パラメーターの使用

これは、DUPLICATE パラメーターで ADD、REPLACE、および ERROR オプションを使用する例です。

```
data dups(overwrite=yes);
  dcl double x y;
  method init();
    do x = 1 to 5;
      y = 2*x;
      output;
    end;
    x = 3; y = 99; output;
    x = 4; y = 100; output;
  end;
enddata;
run;

data _null_;
  dcl double x y;
  dcl int rc1 rc2 rc3 rc4;
  dcl package hash h(8, 'dups', 'yes');
  dcl package hiter hi;
  method init();
    rc = h.defineKey('x');
    rc = h.defineData('x');
    rc = h.defineData('y');
    rc = h.defineDone();

    hi = _new_ hiter('h');
      do while (hi.next() = 0);
      put x= y=;
      end;
    put;
  end;
enddata;
run;
```

```
data _null_;
  dcl double x y;
  dcl int rc1 rc2 rc3 rc4;
  dcl package hash h(8, 'dups', 'yes', 'add');
  dcl package hiter hi;
  method init();
    rc = h.defineKey('x');
    rc = h.defineData('x');
    rc = h.defineData('y');
    rc = h.defineDone();

    hi = _new_ hiter('h');
      do while (hi.next() = 0);
        put x= y=;
      end;
    put;
  end;
enddata;
run;

data _null_;
  dcl double x y;
  dcl int rc1 rc2 rc3 rc4;
  dcl package hash h(8, 'dups', 'yes', 'replace');
  dcl package hiter hi;
  method init();
    rc = h.defineKey('x');
    rc = h.defineData('x');
    rc = h.defineData('y');
    rc = h.defineDone();

    hi = _new_ hiter('h');
      do while (hi.next() = 0);
      put x= y=;
      end;
    put;
  end;
enddata;
run;

data _null_;
  dcl double x y;
  dcl int rc1 rc2 rc3 rc4;
  dcl package hash h(8, 'dups', 'yes', 'error');
  dcl package hiter hi;
  method init();
    rc = h.defineKey('x');
    rc = h.defineData('x');
    rc = h.defineData('y');
    rc = h.defineDone();

    hi = _new_ hiter('h');
      do while (hi.next() = 0);
        put x= y=;
      end;
```

```
    put;
  end;
enddata;
run;
```

ADD オプションを使用すると、SAS ログに次の行が書き込まれます。

```
x=1 y=2
x=2 y=4
x=3 y=6
x=4 y=8
x=5 y=10

NOTE: Execution succeeded.No rows affected.
```

REPLACE オプションを使用すると、SAS ログに次の行が書き込まれます。

```
x=1 y=2
x=2 y=4
x=3 y=99
x=4 y=100
x=5 y=10

NOTE: Execution succeeded.No rows affected.
```

ERROR オプションを使用すると、エラーメッセージが SAS ログに書き込まれます。

```
ERROR: Line 174: Error reported by DS2 package d2hash:
ERROR: Line 174: Duplicate key found when loading hash object from data source dups.
ERROR: Line 174: Error reported by DS2 package d2hash:
ERROR: Line 174: Hash data source load failed.
ERROR: Line 174: Fatal run-time error.
ERROR: General error
```

例 4: ハッシュパッケージでの ORDERED パラメーターの使用

次の例では、ハッシュパッケージ H の昇順と、ハッシュパッケージ H2 の降順を設定します。

```
data _null_;
  dcl double x y;
  dcl package hash h(8, '', 'ascending');
  dcl package hash h2(8, '', 'descending');
  dcl package hiter hi('h');
  dcl package hiter hi2('h2');
  method init();
   rc = h.defineKey('x');
   rc = h.defineKey('y');
   rc = h.defineDone();

   rc = h2.defineKey('y');
   rc = h2.defineKey('x');
   rc = h2.defineDone();

   do x = 1 to 10;
    y = 2 * x;
    rc = h.add();
    rc = h2.add();
```

```
    end;

    hi.first(); put y=;
    do while (hi.next() = 0);
      put y=;
    end;
    put;

    hi2.first(); put x=;
    do while (hi2.next() = 0);
      put x=;
    end;
  end;
 enddata;
run;
```

関連項目

- “ハッシュパッケージの使用” (*SAS DS2 プログラマガイド*)
- “ハッシュパッケージの初期化データの提供” (*SAS DS2 プログラマガイド*)
- “パッケージのコンストラクターとデストラクター” (*SAS DS2 プログラマガイド*)

演算子:

- “_NEW_ 演算子: ハッシュパッケージ” (1694 ページ)

ステートメント:

- “DECLARE PACKAGE ステートメント: ハッシュ反復子パッケージ” (1665 ページ)

DECLARE PACKAGE ステートメント: ハッシュ反復子パッケージ

ハッシュ反復子パッケージ変数を作成し、ハッシュ反復子パッケージのインスタンスを作成するオプションを提供します。

カテゴリ: ローカル

ヒント: ハッシュ反復子パッケージには PACKAGE ステートメントは必要ありません。

構文

DECLARE PACKAGE HITER *variable* [('*hash-name*' | *hash-package-instance*)];

必須引数

variable
ハッシュ反復子パッケージ変数のインスタンスを参照できる名前を指定します。

オプション引数

'*hash-name*'
ハッシュ反復子が関連付けられているハッシュパッケージの名前を指定します。

hash-package-instance
ハッシュ反復子が関連付けられているハッシュパッケージのインスタンスを指定します。

詳細

DS2 パッケージは、特定のインスタンスを構築して他の DS2 プログラムで使用できる変数とメソッドのコレクションです。

ハッシュおよびハッシュ反復子パッケージを使用して、一意のルックアップキーに基づいてデータを迅速かつ効率的に保存、検索、および取得します。ハッシュおよびハッシュ反復子パッケージは、DS2 プログラム用に事前定義されています。

パッケージが宣言されると、パッケージのインスタンスを参照できる変数が作成されます。コンストラクター引数がパッケージ変数宣言で提供されている場合、パッケージインスタンスが構築され、構築されたパッケージインスタンスを参照するようにパッケージ変数が設定されます。複数のパッケージ変数を作成でき、単一の DECLARE PACKAGE ステートメントで複数のパッケージインスタンスを構築できます。各パッケージインスタンスは、パッケージの完全に別個のコピーを表します。

DECLARE PACKAGE ステートメントを使用して、ハッシュ反復子パッケージを宣言します。これにより、ハッシュ反復子パッケージがハッシュおよびハッシュ反復子名に関連付けられます。

注: ハッシュ反復子パッケージを作成する前に、ハッシュパッケージを宣言してインスタンス化する必要があります。

ハッシュ反復子パッケージのインスタンスを構築する方法は 2 つあります。

- DECLARE PACKAGE ステートメントを_NEW_演算子と一緒に使用します。

```
declare package hiter myhiter;
myhiter = _new_ hiter('h');
```

- DECLARE PACKAGE ステートメントをそのコンストラクター構文と一緒に使用します。

```
declare package hiter myiter('h');
```

注: パッケージ変数は、すべての変数スコーピングルールの対象となります。詳細については、“パッケージ、スコープ、および有効期間” (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- “ハッシュ反復子パッケージの使用” (*SAS DS2 プログラマガイド*)
- “ハッシュパッケージの使用” (*SAS DS2 プログラマガイド*)
- “パッケージのコンストラクターとデストラクター” (*SAS DS2 プログラマガイド*)

演算子:

- “_NEW_ 演算子: ハッシュ反復子パッケージ” (1698 ページ)

ステートメント:

- “DECLARE PACKAGE ステートメント: ハッシュパッケージ” (1656 ページ)

DEFINEDATA メソッド

暗黙の変数を使用して、ハッシュパッケージのデータ変数を定義します。

適用対象: ハッシュパッケージ

注: ハッシュインスタンスに渡されるすべての変数は、グローバル変数である必要があります。

構文

package.**DEFINEDATA**('*data*');

必須引数

package
ハッシュパッケージ変数のインスタンスを指定します。

'*data*'
データ変数の名前を指定します。

詳細

ハッシュパッケージは、一意のルックアップキーを使用してデータを保存および取得します。キーとデータは変数であり、ドット表記メソッド呼び出しを使用してハッシュパッケージを初期化するために使用します。

作成するデータ変数ごとに DEFINEDATA メソッドを呼び出します。すべてのキー変数とデータ変数を定義したら、ハッシュパッケージの初期化を完了するために DEFINEDONE メソッドを呼び出す必要があります。

注: または、DATA 変数リストメソッドや DECLARE PACKAGE ステートメントのコンストラクターを使用して、データ変数を指定することもできます。

キーとデータは、任意の数の文字変数または数値変数で構成されます。

例

この例では、2 つのデータ変数と 1 つのキー変数を含むハッシュパッケージを作成します。出力は降順で並べ替えられます。

```
data _null_;
  dcl double x;
  dcl double rc;
  dcl date d;
  /* The output will be in descending order. */
  dcl package hash h(8, '', 'descending', '', '');
  dcl package hiter hi('h');

  method init();
    rc = h.definekey('d');
    rc = h.definedata('d');
    rc = h.definedata('x');
    rc = h.defineDone();
    d = date '1929-08-24'; x = 1; h.add();
    d = date '1930-09-25'; x = 2; h.add();
    d = date '1930-10-26'; x = 3; h.add();
    d = date '1930-10-27'; x = 4; h.add();
    d = date '1933-12-28'; x = 5; h.add();
    d = date '1999-12-01'; x = 999;
    do while (hi.next() = 0);
      put d= x=;
      d = date '1999-12-01'; x = 999;
    end;
  end;
enddata;
run;
```

次の行が SAS ログに書き込まれます。

```
d=1933-12-28 x=5
d=1930-10-27 x=4
d=1930-10-26 x=3
d=1930-09-25 x=2
d=1929-08-24 x=1
```

関連項目

- “キー変数とデータ変数の定義” (*SAS DS2 プログラマガイド*)
- “暗黙の変数と変数リストのメソッド” (*SAS DS2 プログラマガイド*)

メソッド:

- “DATA メソッド” (1652 ページ)
- “DEFINEDONE メソッド” (1669 ページ)

ステートメント:

- “DECLARE PACKAGE ステートメント: ハッシュパッケージ” (1656 ページ)

DEFINEDONE メソッド

すべてのキーとデータの定義が完了していることを示します。

適用対象: ハッシュパッケージ

注: ハッシュインスタンスに渡されるすべての変数は、グローバル変数である必要があります。

構文

package.**DEFINEDONE**( );

必須引数

package
ハッシュパッケージ変数のインスタンスを指定します。

詳細

ハッシュパッケージは、一意のルックアップキーを使用してデータを保存および取得します。キーとデータは変数であり、ドット表記メソッド呼び出しを使用してハッシュパッケージを初期化するために使用します。

キー変数とデータ変数は、3 つの方法のいずれかで定義できます。

- 暗黙の変数メソッド DEFINEDATA および DEFINEKEY を使用します。
- 変数リストメソッド DATA および KEYS を使用します。
- DECLARE PACKAGE ステートメントのコンストラクターとして、キーおよびデータ変数リストを使用します。

DECLARE PACKAGE ステートメントでコンストラクターを使用してハッシュパッケージを完全に定義していない場合は、DEFINEDONE メソッドを呼び出して、ハッシュパッケージの初期化を完了する必要があります。

例

次の例では、ハッシュパッケージを作成し、キー変数とデータ変数を定義して、ハッシュパッケージの初期化を完了します。

```
/* definedone with definedata method */
declare hash h(h);
 rc = h.defineKey('k');
 rc = h.defineData('d');
 rc = h.defineDone();

/* same package using definedone with data method */
declare hash h;
rc=h.keys([k]);
rc=h.data([d]);
rc=definedone();
```

関連項目

- “キー変数とデータ変数の定義” (*SAS DS2 プログラマガイド*)
- “暗黙の変数と変数リストのメソッド” (*SAS DS2 プログラマガイド*)

メソッド:

- “DEFINEDATA メソッド” (1667 ページ)
- “DEFINEKEY メソッド” (1670 ページ)

ステートメント:

- “DECLARE PACKAGE ステートメント: ハッシュパッケージ” (1656 ページ)

DEFINEKEY メソッド

暗黙の変数を使用して、ハッシュパッケージのキー変数を定義します。

適用対象: ハッシュパッケージ

注: ハッシュインスタンスに渡されるすべての変数は、グローバル変数である必要があります。

構文

package.**DEFINEKEY**('*key*');

必須引数

package
ハッシュパッケージ変数のインスタンスを指定します。

'*key*'
キー変数の名前を指定します。

詳細

ハッシュパッケージは、一意のルックアップキーを使用してデータを保存および取得します。キーとデータは変数であり、ドット表記メソッド呼び出しを使用してハッシュパッケージを初期化するために使用します。

作成するキー変数ごとに DEFINEKEY メソッドを呼び出します。すべてのキー変数とデータ変数を定義したら、ハッシュパッケージの初期化を完了するために DEFINEDONE メソッドを呼び出す必要があります。

注: または、KEYS 変数リストメソッドや DECLARE PACKAGE ステートメントのコンストラクターを使用して、キー変数を指定することもできます。

キーとデータは、任意の数の文字変数または数値変数で構成されます。

例

次の例では、ハッシュパッケージを作成し、キー変数とデータ変数を定義します。

```
declare hash h();
rc = h.definekey('k');
rc = h.definedata('d');
rc = h.definedone();
end;
```

関連項目

- “キー変数とデータ変数の定義” (*SAS DS2 プログラマガイド*)
- “暗黙の変数と変数リストのメソッド” (*SAS DS2 プログラマガイド*)

メソッド:

- “DEFINEDONE メソッド” (1669 ページ)
- “KEYS メソッド” (1689 ページ)

ステートメント:

- “DECLARE PACKAGE ステートメント: ハッシュパッケージ” (1656 ページ)

DELETE メソッド: ハッシュ, およびハッシュ反復子パッケージ

ハッシュまたはハッシュ反復子パッケージを削除します。

適用対象: ハッシュおよびハッシュ反復子パッケージ

注: DELETE メソッドは必須ではありません。ハッシュまたはハッシュ反復子パッケージ変数がハッシュまたはハッシュ反復子パッケージインスタンスを参照する場合、パッケージインスタンスは自動的に削除されます。

構文

package.**DELETE**( );

必須引数

package
ハッシュまたはハッシュ反復子パッケージ変数のインスタンスを指定します。

詳細

ハッシュまたはハッシュ反復子パッケージが不要になった場合は、DELETE メソッドを使用して削除します。ハッシュまたはハッシュ反復子パッケージを削除した後に使用しようとすると、ログにエラーが書き込まれます。

ハッシュパッケージ内からすべてのアイテムを 削除し、ハッシュパッケージを保存して再度使用する場合は、CLEAR メソッドを使用します。

関連項目

"パッケージのコンストラクターとデストラクター" (SAS DS2 プログラマガイド)

DUPLICATE メソッド

テーブルをハッシュパッケージにロードするときに重複キーを無視するかどうかを決定します。デフォルトでは、最初のキーを保存し、後続のすべての重複を無視します。

適用対象: ハッシュパッケージ

注: ハッシュインスタンスに渡されるすべての変数は、グローバル変数である必要があります。

構文

package.**DUPLICATE**('*option*');

必須引数

package
ハッシュパッケージ変数のインスタンスを指定します。

'*option*'
option には次のいずれかの値を指定できます。

'REPLACE'
最後の重複キーレコードを保存します。

'ERROR'
重複キーが見つかった場合、ログにエラーを報告します。

'ADD'
見つかった最初のキーレコードを保存し、重複は保存しません。

デフォルト ADD

詳細

デフォルトでは、ハッシュパッケージ内のすべてのキーは一意です。これは、キーごとに 1 セットのデータ変数が存在することを意味します。状況によっては、ハッシュパッケージに重複キーを含めること、つまり、複数のデータ変数のセットを 1 つのキーに関連付けることが必要になる場合があります。

テーブルに重複キーが含まれている場合、デフォルトでは、最初のインスタンスはハッシュパッケージに保存され、後続のインスタンスは無視されます。最後のインスタンスをハッシュパッケージに保存するには、DUPLICATE メソッドを使用します。重複キーがある場合、DUPLICATE メソッドは SAS ログにもエラーを書き込みます。

ただし、MULTIDATA パラメーターまたはメソッドを使用する場合、ハッシュパッケージではキーごとに複数の値を保存できます。ハッシュパッケージは、キーに関連付けられたリストに複数の値を保持します。このリストは、HAS_NEXT や FIND_NEXT などのいくつかのメソッドを使用して走査および操作できます。

注: または、DECLARE PACKAGE ステートメントのコンストラクターとして *duplicate* パラメーターを使用するか、_NEW_演算子を使用して、重複キーを指定することもできます。

関連項目

- “一意でないキーとデータのペア” (*SAS DS2 プログラマガイド*)
- “ハッシュパッケージの初期化データの提供” (*SAS DS2 プログラマガイド*)

メソッド:

- “MULTIDATA メソッド” (1692 ページ)

演算子:

- “_NEW_ 演算子: ハッシュパッケージ” (1694 ページ)

ステートメント:

- “DECLARE PACKAGE ステートメント: ハッシュパッケージ” (1656 ページ)

FIND メソッド

指定されたキーがハッシュパッケージに保存されているかどうかを判別します。

適用対象: ハッシュパッケージ

注: かっこの前のエスケープ文字(\)は、構文にかっこが必要であることを示します。

構文

形式 1: *package*.**FIND**( );

形式 2: *package*.**FIND**(\[*keys*\], \[*data*\]);

形式 3: *package*.**FIND**(\[*keys*\]);

必須引数

package
ハッシュパッケージ変数のインスタンスを指定します。

[*keys*]
変数リストを使用してキー値を指定します。

制限事項 キーのみを指定した場合、FIND メソッドはキーのみのハッシュパッケージに対してのみ機能します。

参照項目: “DS2 変数リスト” (*SAS DS2 プログラマガイド*)

[*data*]
ハッシュデータをコピーする変数を指定します。

参照項目: “DS2 変数リスト” (*SAS DS2 プログラマガイド*)

詳細

キー変数値を使用して、キーがハッシュテーブルに存在するかどうかを判断します。キーが存在する場合、データ値はデータ変数にコピーされます。FIND メソッドは、キーがハッシュテーブルで見つかった場合はゼロ値を返し、キーが見つからなかった場合はゼロ以外の値を返します。

キーとデータ変数を FIND メソッドに渡す方法は 2 つあります。

- 暗黙の変数メソッド(形式 1)

  キー変数とデータ変数は FIND メソッドの呼び出しで暗黙的に指定されるため、指定する必要はありません。

- 変数リストメソッド(形式 2 および 3)

  指定されたキー変数とデータ変数は、FIND メソッドに明示的に渡されます。ハッシュパッケージにキーのみが含まれている場合は、形式 3 を使用します。

比較

FIND メソッドは、キーがハッシュパッケージにあるかどうかを示す値を返します。キーがハッシュパッケージに含まれている場合、FIND メソッドはデータ変数をデータアイテムの値に設定し、メソッド呼び出し後に使用できるようにします。CHECK メソッドは、キーがハッシュパッケージにあるかどうかを示す値のみを返します。データ変数は更新されません。

例

“例 3: ADD メソッドと FIND メソッドの使用” (1648 ページ)を参照してください。

関連項目

- “暗黙の変数と変数リストのメソッド” (*SAS DS2 プログラマガイド*)
- “一意でないキーとデータのペア” (*SAS DS2 プログラマガイド*)
- “データの格納と取得” (*SAS DS2 プログラマガイド*)

メソッド:

- “CHECK メソッド” (1649 ページ)
- “KEYS メソッド” (1689 ページ)
- “DEFINEKEY メソッド” (1670 ページ)

- “FIND_NEXT メソッド” (1676 ページ)
- “FIND_PREV メソッド” (1679 ページ)

FIND_NEXT メソッド

現在のリストアイテムを 現在のキーの複数アイテムリストの 次のアイテムに 設定し、対応するデータ変数のデータを設定します。

適用対象: ハッシュパッケージ

注: かっこの前のエスケープ文字(\)は、構文にかっこが必要であることを示します。

構文

形式 1: *package*.**FIND_NEXT**( );

形式 2: *package*.**FIND_NEXT**(\[*data*\]);

必須引数

package
ハッシュパッケージ変数のインスタンスを指定します。

[*data*]
現在のキーに関連付けられているデータをコピーする変数を指定します。

参照項目: “DS2 変数リスト” (*SAS DS2 プログラマガイド*)

詳細

FIND メソッドは、キーがハッシュパッケージに存在するかどうかを判別します。

HAS_NEXT メソッドは、複数のデータアイテムがキーに関連付けられているかどうかを判別します。キーに別のデータアイテムがあると判断した場合、FIND_NEXT メソッドを使用してそのデータアイテムを 取得できます。このメソッドは、データ変数をデータアイテムの値に設定して、メソッド呼び出し後に使用できるようにします。データアイテムリストに入ると、HAS_NEXT メソッドと FIND_NEXT メソッドを使用してリストを走査できます。別のアイテムがある場合、メソッドはゼロを返します。それ以外の場合は、ゼロ以外の値を返します。

FIND_NEXT メソッドにデータ変数を渡すには、2 つの方法があります。

- 暗黙の変数メソッド(形式 1)

  データ変数は FIND_NEXT メソッドの呼び出しで暗示され、指定する必要はありません。

- 変数リストメソッド(形式 2)

指定されたデータ変数は、FIND_NEXT メソッドに明示的に渡されます。

例

この例では、FIND_NEXT メソッドを使用して、複数のキーに複数のデータアイテムがあるテーブルを反復処理します。

```
data testcases;
  dcl double k;
  dcl double expected;
  method init();
    k=0; expected=14; output; /* magic number */

    k=1; expected=1; output;
    k=2; expected=2; output;
    k=3; expected=1; output;
    k=4; expected=3; output;
    k=5; expected=2; output;
    k=6; expected=1; output;
    k=7; expected=1; output;
    k=8; expected=1; output;
    k=9; expected=1; output;
  end;
enddata;
run;

data inp;
  dcl double k v;
  method init();
    do k = 1 to 10; v = k * k; output; end;
    k = 2; v = 3;    output; /* newval < oldval */
    k = 4; v = 4242;  output; /* newval > oldval */
    k = 4; v = 0;    output; /* newval < oldval */
    k = 5; v = 25;    output; /* newval = oldval */
  end;
enddata;
run;

data _null_;
  dcl double k v;
  dcl package hash h(8, 'inp', 'ascending', '', '', 'multidata');
  method init();
    h.defineKey('k');
    h.defineData('k');
    h.defineData('v');
    h.defineDone();
  end;
  method run();
    dcl double actual;
    /***********************************************/
    set testcases;

    rc = h.find();
    if (rc ~= 0) then
```

```
      actual = h.get_num_items();
    else do;
      put k= rc=;
      actual = 0;
      do while (rc = 0);
        actual+1;
        rc = h.find_next();
        put k= rc=;
      end;
    end;
  end;
enddata;
run;
```

次の行が SAS ログに書き込まれます。

```
k=1 rc=0
k=1 rc=4
k=2 rc=0
k=2 rc=0
k=2 rc=4
k=3 rc=0
k=3 rc=4
k=4 rc=0
k=4 rc=0
k=4 rc=0
k=4 rc=4
k=5 rc=0
k=5 rc=0
k=5 rc=4
k=6 rc=0
k=6 rc=4
k=7 rc=0
k=7 rc=4
k=8 rc=0
k=8 rc=4
k=9 rc=0
k=9 rc=4
```

関連項目

- “暗黙の変数と変数リストのメソッド” (*SAS DS2 プログラマガイド*)
- “一意でないキーとデータのペア” (*SAS DS2 プログラマガイド*)
- “データの格納と取得” (*SAS DS2 プログラマガイド*)

メソッド:

- “FIND メソッド” (1674 ページ)
- “FIND_PREV メソッド” (1679 ページ)
- “HAS_NEXT メソッド” (1685 ページ)

FIND_PREV メソッド

現在のリストアイテムを現在のキーの複数アイテムリストの前のアイテムに設定し、対応するデータ変数のデータを設定します。

適用対象: ハッシュパッケージ

注: かっこの前のエスケープ文字(\)は、構文にかっこが必要であることを示します。

構文

形式 1: *package*.**FIND_PREV**( );

形式 2: *package*.**FIND_PREV**(\[*data*\]);

必須引数

package
ハッシュパッケージ変数のインスタンスを指定します。

[*data*]
現在のキーに関連付けられているデータをコピーする変数を指定します。

参照項目: "DS2 変数リスト" (*SAS DS2 プログラマガイド*)

詳細

FIND メソッドは、キーがハッシュパッケージに存在するかどうかを判別します。

HAS_PREV メソッドは、複数のデータアイテムがキーに関連付けられているかどうかを判別します。キーに前のデータアイテムがあると判断した場合、FIND_PREV メソッドを使用してそのデータアイテムを取得できます。このメソッドは、データ変数をデータアイテムの値に設定して、メソッド呼び出し後に使用できるようにします。データアイテムリストに入ると、HAS_NEXT メソッドと FIND_NEXT メソッドに加えて、HAS_PREV メソッドと FIND_PREV メソッドを使用してリストを走査できます。別のアイテムがある場合、メソッドはゼロを返します。それ以外の場合は、ゼロ以外の値を返します。

FIND_PREV メソッドにデータ変数を渡すには、2 つの方法があります。

- 暗黙の変数メソッド(形式 1)

  データ変数は FIND_PREV メソッドの呼び出しで暗示され、指定する必要はありません。

- 変数リストメソッド(形式 2)

  指定されたデータ変数は、FIND_PREV メソッドに明示的に渡されます。

例

“例: 要約値の取得” (1725 ページ)を参照してください。

関連項目

- “暗黙の変数と変数リストのメソッド” (*SAS DS2 プログラマガイド*)
- “一意でないキーとデータのペア” (*SAS DS2 プログラマガイド*)
- “データの格納と取得” (*SAS DS2 プログラマガイド*)

メソッド:

- “FIND メソッド” (1674 ページ)
- “FIND_NEXT メソッド” (1676 ページ)
- “HAS_PREV メソッド” (1687 ページ)

FIRST メソッド

基になるハッシュパッケージの最初の値を返します。

適用対象: ハッシュ反復子パッケージ

注: かっこの前のエスケープ文字(\)は、構文にかっこが必要であることを示します。

構文

形式 1: *package*.**FIRST**( );

形式 2: *package*.**FIRST**(\[*data*\]);

必須引数

package
ハッシュ反復子パッケージ変数のインスタンスを指定します。

[*data*]
最初のハッシュアイテムに関連付けられているデータをコピーする変数を指定します。

参照項目: “DS2 変数リスト” (*SAS DS2 プログラマガイド*)

詳細

FIRST メソッドは、ハッシュパッケージの最初のデータアイテムを返します。ハッシュパッケージをインスタンス化するときに、DECLARE PACKAGE ステートメント、_NEW_演算子、または ORDERED メソッドで **YES** または **ASCENDING** を指定した場合、返されるデータアイテムは'最小'キー(最小の数値または最初のアルファベット文字)を持つものです。これは、データアイテムがハッシュパッケージでキーと 値の昇順で並べ替えられているために発生します。NEXT メソッドを繰り返し呼び出すと、ハッシュパッケージが繰り返し走査され、キーの昇順でデータアイテムが返されます。

逆に、ハッシュパッケージをインスタンス化するときに、DECLARE PACKAGE ステートメント、_NEW_演算子、または ORDERED メソッドで **DESCENDING** を指定した場合、返されるデータアイテムは'最大'キー(最小の数値または最後のアルファベット文字)を持つものです。これは、データアイテムがハッシュパッケージでキーと 値の降順で並べ替えられているために発生します。NEXT メソッドを繰り返し呼び出すと、ハッシュパッケージが繰り返し走査され、キーの降順でデータアイテムが返されます。

LAST メソッドを使用して、ハッシュパッケージの最後のデータアイテムを返します。

注: FIRST メソッドは、データ変数をデータアイテムの値に設定し、メソッド呼び出し後に使用できるようにします。

FIRST メソッドにデータ変数を渡すには、2 つの方法があります。

- 暗黙の変数メソッド(形式 1)

  データ変数は FIRST メソッドの呼び出しで暗示され、指定する必要はありません。

- 変数リストメソッド(形式 2)

  指定されたデータ変数は、FIRST メソッドに明示的に渡されます。

例

次の例では、ハッシュパッケージ内の別の場所で新しい反復を開始するときに、FIRST、NEXT、PREV、LAST メソッドを使用します。

```
data _null_;
  dcl double x y rc;
  dcl package hash h([x], [y]);
  dcl package hiter hi('h');
  method init();
    do x = 1 to 10;
      y = 2*x;
      rc = h.add([x], [y]);
    end;
    do while (hi.next([y]) = 0);
      put y=;
    end;
```

```
    put;
    hi.first([y]);
    put y=;
    do while (hi.next([y]) = 0);
      put y=;
    end;
    put;
    do while (hi.prev([y]) = 0);
      put y=;
    end;
    put;
    hi.last([y]);
    put y=;
    do while (hi.prev([y]) = 0);
      put y=;
    end;
  end;
enddata;
run;
```

次の行が SAS ログに書き込まれます。

```
y=18
y=10
y=14
y=20
y=2
y=4
y=6
y=8
y=12
y=16

y=18
y=10
y=14
y=20
y=2
y=4
y=6
y=8
y=12
y=16

y=16
y=12
y=8
y=6
y=4
y=2
y=20
y=14
y=10
y=18

y=16
y=12
y=8
y=6
y=4
y=2
y=20
y=14
y=10
y=18
```

関連項目

メソッド:

- “LAST メソッド” (1691 ページ)
- “ORDERED メソッド” (1703 ページ)

演算子:

- “_NEW_ 演算子: ハッシュパッケージ” (1694 ページ)

ステートメント:

- “DECLARE PACKAGE ステートメント: ハッシュパッケージ” (1656 ページ)

HASHEXP メソッド

ハッシュパッケージの内部テーブルサイズを定義します。ハッシュテーブルのサイズは 2^n です。

適用対象: ハッシュパッケージ

注: ハッシュインスタンスに渡されるすべての変数は、グローバル変数である必要があります。

構文

package.**HASHEXP**(*exponent*);

必須引数

package
ハッシュパッケージ変数のインスタンスを指定します。

exponent
内部テーブルサイズの 2 の累乗を指定します。

デフォルト 8

データの種類 INTEGER

詳細

HASHEXP メソッドに指定された値は、ハッシュテーブルサイズを作成するための 2 の累乗の指数として使用されます。たとえば値が 4 の場合、ハッシュテーブルサイズは 2^4、つまり 16 に相当します。*exponent* の最大値は 16 で、これはハッシュテーブルサイズ 2^{16} に相当します。

ハッシュテーブルサイズは、保存できるアイテムの数と同じではありません。ハッシュテーブルをコンテナーの配列と考えてください。16 のハッシュテーブルサイズには 16 個のコンテナーがあります。各コンテナーは、無限の数のアイテムを保持できます。ハッシュテーブルの効率は、アイテムをコンテナーにマッピングしたり、コンテナーからアイテムを取得したりするハッシュ関数の機能にあります。

ハッシュパッケージルックアップルーチンの効率を最大化するには、ハッシュパッケージ内のデータ量に応じてハッシュテーブルのサイズを設定する必要があります。最良の結果が得られるまで、さまざまな *exponent* 値を試してください。たとえば、ハッシュパッケージに 100 万個のアイテムが含まれている場合、16 のハッシュテーブルサイズ(hashexp = 4)はあまり効率的ではありません。512 または 1024 (hashexp = 9 または 10)のハッシュテーブルサイズを使用すると、パフォーマンスが向上します。

注: または、DECLARE PACKAGE ステートメントの *hashexp* パラメーターまたは _NEW_演算子を使用して、ハッシュテーブルのサイズを指定することもできます。

関連項目

- “ハッシュパッケージの初期化データの提供” (*SAS DS2 プログラマガイド*)

演算子:

- “_NEW_ 演算子: ハッシュパッケージ” (1694 ページ)

ステートメント:

- “DECLARE PACKAGE ステートメント: ハッシュパッケージ” (1656 ページ)

HAS_NEXT メソッド

現在のキーの複数データアイテムリストに次のアイテムがあるかどうかを判別します。

適用対象: ハッシュパッケージ

構文

package.**HAS_NEXT**( );

必須引数

package
ハッシュパッケージ変数のインスタンスを指定します。

詳細

キーに複数のデータアイテムがある場合は、HAS_NEXT メソッドを使用して、現在のキーの複数データアイテムリストに次のアイテムがあるかどうかを判別できます。別のアイテムがある場合、メソッドはゼロ値を返します。それ以外の場合は、ゼロ以外の値を返します。

FIND メソッドは、キーがハッシュパッケージに存在するかどうかを判別します。HAS_NEXT メソッドは、複数のデータアイテムがキーに関連付けられているかどうかを判別します。キーに別のデータアイテムがあることが判明した場合、FIND_NEXT メソッドを使用してそのデータアイテムを取得できます。FIND_NEXT メソッドは、データ変数をデータアイテムの値に設定し、メソッド呼び出し後に使用できるようにします。データアイテムリストに入ると、HAS_NEXT メソッドと

FIND_NEXT メソッドに加えて、HAS_PREV メソッドと FIND_PREV メソッドを使用してリストを走査できます。

例: データアイテムの検索

この例では、1 つのキーに複数のデータアイテムがあるハッシュパッケージを 作成します。FIND メソッドと HAS_NEXT メソッドを使用して、キー 1、5、および 11 で複数のデータアイテムを 検索します。

```
data testdata (overwrite=yes);
  dcl double k v;
  method init();
    k=1; v=1; output;
    k=2; v=4; output;
    k=3; v=9; output;
    k=4; v=16; output;
    k=5; v=25; output;
    k=6; v=36; output;
    k=7; v=49; output;
    k=8; v=64; output;
    k=9; v=81; output;
    k=10; v=100; output;
    k=2; v=3; output;
    k=4; v=4242; output;
    k=4; v=0; output;
    k=5; v=25; output;
  end;
enddata;
run;


data _null_;
  dcl double k v rc;
  dcl package hash h(8, 'testdata', 'ascending', '', '', 'multidata');
  method init();
    h.defineKey('k');
    h.defineData('k');
    h.defineData('v');
    h.defineDone();
  end;
  method term();
    do k = 1, 5, 11;
      put k=;
      rc = h.find();
      put '..find=' rc;
      /***************************/
      do while (rc=0);
        put '    found:' k= v=;
        rc = h.has_next();
        put '..has_next=' rc;
        if (rc = 0) then do;
          h.find_next();
          /* ignore return code */
          /* already know it from 'has_next', above */
```

```
            end;
          end;
          /**************************/
        end;
      end;
    enddata;
    run;
```

次の行が SAS ログに書き込まれます。

```
k=1
..find= 0
   found: k=1 v=1
..has_next= 4
k=5
..find= 0
   found: k=5 v=25
..has_next= 0
   found: k=5 v=25
..has_next= 4
k=11
..find= 4
```

関連項目

- *“一意でないキーとデータのペア” (SAS DS2 プログラマガイド)*

メソッド:

- “FIND メソッド” (1674 ページ)
- “FIND_NEXT メソッド” (1676 ページ)
- “HAS_PREV メソッド” (1687 ページ)

HAS_PREV メソッド

現在のキーの複数データアイテムリストに前のアイテムがあるかどうかを判別します。

適用対象: ハッシュパッケージ

構文

package.**HAS_PREV**( );

必須引数

package
ハッシュパッケージ変数のインスタンスを指定します。

詳細

キーに複数のデータアイテムがある場合は、HAS_PREV メソッドを使用して、現在のキーの複数データアイテムリストに前のアイテムがあるかどうかを判別できます。前のアイテムがある場合、メソッドはゼロを返します。それ以外の場合は、ゼロ以外の値を返します。

FIND メソッドは、キーがハッシュパッケージに存在するかどうかを判別します。HAS_NEXT メソッドは、複数のデータアイテムがキーに関連付けられているかどうかを判別します。キーに前のデータアイテムがあることが判明した場合、FIND_PREV メソッドを使用してそのデータアイテムを取得できます。FIND_PREV メソッドは、データ変数をデータアイテムの値に設定し、メソッド呼び出し後に使用できるようにします。データアイテムリストに入ると、HAS_NEXT メソッドと FIND_NEXT メソッドに加えて、HAS_PREV メソッドと FIND_PREV メソッドを使用してリストを走査できます。

例

“例: 要約値の取得” (1725 ページ)を参照してください。

関連項目

- “一意でないキーとデータのペア” (*SAS DS2 プログラマガイド*)

メソッド:

- “FIND メソッド” (1674 ページ)
- “FIND_PREV メソッド” (1679 ページ)
- “HAS_NEXT メソッド” (1685 ページ)

ITEM_SIZE 属性

ハッシュパッケージ内のアイテムのサイズをバイト単位で返します。

適用対象: ハッシュパッケージ

構文

variable-name=*package*.**ITEM_SIZE**;

必須引数

variable-name
メソッドの完了後のハッシュパッケージにあるアイテムのサイズを 含む変数の名前を指定します。

package
ハッシュパッケージ変数のインスタンスを指定します。

詳細

ITEM_SIZE 属性は、アイテムのサイズ(バイト単位)、キー変数、データ変数、一部の内部情報を返します。ハッシュパッケージが使用しているメモリを推定するには、ITEM_SIZE 属性と NUM_ITEMS 属性を指定します。ITEM_SIZE 属性は、ハッシュパッケージに必要な初期オーバーヘッドを反映したものではなく、必要な内部調整も考慮されていません。したがって、ITEM_SIZE を使用しても正確なメモリ使用量は得られませんが、適切な概算値が返されます。

例

例については、“NUM_ITEMS 属性” (1701 ページ)を参照してください。

関連項目

属性:

- “NUM_ITEMS 属性” (1701 ページ)

KEYS メソッド

変数リストを使用して、ハッシュパッケージのキー変数を定義します。

適用対象: ハッシュパッケージ

注: かっこの前のエスケープ文字(\)は、構文にかっこが必要であることを示します。
ハッシュインスタンスに渡されるすべての 変数は、グローバル変数である必要があります。

構文

package.**KEYS**(\[*keys*\]);

必須引数

package
ハッシュパッケージ変数のインスタンスを指定します。

[*keys*]
変数リストを使用してキー変数の名前を指定します。

参照項目: “DS2 変数リスト” (*SAS DS2 プログラマガイド*)

詳細

ハッシュパッケージは、一意のルックアップキーを使用してデータを保存および取得します。キーとデータは変数であり、ドット表記メソッド呼び出しを使用してハッシュパッケージを初期化するために使用します。

変数リストを使用して、KEYS メソッドでハッシュパッケージのキー変数を指定します。すべてのキー変数とデータ変数を定義したら、ハッシュパッケージの初期化を完了するために DEFINEDONE メソッドを呼び出す必要があります。

注: または、DEFINEKEYS メソッドや DECLARE PACKAGE ステートメントのコンストラクターを使用して、キー変数を指定することもできます。

注: キー変数のみを含み、データ変数を含まないハッシュパッケージを作成できます。これはキーのみのハッシュパッケージです。

キーとデータは、任意の数の文字変数または数値変数で構成されます。

例

次の例では、ハッシュパッケージを作成し、キー変数とデータ変数を定義します。

```
declare hash h();
rc = h.keys([k]);
rc = h.data([d]);
rc = h.definedata();
end;
```

関連項目

- “キー変数とデータ変数の定義” (*SAS DS2 プログラマガイド*)
- “暗黙の変数と変数リストのメソッド” (*SAS DS2 プログラマガイド*)

メソッド:

- “DEFINEKEY メソッド” (1670 ページ)

- “DEFINEDONE メソッド” (1669 ページ)

ステートメント:

- “DECLARE PACKAGE ステートメント: ハッシュパッケージ” (1656 ページ)

LAST メソッド

基になるハッシュパッケージの最後の値を返します。

適用対象: ハッシュ反復子パッケージ

注: かっこの前のエスケープ文字(\)は、構文にかっこが必要であることを示します。

構文

形式 1: *package*.**LAST**( );

形式 2: *package*.**LAST**(\[*data*\]);

必須引数

package
ハッシュ反復子パッケージ変数のインスタンスを指定します。

[*data*]
最初のハッシュアイテムに関連付けられているデータをコピーする変数を指定します。

参照項目: “DS2 変数リスト” (*SAS DS2 プログラマガイド*)

詳細

LAST メソッドは、ハッシュパッケージの最後のデータアイテムを返します。ハッシュパッケージをインスタンス化するときに、DECLARE PACKAGE ステートメント、_NEW_演算子、または ORDERED メソッドで **YES** または **ASCENDING** を指定した場合、返されるデータアイテムは、最大の数値を持つデータアイテム、または最後のアルファベット文字を持つデータアイテムになります。これは、データアイテムがハッシュパッケージでキーと値の昇順で並べ替えられるためです。

逆に、ハッシュパッケージをインスタンス化するときに、DECLARE PACKAGE ステートメント、_NEW_演算子、または ORDERED メソッドで **DESCENDING** を指定した場合、返されるデータアイテムは、最小の数値を持つデータアイテム、または最初のアルファベット文字を持つデータアイテムになります。これは、データアイテムがハッシュパッケージでキーと値の降順で並べ替えられるためです。

FIRST メソッドを使用して、ハッシュパッケージの最初のデータアイテムを返します。

LAST メソッドにデータ変数を渡すには、2 つの方法があります。

- 暗黙の変数メソッド(形式 1)

  データ変数は LAST メソッドの呼び出しで暗示され、指定する必要はありません。
- 変数リストメソッド(形式 2)

  指定されたデータ変数は、LAST メソッドに明示的に渡されます。

例

例については、“FIRST メソッド” (1680 ページ)を参照してください。

関連項目

- “DS2 変数リスト” (*SAS DS2 プログラマガイド*)

メソッド:

- “FIRST メソッド” (1680 ページ)
- “ORDERED メソッド” (1703 ページ)

演算子:

- “_NEW_ 演算子: ハッシュパッケージ” (1694 ページ)

ステートメント:

- “DECLARE PACKAGE ステートメント: ハッシュパッケージ” (1656 ページ)

MULTIDATA メソッド

キーごとに複数のデータアイテムを許可するかどうかを指定します。

適用対象: ハッシュパッケージ

注: ハッシュインスタンスに渡されるすべての変数は、グローバル変数である必要があります。

構文

package.**MULTIDATA**(['*option*']);

必須引数

package
ハッシュパッケージ変数のインスタンスを指定します。

オプション引数

'*option*'
option には次のいずれかの値を指定できます。

'Y'
キーごとに複数のデータアイテムを許可します。

別名 'Y'または'MULTIDATA'

'N'
重複キーが見つかった場合、ログにエラーを報告します。

別名 'N'または'SINGLEDATA'

デフォルト NO

詳細

デフォルトでは、ハッシュパッケージ内のすべてのキーは一意です。これは、キーごとに 1 セットのデータ変数が存在することを意味します。状況によっては、ハッシュパッケージに重複キーを含めること、つまり、複数のデータ変数のセットを 1 つのキーに関連付けることが必要になる場合があります。

テーブルに重複キーが含まれている場合、デフォルトでは、最初のインスタンスはハッシュパッケージに保存され、後続のインスタンスは無視されます。最後のインスタンスをハッシュパッケージに保存するには、DUPLICATE メソッドを使用します。重複キーがある場合、DUPLICATE メソッドは SAS ログにもエラーを書き込みます。

ただし、MULTIDATA パラメーターまたはメソッドを使用する場合、ハッシュパッケージではキーごとに複数の値を保存できます。ハッシュパッケージは、キーに関連付けられたリストに複数の値を保持します。このリストは、HAS_NEXT や FIND_NEXT などのいくつかのメソッドを使用して走査および操作できます。

注: または、DECLARE PACKAGE ステートメントの *multidata* パラメーターまたは _NEW_演算子を使用して、各キーに複数のデータアイテムを許可するかどうかを指定できます。

関連項目

- “一意でないキーとデータのペア” (*SAS DS2 プログラマガイド*)
- “ハッシュパッケージの初期化データの提供” (*SAS DS2 プログラマガイド*)

メソッド:

- “DUPLICATE メソッド” (1672 ページ)

演算子:

- “_NEW_ 演算子: ハッシュパッケージ” (1694 ページ)

ステートメント:

- “DECLARE PACKAGE ステートメント: ハッシュパッケージ” (1656 ページ)

NEW 演算子: ハッシュパッケージ

ハッシュパッケージのインスタンスを構築します。

注: 構文規則の中かっこは、構文のグループ化を示します。中かっこの前のエスケープ文字(\)は、構文に中かっこが必要であることを示します。*sql-text* は中かっこ({ })で囲む必要があります。

かっこの前のエスケープ文字(\)は、構文にかっこが必要であることを示します。

ハッシュインスタンスに渡されるすべての変数は、グローバル変数である必要があります。

構文

形式 1: *package-variable*=**_NEW_ HASH**();

形式 2: *package-variable*=**_NEW_ HASH**
(*hashexp*, '*datasource*', '*ordered*', '*duplicate* ', '*suminc*', '*multidata*');

形式 3: *package-variable*=**_NEW_ HASH**
(\[*keys*\], \[*data*\], [*hashexp*, '*datasource*', '*ordered*', '*duplicate* ', '*suminc*', '*multidata*']);

形式 4: *package-variable*=**_NEW_ HASH**
(\[*keys*\], [*hashexp*, {'*datasource*' | \{*sql-text*\} }, '*ordered*', '*duplicate* ', '*suminc*', '*multidata*']);

必須引数

package-variable
パッケージのインスタンスを参照できる名前を指定します。

オプション引数

[*keys*]
変数リストを使用してキー値を指定します。

参照項目: “DS2 変数リスト” (*SAS DS2 プログラマガイド*)

[*data*]
変数リストを使用してデータ変数を指定し、それらを指定されたキーに関連付けます。

参照項目: "DS2 変数リスト" (*SAS DS2 プログラマガイド*)

{*sql-text*}
テーブル行のセットに解決される有効な FedSQL SELECT ステートメントです。

制限事項 この引数は CAS サーバーではサポートされていません。

要件 FedSQL クエリは中かっこ({ })で囲む必要があります。

注 FedSQL クエリは、{SELECT <select-list> FROM <table-specification>;}の形式で指定されます。詳細については、*SAS FedSQL 言語リファレンス*の SELECT ステートメントを参照してください。

引数

hashexp
ハッシュパッケージの内部テーブルサイズです。ここで、ハッシュテーブルのサイズは 2^n です。

hashexp の値は、ハッシュテーブルサイズを作成するための 2 の累乗の指数として使用されます。たとえば、hashexp の値が 4 の場合、ハッシュテーブルサイズは 2^4、つまり 16 に相当します。hashexp の最大値は 16 で、これはハッシュテーブルサイズ 2^{16} に相当します。

ハッシュテーブルサイズは、保存できるアイテムの数と同じではありません。ハッシュテーブルをコンテナーの配列と考えてください。16 のハッシュテーブルサイズには 16 個のコンテナーがあります。各コンテナーは、無限の数のアイテムを保持できます。ハッシュテーブルの効率は、アイテムをコンテナーにマッピングしたり、コンテナーからアイテムを取得したりするハッシュ関数の機能にあります。

ハッシュパッケージルックアップルーチンの効率を最大化するには、ハッシュパッケージ内のデータ量に応じてハッシュテーブルのサイズを設定する必要があります。最良の結果が得られるまで、さまざまな hashexp 値を試してください。たとえば、ハッシュパッケージに 100 万個のアイテムが含まれている場合、16 のハッシュテーブルサイズ(hashexp = 4)はあまり効率的ではありません。512 または 1024 (hashexp = 9 または 10)のハッシュテーブルサイズを使用すると、パフォーマンスが向上します。

要件 –1 のプレースホルダーまたは *hashexp* の値のいずれかを入力する必要があります。プレースホルダーを入力すると、デフォルト値の 8 が使用されます。これは、ハッシュテーブルサイズが 2^8 または 256 に相当します。

データの種類 INTEGER

'*datasource*'
ハッシュパッケージにロードするテーブルの名前です。

テーブルの名前は、リテラルまたは文字変数にすることができます。テーブル名は、単一引用符で囲む必要があります。

制限事項 datasource 引数が null でない場合、CAS サーバーでは _NEW_ 演算子はサポートされていません。

要件 引用符で囲まれた空の文字列のプレースホルダー('')または *datasource* の値を入力する必要があります。プレースホルダーが入力されている場合、ハッシュはデータソースからロードされません。

'*ordered*'
ハッシュ反復子パッケージでハッシュパッケージを使用する場合、またはハッシュパッケージの OUTPUT メソッドを使用する場合に、データがキーと値の順序で返されるかどうか、またはどのように返されるかを指定します。有効な値は次のとおりです。

'ASCENDING'
データはキーと値の昇順で返されます。'**ASCENDING**'を指定することは、'**YES**'を指定することと同じです。

別名 'A'

'DESCENDING'
データはキーと値の降順で返されます。

別名 'D'

'YES'
データはキーと値の昇順で返されます。'**YES**'を指定することは、'**ASCENDING**'を指定することと同じです。

'NO'
データは未定義の順序で返されます。

要件 引用符で囲まれた空の文字列のプレースホルダー('')または *ordered* の値を入力する必要があります。プレースホルダーを入力すると、デフォルトの順序である'NO'が使用されます。

'*duplicate*'
テーブルをハッシュパッケージにロードするときに重複キーを無視するかどうかを決定します。デフォルトでは、最初のキーを保存し、後続のすべての重複を無視します。有効な値は次のとおりです。

'REPLACE'
最後の重複キーレコードを保存します。

'ERROR'
重複キーが見つかった場合、ログにエラーを報告します。

'ADD'
見つかった最初のキーレコードを保存し、重複は保存しません。

要件 引用符で囲まれた空の文字列のプレースホルダー('')または *duplicate* の値を入力する必要があります。プレースホルダーを入力すると、デフォルトの'ADD'が使用されます。

参照項目: “例 3: ハッシュパッケージでの Duplicate パラメーターの使用” (1662 ページ)

'*suminc*'
ハッシュパッケージキーの要約カウントを維持する変数を指定します。

要件 引用符で囲まれた空の文字列のプレースホルダー('')または *suminc* の値を入力する必要があります。

注 この変数は、合計増分、つまり、キーへの各参照のキー要約に追加する量を保持します。*suminc* 値は、SUM 関数のように、欠損値または null 値をゼロとして扱います。たとえば、キー要約は、変数の現在の値を使用して変更されます。

'*multidata*'
キーごとに複数のデータアイテムを許可するかどうかを指定します。有効な値は次のとおりです。

'YES'
キーごとに複数のデータアイテムを許可します。

別名 'Y'または'MULTIDATA'

'NO'
キーごとに 1 つのデータアイテムのみを許可します。

別名 'N'または'SINGLEDATA'

要件 引用符で囲まれた空の文字列のプレースホルダー('')または ordered の値を入力する必要があります。プレースホルダーを入力すると、デフォルトの順序である'NO'が使用されます。

詳細

DS2 パッケージは、特定のインスタンスを構築して他の DS2 プログラムで使用できる変数とメソッドのコレクションです。

DECLARE PACKAGE ステートメントを使用して、ハッシュパッケージ変数を宣言します。ハッシュパッケージ変数を宣言した後、_NEW_演算子を使用してハッシュパッケージのインスタンスをインスタンス化し、新しいパッケージインスタンスをハッシュパッケージ変数に割り当てます。

```
declare package hash myhash();
myhash = _new_ hash();
```

特定のハッシュパッケージインスタンスは、一連のキー変数、一連のデータ変数、およびオプションの初期化データによって定義されます。ハッシュパッケージインスタンスは、構築時に完全に定義することも、構築時と後続の一連のメソッド呼び出しで定義することもできます。ハッシュパッケージインスタンスの構築時にキー変数とデータ変数が指定されている場合(形式 3 および 4)、ハッシュインスタンスは完全な定義として構築されます。ハッシュパッケージインスタンスの構築時にキー変数とデータ変数が提供されない場合(形式 2)、ハッシュインスタンスの追加の定義

は、後続の一連のメソッド呼び出しとそれに続く単一の DEFINEDONE メソッド呼び出しで指定できます。DEFINEDONE メソッドは、キー変数、データ変数、およびその他の初期化データの指定が完了したことを示します。

たとえば、ハッシュパッケージのコンストラクター構文のパラメーターを使用して、初期化データを提供できます。

```
declare package hash h;
h = _new_ hash(0, 'mytable', 'yes', 'replace', 'sumnum', 'y');
```

注: DECLARE PACKAGE ステートメントコンストラクターを使用して、ハッシュまたはハッシュ反復子パッケージを 1 つのステップで宣言およびインスタンス化できます。詳細については、“コンストラクターを使用したハッシュインスタンスの定義” (*SAS DS2 プログラマガイド*)および“DECLARE PACKAGE ステートメント: ハッシュパッケージ” (1656 ページ)を参照してください。

注: パッケージ変数は、すべての 変数スコーピングルールの対象となります。詳細については、“パッケージ、スコープ、および有効期間” (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- “ハッシュパッケージの使用” (*SAS DS2 プログラマガイド*)
- “ハッシュ反復子パッケージの使用” (*SAS DS2 プログラマガイド*)
- “パッケージのコンストラクターとデストラクター” (*SAS DS2 プログラマガイド*)

演算子:

- “_NEW_ 演算子: ハッシュ反復子パッケージ” (1698 ページ)

ステートメント:

- “DECLARE PACKAGE ステートメント: ハッシュパッケージ” (1656 ページ)

NEW 演算子: ハッシュ反復子パッケージ

ハッシュ反復子パッケージのインスタンスを作成します。

注: かっこの前のエスケープ文字(\)は、構文にかっこが必要であることを示します。

構文

package-variable = **_NEW_ HITER** ('*hash-name*');

必須引数

package-variable
パッケージのインスタンスを参照できる名前を指定します。

'*hash-name*'
ハッシュ反復子パッケージに関連付けられているハッシュパッケージを指定します。

要件 ハッシュ反復子パッケージを作成する前に、ハッシュパッケージを宣言してインスタンス化する必要があります。

詳細

DS2 パッケージは、特定のインスタンスを構築して他の DS2 プログラムで使用できる変数とメソッドのコレクションです。

DECLARE PACKAGE ステートメントを使用して、ハッシュ反復子パッケージ変数を宣言します。ハッシュ反復子パッケージ変数を宣言した後、_NEW_演算子を使用してハッシュ反復子パッケージのインスタンスをインスタンス化し、新しいパッケージインスタンスをハッシュ反復子パッケージ変数に割り当てます。

```
declare package hiter myiter;
myiter = _new_ hiter('myhash');
```

DECLARE PACKAGE と_NEW_演算子を使用してハッシュ反復子パッケージを宣言およびインスタンス化する 2 ステップのプロセスのかわりに、コンストラクターメソッドとして DECLARE PACKAGE ステートメントを使用することにより、パッケージを 1 ステップで宣言およびインスタンス化できます。これは、DECLARE PACKAGE ステートメントのみを使用した同じ例です。

```
declare package hiter myiter('myhash');
```

注: パッケージ変数は、すべての変数スコーピングルールの対象となります。詳細については、“パッケージ、スコープ、および有効期間” (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- “ハッシュ反復子パッケージの使用” (*SAS DS2 プログラマガイド*)
- “ハッシュパッケージの使用” (*SAS DS2 プログラマガイド*)
- “パッケージのコンストラクターとデストラクター” (*SAS DS2 プログラマガイド*)

演算子:

- “_NEW_ 演算子: ハッシュパッケージ” (1694 ページ)

ステートメント:

- “DECLARE PACKAGE ステートメント: ハッシュ反復子パッケージ” (1665 ページ)

NEXT メソッド

基になるハッシュパッケージの次の値を返します。

適用対象: ハッシュ反復子パッケージ

注: かっこの前のエスケープ文字(\)は、構文にかっこが必要であることを示します。

構文

形式 1: *package*.**NEXT**( );

形式 2: *package*.**NEXT**(\[*data*\]);

必須引数

package
ハッシュ反復子パッケージ変数のインスタンスを指定します。

[*data*]
次のハッシュアイテムに関連付けられているデータをコピーする変数を指定します。

参照項目: “DS2 変数リスト” (*SAS DS2 プログラマガイド*)

詳細

NEXT メソッドを繰り返し使用して、ハッシュパッケージを走査し、キー順にデータアイテムを返します。FIRST メソッドは、ハッシュパッケージの最初のデータアイテムを返します。PREV メソッドを使用すると、ハッシュパッケージの前のデータアイテムを返すことができます。

NEXT メソッドにデータ変数を渡すには、2 つの方法があります。

- 暗黙の変数メソッド(形式 1)

  データ変数は NEXT メソッドの呼び出しで暗示され、指定する必要はありません。

- 変数リストメソッド(形式 2)

  指定されたデータ変数は、NEXT メソッドに明示的に渡されます。

例

例については、“FIRST メソッド” (1680 ページ)を参照してください。

関連項目

メソッド:

- “FIRST メソッド” (1680 ページ)
- “PREV メソッド” (1707 ページ)

NUM_ITEMS 属性

ハッシュパッケージ内のアイテムの数を返します。

適用対象: ハッシュパッケージ

構文

variable-name=*package*.**NUM_ITEMS**;

必須引数

variable-name
メソッドの完了後のハッシュパッケージにあるアイテムの数を含む変数の名前を指定します。

package
ハッシュパッケージ変数のインスタンスを指定します。

詳細

NUM_ITEMS 属性は、ハッシュテーブルに保存されているキーとデータのペアの数を返します。

例

次の例では、NUM_ITEMS 属性を使用してハッシュパッケージ内のアイテムの数をカウントし、ITEM_SIZE 属性を使用してハッシュパッケージ内のアイテムのサイズを報告します。

```
data _null_;
  dcl int item_size num_items;
  dcl double x;
  dcl timestamp t;
  dcl package hash h(8, '', 'yes');
  dcl package hiter hi('h');
  method init();
    rc = h.defineKey('t');
    rc = h.defineData('t');
    rc = h.defineData('x');
    rc = h.defineDone();

    item_size = h.item_size;
    num_items = h.num_items;
    put item_size=;
    put num_items=;
    put;

    t = timestamp '1927-08-24 12:51:36.00'; x = 1; h.add();
    t = timestamp '1928-08-24 12:51:36.00'; x = 1; h.add();
    t = timestamp '1929-08-24 12:51:36.00'; x = 1; h.add();
    t = timestamp '1929-09-24 12:51:36.00'; x = 1; h.add();
    t = timestamp '1929-09-25 12:51:36.00'; x = 1; h.add();
    t = timestamp '1929-09-25 13:51:36.00'; x = 1; h.add();
    t = timestamp '1929-09-25 13:52:36.00'; x = 1; h.add();
    t = timestamp '1929-09-25 13:52:37.00'; x = 1; h.add();
    t = timestamp '1929-09-25 13:52:37.01'; x = 1; h.add();
    t = timestamp '1930-09-25 13:52:37.01'; x = 1; h.add();
    t = timestamp '1930-10-25 13:52:37.01'; x = 1; h.add();

    num_items = h.num_items;
    put num_items=;

   do while (hi.next() = 0);
     put t= x=;
   end;
   put;

   t = timestamp '1929-09-25 13:51:36.00'; x = 1; h.remove();
   t = timestamp '1929-09-25 13:52:36.00'; x = 1; h.remove();

   num_items = h.num_items;
   put num_items=;

   do while (hi.next() = 0);
     put t= x=;
   end;
 end;
enddata;
run;
```

次の行が SAS ログに書き込まれます。

```
item_size=72
num_items=0

num_items=11
t=1927-08-24 12:51:36 x=1
t=1928-08-24 12:51:36 x=1
t=1929-08-24 12:51:36 x=1
t=1929-09-24 12:51:36 x=1
t=1929-09-25 12:51:36 x=1
t=1929-09-25 13:51:36 x=1
t=1929-09-25 13:52:36 x=1
t=1929-09-25 13:52:37 x=1
t=1929-09-25 13:52:37.010000000 x=1
t=1930-09-25 13:52:37.010000000 x=1
t=1930-10-25 13:52:37.010000000 x=1

num_items=9
t=1927-08-24 12:51:36 x=1
t=1928-08-24 12:51:36 x=1
t=1929-08-24 12:51:36 x=1
t=1929-09-24 12:51:36 x=1
t=1929-09-25 12:51:36 x=1
t=1929-09-25 13:52:37 x=1
t=1929-09-25 13:52:37.010000000 x=1
t=1930-09-25 13:52:37.010000000 x=1
t=1930-10-25 13:52:37.010000000 x=1
```

関連項目

属性:

- “ITEM_SIZE 属性” (1688 ページ)

ORDERED メソッド

ハッシュ反復子パッケージでハッシュパッケージを使用する場合、またはハッシュパッケージの OUTPUT メソッドを使用する場合に、データがキーと値の順序で返されるかどうか、またはどのように返されるかを指定します。

適用対象: ハッシュパッケージ

注: ハッシュインスタンスに渡されるすべての変数は、グローバル変数である必要があります。

構文

package.**ORDERED**('*option*');

必須引数

package
ハッシュパッケージ変数のインスタンスを指定します。

option
option には次のいずれかの値を指定できます。

'ASCENDING'
データはキーと値の昇順で返されます。'**ASCENDING**'を指定することは、'**YES**'を指定することと同じです。

別名 'A'

'DESCENDING'
データはキーと値の降順で返されます。

別名 'D'

'YES'
データはキーと値の昇順で返されます。'**YES**'を指定することは、'**ASCENDING**'を指定することと同じです。

'NO'
データは未定義の順序で返されます。

デフォルト NO

詳細

ハッシュパッケージをインスタンス化するときに、ORDERED メソッドで **YES** または **ASCENDING** を指定した場合、返されるデータアイテムは'最小'キー(最小の数値または最初のアルファベット文字)を持つものです。これは、データアイテムがハッシュパッケージでキーと値の昇順で並べ替えられているために発生します。NEXT メソッドを繰り返し呼び出すと、ハッシュパッケージが繰り返し走査され、キーの昇順でデータアイテムが返されます。

ハッシュパッケージをインスタンス化するときに、ORDERED メソッドで **DESCENDING** パラメーターを指定した場合、返されるデータアイテムは'最大'キー(最大の数値または最後のアルファベット文字)を持つものです。これは、データアイテムがハッシュパッケージでキーと値の降順で並べ替えられているために発生します。NEXT メソッドを繰り返し呼び出すと、ハッシュパッケージが繰り返し走査され、キーの降順でデータアイテムが返されます。

FIRST メソッドを使用して、ハッシュパッケージの最初のデータアイテムを返します。LAST メソッドを使用して、ハッシュパッケージの最後のデータアイテムを返します。

注: または、DECLARE PACKAGE ステートメントの *ordered* パラメーターまたは _NEW_演算子を使用して、データがキーと値の順序で返されるかどうかを指定できます。

関連項目

- “暗黙の変数と変数リストのメソッド” (*SAS DS2 プログラマガイド*)

メソッド:

- “FIRST メソッド” (1680 ページ)
- “LAST メソッド” (1691 ページ)

演算子:

- “_NEW_ 演算子: ハッシュパッケージ” (1694 ページ)

ステートメント:

- “DECLARE PACKAGE ステートメント: ハッシュパッケージ” (1656 ページ)

OUTPUT メソッド

ハッシュパッケージ内のデータを含むテーブルを作成します。

適用対象: ハッシュパッケージ

制限事項: このメソッドは CAS サーバーではサポートされていません。

構文

package.**OUTPUT** (['*output-table*[']);

必須引数

package
ハッシュ反復子パッケージ変数のインスタンスを指定します。

['*output-table*[']
出力テーブルの名前を指定します。

ハッシュテーブルの名前は、リテラルまたは文字変数にすることができます。

ヒント テーブルの名前は、リテラルまたは文字変数にすることができます。リテラルを使用する場合は、テーブル名を単一引用符で囲む必要があります。

詳細

ハッシュパッケージキーは、出力テーブルの一部として自動的に保存されません。キーを出力テーブルに含めるには、DEFINEDATA メソッド、DATA メソッド、または DECLARE PACKAGE ステートメントを使用して、データアイテムとして定義する必要があります。

例

```
data a(overwrite=yes);
  dcl double x;
    method init();
      do x = 1 to 10;
        output;
      end;
    end;
enddata;
run;
data _null_;
  method init();
    dcl package hash h(4, 'a');
    rc = h.defineData('x');
    rc = h.defineKey('x');
    rc = h.defineDone();
    x = 11;
    h.add();
    x = 12;
    h.add();
    x = 13;
    h.add();
    x = 14;
    h.add();
    rc = h.output('out');
  end;
enddata;
run;
```

関連項目

メソッド:

- “DATA メソッド” (1652 ページ)
- “DEFINEDATA メソッド” (1667 ページ)

ステートメント:

- “DECLARE PACKAGE ステートメント: ハッシュパッケージ” (1656 ページ)

PREV メソッド

基になるハッシュパッケージの前の値を返します。

適用対象: ハッシュ反復子パッケージ

注: かっこの前のエスケープ文字(\)は、構文にかっこが必要であることを示します。

構文

形式 1: *package*.**PREV**( );

形式 2: *package*.**PREV**(\[*data*\]);

必須引数

package
ハッシュ反復子パッケージ変数のインスタンスを指定します。

[*data*]
前のハッシュアイテムに関連付けられているデータをコピーする変数を指定します。

参照項目: "DS2 変数リスト" (*SAS DS2 プログラマガイド*)

詳細

PREV メソッドを繰り返し使用して、ハッシュパッケージを走査し、キーの逆順にデータアイテムを返します。FIRST メソッドは、ハッシュパッケージの最初のデータアイテムを返します。LAST メソッドは、ハッシュパッケージの最後のデータアイテムを返します。NEXT メソッドを使用すると、ハッシュパッケージの次のデータアイテムを返すことができます。

PREV メソッドにデータ変数を渡すには、2 つの方法があります。

- 暗黙の変数メソッド(形式 1)

  データ変数は PREV メソッドの呼び出しで暗示され、指定する必要はありません。

- 変数リストメソッド(形式 2)

  指定されたデータ変数は、PREV メソッドに明示的に渡されます。

例

例については、“FIRST メソッド” (1680 ページ)を参照してください。

関連項目

メソッド:

- “FIRST メソッド” (1680 ページ)
- “LAST メソッド” (1691 ページ)
- “NEXT メソッド” (1700 ページ)

REF メソッド

FIND メソッドと ADD メソッドを単一のメソッド呼び出しに統合します。

適用対象: ハッシュパッケージ

注: かっこの前のエスケープ文字(\)は、構文にかっこが必要であることを示します。

構文

形式 1: *package*.**REF**( );

形式 2: *package*.**REF**(\[*keys*\], \[*data*\]);

形式 3: *package*.**REF**(\[*keys*\]);

必須引数

package
ハッシュパッケージ変数のインスタンスを指定します。

[*keys*]
変数リストを使用してキー変数を指定します。

制限事項 キーのみを指定した場合、FIND メソッドはキーのみのハッシュパッケージに対してのみ機能します。

参照項目: “DS2 変数リスト” (*SAS DS2 プログラマガイド*)

[*data*]
ハッシュデータを追加する変数を指定します。

参照項目: “DS2 変数リスト” (*SAS DS2 プログラマガイド*)

詳細

FIND メソッドと ADD メソッドを単一の REF メソッドに統合できます。

キー変数とデータ変数を REF メソッドに渡す方法は 2 つあります。

- 暗黙の変数メソッド(形式 1)

  キー変数とデータ変数は REF メソッドの呼び出しで暗黙的に指定されるため、指定する必要はありません。

- 変数リストメソッド(形式 2 および 3)

  指定されたキー変数とデータ変数は、REF メソッドに明示的に渡されます。ハッシュパッケージにキーのみが含まれている場合は、形式 3 を使用します。

例

```
data _null_;
  dcl double x y;
  dcl int rc;
  dcl package hash h([x], [x y]);
  dcl package hiter hi('h');
  method init();
    x = 7; y = 13;
    rc = h.add([x], [x y]);
    put x= rc=;
    do x = 5 to 10;
      y = 2*x;
      rc = h.ref([x], [x y]);
      put x= rc=;
    end;
    do while (hi.next([x y]) = 0);
      put x= y=;
    end;
  end;
enddata;
run;
```

次の行が SAS ログに書き込まれます。

```
x=7 rc=0
x=5 rc=0
x=6 rc=0
x=7 rc=0
x=8 rc=0
x=9 rc=0
x=10 rc=0
x=9 y=18
x=5 y=10
x=7 y=13
x=10 y=20
x=6 y=12
x=8 y=16
```

関連項目

- “暗黙の変数と変数リストのメソッド” (*SAS DS2 プログラマガイド*)
- “データの格納と取得” (*SAS DS2 プログラマガイド*)

メソッド:

- “ADD メソッド: ハッシュパッケージ” (1646 ページ)
- “CHECK メソッド” (1649 ページ)
- “FIND メソッド” (1674 ページ)

REMOVE メソッド

指定されたキーに関連付けられているデータをハッシュパッケージから削除します。

適用対象: ハッシュパッケージ

注: かっこの前のエスケープ文字(\)は、構文にかっこが必要であることを示します。

構文

形式 1: *package*.**REMOVE**( );

形式 2: *package*.**REMOVE**(\[*keys*\]);

必須引数

package
ハッシュパッケージ変数のインスタンスを指定します。

[*keys*]
変数リストを使用してキー値を指定します。

参照項目: “DS2 変数リスト” (*SAS DS2 プログラマガイド*)

詳細

REMOVE メソッドは、キー変数の値を使用して、ハッシュテーブルにある既存のキーを検索して削除します。

キーを指定してから、REMOVE メソッドを使用して、ハッシュパッケージにあるキーとデータを削除します。

REMOVE メソッドにキー変数を渡すには、2 つの方法があります。

- 暗黙の変数メソッド(形式 1)

キー変数は REMOVE メソッドの呼び出しで暗示され、指定する必要はありません。

- 変数リストメソッド(形式 2)

  指定されたキー変数は、REMOVE メソッドに明示的に渡されます。

注: REMOVE メソッドは、データ変数の値を変更しません。ハッシュパッケージの値のみを削除します。

注: ハッシュパッケージをインスタンス化するときに、DECLARE PACKAGE ステートメント、_NEW_演算子、または MULTIDATA メソッドで **YES** を指定すると、REMOVE メソッドは指定されたキーのすべてのデータアイテムを 削除します。

例

```
data _null_;
  dcl double x rc;
  dcl timestamp t;
  dcl package hash h(0, '', 'yes');
  dcl package hiter hi('h');
  method init();
    rc = h.Keys([t]);
    rc = h.Data([t x]);
    rc = h.defineDone();
    t = timestamp '1927-08-24 12:51:36.00'; x = 1; h.add();
    t = timestamp '1928-08-24 12:51:36.00'; x = 1; h.add();
    t = timestamp '1929-08-24 12:51:36.00'; x = 1; h.add();
    t = timestamp '1929-09-24 12:51:36.00'; x = 1; h.add();
    t = timestamp '1929-09-25 12:51:36.00'; x = 1; h.add();
    t = timestamp '1929-09-25 13:51:36.00'; x = 1; h.add();
    t = timestamp '1929-09-25 13:52:36.00'; x = 1; h.add();
    t = timestamp '1929-09-25 13:52:37.00'; x = 1; h.add();
    t = timestamp '1929-09-25 13:52:37.01'; x = 1; h.add();
    t = timestamp '1930-09-25 13:52:37.01'; x = 1; h.add();
    t = timestamp '1930-10-25 13:52:37.01'; x = 1; h.add();
    t = timestamp '1999-12-01 12:00:00.00'; x = 999;
    do while (hi.next([t x]) = 0);
      put t= x=;
      t = timestamp '1999-12-01 12:00:00.00'; x = 999;
    end;
    put '**************************************';
    t = timestamp '1929-09-25 13:51:36.00'; x = 1; h.remove();
    t = timestamp '1929-09-25 13:52:36.00'; x = 1; h.remove();
    t = timestamp '1999-12-01 12:00:00.00'; x = 999;
    do while (hi.next([t x]) = 0);
      put t= x=;
      t = timestamp '1999-12-01 12:00:00.00'; x = 999;
    end;
  end;
enddata;
```

```
run;
```

次の行が SAS ログに書き込まれます。

```
t=1927-08-24 12:51:36 x=1
t=1928-08-24 12:51:36 x=1
t=1929-08-24 12:51:36 x=1
t=1929-09-24 12:51:36 x=1
t=1929-09-25 12:51:36 x=1
t=1929-09-25 13:51:36 x=1
t=1929-09-25 13:52:36 x=1
t=1929-09-25 13:52:37 x=1
t=1929-09-25 13:52:37.010000000 x=1
t=1930-09-25 13:52:37.010000000 x=1
t=1930-10-25 13:52:37.010000000 x=1
**************************************
t=1927-08-24 12:51:36 x=1
t=1928-08-24 12:51:36 x=1
t=1929-08-24 12:51:36 x=1
t=1929-09-24 12:51:36 x=1
t=1929-09-25 12:51:36 x=1
t=1929-09-25 13:52:37 x=1
t=1929-09-25 13:52:37.010000000 x=1
t=1930-09-25 13:52:37.010000000 x=1
t=1930-10-25 13:52:37.010000000 x=1
```

関連項目

- “暗黙の変数と変数リストのメソッド” (*SAS DS2 プログラマガイド*)
- “データの置換と削除” (*SAS DS2 プログラマガイド*)

メソッド:

- “ADD メソッド: ハッシュパッケージ” (1646 ページ)
- “DEFINEKEY メソッド” (1670 ページ)
- “KEYS メソッド” (1689 ページ)

REMOVEALL メソッド

ハッシュパッケージからすべてのキーに 関連付けられているデータを削除します。

適用対象: ハッシュパッケージ

構文

形式 1: *package*.**REMOVEALL**();

形式 2: *package*.**REMOVEALL**(\[*keys*\]);

必須引数

package
ハッシュパッケージ変数のインスタンスを指定します。

[*keys*]
変数リストを使用してキー値を指定します。

参照項目: “DS2 変数リスト” (*SAS DS2 プログラマガイド*)

詳細

REMOVEALL メソッドは、ハッシュパッケージからキーとデータの両方を削除します。

REMOVEALL メソッドにキー変数を渡すには、2 つの方法があります。

- 暗黙の変数メソッド(形式 1)

  キー変数は REMOVEALL メソッドの呼び出しで暗示され、指定する必要はありません。

- 変数リストメソッド(形式 2)

  指定されたキー変数は、REMOVEALL メソッドに明示的に渡されます。

注: REMOVEALL メソッドは、データ変数の値を変更しません。ハッシュパッケージの値のみを削除します。

関連項目

- “暗黙の変数と変数リストのメソッド” (*SAS DS2 プログラマガイド*)
- “データの置換と削除” (*SAS DS2 プログラマガイド*)

メソッド:

- “MULTIDATA メソッド” (1692 ページ)
- “REMOVE メソッド” (1710 ページ)
- “REMOVEDUP メソッド” (1714 ページ)

演算子:

- “_NEW_ 演算子: ハッシュパッケージ” (1694 ページ)

ステートメント:

- “DECLARE PACKAGE ステートメント: ハッシュパッケージ” (1656 ページ)

REMOVEDUP メソッド

現在のキーの現在のデータアイテムに関連付けられているデータをハッシュパッケージから削除します。

適用対象: ハッシュパッケージ

構文

package.**REMOVEDUP**( );

必須引数

package
ハッシュパッケージ変数のインスタンスを指定します。

詳細

REMOVEDUP メソッドは、複数のデータアイテムを持つキーに対して、ハッシュパッケージから現在のデータアイテムを削除します。

注: REMOVEDUP メソッドは、データ変数の値を変更しません。ハッシュパッケージの値のみを削除します。

注: キーのデータアイテムリストにデータアイテムが 1 つしかない場合、キーとデータがハッシュパッケージから削除されます。

比較

REMOVEDUP メソッドは、現在のキーの現在のデータアイテムに関連付けられているデータをハッシュパッケージから削除します。REMOVE メソッドは、指定されたキーに関連付けられているデータをハッシュパッケージから削除します。

例

この例では、複数のキーがデータアイテムを 複数持つハッシュパッケージを作成します。キーの重複したデータアイテムが削除されます。

```
data testdup;
  length key data 8;
  input key data;
  datalines;
  1 10
  2 11
  1 15
  3 20
  2 16
  2 9
  3 100
  5 5
  1 5
  4 6
  5 99
;

proc ds2;
data _null_;
  dcl double key "data" k d;
  method init();
    dcl package hash  h([key], [key "data"], 8, 'testdup', 'yes', '',
      '', 'yes');
    dcl package hiter i(h);
    dcl int rc;

    do k = 1 to 5;
      do while (h.find([k], [k d]) = 0 and h.has_next() = 0);
        h.find_next([k d]);
        h.removedup ();
      end;
    end;

    rc = i.first([k d]);
    do while (rc = 0);
      put k= d=;
      rc = i.next([k d]);
    end;
  end;
enddata;
run;
quit;
```

次の行が SAS ログに書き込まれます。

```
key=1 data=10
key=2 data=11
key=3 data=20
key=4 data=6
key=5 data=5
```

関連項目

- *“データの置換と削除” (SAS DS2 プログラマガイド)*

メソッド:

- “REMOVE メソッド” (1710 ページ)
- “REMOVEALL メソッド” (1712 ページ)

REPLACE メソッド

指定されたキーに関連付けられているデータを新しいデータで置き換えます。

適用対象: ハッシュパッケージ

注: かっこの前のエスケープ文字(\)は、構文にかっこが必要であることを示します。

構文

形式 1: *package*.**REPLACE**( );

形式 2: *package*.**REPLACE**(\[*keys*\], \[*data*\]);

形式 3: *package*.**REPLACE**(\[*keys*\]);

必須引数

package
ハッシュパッケージ変数のインスタンスを指定します。

[*keys*]
変数リストを使用してキー値を指定します。

制限事項 キーのみを指定した場合、REPLACE メソッドはキーのみのハッシュパッケージに対してのみ機能します。

参照項目: “DS2 変数リスト” (*SAS DS2 プログラマガイド*)

[*data*]
データが置き換えられる変数を指定します。

参照項目: “DS2 変数リスト” (*SAS DS2 プログラマガイド*)

詳細

REPLACE メソッドは、キー変数の値を使用して、ハッシュテーブルのキーとデータのペアを検索します。ペアが見つかった場合、データはデータ変数(形式 1 および 2)の現在の値で置き換えられます。

キーのみを持つハッシュパッケージ(形式 3)に対しては、REPLACE メソッドはキーの要約統計量のみ更新します。

例

```
data _null_;
  dcl double x rc;
  dcl timestamp t;
  dcl package hash h(0, '', 'yes');
  dcl package hiter hi('h');
  method init();
    rc = h.defineKey('t');
    rc = h.defineData('t');
    rc = h.defineData('x');
    rc = h.defineDone();
    t = timestamp '1927-08-24 12:51:36.00'; x = 1; h.add();
    t = timestamp '1928-08-24 12:51:36.00'; x = 1; h.add();
    t = timestamp '1929-08-24 12:51:36.00'; x = 1; h.add();
    t = timestamp '1929-09-24 12:51:36.00'; x = 1; h.add();
    t = timestamp '1929-09-25 12:51:36.00'; x = 1; h.add();
    t = timestamp '1929-09-25 13:51:36.00'; x = 1; h.add();
    t = timestamp '1929-09-25 13:52:36.00'; x = 1; h.add();
    t = timestamp '1929-09-25 13:52:37.00'; x = 1; h.add();
    t = timestamp '1929-09-25 13:52:37.01'; x = 1; h.add();
    t = timestamp '1930-09-25 13:52:37.01'; x = 1; h.add();
    t = timestamp '1930-10-25 13:52:37.01'; x = 1; h.add();
    t = timestamp '1999-12-01 12:00:00.00'; x = 999;
    do while (hi.next() = 0);
      put t= x=;
      t = timestamp '1999-12-01 12:00:00.00'; x = 999;
    end;
    put '****************************************';
    t = timestamp '1929-09-25 13:51:36.00'; x = 64; h.replace();
    t = timestamp '1929-09-25 13:52:36.00'; x = 64; h.replace();
    t = timestamp '1999-12-01 12:00:00.00'; x = 999;
    do while (hi.next() = 0);
      put t= x=;
      t = timestamp '1999-12-01 12:00:00.00'; x = 999;
    end;
  end;
enddata;
```

次の行が SAS ログに書き込まれます。

```
t=1927-08-24 12:51:36 x=1
t=1928-08-24 12:51:36 x=1
t=1929-08-24 12:51:36 x=1
t=1929-09-24 12:51:36 x=1
t=1929-09-25 12:51:36 x=1
t=1929-09-25 13:51:36 x=1
t=1929-09-25 13:52:36 x=1
t=1929-09-25 13:52:37 x=1
t=1929-09-25 13:52:37.010000000 x=1
t=1930-09-25 13:52:37.010000000 x=1
t=1930-10-25 13:52:37.010000000 x=1
****************************************
t=1927-08-24 12:51:36 x=1
t=1928-08-24 12:51:36 x=1
t=1929-08-24 12:51:36 x=1
t=1929-09-24 12:51:36 x=1
t=1929-09-25 12:51:36 x=1
t=1929-09-25 13:51:36 x=64
t=1929-09-25 13:52:36 x=64
t=1929-09-25 13:52:37 x=1
t=1929-09-25 13:52:37.010000000 x=1
t=1930-09-25 13:52:37.010000000 x=1
t=1930-10-25 13:52:37.010000000 x=1
```

関連項目

- “暗黙の変数と変数リストのメソッド” (*SAS DS2 プログラマガイド*)
- “データの置換と削除” (*SAS DS2 プログラマガイド*)

メソッド:

- “REPLACEDUP メソッド” (1718 ページ)

REPLACEDUP メソッド

現在のキーの現在のデータアイテムに関連付けられているデータを新しいデータで置き換えます。

適用対象: ハッシュパッケージ

構文

package.**REPLACEDUP**( );

必須引数

package
ハッシュパッケージ変数のインスタンスを指定します。

詳細

REPLACEDUP メソッドは、複数のデータアイテムを持つキーに対して、ハッシュパッケージからの現在のデータアイテムを 置き換えます。

注: REPLACEDUP メソッドは、データ変数の値をデータアイテムの値で置き換えません。ハッシュパッケージの値のみを置き換えます。

注: REPLACEDUP メソッドを呼び出してもキーが見つからない場合は、キーとデータがハッシュパッケージに追加されます。

比較

REPLACEDUP メソッドは、現在のキーの現在のデータアイテムに関連付けられているデータを新しいデータで置き換えます。REPLACE メソッドは、指定されたキーに関連付けられているデータを新しいデータで置き換えます。

例

この例では、複数のキーがデータアイテムを 複数持つハッシュパッケージを作成します。重複するデータアイテムが見つかった場合、データアイテムの値に 300 が追加されます。

```
data testdup;
  length key data 8;
  input key data;
  datalines;
  1 10
  2 11
  1 15
  3 20
  2 16
  2 9
  3 100
  5 5
  1 5
  4 6
  5 99
;
proc ds2;
data _null_;
  dcl double key "data" k d;
  method init();
    dcl package hash h([key], [key "data"], 8, 'testdup','yes', '', '', 'yes');
    dcl package hiter i(h);
    dcl int rc;
```

```
    do k = 1 to 5;
      do while (h.find([k], [k d]) = 0);
        put k= d=;
            do while (h.has_next() = 0);
          h.find_next([k d]);
          put 'dup ' k= d=;
          d = d + 300;
          h.replacedup ();
              h.has_next();
        end;
      end;
    end;

    put 'iterating...';
    rc = i.first([k d]);
    do while (rc = 0);
      put k= d=;
      rc = i.next([k d]);
    end;
  end;
enddata;
run;
quit;
```

次の行が SAS ログに書き込まれます。

```
key=1 data=10
dup key=1 15
dup key=1 5
key=2 data=11
dup key=2 16
dup key=2 9
key=3 data=20
dup key=3 100
key=4 data=6
key=5 data=5
dup key=5 99
iterating...
key=1 data=10
key=1 data=315
key=1 data=305
key=2 data=11
key=2 data=316
key=2 data=309
key=3 data=20
key=3 data=400
key=4 data=6
key=5 data=5
key=5 data=399
```

関連項目

- “データの置換と削除” (*SAS DS2 プログラマガイド*)

メソッド:

- “REPLACE メソッド” (1716 ページ)

SETCUR メソッド

反復の開始キーアイテムを 指定します。

適用対象: ハッシュ反復子パッケージ

注: かっこの前のエスケープ文字(\)は、構文にかっこが必要であることを示します。

構文

形式 1: *package*.**SETCUR**( );

形式 2: *package*.**SETCUR**(\[*keys*\], \[*data*\]);

必須引数

package
ハッシュ反復子パッケージ変数の名前を指定します。

[*keys*]
変数リストを使用してキー変数を指定します。

参照項目: “DS2 変数リスト” (*SAS DS2 プログラマガイド*)

[*data*]
データアイテムを 保存する変数を指定します。

参照項目: “DS2 変数リスト” (*SAS DS2 プログラマガイド*)

詳細

ハッシュ反復子を使用すると、ハッシュパッケージの任意のアイテムから反復を開始できます。SETCUR メソッドは、反復の開始キーを設定します。指定されたキー変数を使用して、開始アイテムを参照します。アイテムが存在する場合、そのアイテムに関連付けられているデータはデータ変数に保存されます。

FIRST メソッドまたは LAST メソッドを使用して、それぞれ、最初または最終のアイテムから反復を開始できます。

キー変数とデータ変数を SETCUR メソッドに渡す方法には、2 つの方法があります。

- 暗黙の変数メソッド(形式 1)

  キー変数とデータ変数は SETCUR メソッドの呼び出しで暗黙的に指定されるため、指定する必要はありません。

- 変数リストメソッド(形式 2)

  指定されたキー変数とデータ変数は、SETCUR メソッドに明示的に渡されます。

例

次の例では、SETCUR メソッドを使用して、最初または最後のアイテムではなく RA =1831.6 から反復を開始します。

```
declare hiter iter('myhash');
myhash.defineKey('ra');
myhash.defineData('obj', 'ra');
myhash.defineDone();
ra='18 31.6';
rc = iter.setcur();
  do while (rc = 0);
    put obj= ra=;
    rc = iter.next();
  end;
```

関連項目

- “DS2 変数リスト” (*SAS DS2 プログラマガイド*)

メソッド:

- “FIRST メソッド” (1680 ページ)
- “LAST メソッド” (1691 ページ)

SUM メソッド

ハッシュテーブルから指定されたキーの要約値を取得し、その値を変数に保存します。

適用対象: ハッシュパッケージ

注: かっこの前のエスケープ文字(\)は、構文にかっこが必要であることを示します。

構文

形式 1: *summary-variable*=*package*.**SUM**();

形式 2: *summary-variable*= *package*.**SUM**(\[*keys*\]);

必須引数

summary-variable
現在のキーに対する現在の要約値を保持する変数を指定します。

注 成功か失敗かを示すリターンコードは、メソッドによって返されません。

package
ハッシュパッケージ変数のインスタンスを指定します。

[*keys*]
変数リストを使用してキー値を指定します。

参照項目: "DS2 変数リスト" (*SAS DS2 プログラマガイド*)

詳細

SUM メソッドを使用して、ハッシュパッケージからキーの要約を取得します。SUM メソッドは、キーに 1 つのデータアイテムのみ存在する場合、指定されたキーに対する要約値を取得します。詳細については、"キー要約の維持" (*SAS DS2 プログラマガイド*)を参照してください。

SUM メソッドにキー変数を渡すには、2 つの方法があります。

- 暗黙の変数メソッド(形式 1)

  キー変数は SUM メソッドの呼び出しで暗示され、指定する必要はありません。
- 変数リストメソッド(形式 2)

  指定されたキー変数は、SUM メソッドに明示的に渡されます。

比較

SUMDUP メソッドは、キーに複数のデータアイテムが存在する場合、現在のキーの現在のデーアイテムに 対する要約値を取得します。

例: 指定されたキーに対するキー要約の取得

次の例では、SUM メソッドを使用して、指定されたキー 99 に対するキー要約を取得します。

```
data _null_;
  declare double k count total;
  declare package hash myhash(0, '', 'a', '', 'count');
  method init();
    myhash.defineKey('k');
    myhash.defineDone();

    k = 99;
    count = 1;
    myhash.add();

    /* COUNT is given the value 2.5 and the */
    /* FIND sets the summary to 3.5*/
    count = 2.5;
    myhash.find();
```

```
    /* The COUNT of 3 is added to the FIND and */
    /* sets the summary to 6.5. */
    count = 3;
    myhash.find();


    /* The COUNT of -1 sets the summary to 5.5. */
    count = -1;
    myhash.find();

    /* The SUM method gives the current value of */
    /* the key summary to the variable TOTAL. */
    total = myhash.sum();

    /* The PUT statement prints total=5.5 in the log. */
    put total=;

  end;
enddata;
run;
```

関連項目

- “暗黙の変数と変数リストのメソッド” (*SAS DS2 プログラマガイド*)

メソッド:

- “SUMDUP メソッド” (1724 ページ)

SUMDUP メソッド

現在のキーの現在のデータアイテムに対する要約値を取得し、その値を変数に保存します。

適用対象: ハッシュパッケージ

注: かっこの前のエスケープ文字(\)は、構文にかっこが必要であることを示します。

構文

形式 1: *summary-variable*=*package*.**SUMDUP**();

形式 2: *summary-variable*= *package*.**SUMDUP**(\[*keys*\]);

必須引数

summary-variable
現在のキーの現在のデータアイテムに対する現在の要約値を保持する変数を指定します。

注 成功か失敗かを示すリターンコードは、メソッドによって返されません。

package
ハッシュパッケージ変数のインスタンスを指定します。

[*keys*]
変数リストを使用してキー値を指定します。

参照項目: "DS2 変数リスト" (*SAS DS2 プログラマガイド*)

詳細

キーに複数のデータアイテムがある場合は、SUMDUP メソッドを使用してハッシュパッケージからキーの要約を取得します。詳細については、"キー要約の維持" (*SAS DS2 プログラマガイド*)を参照してください。

SUMDUP メソッドにキー変数を渡すには、2 つの方法があります。

- 暗黙の変数メソッド(形式 1)

  キー変数は SUMDUP メソッドの呼び出しで暗示され、指定する必要はありません。
- 変数リストメソッド(形式 2)

  指定されたキー変数は、SUMDUP メソッドに明示的に渡されます。

比較

SUMDUP メソッドは、キーに複数のデータアイテムが存在する場合、現在のキーの現在のデーアイテムに 対する要約値を取得します。SUM メソッドは、キーに 1 つのデータアイテムのみ存在する場合、指定されたキーに対する要約値を取得します。

例: 要約値の取得

次の例では、SUMDUP メソッドを使用して、現在のデータアイテムに対する要約値を取得します。

```
data hashinp;
  dcl double k v;
  method init();
    k=1; v=2; output;
    k=1; v=4; output;
    k=1; v=8; output;
```

```
    k=2; v=2; output;
    k=3; v=4; output;
    k=2; v=8; output;
   end;
enddata;
run;

data results(keep=(k v sumres));
  dcl double k v si sumres;
  dcl package hash h(8, 'hashinp', 'ascending', '', 'si', 'multidata');

  method testsuminc(double kval, double suminc);
    dcl double rc;
    si = suminc;
    put 'input:' kval= suminc=;
    k=kval; rc=h.find();
    if (rc=0) then do;
      rc=h.has_next();
      do while(rc=0);
        put 'next:' k= v=;
        h.find_next();
        rc=h.has_next();
      end;
      rc=h.has_prev();
       do while(rc=0);
         put 'prev:' k= v=;
         h.find_prev();
         rc=h.has_prev();
       end;
       si=0;
      k=kval; rc=h.find();
       do while(rc=0);
         sumres = h.sumdup();
        output;
         rc=h.find_next();
      end;
    end;
  end;

  method init();
    h.defineKey('k');
    h.defineData('k');
    h.defineData('v');
    h.defineDone();
      testsuminc(1.0, 2.0);
      testsuminc(2.0, 20.0);
      testsuminc(3.0, 4.0);
  end;
  method term();
    dcl package hiter hi('h');
    rc=hi.first();
    do while (rc=0);
      put 'final:' k= v=;
      rc=hi.next();
    end;
   end;
```

```
enddata;
run;

data _null_;
  method run();
    set results;
    put k= v= sumres=;
  end;
enddata;
run;
```

次の行が SAS ログに書き込まれます。

```
k=1 v=2 sumres=4
k=1 v=4 sumres=4
k=1 v=8 sumres=2
k=2 v=2 sumres=40
k=2 v=8 sumres=20
k=3 v=4 sumres=4
```

関連項目

- “暗黙の変数と変数リストのメソッド” (*SAS DS2 プログラマガイド*)

メソッド:

- “SUM メソッド” (1722 ページ)

SUMINC メソッド

ハッシュパッケージキーの要約カウントを維持する変数を指定します。

適用対象: ハッシュパッケージ

注: ハッシュインスタンスに渡されるすべての変数は、グローバル変数である必要があります。

構文

package.**SUMINC**(*suminc-variable*);

必須引数

package
ハッシュパッケージ変数のインスタンスを指定します。

suminc-variable
ハッシュパッケージキーの要約カウントを維持する変数を指定します。

詳細

SUMINC パラメーターまたはメソッドを使用して、ハッシュパッケージキーの要約カウントを維持できます。SUMINC は、レコードが FIND、CHECK、または REF メソッドによって使用されるたびに、レコードに要約値を保存できるように、各レコードに内部ストレージを割り当てるようにハッシュパッケージに指示します。SUMINC 値は、FIND、CHECK、または REF メソッドの後にハッシュパラメーターキーの要約カウントを維持するためにも使用されます。SUMINC には、合計増分、つまり、キーへの各参照のキー要約に追加する量を保持する変数が指定されます。SUMINC 値は、0 より大きい、小さい、または等しい場合があります。

SUMINC 値は、ADD メソッドの要約を初期化するためにも使用されます。ADD メソッドが発生するたびに、SUMINC 値のキーが初期化されます。

SUMINC 変数は、SUM 関数のように、欠損値または null 値をゼロとして扱います。たとえば、キー要約は、変数の現在の値を使用して変更されます。

詳細については、“キー要約の維持” (*SAS DS2 プログラマガイド*)を参照してください。

注: または、DECLARE PACKAGE ステートメントの *suminc* パラメーターまたは _NEW_演算子を使用して、現在のキーの現在のデータアイテムの要約値を取得することもできます。

例

例については、“SUMDUP メソッド” (1724 ページ)を参照してください。

関連項目

- “ハッシュパッケージの初期化データの提供” (*SAS DS2 プログラマガイド*)

メソッド:

- “CHECK メソッド” (1649 ページ)
- “FIND メソッド” (1674 ページ)
- “REF メソッド” (1708 ページ)

演算子:

- “_NEW_ 演算子: ハッシュパッケージ” (1694 ページ)

ステートメント:

- “DECLARE PACKAGE ステートメント: ハッシュパッケージ” (1656 ページ)

17

DS2 HTTP パッケージのメソッド、演算子、およびステートメント

メソッド命名規則 **1730**

SAS LOCKDOWN サポート **1730**

ディクショナリ **1730**

ABORT メソッド 1730
ADDREQUESTHEADER メソッド 1731
ADDSASOAUTHTOKEN メソッド 1732
CREATEGETMETHOD メソッド 1733
CREATEHEADMETHOD メソッド 1738
CREATEPOSTMETHOD メソッド 1739
DECLARE PACKAGE ステートメント: HTTP パッケージ 1741
DELETE メソッド: HTTP パッケージ 1742
EXECUTEMETHOD メソッド 1743
EXECUTEMETHODSTREAM メソッド 1744
GETRESPONSEBODYASBINARY メソッド 1745
GETRESPONSEBODYASSTRING メソッド 1746
GETRESPONSECONTENTTYPE メソッド 1748
GETRESPONSEHEADERSASSTRING メソッド 1750
GETSTATUSCODE メソッド 1751
NEW 演算子: HTTP パッケージ 1753
SETOAUTHTOKEN メソッド 1754
SETPASSWORD メソッド 1755
SETPROXYPASSWORD メソッド 1756
SETPROXYURL メソッド 1757
SETPROXYUSERNAME メソッド 1758
SETREQUESTBODYASBINARY メソッド 1759
SETREQUESTBODYASSTRING メソッド 1760
SETREQUESTBODYCHARACTERSET メソッド 1761
SETREQUESTCONTENTTYPE メソッド 1762
SETRESPONSEBODYCHARACTERSET メソッド 1763
SETSOCKETTIMEOUT メソッド 1764

SETURL メソッド 1765
SETUSERNAME メソッド 1766
STREAMRESPONSEBODYASBINARY メソッド 1767
STREAMRESPONSEBODYASSTRING メソッド 1768

メソッド命名規則

同様のメソッドのセットについて説明する場合、このドキュメントでは、短い名前とアスタリスク(*)を使用して、メソッドのセット全体を示します。

たとえば、説明に SETREQUESTBODYASBINARY メソッドと SETREQUESTBODYASSTRING メソッドの両方が含まれる場合、テキストでは "SETREQUESTBODY*メソッド"と表記しています。

SAS LOCKDOWN サポート

DS2 HTTP パッケージは、SAS LOCKDOWN 制限をサポートします。SAS HTTP (または URL)メソッドがロックダウン状態にある場合、パッケージは使用できません。

ディクショナリ

ABORT メソッド

HTTP メソッドの実行を停止します。

構文

package.**ABORT**( );

必須引数

package
HTTP パッケージ変数のインスタンスを指定します。

詳細

ABORT メソッドを呼び出して、現在実行中の HTTP メソッドを停止します。ABORT メソッドは、データをストリーミングしていて、応答本文全体を表示する必要がないと判断した場合に役立ちます。

関連項目

- “HTTP パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “EXECUTEMETHOD メソッド” (1743 ページ)
- “EXECUTEMETHODSTREAM メソッド” (1744 ページ)

ADDREQUESTHEADER メソッド

HTTP メソッド要求にヘッダーを追加します。

構文

package.**ADDREQUESTHEADER**(*'name'*, *'value'*);

必須引数

package
HTTP パッケージ変数のインスタンスを指定します。

'name'
ヘッダーフィールドの名前を指定します。

'value'
ヘッダーフィールドの値を指定します。

詳細

ヘッダーは、ヘッダーのリストの最後に追加されます。

例

```
h.addRequestHeader('Cookie', ' SSID=3lk3q84095jk79lgjf');
```

関連項目

- “HTTP パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “CREATEGETMETHOD メソッド” (1733 ページ)
- “CREATEHEADMETHOD メソッド” (1738 ページ)
- “CREATEPOSTMETHOD メソッド” (1739 ページ)

ADDSASOAUTHTOKEN メソッド

OAuth アクセストークンを SAS 環境から検索します。トークンが見つかった場合、それが要求の OAuth トークンとして設定されます。

構文

package.**ADDSASOAUTHTOKEN**( );

必須引数

package
HTTP パッケージ変数のインスタンスを指定します。

詳細

Open Authorization (OAuth)は、インターネット上でのトークンベースの認証と承認のためのオープンスタンダードです。OAuth は HTTP で動作するように設計されており、リソース所有者の承認を得て、権限サーバーによってサードパーティクライアントにアクセストークンを発行できるようにします。そして、サードパーティクライアントはアクセストークンを使用して、リソースサーバーによってホストされている保護されたリソースにアクセスします。

HTTP パッケージの場合、ADDSASOAUTHTOKEN メソッドを使用して、SAS が OAuth アクセストークンを環境から検索できるようにします。このトークンは、Authorization ヘッダーフィールドの Bearer 値として要求ヘッダーに追加されます。

OAuth の詳細については、OAuth 2.0 を参照してください。

注: リターンコード値 0 は成功を示し、値 1 は失敗を示し、値 3 はトークンが見つからないことを示し、値 4 はトークンの有効期限が切れていることを示します。

比較

ADDSASOAUTHTOKEN メソッドを使用して、SAS が OAuth アクセストークンを環境から検索できるようにします。OAuth アクセストークンは SETOAUTHTOKEN メソッドで指定します。

関連項目

メソッド:

- “SETOAUTHTOKEN メソッド” (1754 ページ)

CREATEGETMETHOD メソッド

Web サーバーからリソースを取得するための HTTP GET メソッドを作成します。

構文

package.**CREATEGETMETHOD** {( ) | (*'url'*)};

必須引数

package
HTTP パッケージ変数のインスタンスを指定します。

( ) | (*'url'*)
URL なしで HTTP GET メソッドを生成するか、リソースの URL を指定します。

ヒント URL には、クエリ文字列(名前/値ペア)を含めることができます。

詳細

CREATEGETMETHOD メソッドを使用して、HTTP GET メソッドを作成します。GET メソッドを作成したら、EXECUTEMETHOD*メソッドの 1 つを呼び出して、指定されたリソースを Web サーバーに要求します。

URL は、CREATEGETMETHOD メソッドの引数としてか、または SETURL メソッドを使用して設定する必要があります。

例

例 1: GET メソッドを作成して HTTP リソースを取得する

次の例では、HTTP リソースを要求し、Web サーバーからの応答のヘッダーと本文を表示します。

```
data _null_;
 method init();
   declare package http h();
   declare varchar(1024) character set utf8 body headers;
   declare int rc status;

   /* Build and send a GET method for the 'FR' resource */
   h.createGetMethod('http://api.worldbank.org/countries/FR');
   h.executeMethod();

   /* If the resource was returned by the server, show the response */
   status = h.getStatusCode();   /* get the HTTP status code */
   put 'Country code: FR, executeMethod() status:' status;
   if status eq 200 then do;     /* 200 = OK */
    /* retrieve the headers from the response that came from the server */
    h.getResponseHeadersAsString(headers, rc);
    put 'Headers:';
    put headers;
    /* retrieve the body from the response that came from the server */
    h.getResponseBodyAsString(body, rc);
    put 'Body:';
    put body;
   end;
 end;
enddata; run;
```

次の行がログに書き込まれます。

```
Country code: FR, executeMethod() status: 200
Headers:
HTTP/1.1 200 OK
Content-Length: 627
Content-Type: text/xml; charset=UTF-8
Server: WorldBank
Web Server2
Date: Sat, 22 Mar 2014 00:43:30 GMT
X-Cache: MISS from transproxy
Via: 1.1
transproxy (squid)
Connection: keep-alive

Body:
<?xml version="1.0" encoding="utf-8"?>
<wb:countries page="1" pages="1" per_page="50" total="1"
xmlns:wb="http://www.worldbank.org">
  <wb:country id="FRA">
<wb:iso2Code>FR</wb:iso2Code>
    <wb:name>France</wb:name>
    <wb:region id="ECS">Europe
&amp; Central Asia (all income levels)</wb:region>
    <wb:adminregion id="" />
<wb:incomeLevel id="OEC">High income: OECD</wb:incomeLevel>
    <wb:lendingType id="LNX">Not
classified</wb:lendingType>
    <wb:capitalCity>Paris</wb:capitalCity>
<wb:longitude>2.35097</wb:longitude>
    <wb:latitude>48.8566</wb:latitude>
</wb:country>
</wb:countries>
```

例 2: リソース URL の式を使用して GET メソッドを作成する

次の例では、入力データセット内の国コードごとに個別の GET 要求を生成します。各コードは、リソースの URL を形成する式に連結されます。最後のコード ZZ は、有効な国コードではないことに注意してください。

```
proc ds2;
data country_codes /overwrite=yes;
dcl char(2) code;
method init();
  code='ES'; output;
  code='FR'; output;
  code='GB'; output;
  code='ZZ'; output; /* unknown country code */
end;
enddata; run; quit;

proc ds2;
data _null_;
  method run();
    declare package http h();
    declare varchar(1024) character set utf8 body;
    declare int rc status;

    /* Build and send a GET method for each country code */
    set country_codes;
    h.createGetMethod('http://api.worldbank.org/countries/'
      || code || '/indicators/NY.GNP.PCAP.CD/?date=1990:1990');
    h.executeMethod();
```

```
   /* If the resource was returned by the server, show the response */
   status = h.getStatusCode();     /* get the HTTP status code */
   put 'Country code:' code 'executeMethod() status code:' status;
   if status eq 200 then do;       /* 200 = OK */
    /* retrieve the body from the response that came from the server */
    h.getResponseBodyAsString(body, rc);
    put 'Body:';
    put body;
   end;
   put;
  end;
enddata; run; quit;
```

次の行がログに書き込まれます。

```
Country code: ES executeMethod() status code: 200
Body:
<?xml version="1.0" encoding="utf-8"?>
<wb:data page="1" pages="1" per_page="50" total="1"
xmlns:wb="http://www.worldbank.org">
 <wb:data>
   <wb:indicator id="NY.GNP.PCAP.CD">GNI per
capita, Atlas method (current US$)</wb:indicator>
   <wb:country id="ES">Spain</wb:country>
 <wb:date>1990</wb:date>
   <wb:value>11880</wb:value>
   <wb:decimal>0</wb:decimal>
</wb:data>
</wb:data>

Country code= FR executeMethod() status code = 200
Body:
<?xml version="1.0" encoding="utf-8"?>
<wb:data page="1" pages="1" per_page="50" total="1"
xmlns:wb="http://www.worldbank.org">
 <wb:data>
   <wb:indicator id="NY.GNP.PCAP.CD">GNI per
capita, Atlas method (current US$)</wb:indicator>
   <wb:country id="FR">France</wb:country>
  <wb:date>1990</wb:date>
   <wb:value>20050</wb:value>
   <wb:decimal>0</wb:decimal>
</wb:data>
</wb:data>

Country code= GB executeMethod() status code = 200
Body:
<?xml version="1.0" encoding="utf-8"?>
<wb:data page="1" pages="1" per_page="50" total="1"
xmlns:wb="http://www.worldbank.org">
 <wb:data>
   <wb:indicator id="NY.GNP.PCAP.CD">GNI per
capita, Atlas method (current US$)</wb:indicator>
   <wb:country id="GB">United
Kingdom</wb:country>
   <wb:date>1990</wb:date>
   <wb:value>16620</wb:value>
<wb:decimal>0</wb:decimal>
 </wb:data>
</wb:data>

Country code= ZZ executeMethod() status code = 200
Body:
<?xml version="1.0" encoding="utf-8"?>
<wb:error xmlns:wb="http://www.worldbank.org">
<wb:message id="120" key="Parameter 'country' has an
invalid value">The provided parameter value
is not valid</wb:message>
</wb:error>
```

関連項目

- “HTTP パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “EXECUTEMETHOD メソッド” (1743 ページ)

- “EXECUTEMETHODSTREAM メソッド” (1744 ページ)
- “SETURL メソッド” (1765 ページ)

CREATEHEADMETHOD メソッド

Web リソースが存在するかどうかをテストし、そのヘッダーを取得するための HTTP HEAD メソッドを作成します。

構文

package.**CREATEHEADMETHOD**{( ) | ('*url*')};

必須引数

package
HTTP パッケージ変数のインスタンスを指定します。

( ) | ('*url*')
URL なしで HTTP HEAD メソッドを生成するか、リソースの URL を指定します。

ヒント URL には、クエリ文字列(名前/値ペア)を含めることができます。

詳細

CREATEHEADMETHOD メソッドを使用して、HTTP HEAD メソッドを作成します。HEAD メソッドを作成したら、EXECUTEMETHOD メソッドを呼び出して、要求を Web サーバーに送信します。

URL は、CREATEHEADMETHOD メソッドの引数としてか、または SETURL メソッドを使用して設定する必要があります。

例

次の例では、HEAD メソッドを作成し、要求を Web サーバーに送信して、応答のヘッダーを表示します。

```
data _null_;
 method init();
   declare package http h();
   declare varchar(1024) character set utf8 headers;
   declare int rc status;
   declare char(2) code;

   /* Build and send a HEAD method for a resource */
```

```
  code = 'GB';
  h.createHeadMethod('http://api.worldbank.org/countries/'
   || code || '/indicators/NY.GNP.PCAP.CD/?date=1990:1990');
  h.executeMethod();

  /* If the resource headers were returned by the server, show them */
  status = h.getStatusCode();   /* get the HTTP status code */
  put 'Country code:' code 'executeMethod() status:' status;
  if status eq 200 then do;     /* 200 = OK */
   /* retrieve the headers from the response that came from the server */
   h.getResponseHeadersAsString(headers, rc);
   put 'Headers:';
   put headers;
  end;
 end;
enddata; run;
```

次の行がログに書き込まれます。

```
Country code: GB executeMethod() status: 200
Headers:
HTTP/1.1 200 OK
Content-Length: 0
Content-Type: text/xml; charset=UTF-8
Server: WorldBank Web
Server1
Date: Tue, 25 Mar 2014 21:05:48 GMT
X-Cache: MISS from transproxy
Via: 1.1 transproxy
(squid)
Connection: keep-alive
```

関連項目

- “HTTP パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “EXECUTEMETHOD メソッド” (1743 ページ)
- “SETURL メソッド” (1765 ページ)

CREATEPOSTMETHOD メソッド

リソースのデータを受け入れるように Web サーバーに要求する HTTP POST メソッドを作成します。

構文

package.**CREATEPOSTMETHOD** {( ) | (*'url'*)};

必須引数

package
HTTP パッケージ変数のインスタンスを指定します。

( ) | ('*url*')
URL なしで HTTP POST メソッドを生成するか、リソースの URL を指定します。

ヒント　URL には、クエリ文字列(名前/値ペア)を含めることができます。

詳細

CREATEPOSTMETHOD メソッドを使用して、HTTP POST メソッドを作成します。

URL は、CREATEPOSTMETHOD メソッドの引数としてか、または SETURL メソッドを使用して設定する必要があります。

POST メソッドには本体が必要なため、次の手順で POST メソッドを完成させます。

- コンテンツの種類に応じて、SETREQUESTBODY*メソッドの 1 つを呼び出して、本体コンテンツを追加します。
- SETREQUESTCONTENTTYPE メソッドを呼び出して本体コンテンツの種類を示し、Content-Type:ヘッダーを設定します。

次に、EXECUTEMETHOD メソッドを呼び出して、要求を Web サーバーに送信します。

注: コンテンツの長さは、データをトランスコードした後、DS2HTTP クライアントによって計算されます。

関連項目

- "HTTP パッケージの使用" (*SAS DS2 プログラマガイド*)

メソッド:

- "ADDREQUESTHEADER メソッド" (1731 ページ)
- "EXECUTEMETHOD メソッド" (1743 ページ)
- "SETREQUESTBODYASBINARY メソッド" (1759 ページ)
- "SETREQUESTBODYASSTRING メソッド" (1760 ページ)
- "SETREQUESTCONTENTTYPE メソッド" (1762 ページ)
- "SETURL メソッド" (1765 ページ)

DECLARE PACKAGE ステートメント: HTTP パッケージ

パッケージ変数を作成し、HTTP パッケージのインスタンスを作成できるようにします。

カテゴリ: ローカル

注: DS2 HTTP パッケージには、SAS LOCKDOWN オプションの制限があります。詳細については、"LOCKDOWN" (*SAS プログラマガイド: エッセンシャル*)を参照してください。

ヒント: HTTP パッケージには PACKAGE ステートメントは必要ありません。

構文

DECLARE PACKAGE HTTP *variable*( );

必須引数

variable
HTTP パッケージのインスタンスを参照できる変数を指定します。

詳細

DS2 パッケージは、特定のインスタンスを構築して他の DS2 プログラムで使用できる変数とメソッドのコレクションです。

HTTP パッケージを使用して、HTTP Web サーバーにアクセスするための HTTP クライアントを構築します。HTTP パッケージは、DS2 プログラム用に事前定義されています。

HTTP パッケージは、DECLARE PACKAGE ステートメントを使用して宣言します。パッケージが宣言されると、パッケージのインスタンスを参照できる変数が作成されます。複数のパッケージ変数を作成でき、単一の DECLARE PACKAGE ステートメントで複数のパッケージインスタンスを構築できます。各パッケージインスタンスは、パッケージの完全に別個のコピーを表します。

HTTP パッケージのインスタンスを構築する方法は 2 つあります。

- DECLARE PACKAGE ステートメントを_NEW_演算子と一緒に使用します。

```
declare package http httpclt;
httpclt = _new_ http();
```

- DECLARE PACKAGE ステートメントをそのコンストラクター構文と一緒に使用します。

```
declare package http httpclt();
```

注: パッケージ変数は、すべての変数スコーピングルールの対象となります。詳細については、"パッケージ、スコープ、および有効期間" (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- "HTTP パッケージの使用" (*SAS DS2 プログラマガイド*)
- "パッケージのコンストラクターとデストラクター" (*SAS DS2 プログラマガイド*)

演算子:

- "_NEW_ 演算子: HTTP パッケージ" (1753 ページ)

DELETE メソッド: HTTP パッケージ

HTTP パッケージインスタンスを削除し、そのリソースを解放します。

注: DELETE メソッドは必須ではありません。HTTP パッケージインスタンスを参照する HTTP パッケージ変数がない場合、パッケージインスタンスは自動的に削除されます。

構文

package.**DELETE**( );

必須引数

package
HTTP パッケージ変数の名前を指定します。

詳細

HTTP パッケージが不要になった場合は、DELETE メソッドを使用して削除します。HTTP パッケージインスタンスを削除した後に使用しようとすると、ログにエラーが書き込まれます。

関連項目

"パッケージのコンストラクターとデストラクター" (*SAS DS2 プログラマガイド*)

EXECUTEMETHOD メソッド

HTTP メソッドを実行し、完全なエンティティとして 応答本文を取得できるようにします。

構文

package.**EXECUTEMETHOD**( );

必須引数

package
HTTP パッケージ変数のインスタンスを指定します。

詳細

EXECUTEMETHOD メソッドを呼び出して Web サーバーに要求を送信する前に、GET、HEAD、または POST メソッドを作成する必要があります。EXECUTEMETHOD メソッドは、最後に作成されたメソッドを送信します。

応答に本文がある場合は、GETRESPONSEBODYAS*メソッドの 1 つを使用して、応答本文をエンティティ 全体として取得する必要があります。

注: EXECUTEMETHOD メソッドは、応答本文のストリーミングをサポートしていません。EXECUTEMETHOD メソッドを使用して要求を送信した後、STREAMRESPONSEBODYAS*メソッドの 1 つを使用して応答本文を取得すると、ランタイムエラーが発生します。かわりに、EXECUTEMETHODSTREAM メソッドを使用してください。

関連項目

- “HTTP パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “EXECUTEMETHODSTREAM メソッド” (1744 ページ)
- “GETSTATUSCODE メソッド” (1751 ページ)
- “GETRESPONSECONTENTTYPE メソッド” (1748 ページ)
- “GETRESPONSEBODYASBINARY メソッド” (1745 ページ)
- “GETRESPONSEBODYASSTRING メソッド” (1746 ページ)

■ “SETSOCKETTIMEOUT メソッド” (1764 ページ)

EXECUTEMETHODSTREAM メソッド

HTTP メソッドを実行し、HTTP サーバーからの応答本文のストリーミングを有効にします。

構文

package.**EXECUTEMETHODSTREAM**( );

必須引数

package
HTTP パッケージ変数のインスタンスを指定します。

詳細

STREAMRESPONSEBODYASBINARY または STREAMRESPONSEBODYASSTRING メソッドのいずれかを使用して、Web サーバーから応答本文をチャンクでストリーミングする場合は、EXECUTEMETHODSTREAM メソッドを使用します。これは、サーバーからすべてのデータが到着するのを待たずに応答を処理する場合に役立ちます。

EXECUTEMETHODSTREAM メソッドを呼び出して Web サーバーに要求を送信する前に、GET メソッドを作成する必要があります。EXECUTEMETHODSTREAM メソッドは、最後に作成されたメソッドを送信します。

注: EXECUTEMETHODSTREAM メソッドは、エンティティ全体としての応答本文の取得をサポートしていません。EXECUTEMETHODSTREAM メソッドを使用して要求を送信した後、GETRESPONSEBODYAS*メソッドの 1 つを使用して応答本文を取得すると、ランタイムエラーが発生します。かわりに、EXECUTEMETHOD メソッドを使用してください。

関連項目

■ “HTTP パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

■ “EXECUTEMETHOD メソッド” (1743 ページ)
■ “GETSTATUSCODE メソッド” (1751 ページ)

- “GETRESPONSECONTENTTYPE メソッド” (1748 ページ)
- “SETSOCKETTIMEOUT メソッド” (1764 ページ)
- “STREAMRESPONSEBODYASBINARY メソッド” (1767 ページ)
- “STREAMRESPONSEBODYASSTRING メソッド” (1768 ページ)

GETRESPONSEBODYASBINARY メソッド

HTTP 応答から本文全体をバイナリ形式で返します。

制限事項: GETRESPONSEBODYASBINARY メソッドを使用して HTTP 応答から本文全体をバイナリ形式で返す場合、SETRESPONSEBODYCHARACTERSET メソッドを使用することはできません。

構文

package.**GETRESPONSEBODYASBINARY**(*variable*, *rc*);

必須引数

package
HTTP パッケージ変数のインスタンスを指定します。

variable
応答本文全体を保持するバイナリ変数を指定します。

rc
リターンコード値を保持する変数を指定します。

注: リターンコード値 0 は、成功を示します。値 1 は、失敗を示します。

詳細

GETRESPONSEBODYASBINARY メソッドを使用して、バイナリ形式で応答本文を取得します。応答本文は 1 つのエンティティとして返されます。

EXECUTEMETHOD メソッドの後に GETRESPONSEBODYASBINARY メソッドを呼び出すと、Web サーバーから返された応答本文を取得することができます。

注: GETREAPONSEBODYASBINARY メソッドは、Web クライアントが Web サーバーからすべてのデータを受信するまで戻りません。

注: 応答本文はトランスコードされません。

注: EXECUTEMETHOD メソッドは、応答本文のストリーミングをサポートしていません。

関連項目

- “HTTP パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “CREATEGETMETHOD メソッド” (1733 ページ)
- “EXECUTEMETHOD メソッド” (1743 ページ)
- “GETRESPONSEBODYASSTRING メソッド” (1746 ページ)
- “SETRESPONSEBODYCHARACTERSET メソッド” (1763 ページ)

GETRESPONSEBODYASSTRING メソッド

HTTP 応答から本文全体を文字列形式で返します。

要件 応答本文の取得後に、応答本文の文字セットを設定しないようにします。応答本文の文字セットは、GETRESPONSEBODYASSTRING メソッドで応答本文を取得する前に、SETRESPONSEBODYCHARACTERSET メソッドで指定する必要があります。

構文

package.**GETRESPONSEBODYASSTRING**(*variable*, *rc*);

必須引数

package
HTTP パッケージ変数のインスタンスを指定します。

variable
応答本文全体を保持する変数を指定します。

rc
リターンコード値を保持する変数を指定します。

注: リターンコード値 0 は、成功を示します。値 1 は、失敗を示します。

詳細

GETRESPONSEBODYASSTRING メソッドを使用して、文字列形式で応答本文を取得します。応答本文は 1 つのエンティティとして 返されます。

コンテンツの種類が設定されていない場合、HTTP クライアントは応答コンテンツの種類から文字セットを推測します。HTTP クライアントが応答コンテンツの種類から文字セットを推測できない場合、文字セットのデフォルトは UTF-8 となります。しかし、SETRESPONSEBODYCHARACTERSET メソッドを使用して、応答本文を取得するときに使用する文字セットを指定できます。

EXECUTEMETHOD メソッドの後に GETRESPONSEBODYASSTRING メソッドを呼び出すと、Web サーバーから返された応答本文を取得することができます。

注: GETREAPONSEBODYASSTRING メソッドは、Web クライアントが Web サーバーからすべてのデータを 受信するまで戻りません。

注: エンコーディングが異なる場合、応答本文は文字列のエンコーディングにトランスコードされます。

注: EXECUTEMETHOD メソッドは、応答本文のストリーミングをサポートしていません。

例

```
data _null_;
 method init();
   declare package http h();
   declare varchar(1024) character set utf8 body;
   declare int rc status;
   declare char(2) code;

   /* Build and send a GET method for a resource */
   code = 'FR';
   h.createGetMethod('http://api.worldbank.org/countries/'
     || code || '/indicators/NY.GNP.PCAP.CD/?date=1990:1991');
   h.executeMethod();

   /* If the resource was returned by the server, show the response */
   status = h.getStatusCode();   /* get the HTTP status code */
   put 'Requested resource for country code:' code 'executeMethod() status:' status;
   if status eq 200 then do;     /* 200 = OK */
     /* retrieve the body from the response that came from the server */
     h.getResponseBodyAsString(body, rc);
     put 'Body:';
     put body;
   end;
 end;
enddata; run;
```

次の行がログに書き込まれます。

```
Requested resource for country code: FR executeMethod() status: 200
Body:
<?xml version="1.0" encoding="utf-8"?>
<wb:data page="1" pages="1" per_page="50" total="2"
xmlns:wb="http://www.worldbank.org">
 <wb:data>
  <wb:indicator id="NY.GNP.PCAP.CD">GNI per
capita, Atlas method (current US$)</wb:indicator>
  <wb:country id="FR">France</wb:country>
 <wb:date>1991</wb:date>
  <wb:value>20880</wb:value>
  <wb:decimal>0</wb:decimal>
</wb:data>
 <wb:data>
  <wb:indicator id="NY.GNP.PCAP.CD">GNI per capita, Atlas method
(current US$)</wb:indicator>
  <wb:country id="FR">France</wb:country>
<wb:date>1990</wb:date>
  <wb:value>20050</wb:value>
  <wb:decimal>0</wb:decimal>
</wb:data>
</wb:data>
```

関連項目

- “HTTP パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “CREATEGETMETHOD メソッド” (1733 ページ)
- “EXECUTEMETHOD メソッド” (1743 ページ)
- “GETRESPONSEBODYASBINARY メソッド” (1745 ページ)
- “SETRESPONSEBODYCHARACTERSET メソッド” (1763 ページ)

GETRESPONSECONTENTTYPE メソッド

HTTP 応答からコンテンツの種類を返します。

構文

content-type-variable=*package*.**GETRESPONSECONTENTTYPE**( );

必須引数

content-type-variable
HTTP 応答のコンテンツの種類の値を保持する変数を指定します。

注: 可能なコンテンツの種類のリストについては、HTTP リファレンスを参照してください。

package
HTTP パッケージのインスタンスを指定します。

詳細

GETRESPONSECONTENTTYPE メソッドを使用して、最新の応答メッセージからコンテンツの種類を取得します。

例

```
data _null_;
 method init();
   declare package http h();
   declare varchar(1024) character set utf8 body contentType headers;
   declare int rc status;

   /* Build and send a HEAD method for the 'ES' resource */
   h.createHeadMethod('http://api.worldbank.org/countries/ES');
   h.executeMethod();

   /* If the resource was returned by the server, show the response */
   status = h.getStatusCode();   /* get the HTTP status code */
   put 'Country code: ES, executeMethod() status:' status;
   if status eq 200 then do;     /* 200 = OK */
    /* retrieve the content type from the response that came from the server */
    contentType = h.getResponseContentType();
    put 'Content type:' contentType;
   end;
 end;
enddata; run;
```

次の行がログに書き込まれます。

```
Country code: ES, executeMethod() status: 200
Content type: text/xml; charset=UTF-8
```

関連項目

- “HTTP パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “EXECUTEMETHOD メソッド” (1743 ページ)
- “EXECUTEMETHODSTREAM メソッド” (1744 ページ)

GETRESPONSEHEADERSASSTRING メソッド

HTTP メソッドからの応答ヘッダーを文字列形式で返します。

構文

package.**GETRESPONSEHEADERSASSTRING**(*variable*, *rc*);

必須引数

package
HTTP パッケージ変数のインスタンスを指定します。

variable
応答ヘッダーを保持する変数を指定します。

rc
リターンコード値を保持する変数を指定します。

注: リターンコード値 0 は、成功を示します。値 1 は、失敗を示します。

詳細

GETRESPONSEHEADERSASSTRING メソッドを使用して、HTTP 応答からすべてのヘッダーを取得します。EXECUTEMETHOD メソッドまたは EXECUTEMETHODSTREAM メソッドのいずれかを使用して HTTP 要求をサーバーに送信した後、GETRESPONSEHEADERSASSTRING メソッドを使用できます。

例

次の例では、HEAD メソッドを作成して送信し、GETRESPONSEHEADERSASSTRING メソッドを使用して、Web サーバーからの応答のヘッダーを表示します。

```
data _null_;
 method init();
   declare package http h();
   declare varchar(1024) character set utf8 headers;
   declare int rc status;
   declare char(2) code;

   /* Build and send a HEAD method for a resource */
   code = 'GB';
```

```
   h.createHeadMethod('http://api.worldbank.org/countries/'
    || code || '/indicators/NY.GNP.PCAP.CD/?date=1990:1990');
   h.executeMethod();

   /* If the resource headers were returned by the server, show them */
   status = h.getStatusCode();   /* get the HTTP status code */
   put 'Country code:' code 'executeMethod() status:' status;
   if status eq 200 then do;     /* 200 = OK */
    /* retrieve the headers from the response that came from the server */
    h.getResponseHeadersAsString(headers, rc);
    put 'Headers:';
    put headers;
   end;
  end;
enddata; run;
```

次の行がログに書き込まれます。

```
Country code: GB executeMethod() status: 200
Headers:
HTTP/1.1 200 OK
Content-Length: 0
Content-Type: text/xml; charset=UTF-8
Server: WorldBank Web
Server1
Date: Tue, 25 Mar 2014 21:05:48 GMT
X-Cache: MISS from transproxy
Via: 1.1 transproxy
(squid)
Connection: keep-alive
```

関連項目

- “HTTP パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “CREATEGETMETHOD メソッド” (1733 ページ)
- “CREATEHEADMETHOD メソッド” (1738 ページ)
- “CREATEPOSTMETHOD メソッド” (1739 ページ)
- “EXECUTEMETHOD メソッド” (1743 ページ)
- “EXECUTEMETHODSTREAM メソッド” (1744 ページ)

GETSTATUSCODE メソッド

最後に実行された HTTP メソッドから HTTP ステータスコードを返します。

構文

status-code-variable=*package*.**GETSTATUSCODE**( );

必須引数

status-code-variable
HTTP ステータスコード値を保持する変数を指定します。

注: 可能なステータスコードについては、HTTP リファレンスを参照してください。

package
HTTP パッケージ変数のインスタンスを指定します。

詳細

GETSTATUSCODE メソッドを使用して、最後に実行された HTTP メソッドから HTTP ステータスコードを取得します。

例

```
data _null_;
 method init();
   declare package http h();
   declare int status;

   /* Build and send a HEAD method for a resource */
   h.createHeadMethod('http://support.sas.com/documentation/');
   h.executeMethod();

   /* If the resource was returned by the server, show the response */
   status = h.getStatusCode();   /* get the HTTP status code */
   put 'HEAD method created for resource:';
   put 'http://support.sas.com/documentation/';
   put 'executeMethod() status:' status;
 end;
enddata; run;
```

次の行がログに書き込まれます。

```
HEAD method created for resource:
http://support.sas.com/documentation/
executeMethod() status: 200
```

関連項目

- “HTTP パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “EXECUTEMETHOD メソッド” (1743 ページ)
- “EXECUTEMETHODSTREAM メソッド” (1744 ページ)

NEW 演算子: HTTP パッケージ

HTTP パッケージのインスタンスを構築します。

注: かっこの前のエスケープ文字(\)は、構文にかっこが必要であることを示します。

構文

package-variable=**_NEW_ HTTP**( );

必須引数

package-variable
パッケージのインスタンスを参照できる名前を指定します。

詳細

DS2 パッケージは、特定のインスタンスを構築して他の DS2 プログラムで使用できる変数とメソッドのコレクションです。

DECLARE PACKAGE ステートメントを使用して、HTTP パッケージ変数を宣言します。HTTP パッケージ変数を宣言した後、_NEW_演算子を使用して HTTP パッケージのインスタンスをインスタンス化し、新しいパッケージインスタンスを HTTP パッケージ変数に割り当てます。

```
declare package http h;
h = _new_ http();
```

DECLARE PACKAGE ステートメントと_NEW_演算子を使用して HTTP パッケージを宣言およびインスタンス化する 2 ステップのプロセスのかわりに、コンストラクターメソッドとして DECLARE PACKAGE ステートメントを使用することにより、パッケージを 1 ステップで宣言およびインスタンス化できます。これは、DECLARE PACKAGE ステートメントのみを使用した同じ例です。

```
declare package http h();
```

注: パッケージ変数は、すべての変数スコーピングルールの対象となります。詳細については、"パッケージ、スコープ、および有効期間" (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- "HTTP パッケージの使用" (*SAS DS2 プログラマガイド*)
- "パッケージのコンストラクターとデストラクター" (*SAS DS2 プログラマガイド*)

ステートメント:

- "DECLARE PACKAGE ステートメント: HTTP パッケージ" (1741 ページ)
- "PACKAGE ステートメント" (1392 ページ)

SETOAUTHTOKEN メソッド

要求の OAuth アクセストークンを含む文字列を指定します。

構文

package.**SETOAUTHTOKEN**('*variable*');

必須引数

package
HTTP パッケージ変数のインスタンスを指定します。

'*variable*'
OAuth アクセストークンを含む文字列変数を指定します。

詳細

Open Authorization (OAuth)は、インターネット上でのトークンベースの認証と承認のためのオープンスタンダードです。OAuth は HTTP で動作するように設計されており、リソース所有者の承認を得て、権限サーバーによってサードパーティクライアントにアクセストークンを発行できるようにします。そして、サードパーティクライアントはアクセストークンを使用して、リソースサーバーによってホストされている保護されたリソースにアクセスします。

HTTP パッケージの場合、SETOAUTHTOKEN メソッドを使用して、Authorization ヘッダーフィールドの Bearer 値として要求ヘッダーに追加される OAuth アクセストークンを提供します。

OAuth の詳細については、OAuth 2.0 を参照してください。

比較

OAuth アクセストークンは SETOAUTHTOKEN メソッドで指定します。ADDSASOAUTHTOKEN メソッドを使用して、SAS が OAuth アクセストークンを環境から検索できるようにします。

関連項目

メソッド:

- “ADDSASOAUTHTOKEN メソッド” (1732 ページ)

SETPASSWORD メソッド

指定されたユーザー名に関連付けられたパスワードを表す文字列を指定します。

構文

package.**SETPASSWORD**('*variable*');

必須引数

package
HTTP パッケージ変数のインスタンスを指定します。

'*variable*'
パスワードを含む文字列変数を指定します。

詳細

SETURL メソッドで指定した URL にパスワードが必要な場合は、SETPASSWORD メソッドで指定します。同様に、URL にユーザー名が必要な場合は、SETUSERNAME メソッドで指定します。

比較

SETPASSWORD メソッドは、非プロキシユーザー名のパスワードを指定します。SETPROXYPASSWORD メソッドは、プロキシユーザー名のパスワードを指定します。

関連項目

メソッド:

- “SETPROXYPASSWORD メソッド” (1756 ページ)
- “SETURL メソッド” (1765 ページ)
- “SETUSERNAME メソッド” (1766 ページ)

SETPROXYPASSWORD メソッド

指定されたプロキシユーザー名に関連付けられたパスワードを表す文字列を指定します。

構文

package.**SETPROXYPASSWORD**('*variable*');

必須引数

package
HTTP パッケージ変数のインスタンスを指定します。

'*variable*'
パスワードを含む文字列変数を指定します。

詳細

SETPROXYURL メソッドで指定したプロキシ URL にパスワードが必要な場合は、SETPROXYPASSWORD メソッドで指定します。同様に、プロキシ URL にユーザー名が必要な場合は、SETPROXYUSERNAME メソッドで指定します。

比較

SETPROXYPASSWORD メソッドは、プロキシユーザー名のパスワードを指定します。SETPASSWORD メソッドは、非プロキシユーザー名のパスワードを指定します。

関連項目

メソッド:

- “SETPASSWORD メソッド” (1755 ページ)
- “SETPROXYURL メソッド” (1757 ページ)
- “SETPROXYUSERNAME メソッド” (1758 ページ)

SETPROXYURL メソッド

HTTP クライアントが接続するプロキシ URL を指定します。

構文

package.**SETPROXYURL**('*variable*');

必須引数

package
HTTP パッケージ変数のインスタンスを指定します。

'*variable*'
HTTP クライアントが接続するプロキシ URL を含む変数を指定します。

詳細

SETPROXYURL メソッドを使用すると、HTTP クライアントが接続するプロキシ URL を指定できます。プロキシ URL は、Web ブラウザーを介してプロキシサーバーにアクセスするための Web アドレスです。

比較

SETPROXYURL は、プロキシ URL を指定します。SETURL は、URL を指定します。

関連項目

メソッド:

- “SETPROXYPASSWORD メソッド” (1756 ページ)
- “SETPROXYUSERNAME メソッド” (1758 ページ)
- “SETURL メソッド” (1765 ページ)

SETPROXYUSERNAME メソッド

指定されたプロキシ URL に接続しようとしているユーザーのドメイン修飾ユーザー名を表す文字列を指定します。

構文

package.**SETPROXYUSERNAME**('*variable*');

必須引数

package
HTTP パッケージ変数のインスタンスを指定します。

'*variable*'
ドメイン修飾ユーザー名を含む変数を指定します。

詳細

SETPROXYURL メソッドで指定したプロキシ URL にユーザー名が必要な場合は、SETPROXYUSERNAME メソッドで指定します。同様に、プロキシ URL にパスワードが必要な場合は、SETPROXYPASSWORD メソッドで指定します。

比較

SETPROXYUSERNAME は、指定されたプロキシ URL に接続しようとするユーザーのドメイン修飾ユーザー名を指定します。SETUSERNAME は、指定された URL に接続しようとするユーザーのドメイン修飾ユーザー名を指定します。

関連項目

メソッド:

- “SETPROXYPASSWORD メソッド” (1756 ページ)
- “SETURL メソッド” (1765 ページ)
- “SETUSERNAME メソッド” (1766 ページ)

SETREQUESTBODYASBINARY メソッド

指定された本文をバイナリ形式で HTTP メソッドによる要求に追加します。

制限事項: SETREQUESTBODYASBINARY メソッドを使用して本文全体をバイナリ形式で返す場合、SETREQUESTBODYCHARACTERSET メソッドを使用することはできません。

構文

package.**SETREQUESTBODYASBINARY**(*variable*);

必須引数

package
HTTP パッケージ変数のインスタンスを指定します。

variable
要求本文データを含むバイナリ変数を指定します。

詳細

SETREQUESTBODYASBINARY メソッドを使用して、要求本文をバイナリ形式で HTTP メソッドに追加します。

HTTP メソッドを完成させるには、SETREQUESTCONTENTTYPE メソッドを呼び出して本体コンテンツの種類を示し、Content-Type:ヘッダーを設定します。次に、EXECUTEMETHOD メソッドを呼び出して、要求を Web サーバーに送信します。

注: コンテンツの長さは、データをトランスコードした後、DS2HTTP クライアントによって計算されます。

関連項目

- “HTTP パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “ADDREQUESTHEADER メソッド” (1731 ページ)
- “CREATEPOSTMETHOD メソッド” (1739 ページ)
- “EXECUTEMETHOD メソッド” (1743 ページ)
- “SETREQUESTBODYASSTRING メソッド” (1760 ページ)
- “SETREQUESTBODYCHARACTERSET メソッド” (1761 ページ)
- “SETREQUESTCONTENTTYPE メソッド” (1762 ページ)

SETREQUESTBODYASSTRING メソッド

指定された本文を文字列形式で HTTP メソッドによる要求に追加します。

要件 要求本文の設定後に、要求本文の文字セットを指定しないようにします。SETREQUESTBODYASSTRING メソッドを使用して要求本文を設定する前に、SETREQUESTBODYCHARACTERSET メソッドを使用して要求本文をエンコードする文字セットを指定する必要があります。

構文

package.**SETREQUESTBODYASSTRING**(*variable*);

必須引数

package
HTTP パッケージ変数のインスタンスを指定します。

variable
要求本文データを含む文字列変数を指定します。

詳細

SETREQUESTBODYASSTRING メソッドを使用して、要求本文を文字列形式で HTTP メソッドに追加します。

HTTP メソッドを完成させるには、SETREQUESTCONTENTTYPE メソッドを呼び出して本体コンテンツの種類を示し、Content-Type:ヘッダーを設定します。次に、EXECUTEMETHOD メソッドを呼び出して、要求を Web サーバーに送信します。

コンテンツの種類が設定されていない場合、HTTP クライアントは要求コンテンツの種類から文字セットを推測します。HTTP クライアントが要求コンテンツの種類から文字セットを推測できない場合、文字セットのデフォルトは UTF-8 となります。しかし、SETREQUESTBODYCHARACTERSET メソッドを使用して、要求本文をエンコードするときに使用する文字セットを指定できます。

注: コンテンツの長さは、データをトランスコードした後、DS2HTTP クライアントによって計算されます。

注: 提供された要求データのエンコーディングが指定されたコンテンツの種類と異なる場合、データはコンテンツの種類で指定されたエンコーディングにトランスコードされます。

関連項目

- “HTTP パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “ADDREQUESTHEADER メソッド” (1731 ページ)
- “CREATEPOSTMETHOD メソッド” (1739 ページ)
- “EXECUTEMETHOD メソッド” (1743 ページ)
- “SETREQUESTBODYASBINARY メソッド” (1759 ページ)
- “SETREQUESTBODYCHARACTERSET メソッド” (1761 ページ)
- “SETREQUESTCONTENTTYPE メソッド” (1762 ページ)

SETREQUESTBODYCHARACTERSET メソッド

要求本文をエンコードするときに使用する文字セットを指定します。

制限事項: SETREQUESTBODYASBINARY メソッドを使用して本文全体をバイナリ形式で返す場合、SETREQUESTBODYCHARACTERSET メソッドを使用することはできません。

要件 要求本文の設定後に、要求本文の文字セットを指定しないようにします。SETREQUESTBODYASSTRING メソッドを使用して要求本文を設定する前に、SETREQUESTBODYCHARACTERSET メソッドを使用して要求本文をエンコードする文字セットを指定する必要があります。

構文

package.**SETREQUESTBODYCHARACTERSET**('*variable*');

必須引数

package
HTTP パッケージ変数のインスタンスを指定します。

'*variable*'
文字セットの指定を含む文字列変数を指定します。

参照項目: 文字セットエンコーディング値の完全なリストについては、*SAS 各国語サポート(NLS): リファレンスガイド*の“SAS 言語要素中のエンコーディング値”を参照してください。

詳細

要求本文をエンコードする文字セットを指定しない場合、HTTP クライアントは要求コンテンツの種類から文字セットを推測します。HTTP クライアントが要求コンテンツの種類から文字セットを推測できない場合、文字セットのデフォルトは UTF-8 となります。

関連項目

メソッド:

- “SETREQUESTBODYASBINARY メソッド” (1759 ページ)
- “SETREQUESTBODYASSTRING メソッド” (1760 ページ)
- “SETRESPONSEBODYCHARACTERSET メソッド” (1763 ページ)

SETREQUESTCONTENTTYPE メソッド

HTTP メソッドによる要求の本文コンテンツの種類を指定します。

構文

package.**SETREQUESTCONTENTTYPE**(*content-type*);

必須引数

content-type
コンテンツの種類の値を含む変数を指定します。

注: 可能なコンテンツの種類については、HTTP リファレンスを参照してください。

package
HTTP パッケージ変数のインスタンスを指定します。

詳細

SETREQUESTCONTENTTYPE メソッドを使用して、HTTP メソッドによる本体コンテンツの種類を指定します。

関連項目

- “HTTP パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “CREATEPOSTMETHOD メソッド” (1739 ページ)
- “EXECUTEMETHOD メソッド” (1743 ページ)
- “SETREQUESTBODYASBINARY メソッド” (1759 ページ)
- “SETREQUESTBODYASSTRING メソッド” (1760 ページ)

SETRESPONSEBODYCHARACTERSET メソッド

応答本文をデコードするときに使用する文字セットを指定します。

制限事項: GETRESPONSEBODYASBINARY メソッドを使用して HTTP 応答から本文全体をバイナリ形式で返す場合、SETRESPONSEBODYCHARACTERSET メソッドを使用することはできません。

要件 応答本文の取得後に、応答本文の文字セットを設定しないようにします。応答本文の文字セットは、GETRESPONSEBODYASSTRING メソッドで要求本文を取得する前に、SETRESPONSEBODYCHARACTERSET メソッドで指定する必要があります。

構文

package.**SETRESPONSEBODYCHARACTERSET**('*variable*');

必須引数

package
HTTP パッケージ変数のインスタンスを指定します。

'*variable*'
文字セットの指定を含む文字列変数を指定します。

参照項目: 文字セットエンコーディング値の完全なリストについては、*SAS 各国語サポート(NLS): リファレンスガイド*の“SAS 言語要素中のエンコーディング値”を参照してください。

詳細

応答本文をエンコードする文字セットを指定しない場合、HTTP クライアントは応答コンテンツの種類から文字セットを推測します。HTTP クライアントが応答コンテンツの種類から文字セットを推測できない場合、文字セットのデフォルトは UTF-8 となります。

関連項目

メソッド:

- “GETRESPONSEBODYASBINARY メソッド” (1745 ページ)
- “GETRESPONSEBODYASSTRING メソッド” (1746 ページ)
- “SETREQUESTBODYCHARACTERSET メソッド” (1761 ページ)

SETSOCKETTIMEOUT メソッド

HTTP Web サーバーからの応答を待つ、ソケットタイムアウト値を指定します。

構文

package.**SETSOCKETTIMEOUT**(*time-out-value*);

必須引数

package
HTTP パッケージ変数のインスタンスを指定します。

time-out-value
Web サーバーからの応答を待つ、デフォルトのソケットタイムアウトをミリ秒単位で指定します。

詳細

SETSOCKETTIMEOUT メソッドを使用して、Web サーバーからの応答を待つ時間を指定します。

デフォルトのタイムアウトが長すぎたり短すぎたりする場合に、新しい CREATE メソッドごとに SETSOCKETTIMEOUT メソッドを設定できます。

例

次の例では、ソケットのタイムアウト値を設定します。

```
data _null_;
  method init();
    declare package http h();
    declare int status_code rc;

    /* get method call with 10 second delay */
    rc = h.createGetMethod('http://httpbin.org/delay/10');
    if (rc ne 0) then put 'TEST ERROR:' rc=;

    /* set 5 second timeout -- execute should timeout */
    rc = h.setSocketTimeout(5000);
    if (rc ne 0) then put 'TEST ERROR:' rc=;
    rc = h.executeMethod();
    if (rc eq 0) then put 'TEST ERROR:' rc=;
  end;
enddata;
run;
```

関連項目

"HTTP パッケージの使用" (*SAS DS2 プログラマガイド*)

SETURL メソッド

HTTP クライアントが接続する URL を指定します。

構文

package.**SETURL**('*variable*');

必須引数

package
HTTP パッケージ変数のインスタンスを指定します。

'*variable*'
HTTP クライアントが接続する URL を含む変数を指定します。

詳細

SETURL メソッドを使用すると、HTTP クライアントが接続する URL を指定できます。

クライアントで設定された URL を使用するには、引数を受け入れないバージョンの CREATEGETMETHOD、CREATEHEADMETHOD、CREATEPOSTMETHOD メソッドを使用する必要があります。

SETURL メソッドの使用例を次に示します。

```
h.createGetMethod();
h.setURL('http://httpbin.org/get');
```

比較

SETURL は、URL を指定します。SETPROXYURL は、プロキシ URL を指定します。

関連項目

メソッド:

- “CREATEGETMETHOD メソッド” (1733 ページ)
- “CREATEHEADMETHOD メソッド” (1738 ページ)
- “CREATEPOSTMETHOD メソッド” (1739 ページ)
- “SETPROXYURL メソッド” (1757 ページ)
- “SETUSERNAME メソッド” (1766 ページ)
- “SETPASSWORD メソッド” (1755 ページ)

SETUSERNAME メソッド

指定された URL に接続しようとしているユーザーのドメイン修飾ユーザー名を表す文字列を指定します。

構文

package.**SETUSERNAME**('*variable*');

必須引数

package
HTTP パッケージ変数のインスタンスを指定します。

'*variable*'
ドメイン修飾ユーザー名を含む文字列変数を指定します。

詳細

SETURL メソッドで指定した URL でユーザー名が必要な場合は、SETUSERNAME メソッドで指定します。同様に、URL にパスワードが必要な場合は、SETPASSWORD メソッドで指定します。

比較

SETUSERNAME メソッドは、指定された URL に接続しようとするユーザーのドメイン修飾ユーザー名を指定します。SETPROXYUSERNAME メソッドは、指定されたプロキシ URL に接続しようとするユーザーのドメイン修飾ユーザー名を指定します。

関連項目

メソッド:

- “SETPROXYUSERNAME メソッド” (1758 ページ)
- “SETPASSWORD メソッド” (1755 ページ)
- “SETURL メソッド” (1765 ページ)

STREAMRESPONSEBODYASBINARY メソッド

チャンク化された HTTP 応答の本体をバイナリ形式でストリーミングします。

構文

package.**STREAMRESPONSEBODYASBINARY**(*variable*, *rc*);

必須引数

package
HTTP パッケージ変数のインスタンスを指定します。

variable
応答データチャンクを保持する変数を指定します。

rc
リターンコード値を保持する変数を指定します。

注: リターンコード値 0 は、成功を示します。値 1 は、失敗を示します。

詳細

STREAMRESPONSEBODYASBINARY メソッドを使用して、バイナリ形式で応答本文を取得します。チャンク化された応答本文がストリーミングされます。

EXECUTEMETHODSTREAM メソッドを使用した後、STREAMRESPONSEBODYASBINARY メソッドを呼び出すことができます。

本文データのストリーミングを完了したくない場合は、ABORT メソッドを呼び出してメソッドの実行を停止します。

注: EXECUTEMETHODSTREAM メソッドは、1 つのエンティティとしての 応答本文の取得をサポートしません。

関連項目

- “HTTP パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “ABORT メソッド” (1730 ページ)
- “EXECUTEMETHODSTREAM メソッド” (1744 ページ)
- “STREAMRESPONSEBODYASSTRING メソッド” (1768 ページ)

STREAMRESPONSEBODYASSTRING メソッド

チャンク化された HTTP 応答の本体を文字列形式でストリーミングします。

適用対象: HTTP パッケージ

構文

package.**STREAMRESPONSEBODYASSTRING**(*variable*, *rc*);

必須引数

package
HTTP パッケージ変数のインスタンスを指定します。

variable
応答データチャンクを保持する変数を指定します。

rc
リターンコード値を保持する変数を指定します。

注: リターンコード値 0 は、成功を示します。値 1 は、失敗を示します。

詳細

STREAMRESPONSEBODYASSTRING メソッドを使用して、文字列形式で応答本文を取得します。チャンク化された応答本文がストリーミングされます。

EXECUTEMETHODSTREAM メソッドを使用した後、STREAMRESPONSEBODYASBINARY メソッドを呼び出すことができます。

本文データのストリーミングを完了したくない場合は、ABORT メソッドを呼び出してメソッドの実行を停止します。

注: EXECUTEMETHODSTREAM メソッドは、1 つのエンティティとしての 応答本文の取得をサポートしません。

関連項目

- “HTTP パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “ABORT メソッド” (1730 ページ)
- “EXECUTEMETHODSTREAM メソッド” (1744 ページ)
- “STREAMRESPONSEBODYASBINARY メソッド” (1767 ページ)

18

DS2 JSON パッケージのメソッド、演算子、およびステートメント

ディクショナリ **1772**
CREATEPARSER メソッド 1772
CREATEWRITER メソッド 1773
DECLARE PACKAGE ステートメント: JSON パッケージ 1774
DELETE メソッド: JSON パッケージ 1775
DESTROYPARSER メソッド 1776
DESTROYWRITER メソッド 1777
GETNEXTTOKEN メソッド 1778
ISBOOLEANFALSE メソッド 1780
ISBOOLEANTRUE メソッド 1781
ISFLOAT メソッド 1782
ISINTEGER メソッド 1783
ISLABEL メソッド 1784
ISLEFTBRACE メソッド 1785
ISLEFTBRACKET メソッド 1785
ISNULL メソッド 1786
ISNUMERIC メソッド 1787
ISPARTIAL メソッド 1788
ISRIGHTBRACE メソッド 1789
ISRIGHTBRACKET メソッド 1790
ISSTRING メソッド 1790
NEW 演算子: JSON パッケージ 1791
SETPARSERINPUT メソッド 1792
WRITEARRAYOPEN メソッド 1793
WRITEBOOLEANFALSE メソッド 1794
WRITEBOOLEANTRUE メソッド 1795
WRITECLOSE メソッド 1795
WRITEDOUBLE メソッド 1796
WRITENULL メソッド 1799
WRITEOBJOPEN メソッド 1800
WRITERGETTEXT メソッド 1802
WRITESTRING メソッド 1803

ディクショナリ

CREATEPARSER メソッド

JSON パーサーインスタンスを作成します。

構文

形式 1: *package*.**CREATEPARSER** ( );

形式 2: *package*.**CREATEPARSER** (*json-text*, *tipping-size*);

形式 3: *package*.**CREATEPARSER** (*json-text*);

形式 4: *package*.**CREATEPARSER** (*tipping-size*);

必須引数

package
JSON パッケージのインスタンスを指定します。

json-text
解析する入力 JSON テキストを指定します。

データの種類 NCHAR、NVARCHAR

tipping-size
tipping-size より長い文字列に対して文字列コールバックルーチンを呼び出す前に蓄積する出力 JSON テキストの最小文字数を指定します。

デフォルト 0 は、長さに関係なく、完全な文字列のみが文字列コールバックに返されることを示します。

制限事項 *tipping-size* が設定されている場合、返される最大文字数は(*tipping-size* + 4)です。

データの種類 INTEGER

詳細

CREATEPARSER メソッド構文の形式 1 または形式 4 を使用する場合は、後で SETPARSERINPUT メソッドを呼び出して、解析する JSON テキストを提供する必要があります。

関連項目

- “JSON パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “DESTROYPARSER メソッド” (1776 ページ)
- “SETPARSERINPUT メソッド” (1792 ページ)

CREATEWRITER メソッド

JSON ライターインスタンスを作成します。

構文

package.**CREATEWRITER** ([PRETTY]);

必須引数

package
JSON パッケージのインスタンスを指定します。

オプション引数

PRETTY
インデントを使用して JSON コンテナー構造を示す、より人間が読みやすい形式を作成します。それ以外の場合、出力は 1 行で書き込まれます。

データの種類 NVARCHAR

注 将来のリリースでは、さらに多くのフラグが使用できるようになる可能性があります。

詳細

DS2 JSON パッケージのライターは、現在ストリーミングをサポートしていません。かわりに、WRITE インスタンスは、WRITERGETTEXT メソッドを呼び出して取得されるまでメモリに収集されます。

例

例については、“WRITEARRAYOPEN メソッド” (1793 ページ)を参照してください。

関連項目

- “JSON パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “DESTROYWRITER メソッド” (1777 ページ)
- “WRITERGETTEXT メソッド” (1802 ページ)

DECLARE PACKAGE ステートメント: JSON パッケージ

パッケージ変数を作成し、JSON パッケージのインスタンスを作成できるようにします。

カテゴリ: ローカル

構文

DECLARE PACKAGE JSON *variable*( );

必須引数

variable
JSON パッケージのインスタンスを参照できる名前を指定します。

詳細

DS2 パッケージは、特定のインスタンスを構築して他の DS2 プログラムで使用できる変数とメソッドのコレクションです。

JSON パッケージを使用して、JSON テキストを作成および解析します。JSON パッケージは、DS2 プログラム用に事前定義されています。

パッケージが宣言されると、パッケージのインスタンスを参照できる変数が作成されます。コンストラクター引数がパッケージ変数宣言で提供されている場合、パッケージインスタンスが構築され、構築されたパッケージインスタンスを参照するようにパッケージ変数が設定されます。複数のパッケージ変数を作成でき、単一の DECLARE PACKAGE ステートメントで複数のパッケージインスタンスを構築できます。各パッケージインスタンスは、パッケージの完全に別個のコピーを表します。

JSON パッケージのインスタンスを構築する方法は 2 つあります。

- DECLARE PACKAGE ステートメントを_NEW_演算子と一緒に使用します。

```
declare package json j;
j = _new_ json();
```

- DECLARE PACKAGE ステートメントをそのコンストラクター構文と一緒に使用します。

```
declare package json j();
```

関連項目

- “JSON パッケージの使用” (*SAS DS2 プログラマガイド*)
- “パッケージのコンストラクターとデストラクター” (*SAS DS2 プログラマガイド*)

演算子:

- “_NEW_ 演算子: JSON パッケージ” (1791 ページ)

DELETE メソッド: JSON パッケージ

JSON パッケージを削除します。

構文

package.**DELETE**( );

必須引数

package
JSON パッケージ変数のインスタンスを指定します。

詳細

JSON パッケージが不要になった場合は、DELETE メソッドを使用して削除します。DELETE メソッドは必須ではありません。JSON パッケージの参照カウントがゼロになると、パッケージは自動的に削除されます。

注: DS2 コードを記述して、参照カウントがゼロにならないように循環参照を作成することができます。その場合、インスタンスは、DELETE メソッドを使用して明示的に削除されない限り、RUN ブロックの最後まで存続します。DS2 で循環参照を作成する方法の例を次に示します。

```
proc ds2;
package e / overwrite=yes;
  dcl package e next;
  dcl package e prev;

  method setnext(package e e);
    next = e;
  end;

  method setprev(package e e);
    prev = e;
  end;
endpackage;

data _null_;
  dcl package e e1();
  dcl package e e2();

  method init();
    e1.setnext(e2);
    e2.setprev(e1);
  end;
enddata;
run;
quit;
```

関連項目

"パッケージのコンストラクターとデストラクター" (SAS DS2 プログラマガイド)

DESTROYPARSER メソッド

JSON パーサーインスタンスを破棄します。

構文

package.**DESTROYPARSER** ( );

必須引数

package
JSON パッケージのインスタンスを指定します。

関連項目

- “JSON パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “CREATEPARSER メソッド” (1772 ページ)

DESTROYWRITER メソッド

JSON ライターインスタンスを破棄します。

構文

package.**DESTROYWRITER** ( );

必須引数

package
JSON パッケージのインスタンスを指定します。

例

例については、“WRITEARRAYOPEN メソッド” (1793 ページ)を参照してください。

関連項目

- “JSON パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

■ “CREATEWRITER メソッド” (1773 ページ)

GETNEXTTOKEN メソッド

JSON テキストから次に検証された JSON 言語アイテムまたは要素を返します。

構文

形式 1: *package*.**GETNEXTTOKEN** (*rc*, *token-type*, *parse-flags*);

形式 2: *package*.**GETNEXTTOKEN** (*rc*, *token*, *token-type*, *parse-flags*);

形式 3: *package*.**GETNEXTTOKEN** (*rc*, *token*, *token-type*, *parse-flags*, *line-number*, *column-number*);

必須引数

package
JSON パッケージのインスタンスを指定します。

rc
リターンコード値を保持する変数を指定します。可能なリターンコード値は次のとおりです。

0 成功

100 出力トークン引数の最大長が十分でなかったため、切り捨てが発生しました。

101 完了しました。ユースケースに応じて、このリターンコード値が予期される場合と予期されない場合があります。

300 テキストの終わりです。ユースケースに応じて、このリターンコード値が予期される場合と予期されない場合があります。

301 テキストの解析中にエラーが発生しました。

データの種類 INTEGER

token-type
検索するトークンの種類を保持する変数を指定します。*Token-type* には次のいずれかの値を指定できます。

4 ブール true

8 ブール false

16 左角かっこ([ )

32 右角かっこ( ])

64 左中かっこ({)

128 右中かっこ(})

256 文字列

512 数値

1024 Null

データの種類 INTEGER

parse-flags
JSON プロセッサのパーサーによって提供される出力フラグセットを保存する変数を指定します。解析されるトークンが文字列の場合、有効な出力値は次のとおりです。

1 トークンはオブジェクトのラベルです

2 トークンが不完全です

3 トークンはオブジェクト内のラベルであり、不完全です

解析されるトークンが数値の場合、有効な値は次のとおりです。

3 トークンは整数です

4 トークンは浮動小数点数です

データの種類 INTEGER

token
次のトークンまたは文字列を保持する変数を指定します。

データの種類 CHAR

line-number
トークンが配置されているテキスト内の行番号を保持する変数を指定します。

データの種類 BIGINT

ヒント 行番号を使用して、テキスト内のトークンの場所を判別するのに役立てることができます。

column-number
トークンが配置されているテキスト内の列番号を保持する変数を指定します。

データの種類 BIGINT

ヒント 列番号を使用して、テキスト内のトークンの場所を判別するのに役立てることができます。

詳細

parseFlags を除き、GETNEXTTOKEN メソッドの引数はすべて IN_OUT 引数です。これらは、GETNEXTTOKEN メソッドの呼び出し元に値を返すために使用されます。値は参照によって渡されます。

CREATEPARSER メソッドを使用して、入力 JSON テキストを提供できます。IS*メソッドを使用して、トークンタイプをテストできます。または、IF 式を使用して *token* を文字列値に関連付けることもできます。GETNEXTTOKEN メソッドの使用例については、“例: HTTP および JSON パッケージを使用して REST 形式のファイルを抽出および解析する” (2180 ページ)を参照してください。

関連項目

概念:

- “JSON テキストの解析” (*SAS DS2 プログラマガイド*)

メソッド:

- “CREATEPARSER メソッド”
- “ISBOOLEANFALSE メソッド” (1780 ページ)
- “ISBOOLEANTRUE メソッド” (1781 ページ)
- “ISFLOAT メソッド” (1782 ページ)
- “ISINTEGER メソッド” (1783 ページ)
- “ISLABEL メソッド” (1784 ページ)
- “ISLEFTBRACE メソッド” (1785 ページ)
- “ISLEFTBRACKET メソッド” (1785 ページ)
- “ISNULL メソッド” (1786 ページ)
- “ISNUMERIC メソッド” (1787 ページ)
- “ISPARTIAL メソッド” (1788 ページ)
- “ISRIGHTBRACE メソッド” (1789 ページ)
- “ISRIGHTBRACKET メソッド” (1790 ページ)
- “ISSTRING メソッド” (1790 ページ)

ISBOOLEANFALSE メソッド

トークンが偽の場合、true を返します。

構文

package.**ISBOOLEANFALSE**(*token-type*);

必須引数

package
JSON パッケージのインスタンスを指定します。

token-type
GETNEXTTOKEN メソッドから取得したトークンタイプを指定します。

データの種類 INTEGER

詳細

ISBOOLEANFALSE メソッドを使用して、GETNEXTTOKEN メソッドから取得したトークンタイプをテストできます。

関連項目

- “JSON パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “GETNEXTTOKEN メソッド” (1778 ページ)
- “ISBOOLEANTRUE メソッド” (1781 ページ)
- “WRITEBOOLEANFALSE メソッド” (1794 ページ)

ISBOOLEANTRUE メソッド

トークンが真の場合、true を返します。

構文

package.**ISBOOLEANTRUE**(*token-type*);

必須引数

package
JSON パッケージのインスタンスを指定します。

token-type
GETNEXTTOKEN メソッドから取得したトークンタイプを指定します。

データの種類 INTEGER

詳細

ISBOOLEANTRUE メソッドを使用して、GETNEXTTOKEN メソッドから取得したトークンタイプをテストできます。

関連項目

- “JSON パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “GETNEXTTOKEN メソッド” (1778 ページ)
- “ISBOOLEANFALSE メソッド” (1780 ページ)
- “WRITEBOOLEANTRUE メソッド” (1795 ページ)

ISFLOAT メソッド

トークンがテキスト形式の浮動小数点数の場合は true を返します。

構文

package.**ISFLOAT**(*token-type*, *parse-flags*);

必須引数

package
JSON パッケージのインスタンスを指定します。

token-type
GETNEXTTOKEN メソッドから取得したトークンタイプを指定します。

データの種類 INTEGER

parse-flags
GETNEXTTOKEN メソッドから取得した解析フラグを指定します。

データの種類 INTEGER

詳細

ISFLOAT メソッドを使用して、GETNEXTTOKEN メソッドから取得したトークンタイプをテストできます。

関連項目

- “JSON パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “GETNEXTTOKEN メソッド” (1778 ページ)
- “ISINTEGER メソッド” (1783 ページ)
- “ISNUMERIC メソッド” (1787 ページ)

ISINTEGER メソッド

トークンがテキスト形式の整数の場合は true を返します。

構文

package.**ISINTEGER** (*token-type*, *parse-flags*);

必須引数

package
JSON パッケージのインスタンスを指定します。

token-type
GETNEXTTOKEN メソッドから取得したトークンタイプを指定します。

データの種類 INTEGER

parse-flags
GETNEXTTOKEN メソッドから取得した解析フラグを指定します。

データの種類 INTEGER

詳細

ISINTEGER メソッドを使用して、GETNEXTTOKEN メソッドから取得したトークンタイプをテストできます。

関連項目

- “JSON パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “GETNEXTTOKEN メソッド” (1778 ページ)
- “ISFLOAT メソッド” (1782 ページ)
- “ISNUMERIC メソッド” (1787 ページ)

ISLABEL メソッド

トークンがオブジェクトラベルの場合は true を返します。

構文

package.**ISLABEL**(*token-type*, *parse-flags*);

必須引数

package
JSON パッケージのインスタンスを指定します。

token-type
GETNEXTTOKEN メソッドから取得したトークンタイプを指定します。

データの種類 INTEGER

parse-flags
GETNEXTTOKEN メソッドから取得した解析フラグを指定します。

データの種類 INTEGER

詳細

ISLABEL メソッドを使用して、GETNEXTTOKEN メソッドから取得したトークンタイプをテストできます。

関連項目

- “JSON パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “GETNEXTTOKEN メソッド” (1778 ページ)

ISLEFTBRACE メソッド

トークンが左中かっこ({)の場合は true を返します。

構文

package.**ISLEFTBRACE**(*token-type*);

必須引数

package
JSON パッケージのインスタンスを指定します。

token-type
GETNEXTTOKEN メソッドから取得したトークンタイプを指定します。

データの種類 INTEGER

詳細

ISLEFTBRACE メソッドを使用して、GETNEXTTOKEN メソッドから取得したトークンタイプをテストできます。

関連項目

- "JSON パッケージの使用" (*SAS DS2 プログラマガイド*)

メソッド:

- "GETNEXTTOKEN メソッド" (1778 ページ)
- "ISRIGHTBRACE メソッド" (1789 ページ)

ISLEFTBRACKET メソッド

トークンが左角かっこ([ )の場合は true を返します。

構文

package.**ISLEFTBRACKET**(*token-type*);

必須引数

package
JSON パッケージのインスタンスを指定します。

token-type
GETNEXTTOKEN メソッドから取得したトークンタイプを指定します。

データの種類 INTEGER

詳細

ISLEFTBRACKET メソッドを使用して、GETNEXTTOKEN メソッドから取得したトークンタイプをテストできます。

関連項目

- “JSON パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “GETNEXTTOKEN メソッド” (1778 ページ)
- “ISRIGHTBRACKET メソッド” (1790 ページ)

ISNULL メソッド

トークンが null の場合は true を返します。

構文

package.**ISNULL**(*token-type*);

必須引数

package
JSON パッケージのインスタンスを指定します。

token-type
GETNEXTTOKEN メソッドから取得したトークンタイプを受け取ります。

データの種類 INTEGER

詳細

ISNULL メソッドを使用して、GETNEXTTOKEN メソッドから取得したトークンタイプをテストできます。

関連項目

- “JSON パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “GETNEXTTOKEN メソッド” (1778 ページ)
- “WRITENULL メソッド” (1799 ページ)

ISNUMERIC メソッド

トークンがテキスト形式の数値の場合は true を返します。

構文

package.**ISNUMERIC**(*token-type*);

必須引数

package
JSON パッケージのインスタンスを指定します。

token-type
GETNEXTTOKEN メソッドから取得したトークンタイプを受け取ります。

データの種類 INTEGER

詳細

ISNUMERIC メソッドを使用して、GETNEXTTOKEN メソッドから取得したトークンタイプをテストできます。

関連項目

- “JSON パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “GETNEXTTOKEN メソッド” (1778 ページ)
- “ISFLOAT メソッド” (1782 ページ)
- “ISINTEGER メソッド” (1783 ページ)

ISPARTIAL メソッド

チップが原因でトークンが不完全な場合は true を返します。

構文

package.**ISPARTIAL**(*parse-flags*);

必須引数

package
JSON パッケージのインスタンスを指定します。

parse-flags
GETNEXTTOKEN メソッドから取得した解析フラグを受け取ります。

データの種類 INTEGER

詳細

ISPARTIAL メソッドを使用して、GETNEXTTOKEN メソッドから取得したトークンタイプをテストできます。

関連項目

- “JSON パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “GETNEXTTOKEN メソッド” (1778 ページ)

ISRIGHTBRACE メソッド

トークンが右中かっこ(})の場合は true を返します。

構文

package.**ISRIGHTBRACE**(*token-type*);

必須引数

package
JSON パッケージのインスタンスを指定します。

token-type
GETNEXTTOKEN メソッドから取得したトークンタイプを受け取ります。

データの種類 INTEGER

詳細

ISRIGHTBRACE メソッドを使用して、GETNEXTTOKEN メソッドから取得したトークンタイプをテストできます。

関連項目

- “JSON パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “GETNEXTTOKEN メソッド” (1778 ページ)
- “ISLEFTBRACE メソッド” (1785 ページ)

ISRIGHTBRACKET メソッド

トークンが右角かっこ( ])の場合は true を返します。

構文

package.**ISRIGHTBRACKET**(*token-type*);

必須引数

package
JSON パッケージのインスタンスを指定します。

token-type
GETNEXTTOKEN メソッドから取得したトークンタイプを受け取ります。

データの種類 INTEGER

詳細

ISRIGHTBRACKET メソッドを使用して、GETNEXTTOKEN メソッドから取得したトークンタイプをテストできます。

関連項目

- “JSON テキストの解析” (*SAS DS2 プログラマガイド*)

メソッド:

- “GETNEXTTOKEN メソッド” (1778 ページ)
- “ISLEFTBRACKET メソッド” (1785 ページ)

ISSTRING メソッド

トークンが文字列の場合は true を返します。

構文

package.**ISSTRING**(*token-type*);

必須引数

package
JSON パッケージのインスタンスを指定します。

token-type
GETNEXTTOKEN メソッドから取得したトークンタイプを受け取ります。

データの種類 INTEGER

詳細

ISSTRING メソッドを使用して、GETNEXTTOKEN メソッドから取得したトークンタイプをテストできます。

関連項目

- "JSON パッケージの使用" (*SAS DS2 プログラマガイド*)

メソッド:

- "GETNEXTTOKEN メソッド" (1778 ページ)

NEW 演算子: JSON パッケージ

JSON パッケージのインスタンスを構築します。

注: かっこの前のエスケープ文字(\)は、構文にかっこが必要であることを示します。

構文

package-variable = **_NEW_ JSON**( );

必須引数

package-variable
パッケージのインスタンスを参照できる名前を指定します。

詳細

DS2 パッケージは、特定のインスタンスを構築して他の DS2 プログラムで使用できる変数とメソッドのコレクションです。

DECLARE PACKAGE ステートメントを使用して、JSON パッケージ変数を宣言します。JSON パッケージ変数を宣言した後、_NEW_演算子を使用して JSON パッケージのインスタンスをインスタンス化し、新しいパッケージインスタンスを JSON パッケージ変数に割り当てます。

```
declare package json jsontxt;
jsontxt = _new_ json( );
```

DECLARE PACKAGE と_NEW_演算子を使用して JSON パッケージを宣言およびインスタンス化する 2 ステップのプロセスのかわりに、コンストラクターメソッドとして DECLARE PACKAGE ステートメントを使用することにより、パッケージを 1 ステップで宣言およびインスタンス化できます。これは、DECLARE PACKAGE ステートメントのみを使用した同じ例です。

```
declare package json jsontxt( );
```

注: パッケージ変数は、すべての変数スコーピングルールの対象となります。詳細については、“パッケージ、スコープ、および有効期間” (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- “JSON パッケージの使用” (*SAS DS2 プログラマガイド*)
- “パッケージのコンストラクターとデストラクター” (*SAS DS2 プログラマガイド*)

ステートメント:

- “DECLARE PACKAGE ステートメント: JSON パッケージ” (1774 ページ)

SETPARSERINPUT メソッド

パーサーがさらにテキストを必要とする場合に、パーサーに JSON テキストを提供します。

制限事項: このメソッドは、パーサーにテキストがない場合にのみ有効です。パーサーにテキストがある場合は、エラーが返されます。

構文

package.**SETPARSERINPUT**([*json-text*,]);

必須引数

package
JSON パッケージのインスタンスを指定します。

オプション引数

json-text
パーサーの JSON テキストを指定します。

データの種類 CHAR、VARCHAR、NCHAR、NVARCHAR

関連項目

- “JSON パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “CREATEPARSER メソッド” (1772 ページ)

WRITEARRAYOPEN メソッド

配列の開始を示す開き角かっこ([ )を書き込みます。

構文

package.**WRITEARRAYOPEN**( );

必須引数

package
JSON パッケージのインスタンスを指定します。

詳細

WRITEARRAYOPEN メソッドは、配列コンテナーを明示的に開きます。この開いた配列コンテナーは、WRITECLOSE メソッドを使用して明示的に閉じる必要があります。

例

次の例では、ライターインスタンスを作成し、数値を JSON 配列コンテナーに書き込みます。

```
data _null_;
   method init();
      dcl package json j();
      dcl double dblVal;
      dcl int rc;
      dcl nvarchar(15) jsontxt;

      rc = j.createWriter();
      rc = j.writeArrayOpen();
      dblVal = 12345678.1234;
      rc = j.writeDouble( dblVal,13, 5 );
      rc = j.writeClose();
      j.writerGetText( rc, jsontxt);
      put jsontxt=;
      rc = j.destroywriter();
   end;
enddata;
 run;
```

次の行が SAS ログに書き込まれます。

```
jsontxt=[1.2346e+07   ]
```

関連項目

- “JSON パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “WRITECLOSE メソッド” (1795 ページ)

WRITEBOOLEANFALSE メソッド

ブール値の false をテキストに書き込みます。

構文

package.**WRITEBOOLEANFALSE**( );

必須引数

package
JSON パッケージのインスタンスを指定します。

関連項目

- “JSON パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “ISBOOLEANFALSE メソッド” (1780 ページ)
- “WRITEBOOLEANTRUE メソッド” (1795 ページ)

WRITEBOOLEANTRUE メソッド

テキストにブール値の true を書き込みます。

構文

package.**WRITEBOOLEANTRUE**( );

必須引数

package
JSON パッケージのインスタンスを指定します。

関連項目

- “JSON パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “ISBOOLEANTRUE メソッド” (1781 ページ)
- “WRITEBOOLEANFALSE メソッド” (1794 ページ)

WRITECLOSE メソッド

対応するオブジェクト(})または配列( ])を閉じます。

構文

package.**WRITECLOSE**( );

必須引数

package
JSON パッケージのインスタンスを指定します。

詳細

WRITECLOSE メソッドは、WRITEARRAYOPEN メソッドまたは WRITEOBJOPEN メソッドで明示的に開かれた、いずれかの種類の最近開かれたコンテナーを閉じます。コンテナーを WRITE*OPEN メソッドで明示的に開いた場合にのみ、コンテナーの WRITECLOSE メソッドを呼び出す必要があります。

例

例については、“WRITEARRAYOPEN メソッド” (1793 ページ)を参照してください。

関連項目

- “JSON パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “WRITEOBJOPEN メソッド” (1800 ページ)
- “WRITEARRAYOPEN メソッド” (1793 ページ)

WRITEDOUBLE メソッド

DOUBLE 値をテキスト形式で書き込みます。

構文

形式 1: *package*.**WRITEDOUBLE**(*double-value*);

形式 2: *package*.**WRITEDOUBLE**(*double-value*, *width*);

形式 3: *package*.**WRITEDOUBLE**(*double-value*, *width*, *precision*);

形式 4: *package*.**WRITEDOUBLE**(*double-value*, *width*, *precision*, *options*);

必須引数

package
JSON パッケージのインスタンスを指定します。

double-value
書き込む値を指定します。

データの種類 DOUBLE

width
フォーマットされた値の幅を指定します。

デフォルト 0

データの種類 DOUBLE

precision
基数文字の後に表示される桁数を示します。

デフォルト 15

データの種類 DOUBLE

options
フォーマットのフラグを指定します。*options* は、次のいずれかの値になります。

BESTFIT
10 進表記の[–]*dddd.ddd* の形式で値をフォーマットするか、指数表記の[–]*d.ddd*e±*dd* の形式で値をフォーマットします。フォーマットスタイルは値によって異なります。

注 有効な幅は 1 文字から 32 文字です。

精度は無視されます。

BESTFITBIG
10 進表記の[–]*dddd.ddd* の形式で値をフォーマットするか、指数表記の[–]*d.ddd*E±*dd* の形式で値をフォーマットします。フォーマットスタイルは値によって異なります。

注 有効な幅は 1 文字から 32 文字です。

精度は無視されます。

SASBEST
BESTw.出力形式に準拠します。値は指定された幅内でフォーマットされます。可能であれば、10 進表記が生成されます。作成できない場合は、指数表記が[–]*ddd*.dddE[–]*dd* の形式で生成されます。

注 基数文字の後のゼロは表示されません。

SASEW
E*w*.出力形式ルールに準拠します。値は指定された幅内でフォーマットされます。常に、指数表記が、[-]*ddd*.*ddd*E±*dd* の形式で生成されます。

注　有効な幅は 7 文字から 32 文字です。

精度は無視されます。

SASEWD
SAS XP Services %*w*.*d*e ルールに準拠します。値は指定された *width* 内でフォーマットされます。常に、指数表記が、[–]*ddd*.*ddd*E±*dd* の形式で生成されます。

注　有効な幅は 7 桁から 32 桁です。

有効な精度は 0 桁から 31 桁です。

SASWD
w.*d* 出力形式ルールに準拠します。

注　値は指定された *width* 内でフォーマットされます。*width* が小さすぎると、SASBEST が示す動作に戻ります。

DECIMAL
指数表記の[–]*dddd*.*dd* の形式で値をフォーマットします。*precision* を使用して、基数の後の桁数を決定します。

注　基数文字は、その後に表示する数字がない場合や、*precision* がゼロに設定されている場合は表示されません。

FRACTION
値の小数部分を[–]0.*dddd* の形式でフォーマットします。

注　基数文字の前の桁は常にゼロです。

INTEGER
値の小数部分を[–]*dddd* の形式でフォーマットします。

注　値の小数部分は無視され、基数文字は結果に追加されません。

SNOTE
指数表記の[–]*d*.*ddd*e±*dd* の形式で値をフォーマットします。指数の前に小文字の'e'を使用します。

SNOTEBIG
指数表記の[–]*d*.*ddd*E±*dd* の形式で値をフォーマットします。指数の前に大文字の'E'を使用します。

デフォルト　BESTFIT

データの種類　CHAR

例

次の例では、DOUBLE 値を配列に書き込みます。

```
data _null_;
   dcl package logger lgr( 'App.tk.D2PKG.JSON' );
   dcl package json j();
   dcl nvarchar(256) matrixA;
   dcl int rc;

   method init();
      rc = j.createWriter();
      rc = j.writeArrayOpen();
      rc = j.writeArrayOpen();
      rc = j.writeDouble(1.1);
      rc = j.writeDouble(1.2);
      rc = j.writeClose();
      rc = j.writeArrayOpen();
      rc = j.writeDouble(2.1);
      rc = j.writeDouble(2.2);
      rc = j.writeClose();
      rc = j.writeClose();
      j.writerGetText( rc, matrixA );
      lgr.log( 4, 'matrix A = $s', matrixA );
      rc = j.destroyWriter();
   end;
enddata;
run;
```

次の行が SAS ログに書き込まれます。

```
NOTE: matrix A = [[1.1,1.2],[2.1,2.2]]
```

関連項目

"JSON パッケージの使用" (*SAS DS2 プログラマガイド*)

WRITENULL メソッド

テキストに null 値を書き込みます。

構文

package.**WRITENULL**( );

必須引数

package
JSON パッケージのインスタンスを指定します。

関連項目

- “JSON パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “ISNULL メソッド” (1786 ページ)

WRITEOBJOPEN メソッド

オブジェクトの開始を示す開き角かっこ({)を書き込みます。

構文

package.**WRITEOBJOPEN**( );

必須引数

package
JSON パッケージのインスタンスを指定します。

詳細

WRITEOBJOPEN メソッドは、オブジェクトコンテナーを明示的に開きます。この開いたオブジェクトコンテナーは、WRITECLOSE メソッドを使用して明示的に閉じる必要があります。

例

```
proc ds2;
data _null_;
  method init();
    dcl package json j();
    dcl nvarchar(16384) text;
    dcl int rc;
```

```
    j.createWriter('PRETTY');
    j.writeObjOpen();
     j.writeString('TableMetadata');    j.writeObjOpen();
       j.writeString('ColumnMetadata'); j.writeArrayOpen();
         j.writeObjOpen();
           j.writeString('Name');      j.writeString('fname');
           j.writeString('Label');     j.writeString('First Name');
           j.writeString('Type');      j.writeString('VARCHAR');
           j.writeString('Length');    j.writeDouble(256);
         j.writeClose();

         j.writeObjOpen();
           j.writeString('Name');      j.writeString('lname');
           j.writeString('Label');     j.writeString('Last Name');
           j.writeString('Type');      j.writeString('VARCHAR');
           j.writeString('Length');    j.writeDouble(256);
         j.writeClose();

         j.writeObjOpen();
           j.writeString('Name');      j.writeString('dob');
           j.writeString('Label');     j.writeString('Date of Birth');
           j.writeString('Type');      j.writeString('DATE');
           j.writeString('Length');    j.writeDouble(10);
         j.writeClose();
       j.writeClose();
     j.writeClose();
    j.writeClose();
    j.writerGetText(rc, text);

     put text;
  end;
enddata;
run;
quit;
```

SAS ログは次の結果を返します。

```
{
 "TableMetadata": {
  "ColumnMetadata": [
   {
    "Name": "fname",
    "Label": "First Name",
    "Type": "VARCHAR",
    "Length": 256
   },
   {
    "Name": "lname",
    "Label": "Last Name",
    "Type": "VARCHAR",
    "Length": 256
   },
   {
    "Name": "dob",
    "Label": "Date of Birth",
    "Type": "DATE",
    "Length": 10
   }
  ]
 }
}
```

関連項目

- “JSON パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “WRITECLOSE メソッド” (1795 ページ)

WRITERGETTEXT メソッド

ライターによって生成された JSON テキストを取得します。

構文

package.**WRITERGETTEXT**(*rc*, *json-text*);

必須引数

package
JSON パッケージのインスタンスを指定します。

rc
リターンコード値を保持する変数を指定します。可能なリターンコード値は次のとおりです。

0 成功

100 出力トークン引数の最大長が十分でなかったため、切り捨てが発生しました。

101 完了しました。ユースケースに応じて、このリターンコード値が予期される場合と予期されない場合があります。

300 テキストの終わりです。ユースケースに応じて、このリターンコード値が予期される場合と予期されない場合があります。

301 ステータス条件が返されました。

データの種類 INTEGER

json-text
JSON テキストを受け取る変数を指定します。

データの種類 NVARCHAR

詳細

DS2 JSON パッケージのライターは、現在ストリーミングをサポートしていません。かわりに、WRITE インスタンスは、WRITERGETTEXT メソッドを呼び出して取得されるまでメモリに収集されます。

例

例については、“WRITEARRAYOPEN メソッド” (1793 ページ)を参照してください。

関連項目

- “JSON パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “CREATEWRITER メソッド” (1773 ページ)

WRITESTRING メソッド

JSON テキストに文字列を書き込みます。

注: 構文規則の中かっこは、構文のグループ化を示します。

構文

package.**WRITESTRING**({[' | "]}*string*{[' | "]}, *flags*);

必須引数

package
JSON パッケージのインスタンスを指定します。

{['"] }*string*{['"] }
書き込む文字列を指定します。

データの種類 NVARCHAR

ヒント 文字列は、単一引用符で囲まれた文字列リテラル、引用符のない通常の識別子、または二重引用符で囲まれた区切られた識別子にすることができます。

flags
文字列の特別な処理のオプションを指定します。次の値が可能です。

0
0 は、フラグがないことを示します。

16
16 (JSN_SkipScan)は、入力文字列の通常のスキャンと JSON エンコーディングをスキップする必要があることを示します。これは、文字列に無効な文字または JSON エスケープ文字が含まれていないことが判明しているか、呼び出し元が文字列に対して JSON スキャンまたはエンコードをすでに実行していることを意味します。後者の場合、指定された文字列と引用符と区切り文字のみが挿入されます。

通常のスキャンを行うには、このフラグを省略することで、通常のスキャンと JSON エンコーディングが実行されます。

32
32 (JSN_TrimBlanks)は、ライターは文字列から末尾の空白を削除します。

48
48 は、ライターはスキャンをスキップし、空白を削除します。

データの種類 INTEGER

例

次の例は、さまざまな種類の文字列値を示しています。

```
data _null_;
  dcl package logger lgr( 'App.tk.D2PKG.JSON' );
  dcl package json j();
  dcl nvarchar(2000) txt;
  dcl varchar(20) "a**b";
```

```
   dcl varchar(20) myStr;
   dcl int rc;

   method init();
      rc = j.createWriter();
      rc = j.writeArrayOpen();
      rc = j.writeString( 'aaaAAA', 0 );
      "a**b" = 'bbbBBB';
      rc = j.writeString( "a**b", 0 );
      myStr = 'cccCCC';
      rc = j.writeString( myStr, 0 );
      rc = j.writeClose();
      j.writerGetText( rc, txt );
      lgr.log( 3, 'txt = $s', txt );
      rc = j.destroyWriter();
   end;
enddata;
run;
```

関連項目

“JSON パッケージの使用” (*SAS DS2 プログラマガイド*)

19

DS2 ロガーパッケージのメソッド、演算子、およびステートメント

ディクショナリ **1807**
DECLARE PACKAGE ステートメント: ロガーパッケージ 1807
DELETE メソッド: ロガーパッケージ 1809
ISLEVELACTIVE メソッド 1810
LOG メソッド: ロガーパッケージ 1811
NEW 演算子: ロガーパッケージ 1814

ディクショナリ

DECLARE PACKAGE ステートメント: ロガーパッケージ

パッケージ変数を作成し、ロガーパッケージのインスタンスを作成するオプションを提供します。

カテゴリ: ローカル

ヒント: ロガーパッケージには PACKAGE ステートメントは必要ありません。

構文

DECLARE PACKAGE LOGGER *variable*([*logger-name*]);

必須引数

variable
ロガーパッケージのインスタンスを参照できる名前を指定します。

オプション引数

logger-name
SAS ログ機能で定義されているロガーの名前を指定します。

デフォルト　SAS ルートロガー

詳細

DS2 パッケージは、特定のインスタンスを構築して他の DS2 プログラムで使用できる変数とメソッドのコレクションです。

ロガーパッケージを使用して、SAS ログ機能とのインターフェイスを取ります。ロガーパッケージは、DS2 プログラム用に事前定義されています。ログ機能の詳細については、*SAS プログラムとジョブの SAS Viya ログ機能*を参照してください。

DECLARE PACKAGE ステートメントを使用して、ロガーパッケージを宣言します。これにより、ロガーパッケージがロガー名に関連付けられます。新しいロガーパッケージを宣言した後、指定されたログレベルでロガーにメッセージを送信できます。

パッケージが宣言されると、パッケージのインスタンスを参照できる変数が作成されます。コンストラクター引数がパッケージ変数宣言で提供されている場合、パッケージインスタンスが構築され、構築されたパッケージインスタンスを参照するようにパッケージ変数が設定されます。複数のパッケージ変数を作成でき、単一の DECLARE PACKAGE ステートメントで複数のパッケージインスタンスを構築できます。各パッケージインスタンスは、パッケージの完全に別個のコピーを表します。

ロガーパッケージのインスタンスを構築する方法は 2 つあります。

- DECLARE PACKAGE ステートメントを_NEW_演算子と一緒に使用します。

```
declare package logger logpkg;
logpkg = _new_ logger();
```

- DECLARE PACKAGE ステートメントをそのコンストラクター構文と一緒に使用します。

```
declare package logger logpkg();
```

ロガーパッケージの詳細については、“ロガーパッケージの使用” (*SAS DS2 プログラマガイド*)を参照してください。

例

この例では、ロガーパッケージのインスタンスを作成します。

```
data _null_;
  dcl package logger l();
  method init();
    l.log('i', 'Hello World!');
  end;
enddata;
```

関連項目

- “ロガーパッケージの使用” (*SAS DS2 プログラマガイド*)
- “パッケージのコンストラクターとデストラクター” (*SAS DS2 プログラマガイド*)

演算子:

- “_NEW_ 演算子: ロガーパッケージ” (1814 ページ)

DELETE メソッド: ロガーパッケージ

ロガーパッケージを削除します。

注: DELETE メソッドは必須ではありません。ロガーパッケージインスタンスを参照するロガーパッケージ変数がない場合、パッケージインスタンスは自動的に削除されます。

構文

package.**DELETE**( );

必須引数

package
ロガーパッケージ変数のインスタンスを指定します。

詳細

ロガーパッケージが不要になった場合は、DELETE メソッドを使用して削除します。ロガーパッケージを削除した後に使用しようとすると、ログにエラーが書き込まれます。

関連項目

"パッケージのコンストラクターとデストラクター" (*SAS DS2 プログラマガイド*)

ISLEVELACTIVE メソッド

ロガー名に関連付けられているロガーパッケージがロガーレベルでメッセージを抑制するかどうかを示す値を返します。

構文

package.**ISLEVELACTIVE**(['])*level*[']);

必須引数

package
ロガーパッケージ変数のインスタンスを指定します。

[']*level*[']
指定されたロガーパッケージにロギング要求が適用されるレベルを指定する数値。

要件 *level* は、重要度値または重要度値の 1 文字の省略形のいずれかを含む文字列である必要があります。有効な値は次のとおりです。

- TRACE の場合は 2 または'T'
- DEBUG の場合は 3 または'D'
- INFO の場合は 4 または'I'
- WARN の場合は 5 または'W'
- ERROR の場合は 6 または'E'
- FATAL の場合は 7 または'F'

level に文字を使用する場合、文字は単一引用符で囲む必要があります。数値は引用符で囲むことができますが、囲む必要はありません。

例

これらは ISLEVELACTIVE メソッドの例です。

```
mylog.islevelactive(5);

testlog.islevelactive('F');
```

関連項目

"ロガーパッケージの使用" (*SAS DS2 プログラマガイド*)

LOG メソッド: ロガーパッケージ

指定されたメッセージを指定されたレベルのロガーに送信します。

構文

形式 1: *package*.**LOG**(['])*level*['], '*raw-message*');

形式 2: *package*.**LOG**(['])*level*['], [*message-format*]'*argument-1*' [..., *argument-9*]);

必須引数

package
ロガーパッケージ変数のインスタンスを指定します。

['])*level*['])
指定されたロガーパッケージにロギング要求が適用されるレベルを指定する数値。

要件 *level* は、重要度値または重要度値の 1 文字の省略形のいずれかを含む文字列である必要があります。有効な値は次のとおりです。レベルを示す単語のどの部分も有効であることに注意してください。たとえば、*i*、*in*、*inf*、*info* が有効です。

- TRACE の場合は 2 または'T'
- DEBUG の場合は 3 または'D'
- INFO の場合は 4 または'I'
- WARN の場合は 5 または'W'
- ERROR の場合は 6 または'E'
- FATAL の場合は 7 または'F'

level に文字を使用する場合、文字は単一引用符で囲む必要があります。数値は引用符で囲むことができますが、囲む必要はありません。

'*raw-message*'
そのレベルで書き込むメッセージを指定します。

範囲 1-65535 文字

要件 メッセージは文字列であり、単一引用符で囲む必要があります。

ヒント メッセージは、任意の文字型式にすることができます。例を次に示します。x.log(5, 'Error while processing function' || trimn(FNAME));ただし、文字

式を使用すると、CHAR データ型から NCHAR データ型に変換されます。文字列を使用する方が高速です。

argument
message-format の$s 形式マーカーを置き換える値を指定します。

操作 LOG メソッドに 3 つ以上のパラメーターがあり、レベルが有効な場合、*message-format* の各$s 形式マーカーは、対応する *argument* のコンテンツに置き換えられます。

ヒント フォーマットされた出力を使用する場合、余分な引数は無視されます。

オプション引数

message-format
ログメッセージの生成に使用されるメッセージ形式を指定します。メッセージ形式には、少なくとも 1 つの$s 形式マーカーが含まれています。

要件 $s 形式マーカーの数は、引数の数以下である必要があります。それ以外の場合は、エラーが発生します。

操作 LOG メソッドに 3 つ以上のパラメーターがあり、レベルが有効な場合、各$s 形式マーカーは、対応する *argument* のコンテンツに置き換えられます。

ヒント メッセージにドル記号($)を表示するには、$s 形式マーカーで$$を使用します。

詳細

未フォーマットメッセージ(形式 1)

ちょうど 2 つのパラメーター、アクティブなロガー、および有効なレベルを使用した LOG メソッド呼び出しは、指定されたメッセージを未加工の形式で関連するロガーに送信します。

フォーマット済みメッセージ(形式 2)

3 つ以上のパラメーター、アクティブなロガー、および有効なレベルを使用した LOG メソッド呼び出しは、フォーマット済みメッセージを関連するロガーに送信します。*message-format* の各$s 形式マーカーは、対応する *argument* のコンテンツに置き換えられます。

例

例 1: 未フォーマットメッセージ

これらは、未フォーマットメッセージの例です。

```
mylog.log('T', 'The output was written to the log');

testlog.log(7, 'The output could not be written');
```

例 2: フォーマット済みメッセージ

次の例は、$s 形式マーカーのいくつかの組み合わせを示しています。

注: このサンプルコードでは、DS2 ステートメントは PROC DS2 を使用して SAS に送信されます。

```
proc ds2;
  data _NULL_;
  dcl package logger root();
  method init();
    /* $$ is not evaluated here - one parameter */
    root.log(n'note', 'one-parm dollar pair: $$');
    /* $s is not evaluated here - one parameter */
    root.log(n'note', 'one-parm dollar-s: $s');
    /* $$ is evaluated here and "mine" is substituted for $s */
    root.log(n'note', 'dollar pair: $$; me:$s', n'mine');
    /* "mine" is substituted for $s */
    root.log(n'note', 'me:$s', n'mine');
    /* "mine" is substituted for the first $s */
    /* "thine" is substituted for the second $s */
    root.log(n'note', 'me:$s thee:$s', n'mine', n'thine');
    /* there are five arguments */
    root.log(n'note', 'one:$s two:$s three:$s four:$s five:$s', 1N, 2, 3.0, n'4', '5');
  end;
enddata;
run;
quit;
```

次の行が SAS ログに書き込まれます。

```
NOTE: one-parm dollar pair: $$
NOTE: one-parm dollar-s: $s
NOTE: dollar pair: $; me:mine
NOTE: me:mine
NOTE: me:mine thee:thine
NOTE: one:1 two:2 three:3 four:4 five:5
NOTE: Execution succeeded.No rows affected.
```

関連項目

- *SAS プログラムとジョブの SAS Viya ログ機能*
- "ロガーパッケージの使用" (*SAS DS2 プログラマガイド*)

NEW 演算子: ロガーパッケージ

ロガーパッケージのインスタンスを構築します。

注: かっこの前のエスケープ文字(\)は、構文にかっこが必要であることを示します。

構文

package-variable = **_NEW_ LOGGER**([*logger-name*]);

必須引数

package-variable
パッケージのインスタンスを参照できる名前を指定します。

詳細

DS2 パッケージは、特定のインスタンスを構築して他の DS2 プログラムで使用できる変数とメソッドのコレクションです。

DECLARE PACKAGE ステートメントを使用して、ロガーパッケージ変数を宣言します。ロガーパッケージ変数を宣言した後、_NEW_演算子を使用してロガーパッケージのインスタンスをインスタンス化し、新しいパッケージインスタンスをロガーパッケージ変数に割り当てます。

```
declare package logger mylog;
mylog = _new_ logger( );
```

DECLARE PACKAGE と_NEW_演算子を使用してロガーパッケージを宣言およびインスタンス化する 2 ステップのプロセスのかわりに、コンストラクターメソッドとして DECLARE PACKAGE ステートメントを使用することにより、パッケージを 1 ステップで宣言およびインスタンス化できます。これは、DECLARE PACKAGE ステートメントのみを使用した同じ例です。

```
declare package logger mylog( );
```

注: パッケージ変数は、すべての 変数スコーピングルールの対象となります。詳細については、"パッケージ、スコープ、および有効期間" (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- “ロガーパッケージの使用” (*SAS DS2 プログラマガイド*)
- “パッケージのコンストラクターとデストラクター” (*SAS DS2 プログラマガイド*)

ステートメント:

- “DECLARE PACKAGE ステートメント: ロガーパッケージ” (1807 ページ)

20

DS2 行列パッケージのメソッド、演算子、およびステートメント

ディクショナリ **1818**
ABS メソッド 1818
ADD メソッド: 行列パッケージ 1819
ALL_AND メソッド 1821
ALL_EQ メソッド 1822
ALL_GE メソッド 1824
ALL_GT メソッド 1825
ALL_LE メソッド 1826
ALL_LT メソッド 1827
ALL_NE メソッド 1829
ALL_OR メソッド 1830
AND メソッド 1831
ANY_AND メソッド 1832
ANY_EQ メソッド 1833
ANY_GE メソッド 1835
ANY_GT メソッド 1836
ANY_LE メソッド 1837
ANY_LT メソッド 1838
ANY_NE メソッド 1840
ANY_OR メソッド 1841
COLS メソッド 1843
COPY メソッド 1843
DECLARE PACKAGE ステートメント: 行列パッケージ 1845
DELETE メソッド: 行列パッケージ 1847
DET メソッド 1847
EDIV メソッド 1848
EMAX メソッド 1850
EMIN メソッド 1852
EMOD メソッド 1854
EMULT メソッド 1855
EPOW メソッド 1857
EQ メソッド 1859
EXP メソッド 1860
FLOOR メソッド 1862

GE メソッド 1863
GT メソッド 1864
IN メソッド 1865
INVERSE メソッド 1869
LE メソッド 1870
LOG メソッド: 行列パッケージ 1872
LT メソッド 1873
NEW 演算子: 行列パッケージ 1875
MULT メソッド 1877
NE メソッド 1880
OR メソッド 1882
OUT メソッド 1883
ROWS メソッド 1885
SQRT メソッド 1886
SUB メソッド 1887
TOARRAY メソッド 1888
TOVARARRAY メソッド 1890
TRANS メソッド 1891

ディクショナリ

ABS メソッド

入力行列の各値の絶対値を含む行列を返します。

構文

r=*package*.**ABS**( );

必須引数

r
入力行列の各値の絶対値を含む行列を指定します。

package
行列パッケージ変数のインスタンスを指定します。

関連項目

“行列演算” (SAS DS2 プログラマガイド)

ADD メソッド: 行列パッケージ

ある行列を別の行列に加算します。

構文

r=*package-1*.**ADD**(*package-2*);

必須引数

r
ADD メソッドによって自動的に作成される行列を指定します。

package-1
加算演算で使用される最初の行列パッケージ変数のインスタンスを指定します。

package-2
加算演算で使用される 2 番目の行列パッケージ変数のインスタンスを指定します。

詳細

行列の加算を行うには、ADD メソッドの行列の次元が同じサイズである必要があります。最初の行列の各[i, j]要素が、2 番目の行列の対応する[i, j]要素に加算されます。

欠損値を持つ行列を追加しても、エラーは発生しません。

1x1 行列を使用してスカラー加算を実行することもできます。

例

例 1: 2 つの行列の加算

これは、ADD メソッドを使用して 2 つの 3x3 行列を加算する例です。最初の行列の各[i, j]要素が、2 番目の行列の対応する[i, j]要素に加算されます。

```
data _null_;
  dcl double a[3,3];
  dcl double b[3,3];
  dcl double c[3,3];

  method run();
    dcl package matrix m;
    dcl package matrix m2;
    dcl package matrix r;
```

```
    dcl double i j;

    a := (1,2,3,4,5,6,7,8,9);
    b := (1,5,9,2,6,10,3,7,1);

    m = _new_ matrix(a, 3, 3);
    m2 = _new_ matrix(b, 3, 3);

    r = m.add(m2);
    r.toarray(c);

    do i = 1 to 3;
      do j = 1 to 3;
        put c[i,j];
      end;
    end;
  end;
enddata;
run;
```

次の行が SAS ログに書き込まれます。

```
2
7
12
6
11
16
10
15
10
```

例 2: 2 つの行列のスカラー加算

これは、1x1 行列を使用してスカラー加算を実行する方法の例です。

```
data _null_;
  dcl double c[3,3];
  dcl double d[1,1];
  dcl double f[3,3];
  method run();
    dcl package matrix m3;
    dcl package matrix m4;
    dcl package matrix r;
    dcl double i j;

    c := (-0, 0, -1, 1, -2.2, 2.2, -3.3, 4.4, 5.5);
    d := (1);

    m3 = _new_ matrix(c, 3, 3);
    m4 = _new_ matrix(d, 1, 1);

    r = m3.add(m4);
    r.toarray(f);

    do i = 1 to 3;
      do j = 1 to 3;
        put f[i,j];
```

```
      end;
    end;
  end;
enddata;
run;
```

この例では、行列 *m3* の各エントリに 1 が加算されています。スカラー加算により、次の値を持つ 3x3 行列が生成されます。

```
1    1    0
2    -1.2  3.2
-2.3  5.4  6.5
```

関連項目

- *"行列演算" (SAS DS2 プログラマガイド)*

メソッド:

- "SUB メソッド" (1887 ページ)

ALL_AND メソッド

ある行列のすべての要素と別の行列のすべての要素の間の ALL_AND 比較でスカラー結果を生成します。

構文

x=*package-1*.**ALL_AND**(*package-2*);

必須引数

x
スカラー結果を指定します。

package-1
ALL AND 比較で使用される最初の行列パッケージ変数のインスタンスを指定します。

package-2
ALL AND 比較で使用される 2 番目の行列パッケージ変数のインスタンスを指定します。

詳細

ALL_AND 論理演算は、最初の行列の[i, j]番目の要素が 2 番目の行列の[i, j]番目の要素との比較を満たすかどうかを示す結果を生成します。スカラー結果は 0 または 1 です。結果を 1 にするには、すべての論理[i, j]演算が真である必要があります。それ以外の場合、結果は 0 になります。

行列のサイズは一致する必要があります。一致しない場合は、スカラー比較を使用することもできます。

比較

ANY_AND 演算は ALL_AND 演算と似ていますが、論理[i, j]演算のいずれかが真の場合、結果が 1 になる点が異なります。それ以外の場合、結果は 0 になります。

例

```
x=m1.all_and(m2);
```

関連項目

- *“行列演算” (SAS DS2 プログラマガイド)*

メソッド:

- “ALL_OR メソッド” (1830 ページ)
- “AND メソッド” (1831 ページ)
- “ANY_AND メソッド” (1832 ページ)

ALL_EQ メソッド

ある行列の要素と別の行列の要素の間の ALL_EQ (ALL equal-to)比較でスカラー結果を生成します。

構文

x=*package-1*.**ALL_EQ**(*package-2*);

必須引数

x
スカラー結果を指定します。

package-1
ALL equal-to 比較で使用される最初の行列パッケージ変数のインスタンスを指定します。

package-2
ALL equal-to 比較で使用される 2 番目の行列パッケージ変数のインスタンスを指定します。

詳細

ALL_EQ 関係演算は、最初の行列の[i, j]番目の要素が 2 番目の行列の[i, j]番目の要素との比較を満たすかどうかを示すスカラー結果を生成します。スカラー結果は 0 または 1 です。結果を 1 にするには、すべての[i, j]要素の比較が真である必要があります。それ以外の場合、結果は 0 になります。

行列のサイズは一致する必要があります。一致しない場合は、スカラー比較を使用できます。

比較

ANY_EQ 演算は ALL_EQ 演算と似ていますが、論理[i, j]演算のいずれかが真の場合、結果が 1 になる点が異なります。それ以外の場合、結果は 0 になります。

例

```
x=m1.all_eq(m2);
```

関連項目

- “行列演算” (*SAS DS2 プログラマガイド*)

メソッド:

- “ALL_NE メソッド” (1829 ページ)
- “ANY_EQ メソッド” (1833 ページ)

ALL_GE メソッド

ある行列の要素と別の行列の要素の間の ALL_GE (ALL greater-than-or-equal-to)比較でスカラー結果を生成します。

構文

x=*package-1*.**ALL_GE**(*package-2*);

必須引数

x
スカラー結果を指定します。

package-1
ALL greater-than-or-equal-to 比較で使用される最初の行列パッケージ変数のインスタンスを指定します。

package-2
ALL greater-than-or-equal-to 比較で使用される 2 番目の行列パッケージ変数のインスタンスを指定します。

詳細

ALL_GE 関係演算は、最初の行列の[i, j]番目の要素が 2 番目の行列の[i, j]番目の要素との比較を満たすかどうかを示すスカラー結果を生成します。スカラー結果は 0 または 1 です。結果を 1 にするには、すべての[i, j]要素の比較が真である必要があります。それ以外の場合、結果は 0 になります。

行列のサイズは一致する必要があります。一致しない場合は、スカラー比較を使用できます。

比較

ANY_GE 演算は ALL_GE 演算と似ていますが、論理[i, j]演算のいずれかが真の場合、結果が 1 になる点が異なります。それ以外の場合、結果は 0 になります。

例

```
x=m1.all_ge(m2);
```

関連項目

- “行列演算” (*SAS DS2 プログラマガイド*)

メソッド:

- “ALL_LE メソッド” (1826 ページ)
- “ALL_NE メソッド” (1829 ページ)
- “ANY_GE メソッド” (1835 ページ)

ALL_GT メソッド

ある行列の要素と別の行列の要素の間の ALL_GT (ALL greater-than)比較でスカラー結果を生成します。

構文

x=*package-1*.**ALL_GT**(*package-2*);

必須引数

x
スカラー結果を指定します。

package-1
ALL greater-than 比較で使用される最初の行列パッケージ変数のインスタンスを指定します。

package-2
ALL greater-than 比較で使用される 2 番目の行列パッケージ変数のインスタンスを指定します。

詳細

ALL_GE 関係演算は、最初の行列の[i, j]番目の要素が 2 番目の行列の [i, j]番目の要素との比較を満たすかどうかを示すスカラー結果を生成します。スカラー結果は 0 または 1 です。結果を 1 にするには、すべての[i, j]要素の比較が真である必要があります。それ以外の場合、結果は 0 になります。

行列のサイズは一致する必要があります。一致しない場合は、スカラー比較を使用できます。

比較

ANY_GT 演算は ALL_GT 演算と似ていますが、論理[i, j]演算のいずれかが真の場合、結果が 1 になる点が異なります。それ以外の場合、結果は 0 になります。

例

```
x=m1.all_gt(m2);
```

関連項目

- *“行列演算” (SAS DS2 プログラマガイド)*

メソッド:

- “ALL_GE メソッド” (1824 ページ)
- “ALL_LT メソッド” (1827 ページ)
- “ANY_GT メソッド” (1836 ページ)

ALL_LE メソッド

ある行列の要素と別の行列の要素の間の ALL_LE (ALL less-than-or-equal-to)比較でスカラー結果を生成します。

構文

x=*package-1*.**ALL_LE**(*package-2*);

必須引数

x
スカラー結果を指定します。

package-1
ALL less-than 比較で使用される最初の行列パッケージ変数のインスタンスを指定します。

package-2
ALL less-than 比較で使用される 2 番目の行列パッケージ変数のインスタンスを指定します。

詳細

ALL_LE 関係演算は、最初の行列の[i, j]番目の要素が 2 番目の行列の[i, j]番目の要素との比較を満たすかどうかを示すスカラー結果を生成します。スカラー結果は 0 または 1 です。結果を 1 にするには、すべての[i, j]要素の比較が真である必要があります。それ以外の場合、結果は 0 になります。

行列のサイズは一致する必要があります。一致しない場合は、スカラー比較を使用できます。

比較

ANY_LE 演算は ALL_LE 演算と似ていますが、論理[i, j]演算のいずれかが真の場合、結果が 1 になる点が異なります。それ以外の場合、結果は 0 になります。

例

```
x=m1.all_le(m2);
```

関連項目

- *“行列演算” (SAS DS2 プログラマガイド)*

メソッド:

- “ALL_GE メソッド” (1824 ページ)
- “ALL_LT メソッド” (1827 ページ)
- “ANY_LE メソッド” (1837 ページ)

ALL_LT メソッド

ある行列の要素と別の行列の要素の間の ALL_LT (ALL less-than)比較でスカラー結果を生成します。

構文

x=*package-1*.**ALL_LT**(*package-2*);

必須引数

x
スカラー結果を指定します。

package-1
ALL less-than 比較で使用される最初の行列パッケージ変数のインスタンスを指定します。

package-2
ALL less-than 比較で使用される 2 番目の行列パッケージ変数のインスタンスを指定します。

詳細

ALL_LT 関係演算は、最初の行列の[i, j]番目の要素が 2 番目の行列の[i, j]番目の要素との比較を満たすかどうかを示すスカラー結果を生成します。スカラー結果は 0 または 1 です。結果を 1 にするには、すべての[i, j]要素の比較が真である必要があります。それ以外の場合、結果は 0 になります。

行列のサイズは一致する必要があります。一致しない場合は、スカラー比較を使用できます。

比較

ANY_LT 演算は ALL_LT 演算と似ていますが、論理[i, j]演算のいずれかが真の場合、結果が 1 になる点が異なります。それ以外の場合、結果は 0 になります。

例

```
x=m1.all_lt(m2);
```

関連項目

- “行列演算” (*SAS DS2 プログラマガイド*)

メソッド:

- “ALL_GT メソッド” (1825 ページ)
- “ALL_LE メソッド” (1826 ページ)
- “ANY_LT メソッド” (1838 ページ)

ALL_NE メソッド

ある行列の要素と別の行列の要素の間の ALL_NE (ALL not-equal-to)比較でスカラー結果を生成します。

構文

x=*package-1*.**ALL_NE**(*package-2*);

必須引数

x
スカラー結果を指定します。

package-1
ALL not-equal-to 比較で使用される最初の行列パッケージ変数のインスタンスを指定します。

package-2
ALL not-equal-to 比較で使用される 2 番目の行列パッケージ変数のインスタンスを指定します。

詳細

ALL_NE 関係演算は、最初の行列の[i, j]番目の要素が 2 番目の行列の[i, j]番目の要素との比較を満たすかどうかを示すスカラー結果を生成します。スカラー結果は 0 または 1 です。結果を 1 にするには、すべての[i, j]要素の比較が真である必要があります。それ以外の場合、結果は 0 になります。

行列のサイズは一致する必要があります。一致しない場合は、スカラー比較を使用できます。

比較

ANY_NE 演算は ALL_NE 演算と似ていますが、論理[i, j]演算のいずれかが真の場合、結果が 1 になる点が異なります。それ以外の場合、結果は 0 になります。

例

```
x=m1.all_ne(m2);
```

関連項目

- “行列演算” (*SAS DS2 プログラマガイド*)

メソッド:

- “ALL_EQ メソッド” (1822 ページ)
- “ANY_NE メソッド” (1840 ページ)

ALL_OR メソッド

ある行列の要素と別の行列の要素の間の ALL_OR 比較でスカラー結果を生成します。

構文

x=*package-1*.**ALL_OR**(*package-2*);

必須引数

x
スカラー結果を指定します。

package-1
ALL OR 比較で使用される最初の行列パッケージ変数のインスタンスを指定します。

package-2
ALL OR 比較で使用される 2 番目の行列パッケージ変数のインスタンスを指定します。

詳細

ALL_OR 論理演算は、最初の行列の[i, j]番目の要素が 2 番目の行列の [i, j]番目の要素との比較を満たすかどうかを示す結果を生成します。スカラー結果は 0 または 1 です。結果を 1 にするには、すべての論理[i, j]演算が真である必要があります。それ以外の場合、結果は 0 になります。

行列のサイズは一致する必要があります。一致しない場合は、スカラー比較を使用できます。

比較

ANY_OR 演算は ALL_OR 演算と似ていますが、論理[i, j]演算のいずれかが真の場合、結果が 1 になる点が異なります。それ以外の場合、結果は 0 になります。

例

```
x=m1.all_or(m2);
```

関連項目

- *"行列演算" (SAS DS2 プログラマガイド)*

メソッド:

- "ALL_AND メソッド" (1821 ページ)
- "ANY_OR メソッド" (1841 ページ)
- "OR メソッド" (1882 ページ)

AND メソッド

AND 論理演算に基づいて 2 つの行列を比較し、結果の行列を返します。

構文

r=*package-1*.**AND**(*package-2*);

必須引数

r
2 つの行列の値間の AND 比較の結果を含む行列を指定します。

package-1
AND 比較で使用される最初の行列パッケージ変数のインスタンスを指定します。

package-2
AND 比較で使用される 2 番目の行列パッケージ変数のインスタンスを指定します。

詳細

AND 論理演算子は、二項関係演算(LT、LE、GE、GT、NE、EQ)と同様に動作します。いずれの場合も、AND 論理演算は 2 つの行列の[i, j]番目の要素に適用され、結果の行列 *r* に配置されます。

行列のサイズは一致する必要があります。一致しない場合は、スカラー比較を使用できます。

例

```
r=m1.and(m2);
```

関連項目

- *"行列演算" (SAS DS2 プログラマガイド)*

メソッド:

- "ALL_AND メソッド" (1821 ページ)
- "ANY_AND メソッド" (1832 ページ)
- "OR メソッド" (1882 ページ)

ANY_AND メソッド

ある行列の要素と別の行列の要素の間の ANY_AND 比較でスカラー結果を生成します。

構文

x=*package-1*.**ANY_AND**(*package-2*);

必須引数

x
スカラー結果を指定します。

package-1
ANY AND 比較で使用される最初の行列パッケージ変数のインスタンスを指定します。

package-2
ANY AND 比較で使用される 2 番目の行列パッケージ変数のインスタンスを指定します。

詳細

ANY_AND 論理演算は、最初の行列の[i, j]番目の要素が 2 番目の行列の[i, j]番目の要素との比較を満たすかどうかを示すスカラー結果を生成します。スカラー結果は 0 または 1 です。論理[i, j]演算のいずれかが真の場合、結果は 1 になります。それ以外の場合、結果は 0 になります。

行列のサイズは一致する必要があります。一致しない場合は、スカラー比較を使用できます。

比較

ALL_AND 演算は ANY_AND 演算と似ていますが、結果が 1 になるためには、すべての論理[i, j]演算が真になる必要がある点が異なります。それ以外の場合、結果は 0 になります。

例

```
x=m1.any_and(m2);
```

関連項目

- “行列演算” (*SAS DS2 プログラマガイド*)

メソッド:

- “ALL_OR メソッド” (1830 ページ)
- “AND メソッド” (1831 ページ)
- “ANY_OR メソッド” (1841 ページ)

ANY_EQ メソッド

ある行列の要素と別の行列の要素の間の ANY_EQ (ANY equal-to)比較でスカラー結果を生成します。

構文

x=*package-1*.**ANY_EQ**(*package-2*);

必須引数

x
スカラー結果を指定します。

package-1
ANY equal-to 比較で使用される最初の行列パッケージ変数のインスタンスを指定します。

package-2
ANY equal-to 比較で使用される 2 番目の行列パッケージ変数のインスタンスを指定します。

詳細

ANY_EQ 関係演算は、最初の行列の[i, j]番目の要素が 2 番目の行列の[i, j]番目の要素との比較を満たすかどうかを示すスカラー結果を生成します。スカラー結果は 0 または 1 です。[i, j]要素の比較のいずれかが真の場合、結果は 1 になります。それ以外の場合、結果は 0 になります。

行列のサイズは一致する必要があります。一致しない場合は、スカラー比較を使用できます。

比較

ALL_EQ 演算は ANY_EQ 演算と似ていますが、結果が 1 になるためには、すべての論理[i, j]演算が真になる必要がある点が異なります。それ以外の場合、結果は 0 になります。

例

```
x=m1.any_eq(m2);
```

関連項目

- “行列演算” (*SAS DS2 プログラマガイド*)

メソッド:

- “ALL_EQ メソッド” (1822 ページ)

- “ANY_NE メソッド” (1840 ページ)
- “EQ メソッド” (1859 ページ)

ANY_GE メソッド

ある行列の要素と別の行列の要素の間の ANY_GE (ANY greater-than-or-equal-to)比較でスカラー結果を生成します。

構文

x=*package-1*.**ANY_GE**(*package-2*);

必須引数

x
スカラー結果を指定します。

package-1
ANY greater-than-or-equal-to 比較で使用される最初の行列パッケージ変数のインスタンスを指定します。

package-2
ANY greater-than-or-equal-to 比較で使用される 2 番目の行列パッケージ変数のインスタンスを指定します。

詳細

ANY_GE 関係演算は、最初の行列の[i, j]番目の要素が 2 番目の行列の[i, j]番目の要素との比較を満たすかどうかを示すスカラー結果を生成します。スカラー結果は 0 または 1 です。[i, j]要素の比較のいずれかが真の場合、結果は 1 になります。それ以外の場合、結果は 0 になります。

行列のサイズは一致する必要があります。一致しない場合は、スカラー比較を使用できます。

比較

ALL_GE 演算は ANY_GE 演算と似ていますが、結果が 1 になるためには、すべての論理[i, j]演算が真になる必要がある点が異なります。それ以外の場合､結果は 0 になります。

例

```
x=m1.any_ge(m2);
```

関連項目

- *“行列演算” (SAS DS2 プログラマガイド)*

メソッド:

- “ALL_GE メソッド” (1824 ページ)
- “ANY_LE メソッド” (1837 ページ)
- “ANY_NE メソッド” (1840 ページ)

ANY_GT メソッド

ある行列の要素と別の行列の要素の間の ANY_GT (ANY greater-than)比較でスカラー結果を生成します。

構文

x=*package-1*.**ANY_GT**(*package-2*);

必須引数

x
スカラー結果を指定します。

package-1
ANY greater-than 比較で使用される最初の行列パッケージ変数のインスタンスを指定します。

package-2
ANY greater-than 比較で使用される 2 番目の行列パッケージ変数のインスタンスを指定します。

詳細

ANY_GT 関係演算は、最初の行列の[i, j]番目の要素が 2 番目の行列の[i, j]番目の要素との比較を満たすかどうかを示すスカラー結果を生成します。スカラー結果は 0 または 1 です。[i, j]要素の比較のいずれかが真の場合、結果は 1 になります。それ以外の場合、結果は 0 になります。

行列のサイズは一致する必要があります。一致しない場合は、スカラー比較を使用できます。

比較

ALL_GT 演算は ANY_GT 演算と似ていますが、結果が 1 になるためには、すべての論理[i, j]演算が真になる必要がある点が異なります。それ以外の場合、結果は 0 になります。

例

```
x=m1.any_gt(m2);
```

関連項目

- *“行列演算” (SAS DS2 プログラマガイド)*

メソッド:

- “ALL_GT メソッド” (1825 ページ)
- “ANY_GE メソッド” (1835 ページ)
- “ANY_LT メソッド” (1838 ページ)

ANY_LE メソッド

ある行列の要素と別の行列の要素の間の ANY_LE (any less-than-or-equal-to)比較でスカラー結果を生成します。

構文

x=*package-1*.**ANY_LE**(*package-2*);

必須引数

x
スカラー結果を指定します。

package-1
ANY less-than 比較で使用される最初の行列パッケージ変数のインスタンスを指定します。

package-2
ANY less-than 比較で使用される 2 番目の行列パッケージ変数のインスタンスを指定します。

詳細

ANY_LE 関係演算は、最初の行列の[i, j]番目の要素が 2 番目の行列の[i, j]番目の要素との比較を満たすかどうかを示すスカラー結果を生成します。スカラー結果は 0 または 1 です。[i, j]要素の比較のいずれかが真の場合、結果は 1 になります。それ以外の場合、結果は 0 になります。

行列のサイズは一致する必要があります。一致しない場合は、スカラー比較を使用できます。

比較

ALL_LE 演算は ANY_LE 演算と似ていますが、結果が 1 になるためには、すべての論理[i, j]演算が真になる必要がある点が異なります。それ以外の場合、結果は 0 になります。

例

```
x=m1.any_le(m2);
```

関連項目

- “行列演算” (*SAS DS2 プログラマガイド*)

メソッド:

- “ALL_LE メソッド” (1826 ページ)
- “ANY_GE メソッド” (1835 ページ)
- “ANY_LT メソッド” (1838 ページ)

ANY_LT メソッド

ある行列の要素と別の行列の要素の間の ANY_LT (ANY less-than)比較でスカラー結果を生成します。

構文

x=*package-1*.**ANY_LT**(*package-2*);

必須引数

x
スカラー結果を指定します。

package-1
ANY less-than 比較で使用される最初の行列パッケージ変数のインスタンスを指定します。

package-2
ANY less-than 比較で使用される 2 番目の行列パッケージ変数のインスタンスを指定します。

詳細

ANY_LT 関係演算は、最初の行列の[i, j]番目の要素が 2 番目の行列の[i, j]番目の要素との比較を満たすかどうかを示すスカラー結果を生成します。スカラー結果は 0 または 1 です。[i, j]要素の比較のいずれかが真の場合、結果は 1 になります。それ以外の場合、結果は 0 になります。

行列のサイズは一致する必要があります。一致しない場合は、スカラー比較を使用できます。

比較

ALL_LT 演算は ANY_LT 演算と似ていますが、結果が 1 になるためには、すべての論理[i, j]演算が真になる必要がある点が異なります。それ以外の場合、結果は 0 になります。

例

次の例では、行列 *m* (= =2)の[1, 2]要素が行列 *m2* (= =5)の[1, 2]要素よりも小さいため、結果は 1 になります。他のすべての要素は、比較を満たしていません。たとえば、行列 *m* の[1, 2]要素が 6 に変更された場合、結果は 0 になります。

```
data _null_;
  dcl double a[3,3];
  dcl double b[3,3];
  dcl double r;

  method run();
    dcl package matrix m;
    dcl package matrix m2;
```

```
    a := (1,2,10,4,7,11,15,9,12);
    b := (1,5,9,2,6,10,3,7,11);

    m = _new_ matrix(a, 3, 3);
    m2 = _new_ matrix(b, 3, 3);

    r = m.any_lt(m2);
    put r=;
  end;
enddata;
run;
```

関連項目

- “行列演算” (*SAS DS2 プログラマガイド*)

メソッド:

- “ALL_LT メソッド” (1827 ページ)
- “ANY_GT メソッド” (1836 ページ)
- “ANY_LE メソッド” (1837 ページ)

ANY_NE メソッド

ある行列の要素と別の行列の要素の間の ANY_NE (ANY not-equal-to)比較でスカラー結果を生成します。

構文

x=*package-1*.**ANY_NE**(*package-2*);

必須引数

x
スカラー結果を指定します。

package-1
ANY not-equal-to 比較で使用される最初の行列パッケージ変数のインスタンスを指定します。

package-2
ANY not-equal-to 比較で使用される 2 番目の行列パッケージ変数のインスタンスを指定します。

詳細

ANY_NE 関係演算は、最初の行列の[i, j]番目の要素が 2 番目の行列の[i, j]番目の要素との比較を満たすかどうかを示すスカラー結果を生成します。スカラー結果は 0 または 1 です。[i, j]要素の比較のいずれかが真の場合、結果は 1 になります。それ以外の場合、結果は 0 になります。

行列のサイズは一致する必要があります。一致しない場合は、スカラー比較を使用できます。

比較

ALL_GE 演算は ANY_GE 演算と似ていますが、結果が 1 になるためには、すべての論理[i, j]演算が真になる必要がある点が異なります。それ以外の場合、結果は 0 になります。

例

```
x=m1.any_ne(m2);
```

関連項目

- *“行列演算” (SAS DS2 プログラマガイド)*

メソッド:

- “ALL_NE メソッド” (1829 ページ)
- “ANY_EQ メソッド” (1833 ページ)

ANY_OR メソッド

ある行列の要素と別の行列の要素の間の ANY_OR 比較でスカラー結果を生成します。

構文

x=*package-1*.**ANY_OR**(*package-2*);

必須引数

x
スカラー結果を指定します。

package-1
ANY OR 比較で使用される最初の行列パッケージ変数のインスタンスを指定します。

package-2
ANY OR 比較で使用される 2 番目の行列パッケージ変数のインスタンスを指定します。

詳細

ANY_OR 論理演算は、最初の行列の[i, j]番目の要素が 2 番目の行列の[i, j]番目の要素との比較を満たすかどうかを示すスカラー結果を生成します。スカラー結果は 0 または 1 です。[i, j]要素の比較のいずれかが真の場合、結果は 1 になります。それ以外の場合、結果は 0 になります。

行列のサイズは一致する必要があります。一致しない場合は、スカラー比較を使用できます。

比較

ALL_OR 演算は ANY_OR 演算と似ていますが、結果が 1 になるためには、すべての論理[i, j]演算が真になる必要がある点が異なります。それ以外の場合、結果は 0 になります。

例

```
x=m1.any_or(m2);
```

関連項目

- “行列演算” (*SAS DS2 プログラマガイド*)

メソッド:

- “ALL_OR メソッド” (1830 ページ)
- “ANY_AND メソッド” (1832 ページ)
- “OR メソッド” (1882 ページ)

COLS メソッド

指定された行列の列数を返します。

構文

variable-name=*package*.**COLS**( );

必須引数

variable-name
メソッドの完了後の列数を含む変数の名前を指定します。

package
行列パッケージ変数のインスタンスを指定します。

例

ROWS メソッドの例を参照してください (1885 ページ)。

関連項目

- “MATRIX パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “ROWS メソッド” (1885 ページ)

COPY メソッド

ある行列を別の行列にコピーします。

構文

r=*package*.**COPY**( );

必須引数

r
COPY メソッドによって自動的に作成される行列を指定します。

package
行列パッケージ変数のインスタンスを指定します。

詳細

COPY メソッドは、行列を新しい行列にコピーします。

例

この例では、3x4 行列の新しいコピーを作成します。

```
data _null_;
  dcl double a[3,4];
  dcl double b[3,4];

  method run();
    dcl package matrix m;
    dcl package matrix r;
    dcl double i j;

    a := (1,2,3,4,5,6,7,8,9,10,11,12);

    m = _new_ matrix(a, 3, 4);
    r = m.copy();
    r.toarray(b);

    do i = 1 to 3;
      do j = 1 to 4;
        put b[i,j];
      end;
    end;
  end;
enddata;
run;
```

次の行が SAS ログに書き込まれます。

```
1
2
3
4
5
6
7
8
9
10
11
12
```

関連項目

"MATRIX パッケージの使用" (*SAS DS2 プログラマガイド*)

DECLARE PACKAGE ステートメント: 行列パッケージ

MATRIX パッケージのインスタンスを作成します。

カテゴリ: ローカル

構文

DECLARE PACKAGE MATRIX *variable* ([*row-dimension*, *column-dimension*]);

必須引数

variable
行列パッケージのインスタンスを参照できる名前を指定します。

オプション引数

row-dimension
行列の行の次元を指定します。

column-dimension
行列の列の次元を指定します。

詳細

DS2 パッケージは、特定のインスタンスを構築して他の DS2 プログラムで使用できる変数とメソッドのコレクションです。

行列パッケージは、SAS/IML 機能の DS2 レベルの実装を提供します。行列パッケージは、DS2 プログラム用に事前定義されています。

パッケージが宣言されると、パッケージのインスタンスを参照できる変数が作成されます。コンストラクター引数がパッケージ変数宣言で提供されている場合、パッケージインスタンスが構築され、構築されたパッケージインスタンスを参照するようにパッケージ変数が設定されます。複数のパッケージ変数を作成でき、単一の DECLARE PACKAGE ステートメントで複数のパッケージインスタンスを構築できます。各パッケージインスタンスは、パッケージの完全に別個のコピーを表します。

行列パッケージは、DECLARE PACKAGE ステートメントを使用してパッケージを宣言およびインスタンス化することによって作成されます。

この例では、空の 2x2 行列を作成し、インスタンスを変数 *m* に保存します。

```
declare package matrix m(2, 2);
```

行列は使用する前に初期化する必要があり、初期化は宣言ではなくコードストリームで行われます。次のアクションを使用して、データを行列インスタンスにロードできます。

- 配列をロードする_NEW_演算子
- 配列をロードする IN メソッド
- 外部データをロードする SET ステートメント

注: パッケージ変数は、すべての変数スコーピングルールの対象となります。詳細については、“パッケージ、スコープ、および有効期間” (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- “MATRIX パッケージの宣言とインスタンス化” (*SAS DS2 プログラマガイド*)
- “パッケージのコンストラクターとデストラクター” (*SAS DS2 プログラマガイド*)

メソッド:

- “IN メソッド” (1865 ページ)

演算子:

- “_NEW_ 演算子: 行列パッケージ” (1875 ページ)

ステートメント:

- “SET ステートメント” (1416 ページ)

DELETE メソッド: 行列パッケージ

行列パッケージを削除します。

注: DELETE メソッドは必須ではありません。行列パッケージインスタンスを参照する行列パッケージ変数がない場合、パッケージインスタンスは自動的に削除されます。

構文

package.**DELETE**( );

必須引数

package
行列パッケージ変数のインスタンスを指定します。

詳細

行列パッケージが不要になった場合は、DELETE メソッドを使用して削除します。行列パッケージを削除した後に使用しようとすると、ログにエラーが書き込まれます。

関連項目

"パッケージのコンストラクターとデストラクター" (SAS DS2 プログラマガイド)

DET メソッド

正方行列の行列式を計算します。

構文

d=*package*.**DET**( );

必須引数

d
DET メソッドによって自動的に作成される行列を指定します。

package
行列パッケージ変数のインスタンスを指定します。

詳細

行列式の入力行列は正方形でなければなりません。そうでない場合は、ランタイムエラーが発生します。DET メソッドからの出力は、行列式と呼ばれる実数または複素数です。

例

次の例では、3x3 行列の行列式を計算します。

```
data _null_;
  dcl double a[3,3];

  method run();
    dcl package matrix m;
    dcl double d;

    a := (1,3,2,5,4,6,9,8,9);
    m = _new_ matrix(a, 3, 3);
    d = m.det();
    put d=;
  end;
enddata;
run;
```

次の行が SAS ログに書き込まれます。

```
d=23
```

関連項目

"行列演算" (SAS DS2 プログラマガイド)

EDIV メソッド

要素ごとのスカラー除算を実行します。

構文

x=*package-1*.**EDIV**(*package-2*);

必須引数

x
EDIV メソッドによって自動的に作成される行列を指定します。

package-1
要素ごとの除算演算で使用される最初の行列パッケージ変数のインスタンスを指定します。

package-2
要素ごとの除算演算で使用される 2 番目の行列パッケージ変数のインスタンスを指定します。

詳細

EDIV メソッドを使用すると、別の行列を使用して、ある行列の要素ごとのスカラー除算を適用できます。2 番目の行列は、次のいずれかになります。

- 同じ次元の行列
- 行の次元が最初の行列の行の次元と一致するベクトル
- 列の次元が最初の行列の列の次元と一致するベクトル
- 各[i,j]要素でスカラー演算を効果的に可能にする 1x1 行列

EDIV メソッドは、2 つの引数行列に対する要素ごとの演算から結果行列を生成します。

例

```
x=m1.ediv(m2);
```

関連項目

- “行列演算” (*SAS DS2 プログラマガイド*)

メソッド:

- “EMULT メソッド” (1855 ページ)

EMAX メソッド

2 つの行列の要素ごとの比較を実行し、最大の要素を返します。

構文

x=*package-1*.**EMAX**(*package-2*);

必須引数

x
EMAX メソッドによって自動的に作成される行列を指定します。

package-1
要素ごとの最大演算で使用される最初の行列パッケージ変数のインスタンスを指定します。

package-2
要素ごとの最大演算で使用される 2 番目の行列パッケージ変数のインスタンスを指定します。

詳細

EMAX メソッドを使用すると、ある行列に対して、別の行列を使用して要素ごとの最大値比較を適用できます。2 番目の行列は、次のいずれかになります。

- 同じ次元の行列
- 行の次元が最初の行列の行の次元と一致するベクトル
- 列の次元が最初の行列の列の次元と一致するベクトル
- 各[i,j]要素でスカラー演算を効果的に可能にする 1x1 行列

EMAX メソッドは、2 つの引数行列に対する要素ごとの演算から結果行列を生成します。

例

例 1: 1x1 行列を使用した最大値の比較

次の例では、2 つの 2x2 行列からの最大値を含む行列を作成します。

```
data _null_;
  dcl double a[2,2];
  dcl double b[2,2];
```

```
  dcl double f[2,2];

  method run();
    dcl package matrix m;
    dcl package matrix m1;
      dcl package matrix r;
      dcl double i j;

    a := (2, 2, 3, 4);
      b := (4, 5, 1, 0);

    m = _new_ matrix(a, 2, 2);
      m1 = _new_ matrix(b, 2, 2);

    r = m.emax(m1);
      r.toarray(f);

       do i = 1 to 2;
       do j = 1 to 2;
         put f[i,j];
       end;
    end;
  end;
enddata;
run;
```

結果の行列の値は、次のとおりです。

```
4  5
3  4
```

例 2: 行列のベクトル演算

この例では、行列 **m1** を行として使用することにより、最大演算子が行列 **m** のすべての行に適用されます。

```
data _null_;
  method init();
    dcl package matrix m;
    dcl package matrix m1;
    dcl package matrix r;
    dcl double i j;
    dcl double a[4];
    dcl double b[2];
    dcl double f[2,2];

    a := (2, 2, 3, 4);
    b := (1, 5);
    m = _new_ matrix(a, 2, 2);
    m1 = _new_ matrix(b, 1, 2);
    r = m.emax(m1);
    r.toarray(f);
    do i = 1 to 2;
      do j = 1 to 2;
        put f[i,j];
      end;
    end;
```

```
  end;
enddata;
run;
```

結果の行列の値は、次のとおりです。

```
2
5
3
5
```

関連項目

- *“行列演算” (SAS DS2 プログラマガイド)*

メソッド:

- “EMIN メソッド” (1852 ページ)

EMIN メソッド

2 つの行列の要素ごとの比較を実行し、最小の要素を返します。

構文

x=*package-1*.**EMIN**(*package-2*);

必須引数

x
EMIN メソッドによって自動的に作成される行列を指定します。

package-1
要素ごとの最小演算で使用される最初の行列パッケージ変数のインスタンスを指定します。

package-2
要素ごとの最小演算で使用される 2 番目の行列パッケージ変数のインスタンスを指定します。

詳細

EMIN メソッドを使用すると、別の行列を使用して、ある行列の要素ごとの最小値比較を適用できます。2 番目の行列は、次のいずれかになります。

- 同じ次元の行列

- 行の次元が最初の行列の行の次元と一致するベクトル
- 列の次元が最初の行列の列の次元と一致するベクトル
- 各[i,j]要素でスカラー演算を効果的に可能にする 1x1 行列

EMIN メソッドは、2 つの引数行列に対する要素ごとの演算から結果行列を生成します。

例

次の例では、2 つの 2x2 行列からの最大値を含む行列を作成します。

```
data _null_;
  dcl double a[2,2];
  dcl double b[2,2];
  dcl double f[2,2];

  method run();
    dcl package matrix m;
    dcl package matrix m1;
      dcl package matrix r;
      dcl double i j;

    a := (2, 2, 3, 4);
     b := (4, 5, 1, 0);

    m = _new_ matrix(a, 2, 2);
     m1 = _new_ matrix(b, 2, 2);

    r = m.emin(m1);
      r.toarray(f);

     do i = 1 to 2;
      do j = 1 to 2;
        put f[i,j];
      end;
    end;
  end;
enddata;
run;
```

結果の行列の値は、次のとおりです。

```
2  2
1  0
```

関連項目

- *“行列演算” (SAS DS2 プログラマガイド)*

メソッド:

- “EMAX メソッド” (1850 ページ)

EMOD メソッド

要素ごとのスカラー演算で、最初の行列の要素を 2 番目の行列の要素で除算した余りを返します。

構文

x=*package-1*.**EMOD**(*package-2*);

必須引数

x
EMOD メソッドによって自動的に作成される行列を指定します。

package-1
要素ごとの MOD 比較で使用される最初の行列パッケージ変数のインスタンスを指定します。

package-2
要素ごとの MOD 比較で使用される 2 番目の行列パッケージ変数のインスタンスを指定します。

詳細

EMOD の要素ごとの演算を使用すると、別の行列を使用して、ある行列の除算演算の余りを求めることができます。2 番目の行列は、次のいずれかになります。

- 同じ次元の行列
- 行の次元が最初の行列の行の次元と一致するベクトル
- 列の次元が最初の行列の列の次元と一致するベクトル
- 各[i,j]要素でスカラー演算を効果的に可能にする 1x1 行列

EMOD メソッドは、2 つの引数行列に対する要素ごとの演算から結果行列を生成します。

例

次の例では、行列 **m** の要素を行列 **m2** の要素で除算します。EMOD メソッドは、余りの行列 **f** を返すために使用されます。

```
data _null_;
  dcl double a[2,2];
  dcl double b[2,2];
```

```
  dcl double f[2,2];

  method run();
    dcl package matrix m;
    dcl package matrix m1;
      dcl package matrix r;
      dcl double i j;

    a := (125, 17, 39, 40);
      b := (40, 5, 12, 8);

    m = _new_ matrix(a, 2, 2);
      m1 = _new_ matrix(b, 2, 2);

    r = m.emod(m1);
      r.toarray(f);

      do i = 1 to 2;
       do j = 1 to 2;
          put f[i,j];
       end;
    end;
  end;
enddata;
run;
```

結果の行列の値は、次のとおりです。

```
5  2
3  0
```

関連項目

"行列演算" (SAS DS2 プログラマガイド)

EMULT メソッド

要素ごとのスカラー乗算を実行します。

構文

r=*package-1*.**EMULT**(*package-2*);

必須引数

r
EMULT メソッドによって自動的に作成される行列を指定します。

package-1
要素ごとの乗算演算で使用される最初の行列パッケージ変数のインスタンスを指定します。

package-2
要素ごとの乗算演算で使用される 2 番目の行列パッケージ変数のインスタンスを指定します。

詳細

EMULT メソッドを使用すると、別の行列を使用して、ある行列に要素ごとのスカラー乗算を適用できます。2 番目の行列は、次のいずれかになります。

- 同じ次元の行列
- 行の次元が最初の行列の行の次元と一致するベクトル
- 列の次元が最初の行列の列の次元と一致するベクトル
- 各[i,j]要素でスカラー演算を効果的に可能にする 1x1 行列

EMULT メソッドは、2 つの引数行列に対する要素ごとの演算から結果行列を生成します。

例

次の例は、要素ごとのスカラー乗算を実行する方法を示しています。行列 *c* の各要素は 2 倍されます。

```
data _null_;
  dcl double c[3,3];
  dcl double d[1,1];
  dcl double f[3,3];

  method run();
    dcl package matrix m3;
    dcl package matrix m4;
    dcl package matrix r;
     dcl double i j;

    c := (-0, 0, -1, 1, -2.2, 2.2, -3.3, 4.4, 5.5);
    d := (2);

    m3 = _new_ matrix(c, 3, 3);
    m4 = _new_ matrix(d, 1, 1);

    r = m3.emult(m4);
    r.toarray(f);

    do i = 1 to 3;
      do j = 1 to 3;
        put f[i,j];
      end;
```

```
    end;
  end;
enddata;
run;
```

結果の行列の値は、次のとおりです。

```
0    0    -2
2    -4.4  4.4
-6.6  8.8  11
```

関連項目

- *“行列演算” (SAS DS2 プログラマガイド)*

メソッド:

- “EDIV メソッド” (1848 ページ)

EPOW メソッド

要素ごとの演算で、数値を指定された累乗にします。

構文

x=*package-1*.**EPOW**(*package-2*);

必須引数

x
EPOW メソッドによって自動的に作成される行列を指定します。

package-1
要素ごとの演算で使用される最初の行列パッケージ変数のインスタンスを指定します。

package-2
要素ごとの演算で使用される 2 番目の行列パッケージ変数のインスタンスを指定します。

詳細

EPOW の要素ごとの演算を使用すると、別の行列を使用して、ある行列の値を指数関数的に上げることができます。2 番目の行列は、次のいずれかになります。

- 同じ次元の行列

- 行の次元が最初の行列の行の次元と一致するベクトル
- 列の次元が最初の行列の列の次元と一致するベクトル
- 各[i,j]要素でスカラー演算を効果的に可能にする 1x1 行列

EPOW メソッドは、2 つの引数行列に対する要素ごとの演算から結果行列を生成します。

例

次の例では、行列 *c* の各要素を 2 の累乗にします。

```
data _null_;
  dcl double c[3,3];
  dcl double d[1,1];
  dcl double f[3,3];

  method run();
    dcl package matrix m3;
    dcl package matrix m4;
    dcl package matrix r;
     dcl double i j;

    c := (0, 15, 1.7, 13, -2.2, 10, -3.3, 7, 2);
    d := (2);

    m3 = _new_ matrix(c, 3, 3);
    m4 = _new_ matrix(d, 1, 1);

    r = m3.epow(m4);
    r.toarray(f);

    do i = 1 to 3;
      do j = 1 to 3;
        put f[i,j];
      end;
    end;
  end;
enddata;
run;
```

結果の行列の値は、次のとおりです。

```
0    225   2.89
169  4.84  100
10.89 49   4
```

関連項目

“行列演算” (SAS DS2 プログラマガイド)

EQ メソッド

ある行列の要素と別の行列の要素の間の equal-to 比較でスカラー結果を生成します。

構文

r=*package-1*.**EQ**(*package-2*);

必須引数

r
2 つの行列の値間の equal-to 比較の結果を含む行列を指定します。

package-1
equal-to 比較で使用される最初の行列パッケージ変数のインスタンスを指定します。

package-2
equal-to 比較で使用される 2 番目の行列パッケージ変数のインスタンスを指定します。

詳細

EQ 関係演算は、最初の行列の[i, j]番目の要素が 2 番目の行列の[i, j]番目の要素との比較を満たすかどうかを示す行列を生成します。結果は 0 または 1 です。[i, j]要素が等しい場合、結果は 1 になります。それ以外の場合、結果は 0 になります。

行列のサイズは一致する必要があります。一致しない場合は、スカラー比較を使用できます。

例

次の例では、2 つの行列の要素が等しいかどうかを比較します。欠損値が比較されることに注意してください。

```
data _null_;
  dcl double a[2,2];
  dcl double b[2,2];
  dcl double f[2,2];

  method run();
    dcl package matrix m;
    dcl package matrix m1;
     dcl package matrix r;
```

```
    dcl double i j;

  a := (2, 2, 3, .);
   b := (4, 5, 1, .);

  m = _new_ matrix(a, 2, 2);
   m1 = _new_ matrix(b, 2, 2);

  r = m.eq(m1);
   r.toarray(f);

   do i = 1 to 2;
    do j = 1 to 2;
      put f[i,j];
    end;
  end;
 end;
enddata;
run;
```

結果の行列の値は、次のとおりです。

```
0  0
0  1
```

関連項目

- *“行列演算” (SAS DS2 プログラマガイド)*

メソッド:

- “GE メソッド” (1863 ページ)
- “LE メソッド” (1870 ページ)
- “NE メソッド” (1880 ページ)

EXP メソッド

入力行列の各値の指数値を含む行列を返します。

構文

r=*package*.**EXP**( );

必須引数

r
EXP メソッドによって自動的に作成される行列を指定します。

package
行列パッケージ変数のインスタンスを指定します。

詳細

EXP メソッドは、入力行列の各要素を e乗した行列を作成します。

例

次の例では、行列の各要素を e乗します。

```
data _null_;
  dcl double a[3, 3];
  dcl double c[3, 3];

  method run();
    dcl package matrix m;
    dcl package matrix r;
     dcl double i j;

    a := (1, 2, 3, 1, 2, 3, 1, 2, 3);

    m=_new_ matrix(a, 3, 3);
    r=m.exp();
    r.toarray(c);

    do i=1 to 3;
      do j=1 to 3;
        put c[i, j];
      end;
    end;
  end;
enddata;
run;
```

次の行が SAS ログに書き込まれます。

```
2.71828182845904
7.38905609893065
20.0855369231876
2.71828182845904
7.38905609893065
20.0855369231876
2.71828182845904
7.38905609893065
20.0855369231876
```

関連項目

"行列演算" (SAS DS2 プログラマガイド)

FLOOR メソッド

入力行列の各値の整数部を含む行列を返します。

構文

r=*matrix-package*.**FLOOR**( );

必須引数

r
入力行列の各値の整数部を含む行列を指定します。

matrix-package
行列パッケージ変数のインスタンスを指定します。

例

次の例では、入力行列 **m** の整数部を含む行列を作成します。

```
data _null_;
  dcl double a[2, 2];
  dcl double c[2, 2];

  method run();
    dcl package matrix m;
    dcl package matrix r;
    dcl double i j;

    a := (1323.43, -.72, 3.38, 45);

    m=_new_ matrix(a, 2, 2);
    r=m.floor();
    r.toarray(c);

    do i=1 to 2;
      do j=1 to 2;
        put c[i, j];
      end;
    end;
  end;
enddata;
run;
```

結果の行列の値は、次のとおりです。

```
1323  0
3     45
```

関連項目

“行列演算” (SAS DS2 プログラマガイド)

GE メソッド

ある行列の要素と別の行列の要素の間の greater-than-or-equal-to 比較でスカラー結果を生成します。

構文

r=*package-1*.**GE**(*package-2*);

必須引数

r
2 つの行列の値間の greater-than-or-equal-to 比較の結果を含む行列を指定します。

package-1
greater-than-or-equal-to 比較で使用される最初の行列パッケージ変数のインスタンスを指定します。

package-2
greater-than-or-equal-to 比較で使用される 2 番目の行列パッケージ変数のインスタンスを指定します。

詳細

GE 関係演算は、最初の行列の[i, j]番目の要素が 2 番目の行列の[i, j]番目の要素との比較を満たすかどうかを示す行列を生成します。結果は 0 または 1 です。[i, j]要素の greater-than-or-equal-to 比較が真の場合、結果は 1 になります。それ以外の場合、結果は 0 になります。

行列のサイズは一致する必要があります。一致しない場合は、スカラー比較を使用できます。

例

```
r=m1.ge(m2);
```

関連項目

- *“行列演算” (SAS DS2 プログラマガイド)*

メソッド:

- “GT メソッド” (1864 ページ)
- “LE メソッド” (1870 ページ)

GT メソッド

ある行列の要素と別の行列の要素の間の greater-than 比較でスカラー結果を生成します。

構文

r=*package-1*.**GT**(*package-2*);

必須引数

r
2 つの行列の値間の greater-than 比較の結果を含む行列を指定します。

package-1
greater-than 比較で使用される最初の行列パッケージ変数のインスタンスを指定します。

package-2
greater-than 比較で使用される 2 番目の行列パッケージ変数のインスタンスを指定します。

詳細

GT 関係演算は、最初の行列の[i, j]番目の要素が 2 番目の行列の[i, j]番目の要素との比較を満たすかどうかを示す行列を生成します。結果は 0 または 1 です。[i, j]要素の greater-than 比較が真の場合、結果は 1 になります。それ以外の場合、結果は 0 になります。

行列のサイズは一致する必要があります。一致しない場合は、スカラー比較を使用できます。

例

```
r=m1.gt(m2);
```

関連項目

- “行列演算” (*SAS DS2 プログラマガイド*)

メソッド:

- “GE メソッド” (1863 ページ)
- “LT メソッド” (1873 ページ)

IN メソッド

配列を行列にロードします。

別名: LOAD

構文

package.**IN**(*array-name*);

必須引数

package
行列パッケージ変数のインスタンスを指定します。

array-name
行列のロードに使用される配列を指定します。

制限事項 配列の次元は、行列の次元と一致する必要があります。それ以外の場合は、エラーが発生します。

ヒント 変数配列を使用できます。

参照項目: “DS2 配列” (*SAS DS2 プログラマガイド*)

詳細

IN メソッドは、配列を行列にロードします。このプロセスは、プログラムの実行中に配列 **a** が変更される可能性があり、行列 **m** の値を繰り返しリセットする場合に役立つことがあります。次に例を示します。

```
dcl double a[3, 3];
dcl package matrix m;

m=_new_ matrix(3, 3);
m.in(a);
```

変数配列を使用して、データを行列にロードできます。次に例を示します。

```
vararray double va[3,3];
 dcl package matrix m;

 m.in(va);
```

変数配列を使用して、SET ステートメントと OUT ステートメントでデータを入出力することもできます。

例

例 1: データのロードと書き込み

この例では、行列形式のデータセットからデータを読み取り、行列の逆行列を求めて、結果行列を出力テーブルに書き込みます。この例では、IN メソッドと OUT メソッドを使用しています。IN メソッドは、変数配列から行列にデータをロードします。OUT メソッドは、行列のデータを変数配列に書き込みます。SET ステートメントと OUPUT ステートメントは、配列からデータを読み取り、結果を出力配列に書き込むためにプログラムで使用されます。

IN メソッドと OUT メソッドは、行列のどの行をロードするかを指示する整数引数を受け入れるようにオーバーロードされます。IN メソッドの場合、変数配列の行データが行列に読み込まれます。OUT メソッドの場合、行列の行データが変数配列に書き込まれます。

```
/* DATA step to create an array of data */
data x;
  array a[4];

  /* Create a 4x4 matrix. */
  a1 = 1; a2 = 5; a3 = 2; a4 = 3;
  output;

  a1 = 3; a2 = 3; a3 = 1; a4 = 7;
  output;

  a1 = 2; a2 = 3; a3 = 8; a4 = 9;
  output;

  a1 = 3; a2 = 6; a3 = 7; a4 = 4;
  output;
run;

proc ds2;
data inv/overwrite=yes;

  /* global declarations */
  vararray double v[4];
```

```
  vararray double a[4];
  keep v1 v2 v3;

  dcl package matrix m;
  dcl package matrix r;
  dcl package matrix inv;
  dcl double c[4, 4];
  dcl double i j;

  /* Create an empty matrix to hold the input values. */
  method init();
    m=_new_ matrix(4, 4);
    i=1;
  end;

  /* Read and initialize each row of the matrix from VARARRAY a. */
  method run();
    set x;
    m.in(a, i);
    i + 1;
  end;

  method term();
    /* Find the inverse of the matrix. */
    inv=m.inverse();

    /* Check whether it gives an identity matrix. */
    r=m.mult(inv);
    /* Write each row of inverse to the output table */
    /* using VARARRAY v.                             */
    do i=1 to 4;
      inv.out(v, i);
      output;
    end;

    /* Print the result to see if it's the identity. */
    r.toarray(c);
    do i=1 to 4;
      do j=1 to 4;
        put c[i, j];
      end;
    end;
  end;
enddata;
run;
quit;
```

次の行が SAS ログに書き込まれます。

```
1
-2.7755575615628E-17
-1.1102230246251E-16
0
1.1102230246251E-16
1
0
0
1.1102230246251E-16
5.5511151231257E-17
1
0
1.6653345369377E-16
1.6653345369377E-16
-1.6653345369377E-16
1
```

これらは、プログラムの主要な呼び出しです。

```
/*  Input  */
method run();
  set x;
  m.in(a, i);
  i + 1;
end;

/* Output */
  do i=1 to 4;
    inv.out(v, i);
    output;
  end;
```

例 2: IN メソッドと OUT メソッドの使用

IN メソッドと OUT メソッドを使用する別の例については、“例 2: 外部データから読み取られた 2 つの行列の乗算” (1879 ページ)を参照してください。

関連項目

- “行列データ入力” (*SAS DS2 プログラマガイド*)

メソッド:

- “OUT メソッド” (1883 ページ)

ステートメント:

- “OUTPUT ステートメント” (1386 ページ)
- “SET ステートメント” (1416 ページ)

INVERSE メソッド

行列の逆行列を計算します。

構文

im=*matrix-package*.**INVERSE**( );

必須引数

im
INVERSE メソッドによって自動的に作成される行列を指定します。

matrix-package
行列パッケージ変数のインスタンスを指定します。

詳細

単一の行列に対して基本的な行列演算を実行することが可能です。INVERSE 行列演算は、行列の逆行列を計算します。行列が正方でないか、特異である(反転できない)場合、ランタイムエラーが発生します。

例

次の例では、3x3 行列の逆行列を計算し、行列の乗算を使用して、結果の逆行列が単位行列を生成するかどうかを確認します。

```
data _null_;
  dcl double a[3,3];
  dcl double b[3,3];

  method run();
    dcl package matrix m im r;
    dcl double i j;

    a := (1, 2, -1, 2, 1, 0, -1, 1, 2);
    m = _new_ matrix(a, 3, 3);

    im = m.inverse();
    r = m.mult(im);
    r.toarray(b);

    do i = 1 to 3;
      do j = 1 to 3;
```

```
        put b[i,j];
      end;
    end;
  end;
enddata;
run;
```

結果の行列の一部の値は正確にゼロではありません。値は、1.1102230246251E-16 などの非常に小さい数値である可能性があります。これらの小さな数値は、SAS/IML サブシステムが提供する正確な結果です。

```
1                5.5511151231257E-17   0
0                1                     0
1.1102230246251E-16   -1.1102230246251E-16   1
```

関連項目

"行列演算" (SAS DS2 プログラマガイド)

LE メソッド

ある行列の要素と別の行列の要素の間の less-than-or-equal-to 比較でスカラー結果を生成します。

構文

r=*package-1*.**LE**(*package-2*);

必須引数

r
2 つの行列の値間の less-than-or-equal-to 比較の結果を含む行列を指定します。

package-1
less-than-or-equal-to 比較で使用される最初の行列パッケージ変数のインスタンスを指定します。

package-2
less-than-or-equal-to 比較で使用される 2 番目の行列パッケージ変数のインスタンスを指定します。

詳細

LE 関係演算は、最初の行列の[i, j]番目の要素が 2 番目の行列の[i, j]番目の要素との比較を満たすかどうかを示す行列を生成します。結果は 0 または 1 です。[i, j]要素

の less-than-or-equal-to 比較が真の場合、結果は 1 になります。それ以外の場合、結果は 0 になります。

行列のサイズは一致する必要があります。一致しない場合は、スカラー比較を使用できます。

例

例 1: LE メソッドを使用した 2 つの行列の比較

次の例では、LE メソッドを使用します。行列 **m** の[i, j]番目の要素と行列 **m2** の[i, j]番目の要素を比較し、結果のエントリに 0 または 1 を使用します。

```
data _null_;
  dcl double a[3,3];
  dcl double b[3,3];
  dcl double c[3,3];

  method run();
    dcl package matrix m;
    dcl package matrix m2;
    dcl package matrix r;
    dcl double i j;

    a := (1,2,3,4,5,6,7,8,9);
    b := (1,5,9,2,6,10,3,7,11);

    m = _new_ matrix(a, 3, 3);
    m2 = _new_ matrix(b, 3, 3);

    r = m.le(m2);
    r.toarray(c);

    do i = 1 to 3;
      do j = 1 to 3;
        put c[i,j];
      end;
    end;
  end;
enddata;
run;
```

結果の行列の値は、次のとおりです。

```
1   1   1
0   1   1
0   0   1
```

例 2: スカラー行列の使用

次の例では、LE メソッドでスカラー行列を使用しています。

```
data _null_;
  dcl double a[3,3];
  dcl double b[1,1];
```

```
  dcl double c[3,3];

  method run();
    dcl package matrix m;
    dcl package matrix m2;
    dcl package matrix r;
    dcl double i j;

    a := (1,3,3,4,5,6,7,8,9);
    b := (4);

    m = _new_ matrix(a, 3, 3);
    m2 = _new_ matrix(b, 1, 1);

    r = m.le(m2);
    r.toarray(c);

    do i = 1 to 3;
      do j = 1 to 3;
        put c[i,j];
      end;
    end;
  end;
enddata;
run;
```

結果の行列の値は、次のとおりです。

```
1   1   1
1   0   0
0   0   0
```

関連項目

- *“行列演算” (SAS DS2 プログラマガイド)*

メソッド:

- “GE メソッド” (1863 ページ)
- “LT メソッド” (1873 ページ)

LOG メソッド: 行列パッケージ

入力行列の各値の自然対数を含む行列を返します。

構文

r=*package*.**LOG**( );

必須引数

r
LOG メソッドによって自動的に作成される行列を指定します。

package
行列パッケージ変数のインスタンスを指定します。

関連項目

"行列演算" (SAS DS2 プログラマガイド)

LT メソッド

ある行列の要素と別の行列の要素の間の less-than 比較でスカラー結果を生成します。

構文

r=*package-1*.**LT**(*package-2*);

必須引数

r
2 つの行列の値間の less-than 比較の結果を含む行列を指定します。

package-1
less-than 比較で使用される最初の行列パッケージ変数のインスタンスを指定します。

package-2
less-than 比較で使用される 2 番目の行列パッケージ変数のインスタンスを指定します。

詳細

LT 関係演算は、最初の行列の[i, j]番目の要素が 2 番目の行列の[i, j]番目の要素との比較を満たすかどうかを示す行列を生成します。結果は 0 または 1 です。[i, j]要素の less-than 比較が真の場合、結果は 1 になります。それ以外の場合、結果は 0 になります。

行列のサイズは一致する必要があります。一致しない場合は、スカラー比較を使用できます。

例

例 1: LT メソッドを使用した 2 つの行列の比較

次の例では、LT メソッドを使用します。行列 **m** の[i, j]番目の要素と行列 **m2** の[i, j]番目の要素を比較し、結果のエントリに 0 または 1 を使用します。

```
data _null_;
  dcl double a[3,3];
  dcl double b[3,3];
  dcl double c[3,3];

  method run();
    dcl package matrix m;
    dcl package matrix m2;
    dcl package matrix r;
    dcl double i j;

    a := (1,2,3,4,5,6,7,8,9);
    b := (1,5,9,2,6,10,3,7,11);

    m = _new_ matrix(a, 3, 3);
    m2 = _new_ matrix(b, 3, 3);

    r = m.lt(m2);
    r.toarray(c);

    do i = 1 to 3;
      do j = 1 to 3;
        put c[i,j];
      end;
    end;
  end;
enddata;
run;
```

結果の行列の値は、次のとおりです。

```
0  1  1
0  1  1
0  0  1
```

例 2: スカラー行列の使用

次の例では、LT メソッドでスカラー行列を使用しています。

```
data _null_;
  dcl double a[3,3];
  dcl double b[1,1];
  dcl double c[3,3];

  method run();
```

```
    dcl package matrix m;
    dcl package matrix m2;
    dcl package matrix r;
    dcl double i j;

    a := (1,3,3,4,5,6,7,8,9);
    b := (4);

    m = _new_ matrix(a, 3, 3);
    m2 = _new_ matrix(b, 1, 1);

    r = m.lt(m2);
    r.toarray(c);

    do i = 1 to 3;
      do j = 1 to 3;
         put c[i,j];
      end;
    end;
  end;
enddata;
run;
```

結果の行列の値は、次のとおりです。

```
1   1   1
0   0   0
0   0   0
```

関連項目

- *“行列演算” (SAS DS2 プログラマガイド)*

メソッド:

- “GT メソッド” (1864 ページ)
- “LE メソッド” (1870 ページ)

NEW 演算子: 行列パッケージ

行列パッケージのインスタンスを構築します。

注: かっこの前のエスケープ文字(\)は、構文にかっこが必要であることを示します。

構文

package-variable=**_NEW_**

MATRIX([*array-name*,] *rows*, *columns*)

必須引数

package-variable
行列パッケージのインスタンスを参照できる名前を指定します。

rows
行列の行数を指定します。

columns
行列の列数を指定します。

オプション引数

array-name
行列パッケージにロードする配列の名前を指定します。

詳細

DS2 パッケージは、特定のインスタンスを構築して他の DS2 プログラムで使用できる変数とメソッドのコレクションです。

DECLARE PACKAGE ステートメントを使用して、行列パッケージ変数を宣言します。行列パッケージ変数を宣言した後、_NEW_演算子を使用して行列パッケージのインスタンスをインスタンス化し、新しいパッケージインスタンスを行列パッケージ変数に割り当てます。

行列は使用する前に初期化する必要があり、初期化は宣言ではなくコードストリームで行われます。行列を初期化するには、_NEW_演算子または DECLARE ステートメントを使用できます。

行列は、DS2 配列の値で初期化できます。次に例を示します。

```
method init();
  dcl double a[3, 3];
  dcl package matrix m;

  a :=(1, 2, -1, 2, 1, 0, -1, 1, 2);
  m=_new_ matrix(a, 3, 3);
end;
```

この例では、3x3 配列は指定した値で初期化され、行列 *m* を設定するために使用されます。値は行優先順に読み取られ、生成される行列の値は次のとおりです。

```
 1   2  -1
 2   1   0
-1   1   2
```

次の例に示すように、変数配列を使用して行列を初期化することもできます。

```
vararray double a [3, 3];
dcl package matrix m;
```

```
method init();
  a := (1, 2, -1, 2, 1, 0, -1, 1, 2);
  m=_new_ matrix(a, 3, 3);
end;
```

注: パッケージ変数は、すべての変数スコーピングルールの対象となります。詳細については、*"パッケージ、スコープ、および有効期間" (SAS DS2 プログラマガイド)*を参照してください。

関連項目

- *"行列データ入力" (SAS DS2 プログラマガイド)*
- *"パッケージのコンストラクターとデストラクター" (SAS DS2 プログラマガイド)*

ステートメント:

- "DECLARE PACKAGE ステートメント: 行列パッケージ" (1845 ページ)

MULT メソッド

ある行列を別の行列で乗算します。

構文

r=*matrix-package-1*.**MULT**(*matrix-package-2*);

必須引数

r
MULT メソッドによって自動的に作成される行列を指定します。

matrix-package-1
乗算演算で使用される最初の行列パッケージの名前を指定します。

matrix-package-2
乗算演算で使用される 2 番目の行列パッケージの名前を指定します。

詳細

MULT メソッドは、行列の乗算の結果を含む行列を自動的に作成します。結果の行列には、最初の行列と同じ行数と 2 番目の行列と同じ列数があります。

行列の乗算を実行する場合は、次の考慮事項が適用されます。

- 乗算演算で使用される行列の配列次元は互換性がなければなりません。乗算では、最初の行列の列数が 2 番目の行列の行数と等しい必要があります。それ以外の場合は、ランタイムエラーが発生します。
- 欠損値がある行列を乗算すると、ランタイムエラーが発生します。

例

例 1: 単純な行列の乗算

次の例では、MULT メソッドを使用して単純な行列乗算を実行します。3x4 行列 **m** に 4x3 行列 **m2** を掛けて、3x3 の結果を取得します。結果は行列 **r** に保存されます。行列 **r** の値は、3x3 配列 **c** に配置し、SAS ログに書き込むことができます。

```
data _null_;
  dcl double a[3,4];
  dcl double b[4,3];
  dcl double c[3,3];

  method run();
    dcl package matrix m;
    dcl package matrix m2;
    dcl package matrix r;
    dcl double i j;

    a := (1,2,3,4,5,6,7,8,9,10,11,12);
    b := (1,5,9,2,6,10,3,7,11,4,8,12);

    m = _new_ matrix(a, 3, 4);
    m2 = _new_ matrix(b, 4, 3);

    r = m.mult(m2);
    r.toarray(c);

    do i = 1 to 3;
      do j = 1 to 3;
        put c[i,j];
      end;
    end;
  end;
enddata;
run;
```

次の行が SAS ログに書き込まれます。

```
30
70
110
70
174
278
110
278
446
```

例 2: 外部データから読み取られた 2 つの行列の乗算

この例では、外部データから読み取られた 2 つの行列を乗算します。IN メソッドは行列を読み取り、OUT メソッドは出力を書き込みます。IN メソッドは、LOAD メソッドのエイリアスです。詳細については、“IN メソッド” (1865 ページ)および“OUT メソッド” (1883 ページ)を参照してください。

```
proc ds2;
 data x(keep =(a1 a2 a3)) y(keep=(b1 b2 b3 b4))/overwrite=yes;
  vararray double a[3];
  vararray double b[4];

  method init();
    dcl double i j;

    /* Create output for matrix a */
    do i = 1 to 4;
     do j = 1 to 3;
       a[j] = 2 * j + i;
     end;
    output x;
    end;

    /* Create output for matrix b */
    do i = 1 to 3;
     do j = 1 to 4;
       b[j] = 3 * j - 2 * i;
     end;
    output y;
    end;
  end;
enddata;
run;
quit;

proc ds2;
data z(keep =(v1 v2 v3 v4))/overwrite=yes;
  drop i;
  vararray double v[4];
  vararray double a[3];
  vararray double b[4];
  dcl package matrix ma mb;
  dcl package matrix r;

  method init();
    dcl double i;
    ma = _new_ matrix(4,3);
    mb = _new_ matrix(3,4);

    /* Read matrix a (row-by-row) */

    do i = 1 to 4;
      set x;
      ma.in(a, i);
    end;

    /* Read matrix b (row-by-row) */
```

```
    do i = 1 to 3;
      set y;
      mb.in(b, i);
    end;
  end;

  method term();
    dcl double i;
    /* Multiply matrices */
    r = ma.mult(mb);

    /* Output result */
    do i = 1 to 4;
      r.out(v, i);
      output;
    end;
  end;
enddata;
run;
quit;
```

関連項目

- “行列演算” (*SAS DS2 プログラマガイド*)

メソッド:

- “EMULT メソッド” (1855 ページ)

NE メソッド

ある行列の要素と別の行列の要素の間の not-equal-to 比較でスカラー結果を生成します。

構文

r=*matrix-package-1*.**NE**(*matrix-package-2*);

必須引数

r
2 つの行列の値間の not-equal-to 比較の結果を含む行列を指定します。

matrix-package-1
not-equal-to 比較で使用される最初の行列パッケージの名前を指定します。

matrix-package-2
not-equal-to 比較で使用される 2 番目の行列パッケージの名前を指定します。

詳細

NE 関係演算は、最初の行列の[i, j]番目の要素が 2 番目の行列の[i, j]番目の要素との比較を満たすかどうかを示す行列を生成します。結果は 0 または 1 です。[i, j]要素が等しくない場合、結果は 1 になります。それ以外の場合、結果は 0 になります。

行列のサイズは一致する必要があります。一致しない場合は、スカラー比較を使用できます。

例

次の例では、2 つの行列の要素が等しくないかどうかを比較します。欠損値が比較されることに注意してください。

```
data _null_;
  dcl double a[2,2];
  dcl double b[2,2];
  dcl double f[2,2];

  method run();
    dcl package matrix m;
    dcl package matrix m1;
      dcl package matrix r;
      dcl double i j;

    a := (2, 2, 3, .);
      b := (4, 5, 1, .);

    m = _new_ matrix(a, 2, 2);
      m1 = _new_ matrix(b, 2, 2);

    r = m.ne(m1);
      r.toarray(f);

      do i = 1 to 2;
       do j = 1 to 2;
          put f[i,j];
       end;
    end;
  end;
enddata;
run;
```

結果の行列の値は、次のとおりです。

```
1   1
1   0
```

関連項目

- *“行列演算” (SAS DS2 プログラマガイド)*

メソッド:

- “EQ メソッド” (1859 ページ)
- “GE メソッド” (1863 ページ)
- “LE メソッド” (1870 ページ)

OR メソッド

OR 論理演算に基づいて 2 つの行列を比較し、結果の行列を返します。

構文

r=*matrix-package-1*.**OR**(*matrix-package-2*);

必須引数

r
2 つの行列の値間の OR 比較の結果を含む行列を指定します。

matrix-package-1
OR 比較で使用される最初の行列パッケージの名前を指定します。

matrix-package-2
OR 比較で使用される 2 番目の行列パッケージの名前を指定します。

詳細

OR 演算子は、二項関係演算(LT、LE、GE、GT、NE、EQ)と同様に動作します。いずれの場合も、OR 論理演算は 2 つの行列の[i, j]番目の要素に適用され、結果の行列 *r* に配置されます。

例

```
r=m1.or(m2);
```

関連項目

- “行列演算” (*SAS DS2 プログラマガイド*)

メソッド:

- “ALL_OR メソッド” (1830 ページ)
- “AND メソッド” (1831 ページ)
- “ANY_OR メソッド” (1841 ページ)

OUT メソッド

行列の行データを変数配列に書き込みます。

構文

matrix-package.**OUT**(*array-name*);

必須引数

matrix-package
OUT メソッドで使用される行列パッケージを指定します。

array-name
出力の書き込みに使用される行列を指定します。

制限事項 配列の次元は、行列の次元と一致する必要があります。それ以外の場合は、エラーが発生します。

ヒント 変数配列を使用できます。

参照項目: “DS2 配列” (*SAS DS2 プログラマガイド*)

詳細

OUT メソッドは、行列の行データを変数配列に書き込みます。変数配列は、既存の DS2 テーブルと出力ステートメントを使用してデータを入力および出力するために使用できます。たとえば、行列形式のテーブルからデータを読み取り、行列の逆行列を求めて、結果行列を出力テーブルに書き込むことができます。次の例を参照してください。

OUT メソッドをオーバーロードして、行列のどの行を変数配列に書き込むかを指示する整数引数を受け入れることができます。

OUT メソッドの同義語は TOVARARRAY メソッドです。次の 2 つのステートメントは同等です。

```
r.tovararray(va, i);
r.out(va, i);
```

データを書き込む OUT メソッドは、よく IN メソッドと一緒に使用されます。IN メソッドと OUT メソッドは、行列のどの行をロードするかを指示する整数引数を受け

入れるようにオーバーロードされます。IN メソッドの場合、変数配列の行データが行列に読み込まれます。OUT メソッドの場合、行列の行データが変数配列に書き込まれます。このようにして、変数配列を使用して、外部データストレージとの間で一度に 1 行ずつ行列をロードおよびアンロードできます。

例

例 1

この例では、変数配列 *v* を使用して、逆行列の各行を出力テーブルに書き込みます。

```
do i=1 to 4;
  inv.out(v, i);
  output;
end;
```

例 2

標準の IN メソッドと同様に、完全な行列を変数配列に書き込むこともできます。

```
vararray double va(3, 3);
dcl package matrix r;

r=_new_ matrix(3, 3);
r.out(va);
```

この例では、行列を行ごとに記述する必要はありません。行列全体を変数配列に書き込むことができます。このメソッドは、DS2 出力ステートメント(行ベース)が使用されていない場合に使用できます。

例 3: データのロードと書き込み

IN メソッドと OUT メソッドを使用する別の例については、“例 2: 外部データから読み取られた 2 つの行列の乗算” (1879 ページ)を参照してください。

例 4: 行列全体を配列に書き込む

標準の LOAD メソッドと同様に、完全な行列を変数配列に書き込むこともできます。

```
vararray double va[3, 3];
dcl package matrix r;
  r=_new_ matrix(3, 3);
r.out(va);
```

この例は、行列データを行ごとに書き込む必要がないことを示しています。行列全体を配列に書き込むことができます。この手法は、DS2 OUTPUT ステートメント(行ベース)が使用されていない場合に使用できます。

関連項目

- “行列データ出力” (*SAS DS2 プログラマガイド*)

メソッド:

- “IN メソッド” (1865 ページ)
- “OUT メソッド” (1883 ページ)

ステートメント:

- “OUTPUT ステートメント” (1386 ページ)
- “SET ステートメント” (1416 ページ)

ROWS メソッド

指定された行列の行数を返します。

構文

variable-name=*matrix-package*.**ROWS**( );

必須引数

variable-name
メソッドの完了後の行数を含む変数の名前を指定します。

matrix-package
ROWS メソッドで使用する行列パッケージを指定します。

例

次の例では、行列 **m** の行と列の数を返します。

```
data _null_;
  dcl double a[2, 3];

  method run();
    dcl package matrix m;
    dcl double mr mc;

    a := (1,2,3,4,5,6);

    m=_new_ matrix(a, 2, 3);
    mr=m.rows();
```

```
    mc=m.cols();
    put mr=;
    put mc=;

  end;
run;
```

次の行が SAS ログに書き込まれます。

```
mr=2
mc=3
```

関連項目

- “MATRIX パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “COLS メソッド” (1843 ページ)

SQRT メソッド

入力行列の各値の平方根を含む行列を返します。

構文

r=*matrix-package*.**SQRT**( );

必須引数

r
入力行列の各値の平方根を含む行列を指定します。

matrix-package
SQRT メソッドで使用する行列パッケージを指定します。

関連項目

“行列演算” (*SAS DS2 プログラマガイド*)

SUB メソッド

ある行列を別の行列から減算します。

構文

r=*matrix-package-1*.**SUB**(*matrix-package-2*);

必須引数

r
SUB メソッドによって自動的に作成される行列を指定します。

matrix-package-1
減算演算で使用される最初の行列パッケージの名前を指定します。

matrix-package-2
減算演算で使用される 2 番目の行列パッケージの名前を指定します。

詳細

行列減算を実行するには、SUB メソッドの行列次元が同じサイズである必要があります。最初の行列の各[i, j]要素は、2 番目の行列の対応する[i, j]要素から減算されます。

欠損値がある行列を減算しても、エラーは発生しません。

1x1 行列を使用してスカラー減算を実行することもできます。

例

SUB メソッドを使用して 2 つの行列を減算する例を次に示します。

```
data _null_;
  dcl double c[3,3];
  dcl double d[1,1];
  dcl double f[3,3];
  method run();
    dcl package matrix m3;
    dcl package matrix m4;
    dcl package matrix r;
    dcl double i j;

    c := (-0, 0, -1, 1, -2.2, 2.2, -3.3, 4.4, 5.5);
    d := (1);
```

```
    m3 = _new_ matrix(c, 3, 3);
    m4 = _new_ matrix(d, 1, 1);

    r = m3.sub(m4);
    r.toarray(f);

    do i = 1 to 3;
      do j = 1 to 3;
        put f[i,j];
      end;
    end;
  end;
enddata;
run;
```

次の行が SAS ログに書き込まれます。

```
-1
-1
-2
0
-3.2
1.2
-4.3
3.4
4.5
```

関連項目

- *"行列演算" (SAS DS2 プログラマガイド)*

メソッド:

- "ADD メソッド: 行列パッケージ" (1819 ページ)

TOARRAY メソッド

値を行列パッケージから DS2 配列に移動します。

構文

array-name.**TOARRAY**(*matrix-package*);

必須引数

array-name
行列値の移動先となる配列の名前を指定します。

matrix-package
値が配列に移動される行列パッケージを指定します。

詳細

TOARRAY メソッドを使用して、値を行列から DS2 配列に移動します。配列は、DS2 プログラムで直接使用できます。

注: また、TOVARARRAY メソッドを使用して、値を変数配列に移動することもできます。

例

この例では、値を行列から DS2 配列に移動します。

```
data _null_;
  dcl double a[3,3];
  dcl double c[3,3];

  method init();
    dcl double i j;
    dcl package matrix m;

    a := (1,2,3,4,5,6,7,8,9);

    m=_new_ matrix(a,3,3);
    m.toarray(c);

    do i=1 to 3;
      do j=1 to 3;
        put c[i,j];
      end;
    end;
  end;
enddata;
run;
```

結果の配列の値は、次のとおりです。

```
1 2 3
4 5 6
7 8 9
```

関連項目

- “行列データ出力” (*SAS DS2 プログラマガイド*)
- “DS2 配列” (*SAS DS2 プログラマガイド*)

メソッド:

- “TOVARARRAY メソッド” (1890 ページ)

TOVARARRAY メソッド

値を行列パッケージから変数配列に移動します。

構文

variable-array-name.**TOVARARRAY**(*matrix-package*);

必須引数

variable-array-name
行列値の移動先となる変数配列の名前を指定します。

matrix-package
値が変数配列に移動される行列パッケージを指定します。

詳細

TOVARARRAY メソッドを使用して、値を行列から DS2 変数配列に移動します。変数配列は、DS2 プログラムで直接使用できます。

注: また、TOARRAY メソッドを使用して、値を配列に移動することもできます。

例

この例は、値を行列から変数配列に移動する方法を示しています。

```
data _null_;
  dcl double a[3, 3];
  vararray double c[3, 3];

  method run();
    dcl package matrix m;
    dcl double i j;

    a := (1,2,3,4,5,6,7,8,9);

    m=_new_ matrix(a, 3, 3);
    m.tovararray(c);
```

```
    do i=1 to 3;
      do j=1 to 3;
        put c[i, j];
      end;
    end;
  end;
run;
```

結果の配列の値は、次のとおりです。

```
1 2 3
4 5 6
7 8 9
```

関連項目

- *“行列データ出力” (SAS DS2 プログラマガイド)*
- *“変数配列” (SAS DS2 プログラマガイド)*

メソッド:

- “TOARRAY メソッド” (1888 ページ)

TRANS メソッド

入力行列の行と列を転置した行列を返します。

構文

r=*matrix-package*.**TRANS**( );

必須引数

r
入力行列の転置を含む行列を指定します。

matrix-package
TRANS メソッドで使用する行列パッケージを指定します。

詳細

TRANS メソッドは、特定の行列の行と列を交換して、行列の転置を生成します。**v** が行列の $i^{番目}$の行と $j^{番目}$の列の値である場合、転置された行列の $j^{番目}$の行と $i^{番目}$の列

に **v** が含まれます。行列に n 個の行と *p* 個の列が含まれる場合、転置された行列には *p* 個の行と *n* 個の列があります。

例

次の例では、3x4 行列を転置して、4x3 行列を生成します。

```
data _null_;
  dcl double a[3, 4];
  dcl double b[4, 3];

  method run();
    dcl package matrix m;
    dcl package matrix r;
      dcl double i j;

    a := (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12);

    m=_new_ matrix(a, 3, 4);
    r=m.trans();

    r.toarray(b);

    do i=1 to 4;
      do j=1 to 3;
        put b[i, j];
      end;
    end;
  end;
enddata;
run;
```

入力行列を次に示します。

```
1   2  3
4   5  6
7   8  9
10  11  12
```

転置された行列を次に示します。

```
1  5   9
2  6  10
3  7  11
4  8  12
```

関連項目

“行列演算” (SAS DS2 プログラマガイド)

21

DS2 PCRXFIND パッケージのメソッド、演算子、およびステートメント

ディクショナリ **1893**
DECLARE PACKAGE ステートメント: PCRXFIND パッケージ 1893
GETGROUP メソッド 1895
GETGROUPEND メソッド 1897
GETGROUPLENGTH メソッド 1898
GETGROUPSTART メソッド 1899
GETMATCH メソッド 1900
GETMATCHEND メソッド 1901
GETMATCHLENGTH メソッド 1903
GETMATCHSTART メソッド 1904
MATCH メソッド 1905
PARSE メソッド: PCRXFIND パッケージ 1906
NEW 演算子: PCRXFIND パッケージ 1908

ディクショナリ

DECLARE PACKAGE ステートメント: PCRXFIND パッケージ

パッケージ変数を作成し、PCRXFIND パッケージのインスタンスを作成するオプションを提供します。

カテゴリ: ローカル

ヒント: PCRXFIND パッケージには PACKAGE ステートメントは必要ありません。

構文

DECLARE PACKAGE PCRXFIND *variable* ([*regular-expression*]);

必須引数

variable
PCRXFIND パッケージのインスタンスを参照できる名前を指定します。

オプション引数

regular-expression
/expression/flags の形式で正規表現を指定しますこれらの正規表現 *flags* がサポートされています。

i
大文字と小文字の区別なし: 文字列内で大文字と小文字を区別しない表現の一致を許可します。

m
複数行: 文字列を複数行のセットとして扱います。埋め込まれた改行文字の隣で最初の文字の一致または最後の文字の一致が発生することを許可します。

n
非キャプチャ: 番号付きのキャプチャかっこを無効にします。

s
単一行: 文字列を単一の長い行として扱います。単一文字(.)の一致に改行文字を含めることができます。

x
空白文字(タブ、改行、キャリッジリターン、フォームフィード文字)とコメントを一致に含めることで、パターンを拡張します。

データの種類 VARCHAR

詳細

DS2 パッケージは、特定のインスタンスを構築して他の DS2 プログラムで使用できる変数とメソッドのコレクションです。

PCRXFIND パッケージを使用して、別のテキスト文字列で特定のテキスト文字列(*regular-expression*)を検索します。PCRXFIND パッケージは、DS2 プログラム用に事前定義されています。

DECLARE PACKAGE ステートメントを使用して、PCRXFIND パッケージを宣言します。新しい PCRXFIND パッケージを宣言した後、Perl 互換の正規表現を解析できます。(オプション)この表現は、パッケージが宣言されたときに渡すことができます。次に、ロードされた表現を使用して、一致の開始と終了のインデックス、および以前に解析された正規表現内の各かっこで囲まれたキャプチャグループに関連するインデックスを提供することにより、一致操作を実行できます。

パッケージが宣言されると、パッケージのインスタンスを参照できる変数が作成されます。コンストラクター引数がパッケージ変数宣言で提供されている場合、パッケージインスタンスが構築され、構築されたパッケージインスタンスを参照するようにパッケージ変数が設定されます。複数のパッケージ変数を作成でき、単一の DECLARE PACKAGE ステートメントで複数のパッケージインスタンスを構築できます。各パッケージインスタンスは、パッケージの完全に別個のコピーを表します。

PCRXFIND パッケージのインスタンスを構築する方法は 3 つあります。

- DECLARE PACKAGE ステートメントをそのコンストラクター構文と一緒に使用します。

  declare package pcrxfind finder(<'*regular-expression*'>);

- DECLARE PACKAGE ステートメントを_NEW_演算子と一緒に使用します。

  ```
  declare package pcrxfind finder;
  finder = _new_ pcrxfind(<'regular-expression'>);
  ```

- 表現を指定せずに DECLARE PACKAGE ステートメントを使用します。

  ```
  declare package pcrxfind finder;
  finder = _new_ pcrxfind();
  ```

関連項目

演算子:

- “_NEW_ 演算子: PCRXFIND パッケージ” (1908 ページ)

GETGROUP メソッド

前の一致に基づいて与えられたキャプチャグループによって表されるテキストを、ユーザーが指定した resultString に保存します。

要件

GETGROUP メソッドを使用する前に、MATCH メソッドを使用して一致に成功する(インデックスを返す)必要があります。一致に成功せずに GETGROUP メソッドを使用すると、エラーが発生し、プログラムの実行が停止します。

構文

returnCode=*package*.**GETGROUP** (IN_OUT *resultString*, *groupNumber*);

必須引数

package
PCRXFIND パッケージ変数のインスタンスを指定します。

groupNumber
resultString にテキストが保存されているキャプチャグループの番号を指定します。

データの種類 INTEGER

resultString
グループのコンテンツを保存する文字列を指定します。

データの種類 VARCHAR

戻り値

returnCode
ステータスインジケーターを指定します。

0
正常に実行されたことを示します。

–1
指定されたキャプチャグループが前の一致に関与していなかったことを示します。

データの種類 INTEGER

詳細

GETGROUP メソッドは、指定されたキャプチャグループのテキストを与えられた文字列に保存します。

関連項目

メソッド:

- “GETGROUPEND メソッド” (1897 ページ)
- “GETGROUPLENGTH メソッド” (1898 ページ)
- “GETGROUPSTART メソッド” (1899 ページ)

GETGROUPEND メソッド

与えられたキャプチャグループの終了インデックスを返します。

要件 GETGROUPEND メソッドを使用する前に、MATCH メソッドを使用して一致に成功する必要があります。一致に成功せずに GETGROUPEND メソッドを使用すると、エラーが発生し、プログラムの実行が停止します。

構文

ending-index=*package*.**GETGROUPEND** (*groupNumber*) returns int;

必須引数

groupNumber
キャプチャグループの番号を指定します。

データの種類 INTEGER

package
PCRXFIND パッケージ変数のインスタンスを指定します。

戻り値

ending-index
与えられたキャプチャグループの後の文字列内の次の位置を返します。

1 or greater
終了位置を指定します。

-1
指定されたキャプチャグループが前の一致に関与していなかったことを示します。

データの種類 INTEGER

詳細

GETGROUPEND メソッドは、一致グループまたはキャプチャグループの 後の文字列内の位置を返します。開始インデックスに長さを追加します。

```
int endIndex = rx.getGroupStart(n) + rx.getGroupLength(n);
```

関連項目

メソッド:

- “GETGROUP メソッド” (1895 ページ)
- “GETGROUPLENGTH メソッド” (1898 ページ)
- “GETGROUPSTART メソッド” (1899 ページ)

GETGROUPLENGTH メソッド

与えられたキャプチャグループのテキストの長さを文字数で返します。

要件 GETGROUPLENGTH メソッドを使用する前に、MATCH メソッドを使用して一致に成功する必要があります。一致に成功せずに GETGROUPLENGTH メソッドを使用すると、エラーが発生し、プログラムの実行が停止します。

構文

length=*package*.**GETGROUPLENGTH** (*groupNumber*);

必須引数

package
PCRXFIND パッケージ変数のインスタンスを指定します。

groupNumber
キャプチャグループの番号を指定します。

データの種類 INTEGER

戻り値

length
与えられたキャプチャグループのテキストの文字数を保持する変数を指定します。

1 or greater
与えられたキャプチャグループのテキストの長さを指定します。

-1
指定されたキャプチャグループが前の一致に関与していなかったことを示します。

データの種類 INTEGER

例

例については、“PCRXFIND パッケージの使用” (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

メソッド:

- “GETGROUP メソッド” (1895 ページ)
- “GETGROUPEND メソッド” (1897 ページ)
- “GETGROUPSTART メソッド” (1899 ページ)

GETGROUPSTART メソッド

与えられたキャプチャグループの開始インデックスを返します。

要件 GETGROUPSTART メソッドを使用する前に、MATCH メソッドを使用して一致に成功する必要があります。一致に成功せずに GETGROUPSTART メソッドを使用すると、エラーが発生し、プログラムの実行が停止します。

構文

starting-index=*package*.**GETGROUPSTART** (*groupNumber*);

必須引数

groupNumber
キャプチャグループの番号を指定します。

データの種類 INTEGER

package
PCRXFIND パッケージ変数のインスタンスを指定します。

戻り値

starting-index
前の一致の開始インデックスを指定します。

1 or greater
開始位置を指定します。

-1
指定されたキャプチャグループが前の一致に関与していなかったことを示します。

データの種類 INTEGER

関連項目

メソッド:

- “GETGROUP メソッド” (1895 ページ)
- “GETGROUPLENGTH メソッド” (1898 ページ)
- “GETGROUPEND メソッド” (1897 ページ)

GETMATCH メソッド

前の一致のテキストを結果文字列に配置します。

要件 GETMATCH メソッドを使用する前に、MATCH メソッドを使用して一致に成功する必要があります。一致に成功せずに GETMATCH メソッドを使用すると、エラーが発生し、プログラムの実行が停止します。

注: 詳細と例については、“事前定義された DS2 パッケージ” (*SAS DS2 プログラマガイド*) を参照してください。

構文

returnCode=*package*.**GETMATCH** (IN_OUT *resultString*);

必須引数

package
PCRXFIND パッケージ変数のインスタンスを指定します。

resultString
一致のコンテンツを保存する文字列を指定します。

データの種類 VARCHAR

戻り値

returnCode
ステータスインジケーターを指定します。

0
正常に実行されたことを示します。

データの種類 Integer

詳細

GETMATCH メソッドは、与えられた文字列に前の一致のコンテンツを保存します。

関連項目

メソッド:

- “GETMATCHEND メソッド” (1901 ページ)
- “GETMATCHLENGTH メソッド” (1903 ページ)
- “GETMATCHSTART メソッド” (1904 ページ)

GETMATCHEND メソッド

前の一致からのテキストの終了後の次の位置を返します。

要件 GETMATCHEND メソッドを使用する前に、MATCH メソッドを使用して一致に成功する必要があります。一致に成功せずに GETMATCHEND メソッドを使用すると、エラーが発生し、プログラムの実行が停止します。

構文

ending-index=*package*.**GETMATCHEND**();

必須引数

package
PCRXFIND パッケージ変数のインスタンスを指定します。

戻り値

ending-index
一致した文字列の最後の文字の後の位置を指定します。

2 or greater
一致した文字列の後の最初の値に対して 2 つ以上が返されます。

-1
指定されたキャプチャグループが前の一致に関与していなかったことを示します。

データの種類 INTEGER

詳細

GETMATCHEND メソッドを使用して、サブジェクト文字列内の一致した文字列の後の最初の位置を決定します。これにより、一致した文字列が終了するポイントから次の検索を簡単に開始できます。

例

この例では、正規表現を使用してテキスト行を検索します。GETMATCHEND メソッドは、一致した文字列の終了後の次の位置のインデックスを返します。

```
data regex_test/overwrite=yes;
  dcl char(20) text regex;
  dcl double parse_rc match_rc next_search_start;
  method run();
    dcl package pcrxfind prxf();
    text='quick brown fox';
    regex='/brown/';
    parse_rc=prxf.parse(regex);
    if parse_rc=0 then do;
      match_rc=prxf.match(text);
      if match_rc>0 then do;
        next_search_start=prxf.getmatchend();
        put next_search_start=;
      end;
    end;
  end;
enddata;
run;
```

次の行が SAS ログに書き込まれます。

```
next_search_start=12
```

関連項目

メソッド:

- “GETMATCH メソッド” (1900 ページ)
- “GETMATCHLENGTH メソッド” (1903 ページ)
- “GETMATCHSTART メソッド” (1904 ページ)

GETMATCHLENGTH メソッド

前の一致の文字数を返します。

要件 GETMATCHLENGTH メソッドを使用する前に、MATCH メソッドを使用して一致に成功する必要があります。一致に成功せずに GETMATCHLENGTH メソッドを使用すると、エラーが発生し、プログラムの実行が停止します。

構文

length=*package*.**GETMATCHLENGTH** ();

必須引数

package
PCRXFIND パッケージ変数のインスタンスを指定します。

戻り値

length
一致した文字列の文字数を指定します。長さには次のいずれかの値を指定できます。

1 or greater
一致した文字列の文字数。

-1
指定されたキャプチャグループが前の一致に関与していなかったことを示します。

データの種類 Integer

詳細

GETMATCHLENGTH メソッド

関連項目

メソッド:

- “GETMATCH メソッド” (1900 ページ)
- “GETMATCHEND メソッド” (1901 ページ)

■ “GETMATCHSTART メソッド” (1904 ページ)

GETMATCHSTART メソッド

前の一致操作の開始インデックスを返します。

要件 GETMATCHSTART メソッドを使用する前に、MATCH メソッドを使用して一致に成功する必要があります。一致に成功せずに GETMATCHSTART メソッドを使用すると、エラーが発生し、プログラムの実行が停止します。

構文

starting-index=*package*.**GETMATCHSTART**();

必須引数

package
PCRXFIND パッケージ変数のインスタンスを指定します。

戻り値

starting-index
一致した文字列の最初の文字の位置を指定します。これは、前回の MATCH メソッドの呼び出しと同じ値です。

1 or greater
開始インデックスとして 1 以上が返されます。

-1
指定されたキャプチャグループが前の一致に関与していなかったことを示します。

データの種類 INTEGER

詳細

GETMATCHSTART メソッドは、サブジェクト文字列内の一致した文字列の開始位置を決定するために使用されます。

関連項目

メソッド:

■ “GETMATCH メソッド” (1900 ページ)

- “GETMATCHEND メソッド” (1901 ページ)
- “GETMATCHLENGTH メソッド” (1903 ページ)

MATCH メソッド

以前に解析された正規表現を使用して一致操作を実行します。

構文

returnCode=*package*.**MATCH** (VARCHAR *subjectString*, [*offset-index*]) returns int;

必須引数

package
PCRXFIND パッケージ変数のインスタンスを指定します。

subjectString
正規表現を実行する文字列を指定します。

データの種類 VARCHAR

オプション引数

offset-index
正規表現を開始するオプションのインデックスを指定します。

データの種類 INTEGER

戻り値

returnCode
ステータスインジケーターを指定します。

1 or greater
一致が開始するインデックスを返します。

-1
正常に実行されたが、一致するものが見つからなかったことを示します。

データの種類 INTEGER

詳細

MATCH メソッドは、一致が開始するインデックスを返すか、正規表現が見つからない場合は–1 を返します。正規表現を解析する前に MATCH メソッドを呼び出すと、

エラーが発生します。一致が正常に実行されると、GetGroupStart などのメソッドは一致結果を操作に使用します。

例

この例では、正規表現を使用してテキスト行を検索します。MATCH メソッドは、一致が開始するインデックスを返します。

```
data regex_test/overwrite=yes;
  dcl char(20) text regex;
  dcl double parse_rc match_rc;
  method run();
    dcl package pcrxfind prxf();
    text='quick brown fox';
    regex='/brown/';
    parse_rc=prxf.parse(regex);
    if parse_rc=0 then do;
      match_rc=prxf.match(text);
      put match_rc=;
    end;
  end;
enddata;
run;
```

次の行が SAS ログに書き込まれます。

```
match_rc=7
```

関連項目

ステートメント

- “DECLARE PACKAGE ステートメント: PCRXFIND パッケージ” (1893 ページ)

PARSE メソッド: PCRXFIND パッケージ

パターンマッチングに使用できる Perl 互換の正規表現をコンパイルします。

注: SAS は、文字値のパターンマッチングに Perl 互換正規表現(PCRE)を採用しています。詳細については、PCRE - Perl Compatible Regular Expressions を参照してください。

構文

returnCode=*pcrxfind-package*.**PARSE** (*pcrxfind-expression*);

必須引数

pcrxfind-expression
/expression/flag1...flagn の形式で正規表現を指定します。

expression
文字列内の部分文字列をマッチングするために使用されるパターン。

> ヒント　*expression* 内のバックスラッシュを使用して、\w や\d などの特殊な正規表現文字をエスケープできます。

> ヒント　文字列の置換部分では、文字をエスケープするためにバックスラッシュは必要ありません。

flag
これらのフラグは、正規表現を解釈するために使用できます。

i
大文字と小文字の区別なし: 文字列内で大文字と小文字を区別しないパターンマッチを許可します。

m
複数行: 文字列を複数行のセットとして扱います。埋め込まれた改行文字の隣で最初の文字の一致または最後の文字の一致が発生することを許可します。

n
非キャプチャ: グループ化メタ文字()がキャプチャされないようにします。

s
単一行: 文字列を単一の長い行として扱います。単一文字(.)の一致に改行文字を含めることができます。

x
一致拡張: 空白文字(タブ、改行、キャリッジリターン、フォームフィード文字)とコメントを一致に含めることができます。

制限事項　マッチングモード修飾子の p、o、c、a、l はサポートされていません。

pcrxfind-package
PCRXFIND パッケージのインスタンスを指定します。

戻り値

returnCode
ステータスインジケーターを指定します。

0
正常に実行されたことを示します。

NULL
失敗を示し、エラーメッセージを返します。

データの種類 Integer

詳細

PCRXFIND パッケージの PARSE メソッドは、Perl 互換の正規表現を解析するために使用されます。

表現をコンパイルすると、同じパッケージが異なる表現を使用して解析できるようになります。MATCH メソッドを使用するには、解析が成功する必要があります。

例

次の例では、PCRXFIND パッケージを作成し、PARSE を使用して、電話番号を*(nnn) nnn-nnnn* の形式でマッチングする正規表現をコンパイルします。

```
dcl package pcrxfind phoneFinder();
method init();
  dcl int rc;
  rc = phoneFinder.parse('/(\(\d{3}\)) \d{3}-\d{4}/');
  if null(rc) then do;
    put 'ERROR: Could not parse the provided expression.';
  end;
end;
```

関連項目

メソッド

- “MATCH メソッド” (1905 ページ)

ステートメント

- “DECLARE PACKAGE ステートメント: PCRXFIND パッケージ” (1893 ページ)

NEW 演算子: PCRXFIND パッケージ

PCRXFIND パッケージのインスタンスを構築します。

注: かっこの前のエスケープ文字(\)は、構文にかっこが必要であることを示します。

構文

package-variable = **_NEW_ PCRXFIND**([*expression*]);

必須引数

package-variable
パッケージのインスタンスを参照できる名前を指定します。

オプション引数

expression
入力テキストの操作に使用される正規表現を指定します。

詳細

DS2 パッケージは、特定のインスタンスを構築して他の DS2 プログラムで使用できる変数とメソッドのコレクションです。

DECLARE PACKAGE ステートメントを使用して、PCRXFIND パッケージ変数を宣言します。PCRXFIND パッケージ変数を宣言した後、_NEW_演算子を使用して PCRXFIND パッケージのインスタンスをインスタンス化し、新しいパッケージインスタンスを PCRXFIND パッケージ変数に割り当てます。

```
declare package PCRXFIND finder;
finder = _new_ PCRXFIND();
```

DECLARE PACKAGE と_NEW_演算子を使用してロガーパッケージを宣言およびインスタンス化する 2 ステップのプロセスのかわりに、コンストラクターメソッドとして DECLARE PACKAGE ステートメントを使用することにより、パッケージを 1 ステップで宣言およびインスタンス化できます。これは、DECLARE PACKAGE ステートメントのみを使用した同じ例です。

```
declare package PCRXFIND finder();
```

注: パッケージ変数は、すべての変数スコーピングルールの対象となります。詳細については、"パッケージ、スコープ、および有効期間" (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- "PCRXFIND パッケージの使用" (*SAS DS2 プログラマガイド*)
- "パッケージのコンストラクターとデストラクター" (*SAS DS2 プログラマガイド*)

ステートメント:

- "DECLARE PACKAGE ステートメント: PCRXFIND パッケージ" (1893 ページ)

22

DS2 PCRXREPLACE パッケージのメソッド、演算子、およびステートメント

ディクショナリ **1911**
DECLARE PACKAGE ステートメント: PCRXREPLACE パッケージ 1911
APPLY メソッド 1913
PARSE メソッド: PCRXREPLACE パッケージ 1914
NEW 演算子: PCRXREPLACE パッケージ 1916

ディクショナリ

DECLARE PACKAGE ステートメント: PCRXREPLACE パッケージ

パッケージ変数を作成し、PCRXREPLACE パッケージのインスタンスを作成するオプションを提供します。

カテゴリ: ローカル

構文

DECLARE PACKAGE PCRXREPLACE *variable* ([*regular-expression*]);

必須引数

variable
PCRXREPLACE パッケージのインスタンスを参照できる名前を指定します。

オプション引数

regular-expression
[s]/*expression*/*substitution*/*flags* の形式で正規表現を指定します。*flags* は、次の 1 つ以上の値で構成できます。

g
グローバル: 文字列内で表現の一致と置換を可能な限り何度も実行できるようにします。

i
大文字と小文字の区別なし: 文字列内で大文字と小文字を区別しない表現の一致を許可します。

m
複数行: 文字列を複数行のセットとして扱います。埋め込まれた改行文字の隣で最初の文字の一致または最後の文字の一致が発生することを許可します。

n
非キャプチャ: 番号付きのキャプチャかっこを無効にします。

s
単一行: 文字列を単一の長い行として扱います。単一文字(.)の一致に改行文字を含めることができます。

x
空白文字(タブ、改行、キャリッジリターン、フォームフィード文字)とコメントを一致に含めることで、パターンを拡張します。

データの種類 VARCHAR

詳細

DS2 パッケージは、特定のインスタンスを構築して他の DS2 プログラムで使用できる変数とメソッドのコレクションです。

PCRXREPLACE パッケージにより、正規 *expression* を使用してテキストを置換します。PCRXREPLACE パッケージは、DS2 プログラム用に事前定義されています。

DECLARE PACKAGE ステートメントを使用して、PCRXREPLACE パッケージを宣言します。(オプション)正規表現を含めて、PCRXREPLACE パッケージを表現および置換テキストに関連付けることができます。新しい PCRXREPLACE パッケージを宣言した後、そのパッケージを使用して文字列の置換操作を実行できます。

パッケージが宣言されると、パッケージのインスタンスを参照できる変数が作成されます。コンストラクター引数がパッケージ変数宣言で提供されている場合、パッケージインスタンスが構築され、構築されたパッケージインスタンスを参照するようにパッケージ変数が設定されます。複数のパッケージ変数を作成でき、単一の

DECLARE PACKAGE ステートメントで複数のパッケージインスタンスを構築できます。各パッケージインスタンスは、パッケージの完全に別個のコピーを表します。

PCRXREPLACE パッケージのインスタンスを構築する方法は 3 つあります。

- DECLARE PACKAGE ステートメントをそのコンストラクター構文と一緒に使用します。

  declare package pcrxreplace swapper('/*expression*/*replacementText*/<*flags*>');

- DECLARE PACKAGE ステートメントを_NEW_演算子と一緒に使用します。

  ```
  declare package pcrxreplace swapper;
  swapper = _new_ pcrxreplace(<regular-expression>);
  ```

- 表現を指定せずに DECLARE PACKAGE ステートメントを使用します。

  ```
  declare package pcrxreplace swapper;
  swapper = _new_ pcrxreplace();
  ```

関連項目

演算子:

- “_NEW_ 演算子: PCRXREPLACE パッケージ” (1916 ページ)

APPLY メソッド

インスタンス化時に提供される置換式を、ロードされた対象データに適用します。

構文

returnCode=*package*.**APPLY**(IN_OUT *subjectString*, [*numberOfTimes*]);

必須引数

package
PCRXREPLACE パッケージ変数のインスタンスを指定します。

subjectString
対象データを指定します。

データの種類 VARCHAR

オプション引数

numberOfTimes
変更が適用される回数を指定します。

データの種類 INTEGER

戻り値

returnCode
リターンコードを配置する変数を指定します。

0
正常に実行されたことを示します。

NULL
失敗を示し、エラーメッセージを返します。

データの種類 INTEGER

詳細

APPLY メソッドは、文字列内の一致した文字のセットを指定された回数だけ置き換えます。

関連項目

ステートメント

- “DECLARE PACKAGE ステートメント: PCRXREPLACE パッケージ” (1911 ページ)

PARSE メソッド: PCRXREPLACE パッケージ

パターンマッチングに使用できる Perl 互換の正規表現をコンパイルします。

注: SAS は、文字値のパターンマッチングに Perl 互換正規表現(PCRE)を採用しています。詳細については、PCRE - Perl Compatible Regular Expressions を参照してください。

構文

returnCode=*pcrxreplace-package*.**PARSE** (*pcrxreplace-expression*);

必須引数

pcrxreplace-expression
[s]/*expression*/*substitution-text*/*flag1*...*flagn* の形式で正規表現を指定します。

expression
文字列内の部分文字列をマッチングするために使用されるパターン。

> ヒント *expression* 内のバックスラッシュを使用して、\w や\d などの特殊な正規表現文字をエスケープできます。

> ヒント 文字列の置換部分では、文字をエスケープするためにバックスラッシュは必要ありません。

flag
これらのフラグは、表現を解釈するために使用できます。

g
グローバル: 文字列内でパターンマッチと置換を可能な限り何度も実行できるようにします。

i
大文字と小文字の区別なし: 文字列内で大文字と小文字を区別しないパターンマッチを許可します。

m
複数行: 文字列を複数行のセットとして扱います。埋め込まれた改行文字の隣で最初の文字の一致または最後の文字の一致が発生することを許可します。

n
非キャプチャ: グループ化メタ文字()がキャプチャされないようにします。

s
単一行: 文字列を単一の長い行として扱います。単一文字(.)の一致に改行文字を含めることができます。

x
一致拡張: 空白文字(タブ、改行、キャリッジリターン、フォームフィード文字)とコメントを一致に含めることができます。

制限事項 マッチングモード修飾子の p、o、c、a、l はサポートされていません。

substitution-text
文字列内のパターンマッチテキストを置換するために使用されるテキスト。

データの種類 VARCHAR

pcrxreplace-package
PCRXREPLACE パッケージのインスタンスを指定します。

戻り値

returnCode
ステータスインジケーターを指定します。

0
正常に実行されたことを示します。

NULL
失敗を示し、エラーメッセージを返します。

データの種類 Integer

詳細

PCRXREPLACE パッケージの PARSE メソッドは、Perl 互換の正規表現を解析するために使用されます。

表現をコンパイルすると、同じパッケージが異なる表現を使用して解析できるようになります。APPLY メソッドを使用するには、解析が成功する必要があります。

例: Perl 正規表現のコンパイル

次の例では、PCRXREPLACE パッケージを作成し、PARSE を使用して、電話番号を *(nnn) nnn-nnnn* の形式に変換する正規表現をコンパイルします。

```
dcl package pcrxreplace normalizer();
method init();
  dcl int rc;
  rc = normalizer.parse(  '/.*(\d{3}).*(\d{3}).*(\d{4})/($1) $2-$3/' );
  if null(rc) then do;
    put 'ERROR: Could not parse the provided expression.';
  end;
end;
```

関連項目

メソッド

- “APPLY メソッド” (1913 ページ)

ステートメント

- “DECLARE PACKAGE ステートメント: PCRXREPLACE パッケージ” (1911 ページ)

NEW 演算子: PCRXREPLACE パッケージ

PCRXREPLACE パッケージのインスタンスを構築します。

注: かっこの前のエスケープ文字(\)は、構文にかっこが必要であることを示します。

構文

package-variable = **_NEW_ PCRXREPLACE**([*expression*]);

必須引数

package-variable
パッケージのインスタンスを参照できる名前を指定します。

オプション引数

expression
入力テキストの値を置き換えるために使用される正規表現を指定します。

詳細

DS2 パッケージは、特定のインスタンスを構築して他の DS2 プログラムで使用できる変数とメソッドのコレクションです。

DECLARE PACKAGE ステートメントを使用して、PCRXREPLACE パッケージ変数を宣言します。PCRXREPLACE パッケージ変数を宣言した後、_NEW_演算子を使用して PCRXREPLACE パッケージのインスタンスをインスタンス化し、新しいパッケージインスタンスを PCRXREPLACE パッケージ変数に割り当てます。

```
declare package pcrxreplace swapper;
swapper = _new_ pcrxreplace();
```

DECLARE PACKAGE と_NEW_演算子を使用してロガーパッケージを宣言およびインスタンス化する 2 ステップのプロセスのかわりに、コンストラクターメソッドとして DECLARE PACKAGE ステートメントを使用することにより、パッケージを 1 ステップで宣言およびインスタンス化できます。これは、DECLARE PACKAGE ステートメントのみを使用した同じ例です。

```
declare package pcrxreplace swapper();
```

注: パッケージ変数は、すべての変数スコーピングルールの対象となります。詳細については、"パッケージ、スコープ、および有効期間" (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- "PCRXREPLACE パッケージの使用" (*SAS DS2 プログラマガイド*)
- "パッケージのコンストラクターとデストラクター" (*SAS DS2 プログラマガイド*)

ステートメント:

- "DECLARE PACKAGE ステートメント: PCRXREPLACE パッケージ" (1911 ページ)

23

DS2 RedisCli パッケージのメソッド、演算子、およびステートメント

概要 **1919**
接続操作の順序 **1920**
ディクショナリ **1922**
CONNECT メソッド 1922
DECLARE PACKAGE ステートメント: RedisCli パッケージ 1925
DELETEKEY メソッド 1926
DISCONNECT メソッド 1927
EXISTS メソッド 1928
GETBINARY メソッド 1929
GETSTRING メソッド 1931
HGET メソッド 1932
HGETALL メソッド 1933
HGETSUBSET メソッド 1934
HSET メソッド 1935
NEW 演算子: RedisCli パッケージ 1936
SELECT メソッド 1937
SETBINARY メソッド 1938
SETSTRING メソッド 1940

概要

RedisCli パッケージは、DS2 コードから Redis サーバーへのアクセスを可能にする DS2 インターフェイスを提供します。RedisCli DS2 パッケージは、Redis コマンド

ラインインターフェイスである redis -cli に似た機能を提供します。このパッケージのサポートは、2023.10 で開始されます。

注: RedisCli パッケージは Redis Cluster をサポートしていません。RedisCli パッケージで次のようなエラーが発生した場合、Redis Cluster への接続を示している可能性があります。

ERROR: Line 95: redisInstRunCmd1sWithCallback encountered a failure in HGETALL: MOVED 3047 192.0.2.1:7000

ERROR: Line 95: redisInstRunCmd1sWithCallback encountered a failure.

ERROR: Line 95: d2hgetall encountered a failure in redisInstRunCmd1sWithCallback.

接続操作の順序

メソッドの呼び出し時に Redis サーバーへの接続が存在しない場合、RedisCli は、Redis サーバーへの接続に必要な情報をいくつかの方法で取得できます。connect メソッドの引数を使用したり、リソース構成に保存されているデフォルトの構成情報を取得したり、SAS_REDIS_*環境変数を使用したりできます。

操作の順序は次のとおりです。最初の 2 つのステップは connect メソッドにのみ適用されることに注意してください。

1 (これは connect メソッドにのみ適用されます。)

   connect メソッドは、2 つの引数が指定されているかどうかを判断します。はいの場合は、ステップ 1a に進みます。それ以外の場合は、ステップ 2 に進みます。

   a 2 番目の引数はポート値として使用されます。

   b 最初の引数がリソース構成名に対応するかどうかを判断します。

      1 はいの場合、リソース構成に構成されているプロパティを取得します。これらの値は、Redis サーバーに接続するときに使用するために保存されます。リソース構成で値が定義されていない場合、connect メソッドは、手順 4 で説明したように、環境変数を使用してそれらの値を取得しようとします。ステップ 4 に進みます。

      2 いいえの場合、値はホスト名であるとみなされます。ステップ 4 に進みます。

2 (これは connect メソッドにのみ適用されます。)

   connect メソッドは、引数が 1 つだけ指定されているかどうかを判断します。はいの場合は、ステップ 2a に進みます。それ以外の場合は、ステップ 3 に進みます。

   a 引数が整数かどうかを判断します。はいの場合は、それをポート値として保存します。ステップ 3 に進みます。

b 引数が既存のリソース構成名に対応するかどうかを判断します。

1 はいの場合、リソース構成に構成されているプロパティを取得します。これらの値は、Redis サーバーに接続するときに使用するために保存されます。リソース構成で値が定義されていない場合、connect メソッドは、手順 4 で説明したように、環境変数を使用してそれらの値を取得しようとします。

2 いいえの場合、値はホスト名であるとみなされます。ステップ 4 に進みます。

3 RedisCli メソッドは、デフォルトのリソース構成(**DefaultRedisResourceCfg**)が存在するかどうかを判断します。デフォルトのリソース構成を使用するには、*DefaultRedisResourceCfg* という名前を付ける必要があります。

a はいの場合、リソース構成に構成されているプロパティを取得します。これらの値は、Redis サーバーに接続するときに使用するために保存されます。リソース構成で値が定義されていない場合、connect メソッドは、手順 4 で説明したように、環境変数を使用してそれらの値を取得しようとします。

b いいえの場合は、ステップ 4 に進みます。

4 RedisCli メソッドは、次の環境変数を使用して値を取得しようとします。これらの値のいずれかが引数または構成リソースを使用して以前に取得されている場合は、それらの値が使用されます。

- SAS_REDIS_HOST

  注: この環境変数が定義されていない場合、デフォルト値の localhost (**127.0.0.1**)が使用されます。

- SAS_REDIS_PORT

  注: この環境変数が定義されていない場合、デフォルト値 **6379** が使用されます。

- SAS_REDIS_AUTH_USER
- SAS_REDIS_AUTH_PASS
- SAS_REDIS_TLS

  注: この環境変数が定義されていない場合、デフォルト値 **false** が使用されます。

- SAS_REDIS_CA_CERT
- SAS_REDIS_TRUST_CERT_PATH
- SAS_REDIS_CLIENT_CERT_FILE
- SAS_REDIS_CLIENT_PRIV_KEY_FILE
- SAS_REDIS_SERVER_DOMAIN

ディクショナリ

CONNECT メソッド

Redis サーバーに接続します。

注:

デフォルトでは、connect メソッドは最初に指定された引数を resource-config として使用します。リソース構成が見つからない場合、引数は host であるとみなされます。詳細については、"接続操作の順序"を参照してください。

RedisCli パッケージは Redis Cluster をサポートしていません。RedisCli パッケージで次のようなエラーが発生した場合、Redis Cluster への接続を示している可能性があります。

ERROR: Line 95: redisInstRunCmd1sWithCallback encountered a failure in HGETALL: MOVED 3047 192.0.2.1:7000

ERROR: Line 95: redisInstRunCmd1sWithCallback encountered a failure.

ERROR: Line 95: d2hgetall encountered a failure in redisInstRunCmd1sWithCallback.

構文

rc=*package*.**CONNECT** ([[*host* | *resource-config*] [, *port*]]);

戻り値

rc

リターンコード値を保持する変数を指定します。リターンコードは 32 ビットの INTEGER 値です。

0

成功を示します。

Nonzero

失敗を示します。

パッケージ変数

package

RedisCli パッケージインスタンスを参照する名前を指定します。

オプション引数

host

Redis サーバーのホスト名または IP アドレスを指定します。

resource-config
Redis サーバーにアクセスするために必要なプロパティを含むリソース構成の名前を指定します。

詳細については、“リソース構成” (1923 ページ)を参照してください。

指定された値を使用してリソース構成が見つからない場合、その値は *host* 値と見なされます。

port
Redis サーバーがリッスンしているポート番号を指定します。

詳細

リソース構成

*リソース構成*には、事前定義された構成情報を使用して Redis サーバーに接続するために必要な情報が含まれています。リソース構成は、コンピューティング環境 (SAS コンテナー Runtime や SAS Micro Analytic Service など)に保存されます。

connect メソッドは、**DefaultRedisResourceCfg** という名前のデフォルトのリソース構成を使用できます。このデフォルトを使用するには、コンピューティング環境に DefaultRedisResourceCfg リソース構成が存在する必要があります。

リソース構成の作成方法については、次のいずれかを参照してください。

- “Configuring a Connection to a Redis Server” (*SAS Micro Analytic Service: Programming and Administration Guide*)
- “Configuring a Redis Connection” (*SAS Container Runtime: Programming and Administration Guide*)

ホストとポート

指定された文字列引数がリソース構成名に解決されない場合、それが Redis サーバーのホスト名として使用されます。指定されたポート番号は、Redis サーバーポートとして使用されます。

あるいは、次の環境変数を使用して、*host* と *port* にデフォルト値を割り当てることができます。

- SAS_REDIS_HOST
- SAS_REDIS_PORT

ホストとポートのデフォルト値は、127.0.0.1(localhost)、そのポートは 6379 です。

ユーザー名とパスワード

次の環境変数を使用して、Redis ユーザー名とパスワードを指定できます。これらの環境変数に割り当てられた値は、リソース構成が指定されていない場合、または使用中のリソース構成にユーザー名、パスワード、またはその両方が含まれていない場合に使用されます。

- SAS_REDIS_AUTH_USER

- SAS_REDIS_AUTH_PASS

ユーザー名の値とパスワードの値の両方が指定された場合、指定されたパスワードが指定されたユーザーに使用されます。パスワードのみを指定した場合、指定されたパスワードは Redis サーバーのデフォルトユーザーに適用されます。

詳細については、Redis AUTH コマンドのドキュメントを参照してください。

注: 環境変数の設定方法については、コンピューティング環境のドキュメント(SAS Micro Analytic Service、SAS Compute Server、SAS Container Runtime など)を参照してください。

TLS セキュリティ

RedisCli は、データ伝送を保護するために Transport Layer Security (TLS)をサポートしています。

デフォルトでは、RedisCli の TLS セキュリティは無効になっています。Redis サーバーで TLS が必要な場合、リソース構成が指定されていないか、またはリソース構成で必要な TLS プロパティの一部が省略されているときは、次の環境変数を使用できます。

環境変数	説明	必須
SAS_REDIS_TLS	RedisCli での TLS の使用を有効または無効にします。 - RedisCli で TLS を有効にするには、この環境変数を 1 に設定します。 - RedisCli で TLS を無効にするには、この環境変数を 0 に設定します。 デフォルトでは、RedisCli では TLS が無効になっています。	RedisCli で TLS を使用するために必須です。
SAS_REDIS_CA_CERT	ロードして検証に使用する CA 証明書ファイルまたはバンドルファイルを指定します。 **注:** SAS Viya 配置では、SSLCALISTLOC 暗号化オプションの値を使用する必要があります。詳細については、"SSLCALISTLOC= System Option" (*SAS Viya Platform Encryption: Data in Motion*)を参照してください。	SAS_REDIS_TRUST_CERT_PATH が設定されていない場合は必須です。
SAS_REDIS_TRUST_CERT_PATH	信頼されている CA 証明書ファイルが OpenSSL 互換の構造で保存されているディレクトリパスを指定します。 **注:** SAS Viya 配置では、SSLCACERTDIR 暗号化オプションの値を使用する必要があります。詳細については、"SSLCACERTDIR= System Option" (*SAS Viya Platform*	SAS_REDIS_CA_CERT が設定されていない場合は必須です。

環境変数	説明	必須
	Encryption: Data in Motion)を参照してください。	
SAS_REDIS_CLIENT_CERT_FILE	双方向 TLS 接続で使用するクライアント証明書ファイルを指定します。 使用する場合は、SAS_REDIS_CLIENT_PRIV_KEY_FILE も指定する必要があります。	サーバーがクライアント証明書を必要とする場合(双方向 TLS)には必須です。
SAS_REDIS_CLIENT_PRIV_KEY_FILE	双方向 TLS 接続での認証に使用するクライアント秘密キーを指定します。	SAS_REDIS_CLIENT_CERT_FILE が設定されている場合は必須です。
SAS_REDIS_SERVER_DOMAIN	Server Name Indication (SNI)を指定します。	いいえ

Redis サーバーでの TLS の詳細については、Redis のセキュリティのドキュメントを参照してください。

環境変数の設定について

環境変数の設定方法については、コンピューティング環境のドキュメント(SAS Micro Analytic Service、SAS Cloud Analytic Services (CAS)、SAS Compute Server、SAS Container Runtime など)を参照してください。

DECLARE PACKAGE ステートメント: RedisCli パッケージ

パッケージ変数を作成し、RedisCli パッケージのインスタンスを作成できるようにします。

構文

DECLARE PACKAGE REDISCLI *variable*();

パッケージ変数

variable
RedisCli パッケージインスタンスを参照する名前を指定します。

詳細

DS2 パッケージは、特定のインスタンスを構築して他の DS2 プログラムで使用できる変数とメソッドのコレクションです。

Redis データベースとのインターフェイスには、RedisCli パッケージを使用します。RedisCli パッケージは、DS2 プログラム用に事前定義されています。

パッケージが宣言されると、パッケージのインスタンスを参照できる変数が作成されます。複数のパッケージ変数を作成でき、単一の DECLARE PACKAGE ステートメントで複数のパッケージインスタンスを構築できます。各パッケージインスタンスは、パッケージの完全に別個のコピーを表します。

RedisCli パッケージのインスタンスを構築する方法は 2 つあります。

- DECLARE PACKAGE ステートメントを_NEW_演算子と一緒に使用します。

```
declare package rediscli rds;
rds = _new_ rediscli();
```

- DECLARE PACKAGE ステートメントをそのコンストラクター構文と一緒に使用します。

```
declare package rediscli rds();
```

注: RedisCli パッケージを構築しても、Redis サーバーへの接続は確立されません。インスタンス化後、 “CONNECT メソッド”を使用してサーバーに接続します。

関連項目

- “パッケージのコンストラクターとデストラクター” (*SAS DS2 プログラマガイド*)

演算子:

- “_NEW_ 演算子: RedisCli パッケージ” (1936 ページ)

DELETEKEY メソッド

指定されたキーを削除します。

構文

rc=*package*.**DELETEKEY** (*key*);

戻り値

rc
リターンコード値を保持する変数を指定します。リターンコードは 32 ビットの INTEGER 値です。

0
成功を示します。

Nonzero
失敗を示します。

パッケージ変数

package
RedisCli パッケージインスタンスを参照する名前を指定します。

必須引数

key
削除するキーを識別する文字列値を指定します。

指定したキーが存在しない場合、コマンドは無視されます。キーが存在しなくてもエラーにはなりません。

DISCONNECT メソッド

Redis サーバーとの接続を解除します。

構文

rc=*package*.**DISCONNECT**();

戻り値

rc
リターンコード値を保持する変数を指定します。リターンコードは 32 ビットの INTEGER 値です。

0
成功を示します。

Nonzero
失敗を示します。

パッケージ変数

package
RedisCli パッケージインスタンスを参照する名前を指定します。

詳細

パッケージインスタンス変数を破棄する前に、Redis サーバーから接続を解除する必要はありません。自動接続解除がどのように発生するかの例を次に示します。

- パッケージインスタンス変数が破棄された場合。
- パッケージインスタンス変数がスコープ外になるか null に設定され、DS2 インフラストラクチャがガベージコレクションプロセスを実行する場合。

EXISTS メソッド

指定されたキーが存在するかどうかを判断します。

構文

rc=*package*.**EXISTS** (*key*, *isSet*);

戻り値

rc
リターンコード値を保持する変数を指定します。リターンコードは 32 ビットの INTEGER 値です。

0
成功を示します。

Nonzero
失敗を示します。

パッケージ変数

package
RedisCli パッケージインスタンスを参照する名前を指定します。

必須引数

key
キーを識別する文字列値を指定します。

isSet
キーが存在するかを示すために更新される変数を指定します。キーが存在する場合、*isSet* 変数には値 1 が割り当てられ、それ以外の場合は値 0 が割り当てられます。これは IN_OUT 引数です。

データの種類　INTEGER

詳細

指定されたキーが存在しない場合、値が BINARY 型ではない場合、またはエラーが発生した場合、*isSet* は欠損値で更新されます。

例

```
data _NULL_;
  method init();
    declare package rediscli redis();
    declare int rc;
    declare int isSet;
    rc = redis.connect();
    if (rc ^=0) then do;
      put 'Error connecting to redis';
      stop;
    end;
    rc = redis.exists("&key", isSet);
    if (rc ^= 0) then do;
      put 'Error checking if key exists';
      stop;
    end;
  end;
enddata;
```

GETBINARY メソッド

指定されたバイナリキーと割り当てられた値を取得します。

適用対象: RedisCli パッケージ

構文

rc=*package*.**GETBINARY** (*key*, *variable*);

戻り値

rc
リターンコード値を保持する変数を指定します。リターンコードは 32 ビットの INTEGER 値です。

0
成功を示します。

Nonzero
失敗を示します。

パッケージ変数

package
RedisCli パッケージインスタンスを参照する名前を指定します。

必須引数

key
取得するキーを識別する文字列値を指定します。

variable
取得したキー値で更新される変数を指定します。これは IN_OUT 引数です。

データの種類	VARBINARY、BINARY
注	VARBINARY が優先データ型です。

詳細

指定されたキーが存在しない場合、またはエラーが発生した場合、*variable* は null 値で更新されます。

例

次に GETBINARY メソッドを使用する例を示します。

```
proc ds2 libs=work;
  data _null_;
    dcl package logger logr( 'App.Test' );
    dcl package rediscli rds();
    method init();
      dcl int rc;
      dcl varchar(64) key;
      dcl varbinary(1) b1 b2;
      dcl int i32;
      rc = rds.connect();
      if rc then logr.log( 'e', 'connect error, rc=$s', rc );
      else do;
        key = 'mykey';
        b1 = x'FF';
        rc = rds.setBinary( key, b1, 'EX', 60 );
        if rc then logr.log( 'e', 'setBinary error, rc=$s', rc );
        else do;
          rc = rds.getBinary( key, b2 );
          if rc then logr.log( 'e', 'getBinary error, rc=$s', rc );
          else do;
            i32 = inputn( b2, 'PIB1.' );
            logr.log( 'd', 'key=$s, i32=$s', key, i32 );
          end;
        end;
      end;
```

```
    end;
  enddata;
  run;
quit;
```

GETSTRING メソッド

指定された文字列キーと割り当てられた値を取得します。

適用対象: RedisCli パッケージ

構文

rc=*package*.**GETSTRING** (*key*, *variable*);

戻り値

rc
リターンコード値を保持する変数を指定します。リターンコードは 32 ビットの INTEGER 値です。

0
成功を示します。

Nonzero
失敗を示します。

パッケージ変数

package
RedisCli パッケージインスタンスを参照する名前を指定します。

必須引数

key
取得するキーを識別する文字列値を指定します。

variable
取得したキー値で更新される変数を指定します。これは IN_OUT 引数です。

データの種類 CHAR、NCHAR、VARCHAR、NVARCHAR

詳細

指定されたキーが存在しない場合、またはエラーが発生した場合、*variable* は欠損値で更新されます。

HGET メソッド

指定されたキーに格納されているハッシュ内のフィールドに関連付けられた値を取得します。

構文

rc=*package*.**HGET** (*key*, *field*, *variable*);

戻り値

rc
リターンコード値を保持する変数を指定します。リターンコードは 32 ビットの INTEGER 値です。

0
成功を示します。

Nonzero
失敗を示します。

パッケージ変数

package
RedisCli パッケージインスタンスを参照する名前を指定します。

必須引数

key
ハッシュの一意識別子である文字列値を指定します。

field
値を取得するフィールドを識別する文字列値を指定します。

variable
取得したフィールド値で更新される変数を指定します。これは IN_OUT 引数です。

データの種類 CHAR、NCHAR、VARCHAR、NVARCHAR、VARBINARY

詳細

指定されたキーまたはフィールドが存在しない場合、またはエラーが発生した場合、*variable* は欠損値で更新されます。

HGETALL メソッド

指定されたキーに保存されているハッシュのすべてのフィールドと 値を取得し、指定されたデータグリッドに追加します。

構文

rc=*package*.**HGETALL** (*key*, *datagrid*);

戻り値

rc
リターンコード値を保持する変数を指定します。リターンコードは 32 ビットの INTEGER 値です。

0
成功を示します。

Nonzero
失敗を示します。

パッケージ変数

package
RedisCli パッケージインスタンスを参照する名前を指定します。

必須引数

key
ハッシュの一意識別子である文字列またはバイナリ値を指定します。

datagrid
指定されたキーから取得されたハッシュのフィールドと値を保存するデータグリッドパッケージインスタンスを指定します。この操作については、次の情報に注意する必要があります。

- 取得されたフィールドと値のペアは、追加の行としてデータグリッドに追加されます。
- HGETALL メソッドは、既存の列のデータ型を一致させようとします。値を列のデータ型に変換できない場合、割り当てられた値が欠損しています。
- フィールドが取得され、そのフィールドが既存の列と一致しない場合、新しい列がデータグリッドに追加されます。

  新しい列の型は文字列です。

詳細

エラーが発生した場合、または指定されたキーが Redis サーバー上のハッシュを参照していない場合、行はデータグリッドに追加されません。

HGETSUBSET メソッド

指定されたデータグリッドに行を追加します。次に、ハッシュのフィールドでデータグリッドの列と一致するフィールドごとに、その値を新しく追加されたデータグリッド行に追加します。

構文

rc=*package*.**HGETSUBSET** (*key*, *datagrid*);

戻り値

rc
リターンコード値を保持する変数を指定します。リターンコードは 32 ビットの INTEGER 値です。

0
成功を示します。

Nonzero
失敗を示します。

パッケージ変数

package
RedisCli パッケージインスタンスを参照する名前を指定します。

必須引数

key
ハッシュの一意識別子である文字列またはバイナリ値を指定します。

datagrid
値を照合して追加する列を含むデータグリッドパッケージインスタンスを指定します。この操作については、次の情報に注意する必要があります。

- 取得されたフィールド値は、追加行としてデータグリッドに追加されます。
- HGETSUBSET メソッドは、既存の列のデータ型を一致させようとします。値を列のデータ型に変換できない場合、割り当てられた値が欠損しています。
- フィールドが指定されたデータグリッドに列として存在しない場合、そのフィールドは無視されます。

詳細

エラーが発生した場合、または指定されたキーが Redis サーバー上のハッシュを参照していない場合、行はデータグリッドに追加されません。

HSET メソッド

指定されたフィールドを、指定されたキーに保存されているハッシュ内のそれぞれの値に設定します。

構文

rc=*package*.**HSET** (*key*, *field*, *value*);

戻り値

rc
リターンコード値を保持する変数を指定します。

0
成功を示します。

Nonzero
失敗を示します。

パッケージ変数

package
RedisCli パッケージインスタンスを参照する名前を指定します。

必須引数

key
ハッシュの一意識別子である文字列値を指定します。

field
値を割り当てるフィールドを識別する文字列値を指定します。

value
指定されたフィールドに割り当てる値を指定します。文字列、BINARY、VARBINARY、DOUBLE、BIGINT、INTEGER のいずれかの値を指定できます。

注: 欠損数値が指定されている場合は、長さゼロの文字列値が設定されます。

詳細

このコマンドは、ハッシュに存在する指定されたフィールドの値を上書きします。キーが存在しない場合は、ハッシュを保持する新しいキーが作成されます。

NEW 演算子: RedisCli パッケージ

RedisCli パッケージのインスタンスを構築します。

構文

package=**_NEW_REDISCLI**( );

パッケージ変数

package
RedisCli パッケージインスタンスを参照する名前を指定します。

詳細

DS2 パッケージは、特定のインスタンスを構築して他の DS2 プログラムで使用できる変数とメソッドのコレクションです。

注: RedisCli パッケージを構築しても、Redis サーバーへの接続は確立されません。インスタンス化後、 “CONNECT メソッド”を使用してサーバーに接続します。

DECLARE PACKAGE ステートメントを使用して、RedisCli パッケージ変数を宣言します。RedisCli パッケージ変数を宣言した後、_NEW_演算子を使用して RedisCli パッケージのインスタンスをインスタンス化し、新しいパッケージインスタンスを RedisCli パッケージ変数に割り当てます。

```
declare package rediscli rds;
rds = _new_ rediscli();
```

DECLARE PACKAGE と_NEW_演算子を使用して RedisCli パッケージを宣言およびインスタンス化する 2 ステップのプロセスのかわりに、コンストラクターメソッドとして DECLARE PACKAGE ステートメントを使用することにより、パッケージを 1 ステップで宣言およびインスタンス化できます。これは、DECLARE PACKAGE ステートメントのみを使用した同じ例です。

```
declare package rediscli rds();
```

注: パッケージ変数は、すべての変数スコーピングルールの対象となります。詳細については、*"パッケージ、スコープ、および有効期間" (SAS DS2 プログラマガイド)*を参照してください。

関連項目

"パッケージのコンストラクターとデストラクター" (SAS DS2 プログラマガイド)

SELECT メソッド

指定されたゼロベースの数値インデックスを持つ論理データベースを選択します。

構文

rc=*package*.**SELECT** ([*logicalDB*]);

戻り値

rc
リターンコード値を保持する変数を指定します。リターンコードは INTEGER 値です。

0
成功を示します。

Nonzero
失敗を示します。

パッケージ変数

package
RedisCli パッケージ変数の名前を指定します。

オプション引数

logicalDB
論理データベースの数値インデックスを指定します。

詳細

logicalDB 引数を指定しない場合、SAS_REDIS_LOGICAL_DATABASE 環境変数に割り当てられた値を使用して、論理データベースが選択されます。

SELECT メソッドを使用しない場合、RedisCli はデフォルトのデータベースに接続します。デフォルトのデータベースは、論理データベース 0 (デフォルト値)か、SAS_REDIS_LOGICAL_DATABASE 環境変数に指定されているもののいずれかです。

注: 環境変数の設定方法については、コンピューティング環境のドキュメント(SAS Micro Analytic Service、SAS Cloud Analytic Services (CAS)、SAS Compute Server、SAS Container Runtime など)を参照してください。

SETBINARY メソッド

バイナリ値を設定します。

適用対象: RedisCli パッケージ

構文

rc=*package*.**SETBINARY** (*key*, *value* [,*options* [, *option_value*]]);

戻り値

rc
リターンコード値を保持する変数を指定します。リターンコードは 32 ビットの INTEGER 値です。

0
成功を示します。

Nonzero
失敗を示します。

パッケージ変数

package
RedisCli パッケージインスタンスを参照する名前を指定します。

必須引数

key
キーを識別する文字列値を指定します。

value
指定されたキーに割り当てるバイナリ値を指定します。

オプション引数

options
メソッドの動作を変更する次のオプション文字列のいずれか。複数のオプションを指定する場合は、各オプションをスペースで区切ります。

GET
指定されたキーに保存されている既存の BINARY を返します。キーが存在しない場合は null を返します。キーに保存されている値がバイナリでない場合、エラーが返され、SET 操作は終了します。

KEEPTTL
キーに関連付けられている有効期間を保持します。

NX | XX
NX は、キーがまだ存在しない場合にのみキーを設定します。XX は、キーがすでに存在する場合にのみキーを設定します。

EX | PX | EXAT | PXAT
EX は、有効期限を秒単位で設定します。PX は、有効期限をミリ秒単位で設定します。

EXAT は、キーの有効期限が切れる UNIX タイムスタンプを秒単位で設定します。PXAT は、キーの有効期限が切れる UNIX タイムスタンプをミリ秒単位で設定します。

注: これらの時間ベースのオプションのいずれかを使用する場合、他のオプションも指定されている場合は、それを最後のオプションとして指定してください。これは、時間ベースのオプションの対応する *option_value* がオプションの直後に続く必要があるため必要です。

option_value
EX、PX、EXAT、または PXAT が指定されている場合、そのオプションに割り当てる値を指定します。指定する値は、オプションの直後に続け、その前にカンマを付ける必要があります。次に例を示します。

```
rc = rds.setBinary( key, value, 'NX EX', 2 );
```

例

次に SETBINARY メソッドを使用する例を示します。

```
proc ds2 libs=work;
  data _null_;
    dcl package logger logr( 'App.Test' );
    dcl package rediscli rds();
    method init();
      dcl int rc;
      dcl varchar(64) key;
      dcl varbinary(1) b1 b2;
      dcl int i32;
      rc = rds.connect();
      if rc then logr.log( 'e', 'connect error, rc=$s', rc );
      else do;
```

```
        key = 'mykey';
        b1 = x'FF';
        rc = rds.setBinary( key, b1, 'EX', 60 );
        if rc then logr.log( 'e', 'setBinary error, rc=$s', rc );
        else do;
          rc = rds.getBinary( key, b2 );
          if rc then logr.log( 'e', 'getBinary error, rc=$s', rc );
          else do;
            i32 = inputn( b2, 'PIB1.' );
            logr.log( 'd', 'key=$s, i32=$s', key, i32 );
          end;
        end;
      end;
    end;
  enddata;
  run;
quit;
```

SETSTRING メソッド

文字列キーと値を設定します。

適用対象: RedisCli パッケージ

構文

rc=*package*.**SETSTRING** (*key*, *value* [, *options* [, *option_value*]]);

戻り値

rc
リターンコード値を保持する変数を指定します。リターンコードは 32 ビットの INTEGER 値です。

0
成功を示します。

Nonzero
失敗を示します。

パッケージ変数

package
RedisCli パッケージインスタンスを参照する名前を指定します。

必須引数

key
キーを識別する文字列値を指定します。

value
指定されたキーに割り当てる文字列値を指定します。

オプション引数

options
メソッドの動作を変更する次のオプション文字列のいずれか。複数のオプションを指定する場合は、各オプションをスペースで区切ります。

- **GET:** 指定されたキーに保存されている既存のバイナリを返します。キーが存在しない場合は null を返します。キーに保存されている値が BINARY でない場合、エラーが返され、SET 操作は終了します。
- **KEEPTTL:** キーに関連付けられている有効期間を保持します。
- **NX | XX:** NX は、キーがまだ存在しない場合にのみキーを設定します。XX は、キーがすでに存在する場合にのみキーを設定します。
- **EX | PX | EXAT | PXAT:** EX は、有効期限を秒単位で設定します。PX は、有効期限をミリ秒単位で設定します。

  EXAT は、キーの有効期限が切れる UNIX タイムスタンプを秒単位で設定します。PXAT は、キーの有効期限が切れる UNIX タイムスタンプをミリ秒単位で設定します。

  注: これらの時間ベースのオプションのいずれかを使用する場合、他のオプションも指定されている場合は、それを最後のオプションとして指定してください。これが必要なのは、時間ベースのオプションの対応する *option_value* がオプションの直後に続く必要があるためです。

option_value
EX、PX、EXAT、または PXAT が指定されている場合、そのオプションに割り当てる値。指定する値は、オプションの直後に続け、その前にカンマを付ける必要があります。次に例を示します。

```
rc = rds.setString( key, 'risk1|west|ATM|0123|withdrawal|4', 'NX EX', 2 );
```

詳細

指定できるキーと値のペアは 1 つだけです。指定する値が区切り複合値としてフォーマットされている場合は、パイプ文字(|)を区切り文字として使用します。区切り複合値を使用する例を次に示します。

```
proc ds2 libs=work;
  data _null_;
    dcl package logger logr( 'App.Test' );
    dcl package rediscli rds( 'my instance name' );
    dcl varchar(64) key strg;
    method init();
      dcl int rc;
      rc = rds.connect();
      key = 'group|region|type|terminal|trans:count';
      rc = rds.setString( key, 'risk1|west|ATM|0123|withdrawal|4', 'NX EX', 2 );
      rc = rds.getString( key, strg );
```

```
      if rc then logr.log( 'e', 'getString error, rc=$s', rc );
      else logr.log( 'd', 'getString, key=$s, strg=$s', key, strg );
    end;
  enddata;
  run;
quit;
```

24

DS2 SQLSTMT パッケージのメソッド、演算子、およびステートメント

ディクショナリ **1944**
BINDPARAMETERS メソッド 1944
BINDRESULTS メソッド 1946
BULKINSERTADDROW メソッド 1947
BULKINSERTBINDCOLUMNS メソッド 1948
BULKINSERTFLUSH メソッド 1949
BULKINSERTGETCOLUMNCOUNT メソッド 1950
BULKINSERTGETCOLUMNNAME メソッド 1951
BULKINSERTGETCOLUMNTYPENAME メソッド 1952
BULKINSERTISPREPARED メソッド 1954
BULKINSERTPREPARE メソッド 1955
CLOSERESULTS メソッド 1956
DECLARE PACKAGE ステートメント: SQLSTMT パッケージ 1957
DELETE メソッド: SQLSTMT パッケージ 1959
EXECUTE メソッド 1960
FETCH メソッド 1961
GETBIGINT メソッド 1962
GETBINARY メソッド 1964
GETCHAR メソッド 1965
GETCOLUMNCOUNT メソッド 1967
GETCOLUMNNAME メソッド 1967
GETCOLUMNTYPENAME メソッド 1968
GETDATE メソッド 1969
GETDECIMAL メソッド 1971
GETDOUBLE メソッド 1972
GETINTEGER メソッド 1974
GETNCHAR メソッド 1975
GETNUMERIC メソッド 1977
GETNVARCHAR メソッド 1978
GETREAL メソッド 1980
GETSMALLINT メソッド 1981

GETTIME メソッド ... 1983
GETTIMESTAMP メソッド ... 1984
GETTINYINT メソッド ... 1986
GETVARBINARY メソッド ... 1987
GETVARCHAR メソッド ... 1989
ISPREPARED メソッド ... 1991
NEW 演算子: SQLSTMT パッケージ ... 1991
PREPARE メソッド ... 1994
REGISTEROUTPARAMETER メソッド ... 1995
SETBIGINT メソッド ... 1997
SETBINARY メソッド ... 1999
SETCHAR メソッド ... 2000
SETDATE メソッド ... 2001
SETDECIMAL メソッド ... 2002
SETDOUBLE メソッド ... 2004
SETINTEGER メソッド ... 2005
SETNCHAR メソッド ... 2006
SETNUMERIC メソッド ... 2008
SETNVARCHAR メソッド ... 2009
SETREAL メソッド ... 2010
SETSMALLINT メソッド ... 2011
SETTIME メソッド ... 2013
SETTIMESTAMP メソッド ... 2014
SETTINYINT メソッド ... 2015
SETVARBINARY メソッド ... 2017
SETVARCHAR メソッド ... 2018

ディクショナリ

BINDPARAMETERS メソッド

変数のリストを FedSQL ステートメントのパラメーターにバインドします。

制限事項: このメソッドは CAS サーバーではサポートされていません。

構文

package.**BINDPARAMETERS** ([*parameter-variable-list*]);

必須引数

package
SQLSTMT パッケージのインスタンスを指定します。

オプション引数

[*parameter-variable-list*]
FedSQL ステートメントのパラメーターにバインドする変数を含む変数リストまたは名前付き変数リストを指定します。

要件 変数は、角かっこ([])で囲む必要がある変数リストまたは名前付き変数リストの形式である必要があります。

ヒント 変数リストの変数の数は、FedSQL ステートメントのパラメーターの数と一致する必要があります。

参照項目: "FedSQL ステートメントのパラメーター値の指定" (*SAS DS2 プログラマガイド*)

詳細

FedSQL ステートメントにパラメーターが含まれている場合、FedSQL ステートメントを実行するには、パラメーターと置き換える値を取得する必要があります。置換値は、バインドされた変数の現在の値または SQLSTMT パッケージの SET*type* メソッドのいずれかを使用して指定できます。

BINDPARAMETERS メソッドは、指定された変数リストの変数を FedSQL ステートメントのパラメーターにバインドします。

パラメーター値は、バインドされた変数または SET*type* メソッドで排他的に指定する必要があります。変数がパラメーターにバインドされ、SET*type* メソッドが呼び出されると、ランタイムエラーが発生します。

バインドされた変数の型が対応するパラメーターの型と異なる場合、バインドされた変数の値はパラメーターの型に変換されます。

BINDPARAMETERS メソッドはステータスインジケーターを返します。正常に実行されるとゼロが返され、エラーが発生した場合は 1 が返されます。

FedSQL ステートメントの実行後に BINDPARAMETERS メソッドが呼び出された場合も、ランタイムエラーが発生します。

関連項目

- "SQLSTMT パッケージの使用" (*SAS DS2 プログラマガイド*)

メソッド:

- "BINDRESULTS メソッド" (1946 ページ)
- この章の SET "type"メソッド

BINDRESULTS メソッド

変数のリストを FedSQL ステートメントの結果セットの列にバインドします。

制限事項: このメソッドは CAS サーバーではサポートされていません。

構文

package.**BINDRESULTS** (*parameter-variable-list*);

必須引数

package
SQLSTMT パッケージのインスタンスを指定します。

parameter-variable-list
結果セットの列にバインドする変数を含む変数リストまたは名前付き変数リストを指定します。

要件 変数は、角かっこ([])で囲む必要がある変数リストまたは名前付き変数リストの形式である必要があります。

ヒント 変数リストの変数の数は、結果セットの列の数と一致する必要があります。

参照項目: "FedSQL ステートメントのパラメーター値の指定" (*SAS DS2 プログラマガイド*)

詳細

FETCH メソッドは、結果セットから次のデータ行を返します。変数が BINDRESULTS メソッドで結果セットの列にバインドされている場合、各結果セットの列のフェッチされたデータは、その列にバインドされている変数に配置されます。変数の型が対応する列の型と異なる場合、列のデータ値は変数の型に変換されます。

結果データには、バインドされた変数または GET*type* メソッドで排他的にアクセスする必要があります。変数が結果セットの列にバインドされ、GET*type* メソッドが呼び出されると、ランタイムエラーが発生します。

結果データがフェッチされた後に BINDRESULTS メソッドが呼び出されると、ランタイムエラーが発生します。

BINDRESULTS メソッドはステータスインジケーターを返します。正常に実行されるとゼロが返され、エラーが発生した場合は 1 が返されます。

関連項目

- “SQLSTMT パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “BINDPARAMETERS メソッド” (1944 ページ)
- この章の SET "type"メソッド

BULKINSERTADDROW メソッド

バルク挿入用のデータ行を追加します。

構文

package.**BULKINSERTADDROW**();

必須引数

package
SQLSTMT パッケージのインスタンスを指定します。

要件 FedSQL ステートメントは、ステートメントが文字列変数に保存されていない限り、単一引用符(')で囲む必要があります。

注 ステートメントは文字列リテラルです。

FedSQL 言語の識別子のルールは、識別子の DS2 ルールではなく、SQLSTMT パッケージで使用される変数に適用されます。これは、FedSQL (DS2 ではない)が FedSQL ステートメントを含む文字列を解析するために発生します。

注意 **SQLSTMT パッケージで使用できるのは FedSQL ステートメントのみです。DBMS 固有の SQL は使用できません。** 詳細については、*SAS FedSQL 言語リファレンス*を参照してください。

詳細

各挿入列のデータは、BULKINSERTBINDCOLUMNS メソッドによってバインドされた対応する変数からコピーされます。挿入列のデータ型が対応する変数のデータ型と一致しない場合、コピーされたデータ値は挿入列のデータ型に変換されます。

BULKINSERTBINDCOLUMNS メソッドを呼び出す前に BULKINSERTADDROW メソッドを呼び出すと、ランタイムエラーが発生します。

BULKINSERTADDROW メソッドはステータスインジケーターを返します。正常に実行されるとゼロが返され、エラーが発生した場合は 1 が返されます。

関連項目

"BULKINSERTBINDCOLUMNS メソッド"

BULKINSERTBINDCOLUMNS メソッド

変数のリストを、バルク挿入用に選択された列にバインドします。

制限事項: このメソッドは CAS サーバーではサポートされていません。

構文

package.**BULKINSERTBINDCOLUMNS**(*variable-list*);

必須引数

package
SQLSTMT パッケージのインスタンスを指定します。

要件 FedSQL ステートメントは、ステートメントが文字列変数に保存されていない限り、単一引用符(')で囲む必要があります。

注 ステートメントは文字列リテラルです。

FedSQL 言語の識別子のルールは、識別子の DS2 ルールではなく、SQLSTMT パッケージで使用される変数に適用されます。これは、FedSQL (DS2 ではない)が FedSQL ステートメントを含む文字列を解析するために発生します。

注意 **SQLSTMT パッケージで使用できるのは FedSQL ステートメントのみです。DBMS 固有の SQL は使用できません。** 詳細については、*SAS FedSQL 言語リファレンス*を参照してください。

variable-list
列にバインドする変数を含む変数リストを指定します。

要件 変数は、角かっこ([])で囲む必要がある変数リストまたは名前付き変数リストの形式である必要があります。

ヒント 変数リストの変数の数は、結果セットの列の数と一致する必要があります。

参照項目: "FedSQL ステートメントのパラメーター値の指定" (*SAS DS2 プログラマガイド*)

詳細

variable-list で指定される変数の数は、バルク挿入用に選択された列の数と一致する必要があります。挿入列のデータ型が対応する変数のデータ型と一致しない場合、コピーされたデータ値は挿入列のデータ型に変換されます。

バルク挿入ステートメントが BULKINSERTPREPARE メソッドによって準備される前に BULKINSERTBINDCOLUMNS メソッドを呼び出すと、ランタイムエラーが発生します。

BULKINSERTBINDCOLUMNS メソッドはステータスインジケーターを返します。正常に実行されるとゼロが返され、エラーが発生した場合は 1 が返されます。

関連項目

- “BULKINSERTISPREPARED メソッド”
- “BULKINSERTPREPARE メソッド”

BULKINSERTFLUSH メソッド

バルク挿入行バッファー内のすべての行をフラッシュします。

制限事項: このメソッドは CAS サーバーではサポートされていません。

構文

package.**BULKINSERTFLUSH**();

必須引数

package
SQLSTMT パッケージのインスタンスを指定します。

要件 FedSQL ステートメントは、ステートメントが文字列変数に保存されていない限り、単一引用符(')で囲む必要があります。

注 ステートメントは文字列リテラルです。

FedSQL 言語の識別子のルールは、識別子の DS2 ルールではなく、SQLSTMT パッケージで使用される変数に適用されます。これは、FedSQL (DS2 ではない)が FedSQL ステートメントを含む文字列を解析するために発生します。

注意 **SQLSTMT パッケージで使用できるのは FedSQL ステートメントのみです。DBMS 固有の SQL は使用できません。** 詳細については、*SAS FedSQL 言語リファレンス*を参照してください。

詳細

BULKINSERTFLUSH メソッドは、現在の行バッファー内のすべての行のバルク挿入を実行します。

現在の行バッファーの暗黙的なフラッシュは、次のいずれかのシナリオで発生します。

- バルク挿入行バッファーがいっぱいです。
- SQL ステートメントが削除されます。

BULKINSERTFLUSH メソッドはステータスインジケーターを返します。正常に実行されるとゼロが返され、エラーが発生した場合は 1 が返されます。

BULKINSERTBINDCOLUMNS メソッドを呼び出す前に BULKINSERTFLUSH メソッドを呼び出すと、ランタイムエラーが発生します。

関連項目

“BULKINSERTBINDCOLUMNS メソッド”

BULKINSERTGETCOLUMNCOUNT メソッド

バルク挿入用に選択された列の数を返します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

構文

package.**BULKINSERTGETCOLUMNCOUNT**();

必須引数

package
SQLSTMT パッケージのインスタンスを指定します。

要件 FedSQL ステートメントは、ステートメントが文字列変数に保存されていない限り、単一引用符(')で囲む必要があります。

注 ステートメントは文字列リテラルです。

FedSQL 言語の識別子のルールは、識別子の DS2 ルールではなく、SQLSTMT パッケージで使用される変数に適用されます。これは、FedSQL (DS2 ではない)が FedSQL ステートメントを含む文字列を解析するために発生します。

注意 **SQLSTMT パッケージで使用できるのは FedSQL ステートメントのみです。DBMS 固有の SQL は使用できません。** 詳細については、*SAS FedSQL 言語リファレンス*を参照してください。

詳細

バルク挿入ステートメントが BULKINSERTPREPARE メソッドによって準備される前に BULKINSERTGETCOLUMNCOUNT メソッドを呼び出すと、ランタイムエラーが発生します。

関連項目

- "BULKINSERTISPREPARED メソッド"
- "BULKINSERTPREPARE メソッド"

BULKINSERTGETCOLUMNNAME メソッド

指定されたインデックスのバルク挿入列の列名を返します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

構文

形式 1: *package*.**BULKINSERTGETCOLUMNNAME**(*index*);

形式 2: *package*.**BULKINSERTGETCOLUMNNAME**(*index*,*variable*,*rc*);

必須引数

index
返す列を指定します。列インデックスは順番に並べられ、1 から始まります。

package
SQLSTMT パッケージのインスタンスを指定します。

要件 FedSQL ステートメントは、ステートメントが文字列変数に保存されていない限り、単一引用符(')で囲む必要があります。

注 ステートメントは文字列リテラルです。

FedSQL 言語の識別子のルールは、識別子の DS2 ルールではなく、SQLSTMT パッケージで使用される変数に適用されます。これは、FedSQL (DS2 ではない)が FedSQL ステートメントを含む文字列を解析するために発生します。

注意 **SQLSTMT パッケージで使用できるのは FedSQL ステートメントのみです。DBMS 固有の SQL は使用できません。** 詳細については、*SAS FedSQL 言語リファレンス*を参照してください。

rc
リターンコードを配置する変数を指定します。次の値が可能です。

0
(SUCCESS)結果の列名が返されます。

1
(ERROR)データ取得中にエラーが発生しました。

variable
返された列名を保持する変数を指定します。

詳細

バルク挿入ステートメントが BULKINSERTPREPARE メソッドによって準備される前に BULKINSERTGETCOLUMNNAME メソッドを呼び出すと、ランタイムエラーが発生します。

関連項目

- BULKINSERTISPREPARED
- “BULKINSERTPREPARE メソッド”

BULKINSERTGETCOLUMNTYPENAME メソッド

指定された列インデックスのバルク挿入列の型名を返します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

構文

形式 1: *package*.**BULKINSERTGETCOLUMNTYPENAME**(*index*);

package.**BULKINSERTGETCOLUMNTYPENAME**(*index*,*variable*,*rc*);

必須引数

index
返す列を指定します。列インデックスは順番に並べられ、1 から始まります。

package
SQLSTMT パッケージのインスタンスを指定します。

要件 FedSQL ステートメントは、ステートメントが文字列変数に保存されていない限り、単一引用符(')で囲む必要があります。

注 ステートメントは文字列リテラルです。

FedSQL 言語の識別子のルールは、識別子の DS2 ルールではなく、SQLSTMT パッケージで使用される変数に適用されます。これは、FedSQL (DS2 ではない)が FedSQL ステートメントを含む文字列を解析するために発生します。

注意 **SQLSTMT パッケージで使用できるのは FedSQL ステートメントのみです。DBMS 固有の SQL は使用できません。** 詳細については、*SAS FedSQL 言語リファレンス*を参照してください。

rc
リターンコードを配置する変数を指定します。次の値が可能です。

0
(SUCCESS)結果の列の型名が返されます。

1
(FAILURE)データ取得中にエラーが発生しました。

variable
返された列の型名を保持する変数を指定します。

詳細

バルク挿入ステートメントが BULKINSERTPREPARE メソッドによって準備される前に BULKINSERTGETCOLUMNTYPENAME メソッドを呼び出すと、ランタイムエラーが発生します。

関連項目

- “BULKINSERTISPREPARED メソッド”
- “BULKINSERTPREPARE メソッド”

BULKINSERTISPREPARED メソッド

SQLSTMT パッケージインスタンスでバルク挿入の準備ができているかどうかを示す値を返します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

構文

package.**BULKINSERTISPREPARED**();

必須引数

package
SQLSTMT パッケージのインスタンスを指定します。

要件 FedSQL ステートメントは、ステートメントが文字列変数に保存されていない限り、単一引用符(')で囲む必要があります。

注 ステートメントは文字列リテラルです。

FedSQL 言語の識別子のルールは、識別子の DS2 ルールではなく、SQLSTMT パッケージで使用される変数に適用されます。これは、FedSQL (DS2 ではない)が FedSQL ステートメントを含む文字列を解析するために発生します。

注意 **SQLSTMT パッケージで使用できるのは FedSQL ステートメントのみです。DBMS 固有の SQL は使用できません。** 詳細については、*SAS FedSQL 言語リファレンス*を参照してください。

詳細

BULKINSERTISPREPARED メソッドはステータスインジケーターを返します。SQLSTMT インスタンでバルク挿入の準備ができていない場合は、ゼロ(FALSE)が返されます。SQLSTMT パッケージインスタンスでバルク挿入の準備ができている場合は、ゼロ以外の値(TRUE)が返されます。

関連項目

“BULKINSERTPREPARE メソッド”

BULKINSERTPREPARE メソッド

バルク挿入用に SQLSTMT パッケージインスタンスを準備します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

構文

package.**BULKINSERTPREPARE**(*select-string*[, *connection-string*]);

必須引数

package
SQLSTMT パッケージのインスタンスを指定します。

要件 FedSQL ステートメントは、ステートメントが文字列変数に保存されていない限り、単一引用符(')で囲む必要があります。

注 ステートメントは文字列リテラルです。

FedSQL 言語の識別子のルールは、識別子の DS2 ルールではなく、SQLSTMT パッケージで使用される変数に適用されます。これは、FedSQL (DS2 ではない)が FedSQL ステートメントを含む文字列を解析するために発生します。

注意 **SQLSTMT パッケージで使用できるのは FedSQL ステートメントのみです。DBMS 固有の SQL は使用できません。** 詳細については、*SAS FedSQL 言語リファレンス*を参照してください。

select-string
FedSQL SELECT ステートメントを含む文字列を指定します。文字列は文字列リテラルにすることができます。

オプション引数

connection-string
完全に指定された接続文字列が含まれます。

デフォルト 接続文字列が提供されていない場合、SQLSTMT パッケージインスタンスは、現在割り当てられている libref の属性を使用して、HPDS2 プロシジャまたは DS2 プロシジャによって 生成された接続文字列を使用します。

詳細

select-string 引数で指定された FEDSQL SELECT ステートメントによって選択された列が、バルク挿入の列として選択されます。

BULKINSERTPREPARE メソッドはステータスインジケーターを返します。正常に実行されるとゼロが返され、エラーが発生した場合は 1 が返されます。

関連項目

“BULKINSERTPREPARE メソッド”

CLOSERESULTS メソッド

ステートメントの最後の実行から結果セットを解放します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

構文

package.**CLOSERESULTS**( );

必須引数

package
SQLSTMT パッケージのインスタンスを指定します。

詳細

SQLSTMT インスタンスは、1 つの結果セットのみを維持します。FedSQL ステートメントが実行または削除されると、結果セットが自動的に解放されます。

CLOSERESULTS メソッドはステータスインジケーターを返します。正常に実行されるとゼロが返され、エラーが発生した場合は 1 が返されます。

関連項目

“SQLSTMT パッケージの使用” (*SAS DS2 プログラマガイド*)

DECLARE PACKAGE ステートメント: SQLSTMT パッケージ

パッケージ変数を作成し、SQLSTMT パッケージのインスタンスを作成できるようにします。

カテゴリ: ローカル

制限事項: このステートメントは CAS サーバーではサポートされていません。

注: 構文規則の中かっこは、構文のグループ化を示します。中かっこの前のエスケープ文字(\)は、構文に中かっこが必要であることを示します。*sql-text* は中かっこ({ })で囲む必要があります。

構文

形式 1: **DECLARE PACKAGE SQLSTMT** *variable* [('*sql-text*' [,\[*parameter-variable-list*\]])];

形式 2: **DECLARE PACKAGE SQLSTMT** *variable* [('*sql-text*' [, *connection-string*])];

形式 3: **DECLARE PACKAGE SQLSTMT** *variable* ( );

形式 4: **DECLARE PACKAGE SQLSTMT** *variable* ;

必須引数

variable
SQLSTMT パッケージのインスタンスを参照できる名前を指定します。

オプション引数

'*sql-text*'
テーブルの行の挿入、更新、選択、削除を行う FedSQL ステートメントを含む有効な FedSQL ステートメントまたは文字列変数です。

注意
SQLSTMT パッケージで使用できるのは FedSQL ステートメントのみです。DBMS 固有の SQL は使用できません。 詳細については、*SAS FedSQL 言語リファレンス*を参照してください。

要件 FedSQL ステートメントは、ステートメントが文字列変数に保存されていない限り、単一引用符(')で囲む必要があります。

注 ステートメントは文字列リテラルです。

FedSQL 言語の識別子のルールは、識別子の DS2 ルールではなく、SQLSTMT パッケージで使用される変数に適用されます。これは、FedSQL

が DS2 ではなく SQL ステートメントを含む文字列を解析するために発生します。

[*parameter-variable-list*]
FedSQL ステートメントに含まれるパラメーターにバインドされる変数を指定します。

要件 変数は、角かっこ([])で囲む必要がある変数リストまたは名前付き変数リストの形式である必要があります。

パラメーターデータは、バインドされた変数または SQLSTMT SET*type* メソッドのいずれかで排他的に指定する必要があります。

参照項目: "FedSQL ステートメントのパラメーター値の指定" (*SAS DS2 プログラマガイド*)

connection-string
完全に指定された接続文字列が含まれます。

デフォルト 接続文字列が提供されていない場合、SQLSTMT パッケージインスタンスは、現在割り当てられている libref の属性を使用して、HPDS2 プロシジャまたは DS2 プロシジャによって 生成された接続文字列を使用します。

注 接続文字列は文字列リテラルです。

ヒント 接続文字列は、データへの接続方法を定義します。接続文字列は、サブミットするクエリ言語と、1 つまたは複数のデータソースに接続するために必要な情報を識別します。

参照項目: このパラメーターは、主に SAS Federation Server で使用するために設計されています。完全に指定された接続文字列の作成の詳細については、*SAS Federation Server: 管理者ガイド*を参照してください。

詳細

DS2 パッケージは、特定のインスタンスを構築して他の DS2 プログラムで使用できる変数とメソッドのコレクションです。

SQLSTMT パッケージは、FedSQL ステートメントを DBMS に渡して実行する方法を提供します。FedSQL ステートメントは、テーブルを作成、変更、または削除する可能性があります。FedSQL ステートメントがテーブルから行を選択する場合、SQLSTMT パッケージは、結果セットで返された行を問い合わせるためのメソッドを提供します。SQLSTMT パッケージは、DS2 プログラム用に事前定義されています。

SQLSTMT パッケージは、DECLARE PACKAGE ステートメントを使用して宣言します。これにより、SQLSTMT パッケージが SQLSTMT 名に関連付けられます。

SQLSTMT パッケージのインスタンスを構築する方法は 2 つあります。

- DECLARE PACKAGE ステートメントを_NEW_演算子と一緒に使用します。

```
declare package sqlstmt sqlpkg;
sqlpkg = _new_ sqlstmt('update db2.dataset2 set y=? where x=?',[y x]);
```

- DECLARE PACKAGE ステートメントをそのコンストラクター構文と一緒に使用します。

  ```
  declare package sqlstmt sqlpkg('update db2.dataset2 set y=? where x=?',[y x]);
  ```

DECLARE ステートメントのかっこ内に構築用の引数が含まれている場合(および SQLSTMT パッケージに有効な引数がない場合)、パッケージインスタンスが割り当てられます。かっこが含まれていない場合、変数は作成されますが、パッケージインスタンスは割り当てられません。

SQL テキストを使用して SQLSTMT インスタンスを作成すると(形式および 2)、FedSQL ステートメントが割り当てられて準備されます。FedSQL ステートメントにパラメーターが含まれている場合、FedSQL ステートメントを実行するには、パラメーターと置き換える値を取得する必要があります。置換値は、バインドされた変数の現在の値または SQLSTMT パッケージの SET*type* メソッドのいずれかを使用して指定できます。DECLARE PACKAGE ステートメントは、オプションの変数リストの変数を FedSQL ステートメントのパラメーターにバインドします。

FedSQL テキストなしで SQLSTMT インスタンスが作成されると(形式 3)、SQLSMT インスタンスが割り当てられ、準備されていない状態のままになります。PREPARE メソッドを使用して、後で FedSQL ステートメントを準備します。

注: パッケージ変数は、すべての変数スコーピングルールの対象となります。詳細については、“パッケージ、スコープ、および有効期間” (*SAS DS2 プログラマガイド*)を参照してください。

SQLSTMT パッケージの詳細については、“SQLSTMT パッケージの使用” (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- “SQLSTMT パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “BINDPARAMETERS メソッド” (1944 ページ)
- “PREPARE メソッド” (1994 ページ)
- この章の SET "type"メソッド

演算子:

- “_NEW_ 演算子: SQLSTMT パッケージ” (1991 ページ)

DELETE メソッド: SQLSTMT パッケージ

SQLSTMT パッケージのインスタンスを削除します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

注: DELETE メソッドは必須ではありません。SQLSTMT パッケージインスタンスを参照する SQLSTMT パッケージ変数がない場合、パッケージインスタンスは自動的に削除されます。

構文

package.**DELETE**( );

必須引数

package
SQLSTMT パッケージ変数のインスタンスを指定します。

詳細

SQLSTMT パッケージが不要になった場合は、DELETE メソッドを使用して削除します。SQLSTMT パッケージを削除した後に使用しようとすると、ログにエラーが書き込まれます。

関連項目

- "SQLSTMT パッケージの使用" (*SAS DS2 プログラマガイド*)
- "パッケージのコンストラクターとデストラクター" (*SAS DS2 プログラマガイド*)

EXECUTE メソッド

FedSQL ステートメントを実行します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

構文

package.**EXECUTE**( );

必須引数

package
SQLSTMT パッケージのインスタンスを指定します。

詳細

EXECUTE メソッドは、FedSQL ステートメントを実行し、ステータスインジケーターを返します。正常に実行されるとゼロ、エラーが発生した場合は 1、データがない場合(NODATA)は 2 が返されます。NODATA 条件は、FedSQL UPDATE または DELETE ステートメントがどの行にも影響を与えない場合に存在します。

SQLSTMT インスタンスは、1 つの結果セットのみを維持します。前回の実行の結果セットがある場合は、FedSQL ステートメントが実行される前に解放されます。

FedSQL ステートメントは、実行時に動的に実行されます。ステートメントは実行時に準備されるため、DS2 プログラムの実行中に動的に構築およびカスタマイズできます。

関連項目

"SQLSTMT パッケージの使用" (*SAS DS2 プログラマガイド*)

FETCH メソッド

FedSQL ステートメントの結果セットから次のデータ行をフェッチします。

制限事項: このメソッドは CAS サーバーではサポートされていません。

構文

package.**FETCH** ([*result-variable-list*]);

必須引数

package
SQLSTMT パッケージのインスタンスを指定します。

[*result-variable-list*]
結果セットの列にバインドする変数を含む変数リストを指定します。

詳細

FETCH メソッドは、結果セットから次のデータ行を返します。ステータスインジケーターが返されます。正常に実行されるとゼロ、エラーが発生した場合は 1、データがない場合(NODATA)は 2 が返されます。フェッチされる次の行が結果セットの終了後にある場合、NODATA 条件が存在します。

変数が BINDRESULTS メソッドまたは FETCH メソッドで結果セットの列にバインドされている場合、各結果セットの列のフェッチされたデータは、その列にバインドされている変数に配置されます。変数が結果セットの列にバインドされていない場合、フェッチされたデータは GET*type* メソッドで返すことができます。

ステートメントが実行される前に FETCH が呼び出されると、ランタイムエラーが発生します。

SQLSTMT インスタンスは、1 つの結果セットのみを維持します。前回の実行の結果セットがある場合は、FedSQL ステートメントが実行される前に解放されます。また、CLOSERESULTS メソッドを使用して、いつでも結果セットを解放することができます。

関連項目

- “SQLSTMT パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “BINDRESULTS メソッド” (1946 ページ)
- この章の GET "type"メソッド

GETBIGINT メソッド

指定された結果セット列の値を BIGINT 型として返します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

注: GETBIGINT メソッドを 1 回呼び出すと、結果セット列データを返すことができます。後続のメソッド呼び出しの結果、*rc* ステータスインジケーターの値は 2 (NODATA)になります。

構文

variable=*package*.**GETBIGINT** (*index*);

package.**GETBIGINT** (*index*, *variable*, *rc*);

必須引数

variable
指定された結果セット列の値を保持する変数を指定します。

注 指定された結果セット列の型が BIGINT 型でない場合、列の値は BIGINT 型に変換されてから返されます。

package
SQLSTMT パッケージのインスタンスを指定します。

index
1 から採番された結果セット列インデックスを指定します。

rc
リターンコードを配置する変数を指定します。次の値が可能です。

0
(SUCCESS)結果セットの列データが取得されます。

1
(ERROR)データ取得中にエラーが発生しました。

2
(NODATA)取得するデータはこれ以上ありません。

詳細

FETCH メソッドは、結果セットから次のデータ行を返します。変数が結果セットの列にバインドされていない場合、フェッチされたデータは GET*type* メソッドで返すことができます。

GETBIGINT メソッドは、指定された結果セット列の値を BIGINT 型として返します。指定された結果セット列の型が BIGINT 型でない場合、列の値は BIGINT 型に変換されてから返されます。

結果セットがフェッチされる前に GETBIGINT メソッドが呼び出されると、ランタイムエラーが発生します。

結果セットには、バインドされた変数または GET*type* メソッドで排他的にアクセスする必要があります。変数が結果セットの列にバインドされ、GET*type* メソッドが呼び出されると、ランタイムエラーが発生します。

詳細については、"結果セットデータへのアクセス" (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- "SQLSTMT パッケージの使用" (*SAS DS2 プログラマガイド*)

メソッド:

- "GETINTEGER メソッド" (1974 ページ)
- "GETSMALLINT メソッド" (1981 ページ)
- "GETTINYINT メソッド" (1986 ページ)
- "SETBIGINT メソッド" (1997 ページ)
- "SETINTEGER メソッド" (2005 ページ)
- "SETSMALLINT メソッド" (2011 ページ)

■ “SETTINYINT メソッド” (2015 ページ)

GETBINARY メソッド

指定された結果セット列を BINARY 型として返します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

注: GETBINARY メソッドを繰り返し呼び出して、結果セット列データを返すことができます。すべてのデータが取得されると、*rc* ステータスインジケーターに値 2 (NODATA)が返されます。

構文

variable=*package*.**GETBINARY** (*index*);

package.**GETBINARY** (*index*, *variable*, *rc*);

必須引数

variable
指定された結果セット列の値を保持する変数を指定します。

注 指定された結果セット列の型が BINARY 型でない場合、列の値は BINARY 型に変換されてから返されます。

package
SQLSTMT パッケージのインスタンスを指定します。

index
1 から採番された結果セット列インデックスを指定します。

rc
リターンコードを配置する変数を指定します。次の値が可能です。

0
(SUCCESS)結果セットの列データが取得されます。

1
(ERROR)データ取得中にエラーが発生しました。

2
(NODATA)取得するデータはこれ以上ありません。

詳細

FETCH メソッドは、結果セットから次のデータ行を返します。変数が結果セットの列にバインドされていない場合、フェッチされたデータは GET*type* メソッドで返すことができます。

GETBINARY メソッドは、指定された結果セット列の値を BINARY 型として返します。指定された結果セット列の型が BINARY 型でない場合、列の値は BINARY 型に変換されてから返されます。

結果セットがフェッチされる前に GETBINARY メソッドが呼び出されると、ランタイムエラーが発生します。

結果セットには、バインドされた変数または GET*type* メソッドで排他的にアクセスする必要があります。変数が結果セットの列にバインドされ、GET*type* メソッドが呼び出されると、ランタイムエラーが発生します。

詳細については、“結果セットデータへのアクセス” (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- “SQLSTMT パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “GETVARBINARY メソッド” (1987 ページ)
- “SETBINARY メソッド” (1999 ページ)
- “SETVARBINARY メソッド” (2017 ページ)

GETCHAR メソッド

指定された結果セット列を CHAR 型として返します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

注: GETBINARY メソッドを繰り返し呼び出して、結果セット列データを返すことができます。すべてのデータが取得されると、*rc* ステータスインジケーターに値 2 (NODATA)が返されます。

構文

variable=*package*.**GETCHAR** (*index*);

package.**GETCHAR** (*index*, *variable*, *rc*);

必須引数

variable
指定された結果セット列の値を保持する変数を指定します。

注 指定された結果セット列の型が CHAR 型でない場合、列の値は CHAR 型に変換されてから返されます。

package
SQLSTMT パッケージのインスタンスを指定します。

index
1 から採番された結果セット列インデックスを指定します。

rc
リターンコードを配置する変数を指定します。次の値が可能です。

0
(SUCCESS)結果セットの列データが取得されます。

1
(ERROR)データ取得中にエラーが発生しました。

2
(NODATA)取得するデータはこれ以上ありません。

詳細

FETCH メソッドは、結果セットから次のデータ行を返します。変数が結果セットの列にバインドされていない場合、フェッチされたデータは GET*type* メソッドで返すことができます。

GETCHAR メソッドは、指定された結果セット列の値を CHAR 型として返します。指定された結果セット列の型が BIGINT 型でない場合、列の値は CHAR 型に変換されてから返されます。

結果セットがフェッチされる前に GETCHAR メソッドが呼び出されると、ランタイムエラーが発生します。

結果セットには、バインドされた変数または GET*type* メソッドで排他的にアクセスする必要があります。変数が結果セットの列にバインドされ、GET*type* メソッドが呼び出されると、ランタイムエラーが発生します。

詳細については、“結果セットデータへのアクセス” (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- “SQLSTMT パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “GETNCHAR メソッド” (1975 ページ)
- “GETNVARCHAR メソッド” (1978 ページ)
- “GETVARCHAR メソッド” (1989 ページ)
- “SETCHAR メソッド” (2000 ページ)
- “SETNCHAR メソッド” (2006 ページ)
- “SETNVARCHAR メソッド” (2009 ページ)

- “SETVARCHAR メソッド” (2018 ページ)

GETCOLUMNCOUNT メソッド

結果セットの列数を返します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

構文

variable=*package*.**GETCOLUMNCOUNT**();

必須引数

variable
結果セットの列数を保持する変数を指定します。

package
SQLSTMT パッケージのインスタンスを指定します。

詳細

SQLSTMT が準備される前に GETCOLUMNCOUNT メソッドが呼び出されると、ランタイムエラーが発生します。

関連項目

メソッド:

- “GETCOLUMNNAME メソッド” (1967 ページ)
- “GETCOLUMNTYPENAME メソッド” (1968 ページ)

GETCOLUMNNAME メソッド

指定されたインデックスを持つ結果セット列の列名を返します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

構文

variable=*package*.**GETCOLUMNNAME**(*index*);

package.getColumnName(*index*, *variable*, *rc*);

必須引数

variable
指定された結果セット列の名前を保持する変数を指定します。

package
SQLSTMT パッケージのインスタンスを指定します。

index
1 から採番された結果セット列インデックスを指定します。

rc
リターンコードを配置する変数を指定します。次の値が可能です。

0
(SUCCESS)結果セットの列データが取得されます。

1
(ERROR)データ取得中にエラーが発生しました。

詳細

SQLSTMT が準備される前に GETCOLUMNNAME メソッドが呼び出されると、ランタイムエラーが発生します。

関連項目

メソッド:

- “GETCOLUMNCOUNT メソッド” (1967 ページ)
- “GETCOLUMNTYPENAME メソッド” (1968 ページ)

GETCOLUMNTYPENAME メソッド

指定されたインデックスを持つ結果セット列のデータ型を返します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

構文

variable=*package*.**GETCOLUMNTYPENAME**(*index*);

package.getColumnTypeName(*index*, *variable*, *rc*);

必須引数

variable
結果セット列のデータ型を保持する変数を指定します。

package
SQLSTMT パッケージのインスタンスを指定します。

index
1 から採番された結果セット列インデックスを指定します。

rc
リターンコードを配置する変数を指定します。次の値が可能です。

0
(SUCCESS)結果セットの列データが取得されます。

1
(ERROR)データ取得中にエラーが発生しました。

詳細

GETCOLUMNTYPENAME は、DS2 でサポートされているデータ型のみを返します。詳細については、“DS2 データ型” (*SAS DS2 プログラマガイド*)を参照してください。

SQLSTMT が準備される前に GETCOLUMNTYPENAME メソッドが呼び出されると、ランタイムエラーが発生します。

関連項目

メソッド:

- “GETCOLUMNCOUNT メソッド” (1967 ページ)
- “GETCOLUMNTYPENAME メソッド” (1968 ページ)

GETDATE メソッド

指定された結果セット列を DATE 型として返します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

注: GETDATE メソッドを 1 回呼び出すと、結果セット列データを返すことができます。後続のメソッド呼び出しの結果、*rc* ステータスインジケーターの値は 2 (NODATA) になります。

構文

variable=*package*.**GETDATE** (*index*);

package.**GETDATE** (*index*, *variable*, *rc*);

必須引数

variable
指定された結果セット列の値を保持する変数を指定します。

注 指定された結果セット列の型が DATE 型でない場合、列の値は DATE 型に変換されてから返されます。

package
SQLSTMT パッケージのインスタンスを指定します。

index
1 から採番された結果セット列インデックスを指定します。

rc
リターンコードを配置する変数を指定します。次の値が可能です。

0
(SUCCESS)結果セットの列データが取得されます。

1
(ERROR)データ取得中にエラーが発生しました。

2
(NODATA)取得するデータはこれ以上ありません。

詳細

FETCH メソッドは、結果セットから次のデータ行を返します。変数が結果セットの列にバインドされていない場合、フェッチされたデータは GET*type* メソッドで返すことができます。

GETDATE メソッドは、指定された結果セット列の値を DATE 型として返します。指定された結果セット列の型が DATE 型でない場合、列の値は DATE 型に変換されてから返されます。

結果セットがフェッチされる前に GETDATE メソッドが呼び出されると、ランタイムエラーが発生します。

結果セットには、バインドされた変数または GET*type* メソッドで排他的にアクセスする必要があります。変数が結果セットの列にバインドされ、GET*type* メソッドが呼び出されると、ランタイムエラーが発生します。

詳細については、“結果セットデータへのアクセス” (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- “SQLSTMT パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “GETTIME メソッド” (1983 ページ)
- “GETTIMESTAMP メソッド” (1984 ページ)
- “SETDATE メソッド” (2001 ページ)
- “SETTIME メソッド” (2013 ページ)
- “SETTIMESTAMP メソッド” (2014 ページ)

GETDECIMAL メソッド

指定された結果セット列を DECIMAL 型として返します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

注: GETDECIMAL メソッドを 1 回呼び出すと、結果セット列データを返すことができます。後続のメソッド呼び出しの結果、*rc* ステータスインジケーターの値は 2 (NODATA)になります。

構文

variable=*package*.**GETDECIMAL** (*index*);

package.**GETDECIMAL** (*index*, *variable*, *rc*);

必須引数

variable
指定された結果セット列の値を保持する変数を指定します。

注 指定された結果セット列の型が DECIMAL 型でない場合、列の値は DECIMAL 型に変換されてから返されます。

package
SQLSTMT パッケージのインスタンスを指定します。

index
1 から採番された結果セット列インデックスを指定します。

rc
リターンコードを配置する変数を指定します。次の値が可能です。

0
(SUCCESS)結果セットの列データが取得されます。

1
(ERROR)データ取得中にエラーが発生しました。

2
(NODATA)取得するデータはこれ以上ありません。

詳細

FETCH メソッドは、結果セットから次のデータ行を返します。変数が結果セットの列にバインドされていない場合、フェッチされたデータは GET*type* メソッドで返すことができます。

GETDECIMAL メソッドは、指定された結果セット列の値を DECIMAL 型として返します。指定された結果セット列の型が DECIMAL 型でない場合、列の値は DECIMAL 型に変換されてから返されます。

結果セットがフェッチされる前に GETDECIMAL メソッドが呼び出されると、ランタイムエラーが発生します。

結果セットには、バインドされた変数または GET*type* メソッドで排他的にアクセスする必要があります。変数が結果セットの列にバインドされ、GET*type* メソッドが呼び出されると、ランタイムエラーが発生します。

詳細については、“結果セットデータへのアクセス” (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- “SQLSTMT パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “SETDECIMAL メソッド” (2002 ページ)

GETDOUBLE メソッド

指定された結果セット列を DOUBLE 型として返します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

注: GETDOUBLE メソッドを 1 回呼び出すと、結果セット列データを返すことができます。後続のメソッド呼び出しの結果、*rc* ステータスインジケーターの値は 2 (NODATA)になります。

構文

variable=*package*.**GETDOUBLE** (*index*);

package.**GETDOUBLE** (*index*, *variable*, *rc*);

必須引数

variable
指定された結果セット列の値を保持する変数を指定します。

注 指定された結果セット列の型が DOUBLE 型でない場合、列の値は DOUBLE 型に変換されてから返されます。

package
SQLSTMT パッケージのインスタンスを指定します。

index
1 から採番された結果セット列インデックスを指定します。

rc
リターンコードを配置する変数を指定します。次の値が可能です。

0
(SUCCESS)結果セットの列データが取得されます。

1
(ERROR)データ取得中にエラーが発生しました。

2
(NODATA)取得するデータはこれ以上ありません。

詳細

FETCH メソッドは、結果セットから次のデータ行を返します。変数が結果セットの列にバインドされていない場合、フェッチされたデータは GET*type* メソッドで返すことができます。

GETDOUBLE メソッドは、指定された結果セット列の値を DOUBLE 型として返します。指定された結果セット列の型が DOUBLE 型でない場合、列の値は DOUBLE 型に変換されてから返されます。

結果セットがフェッチされる前に GETDOUBLE メソッドが呼び出されると、ランタイムエラーが発生します。

結果セットには、バインドされた変数または GET*type* メソッドで排他的にアクセスする必要があります。変数が結果セットの列にバインドされ、GET*type* メソッドが呼び出されると、ランタイムエラーが発生します。

詳細については、“結果セットデータへのアクセス” (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- “SQLSTMT パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “SETDOUBLE メソッド” (2004 ページ)

GETINTEGER メソッド

指定された結果セット列を INTEGER 型として取得します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

注: GETINTEGER メソッドを 1 回呼び出すと、結果セット列データを返すことができます。後続のメソッド呼び出しの結果、*rc* ステータスインジケーターの値は 2 (NODATA)になります。

構文

variable=*package*.**GETINTEGER** (*index*);

package.**GETINTEGER** (*index*, *variable*, *rc*);

必須引数

variable
指定された結果セット列の値を保持する変数を指定します。

注 指定された結果セット列の型が INTEGER 型でない場合、列の値は INTEGER 型に変換されてから返されます。

package
SQLSTMT パッケージのインスタンスを指定します。

index
1 から採番された結果セット列インデックスを指定します。

rc
リターンコードを配置する変数を指定します。次の値が可能です。

0
(SUCCESS)結果セットの列データが取得されます。

1
(ERROR)データ取得中にエラーが発生しました。

2
(NODATA)取得するデータはこれ以上ありません。

詳細

FETCH メソッドは、結果セットから次のデータ行を返します。変数が結果セットの列にバインドされていない場合、フェッチされたデータは GET*type* メソッドで返すことができます。

GETINTEGER メソッドは、指定された結果セット列の値を INTEGER 型として返します。指定された結果セット列の型が INTEGER 型でない場合、列の値は INTEGER 型に変換されてから返されます。

結果セットがフェッチされる前に GETINTEGER メソッドが呼び出されると、ランタイムエラーが発生します。

結果セットには、バインドされた変数または GET*type* メソッドで排他的にアクセスする必要があります。変数が結果セットの列にバインドされ、GET*type* メソッドが呼び出されると、ランタイムエラーが発生します。

詳細については、“結果セットデータへのアクセス” (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- “SQLSTMT パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “GETBIGINT メソッド” (1962 ページ)
- “GETSMALLINT メソッド” (1981 ページ)
- “GETTINYINT メソッド” (1986 ページ)
- “SETBIGINT メソッド” (1997 ページ)
- “SETINTEGER メソッド” (2005 ページ)
- “SETSMALLINT メソッド” (2011 ページ)
- “SETTINYINT メソッド” (2015 ページ)

GETNCHAR メソッド

指定された結果セット列を NCHAR 型として取得します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

注: GETBINARY メソッドを繰り返し呼び出して、結果セット列データを返すことができます。すべてのデータが取得されると、*rc* ステータスインジケーターに値 2 (NODATA)が返されます。

構文

variable=*package*.**GETNCHAR** (*index*);

package.**GETNCHAR** (*index*, *variable*, *rc*);

必須引数

variable
指定された結果セット列の値を保持する変数を指定します。

注 指定された結果セット列の型が NCHAR 型でない場合、列の値は NCHAR 型に変換されてから返されます。

package
SQLSTMT パッケージのインスタンスを指定します。

index
1 から採番された結果セット列インデックスを指定します。

rc
リターンコードを配置する変数を指定します。次の値が可能です。

0
(SUCCESS)結果セットの列データが取得されます。

1
(ERROR)データ取得中にエラーが発生しました。

2
(NODATA)取得するデータはこれ以上ありません。

詳細

FETCH メソッドは、結果セットから次のデータ行を返します。変数が結果セットの列にバインドされていない場合、フェッチされたデータは GET*type* メソッドで返すことができます。

GETNCHAR メソッドは、指定された結果セット列の値を NCHAR 型として返します。指定された結果セット列の型が NCHAR 型でない場合、列の値は NCHAR 型に変換されてから返されます。

結果セットがフェッチされる前に GETNCHAR メソッドが呼び出されると、ランタイムエラーが発生します。

結果セットには、バインドされた変数または GET*type* メソッドで排他的にアクセスする必要があります。変数が結果セットの列にバインドされ、GET*type* メソッドが呼び出されると、ランタイムエラーが発生します。

詳細については、“結果セットデータへのアクセス” (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- “SQLSTMT パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “GETCHAR メソッド” (1965 ページ)
- “GETNVARCHAR メソッド” (1978 ページ)
- “GETVARCHAR メソッド” (1989 ページ)
- “SETCHAR メソッド” (2000 ページ)
- “SETNCHAR メソッド” (2006 ページ)
- “SETNVARCHAR メソッド” (2009 ページ)
- “SETVARCHAR メソッド” (2018 ページ)

GETNUMERIC メソッド

指定された結果セット列を NUMERIC 型として返します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

注: GETNUMERIC メソッドを 1 回呼び出すと、結果セット列データを返すことができます。後続のメソッド呼び出しの結果、*rc* ステータスインジケーターの値は 2 (NODATA)になります。

構文

variable=*package*.**GETNUMERIC** (*index*);

package.**GETNUMERIC** (*index*, *variable*, *rc*);

必須引数

variable
指定された結果セット列の値を保持する変数を指定します。

注 指定された結果セット列の型が NUMERIC 型でない場合、列の値は NUMERIC 型に変換されてから返されます。

package
SQLSTMT パッケージのインスタンスを指定します。

index
1 から採番された結果セット列インデックスを指定します。

rc
リターンコードを配置する変数を指定します。次の値が可能です。

0
(SUCCESS)結果セットの列データが取得されます。

1
(ERROR)データ取得中にエラーが発生しました。

2
(NODATA)取得するデータはこれ以上ありません。

詳細

FETCH メソッドは、結果セットから次のデータ行を返します。変数が結果セットの列にバインドされていない場合、フェッチされたデータは GET*type* メソッドで返すことができます。

GETNUMERIC メソッドは、指定された結果セット列の値を NUMERIC 型として返します。指定された結果セット列の型が NUMERIC 型でない場合、列の値は NUMERIC 型に変換されてから返されます。

結果セットがフェッチされる前に GETNUMERIC メソッドが呼び出されると、ランタイムエラーが発生します。

結果セットには、バインドされた変数または GET*type* メソッドで排他的にアクセスする必要があります。変数が結果セットの列にバインドされ、GET*type* メソッドが呼び出されると、ランタイムエラーが発生します。

詳細については、“結果セットデータへのアクセス” (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- “SQLSTMT パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “SETNUMERIC メソッド” (2008 ページ)

GETNVARCHAR メソッド

指定された結果セット列を NVARCHAR 型として返します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

注: GETVARCHAR メソッドを繰り返し呼び出して、結果セット列データを返すことができます。すべてのデータが取得されると、*rc* ステータスインジケーターに値 2 (NODATA)が返されます。

構文

variable=*package*.**GETNVARCHAR** (*index*);

package.**GETNVARCHAR** (*index*, *variable*, *rc*);

必須引数

variable
指定された結果セット列の値を保持する変数を指定します。

注 指定された結果セット列の型が NVARCHAR 型でない場合、列の値は NVARCHAR 型に変換されてから返されます。

package
SQLSTMT パッケージのインスタンスを指定します。

index
1 から採番された結果セット列インデックスを指定します。

rc
リターンコードを配置する変数を指定します。次の値が可能です。

0
(SUCCESS)結果セットの列データが取得されます。

1
(ERROR)データ取得中にエラーが発生しました。

2
(NODATA)取得するデータはこれ以上ありません。

詳細

FETCH メソッドは、結果セットから次のデータ行を返します。変数が結果セットの列にバインドされていない場合、フェッチされたデータは GET*type* メソッドで返すことができます。

GETNVARCHAR メソッドは、指定された結果セット列の値を NVARCHAR 型として返します。指定された結果セット列の型が NVARCHAR 型でない場合、列の値は NVARCHAR 型に変換されてから返されます。

結果セットがフェッチされる前に GETNVARCHAR メソッドが呼び出されると、ランタイムエラーが発生します。

結果セットには、バインドされた変数または GET*type* メソッドで排他的にアクセスする必要があります。変数が結果セットの列にバインドされ、GET*type* メソッドが呼び出されると、ランタイムエラーが発生します。

詳細については、“結果セットデータへのアクセス” (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- “SQLSTMT パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “GETCHAR メソッド” (1965 ページ)
- “GETNCHAR メソッド” (1975 ページ)
- “GETVARCHAR メソッド” (1989 ページ)
- “SETCHAR メソッド” (2000 ページ)
- “SETNCHAR メソッド” (2006 ページ)
- “SETNVARCHAR メソッド” (2009 ページ)
- “SETVARCHAR メソッド” (2018 ページ)

GETREAL メソッド

指定された結果セット列を REAL 型として返します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

注: GETREAL メソッドを 1 回呼び出すと、結果セット列データを返すことができます。後続のメソッド呼び出しの結果、*rc* ステータスインジケーターの値は 2 (NODATA) になります。

構文

variable=*package*.**GETREAL** (*index*);

package.**GETREAL** (*index*, *variable*, *rc*);

必須引数

variable
指定された結果セット列の値を保持する変数を指定します。

注 指定された結果セット列の型が REAL 型でない場合、列の値は REAL 型に変換されてから返されます。

package
SQLSTMT パッケージのインスタンスを指定します。

index
1 から採番された結果セット列インデックスを指定します。

rc
リターンコードを配置する変数を指定します。次の値が可能です。

0
(SUCCESS)結果セットの列データが取得されます。

1
(ERROR)データ取得中にエラーが発生しました。

2
(NODATA)取得するデータはこれ以上ありません。

詳細

FETCH メソッドは、結果セットから次のデータ行を返します。変数が結果セットの列にバインドされていない場合、フェッチされたデータは GET*type* メソッドで返すことができます。

GETREAL メソッドは、指定された結果セット列の値を REAL 型として返します。指定された結果セット列の型が REAL 型でない場合、列の値は REAL 型に変換されてから返されます。

結果セットがフェッチされる前に GETREAL メソッドが呼び出されると、ランタイムエラーが発生します。

結果セットには、バインドされた変数または GET*type* メソッドで排他的にアクセスする必要があります。変数が結果セットの列にバインドされ、GET*type* メソッドが呼び出されると、ランタイムエラーが発生します。

詳細については、“結果セットデータへのアクセス” (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- “SQLSTMT パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “SETREAL メソッド” (2010 ページ)

GETSMALLINT メソッド

指定された結果セット列を SMALLINT 型として返します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

注: GETSMALLINT メソッドを 1 回呼び出すと、結果セット列データを返すことができます。後続のメソッド呼び出しの結果、*rc* ステータスインジケーターの値は 2 (NODATA)になります。

構文

variable=*package*.**GETSMALLINT** (*index*);

package.**GETSMALLINT** (*index*, *variable*, *rc*);

必須引数

variable
指定された結果セット列の値を保持する変数を指定します。

注 指定された結果セット列の型が SMALLINT 型でない場合、列の値は SMALLINT 型に変換されてから返されます。

package
SQLSTMT パッケージのインスタンスを指定します。

index
1 から採番された結果セット列インデックスを指定します。

rc
リターンコードを配置する変数を指定します。次の値が可能です。

0
(SUCCESS)結果セットの列データが取得されます。

1
(ERROR)データ取得中にエラーが発生しました。

2
(NODATA)取得するデータはこれ以上ありません。

詳細

FETCH メソッドは、結果セットから次のデータ行を返します。変数が結果セットの列にバインドされていない場合、フェッチされたデータは GET*type* メソッドで返すことができます。

GETSMALLINT メソッドは、指定された結果セット列の値を SMALLINT 型として返します。指定された結果セット列の型が SMALLINT 型でない場合、列の値は SMALLINT 型に変換されてから返されます。

結果セットがフェッチされる前に GETSMALLINT メソッドが呼び出されると、ランタイムエラーが発生します。

結果セットには、バインドされた変数または GET*type* メソッドで排他的にアクセスする必要があります。変数が結果セットの列にバインドされ、GET*type* メソッドが呼び出されると、ランタイムエラーが発生します。

詳細については、“結果セットデータへのアクセス” (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- “SQLSTMT パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “GETBIGINT メソッド” (1962 ページ)
- “GETINTEGER メソッド” (1974 ページ)
- “GETTINYINT メソッド” (1986 ページ)
- “SETBIGINT メソッド” (1997 ページ)
- “SETINTEGER メソッド” (2005 ページ)
- “SETSMALLINT メソッド” (2011 ページ)
- “SETTINYINT メソッド” (2015 ページ)

GETTIME メソッド

指定された結果セット列を TIME 型として返します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

注: GETTIME メソッドを 1 回呼び出すと、結果セット列データを返すことができます。後続のメソッド呼び出しの結果、*rc* ステータスインジケーターの値は 2 (NODATA) になります。

構文

variable=*package*.**GETTIME** (*index*);

package.**GETTIME** (*index*, *variable*, *rc*);

必須引数

variable
指定された結果セット列の値を保持する変数を指定します。

注 指定された結果セット列の型が TIME 型でない場合、列の値は TIME 型に変換されてから返されます。

package
SQLSTMT パッケージのインスタンスを指定します。

index
1 から採番された結果セット列インデックスを指定します。

rc
リターンコードを配置する変数を指定します。次の値が可能です。

0
(SUCCESS)結果セットの列データが取得されます。

1
(ERROR)データ取得中にエラーが発生しました。

2
(NODATA)取得するデータはこれ以上ありません。

詳細

GETTIME メソッドは、指定された結果セット列の値を TIME 型として返します。指定された結果セット列の型が TIME 型でない場合、列の値は TIME 型に変換されてから返されます。

結果セットがフェッチされる前に GETTIME メソッドが呼び出されると、ランタイムエラーが発生します。

結果セットには、バインドされた変数または GET*type* メソッドで排他的にアクセスする必要があります。変数が結果セットの列にバインドされ、GET*type* メソッドが呼び出されると、ランタイムエラーが発生します。

詳細については、"結果セットデータへのアクセス" (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- "SQLSTMT パッケージの使用" (*SAS DS2 プログラマガイド*)

メソッド:

- "GETDATE メソッド" (1969 ページ)
- "GETTIMESTAMP メソッド" (1984 ページ)
- "SETDATE メソッド" (2001 ページ)
- "SETTIME メソッド" (2013 ページ)
- "SETTIMESTAMP メソッド" (2014 ページ)

GETTIMESTAMP メソッド

指定された結果セット列を TIMESTAMP 型として返します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

注: GETTIMESTAMP メソッドを 1 回呼び出すと、結果セット列データを返すことができます。後続のメソッド呼び出しの結果、*rc* ステータスインジケーターの値は 2 (NODATA)になります。

構文

variable=*package*.**GETTIMESTAMP** (*index*);

package.**GETTIMESTAMP** (*index*, *variable*, *rc*);

必須引数

variable
指定された結果セット列の値を保持する変数を指定します。

注 指定された結果セット列の型が TIMESTAMP 型でない場合、列の値は TIMESTAMP 型に変換されてから返されます。

package
SQLSTMT パッケージのインスタンスを指定します。

index
1 から採番された結果セット列インデックスを指定します。

rc
リターンコードを配置する変数を指定します。次の値が可能です。

0
(SUCCESS)結果セットの列データが取得されます。

1
(ERROR)データ取得中にエラーが発生しました。

2
(NODATA)取得するデータはこれ以上ありません。

詳細

FETCH メソッドは、結果セットから次のデータ行を返します。変数が結果セットの列にバインドされていない場合、フェッチされたデータは GET*type* メソッドで返すことができます。

GETTIMESTAMP メソッドは、指定された結果セット列の値を TIMESTAMP 型として返します。指定された結果セット列の型が TIMESTAMP 型でない場合、列の値は TIMESTAMP 型に変換されてから返されます。

結果セットがフェッチされる前に GETTIMESTAMP メソッドが呼び出されると、ランタイムエラーが発生します。

結果セットには、バインドされた変数または GET*type* メソッドで排他的にアクセスする必要があります。変数が結果セットの列にバインドされ、GET*type* メソッドが呼び出されると、ランタイムエラーが発生します。

詳細については、“結果セットデータへのアクセス” (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- “SQLSTMT パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “GETDATE メソッド” (1969 ページ)
- “GETTIME メソッド” (1983 ページ)
- “SETDATE メソッド” (2001 ページ)
- “SETTIME メソッド” (2013 ページ)
- “SETTIMESTAMP メソッド” (2014 ページ)

GETTINYINT メソッド

指定された結果セット列を TINYINT 型として返します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

注: GETTINYINT メソッドを 1 回呼び出すと、結果セット列データを返すことができます。後続のメソッド呼び出しの結果、*rc* ステータスインジケーターの値は 2 (NODATA)になります。

構文

variable=*package*.**GETTINYINT** (*index*);

package.**GETTINYINT** (*index*, *variable*, *rc*);

必須引数

variable
指定された結果セット列の値を保持する変数を指定します。

注 指定された結果セット列の型が TINYINT 型でない場合、列の値は TINYINT 型に変換されてから返されます。

package
SQLSTMT パッケージのインスタンスを指定します。

index
1 から採番された結果セット列インデックスを指定します。

rc
リターンコードを配置する変数を指定します。次の値が可能です。

0
(SUCCESS)結果セットの列データが取得されます。

1
(ERROR)データ取得中にエラーが発生しました。

2
(NODATA)取得するデータはこれ以上ありません。

詳細

FETCH メソッドは、結果セットから次のデータ行を返します。変数が結果セットの列にバインドされていない場合、フェッチされたデータは GET*type* メソッドで返すことができます。

GETTINYINT メソッドは、指定された結果セット列の値を TINYINT 型として返します。指定された結果セット列の型が TINYINT 型でない場合、列の値は TINYINT 型に変換されてから返されます。

結果セットがフェッチされる前に GETTINYINT メソッドが呼び出されると、ランタイムエラーが発生します。

結果セットには、バインドされた変数または GET*type* メソッドで排他的にアクセスする必要があります。変数が結果セットの列にバインドされ、GET*type* メソッドが呼び出されると、ランタイムエラーが発生します。

詳細については、“結果セットデータへのアクセス” (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- “SQLSTMT パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “GETBIGINT メソッド” (1962 ページ)
- “GETINTEGER メソッド” (1974 ページ)
- “GETSMALLINT メソッド” (1981 ページ)
- “SETBIGINT メソッド” (1997 ページ)
- “SETINTEGER メソッド” (2005 ページ)
- “SETSMALLINT メソッド” (2011 ページ)
- “SETTINYINT メソッド” (2015 ページ)

GETVARBINARY メソッド

指定された結果セット列を VARBINARY 型として返します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

注: GETVARBINARY メソッドを繰り返し呼び出して、結果セット列データを返すことができます。すべてのデータが取得されると、*rc* ステータスインジケーターに値 2 (NODATA)が返されます。

構文

variable=*package*.**GETVARBINARY** (*index*);

package.**GETVARBINARY** (*index*, *variable*, *rc*);

必須引数

variable
指定された結果セット列の値を保持する変数を指定します。

注 指定された結果セット列の型が VARBINARY 型でない場合、列の値は VARBINARY 型に変換されてから返されます。

package
SQLSTMT パッケージのインスタンスを指定します。

index
1 から採番された結果セット列インデックスを指定します。

rc
リターンコードを配置する変数を指定します。次の値が可能です。

0
(SUCCESS)結果セットの列データが取得されます。

1
(ERROR)データ取得中にエラーが発生しました。

2
(NODATA)取得するデータはこれ以上ありません。

詳細

FETCH メソッドは、結果セットから次のデータ行を返します。変数が結果セットの列にバインドされていない場合、フェッチされたデータは GET*type* メソッドで返すことができます。

GETVARBINARY メソッドは、指定された結果セット列の値を VARBINARY 型として返します。指定された結果セット列の型が VARBINARY 型でない場合、列の値は VARBINARY 型に変換されてから返されます。

結果セットがフェッチされる前に GETVARBINARY メソッドが呼び出されると、ランタイムエラーが発生します。

結果セットには、バインドされた変数または GET*type* メソッドで排他的にアクセスする必要があります。変数が結果セットの列にバインドされ、GET*type* メソッドが呼び出されると、ランタイムエラーが発生します。

詳細については、“結果セットデータへのアクセス” (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- “SQLSTMT パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “GETBINARY メソッド” (1964 ページ)
- “SETBINARY メソッド” (1999 ページ)
- “SETVARBINARY メソッド” (2017 ページ)

GETVARCHAR メソッド

指定された結果セット列を VARCHAR 型として返します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

注: GETVARCHAR メソッドを繰り返し呼び出して、結果セット列データを返すことができます。すべてのデータが取得されると、*rc* ステータスインジケーターに値 2 (NODATA)が返されます。

構文

variable=*package*.**GETVARCHAR** (*index*);

package.**GETVARCHAR** (*index*, *variable*, *rc*);

必須引数

variable
指定された結果セット列の値を保持する変数を指定します。

注 指定された結果セット列の型が VARCHAR 型でない場合、列の値は VARCHAR 型に変換されてから返されます。

package
SQLSTMT パッケージのインスタンスを指定します。

index
1 から採番された結果セット列インデックスを指定します。

rc
リターンコードを配置する変数を指定します。次の値が可能です。

0
(SUCCESS)結果セットの列データが取得されます。

1
(ERROR)データ取得中にエラーが発生しました。

2
(NODATA)取得するデータはこれ以上ありません。

詳細

FETCH メソッドは、結果セットから次のデータ行を返します。変数が結果セットの列にバインドされていない場合、フェッチされたデータは GET*type* メソッドで返すことができます。

GETVARCHAR メソッドは、指定された結果セット列の値を VARCHAR 型として返します。指定された結果セット列の型が VARCHAR 型でない場合、列の値は VARCHAR 型に変換されてから返されます。

結果セットがフェッチされる前に GETVARCHAR メソッドが呼び出されると、ランタイムエラーが発生します。

結果セットには、バインドされた変数または GET*type* メソッドで排他的にアクセスする必要があります。変数が結果セットの列にバインドされ、GET*type* メソッドが呼び出されると、ランタイムエラーが発生します。

詳細については、“結果セットデータへのアクセス” (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- “SQLSTMT パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “GETCHAR メソッド” (1965 ページ)
- “GETNCHAR メソッド” (1975 ページ)
- “GETNVARCHAR メソッド” (1978 ページ)
- “SETCHAR メソッド” (2000 ページ)
- “SETNCHAR メソッド” (2006 ページ)
- “SETNVARCHAR メソッド” (2009 ページ)
- “SETVARCHAR メソッド” (2018 ページ)

ISPREPARED メソッド

SQLSTMT パッケージインスタンスの準備ができているかどうかを示す値を返します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

構文

package.**ISPREPARED**( );

必須引数

package
SQLSTMT パッケージのインスタンスを指定します。

詳細

SQLSTMT パッケージインスタンスが準備されていない場合、ISPREPARED メソッドは値 0 (false)を返します。SQLSTMT パッケージインスタンスが準備されている場合は、ゼロ以外の値(true)が返されます。

関連項目

メソッド:

- “PREPARE メソッド” (1994 ページ)

ステートメント:

- “DECLARE PACKAGE ステートメント: SQLSTMT パッケージ” (1957 ページ)

NEW 演算子: SQLSTMT パッケージ

SQLSTMT パッケージのインスタンスを構築します。

制限事項: この演算子は CAS サーバーではサポートされていません。

注: 構文規則の中かっこは、構文のグループ化を示します。中かっこの前のエスケープ文字(\)は、構文に中かっこが必要であることを示します。*sql-text* は中かっこ({ })で囲む必要があります。

かっこの前のエスケープ文字(\)は、構文にかっこが必要であることを示します。

構文

形式 1: *package-variable*=**_NEW_ SQLSTMT** ('*sql-text*' [\[*parameter-variable-list*\]]);

形式 2: *package-variable*=**_NEW_ SQLSTMT** ('*sql-text*' [, *connection-string*]);

形式 3: *package-variable*=**_NEW_ SQLSTMT**( );

必須引数

package-variable
SQLSTMT パッケージのインスタンスを参照できる名前を指定します。

'*sql-text*'
テーブルの行の挿入、更新、選択、削除を行う有効な FedSQL ステートメントです。

注意
SQLSTMT パッケージで使用できるのは FedSQL ステートメントのみです。DBMS 固有の SQL は使用できません。 詳細については、*SAS FedSQL 言語リファレンス*を参照してください。

要件 FedSQL ステートメントは、単一引用符(')で囲む必要があります。

注 ステートメントは、文字列リテラル、式から生成された文字列値、または変数に保存されている文字列値にすることができます。

FedSQL 言語の識別子のルールは、識別子の DS2 ルールではなく、SQLSTMT パッケージで使用される変数に適用されます。これは、FedSQL が DS2 ではなく SQL ステートメントを含む文字列を解析するために発生します。

オプション引数

[*parameter-variable-list*]
FedSQL ステートメントに含まれるパラメーターにバインドされる変数を指定します。

要件 変数は、角かっこ([])で囲む必要がある変数リストまたは名前付き変数リストの形式である必要があります。

パラメーター値は、バインドされた変数または SQLSTMT SET*type* メソッドのいずれかで排他的に指定する必要があります。

参照項目: "FedSQL ステートメントのパラメーター値の指定" (*SAS DS2 プログラマガイド*)

connection-string
完全に指定された接続文字列が含まれます。

デフォルト 接続文字列が提供されていない場合、SQLSTMT パッケージインスタンスは、現在割り当てられている libref の属性を使用して、HPDS2 プロシジャまたは DS2 プロシジャによって 生成された接続文字列を使用します。

注 接続文字列は、文字列リテラル、式から生成された文字列値、または変数に保存されている文字列値にすることができます。

ヒント 接続文字列は、データへの接続方法を定義します。接続文字列は、サブミットするクエリ言語と、1 つまたは複数のデータソースに接続するために必要な情報を識別します。

参照項目: このパラメーターは、主に SAS Federation Server で使用するために設計されています。完全に指定された接続文字列の作成の詳細については、*SAS Federation Server: 管理者ガイド*を参照してください。

詳細

DS2 パッケージは、特定のインスタンスを構築して他の DS2 プログラムで使用できる変数とメソッドのコレクションです。

SQLSTMT パッケージを使用して、テーブルの作成と削除、テーブルからの行の選択、および返された結果セットへのアクセスを行います。SQLSTMT パッケージは、DS2 プログラム用に事前定義されています。

DECLARE PACKAGE ステートメントを使用して、SQLSTMT パッケージを宣言できます。これにより、SQLSTMT パッケージが SQLSTMT 名に関連付けられます。

SQLSTMT パッケージのインスタンスを構築する方法は 2 つあります。

- DECLARE PACKAGE ステートメントを_NEW_演算子と一緒に使用します。

```
declare package sqlstmt sqlpkg;
sqlpkg = _new_ sqlstmt('update db2.dataset2 set y=? where x=?',[y x]);
```

- DECLARE PACKAGE ステートメントをそのコンストラクター構文と一緒に使用します。

```
declare package sqlstmt sqlpkg('update db2.dataset2 set y=? where x=?',[y x]);
```

SQL テキストを使用して SQLSTMT インスタンスを作成すると、FedSQL ステートメントが割り当てられて準備されます。FedSQL ステートメントにパラメーターが含まれている場合、FedSQL ステートメントを実行するには、パラメーターと置き換える値を取得する必要があります。置換値は、バインドされた変数の現在の値または SQLSTMT パッケージの SET*type* メソッドのいずれかを使用して指定できます。DECLARE PACKAGE ステートメントは、オプションの変数リストの変数を FedSQL ステートメントのパラメーターにバインドします。

注: パッケージ変数は、すべての 変数スコーピングルールの対象となります。詳細については、"パッケージ、スコープ、および有効期間" (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- “SQLSTMT パッケージの使用” (*SAS DS2 プログラマガイド*)
- “パッケージのコンストラクターとデストラクター” (*SAS DS2 プログラマガイド*)

メソッド:

- “BINDPARAMETERS メソッド” (1944 ページ)

ステートメント:

- “DECLARE PACKAGE ステートメント: SQLSTMT パッケージ” (1957 ページ)

PREPARE メソッド

FedSQL ステートメントを準備します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

構文

形式 1: **PREPARE** ('*sql-text*');

形式 2: **PREPARE** ('*sql-text*' , *connection-string*);

必須引数

'*sql-text*'
テーブルの行の挿入、更新、選択、削除を行う FedSQL ステートメントを含む有効な FedSQL ステートメントまたは文字列変数です。

注意
SQLSTMT パッケージで使用できるのは FedSQL ステートメントのみです。 DBMS 固有の SQL は使用できません。 詳細については、*SAS FedSQL 言語リファレンス*を参照してください。

要件 FedSQL ステートメントは、ステートメントが文字列変数に保存されていない限り、単一引用符(')で囲む必要があります。

注 ステートメントは文字列リテラルです。

FedSQL 言語の識別子のルールは、識別子の DS2 ルールではなく、SQLSTMT パッケージで使用される変数に適用されます。これは、FedSQL (DS2 ではない)が FedSQL ステートメントを含む文字列を解析するために発生します。

connection-string
完全に指定された接続文字列が含まれます。

デフォルト 接続文字列が提供されていない場合、SQLSTMT パッケージインスタンスは、現在割り当てられている libref の属性を使用して、HPDS2 プロシジャまたは DS2 プロシジャによって生成された接続文字列を使用します。

注 接続文字列は文字列リテラルです。

ヒント 接続文字列は、データへの接続方法を定義します。接続文字列は、サブミットするクエリ言語と、1 つまたは複数のデータソースに接続するために必要な情報を識別します。

参照項目: このパラメーターは、主に SAS Federation Server で使用するために設計されています。完全に指定された接続文字列の作成の詳細については、*SAS Federation Server: 管理者ガイド*を参照してください。

詳細

PREPARE メソッドはステータスインジケーターを返します。正常に実行されるとゼロが返され、エラーが発生した場合は 1 が返されます。

PREPARE メソッドを呼び出し、FedSQL ステートメントがすでに準備されている場合、ランタイムエラーが発生します。

関連項目

メソッド:

- “ISPREPARED メソッド” (1991 ページ)

ステートメント:

- “DECLARE PACKAGE ステートメント: SQLSTMT パッケージ” (1957 ページ)

REGISTEROUTPARAMETER メソッド

指定されたパラメーターを出力パラメーターとして登録します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

構文

REGISTEROUTPARAMETER (*index*)

必須引数

index
1 から採番されたパラメーターインデックスです。

詳細

REGISTEROUTPARAMETER は、FedSQL ステートメントの出力パラメーターを DS2 パッケージの METHOD ステートメントの IN_OUT パラメーターにマッピングするために使用されます。

REGISTEROUTPARAMETER メソッドはステータスインジケーターを返します。正常に実行されるとゼロが返され、エラーが発生した場合は 1 が返されます。

注: パラメーターデータは、バインドされた変数で排他的に指定する必要があります。SQLSTMT SET*type* メソッドと REGISTEROUTPARAMETER メソッドが呼び出されると、ランタイムエラーが発生します。

注: FedSQL ステートメントが実行された後、ステートメントの出力パラメーターの値が取得され、バインドされた変数に書き込まれます。

例

次の例は、SQLSTMT パッケージを使用して DS2 パッケージの METHOD ステートメントを実行する方法を示しています。この例は、FedSQL ステートメントの出力パラメーターを DS2 パッケージの METHOD ステートメントの IN_OUT パラメーターにマッピングする方法も示しています。

```
proc ds2;
package swapper / overwrite=yes;
  method swap(in_out int x, in_out int y);
    dcl int t;
    t = x;
    x = y;
    y = t;
  end;
endpackage;
run;

data _null_;
  dcl int a b;
  method init();
    dcl package sqlstmt stmt();
    dcl int rc;

    rc = stmt.prepare('call swapper.swap(?, ?)');
    if (rc) then put 'TEST ERROR: prepare';
```

```
    rc = stmt.bindparameters([a, b]);
    if (rc) then put 'TEST ERROR: bindParameters';

    rc = stmt.registerOutParameter(1);
    if (rc) then put 'TEST ERROR: registerOutParameter';

    rc = stmt.registerOutParameter(2);
    if (rc) then put 'TEST ERROR: registerOutParameter';

    a = 42;
    b = 101;

    put 'Before executing swapper.swap:' a= b=;
    rc = stmt.execute();
    if (rc) then put 'TEST ERROR: execute';
    put 'After executing swapper.swap:' a= b=;
  end;
enddata;
run;
quit;
```

次の行が SAS ログに書き込まれます。

```
Before executing swapper.swap: a=42 b=101
After executing swapper.swap: a=101 b=42
```

関連項目

ステートメント:

- “METHOD ステートメント” (1376 ページ)

SETBIGINT メソッド

指定されたパラメーターに BIGINT 型の指定された値を設定します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

構文

package.**SETBIGINT** (*index*, *value*);

必須引数

package
SQLSTMT パッケージのインスタンスを指定します。

index
1 から採番されたパラメーターインデックスを指定します。

value
指定されたパラメーターに設定する値を指定します。このパラメーターは、*index* で指定されています。

ヒント　*value* は、リテラル、変数、または式にすることができます。

詳細

SQLSTMT インスタンスを作成すると、FedSQL ステートメントが割り当てられて準備されます。FedSQL ステートメントにパラメーターが含まれている場合、FedSQL ステートメントを実行するには、パラメーターと置き換える値を取得する必要があります。置換値は、バインドされた変数の現在の値またはパッケージの SET*type* メソッドのいずれかを使用して指定できます。

指定されたパラメーターの型が BIGINT でない場合、BIGINT 値は指定されたパラメーターの型に変換されます。たとえば、setbigint(1, 3)を使用して、パラメーター 1 に BIGINT 値 3 を設定するときに、パラメーター 1 の型が CHAR である場合、BIGINT 値 3 は CHAR 値 3 に変換され、CHAR 値を使用してパラメーターが指定されます。

パラメーターデータは、バインドされた変数または SET*type* メソッドのいずれかで排他的に指定する必要があります。変数がパラメーターにバインドされた状態で、SETBIGINT メソッドが呼び出されると、ラインタイムエラーが発生します。

SETBIGINT メソッドはステータスインジケーターを返します。正常に実行されるとゼロが返され、エラーが発生した場合は 1 が返されます。

詳細については、“FedSQL ステートメントのパラメーター値の指定” (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- “SQLSTMT パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “GETBIGINT メソッド” (1962 ページ)
- “GETINTEGER メソッド” (1974 ページ)
- “GETSMALLINT メソッド” (1981 ページ)
- “GETTINYINT メソッド” (1986 ページ)
- “SETINTEGER メソッド” (2005 ページ)
- “SETSMALLINT メソッド” (2011 ページ)
- “SETTINYINT メソッド” (2015 ページ)

SETBINARY メソッド

指定されたパラメーターに BINARY 型の指定された値を設定します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

構文

package.**SETBINARY** (*index*, *value*);

必須引数

package
SQLSTMT パッケージのインスタンスを指定します。

index
1 から採番されたパラメーターインデックスを指定します。

value
指定されたパラメーターに設定する値を指定します。このパラメーターは、*index* で指定されています。

ヒント *value* は、リテラル、変数、または式にすることができます。

詳細

SQLSTMT インスタンスを作成すると、FedSQL ステートメントが割り当てられて準備されます。FedSQL ステートメントにパラメーターが含まれている場合、FedSQL ステートメントを実行するには、パラメーターと置き換える値を取得する必要があります。置換値は、バインドされた変数の現在の値またはパッケージの SET*type* メソッドのいずれかを使用して指定できます。

指定されたパラメーターの型が BINARY でない場合、BINARY 値は指定されたパラメーターの型に変換されます。たとえば、setbinary(1, 0110)を使用して、パラメーター 1 に BINARY 値 0110 を設定するときに、パラメーター 1 の型が CHAR である場合、BINARY 値 0110 は CHAR 値 0110 に変換され、CHAR 値を使用してパラメーターが指定されます。

パラメーターデータは、バインドされた変数または SET*type* メソッドのいずれかで排他的に指定する必要があります。変数がパラメーターにバインドされた状態で、SETBINARY メソッドが呼び出されると、ラインタイムエラーが発生します。

SETBINARY メソッドはステータスインジケーターを返します。正常に実行されるとゼロが返され、エラーが発生した場合は 1 が返されます。

詳細については、“FedSQL ステートメントのパラメーター値の指定” (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- “SQLSTMT パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “GETBINARY メソッド” (1964 ページ)
- “GETVARBINARY メソッド” (1987 ページ)
- “SETVARBINARY メソッド” (2017 ページ)

SETCHAR メソッド

指定されたパラメーターに CHAR 型の指定された値を設定します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

構文

package.**SETCHAR** (*index*, *value*);

必須引数

package
SQLSTMT パッケージのインスタンスを指定します。

index
1 から採番されたパラメーターインデックスを指定します。

value
指定されたパラメーターに設定する値を指定します。このパラメーターは、*index* で指定されています。

ヒント *value* は、リテラル、変数、または式にすることができます。

詳細

SQLSTMT インスタンスを作成すると、FedSQL ステートメントが割り当てられて準備されます。FedSQL ステートメントにパラメーターが含まれている場合、FedSQL ステートメントを実行するには、パラメーターと置き換える値を取得する必要があります。置換値は、バインドされた変数の現在の値またはパッケージの SET*type* メソッドのいずれかを使用して指定できます。

指定されたパラメーターの型が CHAR でない場合、CHAR 値は指定されたパラメーターの型に変換されます。たとえば、setchar(1, 3)を使用して、パラメーター 1 に CHAR 値 3 を設定するときに、パラメーター 1 の型が INTEGER である場合、CHAR

値 3 は INTEGER 値 3 に変換され、INTEGER 値を使用してパラメーターが指定されます。

パラメーターデータは、バインドされた変数または SET*type* メソッドのいずれかで排他的に指定する必要があります。変数がパラメーターにバインドされた状態で、SETCHAR メソッドが呼び出されると、ラインタイムエラーが発生します。

SETCHAR メソッドはステータスインジケーターを返します。正常に実行されるとゼロが返され、エラーが発生した場合は 1 が返されます。

詳細については、"FedSQL ステートメントのパラメーター値の指定" (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- "SQLSTMT パッケージの使用" (*SAS DS2 プログラマガイド*)

メソッド:

- "GETCHAR メソッド" (1965 ページ)
- "GETNCHAR メソッド" (1975 ページ)
- "GETNVARCHAR メソッド" (1978 ページ)
- "GETVARCHAR メソッド" (1989 ページ)
- "SETNCHAR メソッド" (2006 ページ)
- "SETNVARCHAR メソッド" (2009 ページ)
- "SETVARCHAR メソッド" (2018 ページ)

SETDATE メソッド

指定されたパラメーターに DATE 型の指定された値を設定します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

構文

package.**SETDATE** (*index*, *value*);

必須引数

package
SQLSTMT パッケージのインスタンスを指定します。

index
1 から採番されたパラメーターインデックスを指定します。

value
指定されたパラメーターに設定する値を指定します。このパラメーターは、*index* で指定されています。

ヒント *value* は、リテラル、変数、または式にすることができます。

詳細

SQLSTMT インスタンスを作成すると、FedSQL ステートメントが割り当てられて準備されます。FedSQL ステートメントにパラメーターが含まれている場合、FedSQL ステートメントを実行するには、パラメーターと置き換える値を取得する必要があります。置換値は、バインドされた変数の現在の値またはパッケージの SET*type* メソッドのいずれかを使用して指定できます。

指定されたパラメーターの型が DATE でない場合、DATE 値は指定されたパラメーターの型に変換されます。

パラメーターデータは、バインドされた変数または SET*type* メソッドのいずれかで排他的に指定する必要があります。変数がパラメーターにバインドされた状態で、SETDATE メソッドが呼び出されると、ラインタイムエラーが発生します。

SETDATE メソッドはステータスインジケーターを返します。正常に実行されるとゼロが返され、エラーが発生した場合は 1 が返されます。

詳細については、“FedSQL ステートメントのパラメーター値の指定” (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- “SQLSTMT パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “GETDATE メソッド” (1969 ページ)
- “GETTIME メソッド” (1983 ページ)
- “GETTIMESTAMP メソッド” (1984 ページ)
- “SETTIME メソッド” (2013 ページ)
- “SETTIMESTAMP メソッド” (2014 ページ)

SETDECIMAL メソッド

指定されたパラメーターに DECIMAL 型の指定された値を設定します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

構文

package.**SETDECIMAL** (*index*, *value*);

必須引数

package
SQLSTMT パッケージのインスタンスを指定します。

index
1 から採番されたパラメーターインデックスを指定します。

value
指定されたパラメーターに設定する値を指定します。このパラメーターは、*index* で指定されています。

ヒント *value* は、リテラル、変数、または式にすることができます。

詳細

SQLSTMT インスタンスを作成すると、FedSQL ステートメントが割り当てられて準備されます。FedSQL ステートメントにパラメーターが含まれている場合、FedSQL ステートメントを実行するには、パラメーターと置き換える値を取得する必要があります。置換値は、バインドされた変数の現在の値またはパッケージの SET*type* メソッドのいずれかを使用して指定できます。

指定されたパラメーターの型が DECIMAL でない場合、DECIMAL 値は指定されたパラメーターの型に変換されます。たとえば、setdecimal(1, 13.4)を使用して、パラメーター 1 に DECIMAL 値 13.4 を設定するときに、パラメーター 1 の型が CHAR である場合、DECIMAL 値 13.4 は CHAR 値 13.4 に変換され、CHAR 値を使用してパラメーターが指定されます。

パラメーターデータは、バインドされた変数または SET*type* メソッドのいずれかで排他的に指定する必要があります。変数がパラメーターにバインドされた状態で、SETDECIMAL メソッドが呼び出されると、ラインタイムエラーが発生します。

SETDECIMAL メソッドはステータスインジケーターを返します。正常に実行されるとゼロが返され、エラーが発生した場合は 1 が返されます。

詳細については、“FedSQL ステートメントのパラメーター値の指定” (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- “SQLSTMT パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “GETDECIMAL メソッド” (1971 ページ)

SETDOUBLE メソッド

指定されたパラメーターに DOUBLE 型の指定された値を設定します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

構文

package.**SETDOUBLE** (*index*, *value*);

必須引数

package
SQLSTMT パッケージのインスタンスを指定します。

index
1 から採番されたパラメーターインデックスを指定します。

value
指定されたパラメーターに設定する値を指定します。このパラメーターは、*index* で指定されています。

ヒント *value* は、リテラル、変数、または式にすることができます。

詳細

SQLSTMT インスタンスを作成すると、FedSQL ステートメントが割り当てられて準備されます。FedSQL ステートメントにパラメーターが含まれている場合、FedSQL ステートメントを実行するには、パラメーターと置き換える値を取得する必要があります。置換値は、バインドされた変数の現在の値またはパッケージの SET*type* メソッドのいずれかを使用して指定できます。

指定されたパラメーターの型が DOUBLE でない場合、DOUBLE 値は指定されたパラメーターの型に変換されます。たとえば、setdouble(1, 33443452)を使用して、パラメーター 1 に DOUBLE 値 33443452 を設定するときに、パラメーター 1 の型が CHAR である場合、DOUBLE 値 33443452 は CHAR 値 33443452 に変換され、CHAR 値を使用してパラメーターが指定されます。

パラメーターデータは、バインドされた変数または SET*type* メソッドのいずれかで排他的に指定する必要があります。変数がパラメーターにバインドされた状態で、SETDOUBLE メソッドが呼び出されると、ラインタイムエラーが発生します。

SETDOUBLE メソッドはステータスインジケーターを返します。正常に実行されるとゼロが返され、エラーが発生した場合は 1 が返されます。

詳細については、“FedSQL ステートメントのパラメーター値の指定” (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- “SQLSTMT パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “GETDOUBLE メソッド” (1972 ページ)

SETINTEGER メソッド

指定されたパラメーターに INTEGER 型の指定された値を設定します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

構文

package.**SETINTEGER** (*index*, *value*);

必須引数

package
SQLSTMT パッケージのインスタンスを指定します。

index
1 から採番されたパラメーターインデックスを指定します。

value
指定されたパラメーターに設定する値を指定します。このパラメーターは、*index* で指定されています。

ヒント *value* は、リテラル、変数、または式にすることができます。

詳細

SQLSTMT インスタンスを作成すると、FedSQL ステートメントが割り当てられて準備されます。FedSQL ステートメントにパラメーターが含まれている場合、FedSQL ステートメントを実行するには、パラメーターと置き換える値を取得する必要があります。置換値は、バインドされた変数の現在の値またはパッケージの SET*type* メソッドのいずれかを使用して指定できます。

指定されたパラメーターの型が INTEGER でない場合、INTEGER 値は指定されたパラメーターの型に変換されます。たとえば、setinteger(1, 3)を使用して、パラメーター 1 に INTEGER 値 3 を設定するときに、パラメーター 1 の型が CHAR である場合、INTEGER 値 3 は CHAR 値 3 に変換され、CHAR 値を使用してパラメーターが指定されます。

パラメーターデータは、バインドされた変数または SET*type* メソッドのいずれかで排他的に指定する必要があります。変数がパラメーターにバインドされた状態で、SETINTEGER メソッドが呼び出されると、ラインタイムエラーが発生します。

SETINTEGER メソッドはステータスインジケーターを返します。正常に実行されるとゼロが返され、エラーが発生した場合は 1 が返されます。

詳細については、"FedSQL ステートメントのパラメーター値の指定" (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- "SQLSTMT パッケージの使用" (*SAS DS2 プログラマガイド*)

メソッド:

- "GETBIGINT メソッド" (1962 ページ)
- "GETINTEGER メソッド" (1974 ページ)
- "GETSMALLINT メソッド" (1981 ページ)
- "GETTINYINT メソッド" (1986 ページ)
- "SETBIGINT メソッド" (1997 ページ)
- "SETSMALLINT メソッド" (2011 ページ)
- "SETTINYINT メソッド" (2015 ページ)

SETNCHAR メソッド

指定されたパラメーターに NCHAR 型の指定された値を設定します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

構文

package.**SETNCHAR** (*index*, *value*);

必須引数

package
SQLSTMT パッケージのインスタンスを指定します。

index
1 から採番されたパラメーターインデックスを指定します。

value
指定されたパラメーターに設定する値を指定します。このパラメーターは、*index* で指定されています。

ヒント *value* は、リテラル、変数、または式にすることができます。

詳細

SQLSTMT インスタンスを作成すると、FedSQL ステートメントが割り当てられて準備されます。FedSQL ステートメントにパラメーターが含まれている場合、FedSQL ステートメントを実行するには、パラメーターと置き換える値を取得する必要があります。置換値は、バインドされた変数の現在の値またはパッケージの SET*type* メソッドのいずれかを使用して指定できます。

指定されたパラメーターの型が NCHAR でない場合、NCHAR 値は指定されたパラメーターの型に変換されます。たとえば、setnchar(1, 3)を使用して、パラメーター 1 に NCHAR 値 3 を設定するときに、パラメーター 1 の型が INTEGER である場合、NCHAR 値 3 は INTEGER 値 3 に変換され、INTEGER 値を使用してパラメーターが指定されます。

パラメーターデータは、バインドされた変数または SET*type* メソッドのいずれかで排他的に指定する必要があります。変数がパラメーターにバインドされた状態で、SETNCHAR メソッドが呼び出されると、ラインタイムエラーが発生します。

SETNCHAR メソッドはステータスインジケーターを返します。正常に実行されるとゼロが返され、エラーが発生した場合は 1 が返されます。

詳細については、“FedSQL ステートメントのパラメーター値の指定” (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- “SQLSTMT パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “GETCHAR メソッド” (1965 ページ)
- “GETNCHAR メソッド” (1975 ページ)
- “GETNVARCHAR メソッド” (1978 ページ)
- “GETVARCHAR メソッド” (1989 ページ)
- “SETCHAR メソッド” (2000 ページ)
- “SETNVARCHAR メソッド” (2009 ページ)
- “SETVARCHAR メソッド” (2018 ページ)

SETNUMERIC メソッド

指定されたパラメーターに NUMERIC 型の指定された値を設定します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

構文

package.**SETNUMERIC** (*index*, *value*);

必須引数

package
SQLSTMT パッケージのインスタンスを指定します。

index
1 から採番されたパラメーターインデックスを指定します。

value
指定されたパラメーターに設定する値を指定します。このパラメーターは、*index* で指定されています。

ヒント *value* は、リテラル、変数、または式にすることができます。

詳細

SQLSTMT インスタンスを作成すると、FedSQL ステートメントが割り当てられて準備されます。FedSQL ステートメントにパラメーターが含まれている場合、FedSQL ステートメントを実行するには、パラメーターと置き換える値を取得する必要があります。置換値は、バインドされた変数の現在の値またはパッケージの SET*type* メソッドのいずれかを使用して指定できます。

指定されたパラメーターの型が NUMERIC でない場合、NUMERIC 値は指定されたパラメーターの型に変換されます。たとえば、setnumeric(1, 3)を使用して、パラメーター 1 に NUMERIC 値 3 を設定するときに、パラメーター 1 の型が CHAR である場合、NUMERIC 値 3 は CHAR 値 3 に変換され、CHAR 値を使用してパラメーターが指定されます。

パラメーターデータは、バインドされた変数または SET*type* メソッドのいずれかで排他的に指定する必要があります。変数がパラメーターにバインドされた状態で、SETNUMERIC メソッドが呼び出されると、ラインタイムエラーが発生します。

SETNUMERIC メソッドはステータスインジケーターを返します。正常に実行されるとゼロが返され、エラーが発生した場合は 1 が返されます。

詳細については、“FedSQL ステートメントのパラメーター値の指定” (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- “SQLSTMT パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “GETNUMERIC メソッド” (1977 ページ)

SETNVARCHAR メソッド

指定されたパラメーターに NVARCHAR 型の指定された値を設定します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

構文

package.**SETNVARCHAR** (*index*, *value*);

必須引数

package
SQLSTMT パッケージのインスタンスを指定します。

index
1 から採番されたパラメーターインデックスを指定します。

value
指定されたパラメーターに設定する値を指定します。このパラメーターは、*index* で指定されています。

ヒント *value* は、リテラル、変数、または式にすることができます。

詳細

SQLSTMT インスタンスを作成すると、FedSQL ステートメントが割り当てられて準備されます。FedSQL ステートメントにパラメーターが含まれている場合、FedSQL ステートメントを実行するには、パラメーターと置き換える値を取得する必要があります。置換値は、バインドされた変数の現在の値またはパッケージの SET*type* メソッドのいずれかを使用して指定できます。

指定されたパラメーターの型が NVARCHAR でない場合、NVARCHAR 値は指定されたパラメーターの型に変換されます。たとえば、setnvarchar(1, 3)を使用して、パラメーター 1 に NVARCHAR 値 3 を設定するときに、パラメーター 1 の型が INTEGER である場合、NVARCHAR 値 3 は INTEGER 値 3 に変換され、INTEGER 値を使用してパラメーターが指定されます。

パラメーターデータは、バインドされた変数または SET*type* メソッドのいずれかで排他的に指定する必要があります。変数がパラメーターにバインドされた状態で、SETNVARCHAR メソッドが呼び出されると、ラインタイムエラーが発生します。

SETNVARCHAR メソッドはステータスインジケーターを返します。正常に実行されるとゼロが返され、エラーが発生した場合は 1 が返されます。

詳細については、"FedSQL ステートメントのパラメーター値の指定" (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- "SQLSTMT パッケージの使用" (*SAS DS2 プログラマガイド*)

メソッド:

- "GETCHAR メソッド" (1965 ページ)
- "GETNCHAR メソッド" (1975 ページ)
- "GETNVARCHAR メソッド" (1978 ページ)
- "GETVARCHAR メソッド" (1989 ページ)
- "SETCHAR メソッド" (2000 ページ)
- "SETNCHAR メソッド" (2006 ページ)
- "SETVARCHAR メソッド" (2018 ページ)

SETREAL メソッド

指定されたパラメーターに REAL 型の指定された値を設定します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

構文

package.**SETREAL** (*index*, *value*);

必須引数

package
SQLSTMT パッケージのインスタンスを指定します。

index
1 から採番されたパラメーターインデックスを指定します。

value
指定されたパラメーターに設定する値を指定します。このパラメーターは、*index* で指定されています。

ヒント *value* は、リテラル、変数、または式にすることができます。

詳細

SQLSTMT インスタンスを作成すると、FedSQL ステートメントが割り当てられて準備されます。FedSQL ステートメントにパラメーターが含まれている場合、FedSQL ステートメントを実行するには、パラメーターと置き換える値を取得する必要があります。置換値は、バインドされた変数の現在の値またはパッケージの SET*type* メソッドのいずれかを使用して指定できます。

指定されたパラメーターの型が REAL でない場合、REAL 値は指定されたパラメーターの型に変換されます。たとえば、setreal(1, 3)を使用して、パラメーター 1 に REAL 値 3 を設定するときに、パラメーター 1 の型が CHAR である場合、REAL 値 3 は CHAR 値 3 に変換され、CHAR 値を使用してパラメーターが指定されます。

パラメーターデータは、バインドされた変数または SET*type* メソッドのいずれかで排他的に指定する必要があります。変数がパラメーターにバインドされた状態で、SETREAL メソッドが呼び出されると、ラインタイムエラーが発生します。

SETREAL メソッドはステータスインジケーターを返します。正常に実行されるとゼロが返され、エラーが発生した場合は 1 が返されます。

詳細については、“FedSQL ステートメントのパラメーター値の指定” (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- “SQLSTMT パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “GETREAL メソッド” (1980 ページ)

SETSMALLINT メソッド

指定されたパラメーターに SMALLINT 型の指定された値を設定します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

構文

package.**SETSMALLINT** (*index*, *value*);

必須引数

package
SQLSTMT パッケージのインスタンスを指定します。

index
1 から採番されたパラメーターインデックスを指定します。

value
指定されたパラメーターに設定する値を指定します。このパラメーターは、*index* で指定されています。

ヒント *value* は、リテラル、変数、または式にすることができます。

詳細

SQLSTMT インスタンスを作成すると、FedSQL ステートメントが割り当てられて準備されます。FedSQL ステートメントにパラメーターが含まれている場合、FedSQL ステートメントを実行するには、パラメーターと置き換える値を取得する必要があります。置換値は、バインドされた変数の現在の値またはパッケージの SET*type* メソッドのいずれかを使用して指定できます。

指定されたパラメーターの型が SMALLINT でない場合、SMALLINT 値は指定されたパラメーターの型に変換されます。たとえば、setsmallint(1, 3)を使用して、パラメーター 1 に SMALLINT 値 3 を設定するときに、パラメーター 1 の型が CHAR である場合、SMALLINT 値 3 は CHAR 値 3 に変換され、CHAR 値を使用してパラメーターが指定されます。

パラメーターデータは、バインドされた変数または SET*type* メソッドのいずれかで排他的に指定する必要があります。変数がパラメーターにバインドされた状態で、SETSMALLINT メソッドが呼び出されると、ラインタイムエラーが発生します。

SETSMALLINT メソッドはステータスインジケーターを返します。正常に実行されるとゼロが返され、エラーが発生した場合は 1 が返されます。

詳細については、“FedSQL ステートメントのパラメーター値の指定” (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- “SQLSTMT パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “GETBIGINT メソッド” (1962 ページ)

- “GETINTEGER メソッド” (1974 ページ)
- “GETSMALLINT メソッド” (1981 ページ)
- “GETTINYINT メソッド” (1986 ページ)
- “SETBIGINT メソッド” (1997 ページ)
- “SETINTEGER メソッド” (2005 ページ)
- “SETTINYINT メソッド” (2015 ページ)

SETTIME メソッド

指定されたパラメーターに TIME 型の指定された値を設定します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

構文

package.**SETTIME** (*index*, *value*);

必須引数

package
SQLSTMT パッケージのインスタンスを指定します。

index
1 から採番されたパラメーターインデックスを指定します。

value
指定されたパラメーターに設定する値を指定します。このパラメーターは、*index* で指定されています。

ヒント *value* は、リテラル、変数、または式にすることができます。

詳細

SQLSTMT インスタンスを作成すると、FedSQL ステートメントが割り当てられて準備されます。FedSQL ステートメントにパラメーターが含まれている場合、FedSQL ステートメントを実行するには、パラメーターと置き換える値を取得する必要があります。置換値は、バインドされた変数の現在の値またはパッケージの SET*type* メソッドのいずれかを使用して指定できます。

指定されたパラメーターの型が TIME でない場合、TIME 値は指定されたパラメーターの型に変換されます。

パラメーターデータは、バインドされた変数または SET*type* メソッドのいずれかで排他的に指定する必要があります。変数がパラメーターにバインドされた状態で、SETTIME メソッドが呼び出されると、ラインタイムエラーが発生します。

SETTIME メソッドはステータスインジケーターを返します。正常に実行されるとゼロが返され、エラーが発生した場合は 1 が返されます。

詳細については、“FedSQL ステートメントのパラメーター値の指定” (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- “SQLSTMT パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “GETDATE メソッド” (1969 ページ)
- “GETTIME メソッド” (1983 ページ)
- “GETTIMESTAMP メソッド” (1984 ページ)
- “SETDATE メソッド” (2001 ページ)
- “SETTIMESTAMP メソッド” (2014 ページ)

SETTIMESTAMP メソッド

指定されたパラメーターに TIMESTAMP 型の指定された値を設定します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

構文

package.**SETTIMESTAMP** (*index*, *value*);

必須引数

package
SQLSTMT パッケージのインスタンスを指定します。

index
1 から採番されたパラメーターインデックスを指定します。

value
指定されたパラメーターに設定する値を指定します。このパラメーターは、*index* で指定されています。

ヒント *value* は、リテラル、変数、または式にすることができます。

詳細

SQLSTMT インスタンスを作成すると、FedSQL ステートメントが割り当てられて準備されます。FedSQL ステートメントにパラメーターが含まれている場合、FedSQL ステートメントを実行するには、パラメーターと置き換える値を取得する必要があります。置換値は、バインドされた変数の現在の値またはパッケージの SET*type* メソッドのいずれかを使用して指定できます。

指定されたパラメーターの型が TIMESTAMP でない場合、TIMESTAMP 値は指定されたパラメーターの型に変換されます。

パラメーターデータは、バインドされた変数または SET*type* メソッドのいずれかで排他的に指定する必要があります。変数がパラメーターにバインドされた状態で、SETTIMESTAMP メソッドが呼び出されると、ラインタイムエラーが発生します。

SETTIMESTAMP メソッドはステータスインジケーターを返します。正常に実行されるとゼロが返され、エラーが発生した場合は 1 が返されます。

詳細については、“FedSQL ステートメントのパラメーター値の指定” (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- “SQLSTMT パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “GETDATE メソッド” (1969 ページ)
- “GETTIME メソッド” (1983 ページ)
- “GETTIMESTAMP メソッド” (1984 ページ)
- “SETDATE メソッド” (2001 ページ)
- “SETTIME メソッド” (2013 ページ)

SETTINYINT メソッド

指定されたパラメーターに TINYINT 型の指定された値を設定します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

構文

package.**SETTINYINT** (*index*, *value*);

必須引数

package
SQLSTMT パッケージのインスタンスを指定します。

index
1 から採番されたパラメーターインデックスを指定します。

value
指定されたパラメーターに設定する値を指定します。このパラメーターは、*index* で指定されています。

ヒント *value* は、リテラル、変数、または式にすることができます。

詳細

SQLSTMT インスタンスを作成すると、FedSQL ステートメントが割り当てられて準備されます。FedSQL ステートメントにパラメーターが含まれている場合、FedSQL ステートメントを実行するには、パラメーターと置き換える値を取得する必要があります。置換値は、バインドされた変数の現在の値またはパッケージの SET*type* メソッドのいずれかを使用して指定できます。

指定されたパラメーターの型が TINYINT でない場合、TINYINT 値は指定されたパラメーターの型に変換されます。たとえば、settinyint(1, 3)を使用して、パラメーター 1 に TINYINT 値 3 を設定するときに、パラメーター 1 の型が CHAR である場合、TINYINT 値 3 は CHAR 値 3 に変換され、CHAR 値を使用してパラメーターが指定されます。

パラメーターデータは、バインドされた変数または SET*type* メソッドのいずれかで排他的に指定する必要があります。変数がパラメーターにバインドされた状態で、SETTINYINT メソッドが呼び出されると、ラインタイムエラーが発生します。

SETTINYINT メソッドはステータスインジケーターを返します。正常に実行されるとゼロが返され、エラーが発生した場合は 1 が返されます。

詳細については、“FedSQL ステートメントのパラメーター値の指定” (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- “SQLSTMT パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “GETBIGINT メソッド” (1962 ページ)
- “GETINTEGER メソッド” (1974 ページ)
- “GETSMALLINT メソッド” (1981 ページ)
- “GETTINYINT メソッド” (1986 ページ)
- “SETBIGINT メソッド” (1997 ページ)

- “SETINTEGER メソッド” (2005 ページ)
- “SETSMALLINT メソッド” (2011 ページ)

SETVARBINARY メソッド

指定されたパラメーターに VARBINARY 型の指定された値を設定します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

構文

package.**SETVARBINARY** (*index*, *value*);

必須引数

package
SQLSTMT パッケージのインスタンスを指定します。

index
1 から採番されたパラメーターインデックスを指定します。

value
指定されたパラメーターに設定する値を指定します。このパラメーターは、*index* で指定されています。

ヒント *value* は、リテラル、変数、または式にすることができます。

詳細

SQLSTMT インスタンスを作成すると、FedSQL ステートメントが割り当てられて準備されます。FedSQL ステートメントにパラメーターが含まれている場合、FedSQL ステートメントを実行するには、パラメーターと置き換える値を取得する必要があります。置換値は、バインドされた変数の現在の値またはパッケージの SET*type* メソッドのいずれかを使用して指定できます。

指定されたパラメーターの型が VARBINARY でない場合、VARBINARY 値は指定されたパラメーターの型に変換されます。たとえば、setvarbinary(1, 0110)を使用して、パラメーター 1 に VARBINARY 値 0110 を設定するときに、パラメーター 1 の型が CHAR である場合、VARBINARY 値 0110 は CHAR 値 0110 に変換され、CHAR 値を使用してパラメーターが指定されます。

パラメーターデータは、バインドされた変数または SET*type* メソッドのいずれかで排他的に指定する必要があります。変数がパラメーターにバインドされた状態で、SETVARBINARY メソッドが呼び出されると、ラインタイムエラーが発生します。

SETVARBINARY メソッドはステータスインジケーターを返します。正常に実行されるとゼロが返され、エラーが発生した場合は 1 が返されます。

詳細については、“FedSQL ステートメントのパラメーター値の指定” (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- “SQLSTMT パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “GETBINARY メソッド” (1964 ページ)
- “GETVARBINARY メソッド” (1987 ページ)
- “SETBINARY メソッド” (1999 ページ)

SETVARCHAR メソッド

指定されたパラメーターに VARCHAR 型の指定された値を設定します。

制限事項: このメソッドは CAS サーバーではサポートされていません。

構文

package.**SETVARCHAR** (*index*, *value*);

必須引数

package
SQLSTMT パッケージのインスタンスを指定します。

index
1 から採番されたパラメーターインデックスを指定します。

value
指定されたパラメーターに設定する値を指定します。このパラメーターは、*index* で指定されています。

ヒント *value* は、リテラル、変数、または式にすることができます。

詳細

SQLSTMT インスタンスを作成すると、FedSQL ステートメントが割り当てられて準備されます。FedSQL ステートメントにパラメーターが含まれている場合、FedSQL ステートメントを実行するには、パラメーターと置き換える値を取得する必要があります。置換値は、バインドされた変数の現在の値またはパッケージの SET*type* メソッドのいずれかを使用して指定できます。

指定されたパラメーターの型が VARCHAR でない場合、VARCHAR 値は指定されたパラメーターの型に変換されます。たとえば、setvarchar(1, pass)を使用して、パラメーター 1 に VARCHAR 値 **pass** を設定するときに、パラメーター 1 の型が CHAR である場合、VARCHAR 値 **pass** は CHAR 値 **pass** に変換され、CHAR 値を使用してパラメーターが指定されます。

パラメーターデータは、バインドされた変数または SET*type* メソッドのいずれかで排他的に指定する必要があります。変数がパラメーターにバインドされた状態で、SETVARCHAR メソッドが呼び出されると、ラインタイムエラーが発生します。

SETVARCHAR メソッドはステータスインジケーターを返します。正常に実行されるとゼロが返され、エラーが発生した場合は 1 が返されます。

詳細については、“FedSQL ステートメントのパラメーター値の指定” (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- “SQLSTMT パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “GETCHAR メソッド” (1965 ページ)
- “GETNCHAR メソッド” (1975 ページ)
- “GETNVARCHAR メソッド” (1978 ページ)
- “GETVARCHAR メソッド” (1989 ページ)
- “SETCHAR メソッド” (2000 ページ)
- “SETNCHAR メソッド” (2006 ページ)
- “SETNVARCHAR メソッド” (2009 ページ)

25

DS2 TZ パッケージのメソッド、演算子、およびステートメント

ディクショナリ **2021**
DECLARE PACKAGE ステートメント: TZ パッケージ 2021
GETLOCALTIME メソッド 2023
GETOFFSET メソッド 2024
GETOFFSETUTC メソッド 2026
GETTIMEZONEID メソッド 2027
GETTIMEZONENAME メソッド 2029
GETUTCTIME メソッド 2030
NEW 演算子: TZ パッケージ 2031
TOISO8601 メソッド 2032
TOLOCALTIME メソッド 2033
TOTIMESTAMPZ メソッド 2034
TOUTCTIME メソッド 2035

ディクショナリ

DECLARE PACKAGE ステートメント: TZ パッケージ

パッケージ変数を作成し、TZ パッケージのインスタンスを作成できるようにします。

カテゴリ: ローカル

構文

DECLARE PACKAGE TZ *variable* ([*time-zone-id*]);

必須引数

variable
TZ パッケージのインスタンスを参照できる名前を指定します。

オプション引数

time-zone-id
タイムゾーン ID を指定します。

デフォルト　TIMEZONE=システムオプションで指定された値。

詳細

DS2 パッケージは、特定のインスタンスを構築して他の DS2 プログラムで使用できる変数とメソッドのコレクションです。

タイムゾーン処理には TZ パッケージを使用します。TZ パッケージは、DS2 プログラム用に事前定義されています。SAS のタイムゾーンの詳細については、*SAS 各国語サポート(NLS): リファレンスガイド*を参照してください。

DECLARE PACKAGE ステートメントを使用して、TZ パッケージを宣言します。これにより、TZ パッケージがタイムゾーン名に関連付けられます。新しい TZ パッケージを宣言した後、それに応じて日付と時間のデータをフォーマットできます。

パッケージが宣言されると、パッケージのインスタンスを参照できる変数が作成されます。コンストラクター引数がパッケージ変数宣言で提供されている場合、パッケージインスタンスが構築され、構築されたパッケージインスタンスを参照するようにパッケージ変数が設定されます。複数のパッケージ変数を作成でき、単一の DECLARE PACKAGE ステートメントで複数のパッケージインスタンスを構築できます。各パッケージインスタンスは、パッケージの完全に別個のコピーを表します。

TZ パッケージのインスタンスを構築する方法は 2 つあります。

- DECLARE PACKAGE ステートメントを_NEW_演算子と一緒に使用します。

```
declare package tz tzpkg;
tzpkg = _new_ tz();
```

- DECLARE PACKAGE ステートメントをそのコンストラクター構文と一緒に使用します。

```
declare package tz tzpkg();
```

関連項目

- *"TZ パッケージの使用" (SAS DS2 プログラマガイド)*

- “パッケージのコンストラクターとデストラクター” (*SAS DS2 プログラマガイド*)

演算子:

- “_NEW_ 演算子: TZ パッケージ” (2031 ページ)

システムオプション:

- “TIMEZONE= システムオプション” (*SAS システムオプション: リファレンス*)

GETLOCALTIME メソッド

現在のローカル時間を返します。

構文

variable=*package*.**GETLOCALTIME** ([*time-zone-ID*]);

必須引数

variable
要求されたローカル時間のタイムゾーン ID の値を保持する変数を指定します。

package
TZ パッケージのインスタンスを指定します。

オプション引数

time-zone-ID
必要なローカル時間のタイムゾーン ID を指定します。

例

次の例では、複数のタイムゾーンを使用しています。

```
data _null_ ;
  method init();
    dcl package tz tokyo('Asia/Tokyo')
      london('Europe/London')
      new_york('America/New_York') ;
    dcl double tokyo_time london_time new_york_time utc_time ;
    dcl integer tokyo_off london_off new_york_off ;

    tokyo_time = tokyo.getLocalTime();
    tokyo_off = tokyo.getOffset();
    london_time = london.getLocalTime() ;
    london_off = london.getOffset() ;
```

```
    new_york_time = new_york.getLocalTime() ;
    new_york_off = new_york.getOffset();

    utc_time = tokyo.getUTCTime(); /* can use any timezone */

    put utc_time = datetime. ;
    put tokyo_time = datetime. tokyo_off time5.;
    put london_time = datetime. london_off time5. ;
    put new_york_time = datetime. new_york_off time5. ;

  end;
enddata;
run;
```

次の行が SAS ログに書き込まれます。

```
utc_time=08APR15:12:48:41
tokyo_time=08APR15:21:48:41  9:00
london_time=08APR15:13:48:41  1:00
new_york_time=08APR15:08:48:41 -4:00
```

関連項目

- *"TZ パッケージの使用" (SAS DS2 プログラマガイド)*
- *"タイムゾーン ID とタイムゾーン名" (SAS 各国語サポート(NLS): リファレンスガイド)*

メソッド:

- "GETUTCTIME メソッド" (2030 ページ)

GETOFFSET メソッド

指定されたローカル時間の協定世界時(UTC)からのタイムゾーンのタイムゾーンオフセットを 返します。ローカル時間が指定されていない場合は、現在のローカル時間が使用されます。

構文

形式 1: *variable*=*package*.**GETOFFSET** ( );

形式 2: *variable*=*package*.**GETOFFSET** (*local-time*);

形式 3: *variable*=*package*.**GETOFFSET** (*time-zone-ID*);

形式 4: *variable*=*package*.**GETOFFSET** (*local-time*, *time-zone-ID*);

必須引数

variable
タイムゾーンオフセットの値を保持する変数を指定します。

package
TZ パッケージのインスタンスを指定します。

local-time
タイムゾーンオフセットを取得するために使用されるローカル時間を指定します。

デフォルト　ローカル時間が指定されていない場合は、現在のローカル時間が使用されます。

ヒント　*localTime* は、SAS 日時値です。1960 年 1 月 1 日 00:00:00 ローカル時間からの秒数として使用されます。

time-zone-ID
必要なタイムゾーンオフセットのタイムゾーン ID を指定します。

参照項目: *"タイムゾーン ID とタイムゾーン名" (SAS 各国語サポート(NLS): リファレンスガイド)*

詳細

UTC は、基準子午線(イギリスのグリニッジ付近)上の時間を指定します。UTC は、ISO 8601 規格の基本形式 *yyyymmdd*T*hhmmss*+|–*hhmm* か、ISO 8601 規格の拡張形式 *yyyy-mm-dd*T*hh*:*mm*:*ss*+|–*hh*:*mm* を使用した日時値です。

タイムゾーンオフセットは、タイムゾーンと UTC の時間差と分差を+|–*hhmm* または+|–*hh*:*mm* の形式で示します。

例

次の例では、'Asia/Tokyo'タイムゾーンから'America/New_York'タイムゾーンへのオフセットを返します。この例は、タイムゾーン ID を表現するさまざまな方法も示しています。

```
data _null_;
  method init();

    declare package tz tzone('asia/tokyo');
    dcl double new_york;
    dcl char(40) cstr;

    new_york = tzone.getOffset('America/New_York');
    put new_york time.;

    new_york = tzone.getOffset(n'America/New_York');
    put new_york time.;
```

```
    cstr = 'America/New_York';
    new_york = tzone.getOffset(cstr);
    put new_york time.;

  end;
 enddata;
run;
```

次の行が SAS ログに書き込まれます。

```
-4:00:00
-4:00:00
-4:00:00
```

関連項目

- “TZ パッケージの使用” (*SAS DS2 プログラマガイド*)

メソッド:

- “GETOFFSETUTC メソッド” (2026 ページ)

GETOFFSETUTC メソッド

指定された UTC 時間の UTC からのタイムゾーンのタイムゾーンオフセットを 返します。

構文

variable=*package*.**GETOFFSETUTC** (*UTC-time*, *time-zone-ID*);

引数なし

引数が指定されていない場合、GETOFFSETUTC メソッドは、指定された TIMEZONE=システムオプションのタイムゾーンオフセットを返します。

必須引数

variable
タイムゾーンオフセットの値を保持する変数を指定します。

package
TZ パッケージのインスタンスを指定します。

UTC-time
タイムゾーンオフセットを取得するために使用される UTC 時間を指定します。

ヒント *UTC-time* は、UTC での SAS 日時値です。これは、UTC で 1960 年 1 月 1 日 00:00:00 からの秒数として保存されます。

time-zone-ID
必要なタイムゾーンオフセットのタイムゾーン ID を指定します。

参照項目: *"タイムゾーン ID とタイムゾーン名" (SAS 各国語サポート(NLS): リファレンスガイド)*

詳細

UTC は、基準子午線(イギリスのグリニッジ付近)上の時間を指定します。UTC は、ISO 8601 規格の基本形式 *yyyymmdd*T*hhmmss*+|–*hhmm* か、ISO 8601 規格の拡張形式 *yyyy-mm-dd*T*hh*:*mm*:*ss*+|–*hh*:*mm* を使用した日時値です。

タイムゾーンオフセットは、タイムゾーンと UTC の時間差と分差を+|–*hhmm* または+|–*hh*:*mm* の形式で示します。

関連項目

- *"TZ パッケージの使用" (SAS DS2 プログラマガイド)*

メソッド:

- "GETOFFSET メソッド" (2024 ページ)

GETTIMEZONEID メソッド

現在のタイムゾーン ID を返します。

構文

variable=*package*.**GETTIMEZONEID** ( );

必須引数

package
TZ パッケージのインスタンスを指定します。

variable
TZ パッケージインスタンスのタイムゾーン ID の値を保持する変数を指定します。

詳細

タイムゾーン ID は、SAS によって定義された地域または領域の値を指定します。タイムゾーン ID の詳細については、*SAS 各国語サポート(NLS): リファレンスガイド*を参照してください。

例

次の例では、TZ パッケージを使用して期間を計算します。

```
options timezone='asia/tokyo';  /* TIMEZONE ID of origin */

data _null_;
  method route(package tz origin,
    package tz dest,
    timestamp departure,
    time duration);

    dcl nvarchar(50) tzid dest_tzid;
    dcl nvarchar(8)  tzname dest_tzname home_tzn;
    dcl double dept dur arrival utc;
    dcl double home_dept home_arr;

    declare package tz home();

    utc = origin.toUTCTime(departure);
    dur = TO_DOUBLE(duration);
    arrival = dest.toLocalTime(utc+dur);
    tzid = origin.getTimezoneID();
    tzname = origin.getTimezoneName();
    dest_tzid = dest.getTimezoneID();
    dest_tzname = dest.getTimezoneName();

    home_dept = home.toLocalTime(utc);
    home_arr = home.toLocalTime(utc+dur);
    home_tzn = home.getTimezoneName();

    put 'Time Zone: ' tzid 'to' dest_tzid;
    put 'Departure Time: ' departure datetime.  tzname  '/'
      home_dept datetime. home_tzn;
    put '   Arrial Time: ' arrival datetime.  dest_tzname  '/'
     home_arr datetime. home_tzn;
    put;

  end ;
  method init();

    /* print itinerary */
    declare package tz NRT('Asia/Tokyo');
    declare package tz ORD('America/Chicago');
    declare package tz RDU('America/New_York');
    route(NRT,ORD,timestamp '2014-10-19 10:45:00',time '11:35:00');
    route(ORD,RDU,timestamp '2014-10-19 11:03:00',time '01:56:00');
```

```
    route(RDU,ORD,timestamp '2014-10-25 07:45:00',time '02:02:00');
    route(ORD,NRT,timestamp '2014-10-25 10:50:00',time '12:55:00');

  end;
enddata;
run;
```

次の行が SAS ログに書き込まれます。

```
Time Zone:  Asia/Tokyo to America/Chicago
Departure Time:  19OCT14:10:45:00JST / 19OCT14:10:45:00JST
  Arrial Time:  19OCT14:08:20:00CDT / 19OCT14:22:20:00JST

Time Zone:  America/Chicago to America/New_York
Departure Time:  19OCT14:11:03:00CDT / 20OCT14:01:03:00JST
  Arrial Time:  19OCT14:13:59:00EDT / 20OCT14:02:59:00JST

Time Zone:  America/New_York to America/Chicago
Departure Time:  25OCT14:07:45:00EDT / 25OCT14:20:45:00JST
  Arrial Time:  25OCT14:08:47:00CDT / 25OCT14:22:47:00JST

Time Zone:  America/Chicago to Asia/Tokyo
Departure Time:  25OCT14:10:50:00CDT / 26OCT14:00:50:00JST
  Arrial Time:  26OCT14:13:45:00JST / 26OCT14:13:45:00JST
```

関連項目

- *"TZ パッケージの使用" (SAS DS2 プログラマガイド)*
- *"タイムゾーン ID とタイムゾーン名" (SAS 各国語サポート(NLS): リファレンスガイド)*

メソッド:

- "GETTIMEZONENAME メソッド" (2029 ページ)

GETTIMEZONENAME メソッド

現在のタイムゾーン名を返します。

構文

variable=*package*.**GETTIMEZONENAME** ( );

必須引数

package
TZ パッケージのインスタンスを指定します。

variable
TZ パッケージインスタンスのタイムゾーン名の値を保持する変数を指定します。

詳細

タイムゾーン名は、SAS によって定義された地域または領域の値を指定します。タイムゾーン名の詳細については、*SAS 各国語サポート(NLS): リファレンスガイド*を参照してください。

例

例については、“GETTIMEZONEID メソッド” (2027 ページ)を参照してください。

関連項目

- “TZ パッケージの使用” (*SAS DS2 プログラマガイド*)
- “タイムゾーン ID とタイムゾーン名” (*SAS 各国語サポート(NLS): リファレンスガイド*)

メソッド:

- “GETTIMEZONEID メソッド” (2027 ページ)

GETUTCTIME メソッド

現在の UTC 時間を返します。

構文

variable=*package*.**GETUTCTIME** ( );

必須引数

variable
現在の UTC 時間の値を保持する変数を指定します。

package
TZ パッケージのインスタンスを指定します。

関連項目

- "TZ パッケージの使用" (*SAS DS2 プログラマガイド*)

メソッド:

- "GETLOCALTIME メソッド" (2023 ページ)

NEW 演算子: TZ パッケージ

TZ パッケージのインスタンスを構築します。

注: かっこの前のエスケープ文字(\)は、構文にかっこが必要であることを示します。

構文

package-variable = **_NEW_ TZ**([*time-zone-id*]);

必須引数

package-variable
パッケージのインスタンスを参照できる名前を指定します。

オプション引数

time-zone-id
タイムゾーン ID を指定します。

デフォルト TIMEZONE=システムオプションで指定された値。

詳細

DS2 パッケージは、特定のインスタンスを構築して他の DS2 プログラムで使用できる変数とメソッドのコレクションです。

DECLARE PACKAGE ステートメントを使用して、TZ パッケージ変数を宣言します。TZ パッケージ変数を宣言した後、_NEW_演算子を使用して TZ パッケージのインスタンスをインスタンス化し、新しいパッケージインスタンスを TZ パッケージ変数に割り当てます。

```
declare package tz localtz;
localtz = _new_ tz( );
```

DECLARE PACKAGE と_NEW_演算子を使用して TZ パッケージを宣言およびインスタンス化する 2 ステップのプロセスのかわりに、コンストラクターメソッドとして DECLARE PACKAGE ステートメントを使用することにより、パッケージを 1 ステッ

プで宣言およびインスタンス化できます。これは、DECLARE PACKAGE ステートメントのみを使用した同じ例です。

```
declare package tz localtz( );
```

注: パッケージ変数は、すべての変数スコーピングルールの対象となります。詳細については、"パッケージ、スコープ、および有効期間" (*SAS DS2 プログラマガイド*)を参照してください。

関連項目

- "TZ パッケージの使用" (*SAS DS2 プログラマガイド*)
- "パッケージのコンストラクターとデストラクター" (*SAS DS2 プログラマガイド*)

ステートメント:

- "DECLARE PACKAGE ステートメント: TZ パッケージ" (2021 ページ)

TOISO8601 メソッド

ローカル時間をタイムゾーンオフセット付きの ISO8601 文字列に変換します。

構文

形式 1: *variable*=*package*.**TOISO8601** (*local-time*);

形式 2: *variable*=*package*.**TOISO8601** (*local-time*, *time-zone-ID*);

必須引数

variable
'2014-10-10T00:01:02.00+09:00'などの ISO8601 文字列の値を保持する変数を指定します。

package
TZ パッケージのインスタンスを指定します。

local-time
変換するローカル時間を指定します。*local-time* は、DOUBLE 形式または TIMESTAMP 形式にすることができます。

time-zone-ID
必要なローカル時間のタイムゾーン ID を指定します。タイムゾーン ID 'UTC'を使用すると、ＵＴＣ時間が指定されます。

関連項目

- "TZ パッケージの使用" (*SAS DS2 プログラマガイド*)
- "タイムゾーン ID とタイムゾーン名" (*SAS 各国語サポート(NLS): リファレンスガイド*)

メソッド:

- "TOLOCALTIME メソッド" (2033 ページ)
- "TOTIMESTAMPZ メソッド" (2034 ページ)
- "TOUTCTIME メソッド" (2035 ページ)

TOLOCALTIME メソッド

UTC 時間をローカル時間に変換します。

構文

形式 1: *variable*=*package*.**TOLOCALTIME** (*UTC-time*);

形式 2: *variable*=*package*.**TOLOCALTIME** (*UTC-time*, *time-zone-ID*);

必須引数

variable
指定された UTC 時間から変換されたローカル時間の値を保持する変数を指定します。

package
TZ パッケージのインスタンスを指定します。

UTC-time
現在の UTC 時間を、DOUBLE 形式または TIMESTAMP 形式で指定します。

time-zone-ID
必要なローカル時間のタイムゾーン ID を指定します。

詳細

UTC は、基準子午線(イギリスのグリニッジ付近)上の時間を指定します。UTC は、ISO 8601 規格の基本形式 *yyyymmdd*T*hhmmss*+|–*hhmm* か、ISO 8601 規格の拡張形式 *yyyy-mm-dd*T*hh*:*mm*:*ss*+|–*hh*:*mm* を使用した日時値です。

例

例については、“GETTIMEZONEID メソッド” (2027 ページ)を参照してください。

関連項目

- “TZ パッケージの使用” (*SAS DS2 プログラマガイド*)
- “タイムゾーン ID とタイムゾーン名” (*SAS 各国語サポート(NLS): リファレンスガイド*)

メソッド:

- “TOISO8601 メソッド” (2032 ページ)
- “TOTIMESTAMPZ メソッド” (2034 ページ)
- “TOUTCTIME メソッド” (2035 ページ)

TOTIMESTAMPZ メソッド

ローカル時間をタイムゾーン付きの TIMESTAMP 文字列に変換します。

構文

variable=*package*.**TOTIMESTAMPZ** (*local-time*[, *time-zone-ID*]);

必須引数

variable
'2014-10-14 00:01:20 Asia/Tokyo'などの文字列の値を保持する変数を指定します。

package
TZ パッケージのインスタンスを指定します。

local-time
変換するローカル時間を指定します。*local-time* は、DOUBLE 形式または TIMESTAMP 形式にすることができます。

オプション引数

time-zone-ID
必要なローカル時間のタイムゾーン ID を指定します。タイムゾーン ID 'UTC'を指定すると、ＵＴＣ時間が使用されます。

関連項目

- "TZ パッケージの使用" (*SAS DS2 プログラマガイド*)
- "タイムゾーン ID とタイムゾーン名" (*SAS 各国語サポート(NLS): リファレンスガイド*)

メソッド:

- "TOISO8601 メソッド" (2032 ページ)
- "TOLOCALTIME メソッド" (2033 ページ)
- "TOUTCTIME メソッド" (2035 ページ)

TOUTCTIME メソッド

ローカル時間を UTC 時間に変換します。

構文

形式 1: *variable*=*package*.**TOUTCTIME** (*local-time*);

形式 2: *variable*=*package*.**TOUTCTIME** (*local-time*, *time-zone-ID*);

必須引数

variable
現在の UTC 時間の値を DOUBLE 形式または TIMESTAMP 形式で保持する変数を指定します。

package
TZ パッケージのインスタンスを指定します。

local-time
変換するローカル時間を指定します。*local-time* は、DOUBLE 形式または TIMESTAMP 形式にすることができます。

time-zone-ID
必要なローカル時間のタイムゾーン ID を指定します。

詳細

UTC は、基準子午線(イギリスのグリニッジ付近)上の時間を指定します。UTC は、ISO 8601 規格の基本形式 *yyyymmdd*T*hhmmss*+|–*hhmm* か、ISO 8601 規格の拡張形式 *yyyy-mm-dd*T*hh*:*mm*:*ss*+|–*hh*:*mm* を使用した日時値です。

例

例については、“GETTIMEZONEID メソッド” (2027 ページ)を参照してください。

関連項目

- “TZ パッケージの使用” (*SAS DS2 プログラマガイド*)
- “タイムゾーン ID とタイムゾーン名” (*SAS 各国語サポート(NLS): リファレンスガイド*)

メソッド:

- “TOISO8601 メソッド” (2032 ページ)
- “TOLOCALTIME メソッド” (2033 ページ)
- “TOTIMESTAMPZ メソッド” (2034 ページ)
- “TOUTCTIME メソッド” (2035 ページ)

4 部

付録

付録 1
データ型リファレンス **2039**

付録 2
DS2 式 **2095**

付録 3
DS2 サンプルプログラム **2127**

2038

付録 1

データ型リファレンス

SAS データセットのデータ型 ... 2040
SPD Engine データセットのデータ型 ... 2041
SPD Server テーブルのデータ型 ... 2042
CAS のデータ型 ... 2044
Amazon Redshift のデータ型 ... 2046
UNIX および PC ホストでの DB2 のデータ型 ... 2048
Google BigQuery のデータ型 ... 2049
Greenplum のデータ型 ... 2051
Hive のデータ型 ... 2052
Impala のデータ型 ... 2054
JDBC のデータ型 ... 2056
MDS のデータ型 ... 2058
Microsoft SQL Server のデータ型 ... 2060
MongoDB のデータ型 ... 2062
MySQL のデータ型 ... 2064
Netezza のデータ型 ... 2066
ODBC のデータ型 ... 2067
Oracle のデータ型 ... 2069
PostgreSQL のデータ型 ... 2070
Salesforce から読み取るためのデータ型 ... 2073
Salesforce に書き込むためのデータ型 ... 2075
SAP のデータ型 ... 2076
SAP HANA のデータ型 ... 2078
SAP IQ のデータ型 ... 2080
SingleStore のデータ型 ... 2081

Snowflake のデータ型 2084
Spark のデータ型 2086
Teradata のデータ型 2088
Vertica のデータ型 2090
Yellowbrick のデータ型 2092

SAS データセットのデータ型

次の表に、SAS データセットのデータ型サポートを示します。

BINARY および VARBINARY データ型は、データ定義ではサポートされていません。

注: この表の情報は、CAS で処理されるデータには適用されません。CAS にロードされたデータは、CAS データ型に変換されます。CAS データ型の変換については、*SAS Cloud Analytic Services: ユーザーガイド*の SAS Data Connector for SAS Data Sets のドキュメントを参照してください。

表 A20.1　FedSQL データ型と SAS データセットで使用されるデータ型のマッピング

データ型定義キーワード [1]	SAS データセットのデータ型	説明	返されるデータ型
BIGINT[2]	DOUBLE	64 ビットの倍精度浮動小数点数。 **注:** 精度が低下する可能性があります。	DOUBLE
CHAR(*n*)	CHAR(*n*)	固定長の文字列。 **注:** ANSI SQL null 値を含めることはできません。	CHAR(*n*)
DATE[3]	DOUBLE	64 ビットの倍精度浮動小数点数。デフォルトでは、DATE9 SAS 出力形式が適用されます。	DOUBLE
DECIMAL|NUMERIC(*p*,*s*)[2]	DOUBLE	64 ビットの倍精度浮動小数点数。	DOUBLE
DOUBLE[2]	DOUBLE	64 ビットの倍精度浮動小数点数。	DOUBLE
FLOAT[2]	DOUBLE	64 ビットの倍精度浮動小数点数。	DOUBLE
INTEGER[2]	DOUBLE	64 ビットの倍精度浮動小数点数。	DOUBLE
NCHAR(*n*)	CHAR(*n*)	固定長の文字列。デフォルトでは、エンコーディングを Unicode UTF-8 に設定します。[4]	CHAR(*n*)

データ型定義キーワード [1]	SAS データセットのデータ型	説明	返されるデータ型
NVARCHAR(*n*)	CHAR(*n*)	固定長の文字列。デフォルトでは、エンコーディングを Unicode UTF-8 に設定します。[4]	CHAR(*n*)
REAL[2]	DOUBLE	64 ビットの倍精度浮動小数点数。	DOUBLE
SMALLINT[2]	DOUBLE	64 ビットの倍精度浮動小数点数。	DOUBLE
TIME(*p*)[3]	DOUBLE	64 ビットの倍精度浮動小数点数。デフォルトでは、TIME8 SAS 出力形式が適用されます。	DOUBLE
TIMESTAMP(*p*)[3]	DOUBLE	64 ビットの倍精度浮動小数点数。デフォルトでは、DATETIME19.2 SAS 出力形式が適用されます。	DOUBLE
TINYINT[2]	DOUBLE	64 ビットの倍精度浮動小数点数。	DOUBLE
VARCHAR(*n*)	CHAR(*n*)	固定長の文字列。 **注:** ANSI SQL null 値を含めることはできません。	CHAR(*n*)

1 データ型のマッピング方法を制御する CT_PRESERVE=接続引数は、データ型を定義できるかどうかに影響する可能性があります。値 FORCE (デフォルト)および FORCE_COL_SIZE は、データ型を定義できるかどうかには影響しません。要求されたデータ型がデータソースにネイティブでない場合、または指定された精度またはスケールがデータソースの範囲内にない場合、値 STRICT および SAFE はエラーになる可能性があります。

2 日付と時間の SAS 出力形式を数値データ型に適用しないでください。日付と時間の値には、DATE、TIME、または TIMESTAMP データ型を使用します。

3 値は倍精度浮動小数点数として保存されるため、算術式で値を使用できます。

4 UTF-8 は MBCS エンコーディングです。動作環境に応じて、UTF-8 文字の幅は 1 バイトから 4 バイトまでさまざまです。保存するマルチバイト文字の最大数である *n* の値に、動作環境の最大長を掛けます。UTF-8 から Wlatin2 などにトランスコーディングする場合、可変長(バイト単位)では値を保持するのに十分でない可能性があり、その結果、文字データが切り捨てられることに注意してください。

SPD Engine データセットのデータ型

次の表に、SPD Engine データセットのデータ型サポートを示します。

BINARY、DECIMAL、NUMERIC、NCHAR、NVARCHAR、VARBINARY データ型は、データ型定義ではサポートされていません。

注: この表の情報は、CAS で処理されるデータには適用されません。CAS にロードされたデータは、CAS データ型に変換されます。CAS データ型の変換については、*SAS Cloud Analytic Services: ユーザーガイド*の SAS Data Connector for SPD Engine Files のドキュメントを参照してください。

表 A20.2　FedSQL データ型と SPD Engine データセットで使用されるデータ型のマッピング

データ型定義キーワード	SPD データセットのデータ型	説明	返されるデータ型
BIGINT[1]	DOUBLE	64 ビットの倍精度浮動小数点数。 **注:** 精度が低下する可能性があります。	DOUBLE
CHAR(*n*)	CHAR(*n*)	固定長の文字列。 **注:** ANSI SQL null 値を含めることはできません。	CHAR(*n*)
DATE[2]	DOUBLE	64 ビットの倍精度浮動小数点数。デフォルトでは、DATE9 SAS 出力形式が適用されます。	DOUBLE
DOUBLE[1]	DOUBLE	64 ビットの倍精度浮動小数点数。	DOUBLE
FLOAT[1]	DOUBLE	64 ビットの倍精度浮動小数点数。	DOUBLE
INTEGER[1]	DOUBLE	64 ビットの倍精度浮動小数点数。	DOUBLE
REAL[1]	DOUBLE	64 ビットの倍精度浮動小数点数。	DOUBLE
SMALLINT[1]	DOUBLE	64 ビットの倍精度浮動小数点数。	DOUBLE
TIME(*p*)[2]	DOUBLE	64 ビットの倍精度浮動小数点数。デフォルトでは、TIME8 SAS 出力形式が適用されます。	DOUBLE
TIMESTAMP(*p*)[2]	DOUBLE	64 ビットの倍精度浮動小数点数。デフォルトでは、DATETIME19.2 SAS 出力形式が適用されます。	DOUBLE
TINYINT[1]	DOUBLE	64 ビットの倍精度浮動小数点数。	DOUBLE
VARCHAR(*n*)	CHAR(*n*)	固定長の文字列。 **注:** ANSI SQL null 値を含めることはできません。	CHAR(*n*)

1 日付と時間の SAS 出力形式を数値データ型に適用しないでください。日付と時間の値には、DATE、TIME、または TIMESTAMP データ型を使用します。
2 値は倍精度浮動小数点数として保存されるため、算術式で値を使用できます。

SPD Server テーブルのデータ型

次の表に、SPD Server テーブルのデータ型サポートを示します。

BINARY、DECIMAL、NUMERIC、NCHAR、NVARCHAR、VARBINARY データ型は、データ型定義ではサポートされていません。

注: このデータソースは CAS サーバーではサポートされていません。

表 A20.3 FedSQL データ型と SPD Server テーブルで使用されるデータ型のマッピング

データ型定義キーワード	SPD データセットのデータ型	説明	返されるデータ型
BIGINT[1]	DOUBLE	64 ビットの倍精度浮動小数点数。 **注:** 精度が低下する可能性があります。	DOUBLE
CHAR(*n*)	CHAR(*n*)	固定長の文字列。 **注:** ANSI SQL null 値を含めることはできません。	CHAR(*n*)
DATE[2]	DOUBLE	64 ビットの倍精度浮動小数点数。デフォルトでは、DATE9 SAS 出力形式が適用されます。	DOUBLE
DOUBLE[1]	DOUBLE	64 ビットの倍精度浮動小数点数。	DOUBLE
FLOAT[1]	DOUBLE	64 ビットの倍精度浮動小数点数。	DOUBLE
INTEGER[1]	DOUBLE	64 ビットの倍精度浮動小数点数。	DOUBLE
REAL[1]	DOUBLE	64 ビットの倍精度浮動小数点数。	DOUBLE
SMALLINT[1]	DOUBLE	64 ビットの倍精度浮動小数点数。	DOUBLE
TIME(*p*)[2]	DOUBLE	64 ビットの倍精度浮動小数点数。デフォルトでは、TIME8 SAS 出力形式が適用されます。	DOUBLE
TIMESTAMP(*p*)[2]	DOUBLE	64 ビットの倍精度浮動小数点数。デ	DOUBLE

データ型定義キーワード	SPD データセットのデータ型	説明	返されるデータ型
		フォルトでは、DATETIME19.2 SAS 出力形式が適用されます。	
TINYINT[1]	DOUBLE	64 ビットの倍精度浮動小数点数。	DOUBLE
VARCHAR(*n*)	CHAR(*n*)	固定長の文字列。 **注:** ANSI SQL null 値を含めることはできません。	CHAR(*n*)

1 日付と時間の SAS 出力形式を数値データ型に適用しないでください。日付と時間の値には、DATE、TIME、または TIMESTAMP データ型を使用します。
2 値は倍精度浮動小数点数として保存されるため、算術式で値を使用できます。

CAS のデータ型

次の表に、CAS テーブルのデータ型サポートを示します。

BINARY、DECIMAL/NUMERIC、および REAL データ型はサポートされていません。

DS2 で BINARY 列を含む CAS テーブルを読み取ったり更新したりするには、最初にテーブルから BINARY 列を削除する必要があります。DS2 は、CAS テーブルで BINARY 列を検出すると、エラーを返します。

表 A20.4 FedSQL データ型と CAS テーブルで使用されるデータ型のマッピング

データ型定義キーワード [1]	CAS データ型	説明	返されるデータ型
BIGINT[2]	INT64	19 桁の精度で大きな符号付きの正確な整数。整数の範囲は-9,223,372,036,854,775,808 から 9,223,372,036,854,775,807 までです。整数データ型は小数値を保存しません。小数部分は破棄されます。 CAS テーブルの BIGINT 列で可能な限り最も負の BIGINT (INT64)値に対応する DS2 値は、SAS 欠損値または null 値として扱われます。	BIGINT
CHAR(*n*)	CHAR(*n*)	固定長の文字列。文字列値の長さが *n* 未満の場合、値は *n* まで空白で埋められます。[4]	CHAR(*n*)[5]

データ型定義キーワード [1]	CAS データ型	説明	返されるデータ型
		注: 文字列内の ANSI SQL null 値は、SAS 欠損値に変換されます。	
DATE[3]	DOUBLE	64 ビットの倍精度浮動小数点数。デフォルトでは、DATE9 が適用されます。SAS 出力形式。	DOUBLE
DOUBLE[2]	DOUBLE	64 ビットの倍精度浮動小数点数。	DOUBLE
FLOAT[2]	DOUBLE	64 ビットの倍精度浮動小数点数。	DOUBLE
INTEGER[2]	INT32	10 桁の精度で通常の符号付きの正確な整数。整数の範囲は-2,147,483,648 から 2,147,483,647 までです。整数データ型は小数値を保存しません。小数部分は破棄されます。 CAS テーブルの INTEGER 列で可能な限り最も負の INTEGER (INT32)値に対応する DS2 値は、SAS 欠損値または null 値として扱われます。	INTEGER
NCHAR(*n*)	CHAR(*n*)	固定長の各国文字列。文字列値の長さが *n* 未満の場合、値は *n* まで空白で埋められます。[4] **注:** 文字列内の ANSI SQL null 値は、SAS 欠損値に変換されます。	CHAR(*n*)
NVARCHAR(*n*)	VARCHAR(*n*)	可変長の文字列。*n* は、保存できる文字列の最大長です。[6] **注:** 文字列内の ANSI SQL null 値は、SAS 欠損値に変換されます。	VARCHAR(*n*)
SMALLINT[2]	INT32	小さな符号付きの正確な整数。	INTEGER
TIME(*p*)[3]	DOUBLE	64 ビットの倍精度浮動小数点数。デフォルトでは、TIME8 が適用されます。SAS 出力形式。	DOUBLE
TIMESTAMP(*p*)[3]	DOUBLE	64 ビットの倍精度浮動小数点数。デフォルトでは、DATETIME25.6 SAS 出力形式が適用されます。	DOUBLE
TINYINT[2]	INT32	非常に小さな符号付きの正確な整数。	INTEGER
VARBINARY	VARBINARY	可変長のバイナリ不透明データ。このデータ型は、画像データ、音声、ドキュメント、およびその他の非構造化データを保存するために使用されます。	VARBINARY
VARCHAR(*n*)	VARCHAR(*n*)	可変長の文字列。*n* は、保存できる文字列の最大長です。[6]	VARCHAR(*n*)[5]

データ型定義キーワード [1]	CAS データ型	説明	返されるデータ型
		注: 文字列内の ANSI SQL null 値は、SAS 欠損値に変換されます。	

1 データ型のマッピング方法を制御する CT_PRESERVE=接続引数は、データ型を定義できるかどうかに影響する可能性があります。値 FORCE (デフォルト)および FORCE_COL_SIZE は、データ型を定義できるかどうかには影響しません。要求されたデータ型がデータソースにネイティブでない場合、または指定された精度またはスケールがデータソースの範囲内にない場合、値 STRICT および SAFE はエラーになる可能性があります。

2 日付と時間の SAS 出力形式を数値データ型に適用しないでください。日付と時間の値には、DATE、TIME、または TIMESTAMP データ型を使用します。

3 値は倍精度浮動小数点数として保存されるため、算術式で値を使用できます。

4 DS2 CHAR および NCHAR 文字データ型の場合、 *n* の値は保存する文字数です。CAS CHAR および NCHAR 文字型の場合、*n* の値は保存するバイト数です。CAS テーブルは UTF-8 文字セットを使用します。UTF-8 は、マルチバイト文字セットエンコーディングです。UTF-8 文字の幅は 1 バイトから 4 バイトまでさまざまです。DS2 文字値が CAS 文字列に保存されるときに、その値が UTF-8 エンコード表現で *n* を超えるバイト数を必要とする場合、文字列データは切り捨てられます。DS2 文字値を使用して CAS 文字列を読み取る場合、その値がアクティブなセッションエンコーディングで *n* を超える文字数を必要とする場合、文字列データは切り捨てられます。

5 DS2 文字型は、デフォルトでセッションエンコーディングを使用します。このデータ型を使用すると、列を定義するときに代替文字セットエンコーディングを指定できます。DS2 データ型が DS2 CAS アクションで UTF-8 以外のエンコーディングを指定している場合、データが CAS に書き込まれるか、CAS から読み取られるときに、データは UTF-8 との間でトランスコードされます。

6 DS2 および CAS の NVARCHAR および VARCHAR データ型の場合、*n* の値は保存する文字数です。

Amazon Redshift のデータ型

次の表に、Amazon Redshift データベースのデータ型サポートを示します。

BINARY、TINYINT、および VARBINARY データ型は、データ型定義ではサポートされていません。

Redshift データ型に関するデータソース固有の情報については、Amazon Redshift データベースのドキュメントを参照してください。

注: この表の情報は、CAS で処理されるデータには適用されません。CAS にロードされたデータは、CAS データ型に変換されます。CAS データ型の変換については、*SAS Cloud Analytic Services: ユーザーガイド*の SAS Data Connector for Amazon Redshift のドキュメントを参照してください。

表 A20.5 FedSQL データ型と Amazon Redshift で使用されるデータ型のマッピング

データ型定義キーワード [1]	Amazon Redshift データ型	説明	返されるデータ型
BIGINT	BIGINT	符号付き 8 バイト整数。	BIGINT
[2]	BOOLEAN	論理ブール(true/false)値。	[2]

データ型定義キーワード [1]	Amazon Redshift データ型	説明	返されるデータ型
CHAR(*n*)	CHAR	固定長の文字列。	CHAR(*n*)
DATE	DATE	日付値。	DATE
DECIMAL|NUMERIC(*p*,*s*)	DECIMAL、NUMERIC	符号付き固定小数点 10 進数。	DECIMAL|NUMERIC(*p*,*s*)[4]
DOUBLE	DOUBLE PRECISION	符号付き倍精度浮動小数点数。	DOUBLE
FLOAT	FLOAT	符号付き倍精度浮動小数点数。	DOUBLE
INTEGER	INTEGER	通常の符号付きの正確な整数。	INTEGER
NCHAR	NCHAR	固定長の Unicode 文字列。	CHAR
NVARCHAR	NVARCHAR	可変長の Unicode 文字列。	CHAR
REAL	REAL	符号付き単精度浮動小数点数。	REAL
SMALLINT	SMALLINT	小さな符号付きの正確な整数。	SMALLINT
TIME(*p*)	[3]	時間値。	TIMESTAMP
TIMESTAMP(*p*)	TIMESTAMP	日時値。	TIMESTAMP(*p*)
[2]	TIMESTAMPZ	タイムゾーン付き日時値。	[2]
VARCHAR(*n*)	VARCHAR	可変長の文字列。	CHAR

1 データ型のマッピング方法を制御する CT_PRESERVE=接続引数は、データ型を定義できるかどうかに影響する可能性があります。値 FORCE (デフォルト)および FORCE_COL_SIZE は、データ型を定義できるかどうかには影響しません。要求されたデータ型がデータソースにネイティブでない場合、または指定された精度またはスケールがデータソースの範囲内にない場合、値 STRICT および SAFE はエラーになる可能性があります。

2 Amazon Redshift データ型は定義できません。データが取得されると、ネイティブデータ型が同様のデータ型にマッピングされます。

3 Amazon Redshift は、TIME(*p*)データ型をサポートしていません。TIME 型の列を定義すると、かわりに TIMESTAMP 型の列が作成されます。

4 FedSQL では、DECIMAL データ型は、データソースに完全に渡すことができる要求でサポートされています。データソースに完全に渡すことができない要求の DECIMAL 列は、DOUBLE として処理されるため、精度に影響を与える可能性があります。詳細については、*"データ型" (SAS FedSQL 言語リファレンス)*を参照してください。

UNIX および PC ホストでの DB2 のデータ型

次の表に、UNIX および PC ホストでの DB2 データベースのデータ型サポートを示します。

NCHAR、NVARCHAR、および TINYINT データ型は、データ型定義ではサポートされていません。

DB2 データベースのデータ型に関するデータソース固有の情報については、DB2 データベースのドキュメントを参照してください。

注: この表の情報は、CAS で処理されるデータには適用されません。CAS にロードされたデータは、CAS データ型に変換されます。CAS データ型の変換については、*SAS Cloud Analytic Services: ユーザーガイド*の SAS Data Connector for DB2 のドキュメントを参照してください。

表 A20.6 FedSQL データ型と UNIX および PC ホストでの DB2 で使用されるデータ型のマッピング

データ型定義キーワード [1]	DB2 データ型	説明	返されるデータ型
BIGINT	BIGINT	大きな符号付きの正確な整数。	BIGINT
BINARY(*n*)	CHAR(*n*) FOR BIT DATA	固定長のバイナリ文字列。	BINARY(*n*)
[2]	BLOB(*n*[K|M|G])	可変長のバイナリラージオブジェクト文字列。	[2]
CHAR(*n*)	CHAR(*n*)	固定長の文字列。	CHAR(*n*)
[2]	CLOB(*n*[K|M|G])	可変長文字のラージオブジェクト文字列。	[2]
DATE	DATE	日付値。	DATE
[2]	DBCLOB(*n*[K|M|G])	可変長のダブルバイト文字ラージオブジェクト。	[2]
DECIMAL|NUMERIC(*p*,*s*)	DECIMAL|NUMERIC(*p*,*s*)	符号付き固定小数点 10 進数。	DECIMAL|NUMERIC(*p*,*s*)[3]

データ型定義キーワード [1]	DB2 データ型	説明	返されるデータ型
DOUBLE	DOUBLE	符号付き倍精度浮動小数点数。	DOUBLE
FLOAT	FLOAT(*p*)	符号付き倍精度浮動小数点数。	DOUBLE
[2]	GRAPHIC(*n*)	固定長のグラフィック文字列。	[2]
INTEGER	INTEGER	通常の符号付きの正確な整数。	INTEGER
[2]	LONG VARCHAR [FOR BIT DATA]	可変長文字またはバイナリ文字列。	[2]
[2]	LONG VARGRAPHIC(*n*)	可変長のグラフィック文字列。	[2]
REAL	REAL	符号付き単精度浮動小数点数。	REAL
SMALLINT	SMALLINT	小さな符号付きの正確な整数。	SMALLINT
TIME(*p*)	TIME(*p*)	時間値。	TIME(*p*)
TIMESTAMP(*p*)	TIMESTAMP(*p*)	日時値。	TIMESTAMP(*p*)
VARBINARY(*n*)	VARCHAR(*n*) FOR BIT DATA	可変長のバイナリ文字列。	VARBINARY(*n*)
VARCHAR(*n*)	VARCHAR(*n*)	可変長の文字列。	VARCHAR(*n*)
[2]	VARGRAPHIC(*n*)	可変長のグラフィック文字列。	[2]

1 データ型のマッピング方法を制御する CT_PRESERVE=接続引数は、データ型を定義できるかどうかに影響する可能性があります。値 FORCE (デフォルト)および FORCE_COL_SIZE は、データ型を定義できるかどうかには影響しません。要求されたデータ型がデータソースにネイティブでない場合、または指定された精度またはスケールがデータソースの範囲内にない場合、値 STRICT および SAFE はエラーになる可能性があります。

2 DB2 データ型は定義できません。データが取得されると、ネイティブデータ型が同様のデータ型にマッピングされます。

3 FedSQL では、DECIMAL データ型は、データソースに完全に渡すことができる要求でサポートされています。データソースに完全に渡すことができない要求の DECIMAL 列は、DOUBLE として処理されるため、精度に影響を与える可能性があります。詳細については、*"データ型" (SAS FedSQL 言語リファレンス)*を参照してください。

Google BigQuery のデータ型

次の表に、Google BigQuery プロジェクトのデータ型サポートを示します。

Google BigQuery の ARRAY、STRUCT、および GEOGRAPHY データ型はサポートされていません。

Google BigQuery のデータ型に関するデータソース固有の情報については、Google BigQuery のドキュメントを参照してください。

注: この表の情報は、CAS で処理されるデータには適用されません。CAS にロードされたデータは、CAS データ型に変換されます。CAS データ型の変換については、*SAS Cloud Analytic Services: ユーザーガイド*の SAS Data Connector for Google BigQuery のドキュメントを参照してください。

表 A20.7 FedSQL データ型と Google BigQuery で使用されるデータ型のマッピング

データ型定義キーワード	BigQuery データ型	BigQuery の説明	返されるデータ型
BIGINT	INT64	大きな符号付きの正確な整数で、範囲は-9,223,372,036,854,775,808 から 9,223,372,036,854,775,807 までです。	BIGINT
[1]	BIGNUMERIC(*p,s*)	倍精度浮動小数点数。	DOUBLE[4, 5]
BINARY	BYTES	可変長のバイナリ文字列。	VARBINARY[2]
[1]	BOOL	True または False として表されるブール値(大文字と小文字を区別しない)。	[1]
CHAR(*n*)	STRING	可変長の Unicode 文字列。	VARCHAR[3]
DATE	DATE	0001-01-01 から 9999-12-31 までの日付値。	DATE
DOUBLE	FLOAT64	倍精度浮動小数点数。	DOUBLE
FLOAT	FLOAT64	倍精度浮動小数点数。	DOUBLE
INTEGER	INT64	通常の符号付きの正確な整数。	BIGINT
NCHAR	STRING	固定長の Unicode 文字列。	VARCHAR
NUMERIC(*p,s*)	NUMERIC(*p,s*)	倍精度浮動小数点数。	DOUBLE[4, 5]
NVARCHAR	STRING	可変長の Unicode 文字列。	VARCHAR
SMALLINT	INT64	小さな符号付きの正確な整数。	BIGINT
TIME(*p*)	TIME	00:00:00 から 23:59:59.999999 までの範囲の時間、分、および秒の時間値。	TIME(*p*)
[1]	TIMESTAMP	日時値。	[1]
TIMESTAMP(*p*)	DATETIME	タイムゾーンや夏時間などの規則に依存しない、マイクロ秒精度の日時値。サポートさ	TIMESTAMP(p)

データ型定義キーワード	BigQuery データ型	BigQuery の説明	返されるデータ型
		れている値は、0001-01-01 00:00:00 から 9999-12-31 23:59:59.999999 UTC です。	
TINYINT	INT64	非常に小さな符号付きの正確な整数。	BIGINT
VARBINARY(*n*)	BYTES	可変長のバイナリ文字列。	VARBINARY[2]
VARCHAR(*n*)	STRING	可変長の Unicode 文字列。	VARCHAR[3]

1 BigQuery データ型は定義できません。データが取得されると、ネイティブデータ型が同様のデータ型にマッピングされます。

2 長さは、MAX_BINARY_LEN= LIBNAME オプションによって決定されます。MAX_BINARY_LEN=が設定されていない場合、デフォルトの長さは 2,000 バイトです。

3 MAX_CHAR_LEN= LIBNAME オプションまたは SCANSTRINGCOLUMNS テーブルオプションが設定されていない限り、デフォルトの長さは 16,384 文字です。

4 2024.09 以降、Google BigQuery NUMERIC(p,s)および BIGNUMERIC(p,s)列は DOUBLE として扱われ、BigQuery FLOAT64 型にマッピングされます。FETCH_NUMERIC_TYPE=テーブルオプションは、これらの型を標準の NUMERIC(p,s)型として扱うように要求できるように提供されています。

5 FETCH_NUMERIC_TYPE=NUMERIC が設定されている場合、NUMERIC(p,s)および BIGNUMERIC(p,s)型は、データソースに完全に渡すことができる要求でサポートされています。データソースに完全に渡すことができない要求の NUMERIC(p,s)および BIGNUMERIC(p,s)列は、DOUBLE として処理されます。詳細については、*"データ型とは" (SAS DS2 プログラマガイド)*を参照してください。

Greenplum のデータ型

次の表に、Greenplum データベースのデータ型サポートを示します。

BINARY、NCHAR、NVARCHAR、TINYINT データ型は、データ型定義ではサポートされていません。

Greenplum データ型に関するデータソース固有の情報については、Greenplum データベースのドキュメントを参照してください。

注: この表の情報は、CAS で処理されるデータには適用されません。CAS にロードされたデータは、CAS データ型に変換されます。CAS データ型の変換については、*SAS Cloud Analytic Services: ユーザーガイド*の SAS Data Connector for Greenplum のドキュメントを参照してください。

表 A20.8 FedSQL データ型と Greenplum で使用されるデータ型のマッピング

データ型定義キーワード[1]	Greenplum データ型	説明	返されるデータ型
BIGINT	INT8	大きな符号付きの正確な整数。	BIGINT

データ型定義キーワード [1]	Greenplum データ型	説明	返されるデータ型
CHAR(*n*)	CHAR(*n*)	固定長の文字列。	CHAR(*n*)
DATE	DATE	日付値。	DATE
DECIMAL|NUMERIC(*p*,*s*)	DECIMAL(*p*,*s*)	符号付き固定小数点 10 進数。	DECIMAL|NUMERIC(*p*,*s*)[3]
DOUBLE	DOUBLE	浮動小数点数。	DOUBLE
FLOAT	FLOAT(*p*)	浮動小数点数。	DOUBLE
INTEGER	INTEGER	通常の符号付きの正確な整数。	INTEGER
REAL	REAL	浮動小数点数。	REAL
SMALLINT	INT8	小さな符号付きの正確な整数。	SMALLINT
TIME(*p*)	TIME(*p*)	時間、分、および秒の時間値。	TIME(*p*)[2]
TIMESTAMP(*p*)	TIMESTAMP(*p*)	日時値。	TIMESTAMP(*p*)
VARBINARY(*n*)	BYTEA	可変長のバイナリ文字列。	VARBINARY(*n*)
VARCHAR(*n*)	VARCHAR(*n*)	可変長の文字列。	VARCHAR(*n*)

1 データ型のマッピング方法を制御する CT_PRESERVE=接続引数は、データ型を定義できるかどうかに影響する可能性があります。値 FORCE (デフォルト)および FORCE_COL_SIZE は、データ型を定義できるかどうかには影響しません。要求されたデータ型がデータソースにネイティブでない場合、または指定された精度またはスケールがデータソースの範囲内にない場合、値 STRICT および SAFE はエラーになる可能性があります。

2 Greenplum サーバーとの通信に使用される ODBC スタイルのインターフェイスにより、サーバーからクライアントへのデータ転送で秒の端数が失われます。

3 FedSQL では、DECIMAL データ型は、データソースに完全に渡すことができる要求でサポートされています。データソースに完全に渡すことができない要求の DECIMAL 列は、DOUBLE として処理されるため、精度に影響を与える可能性があります。詳細については、*"データ型" (SAS FedSQL 言語リファレンス)*を参照してください。

Hive のデータ型

次の表に、Hive のデータ型サポートを示します。

NCHAR および NVARCHAR データ型は、データ定義には使用できません。

Hive 複合データ型は読み取ることができますが、データ定義には使用できません。

Hive データ型に関するデータソース固有の情報については、Hive データベースのドキュメントを参照してください。

注: この表の情報は、CAS で処理されるデータには適用されません。CAS にロードされたデータは、CAS データ型に変換されます。CAS データ型の変換については、*SAS Cloud Analytic Services: ユーザーガイド*の SAS Data Connector for Hadoop のドキュメントを参照してください。

表 A20.9 FedSQL データ型と Hive で使用されるデータ型のマッピング

データ型定義キーワード	Hive データ型	説明	返されるデータ型
[1, 4]	ARRAY<*data-type*>	整数の配列(インデックス可能なリスト)。	STRING[8]
BIGINT	BIGINT	-9,223,372,036,854,775,807 から 9,223,372,036,854,775,807 までの符号付き 8 バイト整数。	BIGINT
BINARY(*n*)	BINARY	最大 32K の可変長のバイナリ文字列。	BINARY
[1]	BOOLEAN	テキストの true または false の値。	TINYINT
CHAR(*n*)	CHAR(n)	最大 255 文字の文字列。 255 文字を超える値で CHAR を指定すると、列はかわりに VARCHAR(n)として作成されます。	CHAR
DATE	DATE[2]	ANSI SQL の日付型。	DATE
DECIMAL/ NUMERIC(*p*,*s*)	DECIMAL	38 桁の精度の固定小数点 10 進数。	DECIMAL(*p*,*s*)[3]
DOUBLE	DOUBLE	8 バイトの倍精度浮動小数点数。	DOUBLE
FLOAT	FLOAT	8 バイトの倍精度浮動小数点数。	DOUBLE
INTEGER	INTEGER	10 桁の精度で通常サイズの符号付きの正確な整数。整数の範囲は-2,147,483,647 から 2,147,483,647 までです。	INTEGER
[1, 4]	MAP<*primitive-type*, *data-type*>	キーと値のペアの連想配列。	STRING[8]
REAL	DOUBLE	64 ビットの倍精度浮動小数点数。	DOUBLE
SMALLINT	SMALLINT	-32,767 から 32,767 までの符号付き 2 バイト整数。	SMALLINT
[1, 4]	STRING	可変長の文字列。	VARCHAR(n)[6]

データ型定義キーワード	Hive データ型	説明	返されるデータ型
[1, 4]	STRUCT<*col-name*: *data-type*>	確立された列要素とデータ型を持つ構造。列要素とデータ型は、ダブルドット(:)表記を使用してマッピングされます。	STRING[8]
TIME(*p*)	[7]	時間値。	STRING
TIMESTAMP(*p*)	TIMESTAMP	オプションのナノ秒精度の UNIX タイムスタンプ。	TIMESTAMP[(*p*)]
TINYINT	TINYINT	-127 から 127 までの符号付き 1 バイト整数。	TINYINT
[1, 4]	UNION<*data-type*, *data-type-n*>	指定されたいくつかのデータ型の 1 つを保持できる型。	STRING[8]
VARCHAR(*n*)[5]	VARCHAR(*n*)	可変長の文字列。	STRING
VARBINARY	BINARY	最大 32K の可変長のバイナリ文字列。	BINARY

1 Hive データ型は定義できません。データが取得されると、ネイティブデータ型が同様のデータ型にマッピングされます。

2 サポートされている日付値は、1582 年 10 月 15 日から 9999 年 12 月 31 日までです。1582 より前の年を含む日付値はエラーを返します。9999 より後の日付値は null 値として読み戻されます。

3 FedSQL では、DECIMAL データ型は、データソースに完全に渡すことができる要求でサポートされています。データソースに完全に渡すことができない要求の DECIMAL 列は、DOUBLE として処理されるため、精度に影響を与える可能性があります。詳細については、*"データ型" (SAS FedSQL 言語リファレンス)*を参照してください。

4 文字列および Hive 複合型は VARCHAR(*n*)としてロードされ、その長さは DBMAX_TEXT=データソース接続オプションによって決定されます。

5 65,535 文字を超える値を持つ VARCHAR 列は、STRING として作成されます。

6 SASFMT TableProperties は、STRING 列を読み取るときに適用されます。

7 Hive は TIME(p)データ型をサポートしていません。データが Hive から読み取られると、SAS TIME8.出力形式を指定する SASFMT TableProperties が定義されている STRING 列が、TIME(*p*)データ型に変換されます。TIME 型がデータ定義に使用される場合、SASFMT TableProperties を使用して STRING 列にマッピングされます。秒の端数は保持されません。SASFMT TableProperties の詳細については、*SAS/ACCESS for Relational Databases: リファレンス*の"SAS Table Properties for Hive and HADOOP"を参照してください。

8 複合型 ARRAY、MAP、STRUCT、UNION は、基になる複合型の STRING 表現として読み取られます。ARRAY 値は、角かっこで囲まれて読み戻されます(例: [1、2、4])。STRUCT と MAP の値は、中かっこで囲まれて読み戻されます。例: {"firstname":"robert","nickname":"bob"}。

Impala のデータ型

次の表に、Impala のデータ型サポートを示します。

BINARY、DECIMAL(*p*.*s*)/NUMERIC、NCHAR、NVARCHAR、VARBINARY データ型は、データ定義には使用できません。

Impala データ型に関するデータソース固有の情報については、ベンダーのドキュメントを参照してください。

注: この表の情報は、CAS で処理されるデータには適用されません。CAS にロードされたデータは、CAS データ型に変換されます。CAS データ型の変換については、*SAS Cloud Analytic Services: ユーザーガイド*の SAS Data Connector for Impala のドキュメントを参照してください。

表 A20.10 FedSQL データ型と Impala で使用されるデータ型のマッピング

データ型定義キーワード	Impala データ型	説明	返されるデータ型
BIGINT	BIGINT	-9,223,372,036,854,775,807 から 9,223,372,036,854,775,807 までの符号付き 8 バイト整数。	BIGINT
CHAR(*n*)	CHAR[1]	最大 255 文字の固定長文字列。	CHAR
DATE	[2]	ANSI SQL の日付型。	TIMESTAMP
DOUBLE	DOUBLE	8 バイトの倍精度浮動小数点数。	DOUBLE
FLOAT	FLOAT	8 バイトの倍精度浮動小数点数。	DOUBLE
INTEGER	INT	-2,147,483,647 から 2,147,483,647 までの符号付き 4 バイト整数。	INTEGER
REAL	DOUBLE	8 バイトの倍精度浮動小数点数。	DOUBLE
SMALLINT	SMALLINT	-32,767 から 32,767 までの符号付き 2 バイト整数。	SMALLINT
TIME	[2]	時間値。	TIMESTAMP
TIMESTAMP	TIMESTAMP	オプションのナノ秒精度の UNIX タイムスタンプ。	TIMESTAMP
TINYINT	TINYINT	-127 から 127 までの符号付き 1 バイト整数。	TINYINT

データ型定義キーワード	Impala データ型	説明	返されるデータ型
VARCHAR(*n*)	VARCHAR[1]	可変長の文字列。	VARCHAR

1 このデータ型のサポートは、CDH 5.2 以降で使用できます。CHAR および VARCHAR 型をサポートしない以前の CDH バージョンを使用する環境では、CHAR または VARCHAR として定義された列は STRING にマッピングされます。
2 Impala はこのデータ型をサポートしていません。DATE 列または TIME 列が定義されると、TIMESTAMP 型の列として作成されます。

JDBC のデータ型

次の表に、JDBC のデータ型サポートを示します。

JDBC データ型に関するデータソース固有の情報については、データベースのドキュメントを参照してください。

注: この表の情報は、CAS で処理されるデータには適用されません。CAS にロードされたデータは、CAS データ型に変換されます。CAS データ型の変換については、*SAS Cloud Analytic Services: ユーザーガイド*の SAS Data Connector for JDBC のドキュメントを参照してください。

表 A20.11 FedSQL データ型と JDBC でサポートされるデータ型のマッピング

データ型定義キーワード	JDBC SQL 識別子	説明	返されるデータ型
BIGINT	SQL_BIGINT	大きな符号付きの正確な整数。	BIGINT
BINARY(*n*)	SQL_BINARY	固定長のバイナリ文字列。	BINARY(*n*)
[1]	SQL_BIT	シングルビットのバイナリ値。	[1]
CHAR(*n*)[2]	SQL_CHAR	固定長の文字列。	CHAR(*n*)
DATE	SQL_TYPE_DATE	日付値。	DATE
DECIMAL|NUMERIC(*p*,*s*)	SQL_DECIMAL|SQL_NUMERIC	符号付き固定小数点 10 進数。	DECIMAL|NUMERIC(*p*,*s*)[3]
DOUBLE	SQL_DOUBLE	符号付き倍精度浮動小数点数。	DOUBLE

データ型定義キーワード	JDBC SQL 識別子	説明	返されるデータ型
FLOAT	SQL_FLOAT	符号付き倍精度浮動小数点数。	DOUBLE
[1]	SQL_GUID	グローバル一意識別子。	[1]
INTEGER	SQL_INTEGER	通常の符号付きの正確な整数。	INTEGER
[1]	SQL_INTERVAL	2 つの年、月、日、日付、時間の間隔。	[1]
[1]	SQL_LONGVARBINARY	可変長のバイナリ文字列。	[1]
[1]	SQL_LONGVARCHAR	可変長の Unicode 文字列。	[1]
NCHAR(*n*)	SQL_WCHAR	固定長の Unicode 文字列。	NCHAR(*n*)
NVARCHAR(*n*)	SQL_WVARCHAR	可変長の Unicode 文字列。	NVARCHAR(*n*)
REAL	SQL_REAL	符号付き単精度浮動小数点数。	REAL
SMALLINT	SQL_SMALLINT	小さな符号付きの正確な整数。	SMALLINT
TIME(*p*)	SQL_TYPE_TIME	時間値。	TIME(*p*)
TIMESTAMP(*p*)	SQL_TYPE_TIMESTAMP	日時値。	TIMESTAMP(*p*)
TINYINT	SQL_TINYINT	非常に小さな符号付きの正確な整数。	TINYINT
VARBINARY(*n*)	SQL_VARBINARY	可変長のバイナリ文字列。	VARBINARY(*n*)
VARCHAR(*n*)[2]	SQL_VARCHAR	可変長の文字列。	VARCHAR(*n*)

データ型定義キーワード	JDBC SQL 識別子	説明	返されるデータ型
[1]	SQL_WLONGVARCHAR	可変長の Unicode 文字列。	[1]

1 JDBC SQL データ型は定義できません。データが取得されると、ネイティブデータ型が同様のデータ型にマッピングされます。
2 CHAR(n)または VARCHAR(n)データ型を使用してマルチバイトデータを保存する場合は、CLIENT_ENCODING=接続オプションでエンコーディングを指定する必要があります。または、エンコーディングを設定する必要がないように、かわりにマルチバイトデータに NCHAR または NVARCHAR データ型を使用します。
3 FedSQL では、DECIMAL データ型は、データソースに完全に渡すことができる要求でサポートされています。データソースに完全に渡すことができない要求の DECIMAL 列は、DOUBLE として処理されるため、精度に影響を与える可能性があります。詳細については、*"データ型" (SAS FedSQL 言語リファレンス)*を参照してください。

MDS のデータ型

次の表に、インメモリデータベースのデータ型サポートを示します。

注: このデータソースは CAS サーバーではサポートされていません。

表 A20.12 FedSQL データ型と MDS テーブルで使用されるデータ型のマッピング

データ型定義キーワード [2]	MDS データ型	説明	返されるデータ型
BIGINT	BIGINT	64 ビットの符号付き整数。	BIGINT
BINARY(*n*)	BINARY(*n*)	固定長のバイナリ文字列。	BINARY(*n*)
CHAR(*n*)	CHAR(*n*)	固定長の文字列。	CHAR(*n*)
DATE	DATE	日付値。	DATE
DECIMAL|NUMERIC(*p*,*s*)	NUMERIC(*p*,*s*)	精度とスケールが固定された正確な整数。	DECIMAL/NUMERIC(*p*,*s*)[3]
DOUBLE	DOUBLE	8 バイトの IEEE 浮動小数点値。 **注:** ANSI SQL null 値をサポートします。	DOUBLE

データ型定義キーワード [2]	MDS データ型	説明	返されるデータ型
FLOAT	DOUBLE	8 バイトの IEEE 浮動小数点値。 **注:** ANSI SQL null 値をサポートします。	DOUBLE
INTEGER	INTEGER	32 ビットの符号付き整数。	INTEGER
NCHAR(*n*)	NCHAR(*n*)	固定長の Unicode 文字列。	NCHAR(*n*)
NVARCHAR(*n*)	NVARCHAR(*n*)	可変長の Unicode 文字列。	NVARCHAR(*n*)
REAL	DOUBLE	8 バイトの IEEE 浮動小数点値。 **注:** ANSI SQL null 値をサポートします。	DOUBLE
SMALLINT	INTEGER	32 ビットの符号付き整数。	INTEGER
TIME(*p*)	TIME(*p*)	時間値。	TIME(*p*)
TIMESTAMP(*p*)	TIMESTAMP(*p*)	日時値。	TIMESTAMP(*p*)
TINYINT	INTEGER	32 ビットの符号付き整数。	INTEGER
[1]	UBIGINT	64 ビットの符号なし整数。	BIGINT
[1]	UINTEGER	32 ビットの符号なし整数。	INTEGER
VARCHAR(*n*)	VARCHAR(n)	可変長の文字列。	VARCHAR(*n*)
VARBINARY(n)	VARBINARY(n)	可変長のバイナリ文字列。	VARBINARY(n)

1 MDS SQL データ型は定義できません。データが取得されると、ネイティブデータ型が同様のデータ型にマッピングされます。

2 データ型のマッピング方法を制御する CT_PRESERVE=接続引数は、データ型を定義できるかどうかに影響する可能性があります。値 FORCE (デフォルト)および FORCE_COL_SIZE は、データ型を定義できるかどうかには影響しません。要求されたデータ型がデータソースにネイティブでない場合、または指定された精度またはスケールがデータソースの範囲内にない場合、値 STRICT および SAFE はエラーになる可能性があります。

3 FedSQL では、DECIMAL データ型は、データソースに完全に渡すことができる要求でサポートされています。データソースに完全に渡すことができない要求の DECIMAL 列は、DOUBLE として処理されるため、精度に影響を与える可能性があります。詳細については、*"データ型" (SAS FedSQL 言語リファレンス)*を参照してください。

Microsoft SQL Server のデータ型

次の表に、Microsoft SQL Server データベースのデータ型サポートを示します。

Microsoft SQL Server データ型に関するデータソース固有の情報については、Microsoft SQL Server データベースのドキュメントを参照してください。

注: この表の情報は、CAS で処理されるデータには適用されません。CAS にロードされたデータは、CAS データ型に変換されます。CAS データ型の変換については、*SAS Cloud Analytic Services: ユーザーガイド*の SAS Data Connector for Microsoft SQL Server のドキュメントを参照してください。

表 A20.13　FedSQL データ型と Microsoft SQL Server で使用されるデータ型のマッピング

データ型定義キーワード [1]	SQL Server SQL 識別子	説明	返されるデータ型
BIGINT	BIGINT	大きな符号付きの正確な整数。	BIGINT
BINARY(*n*)	BINARY(*n*)	固定長のバイナリ文字列。	BINARY(*n*)
[2]	BIT	シングルビットのバイナリ値。	[2]
CHAR(*n*)	CHAR(*n*)	固定長の文字列。	CHAR(*n*)
DATE	DATE	日付値。	DATE
[2]	DATETIME	24 時間表記に基づく秒の端数付き日時値。	TIMESTAMP(*p*)
[2]	DATETIME2(*p*)	より大きな日付範囲、より大きなデフォルトの小数精度、およびオプションのユーザー指定の精度を持つ拡張 DATETIME 値。	TIMESTAMP(*p*)
[2]	DATETIMEOFFSET(*p*)	タイムゾーンを認識した日時値。	[2]

データ型定義キーワード [1]	SQL Server SQL 識別子	説明	返されるデータ型
DECIMAL|NUMERIC(*p*,*s*)	DECIMAL|NUMERIC(*p*,*s*)	固定精度とスケールの符号付き数値。	DECIMAL|NUMERIC(*p*,*s*)[3]
DOUBLE	DOUBLE	符号付き倍精度浮動小数点数。	DOUBLE
FLOAT	FLOAT	符号付き倍精度浮動小数点数。	DOUBLE
[2]	IMAGE	可変長のバイナリ値。	[2]
INTEGER	INTEGER	通常の符号付きの正確な整数。	INTEGER
[2]	MONEY	8 バイトの通貨値。	DECIMAL(19,4)[3]
NCHAR(*n*)	NCHAR	固定長の Unicode 文字列。	NCHAR(*n*)
NVARCHAR(*n*)	NVARCHAR	可変長の Unicode 文字列。	NVARCHAR(*n*)
REAL	REAL	符号付き単精度浮動小数点数。	REAL
[2]	SMALLDATETIME	秒の端数のない日時値。	TIMESTAMP(*p*)
SMALLINT	SMALLINT	小さな符号付きの正確な整数。	SMALLINT
[2]	SMALLMONEY	4 バイトの通貨値。	DECIMAL(10,4)[3]
TIME(*p*)	TIME(*p*)	時間値。	TIME(*p*)
TIMESTAMP(*p*)	BINARY(8)	日時値。	TIMESTAMP(*p*)
TINYINT	TINYINT	非常に小さな符号付きの正確な整数。	TINYINT
[2]	UNIQUEIDENTIFIER	グローバル一意識別子。	CHAR(36)
VARBINARY(*n*)	VARBINARY	可変長のバイナリ文字列。	VARBINARY(*n*)
[2]	VARBINARY(MAX)	可変長のバイナリ文字列。最大 2GB。	[2]

データ型定義キーワード [1]	SQL Server SQL 識別子	説明	返されるデータ型
VARCHAR(*n*)	VARCHAR(*n*)、TEXT	可変長の文字列。[2]	VARCHAR(*n*)
[2]	VARCHAR(MAX)	可変長の Unicode 文字列。最大 2GB。	[2]

1 データ型のマッピング方法を制御する CT_PRESERVE=接続引数は、データ型を定義できるかどうかに影響する可能性があります。値 FORCE (デフォルト)および FORCE_COL_SIZE は、データ型を定義できるかどうかには影響しません。要求されたデータ型がデータソースにネイティブでない場合、または指定された精度またはスケールがデータソースの範囲内にない場合、値 STRICT および SAFE はエラーになる可能性があります。

2 Microsoft SQL Server データ型は定義できません。データが取得されると、ネイティブデータ型が同様のデータ型にマッピングされます。

3 FedSQL では、DECIMAL データ型は、データソースに完全に渡すことができる要求でサポートされています。データソースに完全に渡すことができない要求の DECIMAL 列は、DOUBLE として処理されるため、精度に影響を与える可能性があります。詳細については、*"データ型" (SAS FedSQL 言語リファレンス)*を参照してください。

MongoDB のデータ型

次の表に、MongoDB 非リレーショナルデータベースのデータ型サポートを示します。

MongoDB の Timestamp 型、Min key 型、Max key 型は MongoDB の内部型です。したがって、SAS によって読み取られることはありません。

FedSQL の BINARY、DECIMAL、FLOAT、NCHAR、NVARCHAR、REAL、SMALLINT、TINYINT、VARBINARY データ型は、データ定義ではサポートされていません。

MongoDB データ型に関するデータソース固有の情報については、MongoDB のドキュメントを参照してください。

注: この表の情報は、CAS で処理されるデータには適用されません。CAS にロードされたデータは、CAS データ型に変換されます。CAS データ型の変換については、*SAS Cloud Analytic Services: ユーザーガイド*の SAS Data Connector for MongoDB のドキュメントを参照してください。

表 A20.14 FedSQL データ型と MongoDB で使用されるデータ型のマッピング

データ型定義キーワード	MongoDB データ型	説明	返されるデータ型
[1]	Array	配列。	VARCHAR(n)
BIGINT	int	通常の 64 ビットの正確な整数。	BIGINT

データ型定義キーワード	MongoDB データ型	説明	返されるデータ型
[2]	BinData	バイナリ値。	VARCHAR[3]
[2]	Bool	ブール値。	INTEGER
CHAR(*n*)	String	固定長の UTF-8 文字列。	CHAR(*n*)
DATE	Date	日時値。	DATE
[2]	Decimal	Decimal128 値。	DOUBLE
DOUBLE	Double	符号付き倍精度浮動小数点数。	DOUBLE
INTEGER	Int	通常の 32 ビットの正確な整数。	INTEGER
[2]	Javascript	Javascript 文字列。	[1]
[2]	Long	大きな符号付きの正確な整数。	BIGINT
[2]	Null	null 値。	VARCHAR
[1]	Object	JSON オブジェクト。	VARCHAR(n)
[2]	ObjectId[4]	オブジェクト識別子。	VARCHAR
[2]	Regex	正規表現文字列。	VARCHAR
TIME	Date	時間値。	TIME
TIMESTAMP	Date	日時値。	TIMESTAMP
VARCHAR	String	可変長の UTF-8 文字列。	VARCHAR、CHAR(*n*)[5]

1 配列とオブジェクトの型は、別々の子テーブルとしてモデル化されます。詳細については、*SAS/ACCESS for Nonrelational Databases: リファレンス*の MongoDB のドキュメントを参照してください。Javascript 型は VARCHAR にマッピングされます。

2 MongoDB データ型は定義できません。データが取得されると、ネイティブデータ型が同様のデータ型にマッピングされます。

3 バイナリデータの 16 進文字列表現が含まれます。

4 MongoDB テーブルで ObjectId フィールドを定義または複製することはできません。ただし、MongoDB コレクションにすでに存在する ObjectId フィールドに値を挿入および更新できます。

5 CHAR(n)を使用するようにデフォルトの MongoDB スキーマをカスタマイズできます。詳細については、*SAS/ACCESS for Nonrelational Databases: リファレンス*の MongoDB のドキュメントを参照してください。

MySQL のデータ型

次の表に、MySQL データベースのデータ型サポートを示します。

NCHAR、NVARCHAR、REAL、VARBINARY データ型は、データ型定義ではサポートされていません。

MySQL データ型に関するデータソース固有の情報については、MySQL データベースのドキュメントを参照してください。

注: この表の情報は、CAS で処理されるデータには適用されません。CAS にロードされたデータは、CAS データ型に変換されます。CAS データ型の変換については、*SAS Cloud Analytic Services: ユーザーガイド*の SAS Data Connector for MySQL のドキュメントを参照してください。

表 A20.15 FedSQL データ型と MySQL で使用されるデータ型のマッピング

データ型定義キーワード [1]	MySQL データ型	説明	返されるデータ型
BIGINT	BIGINT	大きな符号付きの正確な整数。	BIGINT
BINARY(*n*)	BINARY(*n*)	固定長のバイナリ文字列。	BINARY(*n*)
[2]	BLOB	バイナリラージオブジェクト文字列。	[2]
CHAR(*n*)	CHAR(*n*)[3]	固定長の文字列。	CHAR(*n*)
DATE	DATE	日付値。	DATE
[2]	DATETIME	日時値。	[2]
DECIMAL|NUMERIC(*p*,*s*)	DECIMAL(*p*,*s*)	符号付き固定小数点 10 進数。	DECIMAL(*p*,*s*)[4]
DOUBLE	DOUBLE	符号付き倍精度浮動小数点数。	DOUBLE
[2]	ENUM(*values*)	許可された値のリストからの文字値。	[2]
FLOAT	FLOAT(*p*)	符号付き倍精度浮動小数点数。	DOUBLE
INTEGER	INT	通常の符号付きの正確な整数。	INTEGER
[2]	JSON	JSON 形式の可変長文字列。	VARCHAR

データ型定義キーワード [1]	MySQL データ型	説明	返されるデータ型
[2]	LONGBLOB	可変長のバイナリラージオブジェクト文字列。	[2]
[2]	LONGTEXT	可変長のテキスト文字列。	[2]
[2]	MEDIUMBLOB	可変長のバイナリラージオブジェクト文字列。	[2]
[2]	MEDIUMINT	通常の符号付きの正確な整数。	[2]
[2]	MEDIUMTEXT	可変長の文字列。	[2]
[2]	SET(*values*)	許可された値のリストからの文字値。	[2]
SMALLINT	SMALLINT	小さな符号付きの正確な整数。	SMALLINT
[2]	TEXT	テキスト文字列。	[2]
TIME(*p*)	TIME(*p*)	時間値。	TIME(*p*)
TIMESTAMP(*p*)	DATETIME(*p*)	日時値。	TIMESTAMP(*p*)
[2]	TINYBLOB	可変長のバイナリラージオブジェクト文字列。	[2]
TINYINT	TINYINT	非常に小さな符号付きの正確な整数。	TINYINT
[2]	TINYTEXT	可変長のテキスト文字列。	[2]
VARCHAR(*n*)	VARCHAR(*n*)	可変長の文字列。	VARCHAR(*n*)

1 データ型のマッピング方法を制御する CT_PRESERVE=接続引数は、データ型を定義できるかどうかに影響する可能性があります。値 FORCE (デフォルト)および FORCE_COL_SIZE は、データ型を定義できるかどうかには影響しません。要求されたデータ型がデータソースにネイティブでない場合、または指定された精度またはスケールがデータソースの範囲内にない場合、値 STRICT および SAFE はエラーになる可能性があります。

2 MySQL データ型は定義できません。データが取得されると、ネイティブデータ型が同様のデータ型にマッピングされます。

3 DS2 は、固定文字値の空白スペースを 処理するために ANSI SQL 規則を使用します。MySQL 8 は、デフォルトで末尾の空白スペースを 重要な値として扱います。この違いは、DS2 で FedSQL ステートメントを使用するときに実行される文字比較に干渉します。DS2 を使用して MySQL 8 CHAR()データに対して FedSQL ステートメントを発行する場合、データが正しく読み取られるように、FedSQL ステートメントで TRIM 関数を使用する必要があります。DS2 で FedSQL ステートメントがサポートされている場所については、“DS2 と FedSQL の使用” (*SAS DS2 プログラマガイド*)を参照してください。また、“例: MySQL 8 CHAR()データから末尾のスペースをフィルタリングする” (2178 ページ)も参照してください。MySQL 8 は、比較のデフォルト照合順序として utf8mb4_0900_ai_ci を使用します。

4 FedSQL では、DECIMAL データ型は、データソースに完全に渡すことができる要求でサポートされています。データソースに完全に渡すことができない要求の DECIMAL 列は、DOUBLE として処理されるため、精度に影響を与える可能性があります。詳細については、“データ型” (*SAS FedSQL 言語リファレンス*)を参照してください。

Netezza のデータ型

次の表に、Netezza データベースのデータ型サポートを示します。

BINARY および VARBINARY データ型は、データ型定義ではサポートされていません。

Netezza データ型に関するデータソース固有の情報については、Netezza データベースのドキュメントを参照してください。

注: この表の情報は、CAS で処理されるデータには適用されません。CAS にロードされたデータは、CAS データ型に変換されます。CAS データ型の変換については、*SAS Cloud Analytic Services: ユーザーガイド*の SAS Data Connector for Netezza のドキュメントを参照してください。

表 A20.16 FedSQL データ型と Netezza で使用されるデータ型のマッピング

データ型定義キーワード [1]	Netezza データ型	説明	返されるデータ型
BIGINT	BIGINT	大きな符号付きの正確な整数。	BIGINT
CHAR(*n*)	CHAR(*n*)	固定長の文字列データ。	CHAR(*n*)
DATE	DATE	日付値。	DATE
DECIMAL|NUMERIC(*p*,*s*)	DECIMAL(*p*,*s*)	固定小数点 10 進数。	DECIMAL(*p*,*s*)[3]
DOUBLE	DOUBLE	浮動小数点数。	DOUBLE
FLOAT	FLOAT(*p*)	64 ビットの倍精度浮動小数点数。	DOUBLE
INTEGER	INTEGER	大きな整数。	INTEGER
NCHAR(*n*)	NCHAR(*n*)	固定長の Unicode 文字列。	NCHAR(*n*)
NVARCHAR(*n*)	NVARCHAR(*n*)	可変長の Unicode 文字列。	NVARCHAR(*n*)
REAL	REAL	浮動小数点数。	REAL
SMALLINT	SMALLINT	小さな整数。	SMALLINT
TIME(*p*)	TIME(*p*)	時間値。	TIME(*p*)[2]
TIMESTAMP(*p*)	TIMESTAMP(*p*)	日時値。	TIMESTAMP(*p*)

データ型定義キーワード [1]	Netezza データ型	説明	返されるデータ型
TINYINT	BYTEINT	極小整数。	TINYINT
VARCHAR(*n*)	VARCHAR(*n*)	可変長の文字列データ。	VARCHAR(*n*)

1 データ型のマッピング方法を制御する CT_PRESERVE=接続引数は、データ型を定義できるかどうかに影響する可能性があります。値 FORCE (デフォルト)および FORCE_COL_SIZE は、データ型を定義できるかどうかには影響しません。要求されたデータ型がデータソースにネイティブでない場合、または指定された精度またはスケールがデータソースの範囲内にない場合、値 STRICT および SAFE はエラーになる可能性があります。

2 Netezza サーバーとの通信に使用される ODBC スタイルのインターフェイスにより、サーバーからクライアントへのデータ転送で秒の端数が失われます。

3 FedSQL では、DECIMAL データ型は、データソースに完全に渡すことができる要求でサポートされています。データソースに完全に渡すことができない要求の DECIMAL 列は、DOUBLE として処理されるため、精度に影響を与える可能性があります。詳細については、*"データ型" (SAS FedSQL 言語リファレンス)*を参照してください。

ODBC のデータ型

次の表に、ODBC に準拠するデータソースのデータ型サポートを示します。

ODBC SQL データ型に関するデータソース固有の情報については、特定の ODBC データソースのドキュメントを参照してください。

注: この表の情報は、CAS で処理されるデータには適用されません。CAS にロードされたデータは、CAS データ型に変換されます。CAS データ型の変換については、*SAS Cloud Analytic Services: ユーザーガイド*の SAS Data Connector for ODBC のドキュメントを参照してください。

表 A20.17 FedSQL データ型と ODBC でサポートされるデータ型のマッピング

データ型定義キーワード [1]	ODBC SQL 識別子	説明	返されるデータ型
BIGINT	SQL_BIGINT	大きな符号付きの正確な整数。	BIGINT
BINARY(*n*)	SQL_BINARY	固定長のバイナリ文字列。	BINARY(*n*)
[2]	SQL_BIT	シングルビットのバイナリ値。	[2]
CHAR(*n*)[3]	SQL_CHAR	固定長の文字列。	CHAR(*n*)
DATE	SQL_TYPE_DATE	日付値。	DATE

データ型定義キーワード [1]	ODBC SQL 識別子	説明	返されるデータ型
DECIMAL| NUMERIC(*p*,*s*)	SQL_DECIMAL| SQL_NUMERIC	符号付き固定小数点 10 進数。	DECIMAL| NUMERIC(*p*,*s*)[4]
DOUBLE	SQL_DOUBLE	符号付き倍精度浮動小数点数。	DOUBLE
FLOAT	SQL_FLOAT	符号付き倍精度浮動小数点数。	DOUBLE
[2]	SQL_GUID	グローバル一意識別子。	[2]
INTEGER	SQL_INTEGER	通常の符号付きの正確な整数。	INTEGER
[2]	SQL_INTERVAL	2 つの年、月、日、日付、時間の間隔。	[2]
[2]	SQL_LONGVARBINARY	可変長のバイナリ文字列。	[2]
[2]	SQL_LONGVARCHAR	可変長の Unicode 文字列。	[2]
NCHAR(*n*)	SQL_WCHAR	固定長の Unicode 文字列。	NCHAR(*n*)
NVARCHAR(*n*)	SQL_WVARCHAR	可変長の Unicode 文字列。	NVARCHAR(*n*)
REAL	SQL_REAL	符号付き単精度浮動小数点数。	REAL
SMALLINT	SQL_SMALLINT	小さな符号付きの正確な整数。	SMALLINT
TIME(*p*)	SQL_TYPE_TIME	時間値。	TIME(*p*)
TIMESTAMP(*p*)	SQL_TYPE_TIMESTAMP	日時値。	TIMESTAMP(*p*)
TINYINT	SQL_TINYINT	非常に小さな符号付きの正確な整数。	TINYINT
VARBINARY(*n*)	SQL_VARBINARY	可変長のバイナリ文字列。	VARBINARY(*n*)
VARCHAR(*n*)[3]	SQL_VARCHAR	可変長の文字列。	VARCHAR(*n*)

データ型定義キーワード [1]	ODBC SQL 識別子	説明	返されるデータ型
[2]	SQL_WLONGVARCHAR	可変長の Unicode 文字列。	[2]

1 データ型のマッピング方法を制御する CT_PRESERVE=接続引数は、データ型を定義できるかどうかに影響する可能性があります。値 FORCE (デフォルト)および FORCE_COL_SIZE は、データ型を定義できるかどうかには影響しません。要求されたデータ型がデータソースにネイティブでない場合、または指定された精度またはスケールがデータソースの範囲内にない場合、値 STRICT および SAFE はエラーになる可能性があります。

2 ODBC SQL データ型は定義できません。データが取得されると、ネイティブデータ型が同様のデータ型にマッピングされます。

3 CHAR(*n*)または VARCHAR(*n*)データ型を使用してマルチバイトデータを DB2、Greenplum、または Oracle データベースに保存する場合は、CLIENT_ENCODING=接続オプションでエンコーディングを指定する必要があります。または、Oracle の場合のみ、エンコーディングを設定する必要がないように、かわりにマルチバイトデータに NCHAR または NVARCHAR データ型を使用します。

4 FedSQL では、DECIMAL データ型は、データソースに完全に渡すことができる要求でサポートされています。データソースに完全に渡すことができない要求の DECIMAL 列は、DOUBLE として処理されるため、精度に影響を与える可能性があります。詳細については、*"データ型" (SAS FedSQL 言語リファレンス)*を参照してください。

Oracle のデータ型

次の表に、Oracle データベースのデータ型サポートを示します。

Oracle データ型に関するデータソース固有の情報については、Oracle データベースのドキュメントを参照してください。

注: この表の情報は、CAS で処理されるデータには適用されません。CAS にロードされたデータは、CAS データ型に変換されます。CAS データ型の変換については、*SAS Cloud Analytic Services: ユーザーガイド*の SAS Data Connector for Oracle のドキュメントを参照してください。

表 A20.18 FedSQL データ型と Oracle で使用されるデータ型のマッピング

データ型定義キーワード [1]	Oracle データ型	説明	返されるデータ型
BIGINT	NUMBER	大きな符号付きの正確な整数。	BIGINT
BINARY(*n*)	RAW(*n*)	固定長または可変長のバイナリ文字列。	BINARY(*n*)
CHAR(*n*)	CHAR(*n*)	固定長の文字列。	CHAR(*n*)
DATE	DATE	日付値。	TIMESTAMP(*p*)[3]
DECIMAL|NUMERIC(*p*,*s*)	NUMBER(*p*,*s*)	符号付き固定小数点 10 進数。	DOUBLE[4]

データ型定義キーワード [1]	Oracle データ型	説明	返されるデータ型
DOUBLE	BINARY_DOUBLE	符号付き倍精度浮動小数点数。	DOUBLE
FLOAT	FLOAT(*p*)	符号付き倍精度浮動小数点数。	DOUBLE
INTEGER	NUMBER(*p*,*s*)	通常の符号付きの正確な整数。	INTEGER
[2]	LONG	可変長の文字列。	[2]
NCHAR(*n*)	NCHAR(*n*)	固定長の Unicode 文字列。	NCHAR(*n*)
NVARCHAR(*n*)	NVARCHAR(*n*)	可変長の Unicode 文字列。	NVARCHAR(*n*)
REAL	FLOAT	符号付き単精度浮動小数点数。	REAL
SMALLINT	NUMBER	小さな符号付きの正確な整数。	SMALLINT
TIME(*p*)	TIME(*p*)	時間値。	TIMESTAMP(*p*)[3]
TIMESTAMP(*p*)	TIMESTAMP(*p*)	日時値。	TIMESTAMP(*p*)
TINYINT	NUMBER	非常に小さな符号付きの正確な整数。	TINYINT
VARBINARY(*n*)	RAW(*n*)	可変長のバイナリ文字列。	VARBINARY(*n*)
VARCHAR(*n*)	VARCHAR2(*n*)[5]	可変長の文字列。	VARCHAR(*n*)

1 データ型のマッピング方法を制御する CT_PRESERVE=接続引数は、データ型を定義できるかどうかに影響する可能性があります。値 FORCE (デフォルト)および FORCE_COL_SIZE は、データ型を定義できるかどうかには影響しません。要求されたデータ型がデータソースにネイティブでない場合、または指定された精度またはスケールがデータソースの範囲内にない場合、値 STRICT および SAFE はエラーになる可能性があります。

2 Oracle データ型は定義できません。データが取得されると、ネイティブデータ型が同様のデータ型にマッピングされます。

3 DATE および TIME データ型によって返されるタイムスタンプは、PUT 関数で DATEPART 関数を使用することにより、日時値に変更できます。

4 ORNUMERIC=接続引数およびテーブルオプションは、Oracle NUMBER 列から読み取られた数値または Oracle NUMBER 列に挿入された数値の処理方法を決定します。デフォルトである ORNUMERIC=YES は、明示的な精度を持つ非整数値が NUMERIC 値として扱われることを示します。BULKLOAD=YES の場合、ORNUMERIC のデフォルト値は NO です。

5 Oracle バージョンが 12c で、Oracle MAX_STRING_SIZE=パラメーターが EXTENDED に設定されている場合、VARCHAR2(*n*)型は最大 32,767 バイトでサポートされます。

PostgreSQL のデータ型

次の表に、PostgreSQL データベースのデータ型サポートを示します。

BINARY、NCHAR、NVARCHAR、TINYINT、VARBINARY データ型は、データ型定義ではサポートされていません。

PostgreSQL データ型に関するデータソース固有の情報については、PostgreSQL データベースのドキュメントを参照してください。

注: この表の情報は、CAS で処理されるデータには適用されません。CAS にロードされたデータは、CAS データ型に変換されます。CAS データ型の変換については、*SAS Cloud Analytic Services: ユーザーガイド*の SAS Data Connector for PostgreSQL のドキュメントを参照してください。

表 A20.19 FedSQL データ型と PostgreSQL で使用されるデータ型のマッピング

データ型定義キーワード [1]	PostgreSQL データ型	説明	返されるデータ型
BIGINT	BIGINT	大きな符号付きの正確な整数または符号付き 8 バイト整数。	BIGINT
[2]	BIGSERIAL	自動増分 8 バイト整数。	[2]
[2]	BIT(*n*)	固定長のビット文字列。	[2]
[2]	BIT VARYING(*n*)	可変長のビット文字列。	[2]
[2]	BOOLEAN	論理ブール(true/false)値。	[2]
[2]	BOX	平面上の長方形のボックス。	[2]
[2]	BYTEA	バイト配列。	[2]
CHAR(*n*)	CHAR(*n*)	固定長の文字列。	CHAR(*n*)
[2]	CIDR	IPv4 または IPv6 ネットワークアドレス。	[2]
[2]	CIRCLE	平面上の円。	[2]
DATE	DATE	日付値。	DATE
DECIMAL|NUMERIC(*p*,*s*)	NUMERIC(*p*,*s*)	符号付き固定小数点 10 進数。	DECIMAL|NUMERIC(*p*,*s*)[3]
DOUBLE	DOUBLE PRECISION	符号付き倍精度浮動小数点数。	DOUBLE
FLOAT	FLOAT(*p*)	符号付き倍精度浮動小数点数。	DOUBLE
[2]	INET	IPv4 または IPv6 ホストアドレス。	[2]
INTEGER	INTEGER	通常の符号付きの正確な整数。	INTEGER

データ型定義キーワード [1]	PostgreSQL データ型	説明	返されるデータ型
2	INTERVAL	期間。	2
2	LINE	平面上の無限の線。	2
2	LSEG	平面上の線分。	2
2	MACADDR	メディアアクセスコントロールアドレス。	2
2	MONEY	通貨額。	2
2	PATH	平面上の幾何学的パス。	2
2	POINT	平面上の幾何学的な点。	2
2	POLYGON	平面上の閉じた幾何学的パス。	2
REAL	REAL	符号付き単精度浮動小数点数。	REAL
2	SERIAL	自動増分 4 バイト整数。	2
SMALLINT	SMALLINT	小さな符号付きの正確な整数。	SMALLINT
2	SMALL SERIAL	自動増分 2 バイト整数。	2
2	TEXT	可変長の文字列。	2
TIME(*p*)	TIME(*p*)	時間値。	TIME(*p*)
TIMESTAMP(*p*)	TIMESTAMP(*p*)	日時値。	TIMESTAMP(*p*)
2	TSQUERY	テキスト検索クエリ。	2
2	TSVECTOR	テキスト検索ドキュメント。	2
2	TXID_SNAPSHOT	ユーザーレベルのトランザクション ID スナップショット。	2
2	UUID	ユニバーサル一意識別子。	2
VARCHAR(*n*)	CHARACTER VARYING(*n*)	可変長の文字列。	VARCHAR(*n*)
2	XML	XML データ。	2

1 データ型のマッピング方法を制御する CT_PRESERVE=接続引数は、データ型を定義できるかどうかに影響する可能性があります。値 FORCE (デフォルト)および FORCE_COL_SIZE は、データ型を定義できるかどうかには影響しません。要求され

たデータ型がデータソースにネイティブでない場合、または指定された精度またはスケールがデータソースの範囲内にない場合、値 STRICT および SAFE はエラーになる可能性があります。

2 PostgreSQL データ型は定義できません。データが取得されると、ネイティブデータ型が同様のデータ型にマッピングされます。

3 FedSQL では、DECIMAL データ型は、データソースに完全に渡すことができる要求でサポートされています。データソースに完全に渡すことができない要求の DECIMAL 列は、DOUBLE として処理されるため、精度に影響を与える可能性があります。詳細については、*"データ型" (SAS FedSQL 言語リファレンス)*を参照してください。

Salesforce から読み取るためのデータ型

次の表に、Salesforce から読み取るためのデータ型のサポートを示します。SAS/ACCESS ソフトウェアは、Salesforce SOAP API を使用して Salesforce から読み取ります。Salesforce SOAP API は、プリミティブ型を返します。

Salesforce の計算型と複合型はサポートされていません。複合型の例は、住所と位置です。

詳細については、Salesforce SOAP API のドキュメントを参照してください。

注: この表の情報は、CAS で処理されるデータには適用されません。CAS にロードされたデータは、CAS データ型に変換されます。CAS データ型の変換については、*SAS Cloud Analytic Services: ユーザーガイド*の SAS Data Connector for Salesforce のドキュメントを参照してください。

表 A20.20 Salesforce から読み取るときのデータ型マッピング

Salesforce フィールドまたはプリミティブ型	説明	FedSQL で返されるデータ型
anyType	コンテキストに基づいてインターフェイス全体で異なる方法で使用できる多相データ型。	VARCHAR(*n*)
base64	attachment レコード、document レコード、および scontrol レコードを含む base-64 バイナリデータ。Body/Binary フィールドにはデータが含まれています。BodyLength フィールドは、データの長さを定義します。	VARCHAR(*n*)[1]
boolean	ブール値。	TINYINT
byte	ビットのセット。	TINYINT
combobox	ユーザーがリストにない値を定義できるようにする列挙値のセット。	VARCHAR(*n*)
currency	通貨値。	DOUBLE

Salesforce フィールドまたはプリミティブ型	説明	FedSQL で返されるデータ型
DataCategoryGroupReference	Article オブジェクトと Question オブジェクトのデータカテゴリグループまたはカテゴリの一意の名前への参照。	VARCHAR(*n*)
date	日付値。	DATE[2]
dateTime	タイムスタンプ。	TIMESTAMP[2]
double	符号付き倍精度浮動小数点数。	DOUBLE[3]
email	メールアドレス。	VARCHAR(*n*)
encryptedstring	暗号化されたテキスト文字列。文字列の最大長は 175 文字です。	VARCHAR(*n*)
ID	オブジェクトの主キーフィールド。	VARCHAR(*n*)
int	サイズがフィールド設定によって異なる、通常の符号付きの正確な整数。	BIGINT
JunctionIdList	基になるジャンクションエンティティの 多対多の関係を表す、参照される ID 値の文字列配列。	VARCHAR(*n*)
masterrecord	マージされ、元のレコードが削除されるレコードの識別子。	VARCHAR(*n*)
multipicklist	複数の値を選択できる列挙値のセット。選択された値は、セミコロンで区切られた値の文字列として返されます。	VARCHAR(*n*)
percent	パーセンテージ値。	DOUBLE
phone	電話番号。	CHAR(40)
picklist	1 つの値を選択できる列挙値のセット。	VARCHAR(*n*)
reference	別のオブジェクトの相互参照。SQL の外部キーに類似しています。	VARCHAR(*n*)
String	フィールド設定によって最大サイズが異なるテキスト文字列。	CHAR(*n*)
textarea	複数行のテキストフィールドとして表示される文字列。	VARCHAR(*n*)
Time	時間値。	TIME[2]

Salesforce フィールドまたはプリミティブ型	説明	FedSQL で返されるデータ型
url	URL (Uniform Resource Locator)値。	VARCHAR(*n*)

1 base64 フィールドは、VARCHAR 型として base64 形式のままです。
2 最も早い有効な日付は、1700-01-01T00:00:00Z GMT、または 1700 年 1 月 1 日の午前 0 時直後です。最も遅い有効な日付は 4000-12-31T00:00:00Z GMT、または 4000 年 12 月 31 日の午前 0 時直後です。これらの値は、タイムゾーンによってオフセットされます。たとえば、太平洋タイムゾーンでは、最も早い有効な日付は 1699-12-31T16:00:00、または 1699 年 12 月 31 日の午後 4 時です。
3 “Salesforce に書き込むためのデータ型” (2075 ページ)で説明されている数値型の精度とスケールに一致する double 型は、その表に示すようにマッピングされます。たとえば、精度が 10 でスケールが 0 の double 列は、INTEGER にマッピングされます。

Salesforce に書き込むためのデータ型

次の表に、Salesforce に書き込むためのデータ型のサポートを示します。

SAS は、Salesforce の Winter'19 リリース以降をサポートしています。

BINARY、FLOAT、および VARBINARY データ型は、データ定義ではサポートされていません。

詳細については、Salesforce メタデータ API のドキュメントを参照してください。

注: この表の情報は、CAS で処理されるデータには適用されません。CAS にロードされたデータは、CAS データ型に変換されます。CAS データ型の変換については、*SAS Cloud Analytic Services: ユーザーガイド*の SAS Data Connector for Salesforce のドキュメントを参照してください。

表 A20.21 Salesforce に書き込むときのデータ型マッピング

データ型定義キーワード	Salesforce カスタムフィールド型	説明	AS 式を使用して CREATE TABLE で返されるデータ型
BIGINT	Number	精度 18、スケール 0 の数値。	BIGINT
CHAR(*n*)[1]	Text	255 文字以下の文字列。	CHAR(*n*)
DATE	Date	日付値。[2]	DATE
DECIMAL|NUMERIC(*p*,*s*)	Number	精度とスケールのある数値。	DOUBLE
DOUBLE	Number	精度 18、スケール 4 の数値。	DOUBLE

データ型定義キーワード	Salesforce カスタムフィールド型	説明	AS 式を使用して CREATE TABLE で返されるデータ型
INTEGER	Number	精度 10、スケール 0 の数値。	INTEGER
NCHAR(*n*)	Text	固定長の Unicode 文字列。	CHAR(*n*)
NVARCHAR(*n*)	Text	可変長の Unicode 文字列。	VARCHAR(*n*)
REAL	Number	精度 18、スケール 4 の数値。	DOUBLE
SMALLINT	Number	精度 5、スケール 0 の数値。	SMALLINT
TIME	Time	時間値。[2]	TIME
TIMESTAMP	Datetime	タイムスタンプ。[2]	TIMESTAMP
TINYINT	Number	精度 3、スケール 0 の数値。	TINYINT
VARCHAR(*n*)[3]	LongTextArea	255 文字より大きく 32K 文字未満の文字列。	VARCHAR(*n*)

1 *n* が 255 文字を超える CHAR(*n*)値は、VARCHAR(*n*)として作成されます。

2 最も早い有効な日付は、1700-01-01T00:00:00Z GMT、または 1700 年 1 月 1 日の午前 0 時直後です。最も遅い有効な日付は 4000-12-31T00:00:00Z GMT、または 4000 年 12 月 31 日の午前 0 時直後です。これらの値は、タイムゾーンによってオフセットされます。たとえば、太平洋タイムゾーンでは、最も早い有効な日付は 1699-12-31T16:00:00、または 1699 年 12 月 31 日の午後 4 時です。

3 *n* が 255 文字以下の VARCHAR(*n*)値は、CHAR(*n*)として作成されます。

SAP のデータ型

次の表に、SAP システムのデータ型サポートを示します。

SAP システムの場合、列定義でサポートされているデータ型はありません。ネイティブ ABAP SAP データ型は、データ取得のみを目的として同様のデータ型にマッピングされます。

ABAP SAP データ型に関するデータソース固有の情報については、SAP システムデータベースのドキュメントを参照してください。

注: このデータソースは CAS サーバーではサポートされていません。

表 A20.22　SAP データ型と FedSQL データ型のマッピング

ABAP SAP データ型	説明	データ取得に使用される FedSQL データ型
ACCP	転記期間。	非 Unicode SAP システムの場合は CHAR(*n*)、Unicode SAP システムの場合は NCHAR(*n*)
CHAR	固定長の文字列。	非 Unicode SAP システムの場合は CHAR(*n*)、Unicode SAP システムの場合は NCHAR(*n*)
CLNT	クライアントフィールド。	非 Unicode SAP システムの場合は CHAR(*n*)、Unicode SAP システムの場合は NCHAR(*n*)
CUKY	通貨キー。この型のフィールドは、CURR 型のフィールドによって参照されます。	非 Unicode SAP システムの場合は CHAR(*n*)、Unicode SAP システムの場合は NCHAR(*n*)
CURR	通貨フィールド。DEC フィールドに対応します。フィールドは、CUKY 型のフィールドを指します。	CHAR(*n*)
DATS	日付値。	DATE
DEC	符号付き固定小数点 10 進数。	CHAR(*n*)
FLTP	浮動小数点数。	DOUBLE
INT1	非常に小さな符号付きの正確な整数。	TINYINT
INT2	小さな符号付きの正確な整数。	SMALLINT
INT4	通常の符号付きの正確な整数。	INTEGER
INT8	大きな符号付きの正確な整数。	BIGINT
LANG	特殊機能用の独自のフィールド形式を持つ言語キー。 変換 Exit ISOLA は、表示される値をデータベースの値に変換しますが、その逆が当てはまります。	非 Unicode SAP システムの場合は CHAR(*n*)、Unicode SAP システムの場合は NCHAR(*n*)
LCHR	固定長の文字列。	非 Unicode SAP システムの場合は VARCHAR(*n*)、Unicode SAP システムの場合は NVARCHAR(*n*)

ABAP SAP データ型	説明	データ取得に使用される FedSQL データ型
LRAW	解釈されない可変長のバイト文字列。	VARBINARY(*n*)
NUMC	テキスト文字列。	非 Unicode SAP システムの場合は CHAR(*n*)、Unicode SAP システムの場合は NCHAR(*n*)
PREC	QUAN フィールドの精度。	CHAR(*n*)
QUAN	DEC フィールドに対応する数量。	CHAR(*n*)
RAW	解釈されないバイト文字列。	BINARY
TIMS	時間値。	TIME(*p*)
UNIT	QUAN データ型によって参照される単位キー。	非 Unicode SAP システムの場合は CHAR(*n*)、Unicode SAP システムの場合は NCHAR(*n*)
VARC	可変長の文字列データ。SAP リリース 3.0 以降、このデータ型のフィールドの作成はサポートされなくなりました。SELECT ステートメントの WHERE 条件を除いて、このデータ型の既存のフィールドを使用できます。	非 Unicode SAS システムの場合は VARCHAR(*n*)、Unicode SAP システムの場合は NVARCHAR(*n*)

SAP HANA のデータ型

次の表に、SAP HANA データベースのデータ型サポートを示します。

SAP HANA データ型に関するデータソース固有の情報については、SAP HANA データベースのドキュメントを参照してください。

注: この表の情報は、CAS で処理されるデータには適用されません。CAS にロードされたデータは、CAS データ型に変換されます。CAS データ型の変換については、*SAS Cloud Analytic Services: ユーザーガイド*の SAS Data Connector for SAP HANA のドキュメントを参照してください。

表 A20.23　FedSQL データ型と SAP HANA で使用されるデータ型のマッピング

データ型定義キーワード [1]	SAP HANA データ型	説明	返されるデータ型
[2]	ALPHANUM(*n*)	可変長の文字列。	NVARCHAR(*n*)
BIGINT	BIGINT	64 ビット整数。	BIGINT
BINARY(*n*)	BINARY(*n*)	固定長のバイナリ値。	BINARY(*n*)
[2]	BLOB	可変長のバイナリラージオブジェクト文字列。	[2]
CHAR(*n*)	CHAR(*n*)	可変長の文字列。	CHAR(*n*)
[2]	CLOB	可変長文字のラージオブジェクト文字列。	[2]
DATE	DATE	年、月、および日の値。	DATE
DECIMAL|NUMERIC(**p**,*s*)	DECIMAL(*p*,*s*)	符号付きの正確な固定小数点 10 進数。	DECIMAL(*p*,*s*)[3]
DOUBLE	DOUBLE	倍精度浮動小数点数。	DOUBLE
FLOAT	FLOAT(*p*)	倍精度浮動小数点数。	DOUBLE
INTEGER	INTEGER	32 ビット整数。	INTEGER
NCHAR(*n*)	NCHAR(*n*)	固定長の Unicode 文字列。	NCHAR(*n*)
[2]	NCLOB	固定長文字のラージオブジェクト文字列。	[2]
NVARCHAR(*n*)	NVARCHAR(*n*)	可変長 Unicode 文字のラージオブジェクト文字列。	NVARCHAR(*n*)
REAL	REAL	浮動小数点数。	REAL
[2]	SECONDATE	日時値。	[2]
[2]	SHORTTEXT(*n*)	可変長の文字列。	NVARCHAR(*n*)
[2]	SMALLDECIMAL(*p*,*s*)	浮動小数点 10 進数。	DECIMAL(*p*,*s*)[3]
SMALLINT	SMALLINT	16 ビット整数。	SMALLINT

データ型定義キーワード [1]	SAP HANA データ型	説明	返されるデータ型
[2]	TEXT	可変長 Unicode 文字のラージオブジェクト文字列。	[2]
TIME(*p*)	TIME(*p*)	時間値。	TIME(*p*)
TIMESTAMP(*p*)	TIMESTAMP(*p*)	日時値。	TIMESTAMP(**p**)
TINYINT	TINYINT	符号なし 8 ビット整数。	TINYINT
VARBINARY(*n*)	VARBINARY(*n*)	可変長のバイナリ文字列。	VARBINARY(*n*)
VARCHAR(*n*)	VARCHAR(*n*)	可変長の文字列。	VARCHAR(*n*)

1 データ型のマッピング方法を制御する CT_PRESERVE=接続引数は、データ型を定義できるかどうかに影響する可能性があります。値 FORCE (デフォルト)および FORCE_COL_SIZE は、データ型を定義できるかどうかには影響しません。要求されたデータ型がデータソースにネイティブでない場合、または指定された精度またはスケールがデータソースの範囲内にない場合、値 STRICT および SAFE はエラーになる可能性があります。

2 SAP HANA データ型は定義できません。データが取得されると、ネイティブデータ型が同様のデータ型にマッピングされます。

3 FedSQL では、DECIMAL データ型は、データソースに完全に渡すことができる要求でサポートされています。データソースに完全に渡すことができない要求の DECIMAL 列は、DOUBLE として処理されるため、精度に影響を与える可能性があります。詳細については、*"データ型" (SAS FedSQL 言語リファレンス)*を参照してください。

SAP IQ のデータ型

次の表に、SAP IQ データベースのデータ型サポートを示します。

SAP IQ データベースのデータ型に関するデータソース固有の情報については、SAP IQ データベースのドキュメントを参照してください。

注: このデータソースは CAS サーバーではサポートされていません。

表 A20.24 FedSQL データ型と SAP IQ で使用されるデータ型のマッピング

データ型定義キーワード [1]	SAP IQ データ型	説明	返されるデータ型
BIGINT	BIGINT	64 ビット整数。	BIGINT
BINARY(*n*)	BINARY(*n*)	固定長のバイナリ文字列。	BINARY(*n*)
[3]	BIT	値 0 または 1 のみを保存する整数。	[3]

データ型定義キーワード [1]	SAP IQ データ型	説明	返されるデータ型
CHAR(*n*)	CHAR(*n*)	可変長の文字列。	VARCHAR(*n*)
DATE	DATE	日付値。	DATE
DECIMAL | NUMERIC(*p*,*s*)	DECIMAL(*p*,*s*)	符号付きの正確な固定小数点 10 進数。	DECIMAL(*p*,*s*)[4]
DOUBLE	DOUBLE	倍精度浮動小数点数。	DOUBLE
FLOAT	FLOAT(*p*)	倍精度浮動小数点数。	DOUBLE
INTEGER	INTEGER	32 ビット整数。	INTEGER
[2]	LONG BINARY	可変長のバイナリ文字列。	VARBINARY(*n*)
REAL	REAL	浮動小数点数。	REAL
SMALLINT	SMALLINT	16 ビット整数。	SMALLINT
TIME(*p*)	TIME(*p*)	時間値。	TIME(*p*)
TIMESTAMP(*p*)	TIMESTAMP(*p*)	日時値。	TIMESTAMP(*p*)
TINYINT	TINYINT	符号なし 8 ビット整数。	TINYINT
VARBINARY(*n*)	VARBINARY(*n*)	可変長のバイナリ文字列。	VARBINARY(*n*)
VARCHAR(*n*)	CHAR(*n*)	可変長の文字列。	VARCHAR(*n*)

1 データ型のマッピング方法を制御する CT_PRESERVE=接続引数は、データ型を定義できるかどうかに影響する可能性があります。値 FORCE (デフォルト)および FORCE_COL_SIZE は、データ型を定義できるかどうかには影響しません。要求されたデータ型がデータソースにネイティブでない場合、または指定された精度またはスケールがデータソースの範囲内にない場合、値 STRICT および SAFE はエラーになる可能性があります。

2 SAP IQ データ型は定義できません。データが取得されると、ネイティブデータ型が同様のデータ型にマッピングされます。

3 SAP IQ データ型を定義または取得できません。

4 DECIMAL データ型は、データソースに完全に渡すことができる要求でサポートされています。データソースに完全に渡すことができない要求の DECIMAL 列は、DOUBLE として処理されるため、精度に影響を与える可能性があります。

SingleStore のデータ型

次の表に、SingleStore データベースのデータ型サポートを示します。

NCHAR、NVARCHAR、REAL、VARBINARY データ型は、データ型定義ではサポートされていません。

SingleStore データ型に関するデータソース固有の情報については、SingleStore データベースのドキュメントを参照してください。

注: この表の情報は、CAS で処理されるデータには適用されません。CAS にロードされたデータは、CAS データ型に変換されます。CAS データ型の変換については、*SAS Cloud Analytic Services: ユーザーガイド*の SAS Data Connector for SingleStore のドキュメントを参照してください。

表 A20.25 FedSQL データ型と SingleStore で使用されるデータ型のマッピング

データ型定義キーワード [1]	SingleStore データ型	説明	返されるデータ型
BIGINT	BIGINT	大きな符号付きの正確な整数。	BIGINT
BINARY(*n*)	BINARY(*n*)	固定長のバイナリ文字列。	BINARY(*n*)
[2]	BLOB	バイナリラージオブジェクト文字列。	[2]
CHAR(*n*)	CHAR(*n*)[3]	固定長の文字列。	CHAR(*n*)
DATE	DATE	日付値。	DATE
[2]	DATETIME	日時値。	[2]
DECIMAL|NUMERIC(*p*,*s*)	DECIMAL(*p*,*s*)	符号付き固定小数点 10 進数。	DECIMAL(*p*,*s*)[4]
DOUBLE	DOUBLE	符号付き倍精度浮動小数点数。	DOUBLE
[2]	ENUM(*values*)	許可された値のリストからの文字値。	[2]
FLOAT	FLOAT(*p*)	符号付き倍精度浮動小数点数。	DOUBLE
INTEGER	INT	通常の符号付きの正確な整数。	INTEGER
[2]	JSON	JSON 形式の可変長文字列。	VARCHAR

データ型定義キーワード [1]	SingleStore データ型	説明	返されるデータ型
[2]	LONGBLOB	可変長のバイナリラージオブジェクト文字列。	[2]
[2]	LONGTEXT	可変長のテキスト文字列。	[2]
[2]	MEDIUMBLOB	可変長のバイナリラージオブジェクト文字列。	[2]
[2]	MEDIUMINT	通常の符号付きの正確な整数。	[2]
[2]	MEDIUMTEXT	可変長の文字列。	[2]
[2]	SET(*values*)	許可された値のリストからの文字値。	[2]
SMALLINT	SMALLINT	小さな符号付きの正確な整数。	SMALLINT
[2]	TEXT	テキスト文字列。	[2]
TIME(*p*)	TIME(*p*)	時間値。	TIME(*p*)
TIMESTAMP(*p*)	DATETIME(*p*)	日時値。	TIMESTAMP(*p*)
[2]	TINYBLOB	可変長のバイナリラージオブジェクト文字列。	[2]
TINYINT	TINYINT	非常に小さな符号付きの正確な整数。	TINYINT
[2]	TINYTEXT	可変長のテキスト文字列。	[2]
VARCHAR(*n*)	VARCHAR(*n*)	可変長の文字列。	VARCHAR(*n*)

1 データ型のマッピング方法を制御する CT_PRESERVE=接続引数は、データ型を定義できるかどうかに影響する可能性があります。値 FORCE (デフォルト)および FORCE_COL_SIZE は、データ型を定義できるかどうかには影響しません。要求されたデータ型がデータソースにネイティブでない場合、または指定された精度またはスケールがデータソースの範囲内にない場合、値 STRICT および SAFE はエラーになる可能性があります。

2 SingleStore データ型は定義できません。データが取得されると、ネイティブデータ型が同様のデータ型にマッピングされます。

3 DS2 は、固定文字値の空白スペースを処理するために ANSI SQL 規則を使用します。MySQL 8 は、デフォルトで末尾の空白スペースを重要な値として扱います。この違いは、DS2 で FedSQL ステートメントを使用するときに実行される文字比較に干渉します。DS2 を使用して MySQL 8 CHAR()データに対して FedSQL ステートメントを発行する場合、データが正しく読み取られるように、FedSQL ステートメントで TRIM 関数を使用する必要があります。DS2 で FedSQL ステートメントがサポートされている場所については、"DS2 と FedSQL の使用" (*SAS DS2 プログラマガイド*)を参照してください。ま

た、“例: MySQL 8 CHAR()データから末尾のスペースをフィルタリングする” (2178 ページ)も参照してください。MySQL 8 は、比較のデフォルト照合順序として utf8mb4_0900_ai_ci を使用します。

4 FedSQL では、DECIMAL データ型は、データソースに完全に渡すことができる要求でサポートされています。データソースに完全に渡すことができない要求の DECIMAL 列は、DOUBLE として処理されるため、精度に影響を与える可能性があります。詳細については、“データ型” (*SAS FedSQL 言語リファレンス*)を参照してください。

Snowflake のデータ型

次の表に、Snowflake のデータ型サポートを示します。

FedSQL の NCHAR、NVARCHAR、および TINYINT データ型は、データ型定義ではサポートされていません。

Snowflake データ型に関するデータソース固有の情報については、Snowflake のドキュメントを参照してください。

注: この表の情報は、CAS で処理されるデータには適用されません。CAS にロードされたデータは、CAS データ型に変換されます。CAS データ型の変換については、*SAS Cloud Analytic Services: ユーザーガイド*の SAS Data Connector for Snowflake のドキュメントを参照してください。

表 A20.26 FedSQL データ型と Snowflake で使用されるデータ型のマッピング

データ型定義キーワード	Snowflake データ型	説明	返されるデータ型
[1]	ARRAY	値が VARIANT 型の値である、任意のサイズの密または疎な配列。	VARCHAR
BIGINT	BIGINT	64 ビット整数。	BIGINT
BINARY(*n*)	BINARY(*n*)	固定長のバイナリ文字列。	BINARY
[1]	BOOLEAN	true、false、または不明(null)の値。	BINARY
CHAR(*n*)	VARCHAR(1)	固定長の文字列。	VARCHAR
DATE	DATE	日付値。	DATE
DECIMAL|NUMERIC(*p*,*s*)	NUMBER(*p*,*s*)	符号付きの正確な固定小数点 10 進数。	DECIMAL(*p*,*s*)[2]
DOUBLE	FLOAT	倍精度浮動小数点数。	DOUBLE

データ型定義キーワード	Snowflake データ型	説明	返されるデータ型
FLOAT	FLOAT、FLOAT4、FLOAT8	倍精度浮動小数点数。	DOUBLE
INTEGER	精度とスケールのない NUMBER。	32 ビット整数。	INTEGER
[1]	OBJECT	キーと値のペアのコレクション。キーは空でない文字列であり、値は VARIANT 型の値です。	VARCHAR
REAL	FLOAT	浮動小数点数。	FLOAT
SMALLINT	精度とスケールのない NUMBER。	16 ビット整数。	SMALLINT
TIME(*p*)	TIME(*p*)	時間値。	TIME(*p*)
TIMESTAMP(*p*)	TIMESTAMP、TIMESTAMP_LTZ、TIMESTAMP_NTZ、TIMESTAMP_TZ	日時値。	TIMESTAMP(*p*)
VARBINARY(*n*)	VARBINARY	可変長のバイナリ文字列。	BINARY
VARCHAR(*n*)	VARCHAR(*n*)	可変長の文字列。デフォルト(および最大)の長さは 16,777,216 です。	VARCHAR
[1]	VARIANT	OBJECT や ARRAY を含む他の Snowflake データ型の値を最大 16MB まで保存できるユニバーサルデータ型。	VARCHAR

1 Snowflake データ型は定義できません。データが取得されると、ネイティブデータ型が同様のデータ型にマッピングされます。

2 FedSQL では、DECIMAL データ型は、データソースに完全に渡すことができる要求でサポートされています。データソースに完全に渡すことができない要求の DECIMAL 列は、DOUBLE として処理されるため、精度に影響を与える可能性があります。詳細については、*"データ型" (SAS FedSQL 言語リファレンス)*を参照してください。

Spark のデータ型

次の表に、Apache Spark のデータ型サポートを示します。

NCHAR および NVARCHAR データ型は、データ定義には使用できません。

Spark SQL データ型に関するデータソース固有の情報については、Spark SQL のドキュメントを参照してください。

注: この表の情報は、CAS で処理されるデータには適用されません。CAS にロードされたデータは、CAS データ型に変換されます。CAS データ型の変換については、*SAS Cloud Analytic Services: ユーザーガイド*の SAS Data Connector for Spark のドキュメントを参照してください。

表 A20.27 FedSQL データ型と Apache Spark で使用されるデータ型のマッピング

データ型定義キーワード	Spark データ型	説明	返されるデータ型
[1]	ARRAY<*data-type*>	整数の配列(インデックス可能なリスト)。	STRING[5, 7]
BIGINT	BIGINT	-9,223,372,036,854,775,808 から 9,223,372,036,854,775,807 までの符号付き 8 バイト整数。	BIGINT
BINARY(*n*)	BINARY	最大 32K の可変長のバイナリ文字列。	BINARY
[1]	BOOLEAN	テキストの true または false の値。	TINYINT
CHAR(*n*)	CHAR(*n*)[9]	最大 255 文字の文字列。[3]	CHAR
DATE	DATE[2]	ANSI SQL の日付値。	DATE
DECIMAL/ NUMERIC(*p*,*s*)	DECIMAL	38 桁の精度の固定小数点 10 進数。	DECIMAL(*p*,*s*)[4]
DOUBLE	DOUBLE	8 バイトの倍精度浮動小数点数。	DOUBLE

データ型定義キーワード	Spark データ型	説明	返されるデータ型
FLOAT	FLOAT	8 バイトの倍精度浮動小数点数。	DOUBLE
INTEGER	INTEGER	符号付き 4 バイト整数。	INTEGER
[1]	MAP<*primitive-type*, *data-type*>	キーと値のペアの連想配列。	STRING[5, 7]
REAL	DOUBLE	64 ビットの倍精度浮動小数点数。	DOUBLE
SMALLINT	SMALLINT	-32,768 から 32,767 までの符号付き 2 バイト整数。	SMALLINT
[1]	STRING	可変長の文字列。	VARCHAR(*n*)[5, 6]
[1]	STRUCT<*col-name*: *data-type*>	確立された列要素とデータ型を持つ構造。列要素とデータ型は、ダブルドット(:)表記を使用してマッピングされます。	STRING[5, 7]
TIME(*p*)	[8]	時間値。	STRING
TIMESTAMP(*p*)	TIMESTAMP	オプションのナノ秒精度の UNIX タイムスタンプ。	TIMESTAMP[(*p*)]
[1]	TIMESTAMP_NTZ[10]	タイムゾーンなしのタイムスタンプ。	TIMESTAMP
TINYINT	TINYINT	-128 から 127 までの符号付き 1 バイト整数。	TINYINT
VARBINARY	BINARY	最大 32K の可変長のバイナリ文字列。	BINARY

データ型定義キーワード	Spark データ型	説明	返されるデータ型
VARCHAR(*n*)	VARCHAR(*n*)[9]	可変長の文字列。	VARCHAR(*n*)[5]

1 Spark データ型は定義できません。データが取得されると、ネイティブデータ型が同様のデータ型にマッピングされます。
2 サポートされている日付値は、0001 年 1 月 1 日から 9999 年 12 月 31 日までです。9999 より後の日付値は null 値として読み戻されます。
3 255 文字を超える値で CHAR を指定すると、列はかわりに VARCHAR(*n*)として作成されます。
4 FedSQL では、DECIMAL データ型は、データソースに完全に渡すことができる要求でサポートされています。データソースに完全に渡すことができない要求の DECIMAL 列は、DOUBLE として処理されるため、精度に影響を与える可能性があります。詳細については、*"データ型" (SAS FedSQL 言語リファレンス)*を参照してください。
5 VARCHAR(*n*)および Hive 複合型の最大長は、DBMAX_TEXT=データソース接続オプションによって決定されます。
6 SASFMT TableProperties は、STRING 列を読み取るときに適用されます。
7 複合型 ARRAY、MAP、STRUCT は、基になる複合型の STRING 表現として読み取られます。ARRAY 値は、角かっこで囲まれて読み戻されます(例: [1、2、4])。STRUCT と MAP の値は、中かっこで囲まれて読み戻されます。例: {"firstname":"robert","nickname":"bob"}。
8 Spark は TIME(*p*)データ型をサポートしていません。データが Spark から読み取られると、SAS TIME8.出力形式を指定する SASFMT TableProperties が定義されている STRING 列が、TIME(*p*)データ型に変換されます。TIME 型がデータ定義に使用される場合、SASFMT TableProperties を使用して STRING 列にマッピングされます。秒の端数は保持されません。SASFMT TableProperties の詳細については、*SAS/ACCESS for Relational Databases: リファレンス*の"SAS Table Properties for Spark and Hadoop"を参照してください。
9 Apache Spark 2.0.0 では、Spark テーブルで許可される列の最大数は https://issues.apache.org/jira/browse/SPARK-18016 によって制限される場合があります。
10 TIMESTAMP_NTZ 型は Databricks でのみサポートされ、Databricks Runtime 13.3 LTS 以上を使用している場合にサポートされます。

Teradata のデータ型

次の表に、Teradata データベースのデータ型サポートを示します。

NCHAR、NVARCHAR、および REAL データ型は、データ型定義ではサポートされていません。

Teradata データ型に関するデータソース固有の情報については、Teradata データベースのドキュメントを参照してください。

注: この表の情報は、CAS で処理されるデータには適用されません。CAS にロードされたデータは、CAS データ型に変換されます。CAS データ型の変換については、*SAS Cloud Analytic Services: ユーザーガイド*の SAS Data Connector for Teradata のドキュメントを参照してください。

表 A20.28 FedSQL データ型と Teradata で使用されるデータ型のマッピング

データ型定義キーワード[1]	Teradata データ型	説明	返されるデータ型
BIGINT	BIGINT	大きな符号付きの正確な整数。	BIGINT

データ型定義キーワード [1]	Teradata データ型	説明	返されるデータ型
BINARY(*n*)	BYTE(*n*)	固定長のバイナリ文字列。	BINARY(*n*)
[2]	BLOB	ラージバイナリオブジェクト。	[2]
CHAR(*n*)	CHAR(*n*)	固定長の文字列。	CHAR(*n*)
[2]	CLOB	ラージ文字オブジェクト。	[2]
DATE	DATE	日付値。	DATE
DECIMAL(*p*,*s*)	DECIMAL(*p*,*s*)	符号付き固定小数点 10 進数。	DECIMAL(*p*,*s*)[3]
NUMERIC(*p*,*s*)	NUMBER(*p*,*s*)	符号付き可変長小数点 10 進数。	NUMERIC(*p*,*s*)[3]
DOUBLE	FLOAT	符号付き倍精度浮動小数点数。	DOUBLE
FLOAT	FLOAT(*p*)	符号付き倍精度浮動小数点数。	DOUBLE
INTEGER	INTEGER	通常の符号付きの正確な整数。	INTEGER
[2]	LONG VARCHAR	可変長の文字列。	[2]
SMALLINT	SMALLINT	小さな符号付きの正確な整数。	SMALLINT
TIME(*p*)	TIME(*p*)	時間値。	TIME(*p*)
TIMESTAMP(*p*)	TIMESTAMP(*p*)	日時値。	TIMESTAMP(*p*)
TINYINT	BYTEINT	非常に小さな符号付きの正確な整数。	TINYINT
VARBINARY(*n*)	VARBYTE(*n*)	可変長のバイナリ文字列。	VARBINARY(*n*)
VARCHAR(*n*)	VARCHAR(*n*)	可変長の文字列。	VARCHAR(*n*)

1 データ型のマッピング方法を制御する CT_PRESERVE=接続引数は、データ型を定義できるかどうかに影響する可能性があります。値 FORCE (デフォルト)および FORCE_COL_SIZE は、データ型を定義できるかどうかには影響しません。要求されたデータ型がデータソースにネイティブでない場合、または指定された精度またはスケールがデータソースの範囲内にない場合、値 STRICT および SAFE はエラーになる可能性があります。

2 Teradata データ型は定義できません。データが取得されると、ネイティブデータ型が同様のデータ型にマッピングされます。

3 FedSQL では、DECIMAL および NUMERIC データ型は、データソースに完全に渡すことができる要求でサポートされています。データソースに完全に渡すことができない要求の DECIMAL および NUMERIC 列は、DOUBLE として処理されるため、精度に影響を与える可能性があります。詳細については、*"データ型" (SAS FedSQL 言語リファレンス)*を参照してください。

Vertica のデータ型

次の表に、Vertica データベースのデータ型サポートを示します。

NCHAR および NVARCHAR データ型は、データ定義ではサポートされていません。

Vertica データ型に関するデータソース固有の情報については、Vertica データベースのドキュメントを参照してください。

注: この表の情報は、CAS で処理されるデータには適用されません。CAS にロードされたデータは、CAS データ型に変換されます。CAS データ型の変換については、*SAS Cloud Analytic Services: ユーザーガイド*の SAS Data Connector for Vertica のドキュメントを参照してください。

表 A20.29 FedSQL データ型と Vertica で使用されるデータ型のマッピング

データ型定義キーワード [1]	Vertica SQL 識別子	説明	返されるデータ型
BIGINT	BIGINT	符号付き 64 ビット整数。8 バイトのストレージが必要です。	BIGINT
BINARY(*n*)	BINARY(*n*)	固定長のバイナリ文字列。	BINARY(*n*)
[2]	BOOLEAN	論理ブール(true/false)値。	[2]
[2]	BYTEA	可変長のバイナリ文字列。	[2]
CHAR(*n*)	CHAR(*n*)	固定長の文字列。	CHAR(*n*)
DATE	DATE	日付値。	DATE
[2]	DATETIME	タイムゾーン付きまたはタイムゾーンなしの日時値。	TIMESTAMP(*p*)
DECIMAL|NUMERIC(*p*,*s*)	DECIMAL|NUMERIC(*p*,*s*)	符号付きの固定精度およびスケール数。デフォルトの精度とスケール: 37, 15。	DECIMAL|NUMERIC(*p*,*s*)[3]

データ型定義キーワード [1]	Vertica SQL 識別子	説明	返されるデータ型
DOUBLE	DOUBLE PRECISION	符号付き 64 ビット IEEE 浮動小数点数。8 バイトのストレージが必要です。	DOUBLE
FLOAT	FLOAT、FLOAT8	符号付き 64 ビット IEEE 浮動小数点数。8 バイトのストレージが必要です。	DOUBLE
INTEGER	INTEGER	符号付き 64 ビット整数。8 バイトのストレージが必要です。	BIGINT
[2]	INTERVAL	期間。	[2]
[2]	INTERVAL DAY TO SECOND	期間。	[2]
[2]	INTERVAL YEAR TO MONTH	期間。	[2]
[2]	LONG VARCHAR	空間データなどの可変長の生のバイト値。	[2]
[2]	LONG BINARY	長い可変長のバイナリ文字列。	[2]
[2]	MONEY(*p*,*s*)	符号付きの固定の精度とスケールの金銭または通貨の値。デフォルトの精度とスケール: 18, 4。	[2]
[2]	NUMBER(*p*,*s*)	符号付きの固定精度およびスケール数。デフォルトの精度とスケール: 38,0。	[2]
[2]	RAW	可変長のバイナリ文字列。	[2]
REAL	REAL	符号付き 64 ビット IEEE 浮動小数点数。8 バイトのストレージが必要です。	DOUBLE

データ型定義キーワード [1]	Vertica SQL 識別子	説明	返されるデータ型
[2]	SMALLDATETIME	タイムゾーン付きまたはタイムゾーンなしの日時値。	TIMESTAMP(*p*)
SMALLINT	SMALLINT	符号付き 64 ビット整数。8 バイトのストレージが必要です。	BIGINT
TIME(*p*)	TIME(*p*)	タイムゾーンのない時間値。	TIME(*p*)
[2]	TIMETZ	タイムゾーン付き時間値。	TIME(*p*)
TIMESTAMP(*p*)	TIMESTAMP	タイムゾーンのない日時値。	TIMESTAMP(*p*)
[2]	TIMESTAMPTZ	タイムゾーン付き日時値。	TIMESTAMP(*p*)
TINYINT	TINYINT	符号付き 64 ビット整数。8 バイトのストレージが必要です。	BIGINT
VARBINARY(*n*)	VARBINARY	可変長のバイナリ文字列。	VARBINARY(*n*)
VARCHAR(*n*)	VARCHAR(*n*)、TEXT	可変長の文字列。	VARCHAR(*n*)

1 データ型のマッピング方法を制御する CT_PRESERVE=接続引数は、データ型を定義できるかどうかに影響する可能性があります。値 FORCE (デフォルト)および FORCE_COL_SIZE は、データ型を定義できるかどうかには影響しません。要求されたデータ型がデータソースにネイティブでない場合、または指定された精度またはスケールがデータソースの範囲内にない場合、値 STRICT および SAFE はエラーになる可能性があります。

2 Vertica データ型は定義できません。データが取得されると、ネイティブデータ型が同様のデータ型にマッピングされます。

3 FedSQL では、DECIMAL データ型は、データソースに完全に渡すことができる要求でサポートされています。データソースに完全に渡すことができない要求の DECIMAL 列は、DOUBLE として処理されるため、精度に影響を与える可能性があります。詳細については、*"データ型" (SAS FedSQL 言語リファレンス)*を参照してください。

Yellowbrick のデータ型

次の表に、Yellowbrick データベースのデータ型サポートを示します。

BINARY、NCHAR、NVARCHAR、TINYINT、VARBINARY データ型は、データ型定義ではサポートされていません。

Yellowbrick データ型に関するデータソース固有の情報については、Yellowbrick データベースのドキュメントを参照してください。

注: この表の情報は、CAS で処理されるデータには適用されません。CAS にロードされたデータは、CAS データ型に変換されます。CAS データ型の変換については、*SAS Cloud Analytic Services: ユーザーガイド*の SAS Data Connector for Yellowbrick のドキュメントを参照してください。

表 A20.30 FedSQL データ型と Yellowbrick で使用されるデータ型のマッピング

データ型定義キーワード [1]	Yellowbrick データ型	説明	返されるデータ型
BIGINT	BIGINT	大きな符号付きの正確な整数または符号付き 8 バイト整数。	BIGINT
[2]	BOOLEAN	論理ブール(true/false)値。	[2]
CHAR(*n*)	CHAR(*n*)	固定長の文字列。	CHAR(*n*)
DATE	DATE	日付値。	DATE
DECIMAL|NUMERIC(*p*,*s*)	NUMERIC(*p*,*s*)	符号付き固定小数点 10 進数。	DECIMAL|NUMERIC(*p*,*s*)[3]
DOUBLE	DOUBLE PRECISION	符号付き倍精度浮動小数点数。	DOUBLE
FLOAT	DOUBLE PRECISION、FLOAT、FLOAT8	符号付き倍精度浮動小数点数。	DOUBLE
INTEGER	INTEGER	通常の符号付きの正確な整数。	INTEGER
[2]	MACADDR、MACADDR8	メディアアクセスコントロールアドレス。	[2]
REAL	REAL、FLOAT4	符号付き単精度浮動小数点数。	REAL
SMALLINT	SMALLINT	小さな符号付きの正確な整数。	SMALLINT
TIME(*p*)	TIME(*p*)	時間値。	TIME(*p*)
TIMESTAMP(*p*)	TIMESTAMP(*p*)	日時値。	TIMESTAMP(*p*)
[2]	UUID	ユニバーサル一意識別子。	[2]

データ型定義キーワード [1]	Yellowbrick データ型	説明	返されるデータ型
VARCHAR(*n*)	CHARACTER VARYING(*n*)	可変長の文字列。	VARCHAR(*n*)

1 データ型のマッピング方法を制御する CT_PRESERVE=接続引数は、データ型を定義できるかどうかに影響する可能性があります。値 FORCE (デフォルト)および FORCE_COL_SIZE は、データ型を定義できるかどうかには影響しません。要求されたデータ型がデータソースにネイティブでない場合、または指定された精度またはスケールがデータソースの範囲内にない場合、値 STRICT および SAFE はエラーになる可能性があります。

2 Yellowbrick データ型は定義できません。データが取得されると、ネイティブデータ型が同様のデータ型にマッピングされます。

3 FedSQL では、DECIMAL データ型は、データソースに完全に渡すことができる要求でサポートされています。データソースに完全に渡すことができない要求の DECIMAL 列は、DOUBLE として処理されるため、精度に影響を与える可能性があります。詳細については、*“データ型” (SAS FedSQL 言語リファレンス)*を参照してください。

付録 2

DS2 式

式とは **2095**

式のタイプ **2096**
- 式の概要 2096
- 一次式 2097
- 複合式 2098
- DOT 演算子式 2100
- 関数式 2101
- IF 式 2101
- IN 式 2104
- LIKE 式 2106
- メソッド式 2108
- OF 演算子 2109
- パッケージメソッド式 2114
- SYSTEM 式 2115
- THIS 式 2115
- SELECT 式 2116
- 添字演算子 2119

式の演算子 **2120**
- 複合式での演算子の優先順位 2120
- 演算子別の式の値 2122
- DS2 での短絡評価 2125

式とは

式は、データ値を生成するプログラミング要素です。式には、一連の命令を形成し、値に解決されるオペランドとオプションの演算子が含まれます。

オペランドは、単一の定数または変数にすることも、式にすることもできます。演算子は、オペランドの計算、比較、または連結を要求する記号です。

DS2 式の例を次に示します。

```
3
"col1"
a = b * c
s || 1 || z
a >= b**(c - 8)
system.put(a*5,hex.)
x'ff00effc'
```

式のタイプ

式の概要

式の基本的なタイプは、一次式です。演算子を使用して式(一次または複合)を結合する式は、複合式です。式は、関数式やメソッド式などの DS2 コンストラクトを呼び出すことができます。式は、“SYSTEM 式”や THIS 式など、関数または変数のスコープに影響を与える可能性があります。 IN 式は、式の結果がリストに含まれているかどうかに基づいて真理値を返します。

式が単純か複合か、コンストラクトを呼び出すか、スコープがグローバルかどうかに関係なく、すべての式には値とデータ型があります。式のデータ型は、変数や関数などの式が取る可能性のある値を制限します。DS2 は複合式に遭遇すると、ルールに従って、式の各部分を評価する順序を決定します。

注: 論理演算では、IF 式などの任意の式の欠損値は、SAS モードで偽と評価されます。ANSI モードでは、null 値は真にも偽にも評価されません。この例を SAS モードで実行すると、False が SAS ログに書き込まれます。この例を ANSI モードで実行すると、Null が SAS ログに書き込まれます。

```
proc ds2;
data _null_;
  declare double d;
  method init();
    if d then put 'True';
      else if not d then put 'False';
      else put 'Null';
  end;
  enddata;
run;
quit;
```

一次式

最も単純な形式では、一次式は、識別子、数値、文字列、2 進数および 16 進数の定数、リテラル値、日付と時間の値、null 値です。

```
a
"var_a"
5.33
'Company'
date '2007-08-24'
```

次の表は、基本的な一次式、そのデータ型、および例を示しています。

表 A21.1 一次式のデータ型と例

式のタイプ	簡単な説明	データ型	例
2 進定数	2 進定数と 16 進定数	VARBINARY	x'FE' b'01000011'
文字定数	文字列	CHAR、VARCHAR	'New Report' a='Stock'; b='Report'; 1
各国文字定数	各国文字列	NCHAR、NVARCHAR	n'New Report' a=n'Stock'; b=n'Report'; 1
SYSTEMTHIS	System、THIS	式の解決された型	system.put(x,5.) this.s
日付/時間定数	日付と時間の値	DATE、TIME、TIMESTAMP[2]	date'2023-01-01' time'20:44:59' timestamp'2023-02-07 07:00:00.7569'
識別子	さまざまな言語要素の名前を提供する	宣言された型またはデフォルトの型。デフォルトは DOUBLE。	a "part1"
整数定数	整数	INTEGER、BIGINT	123

式のタイプ	簡単な説明	データ型	例
新規	パッケージメソッドをインスタンス化する	式の解決された型	a=_new_ package-name(); 1
小数定数	浮動小数点数	DOUBLE、FLOAT、REAL	5. 4.3
Null 定数	Null 式	なし	NULL
かっこ	評価の優先順位を上げるためにかっこで囲まれた演算子とオペランド	かっこで囲まれた式の解決された型。	(x) (a + b) - c 1
欠損定数	SAS 欠損数値式	DOUBLE	. .A
数値定数	正確な数値	DECIMAL、NUMERIC	99n 12345678901234567890123456789O.12N

1 この例では、一次式は式または割り当てステートメントに含まれています。式の例が強調表示されています。
2 TO_DOUBLE 関数を複合式で使用すると、SAS の日付、時間、または日時の値をエンコードする DOUBLE 値を生成できます。

複合式

複合式は、次の式のように、式と演算子を組み合わせてより拡張的な式を作成します。

```
a + b * -c - 5
a = b = 5
x | y & a < c * d
x**y**z - 9 >= f
z >< c + e <> u**y * 10
x || c + 7
a in (1,2,3)
```

複合式の評価は、表 A2.7 (2121 ページ)に示されている演算子の優先順位と、一次式のデータ型に基づいています。計算を行う前に、オペランドのデータ型が同じ一般的なデータ型(数値、文字、バイナリ、日付/時間)である必要があります。データ型が同じ場合、処理は続行されます。同じでない場合、オペランドのデータ型は、演算子とオペランドのデータ型に基づいて変換されます。

演算子の優先順位と一次式のデータ型がどのように評価されるかを理解するために、式 a + b / -c - 5 を考えてみましょう。この式には 4 つの演算子があります。

- 二項+演算子
- 二項/演算子

- 単項-演算子
- 二項-演算子

演算子の優先順位によって、演算の評価順序が決まります。表 A2.7 (2121 ページ)を見ると、単項-演算子の優先順位は 2、二項/演算子の優先順位は 3、二項+演算子と二項-演算子はどちらも優先順位が 4 です。2 つの演算子の優先順位が同じ場合、結合規則によって評価の順序が決まります。二項+演算子と二項-演算子には、左から右への結合があります。したがって、前述の式は評価のために次のようにグループ化されます。
((a + (b / (-c))) – 5)

型変換も演算の評価に影響します。二項算術演算では、両方のオペランドを同じ型にする必要があります。オペランドが同じ型でない場合、DS2 コンパイラは、算術式の優先規則の型変換を使用して、一方のオペランドを他方のオペランドと同じ型に変換します。

表 A21.2　算術演算のデータ型変換

優先順位	いずれかのオペランドのデータ型	式のデータ型
1	DOUBLE、REAL	DOUBLE
2	DECIMAL、NUMERIC	DECIMAL、NUMERIC
3	BIGINT	BIGINT
4	他のすべての数値データ型	INTEGER

次のように仮定します。

- a は、値が 1.35 の DOUBLE 変数です。
- b は、値が 2 の INTEGER 変数です。
- c は、値が 3 の INTEGER 変数です。

式の評価方法は次のとおりです。

1 c は INTEGER 値 3 であるため、-c は INTEGER 値-3 に評価されます。

2 b は INTEGER 値 2 であり、-c は INTEGER 値-3 に評価されるため、式 b/-c は 2/-3 になり、INTEGER 値 0 に評価されます。

3 a は DOUBLE 値 1.35 であり、式 b/-c は INTEGER 値 0 に評価されるため、INTEGER 値 0 は DOUBLE 値 0.0 に変換されます。式 a + b/-c は 1.35 + 0.0 になり、これは DOUBLE 値 1.35 に評価されます。

4 式 a + b /-c が DOUBLE 値 1.35 に評価され、5 が INTEGER 値であるため、INTEGER 値 5 は DOUBLE 値 5.0 に変換されます。式 a + b/-c - 5 は 1.35 - 5.0 になり、これは DOUBLE 値-3.65 に評価されます。

算術演算の非数値オペランドは、DOUBLE に強制変換されます。CHAR + INT の式の型は DOUBLE です。これは、CHAR オペランドが DOUBLE に強制変換されるためです。DOUBLE + INT は、DOUBLE の優先順位が INTEGER よりも高いため、式のデータ型が DOUBLE になります。

データ型変換の詳細については、“DS2 型変換” (*SAS DS2 プログラマガイド*)を参照してください。

別の例として、a = b = 5 という式を考えてみましょう。最初の等号(=)は割り当て演算子です。2 番目の等号は論理等値演算子です。b が 5 以外の値の場合、b=5 は 0 と評価されます。したがって、a には値 0 が割り当てられます。詳細については、“例 2: 複数の等号を持つ式の使用” (1303 ページ)を参照してください。

DOT 演算子式

ドット演算子式は、パッケージの外部にあるコードからパッケージデータにアクセスします。

DOT 演算子式の構文は次のとおりです。

package-variable.[*package-attribute* | *package-method(arguments)*]

package-variable1.*package-variable2*....*package-variableN*.[*package-attribute* | *package-method(arguments)*]

. (dot)
パッケージ属性にアクセスするか、左側のオペランドによって参照されるパッケージ変数から、右側のオペランドとして指定されたパッケージメソッドを呼び出します。

package-attribute
左側のオペランドによって参照されるパッケージの属性を指定します。パッケージ属性は、パッケージ内で定義されるグローバル変数です。このグローバル変数は、パッケージのスカラー変数または配列変数にすることができます。ドット表記は、変数配列属性、標準一時配列属性、および動的一時配列属性へのアクセスをサポートしています。

package-variable
アクセスする属性またはメソッドを持つパッケージのインスタンス化を指定します。

注 これはスカラーパッケージ変数または配列パッケージ変数の要素にすることができます。詳細については、“DECLARE PACKAGE ステートメント”を参照してください。

package-method(arguments)
左側のオペランドによって参照されるパッケージ変数のメソッドを指定します。

ドット演算子の左側のオペランドは、スカラーパッケージ変数または配列パッケージ変数の要素である必要があります。左側のオペランドが null の場合(つまり、パッケージ変数がパッケージインスタンスを参照していない場合)、ドット演算により致命的な欄ライムエラーが発生します。

ドット演算子の右側のオペランドは、左側のオペランドによって参照されるパッケージの属性またはメソッドでなければなりません。ドット演算子の右側のオペランドがスカラーパッケージ変数または配列パッケージ変数の要素である属性である場合、ドット演算子の結果自体を後続のドット演算の左側のオペランドとして使用で

きます。パッケージ変数を左側のオペランドとして指定する式は、ネストされたドット表記式です。

ネストされたドット表記は、パッケージの階層内のパッケージデータへのアクセスをサポートします。ネストされたドット演算子の左側の各オペランドがパッケージ変数またはパッケージ配列変数の要素に解決される限り、無制限の数のネストされたドット演算がサポートされます。

詳細については、“DS2 パッケージ内のドット表記” (*SAS DS2 プログラマガイド*)を参照してください。

関数式

関数式は、DS2 プログラム内の関数を呼び出します。関数を呼び出すには、次の構文を使用します。

function-name ([*argument* [, ...*argument*]])

関数には引数が必要な場合があります。関数式に引数が含まれている場合、引数のデータ型は、必要に応じて、関数の引数の順序とデータ型である関数**シグネチャ**のデータ型に変換されます。関数が引数を取るか取らないかに関係なく、関数呼び出しにはかっこが必要です。たとえば、TIME 関数は引数を取りません。

```
t = time();
```

関数式は、関数によって返される値に解決されます。前述の関数式では、time()は現在時刻に解決されます。

メソッドと関数は似ています。関数はグローバルスコープを持ちます。メソッドはプログラミングブロックであり、ローカルスコープを持ちます。

関数の名前がメソッド名と同じ場合、DS2 はそのメソッドを呼び出します。メソッドと同じ名前の関数は、SYSTEM 式を使用してのみ呼び出すことができます。詳細については、“SYSTEM 式” (2115 ページ)を参照してください。

DS2 関数のリストについては、“DS2 関数” (235 ページ)を参照してください。

IF 式

IF 式の概要

条件付き IF 式は、条件式の評価が真(ゼロ以外の値)か偽(ゼロ)かに基づいて、2 つの値を選択するために使用されます。

IF 式を呼び出すには、次の構文を使用します。

IF *expression-1* **THEN** *expression-2* **ELSE** *expression-3*

expression-1 がゼロ以外の値の場合、IF 式の結果は *expression-2* の値になります。それ以外の場合、IF 式の結果は *expression-3* の値になります。次に例を示します。

```
m=(if missing(u) then 0 else u);
```

IF 式は、他の式を使用できる場所であればどこでも使用できます。IF 式の優先順位は、算術演算子および論理演算子よりも低くなります。したがって、このような混合式ではかっこが必要です。

```
r = 25.5 + (if sum < 15 then -a else b*2);
```

かっこがないと、プラス(+)演算子が最初に評価され、サブ式 25.5 + if から解析エラーが発生します。

ネストされた IF 式

IF 式をネストして、多方向の決定のために多くの値から選択することができます。

IF *condition-expression-1***THEN** *result-expression-1*

ELSE IF *condition-expression-2* **THEN** *result-expression-2*

...

ELSE IF *condition-expression-n* **THEN** *result-expression-n*

ELSE *result-expression-default*

条件式は順番に評価されます。ネストされた IF 式チェーンの結果は、真(ゼロ以外の値)と評価される最初の *condition-expression* に関連付けられた *result-expression* です。すべての *condition-expressions* が偽(ゼロ)と評価された場合、IF 式の結果は *result-expression-default* になります。次に例を示します。

```
grade = if score >= 90 then 'A'
  else if score >= 80 then 'B'
  else if score >= 70 then 'C'
  else if score >= 60 then 'D'
  else if score >= 0  then 'F'
  else NULL;
```

注: 相互に排他的な条件が長く連続する場合は、IF-THEN/ELSE ステートメントを連続で使用するかわりに SELECT ステートメントを使用します。ネストされた IF-THEN/ELSE ステートメントが多数あると、内部エラーが発生する可能性があります。対照的に、SELECT ステートメントは 1 回だけ評価されるため、エラーを発生させずにパフォーマンスを向上させることができます。

たとえば、次のプログラムのように、プログラムに多数の ELSE IF コンストラクトが含まれているとします。

```
if (e1) then ...
else if (e2) then ...
...
else if (en) then ...
else ...
```

かわりに、この SELECT ステートメントを使用します。

```
select;
when (e1): ...
when (e2): ...
...
when (en): ...
```

otherwise: ...

IF 式のデータ型

IF 式のデータ型は、最初の結果式である *expression-2* の型を調べることによって決定されます。

IF *expression-1* **THEN** *expression-2* **ELSE** *expression-3*

expression-2 が数値データ型でない場合、IF 式には *expression-2* の型が割り当てられます。

expression-2 が数値データ型の場合、IF 式には、*expression-2* と *expression-3* の広い方の数値データ型が割り当てられます。たとえば、*expression-2* が SMALLINT で、*expression-3* が DOUBLE の場合、IF 式には DOUBLE 型が割り当てられます。*expression-2* が数値データ型で、*expression-3* が数値データ型でない場合、IF 式には *expression-2* の型が割り当てられます。

ネストされた IF 式チェーンの最初の結果式が数値データ型の場合、すべての結果式が調べられ、ネストされた IF 式チェーンの型として割り当てる最も広い数値データ型が検索されます。次の例では、**t** は TINYINT、**b** は BIGNINT、**d** は DECIMAL(10,5) です。ELSE の 0 には BIGINT 型が割り当てられます。したがって、ネストされた IF 式チェーンには、TINYINT、BIGINT、および DECIMAL(10,5)の中で最も広い数値型である decimal(10,5)型が割り当てられます。

```
r = if n < 0 then t
  else if n = 0 then b
  else if n > 0 then d
  else 0;
```

注: ELSE 値に 0 ではなく 0.0 が使用された場合、ELSE の結果には、BIGINT 型ではなく DOUBLE 型が割り当てられます。DOUBLE 型の ELSE 式の場合、結果式の最も広い数値型は DOUBLE 型になります。したがって、ネストされた IF 式チェーンには、decimal(10,5)型ではなく、DOUBLE 型が割り当てられます。

IF 式の遅延評価

IF 式は、結果式に遅延評価を使用します。

IF *expression-1* **THEN** *expression-2* **ELSE** *expression-3*

たとえば、このコードを使用してゼロ除算をチェックできます。

```
a = if c ne 0 then b/c else null;
```

式 *expression-1* は常に評価されますが、*expression-2* または *expression-3* のいずれかのみが評価されます。IF 式の結果として選択されなかった式は評価されません。

したがって、*expression-1* がゼロ以外の値(真)の場合、*expression-2* のみが評価されます。*expression-1* がゼロ(偽)の場合、*expression-3* のみが評価されます。

遅延評価は、ネストされた IF 式チェーンの結果式にも適用されます。

IF *condition-expression-1* THEN *result-expression-1*
  ELSE IF *condition-expression-2* THEN *result-expression-2*
  ...
  ELSE IF *condition-expression-n* THEN *result-expression-2*
  ELSE *result-expression-default*

選択された結果式は、評価される唯一の結果式です。遅延評価は条件式にも適用されます。条件式は、条件が真(ゼロ以外)と評価されるか、すべての条件が評価されるまで、順番に評価されます。IF 式に *n* 個の条件式があり、*i* 番目の条件が最初の非ゼロ条件である場合、最初の 1 から *i* までの条件のみが評価されます。*i*+1 から *n* までの条件は評価されません。

IN 式

IN 式は、式が定数リストに含まれているかどうかを判別します。*"定数リスト" (SAS DS2 プログラマガイド)*を参照してください。

IN 式の構文は次のとおりです。

search-expression [*not-operator*] **IN** *constant-list*

検索式が定数リスト内のいずれかの定数と等しい場合、検索式は定数リストと一致します。NOT 演算子(またはそのエイリアス~および^)が IN 演算子の前に指定されている場合、IN 式の結果は、検索式が一致するかどうかの論理否定になります(論理 NOT 式の規則 (2122 ページ)を参照)。

IN 式の結果の型は常に INTEGER であり、その値は 1 (TRUE)、0 (FALSE)、または null (不明)のいずれかです。次の表は、検索一致がどのように決定されるかを示しています。

表 A21.3　IN 式の結果基準

TRUE	検索式は、定数リスト内の 1 つ以上の定数と等しいかを比較します。
FALSE	検索式は、定数リスト内のすべての定数と等しくないかを比較します。
UNKNOWN	前述のいずれも当てはまらず、少なくとも 1 つの同等性のテストが不明でした。通常、これは検索式が null の値に解決される場合に発生します。

検索式が定数リスト内の定数と等しいかどうかを判断する場合、DS2 は通常の等式(=)と同じ規則を使用します。比較されるいずれかの項目の値が不明(null 値で表される)の場合、比較の結果は不明です。

SAS モードに関する特別な考慮事項:

- データ型が CHAR または DOUBLE で、検索式が SAS 欠損値と等しい場合、定数リストの欠損値と一致します。

- 定数リストに null 値が指定されている場合、SAS 欠損値に変換されます。
- 定数リスト内の 1 つ以上の定数の型と一致するように検索式を DOUBLE に変換する必要があり、検索式の値が null である場合、null 値を DOUBLE に変換すると、SAS 欠損値が発生します。この場合、定数リストに欠損値が含まれていると、IN 式の結果は TRUE になります。

ANSI モードでは、定数リストに欠損値を指定しても意味がありません。ANSI モードでは、定数リストの欠損値値が null に変換され、不明な値として扱われるためです。

DS2 が null と SAS 欠損値を処理する方法の詳細については、“存在しないデータの DS2 モード” (*SAS DS2 プログラマガイド*)を参照してください。

次の表は、SAS モードと ANSI モードのどちらが有効かによって、さまざまなケースでの IN 式の処理を示しています。これらの例では、次の変数宣言を想定しています。

```
dcl char(10) my_char;
dcl double my_double;
dcl integer my_int;
```

表 A21.4　IN 式の例

モード	入力値	IN 式	結果	説明
両方	my_char = '3.14'	my_char in (3.14, 'def', 'ghi')	1	定数リストの 3.14 は文字列'3.14'に変換されます。
SAS	my_double = .;	my_double in (., 2, 3)	1	検索式の欠損値は、定数リストの欠損値と一致します。
SAS	my_double = .;	my_double in (1, 2, 3)	0	検索式の欠損値が非欠損値と一致しません。
ANSI	my_double = null;	my_double in (1, 2, 3)	null	検索式が不明の場合、定数リストに関係なく IN 式の値が不明です。
ANSI	my_int = -1;	my_int in (1, 2, 'not a number')	null	文字列'not a number'を BIGINT 型に変換しようとすると失敗し、null 値になります。一致するものがなく、不明な比較が少なくとも 1 つあるため、IN 式全体の値は不明です。
SAS	my_int = null;	my_int in (3, 3.14, .)	1	小数値を表すには、定数リストを DOUBLE にプロモートする必要があります。したがって、検索引数も DOUBLE に変換する必要があります。結果は欠損値で

モード	入力値	IN 式	結果	説明
				あり、リスト内の定数と一致します。

LIKE 式

LIKE 式の概要

LIKE 式は、文字列がパターンマッチングの指定と一致するかどうかを判別します。

LIKE 式の構文は次のとおりです。

expression [NOT] **LIKE** *pattern-matching-expression* [ESCAPE *character-expression*]

式は、任意の文字列またはバイナリ文字列のデータ型にすることができます。

expression が *pattern-matching-expression* で指定されたパターンと一致する場合、値 1 (真)が返されます。それ以外の場合は、値 0 (偽)が返されます。

NOT LIKE は、LIKE の逆の値を返します。たとえば、x like y が真の場合、x not like y は偽です。

ESCAPE 引数は、通常パターンマッチングに使用されるパーセント(%)文字とアンダースコア(_)文字のリテラルインスタンスを検索するために使用されます。

検索パターン

パターンは、3 つのクラスの文字で構成されます。

表 A21.5　パターンマッチング文字

文字	説明
アンダースコア(_)	任意の 1 文字に一致
パーセント記号(%)	0 個以上の文字の任意のシーケンスに一致 **注:** 末尾の空白の影響に注意してください。値を一致させるには、TRIM 関数を使用して末尾の空白を削除する必要がある場合があります。
その他の文字	その文字に一致

リテラル%および_の検索

%および_文字は LIKE 式のコンテキストで特別な意味を持つため、入力文字列でこれらの文字リテラルを検索するには、ESCAPE 引数を使用する必要があります。

これらの例では、値 **app**、**a_%**、**a__**、**bbaa1**、**ba_1** を使用しています。

- 条件 like 'a_%'は、**app**、**a_%**、および **a__**に一致します。検索パターンのアンダースコア(_)は、任意の 1 文字(アンダースコアを含む)と一致し、検索パターンのパーセント(%)は、'%'と'_'を含む 0 個以上の文字と一致します。
- 条件 like 'a_^%' escape '^'は、**a_%**のみに一致します。エスケープ文字(^)は、パターンがリテラル'%'を検索することを指定します。
- 条件 like 'a_%' escape '_'は、どの値とも一致しません。エスケープ文字(_)は、パターンが'a'の後に続くリテラル'%'を検索することを指定します。これは、これらの値のいずれにも適用されません。

大文字と小文字が混在する文字列の検索

DS2 LIKE 式では大文字と小文字が区別されます。大文字と小文字が混在する文字列を検索するには、次の例に示すように UPCASE 関数を使用します。

```
upcase(str) like 'SM%'
```

LIKE 式の例

次の表は、Smith, Smooth, Smothers, Smart, Smuggle という文字列を検索したときに一致する結果の例を示しています。

表 A21.6　LIKE 式の例

LIKE 式の例	マッチング結果
str like 'Sm%'	Smith, Smooth, Smothers, Smart, Smuggle
str like '%th'	Smith, Smooth
str LIKE 'S__gg%'	Smuggle
str like 'S_o'	(一致なし)
str like 'S_o%'	Smooth, Smothers
str like 'S%th'	Smith, Smooth

LIKE 式の例	マッチング結果
str not like 'Z'	Smith, Smooth, Smothers, Smart, Smuggle

メソッド式

メソッド式は、METHOD ステートメントによって定義されたメソッドを呼び出します。

メソッドを呼び出すには、次の構文を使用します。

method-name ([*expression*[, ... *expression*]])

メソッドは、メソッド名とシグネチャに基づいて呼び出されます。DS2 は、最初にメソッド名を識別します。メソッド名が関数名と同じ場合、DS2 はそのメソッドを呼び出します。メソッドと同じ名前の関数は、SYSTEM 式を使用して呼び出すことができます。詳細については、“SYSTEM 式” (2115 ページ)を参照してください。

DS2 ではオーバーロードされたメソッドが許可されているため、DS2 は、メソッドシグネチャの引数の数と引数のデータ型(メソッドの引数の順序とデータ型)が最も一致する引数を持つメソッドを呼び出します。最適な一致とは、メソッドパラメーターの数が引数の数と等しく、指定された引数リストに対して正確なパラメーターの型がこれほど多く一致するメソッドシグネチャが他にないようなものです。最適な一致が見つからない場合、エラーが発生します。

> ヒント　DS2 メソッドは、最大 1000 個の引数をサポートできます。1000 を超える引数を持つ DS2 メソッドは、コンパイルエラーを生成する可能性があります。

実行するメソッドが識別されると、DS2 は必要に応じて、引数のデータ型を対応するメソッドパラメーターのデータ型に変換します。その後、メソッドが実行されます。

メソッド式は、メソッドによって返される値に解決されます。

次の例では、メソッド CONCAT および ADD が定義され、INIT()メソッドで呼び出されます。INIT()メソッドで強調表示されている式はメソッド式です。

```
method concat(char(100) x, char(100) y) returns char(200);
  return trim(x)||y;
end;

method add(double x, double y) returns double;
  return x + y;
end;

method init();
  dcl char(200) r;
  r = concat('abc', 'def');
  d = add(100,101);
```

```
end;
```

次の例では、D メソッドがオーバーロードされたメソッドです。DS2 は、メソッド式の引数をメソッドシグネチャと比較して、最適な一致を見つける必要があります。

```
method d(double x, double y) returns double;
  return x + y;
end;

method d(int x, int y) returns int;
  return x + y;
end;

method init();
  dcl double r;
  dcl int i;
  r = d(1.2345, 5.6789);
  i = d(99, 100);
end;
```

最初のメソッドは、シグネチャが DOUBLE データ型の値を必要とする D メソッドを呼び出します。2 番目のメソッドは、シグネチャが INTEGER データ型の値を必要とする D メソッドを呼び出します。

この最後の例は、DS2 がメソッド式の値のデータ型が INTEGER か DOUBLE かを判断できないことを示しています。あいまいであるため、DS2 はエラーを発行します。

```
method d(int x, double y) returns double;
  return x + y;
end;

method d(double x, int y) returns double;
  return x + y;
end;

method run();
  d = d(100, 102);
end;
```

詳細については、“METHOD ステートメント” (1376 ページ)を参照してください。

OF 演算子

概要と基本構文

OF 演算子を使用すると、変数リストと変数配列を関数呼び出しの引数として指定できます。変数グループを引数として関数にサブミットすることは、複合式における OF 演算子の唯一の目的です。

OF 演算子の構文は次のとおりです。

OF *variable-array*[*] | *variable-list*

variable-array[*]は、同種データ型のグローバル変数をグループ化したものです。添字演算子のアスタリスクは必須であり、変数配列内のすべての 要素が関数に渡されることを指定します。

variable-list は、省略形に一致する 1 つ以上のグローバル変数を選択する省略形の仕様です。DS2 は、いくつかのタイプの省略変数リストをサポートしています。詳細については、"省略変数リストの種類" (*SAS DS2 プログラマガイド*)を参照してください。

OF 演算子は、変数配列または変数リスト内の各要素を変数名のカンマ区切りリストに変換し、これが関数呼び出しの引数になります。次に例を示します。

- OF 式 OF x1-x5 は引数リスト x1, x2, x3, x4, x5 に変換されます。
- VARARRAY double a[6] u v w x y z;として定義されたか、OF 演算子によって OF a[*] として参照された配列は、引数リスト: u, v, w, x, y, z に変換されます。

OF 演算子は、関数呼び出しで他の引数とインターリーブできます。たとえば、関数呼び出し

```
MAX(v1, v2, OF x1-x5, OF a[*], v3)
```

は次の変数のリストに変換されます。

```
MAX(v1, v2, x1, x2, x3, x4, x5, u, v, w, x, y, z, v3)
```

関数呼び出しの引数には、任意のタイプの式を使用できます。個々の引数は、OF 演算、添字演算、またはより複雑な式にすることができます。たとえば、次の関数呼び出しは有効です。

```
MAX(OF a[*], a[1], y[3], a[1] + y[3] * 100)
```

この関数呼び出しの引数は、次を含むように展開されます。

- 変数配列 a のすべての 要素。
- 配列 a の要素 1 の値。式の中の添字 1 は、グループ内の変数へのインデックスです。
- y という名前の配列の要素 3 の値。
- 算術式 a[1] + y[3] * 100 の結果。

OF 演算子の制限

OF *variable-array*[*]構文の *variable-array* オペランドは、グローバル変数配列である必要があります。*variable-array* オペランドは、一時配列、varlist、またはメソッドのパラメーターである 変数配列にすることはできません。

OF 演算子に関するその他の制限は次のとおりです。

- DS2 関数の呼び出しでのみ使用できます(ユーザー定義メソッドまたはパッケージメソッドの呼び出しでは使用できません)。
- さまざまな数のパラメーターを受け取る関数、またはパラメーターの数が OF リストの要素の数と一致する関数でのみ使用できます。関数が固定数の引数を受け取る場合、すべての OF リストが展開された後の引数の合計数は、関数が受け取る引数の数と一致する必要があります。

- WHERE 句で指定された関数では使用できません。
- DIF 関数、LAG 関数とは併用できません。
- OF *variable-array*[*]構文は配列インデックスとして使用できません。添字演算を OF 演算子のオペランドにすることはできません。添字演算の前に OF 演算子がある場合、DS2 は次のエラーを返します。Parse encountered constant when expecting '*'. これはサポートされていないものの例です。

  ```
  OF a[1]
  OF v[3]
  ```

  この例では、a は変数配列の要素です。つまり、添字の 1 は、グループ内の変数へのインデックスです。v は、varlist です。添字の 3 は、グループ内の変数へのインデックスです。

  注: 変数情報カテゴリの関数のみが、varlist の要素を引数として受け取ることができます。

関数呼び出し内の OF 演算子の位置

関数呼び出しにおける OF 演算子の位置は柔軟です。引数リストでは、各変数配列と変数リストの前に OF 演算子を置くことができます。しかし、引数リスト内のすべての変数配列と変数リストを展開するには、単一の OF 演算子で十分です。

OF を 1 回指定し、それが引数リストの最初の引数である場合、引数リスト内のほとんどのグローバルスカラー変数はカンマではなくスペースで区切ることができます。たとえば、次の SUM 関数呼び出しと MAX 関数呼び出しの構文は両方とも有効です。

```
SUM (OF x -- z, OF a -- c)
SUM (OF x -- z a -- c)

MAX(z, OF a[*], y)
MAX(OF z a[*] y)
```

ただし、ローカル変数である関数引数と添字演算は、カンマを使用して OF 演算とは区別して識別する必要があります。カンマが必要な場合を示す関数呼び出しの例を次に示します。

```
proc ds2;
data _null_;
 dcl int i1 i2 i j k;
 dcl double d1 d2 d3 x y z;
 vararray double v[3] d1 d2 d3;
 varlist a [x y z i j k];
 method init();
   VSETMISSING (OF z y i:, v[1], a[3]);
 end;
enddata;
run;
quit;
```

この例では、OF 演算には、z、y、i:の 3 つのオペランドがあります。OF 演算は引数リスト(z、y、i1、i2、i)に展開されます。次に、VSETMISSING 関数が 7 つの引数(z、y、 i1、i2、i、v[1]、a[3])で呼び出されます。変数 d1 は変数配列 v の第 1 要素であり、変数 z は配列 a の第 3 要素であるため、VSETMISSING は変数 z、y、i1、i2、i、d1、z の値を missing に設定します。添字演算は OF 演算子のオペランドではないため、他の引数がスペースで区切られている場合でも、添字演算は OF オペレーター、他の引数、互いからカンマを使用して区切る必要があります。カンマは変数展開を停止します。

注: 一部の関数では、引数の順序を変更すると関数呼び出しの動作が変わることがあります。

型変数リストと OF 演算子

型変数リストは OF 演算子でサポートされています。

```
(OF double)
```

OF *variable-array*[*]構文は、型変数リスト指定を持つ変数配列もサポートします。型変数リスト指定を持つ変数配列を次に示します。この指定では、型 INT の変数を選択します。

```
vararray int x[*] int;
```

さらに、OF *variable-array*[*]構文は、型変数リスト指定を含む別の配列の範囲の DIM に基づいた上限を持つ変数配列をサポートします。型変数リスト指定を含む別の配列の範囲の DIM に基づいた上限を持つ変数配列を次に示します。

```
vararray int x[*] int;
vararray int  y[dim(x)];
```

宣言されていない変数と任意の形式の型変数リストがある場合、OF 演算子の使用はサポートされません。OF 演算子を成功させるには、DS2 プログラム内のすべての変数を型変数リストで宣言する必要があります。宣言されていない変数が OF 演算で参照されない場合でも、宣言されていない変数が存在すると、型変数リストで OF 演算を使用できなくなります。

許可されていない変数リスト要求のログ出力を次に示します。

```
20       proc ds2;
21         data _null_;
22           method init();
23             dcl double total;
24             x = 10.0; /* undeclared variable */
25             y = 20.0; /* undeclared variable */
26             total = sum(OF double);
27           end;
28         enddata;
29       run;
ERROR: Compilation error.
ERROR: Line 26: OF operator with undeclared variables and type
variable list specification is not supported.
NOTE: PROC DS2 has set option NOEXEC and will continue to prepare
statements.
```

許可されていない変数配列要求のログ出力を次に示します。

```
1       proc ds2;
2        data _null_;
3        vararray double x[*] double;
4
5        method init();
6        a = 1.0;
7        b = 2.0;
8
9        VSETMISSING(OF x[*]);
10         end;
11         enddata;
12         run;
ERROR: Compilation error.
ERROR: Line 9: OF operator with undeclared variables and type variable list
specification is not supported.
```

DS2 と DATA ステップの演算子の違い

DS2 OF 演算子の機能は、DATA ステップ演算子の機能と似ています。ただし、DS2 OF 演算子と DATA ステップの OF 演算子で異なる結果が出る場合があります。これは、変数リストが各言語でどのように処理されるかによるものです。DATA ステップは、関数呼び出しの時点までの変数のみを取得します。DS2 は、プログラム全体で一致するすべての変数を取得します。DATA ステッププログラムの SUM 関数が変数 A1 と A4 のみを使用する例を次に示します。DS2 プログラムの SUM 関数は、変数 A1、A4、および A2 を使用します。

```
/* DATA step program */
/* sums A1 and A4 with a result of 4 */
data _null_;
A1 = 1.0;
A4 = 3.0;
y = 1;
lab:
if (y = 2) then
X = SUM(OF A: );
A2 = 2.0;
if (y = 1) then do;
y = 2;
goto lab;
end;
put X=;
;
run;

/* DS2 program */
/* sums A1, A4, and A2 with a result of 6*/
proc ds2;
data X (overwrite=yes);
dcl double x A1 A2 A4 y;
method run();
A1 = 1.0;
```

```
A4 = 3.0;
y = 1;
lab:
if (y = 2) then
X = SUM(OF A: );
A2 = 2.0;
if (y = 1) then do;
y = 2;
goto lab;
end;
put X=;
end;
enddata;
run;
quit;
```

パッケージメソッド式

パッケージメソッド式は、パッケージで定義されているメソッドをインスタンス化します。

パッケージメソッド式を呼び出すには、次の構文を使用します。

package-variable.method(*method-arguments*)

パッケージ変数はスカラー変数または配列パッケージ変数の要素にすることができます。配列パッケージ変数は、複数のパッケージインスタンスをグループ化できる一時配列であり、各パッケージインスタンスは配列要素としてアクセスされます。詳細については、DECLARE PACKAGE ステートメントを参照してください。配列パッケージ変数のパッケージインスタンスは、次のようにパッケージメソッド式で参照されます。

package-array\[*index*\].*method*(*method-arguments*)

注: 構文規則の角かっこは、オプションの引数を示します。角かっこの前のエスケープ文字(\)は、構文に角かっこが必要であることを示します。配列の境界は、角かっこ([ ])で囲む必要があります。

パッケージメソッド式は、メソッド式と同様の方法で実行されます。つまり、DS2 がパッケージとメソッドが存在すると判断した後、メソッドシグネチャの最適な一致が判断され、必要に応じて引数のデータ型が変換され、メソッドが実行されます。

> ヒント　DS2 メソッドは、最大 1000 個の引数をサポートできます。1000 を超える引数を持つ DS2 メソッドは、コンパイルエラーを生成する可能性があります。

次の例では、強調表示された式は、スカラーパッケージ変数をパッケージ変数として使用するパッケージメソッド式です。

```
declare package myadd a1() a2();
```

```
a1.sale(3,4);
a1.add(1,2);
a2.bonus(5,12);
a2.add(10,20);
```

DECLARE PACKAGE ステートメントは、a1 および a2 という名前の 2 つのパッケージインスタンスを宣言し、インスタンス化します。最初の 2 つのパッケージメソッド式は、MYADD パッケージからインスタンス化された A1 パッケージの SALE メソッドと ADD メソッドを呼び出します。最後の 2 つのパッケージメソッド式は、A2 パッケージの BONUS メソッドと ADD メソッドを呼び出します。

配列パッケージ変数の使用例については、"パッケージ配列の宣言と使用"を参照してください。

注: DS2 パッケージメソッド式を FedSQL SELECT ステートメントの関数として呼び出すことができます。詳細については、"式での DS2 パッケージの使用" (*SAS FedSQL 言語リファレンス*)を参照してください。

パッケージの詳細については、"PACKAGE ステートメント" (1392 ページ)および "DS2 パッケージ" (*SAS DS2 プログラマガイド*)を参照してください。

SYSTEM 式

メソッドと関数が同じ名前の場合、メソッド呼び出しが関数呼び出しよりも優先されます。関数は、SYSTEM 式を使用してのみ呼び出すことができます。

SYSTEM 式を呼び出すには、次の構文を使用します。

SYSTEM.*function-expression*

SYSTEM 式は、関数式の先頭にドット表記 system.を追加します。たとえば、SUM が関数の名前であると同時にメソッドの名前でもある場合、SUM 関数は、SYSTEM 式 system.sum(a,b,c)を使用してのみ呼び出すことができます。

THIS 式

THIS 式は、DS2 プログラム内の任意の場所からグローバルスカラー変数を同時に宣言して使用する代替方法を提供します。THIS 式は、標準の変数ルックアップを回避するために使用されます。THIS 式では、DS2 はグローバルスコープで識別子のスカラー変数宣言を検索します。そのような宣言がない場合、DS2 はグローバルスコープで識別子を DOUBLE 型で宣言します。グローバル変数は、DS2 プログラム内のすべてのプログラミングブロックで参照できます。

THIS 式を呼び出すには、次の構文を使用します。

THIS.*variable-name*

THIS 式は、変数の先頭にドット表記 THIS.を追加します。

注: DS2 は、THIS.*variable-name* を *variable-name* としてのみ保存します。グローバルスカラー変数と同じ名前のローカル変数があり、DS2 が変数に関する診断メッセージを発行する場合、どの変数が問題であるかを区別できません。たとえば、DS2 が **x** が宣言されていないという警告メッセージを発行した場合、そのメッセージがグローバル変数 **THIS.x** を参照しているのか、ローカル変数 **x** を参照しているのかわかりません。

次の例では、変数 s は、THIS 式を使用してグローバル変数になります。

```
method init();
 this.s = sum(a,b,c,d,e);
end;
method run();
  t = put(this.s 5.4);
end;
```

THIS 式は、同じ名前のローカル変数によって隠されているグローバル変数にアクセスする方法を提供します。次に例を示します。

```
proc ds2;
data;
  declare double x;    /* declare global x */

  method run();
    declare double x;  /* declare local x */

    /* Two variables exist with same name, "x". */
    /* Identifier "x" refers to local x in      */
    /* scope of run method.  Global x is hidden */
    /* by local x.                              */

    this.x = 1.0;    /* assign 1.0 to global x   */
    x = 0.0;         /* assign 0.0 to local x    */

    output;          /* global x with 1.0 output */
  end;
enddata;
run;
quit;
```

SELECT 式

SELECT 式の概要

SELECT 式は、他の式の値に基づいて複数の式から選択するために使用されます。

SELECT 式を呼び出すには、次の構文を使用します。

SELECT [(*select-expression*)]

WHEN (*when-expression*) [...WHEN (*when-expression*)] *result-expression*

[...WHEN (*when-expression*) [...WHEN (*when-expression*)] *result-expression*]

OTHERWISE [*default-result-expression*]

END

SELECT 式は、一致する式が見つかるまで、各 WHEN 式を順番に評価します。次に、関連付けられた *result-expression* が SELECT 式の結果として評価されます。

SELECT 式は、他の式を使用できる場所であればどこでも使用できます。次に例を示します。

```
r = 25.5 + select (t) when (1) -a when (3) b*2 end;
```

選択式を含む SELECT 式

選択式が存在する場合は、それが評価されます。次に、WHEN 式が順番に評価されます。SELECT 式の結果は、選択式と同じ値に評価される最初の WHEN 式の結果式です。すべての WHEN 式が選択式とは異なる値に評価される場合、SELECT 式の結果は、存在する場合のデフォルトの結果式です。それ以外の場合は、欠損値または null 値です。次に例を示します。

```
s = select (t)
  when (1) x*10
  when (3) x
  when (5) x*100
  when (0) 0
  otherwise .
end;
```

t が 5 の場合、SELECT 式は'x*100'に評価されます。

選択式のない SELECT 式

選択式が存在しない場合は、WHEN 式が順番に評価されます。SELECT 式の結果は、真(ゼロ以外の値)と評価される最初の WHEN 式の結果式です。すべての WHEN 式が偽(ゼロ)と評価された場合、選択式の結果は、存在する場合のデフォルトの結果式です。それ以外の場合は、欠損値または null 値です。次に例を示します。

```
grade = select
  when (score >= 90) 'A'
  when (score >= 80) 'B'
  when (score >= 70) 'C'
  when (score >= 60) 'D'
  when (score >= 0 ) 'F'
end;
```

score が 76 の場合、真と評価される最初の *when-expression* は **score>=70** です。*select-expression* は'C'に評価されます。

オプションのそれ以外の式

それ以外のデフォルトの結果式が提供されない場合、DS2 は、WHEN 式が選択されていないときに選択するデフォルトの結果値を提供します。SAS モードで SELECT 式の型が DOUBLE または CHAR の場合、デフォルトの結果値は欠損値(.)です。どちらのモードでも、他のすべてのデータ型の場合、デフォルトの結果値は NULL です。

複数の When 式を使用した結果式

複数の WHEN 式を 1 つの結果式に関連付けることができます。WHEN 式が連続してリストされ、その後に 1 つの結果式が続きます。結果式に関連付けられた WHEN 式のいずれかが最初に一致する WHEN 式である場合、SELECT 式の結果は結果式です。

たとえば、次の SELECT 式は、変数 **t** の値が'A'、'a'、'P'、'p'のいずれかである場合、'airplane'と評価されます。

```
s = select (t)
  when ('A')
  when ('a')
  when ('P')
  when ('p') 'airplane'
  when ('C')
  when ('c') 'car'
  when ('T')
  when ('t') 'train'
  otherwise  'walk'
 end;
```

SELECT 式のデータ型

SELECT 式のデータ型は最初の結果式の型を調べることによって決定されます。最初の結果式が数値データ型でない場合、SELECT 式には最初の結果式の型が割り当てられます。

最初の結果式が数値データ型の場合、すべての結果式が調べられ、SELECT 式の型として割り当てる最も広い数値データ型が検索されます。

次の例では、**t** は TINYINT、**b** は BIGINT、**d** は DECIMAL(10,5)、**s** は CHAR(10)です。選択式には、TINYINT、BIGINT、DECIMAL(10,5)、CHAR(10)の中で最も広い数値型である DECIMAL(10,5)型が割り当てられます。

```
r = select (t)
  when (1) t
  when (2) b
  when (3) d
  otherwise s
end;
```

注: **s** が最初の結果式に割り当てられていた場合、SELECT 式の型は CHAR(10)になります。最初の結果式が非数値型の場合、非数値型が SELECT 式の型として割り当てられます。

SELECT 式の遅延評価

SELECT 式は、WHEN 式に遅延評価を使用します。WHEN 式は、一致する WHEN 式が見つかるまで、またはすべての式が評価されるまで順番に評価されます。SELECT 式に *n* 個の WHEN 式があり、*i* 番目の WHEN 式が選択されている場合、最初の 1 から *i* までの WHEN 式のみが評価されます。*i*+1 から n までの when 式は評価されません。

遅延評価は、SELECT 式の結果式にも適用されます。選択された結果式は、評価される唯一の結果式です。

次の例では、n[i]が 0 に等しい場合、最初の 2 つの WHEN 式(n[i] < 0 と n[i] = 0)のみが評価され、2 番目の結果式(y*10-r2)のみが評価されます。

```
m(select
   when (n[i] < 0) y*100-r1
   when (n[i] = 0) y*10-r2
   when (n[i] > 0) y*10
 end);
```

添字演算子

添字演算子を使用すると、DS2 配列または varlist の要素にアクセスできます。

添字演算の構文は、配列または varlist 識別子と、それに続く配列または varlist 内の各次元のインデックス式で構成されます。構文の形式は次のとおりです。

identifier \[*index-expression*[,...*index-expression*] \]

注: 構文規則の角かっこは、オプションの構文を示します。角かっこの前のエスケープ文字(\)は、構文に角かっこが必要であることを示します。添字演算子のインデックスは、角かっこ([ ])で囲む必要があります。

index-expression のデータ型は INTEGER です。指定された *index-expression* が整数でない場合、INTEGER に強制変換されます。

デフォルトでは、インデックスは 1 ベースです。varlist の下限は 1 です。DECLARE または VARARRAY ステートメントで下限が指定されていない場合、配列の下限は 1 です。ただし、配列には別の下限を指定できます。たとえば、次の配列の下限は-10、上限は 5 です。したがって、有効なインデックスの範囲は-10 から 5 です。

```
declare double ta[-10:5];
vararray double va[-10:5];
```

添字演算子の例は次のとおりです。

```
a[i]
s[j * 2, k-3]
this.c[2, vwind, a[i]]
```

各添字演算は、配列または varlist の 1 つの要素にアクセスします。多次元配列の場合、添字演算では、配列の範囲ごとに *index-expression* をカンマで区切って指定する必要があります。たとえば、次のログ出力を参照してください。

```
80   proc ds2;
81   data _null_;
82     method init();
83       dcl double x[2,2];
84       dcl int i j;
85       x := (100 200 300 400);
86       do i = 1 to dim(x, 1);
87         do j = 1 to dim(x, 1);
88           put x[i,j]=;
89         end;
90       end;
91     end;
92   enddata;
93   run;
93 !      quit;
x[1,1]=100
x[1,2]=200
x[2,1]=300
x[2,2]=400
```

インデックス式が配列または varlist の範囲制限を超えている場合、DS2 はエラーを発行します。

配列を使用した添字演算は、配列のデータ型のスカラー変数を受け入れる任意の式で使用できます。varlist を使用した添字演算は、varlist の要素を受け入れる任意の式で使用できます。現在、これは V*変数情報関数に限定されています。

式の演算子

複合式での演算子の優先順位

演算子は、加算、比較、論理否定など、オペランドに対して実行される演算の種類を表します。式に複数の演算子とオペランドが含まれている場合、DS2 は演算子の優先順位を使用して式を解決します。演算は、優先順位の高い順に実行されます。

最高の優先順位は 1 で、最低の優先順位は 9 です。優先レベル内では、べき乗、最小、および最大演算子を除いて、演算子は左から右に関連付けられます。べき乗、最小、および最大演算子は、右から左に関連付けられます。

表 A2.7 (2121 ページ)の優先順位を使用すると、式 5+a**b*3 では、a**b が最初に計算され、次に 3 が乗算され、その結果が 5 に加算されます。

> ヒント　DS2 では、x < y < z は x < y および y < z ように評価されます。

次の表に、演算子とその優先順位を示します。

表 A21.7　DS2 複合式における演算子とその優先順位

優先順位	演算子名	記号	結合
1	かっこ	( )	左から右へ
1	ドット(.)	.	左から右へ
1	添字	*identifier*[*index*]	左から右へ
1		SELECT 式	左から右へ
2		単項+および–	右から左へ
2	論理 NOT	NOT または^または~	左から右へ
2	べき乗、最大、最小	**、<>、><	左から右へ
3	乗算、除算	*、/	左から右へ
4	プラス、マイナス	+、–	左から右へ
5	連結	||または!!	左から右へ
5	削除	..	左から右へ
6		IN、LIKE	左から右へ
7	等値または非等値	=または eq、^=または ~=	右から左へ
7	より大(または小)か等しい、より大(または小)	>=、<=、>、<[1]	左から右へ
8	論理 AND	AND または&	左から右へ
9	論理 OR	OR、|、または!	左から右へ
10		IF 式	右から左へ

優先順位	演算子名	記号	結合
なし	変数割り当て	=	なし
なし	配列割り当て	:=	なし
なし	名前のない Varlist	[*list-of-variable-names*]	なし
なし		_NEW_	なし
なし		OF	なし
なし		メソッド式	なし

1 DS2 では、x < y < z は x < y および y < z ように評価されます。

演算子別の式の値

次の表は、演算子に基づく式の解決値を示しています。

表 A21.8 演算子別の式の値

演算子	値
論理式 [1]	
NOT または^または~	オペランドがゼロ以外の場合、結果は 0 になります。オペランドがゼロまたは欠損している場合、結果は 1 になります。オペランドが null の場合、結果は null になります。
OR または|または!	■ 2 つのオペランドの論理 OR ■ 一方のオペランドがゼロまたは欠損しており、もう一方のオペランドが null の場合は null ■ 両方のオペランドが null の場合は null ■ いずれかのオペランドがゼロ以外の場合は(もう一方のオペランドが null または欠損している場合でも) 1 ■ 両方のオペランドがゼロまたは欠損している場合は 0
AND または&	■ 2 つのオペランドの論理 AND ■ 一方のオペランドがゼロ以外で、もう一方のオペランドが null の場合は null ■ 両方のオペランドが null の場合は null ■ 両方のオペランドがゼロ以外の場合は 1

演算子	値
	■ いずれかのオペランドがゼロまたは欠損している場合は (もう一方のオペランドが null であっても) 0
算術式	
単項プラス	式オペランドと同じ値を持つ
単項マイナス	オペランドの算術否定
+	オペランドの算術和
–	オペランドの算術差
*	オペランドの算術積
/	オペランドの算術商
**	左のオペランドを右のオペランドでべき乗したもの
><	左右のオペランドの最小値
<>	左右のオペランドの最大値
任意の算術演算子	いずれかまたは両方のオペランドが null の場合は null
関係式	
<	左のオペランドが右のオペランドより小さい場合は 1、それ以外の場合は 0
>	左のオペランドが右のオペランドより大きい場合は 1、それ以外の場合は 0
<=	左のオペランドが右のオペランド以下の場合は 1、それ以外の場合は 0
>=	左のオペランドが右のオペランド以上の場合は 1、それ以外の場合は 0
=または eq	左のオペランドが右のオペランドと等しい場合は 1、それ以外の場合は 0
^=	左のオペランドが右のオペランドと等しくない場合は 1、それ以外の場合は 0
任意の論理演算子	いずれかまたは両方の演算子が null の場合は null

演算子	値
連結式	
|| または!![2]	左のオペランドが右のオペランドとマージされて新しい値を形成する
..	連結する前に各引数を削除する
IF 式	
IF	論理式がゼロでも null でもない場合は THEN 式の値、それ以外の場合は ELSE 式の値。詳細については、“IF 式” (2101 ページ)を参照してください。
IN 式	
IN	定数リストに比較式が含まれる場合は 1。詳細については、“IN 式” (2104 ページ)を参照してください。
LIKE 式	
LIKE	式が指定されたパターンに一致する場合は 1、それ以外の場合は 0 (ゼロ)。詳細については、“LIKE 式” (2106 ページ)を参照してください。
SELECT 式	
SELECT	SELECT 式と一致する最初の WHEN 式の結果式の値。詳細については、“SELECT 式” (2116 ページ)を参照してください。
その他の式	
.	パッケージ属性にアクセスするか、パッケージの外部のコードからパッケージメソッドを呼び出します。詳細については、“DOT 演算子式” (2100 ページ)を参照してください。
=	値または式を変数に割り当てます。
:=	配列に値を割り当てます。
identifier[*index-expression*]	配列または varlist の要素にアクセスします。
[*list-of-variable-names*]	名前のない varlist を作成します。
NEW	パッケージのインスタンスを構築します。詳細については、“_NEW_ 演算子” (1289 ページ)を参照してください。

演算子	値
OF	変数リストまたは変数配列を関数の引数のリストに変換します。
メソッド式	システムメソッドまたはユーザー定義メソッドを呼び出します。詳細については、“DS2 メソッドについて” (52 ページ)を参照してください。

1 DS2 は、TRUE、FALSE、UNKNOWN の 3 つの真理値をサポートしています。ゼロ以外のオペランドは TRUE にマッピングされます。ゼロまたは欠損オペランドは FALSE にマッピングされます。null オペランドは UNKNOWN にマッピングされます。

2 ||連結演算子はスペースを削除しません。TRIM 関数を使用して、末尾のスペースを削除できます。ただし、..演算子は、TRIM とそれに続く連結と同じ機能を実行します。TRIM 関数 (TRIM(a) || TRIM(b))を使用するよりも、..演算子(a .. b)を使用する方が高速です。

DS2 での短絡評価

場合によっては、両方のオペランドを評価せずに論理 AND または OR 式の値を決定できます。たとえば、オペランド A が偽の場合、式(A AND B)の値は、オペランド B の値に関係なく偽になります。最初のオペランドのみに基づいて論理式全体の値を決定できる場合、プログラミング言語は評価を"短絡"し、2 番目のオペランドを評価するために必要な計算処理を回避できます。

オプション LOGICALEXPR=OPTIMIZED を設定することにより、DS2 で短絡評価を使用できます。LOGICALEXPR=オプションは、DS2_OPTIONS ステートメントの引数、DS2 プロシジャオプション、ならびに DS2 runDS2 および runModel アクションのパラメーターとして使用できます。

DS2 は、論理 AND および OR 式のオペランドを左から右に評価します。LOGICALEXPR=OPTIMIZED が有効な場合、DS2 は論理演算子を次のように評価します。

- AND の場合、最初のオペランドがゼロまたは欠損している場合、2 番目のオペランドは評価されず、式全体が false になります。最初のオペランドがゼロ以外または null の場合、DS2 は 2 番目のオペランドを評価する必要があります。
- OR の場合、最初のオペランドがゼロ以外の場合、2 番目のオペランドは評価されず、式全体が TRUE になります。最初のオペランドがゼロ、欠損、または null の場合、DS2 は 2 番目のオペランドを評価する必要があります。

LOGICALEXPR=OPTIMIZED を使用すると、数学的に無効な計算の実行を回避できます。次のプログラムを考えてみましょう。プログラムでは、式 1/sqrt(a)は、a が非負(平方根による)で、sqrt(a)が非ゼロ(除算による)である場合にのみ計算できます。LOGICALEXPR=OPTIMIZED が設定されている場合、AND 式の 2 番目のオペランドが評価されないため、プログラムはエラーなしで完了します。

```
proc ds2 errorstop;
  ds2_options logicalexpr=optimized;
  data _null_;
    declare double a;
    method init();
      a=0;
```

```
        if (a>0 AND 1/sqrt(a) < .5) then
          put 'criteria satisfied';
        else
          put 'unacceptable value';
        put 'execution continues ...';
      end;
    enddata;
  run; quit;
```

次のような行がログに書き込まれます。

```
unacceptable value
execution continues ...
NOTE: Execution succeeded.No rows affected.
```

LOGICALEXPR=OPTIMIZED を使用して、計算コストの高い計算の実行を回避することもできます。次のコードスニペットを考えてみましょう。ユーザー定義パッケージ credit_eval の score メソッドには、複雑なロジックが含まれています。このビジネスケースでは、顧客がすでにローンの債務不履行に陥っている場合、そのメソッドの呼び出しは避ける必要があります。

```
dcl package credit_eval ce;
reject_loan_request = (has_defaulted OR ce.score() < 500);
...
```

AND 演算子と OR 演算子、およびそれらの代替演算子"&"、"|"、"!"には短絡評価が用意されています。

付録 3

DS2 サンプルプログラム

例: 最小値を見つける **2129**
例の概要: 最小値を見つける 2129
コード例: 最小値を見つける 2130
出力例: 最小値を見つける 2131

例: DS2 プログラムの SQL **2131**
例の概要: DS2 プログラムの SQL 2131
コード例: DS2 プログラムの SQL 2131
出力例: DS2 プログラムの SQL 2133

例: 条件に基づいて 2 つの新しいテーブルを作成する **2134**
例の概要: 条件に基づいて 2 つの新しいテーブルを作成する 2134
コード例: 条件に基づいて 2 つの新しいテーブルを作成する 2134
出力例: 条件に基づいて 2 つの新しいテーブルを作成する 2136

例: テキスト出力の大文字/小文字の変更 **2136**
例の概要: テキスト出力の大文字/小文字の変更 2136
コード例: テキスト出力の大文字/小文字の変更 2136
出力例: テキスト出力の大文字/小文字の変更 2137

例: スコープ **2137**
例の概要: スコープ 2137
コード例: スコープ 2138
出力例: スコープ 2139

例: 関数 **2139**
例の概要: 関数 2139
コード例: 関数 2139
出力例: 関数 2140

例: 配列 **2140**
例の概要: 配列 2140
コード例: 配列 2141
出力例: 配列 2143

例: SELECT ステートメント **2145**
例の概要: SELECT ステートメント 2145
コード例: SELECT ステートメント 2145
出力例: SELECT ステートメント 2146

例: ラベル付きの GOTO および LEAVE ステートメント **2146**

例の概要: ラベル付きの GOTO および LEAVE ステートメント 2146
コード例: ラベル付きの GOTO および LEAVE ステートメント 2147
出力例: ラベル付きの GOTO および LEAVE ステートメント 2148

例: オーバーロードされたメソッド **2149**
例: オーバーロードされたメソッド 2149
コード例: オーバーロードされたメソッド 2149
出力例: オーバーロードされたメソッド 2150

例: 複数のスレッドを使用してテーブルを分析する **2151**
例の概要: 複数のスレッドを使用してテーブルを分析する 2151
コード例: 複数のスレッドを使用してテーブルを分析する 2151
出力例: 複数のスレッドを使用してテーブルを分析する 2152

例: 4 つのスレッドを使用して要約統計量を計算する **2153**
例の概要: 4 つのスレッドを使用して要約統計量を計算する 2153
コード例: 4 つのスレッドを使用して要約統計量を計算する 2153
出力例: 4 つのスレッドを使用して要約統計量を計算する 2155

例: データクリーニング **2155**
例の概要: データクリーニング 2155
コード例: データクリーニング 2156
出力例: データクリーニング 2157

例: SUBSTR 関数 **2157**
例の概要: SUBSTR 関数 2157
コード例: SUBSTR 関数 2157
出力例: SUBSTR 関数 2158

例: PUT で SAS 出力形式を使用 **2158**
概要の例: PUT で SAS 出力形式を使用する 2158
コード例: PUT で SAS 出力形式を使用する 2158
出力例: PUT で SAS 出力形式を使用する 2159

例: テーブルデータから統計量を生成する **2159**
例の概要: テーブルデータから統計量を生成する 2159
コード例: テーブルデータから統計量を生成する 2160
出力例: テーブルデータから統計量を生成する 2161

例: 行列と非線形方程式 **2161**
例の概要: 行列と非線形方程式 2161
コード例: 行列と非線形方程式 2162
出力例: 行列と非線形方程式 2164

例: DS2 行列と回帰分析 **2165**
例: DS2 行列と回帰分析 2165
コード例: DS2 行列と回帰分析 2165
出力例: DS2 行列と回帰分析 2167

例: バルク挿入を使用してテーブルを追加する **2168**
例の概要: バルク挿入を使用してテーブルを追加する 2168
コード例: バルク挿入を使用してテーブルを追加する 2168
出力例: バルク挿入を使用してテーブルを追加する 2169

例: 別のパッケージで SQLSTMT パッケージを使用する **2170**
例の概要: 別のパッケージで SQLSTMT パッケージを使用する 2170
コード例: 別のパッケージで SQLSTMT パッケージを使用する 2170

例: 2 つのデータベースを使用してテーブルの値を更新する **2171**
例の概要: 2 つのデータベースを使用してテーブルの値を更新する 2171

コード例: 2 つのデータベースを使用してテーブルの値を更新する 2172

例: FedSQL ステートメントをハッシュパッケージに保存する **2172**
例の概要: FedSQL ステートメントをハッシュインスタンスに保存する 2172
コード例: FedSQL ステートメントをハッシュパッケージに保存する 2173
出力例: FedSQL ステートメントをハッシュパッケージに保存する 2175

例: MySQL 8 CHAR()データから末尾のスペースをフィルタリングする **2178**
例の概要: MySQL 8 CHAR()データから末尾のスペースをフィルタリングする ..2178
コード例: MySQL 8 CHAR()データから末尾のスペースをフィルタリングする ..2178
出力例: MySQL 8 CHAR()データから末尾のスペースをフィルタリングする 2180

例: HTTP および JSON パッケージを使用して REST 形式のファイルを抽出および解析する **2180**
例の概要: HTTP および JSON パッケージを使用して REST 形式のファイルを抽出および解析する 2180
コード例: HTTP および JSON パッケージを使用して REST 形式のファイルを抽出および解析する 2180
出力例: HTTP および JSON パッケージを使用して REST 形式のファイルを抽出および解析する 2182

例: HTTP から JSON テキストを読み取り、SAS データセットを作成する **2184**
例の概要: HTTP から JSON テキストを読み取り、SAS データセットを作成する 2184
コード例: HTTP から JSON テキストを読み取り、SAS データセットを作成する 2184
出力例: HTTP から JSON テキストを読み取り、SAS データセットを作成する ..2187

例: 最小値を見つける

例の概要: 最小値を見つける

この例は、3 つのシステム定義メソッド、INIT、RUN、および TERM の使用方法を示しています。

DS2 プログラムは、次の順序で実行されます。

1 すべてのグローバル変数が宣言されます。

2 INIT メソッドが呼び出されます。INIT は通常、変数の初期化に使用されます。

3 RUN メソッドが呼び出されます。RUN メソッドは、暗黙的なループが存在する場所です。RUN は、すべての 入力テーブルが完全に読み取られるまで実行されます。

4 TERM メソッドが呼び出されます。最終処理が行われます。

テーブル内の最小値を検索する次のプログラムを使用して、この制御フローを説明します。このプログラムでは、INIT メソッドが現在の最小値を見つけるために使用される変数を初期化し、RUN メソッドが入力値を現在の最小値と比較し、TERM メソッドが最小値を出力テーブルに書き込みます。

コード例: 最小値を見つける

```
proc ds2;
/* Create table to work with in this example */
data xy_data;
  dcl double x y;
  method init();
    do x = 1 to 5;
      y = 2*x;
      output;
    end;
  end;
enddata;
run;
/* Find the minimum value for x and y */
data xy_mins;
  dcl double min_x min_y;
  retain min_x min_y;
  keep min_x min_y;

  method init();
    min_x = 999999;
    min_y = 999999;
  end;

  method run();
    set xy_data;
    if x < min_x then min_x = x;
    if y < min_y then min_y = y;
  end;

  method term();
    output;
  end;
enddata;
run;
/* Send result table of minimums to the PROC for output */
data;
  method run();
    set xy_mins;
  end;
enddata;
run;
quit;
```

出力例: 最小値を見つける

The SAS System

MIN_X	MIN_Y
1	2

例: DS2 プログラムの SQL

例の概要: DS2 プログラムの SQL

この例は、DS2 プログラムで SQL ステートメントを使用する方法を示しています。SQL クエリからの出力にアクセスするには、クエリを SET ステートメントに入れます。SET ステートメントを実行すると、クエリによって返された行が順番に読み取られます。

コード例: DS2 プログラムの SQL

この例では、最初の DS2 プログラムは、2004 年から 2014 年まで毎年 2000 ドルの寄付が行われる口座の年間残高を計算します。口座の利率は 7%で、毎年複利計算されます。2 番目のプログラムはテーブルを生成します。3 番目のプログラムは、SQL クエリの結果を書き込みます。クエリは、INVESTMENT_YEAR の値が 2010 より大きい INVESTMENT からすべての行を選択します。

```
proc ds2;
data investment;
  dcl integer investment_year;
  dcl double capital;
  method init();
    capital = 0;
    do investment_year = 2004 to 2014;
      capital = capital + (2000 + .07 * (capital+2000));
      output;
    end;
  end;
enddata;
run;
```

```
data;
  method run();
    set investment;
  end;
enddata;
run;
data;
  method run();
    set {select * from investment where investment_year > 2010};
  end;
enddata;
run;
quit;
```

出力例: DS2 プログラムの SQL

The SAS System

INVESTMENT_YEAR	CAPITAL
2004	2140
2005	4429.8
2006	6879.886
2007	9501.478
2008	12306.58
2009	15308.04
2010	18519.61
2011	21955.98
2012	25632.9
2013	29567.2
2014	33776.9

The SAS System

INVESTMENT_YEAR	CAPITAL
2011	21955.98
2012	25632.9
2013	29567.2
2014	33776.9

例: 条件に基づいて 2 つの新しいテーブルを作成する

例の概要: 条件に基づいて 2 つの新しいテーブルを作成する

この例は、条件に基づいてテーブルを作成する方法を示しています。プログラム 1 と 2 は、2 つの部門が使用するアイテムのコストを保持する 2 つのテーブル DEPT1_ITEMS と DEPT2_ITEMS を作成します。3 番目のプログラムは、2 つのアイテムテーブルのアイテムのコストに 基づいて、HIGHCOSTS と LOWCOSTS の 2 つのテーブルを作成します。プログラム 4 と 5 は、コストテーブルのコンテンツを出力します。

コード例: 条件に基づいて 2 つの新しいテーブルを作成する

```
proc ds2;
/* Program 1 */
data dept1_items (overwrite=yes);
  dcl varchar(20) item;
  dcl double cost;
  method init();
    item = 'staples';   cost =  1.59; output;
    item = 'pens';      cost =  3.26; output;
    item = 'envelopes'; cost = 11.42; output;
  end;
enddata;
run;
/* Program 2 */
data dept2_items (overwrite=yes);
  dcl varchar(20) item;
  dcl double cost;
  method init();
    item = 'erasers'; cost =  5.43; output;
    item = 'paper';   cost = 26.92; output;
    item = 'toner';   cost = 62.29; output;
  end;
enddata;
run;
```

```
/* Program 3 */
data lowCosts (overwrite=yes) highCosts (overwrite=yes);
  method run();
    set dept1_items dept2_items;
    if cost <= 10.00 then
      output lowCosts;
    else
      output highCosts;
    end;
enddata;
run;
/* Program 4 */
data;
  method run();
    set lowCosts;
  end;
enddata;
run;
/* Program 5 */
data;
  method run();
     set highCosts;
  end;
enddata;
run;
quit;
```

出力例: 条件に基づいて 2 つの新しいテーブルを作成する

The SAS System

ITEM	COST
staples	1.59
pens	3.26
erasers	5.43

The SAS System

ITEM	COST
envelopes	11.42
paper	26.92
toner	62.29

例: テキスト出力の大文字/小文字の変更

例の概要: テキスト出力の大文字/小文字の変更

この例のコードは、前の例で作成された 2 つのテーブル DEPT1_COSTS と DEPT2_COSTS を読み取り、COST 列の値が小文字で$10 以下、大文字で$10 より大きい行を書き込みます。

コード例: テキスト出力の大文字/小文字の変更

```
proc ds2;
```

```
data;
  method run();
    set dept1_items dept2_items;
      if cost <= 10.00 then
        item = lowcase(item);
      else
        item = upcase(item);
      end;
enddata;
run;
quit;
```

出力例: テキスト出力の大文字/小文字の変更

The SAS System

ITEM	COST
staples	1.59
pens	3.26
ENVELOPES	11.42
erasers	5.43
PAPER	26.92
TONER	62.29

例: スコープ

例の概要: スコープ

この例は、CHAR 型と VARCHAR 型、およびそれらの演算子と関数の使用法を示しています。また、この例では、変数 A、B、および C はローカルで INIT メソッドにスコープされています。つまり、それらの値は INIT メソッドの外部には表示されず、結果テーブルに書き込まれません。

コード例: スコープ

```
proc ds2;
data;
  dcl char(24) abc abc2;
  method init();
    dcl char(8) a b c;

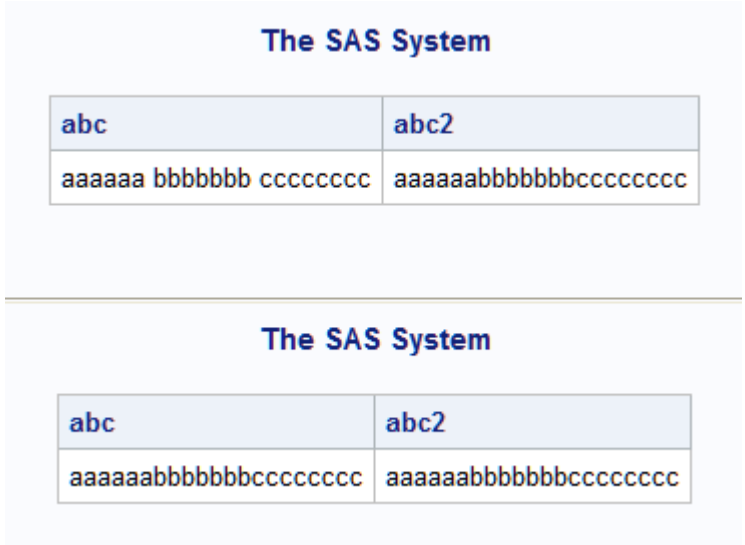
    a = repeat('a',5);
    b = repeat('b',6);
    c = repeat('c',7);

    abc = a || b || c;
    abc2 = trim(a) || trim(b) || c;
  end;
enddata;
run;
data;
  dcl char(24) abc abc2;
  method init();
    dcl varchar(8) a b c;

    a = repeat('a',5);
    b = repeat('b',6);
    c = repeat('c',7);

    abc = a || b || c;
    abc2 = trim(a) || trim(b) || c;
  end;
enddata;
run;
quit;
```

出力例: スコープ



The SAS System

abc	abc2
aaaaaa bbbbbbb cccccccc	aaaaaabbbbbbbccccccccc

The SAS System

abc	abc2
aaaaaabbbbbbbccccccccc	aaaaaabbbbbbbccccccccc

例: 関数

例の概要: 関数

この例は、DS2 で関数を使用する方法を示しています。このコードは、整数を保存するために必要なビット数を計算します。

コード例: 関数

```
proc ds2;
data;
  dcl integer i bits;
  method init();
    do i = 1 to 1000 by 100;
      bits = ceil(log(i) / log(2));
      output;
    end;
  end;
enddata;
run;
```

```
quit;
```

出力例: 関数

The SAS System

i	bits
1	0
101	7
201	8
301	9
401	9
501	9
601	10
701	10
801	10
901	10

例: 配列

例の概要: 配列

最初のプログラムは、基本的な配列プロシジャを示しています。最後のセクションでは、dblNegSubArray の要素は、配列かっこ内の数値式を使用して指定されます。配列の宣言されたサイズの範囲外の数値に解決される式は、エラーメッセージを表示します。

2 番目のプログラムは、配列割り当てのいくつかの例を示しています。配列が割り当てられると、ターゲット配列の型と一致しないデータ型がターゲット配列型に変換されます。

PUT ステートメントの変数または配列要素の後に等号を追加すると、出力の前に変数または配列要素の名前と等号が表示されることに注意してください。

注: この例では、DS2 プロシジャオプション SCOND=NONE が設定されており、DS2 変数厳格モードによって生成されるログメッセージを抑制します。

コード例: 配列

```
proc ds2 scond=none;
/* Basic Arrays */
data _null_;
  dcl char(10)  strArray[4] str;
  dcl double    dblArray[3];
  dcl int       intArray[10];
  dcl double    dblNegSubArray[-4:-1];
  dcl int x;
  method init();
    put 'BASIC ARRAYS';
    strArray[1] = 'abc';
    strArray[2] = 'def';
    strArray[3] = 'ghi';
    strArray[4] = 'jkl';
    put strArray[1]= ;
    put strArray[2]= ;
    put strArray[3]= ;
    put strArray[4]= ;
    put;

    str = strArray[1];
    put str=;
    put;

    dblArray[1] = 3;
    dblArray[2] = 99;
    dblArray[3] = dblArray[2];
    put dblArray[1]= ;
    put dblArray[2]= ;
    put dblArray[3]= ;
    put;

    do j = 1 to 10;
      intArray[j] = j;
      put intArray[j]=;
    end;
    put;

    dblArray[3] = intArray[5];
    put dblArray[3]=;
    put;

    dblNegSubArray[-4] = 102;
    dblNegSubArray[-3] = 101;
    dblNegSubArray[-2] = 100;
```

```
    dblNegSubArray[-1] = 99;

    x = 5;
    y = 7;
    a = dblNegSubArray[x-y];
    b = dblNegSubArray[x-y-1];
    c = dblNegSubArray[x-y-2];
    put a= b= c=;
    put;

    /* These will produce out-of-bounds messages */
    a = dblNegSubArray[x-y-3];
    e = dblNegSubArray[0];
  end;
enddata;
run;
/* Array Assignment */
data _null_;
  dcl int x;
  dcl char(10) s1[4] s2[4];
  dcl double d1[10] d2[10];
  dcl double arr2x3[2,3] arr3x2[3:5,-1:0];

  method init();

    put 'ARRAY ASSIGNMENT';

    /* Assign array of constants to array s1 */
    s1 := ('abc', 'def', 'ghi', 'jkl');

    /* Assign array s to array s2 */
    s2 := s1;
    put s2[1]= ;
    put s2[2]= ;
    put s2[3]= ;
    put s2[4]= ;
    put;

    /*
    * Assign array of constants to array d1.  Use
    * iterators for repeated values.  Mismatched
    * types will be converted automatically.
    */
    d1 := (3*3.14159, 2*'5', 2*(1,2), 99);

    /* Assign array d to array e */
    d2 := d1;
    put d2[1]= ;
    put d2[2]= ;
    put d2[3]= ;
    put d2[4]= ;
    put d2[5]= ;
    put d2[6]= ;
    put d2[7]= ;
    put d2[8]= ;
    put d2[9]= ;
```

```
    put d2[10]=;
    put;

    /* Assign array of constants to array arr3x2 */
    arr2x3 := (2*(1,2,3));

    /* Assign  arr2x3 to arr3x2 */
    arr3x2 := arr2x3;
    put arr2x3[1,1]= arr3x2[3,-1]= ;
    put arr2x3[1,2]= arr3x2[3,0]= ;
    put arr2x3[1,3]= arr3x2[4,-1]= ;
    put arr2x3[2,1]= arr3x2[4,0]= ;
    put arr2x3[2,2]= arr3x2[5,-1]= ;
    put arr2x3[2,3]= arr3x2[5,0]= ;
  end;
enddata;
run;
quit;
```

出力例: 配列

次の行が SAS ログに書き込まれます。

```
BASIC ARRAYS
strArray[1]=abc
strArray[2]=def
strArray[3]=ghi
strArray[4]=jkl

str=abc

dblArray[1]=3
dblArray[2]=99
dblArray[3]=99

intArray[1]=1
intArray[2]=2
intArray[3]=3
intArray[4]=4
intArray[5]=5
intArray[6]=6
intArray[7]=7
intArray[8]=8
intArray[9]=9
intArray[10]=10

dblArray[3]=5

a=100 b=101 c=102

ERROR: Line 225: Invalid index value -5 for subscript 1 of array dblNegSubArray; allowed range
     is [-4,-1].
ERROR: Line 226: Invalid index value 0 for subscript 1 of array dblNegSubArray; allowed range
     is [-4,-1].
NOTE: Execution succeeded. No rows affected.

ARRAY ASSIGNMENT
s2[1]=abc
s2[2]=def
s2[3]=ghi
s2[4]=jkl

d2[1]=3.14159
d2[2]=3.14159
d2[3]=3.14159
d2[4]=5
d2[5]=5
d2[6]=1
d2[7]=2
d2[8]=1
d2[9]=2
d2[10]=99

arr2x3[1,1]=1 arr3x2[3,-1]=1
arr2x3[1,2]=2 arr3x2[3,0]=2
arr2x3[1,3]=3 arr3x2[4,-1]=3
arr2x3[2,1]=1 arr3x2[4,0]=1
arr2x3[2,2]=2 arr3x2[5,-1]=2
arr2x3[2,3]=3 arr3x2[5,0]=3
```

例: SELECT ステートメント

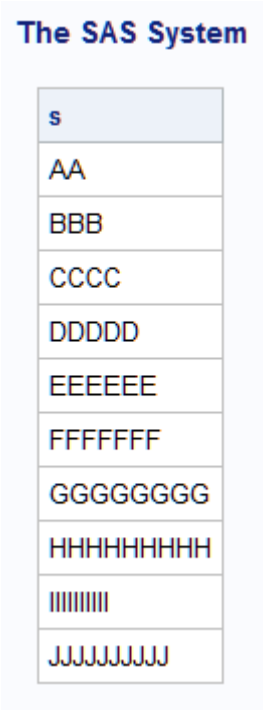
例の概要: SELECT ステートメント

この例は、SELECT ステートメントを示しています。この例では、DO ループが SELECT ステートメントを囲んでいます。SELECT ステートメントはループカウンター I の現在値を読み取り、WHEN ステートメントが真の場合に文字を書き込みます。REPEAT ステートメントは、ループカウンターの値に基づいて文字を繰り返し出力します。

コード例: SELECT ステートメント

```
proc ds2;
data;
  dcl char(10) s;
  method run();
    dcl char(1) x;
    dcl int i;
    do i=1 to 10;
      select(i);
        when(1) x='A';
        when(2) x='B';
        when(3) x='C';
        when(4) x='D';
        when(5) x='E';
        when(6) x='F';
        when(7) x='G';
        when(8) x='H';
        when(9) x='I';
        otherwise x='J';
      end;
      s=repeat(x,i);
      output;
    end;
  end;
enddata;
run;
quit;
```

出力例: SELECT ステートメント



The SAS System

s
AA
BBB
CCCC
DDDDD
EEEEEE
FFFFFFF
GGGGGGGG
HHHHHHHHH
IIIIIIIII
JJJJJJJJJ

例: ラベル付きの GOTO および LEAVE ステートメント

例の概要: ラベル付きの GOTO および LEAVE ステートメント

この例は、分岐を示す 3 つのプログラムを示しています。1 つ目は、GOTO を使用してラベルに分岐します。2 つ目はラベルなしの LEAVE を使用し、3 つ目はラベル付きの LEAVE を使用します。詳細については、“GOTO ステートメント” (1359 ページ)および“LEAVE ステートメント” (1367 ページ)を参照してください。

コード例: ラベル付きの GOTO および LEAVE ステートメント

```
proc ds2;
data;
  dcl double i j;
  method init();
    i = 1;
  head:
    j = 2*i;
    i = i+1;
    if i < 10 then do;
      output;
      goto head;
    end;
  end;
enddata;
run;
data _null_;
  dcl int x y;
  method init();
    put 'loop test 1';
    x = 1;
    y = 2;
    if (x ~= -5) then
      do i = 1 to 10;
        put i;
        if i > 4 then leave;
      end;
    else
      put 'else';
  end;
enddata;
run;
data _null_;
  dcl int g;
  method init();
    put 'label test 1';
    x = 1;
    y = 2;
    if (x > -5) then
      lab:
      do i = 1 to 10;
        do j = 1 to 5;
          put i j;
          if i > 4 then leave lab;
        end;
      end;
    else
      do;
```

```
          put 'else';
        end;
    end;
enddata;
run;
quit;
```

出力例: ラベル付きの GOTO および LEAVE ステートメント

次の出力は、GOTO を使用してラベルに分岐した結果です。

The SAS System

i	j
2	2
3	4
4	6
5	8
6	10
7	12
8	14
9	16

ループテストからのこれらの行は、SAS ログに書き込まれます。

```
loop test 1
1
2
3
4
5
```

ラベルテストからのこれらの行は、SAS ログに書き込まれます。

```
label test 1
1 1
1 2
1 3
1 4
1 5
2 1
2 2
2 3
2 4
2 5
3 1
3 2
3 3
3 4
3 5
4 1
4 2
4 3
4 4
4 5
5 1
```

例: オーバーロードされたメソッド

例: オーバーロードされたメソッド

この例は、オーバーロードされたメソッドを設定する方法を示しています。メソッドの詳細については、“METHOD ステートメント” (1376 ページ)を参照してください。

コード例: オーバーロードされたメソッド

```
proc ds2;
data _null_;
  method concat(nvarchar(200) x, nvarchar(200) y) returns nvarchar(400);
    return x || y;
  end;

  method concat(nvarchar(200) x, nvarchar(200) y, nvarchar(200) z)
    returns nvarchar(600);
    return x || y || z;
  end;

  method run();
    y = concat(n'abc', n'def');
    put 'y= ' y;
```

```
    y = concat(n'abc', n'def', n'ghi');
    put 'y= ' y;
  end;
enddata;
run;
data _null_;
  method d() returns double;
    return 99;
  end;

  method d(double x, double y, double z) returns double;
    return x + y + z;
  end;

  method d(int x, int y, int z) returns int;
    return x + y + z + 500;
  end;

  method run();
    dcl double d;
    d = d(1,2,3);
    put 'd= ' d;
    d = d(100.1, 100.2, 100.3);
    put 'd= ' d;
  end;
enddata;
run;
quit;
```

出力例: オーバーロードされたメソッド

次の行が SAS ログに書き込まれます。

```
y=  abcdef
y=  abcdefghi

d=  506
d=  300.6
```

例: 複数のスレッドを使用してテーブルを分析する

例の概要: 複数のスレッドを使用してテーブルを分析する

この例は、複数のスレッドを使用して分析されるテーブルの作成を示しています。

コード例: 複数のスレッドを使用してテーブルを分析する

```
libname spde spde 'c:\temp\spde';
proc delete data=spde.incomes; run;
proc delete data=spde.results1; run;
proc delete data=spde.results2; run;

proc ds2;
data spde.incomes;
  dcl double income citycode;
  dcl char(8) name;
  method run();
    do j = 1 to 1E6;
      name = 'John';  income = 23234; citycode=1; output;
      name = 'Jane';  income = 62348; citycode=1; output;
      name = 'Joe';   income = 32932; citycode=2; output;
      name = 'Jan';   income = 58239; citycode=2; output;
      name = 'Josh';  income =  6523; citycode=3; output;
      name = 'Jill';  income = 80392; citycode=3; output;
      /* The three people to find during mining */
      if j = 5E5 then do;
        name = 'James'; income = 103243; citycode=1; output;
      end;
      if j = 7E5 then do;
        name = 'Joan'; income = 233923; citycode=2; output;
      end;
      if j = 8E5 then do;
        name = 'Joyce'; income = 132443; citycode=3; output;
      end;
    end;
  end;
```

```
enddata;
run;

thread score;
  method run();
    set spde.incomes;

    accept = 0;
    if citycode = 1 and income > 100000 then accept = 1;
    else if citycode = 2 and income > 200000 then accept = 1;
    else if citycode = 3 and income > 120000 then accept = 1;

    /* faux work */
    do i = 1 to 50; end;

    if accept then output;
  end;
endthread;
run;

data spde.results1;
  dcl thread score score;
  method run();
    set from score threads=1;
  end;
enddata;
run;

data spde.results2;
  dcl thread score score;
  method run();
    set from score threads=2;
  end;
enddata;
run;
quit;
```

出力例: 複数のスレッドを使用してテーブルを分析する

次の行が SAS ログに書き込まれます。

```
NOTE: Execution succeeded.6000003 rows affected.
NOTE: Execution succeeded.No rows affected.
NOTE: Execution succeeded.3 rows affected.
NOTE: Execution succeeded.3 rows affected.
```

例: 4 つのスレッドを使用して要約統計量を計算する

例の概要: 4 つのスレッドを使用して要約統計量を計算する

次の例では、いくつかの要約統計量を計算するスレッドプログラムを作成します。スレッドプログラムは 4 つのスレッドで実行されます。プログラムは、どのスレッドがどの値を生成するかを調べます。

コード例: 4 つのスレッドを使用して要約統計量を計算する

```
/* Expand sashelp.class */
data class;
  do i = 1 to 1E5;
    do j = 1 to nobs;
      set sashelp.class nobs=nobs point=j;
      output;
    end;
  end;
  stop;
run;

proc ds2;

/* Create the thread program
 * When the thread program executes in N threads, each thread receives
 * a unique set of rows from the table work.class.
 * This thread program sums all the student ages it sees along
 * with counting the number of men and women it sees.
 * Each thread's partial sum is output to a data program for
 * final summing.
 */
  thread sum_student_measures / overwrite=yes;
    dcl double id cnt sum_age cnt_male cnt_female;
    keep id cnt sum_age cnt_male cnt_female;
    method run();
      set class;
```

```
      sum_age + age;
      cnt_male + (if sex = 'M' then 1 else 0);
      cnt_female + (if sex = 'F' then 1 else 0);
    end;

    method term();
    /* _threadid_ is the thread's "number" from 0 to N-1, when using N threads. */
      id = _threadid_;
      cnt = _N_;
      output;
    end;
  endthread;
run;

/* Start the thread program in 4 threads, sum the values
 * received and output averages and total counts.
 * The variable ID is the thread number for the thread that
 * produced a particular set of counts.  This is useful
 * in looking at which thread output which values.
 * Note in the output how the PUT statement output isn't ordered
 * by ID.  This is because some threads finish sooner than others.
 * Also notice the variable CNT which indicates that different
 * threads operate on different numbers of rows.
 */
data class_counts / overwrite=yes;
  dcl thread sum_student_measures t();
  dcl double tot_cnt avg_age tot_male tot_female tot_age;
  keep tot_count avg_age tot_male tot_female;
  method run();
    set from t threads=4;
    put id= cnt= sum_age= cnt_male= cnt_female=;

    tot_age + sum_age;
    tot_male + cnt_male;
    tot_female + cnt_female;
    tot_cnt + cnt;
  end;

  method term();
    avg_age = tot_age / tot_cnt;
    output;
  end;
enddata;
run;
quit;

proc print data=class_counts; run;
```

出力例: 4 つのスレッドを使用して要約統計量を計算する

次の行が SAS ログに書き込まれます。プログラムを実行するたびに、ログ出力が異なることに注意してください。

```
id=2 cnt=1 sum_age=0 cnt_male=0 cnt_female=0
id=0 cnt=1404506 sum_age=18702097 cnt_male=739215 cnt_female=665290
id=3 cnt=1 sum_age=0 cnt_male=0 cnt_female=0
id=1 cnt=495496 sum_age=6597903 cnt_male=260785 cnt_female=234710
```

次のテーブルが生成されます。

Obs	avg_age	tot_male	tot_female
1	13.3158	1000000	900000

例: データクリーニング

例の概要: データクリーニング

入力エラーが原因で、データ値の先頭と末尾に空白が含まれる場合があります。この可能性を特にフィルタリングしない限り、同じ形式のデータ値に依存する結合および同様のデータベース操作は正しく実行されません。発生する可能性のある別の問題は、レコードの意図しない重複(つまり、同じキーを持つ複数のレコード(行))です。この例は、データ値から空白を削除し、削除の可能性がある重複行をリストするプログラムを示しています。

最初の DS2 プログラムは、重複する行と文字列値を含む単純なデータテーブル EMPLOYEES1 を作成します。一部の行には空白が含まれています。

2 番目のプログラムは、STRIP 関数を使用して、EMP 列の値から先頭と末尾の空白を削除します。テーブルは EMPLOYEES2 に書き込まれます。

3 番目のプログラムは、埋め込み SQL SELECT ステートメントを使用して重複を表示します。この出力を使用して、削除する必要があるレコードを決定できます。なお、このプログラムでは、最初にデータ値から空白を削除しないと、SELECT ステートメントがデータを正確にグループ化しないため、正しい出力が得られません。

コード例: データクリーニング

```
proc ds2;
data employees1 (overwrite=yes);
  dcl double id;
  dcl char emp;
  method init();
    id =  60918 ; emp = 'user1'; output;
    id =  60919 ; emp = ' user2'; output;
    id =  60920 ; emp = ' user3'; output;
    id =  60918 ; emp = 'user1'; output;
    id =  60922 ; emp = 'user4'; output;
    id =  60925 ; emp = ' user5 '; output;
    id =  60926 ; emp = 'user6'; output;
    id =  60919 ; emp = 'user2'; output;
    id =  60928 ; emp = ' user7'; output;
    id =  60918 ; emp = 'user1'; output;
  end;
enddata;
run;
data employees2 (overwrite=yes);

  method run();
    set employees1;
    emp = strip(emp);
  end;
enddata;
run;
data ;
  method run();
    set {select id as dupid,  count(id) as dups
      from  employees2
      group by id
      having count(id) > 1};

  end;
enddata;
run;
quit;
```

出力例: データクリーニング

The SAS System

DUPID	DUPS
60918	3
60919	2

例: SUBSTR 関数

例の概要: SUBSTR 関数

この例は、SUBSTR 関数の使用方法を示しています。SUBSTR は、一度に 1 文字ずつ変更することにより、CAT という単語を DOG に変換するために使用されます。

コード例: SUBSTR 関数

```
proc ds2;
data _null_;
  dcl char(3) a b;
  method init();
    a = 'cat';
    put 'a=' a;

    substr(a,2,1) = 'o';
    put 'a=' a;

    substr(a,1,1) = 'd';
    put 'a=' a;

    substr(a,3,1) = 'g';
    put 'a=' a;
    b = a;
  end;
enddata;
run;
```

出力例: SUBSTR 関数

次の行が SAS ログに書き込まれます。

```
a= cat
a= cot
a= dot
a= dog
```

例: PUT で SAS 出力形式を使用

概要の例: PUT で SAS 出力形式を使用する

この例は、PUT ステートメントで SAS 出力形式を使用する方法を示しています。

コード例: PUT で SAS 出力形式を使用する

```
proc ds2;
data _null_;
  method init();
    dcl double d;
    dcl int i;
    dcl char(32) x;

    d = 99;
    x = put(d, best12.6);
    put 'x=' x;

    d = 100;
    x = put(d, best.);
    put 'x=' x;

    i = 10847;
    x = put(i, mmddyy8.);
    put 'x=' x;

    c = 'abc';
    x = put(c, $char8.);
    put 'x=' x;
```

```
  end;
enddata;
run;
quit;
```

出力例: PUT で SAS 出力形式を使用する

次の行が SAS ログに書き込まれます。

```
x=         99
x=        100
x= 09/12/89
x= abc
```

例: テーブルデータから統計量を生成する

例の概要: テーブルデータから統計量を生成する

この例では、シミュレートされた株価テーブルの価格変動を分析します。最初の DS2 プログラムは、株価テーブルを作成します。株価の始値、高値、安値、終値が 2 週間分あります。

2 番目のプログラムは分析を実行します。終値の 2 日間の移動平均は、過去 2 日間の終値を CY 配列の要素に割り当て、MEAN 関数を使用して平均し、C_MA として出力することによって作成されます。INIT メソッドは、配列 CY 変数と C_MA 変数を STOCK テーブルの最初の価格に初期化します。RUN メソッドは、テーブルをループするときに配列に値を割り当てます。

CTR 変数は、RUN メソッドを介して毎回増分されます。移動平均は、2 日間の価格が平均化されるまで有効ではないため、出力は 3 番目のレコードから始まります(つまり、CTR > 2 の場合)。行が書き込まれた後、CY 配列が更新されます。

次の統計量も計算されます。

- 今日の終値から昨日の終値 CY[1]を差し引いて計算された終値 **Chng** の毎日の変化
- 始値から終値への変化 O_C
- 高値から安値を差し引いて計算された、その日の価格の範囲 H_L

コード例: テーブルデータから統計量を生成する

```
proc ds2;
data stock (overwrite=yes);
  dcl date d;
  dcl double o h l c;
  method init();
    d = date '2010-09-18' ; o =  20; h = 22.25; l = 18; c = 21.5; output;
    d = date '2010-09-19' ; o =  21; h = 23.5; l = 19.25; c = 22; output;
    d = date '2010-09-20' ; o =  22.25; h = 24.75; l = 20; c = 21; output;
    d = date '2010-09-21' ; o =  21; h = 21.5; l = 18.75; c = 19; output;
    d = date '2010-09-22' ; o =  18; h = 19; l = 17.25; c = 17; output;
    d = date '2010-09-25' ; o =  17; h = 18; l = 15.5; c = 17; output;
    d = date '2010-09-26' ; o =  17.5; h = 20; l = 16; c = 18; output;
    d = date '2010-09-27' ; o =  18.5; h = 21; l = 18; c = 20; output;
    d = date '2010-09-28' ; o =  20; h = 22.25; l = 19.5; c = 21; output;
    d = date '2010-09-29' ; o =  21; h = 23; l = 18.75; c = 21.5; output;
  end;
enddata;
run;
data;
  keep d o h l c Chng O_C H_L C_MA;
  dcl double cy[2] C_MA H_L Chng O_C;
  dcl int ctr;
  method init();
    set stock (locktable=share);
    c_ma = c;
    cy[1] = c;
    cy[2] = c;
  end;

  method run();
    set stock (locktable=share);
    ctr+1;
    C_MA = mean(cy[1], cy[2]);
    H_L = h - l;
    O_C = o - c;
    Chng = c - cy[1];
    if ctr > 2 then output;
      cy[2] = cy[1];
      cy[1] = c;
  end;
enddata;
run;
```

出力例: テーブルデータから統計量を生成する

The SAS System

d	o	h	l	c	Chng	O_C	H_L	C_MA
20SEP2010	22.25	24.75	20	21	-1	1.25	4.75	21.75
21SEP2010	21	21.5	18.75	19	-2	2	2.75	21.5
22SEP2010	18	19	17.25	17	-2	1	1.75	20
25SEP2010	17	18	15.5	17	0	0	2.5	18
26SEP2010	17.5	20	16	18	1	-0.5	4	17
27SEP2010	18.5	21	18	20	2	-1.5	3	17.5
28SEP2010	20	22.25	19.5	21	1	-1	2.75	19
29SEP2010	21	23	18.75	21.5	0.5	-0.5	4.25	20.5

例: 行列と非線形方程式

例の概要: 行列と非線形方程式

この例では、ニュートンの反復プロセスを使用して非線形方程式のシステムを解きます。

この例では、ルート(特定のイプシロン内への)はループの 5 回の反復後に検出され、(.5, 0, –.5236)に等しくなります。ゼロは極端に小さい数として表示されます。

ニュートンの例で注目すべき興味深い機能の 1 つは、結果の行列を再利用する方法です。簡単なメソッド計算 r=m.mult(m2);では、結果の行列インスタンスが自動的に作成されます。ただし、結果インスタンスがすでに存在し、そのサイズが正しい場合、メソッドは効率化のために結果行列を再利用します。ニュートンの計算ループでこれがどのように行われるかに注意してください。行列が最初に作成されるとき、左辺の結果は ji=jm.inverse();などの計算で再利用されます。そうでなければ、古い行列を解放して、毎回新しい行列を割り当てる必要があります。

コード例: 行列と非線形方程式

```
proc ds2;
/*
 * Solve the system of equations
 *
 *   3*x1 + cos(x2*x3) - .5               = 0
 *   x1^2 - 81(x2 + .1)^2 + sin(x3) + 1.06 = 0
 *   e^(-x1*x2) + 20*x3 + (10pi-3)/3      = 0
 *
 * using Newton's method:
 *
 *   X_m = X_m0 - J^-1(X_m0) * F(X_m0)
 *
 */
data _null_;

  /* Infinity norm */

  method norm(double x[3]) returns double;
    return max(abs(x[3]), max(abs(x[1]), abs(x[2])));
  end;

  /* Pi computation */

  method pi() returns double;
    return atan(1)*4;
  end;

  /*
   *  f1(x1, x2, x3)
   *  f2(x1, x2, x3)
   *  f3(x1, x2, x3)
   */

  method compute_f(double f[3,1], double x[3]);
    f[1,1] = 3*x[1] - cos(x[2]*x[3]) - .5;
    f[2,1] = x[1]**2 - 81*(x[2]+.1)**2 + sin(x[3]) + 1.06;
    f[3,1] = exp(-x[1]*x[2]) + 20*x[3] + (10*pi() - 3)/3.0;
  end;


  /*
   *     Jacobian array
   *
   *    @f1() @f1() @f1()
   *    ---   ---   ---
   *    @x1   @x2   @x3
   *
   *    @f2() @f2() @f2()
   *    ---   ---   ---
   *    @x1   @x2   @x3
```

```
 *
 *    @f3() @f3() @f3()
 *    ---   ---   ---
 *    @x1   @x2   @x3
 */

method compute_j(double j[3,3], double x[3]);
  j[1,1] = 3;
  j[1,2] = x[3]*sin(x[2]*x[3]);
  j[1,3] = x[2]*sin(x[2]*x[3]);
  j[2,1] = 2*x[1];
  j[2,2] = -162*(x[2]+.1);
  j[2,3] = cos(x[3]);
  j[3,1] = -x[2]*exp(-x[1]*x[2]);
  j[3,2] = -x[1]*exp(-x[1]*x[2]);
  j[3,3] = 20;
end;

method init();
  dcl double j[3,3];
  dcl double f[3,1];
  dcl double y[3,1];
  dcl double x[3];
  dcl double x0[3];
  dcl double d[3];
  dcl package matrix jm;
  dcl package matrix ji;
  dcl package matrix fm;
  dcl package matrix ym;
  dcl package matrix xm0;
  dcl package matrix xm;
  dcl package matrix diff;
  dcl int niter;
  dcl double eps;

 /* Instantiate matrices */

  jm = _new_ matrix(3, 3);
  fm = _new_ matrix(3, 1);
  xm0 = _new_ matrix(3, 1);

 /* Initial approximation */

  x0[1] = .1;
  x0[2] = .1;
  x0[3] = -.1;

 /* Start loop */

  eps = 1;
  niter = 0;

  put '(' x0[1] ',' x0[2]',' x0[3] ')';

  do while(eps > 10**-6 and niter < 10);
```

```
      /* Compute functions with current approximation : j(x_0), f(x_0) */

        compute_j(j, x0);
        compute_f(f, x0);

       /* Load arrays into matrices */

        jm.load(j);
        fm.load(f);
        xm0.load(x0);

       /* Find inverse of Jacobian matrix */

        ji = jm.inverse();

       /* Multiply by function vector */

        ym = ji.mult(fm);

       /* Compute next approximation */

        xm = xm0.sub(ym);

       /* Compute error term */

        xm.toarray(x);
        diff = xm.sub(xm0);
        diff.toarray(d);
        eps = norm(d);

        x0 := x;
        put '(' x[1] ',' x[2]',' x[3] ')';
        put eps=;
        niter + 1;
      end;

      put niter=;
      put 'root = ('x[1] ',' x[2]',' x[3] ')';
    end;
enddata;
run;
quit;
```

出力例: 行列と非線形方程式

次の結果がログに表示されます。

```
( 0.1 , 0.1 , -0.1 )
( 0.49986967292642 , 0.01946684853741 , -0.52152047193583 )
eps=0.42152047193583
( 0.50001424016421 , 0.00158859137029 , -0.52355696434763 )
eps=0.01787825716712
( 0.50000011346783 , 0.00001244478332 , -0.52359845007288 )
eps=0.00157614658697
( 0.50000000000707 , 7.7578571058927E-10 , -0.523598775578 )
eps=0.00001244400753
( 0.5 , -2.1250842759061E-18 , -0.52359877559829 )
eps=7.7578571271436E-10
niter=5
root = ( 0.5 , -2.1250842759061E-18 , -0.52359877559829 )
```

例: DS2 行列と回帰分析

例: DS2 行列と回帰分析

この例では、DS2 行列を使用して、回帰分析のパラメーター推定値を求めます。

コード例: DS2 行列と回帰分析

```
libname z 'c:\mylib';

data _null_;
  dsid = open('z.hmeq4');
  nobs = attrn(dsid, 'nobs');
  call symput('nobs', nobs);
  hmnv = attrn(dsid, 'nvars');
  call symput ('hmnv', hmnv);
run;

/*
 * This does a regression approximation of the hmeq model variable
 * MORTDUE using the 'independent' variables CLAGE, CLNO, DELINQ,
 * DEROG, NINQ, VALUE and YOJ.
 *
 * We set up a matrix X with the data for the independent variables,
 * and a matrix Y with the dependent data, and then solve for the linear
 * coefficients b in y = X * b:
 *
 *    X'*y = X'*X*b
 *   (X'*X)^-1 * X'*y = b
 */

proc ds2;
```

```
data y(overwrite=yes);
  vararray double s[&hmnv] i clage clno delinq derog ninq value yoj;
  vararray double m[1] mortdue;

  method init();
    dcl package matrix x ym y xtx ti xt bm;
    dcl double b[&hmnv];

    /* Create the hmeq and mortdue matrices */
    x = _new_ matrix(&nobs, &hmnv);
    y = _new_ matrix(&nobs, 1);

    /* Read the data from the hmeq table */
    do j = 1 to &nobs
      set z.hmeq4;
      x.in(s, j);
    end;

  /*
   * x now contains the rows for the variables in hmeq -
   * plus a column of 1's for the constant term in the approximation:
   *
   * i   clage    clno    delinq   derog    ninq     value     yoj
   *
   * 1   94.367   9.0000   0.0000  0.00000   1.0000   39025.00  10.5000
   * 1  121.833  14.0000   2.0000  0.00000   0.0000   68400.00   7.0000
   * 1  149.467  10.0000   0.0000  0.00000   1.0000   16700.00   4.0000
   * 1  179.766  21.2961   0.4494  0.25457   1.1861  101776.05   8.9223
   *
   *            etc.
   */

  /* Read the data from the mortdue table */
    do j = 1 to &nobs
      set z.mortdue;
      y.in(m, j);
    end;

  /*
   * y is now a column vector containing the known values of MORTDUE
   */

  /* Make sure the row count matches */
    xr = x.rows();
    er = y.rows();

    if (xr ne er) then do;
      put 'invalid data';
      stop;
    end;

  /* Compute ti = (X'*X)^-1 */
    xt = x.trans();
    xtx = xt.mult(x);
    ti = xtx.inverse();
```

```
  /* Compute ym = X'*y */
    ym = xt.mult(y);

  /* Compute b = (X'*X)^-1 * X'y */
    bm = ti.mult(ym);

  /*
   * Thus
   *
   *   MORTDUE = b_0 + b_1 * CLAGE + b_2 * CLNO +
   *           b_3 * DELINQ + b_4 * DEROG + b_5 * NINQ +
   *           b_6 * VALUE + b_7 * YOJ + eps
   *
   * The b vector is equivalent to the parameter estimate from
   *
   *   proc reg; model mortdue= clage clno delinq derog ninq value yoj;run;
   *
   *   Variable     DF    Parameter Estimate   F Value 1434.92
   *
   *   Intercept     1         11473
   *   clage         1       -1.64128
   *   clno          1      466.74925
   *   delinq        1      -46.87948
   *   derog         1    -1371.68909
   *   ninq          1      544.05978
   *   value         1        0.56118
   *   yoj           1     -530.88585
   */

      bm.toarray(b);
      do i = 1 to &hmnv
        put b[i];
      end;
    end;
enddata;
run;
quit;
```

出力例: DS2 行列と回帰分析

次の結果がログに表示されます。

```
11472.5599166895
-1.64128448706739
466.749252257557
-46.8794774759862
-1371.68908534735
544.059779616903
0.56117547945582
-530.885851993411
```

例: バルク挿入を使用してテーブルを追加する

例の概要: バルク挿入を使用してテーブルを追加する

この例は、SQLSTMT パッケージをバルク挿入メソッドとともに使用して、2 つのデータセットを追加する方法を示しています。

コード例: バルク挿入を使用してテーブルを追加する

```
/* Create testdata1 table. */
proc ds2;
data testdata1 / overwrite=yes;
  dcl double a b;
  method init();
    a = 1;
    b = 2;
    output;
  end;
enddata;
run; quit;

/* Create testdata2 table. */
proc ds2;
data testdata2 / overwrite=yes;
  dcl double a b;
  method init();
    a = 3;
    b = 4;
    output;
  end;
enddata;
run; quit;

/* Append the contents of testdata1 to testdata2. */
proc ds2;
data _null_;
  dcl double a b;
```

```
dcl package sqlstmt stmt();

method init();
 dcl int rc;

 /* The SQL SELECT statement selects the columns for the bulk
  * insertions.  If the SQL SELECT statement generates a non-empty
  * result set, the DBMS driver may start transferring the rows from
  * the DBMS to the SAS session in anticipation of the rows being
  * fetched.  In the case of bulk insertions, the result set row
  * data will never be fetched, so the result set row transfer is
  * unnecessary.  The "where 1=0" clause produces an empty result
  * set avoiding the unnecessary row transfer. */
 rc = stmt.bulkInsertPrepare('select a,b from testdata2 where 1=0');
 if (rc) then put 'TEST ERROR: bulkInsertPrepare';

 rc = stmt.bulkInsertBindColumns([a, b]);
 if (rc) then put 'TEST ERROR: bulkInsertBindColumns';
end;

method run();
 dcl int rc;

 set testdata1;

 rc = stmt.bulkInsertAddRow();
 if (rc) then put 'TEST ERROR: bulkInsertAddRow ' a= b=;
end;

method term();
 dcl int rc;

 /* Explicit call to bulkInsertFlush is not required
  * since it will automatically be called when
  * sqlstmt instance is destroyed. Explicit call
  * is included to illustrate method call. */
 rc = stmt.bulkInsertFlush();
 if (rc) then put 'TEST ERROR: bulkInsertFlush';
end;
enddata;
run; quit;

/* Print the contents of testdata2. */
proc print data=testdata2;
quit;
```

出力例: バルク挿入を使用してテーブルを追加する

アウトプット A22.1　テストデータテーブル

Obs	a	b
1	3	4
2	1	2

例: 別のパッケージで SQLSTMT パッケージを使用する

例の概要: 別のパッケージで SQLSTMT パッケージを使用する

この例は、パッケージ SQLSTMT のインスタンスを別のパッケージで使用する方法を示しています。また、_NEW_演算子を使用して、FedSQL ステートメントによって参照されるテーブルの作成後まで SQLSTMT インスタンスの構築を遅らせる方法も示しています。参照テーブルが作成される前に SQLSTMT インスタンスが作成された場合、コンストラクターで FedSQL ステートメントの準備が失敗します。

コード例: 別のパッケージで SQLSTMT パッケージを使用する

```
package fibonnaci;
  declare int nMax;    /* maximum index in this series */

  method fibonnaci(int n);
    nMax = n;
  end;

  method output(char(100) tablename);
    declare int    n;    /* index of fib in Fibonacci series */
    declare double fib;  /* Fibonacci number */

    declare package sqlstmt stmt;
    /* note that stmt is not created */

    sqlexec('create table ' || tablename || '
      (n int, fib double)');

    /* Using _new_ allows delay of prepare of SQL statement until after */
    /* output table is created */

    stmt = _new_ sqlstmt('insert into ' || tablename || '
      values (?, ?)');

    /* fibonnaci(0) = 0 */
    stmt.setInteger(1, 0);    /* column n */
```

```
    stmt.setDouble(2, 0);    /* column fib */
    stmt.execute();

    /* fibonnaci(1) = 1 */
    stmt.setInteger(1, 1);   /* column n */
    stmt.setDouble(2, 2);    /* column fib */
    stmt.execute();

    first = 0;
    second = 1;

    do n = 2 to nMax;
      fib = first + second;
      stmt.setInteger(1, n);
      stmt.setDouble(2, fib);
      stmt.execute();

      first = second;
      second = fib;
    end;
  end;
endpackage;
```

次のコードブロックは、パッケージ FIBONACCI のインスタンスを作成し、フィボナッチインスタンスを使用してフィボナッチ数列の出力テーブルを生成します。

```
data _null_;
  method init();
    declare package fibonnaci fibseries(20);
    fibseries.output('fibdata');
  end;
enddata;
run;
```

例: 2 つのデータベースを使用してテーブルの値を更新する

例の概要: 2 つのデータベースを使用してテーブルの値を更新する

次の例は、SQLSTMT パッケージが、あるデータベースのテーブルを、第二のデータベースの別のテーブルの値に基づいて更新することを容易にする方法を示しています。プログラム例では、stmt1 は ORACLE テーブル db1.dataset1 からすべての x 列と y 列に対してクエリを実行します。次に、db1.dataset1 からの結果セット内の各行セットについて、stmt2 は、db1.dataset1 から読み取られた x 値と同じ x 列値を持つ BASE テーブル db2.dataset2 内の行を探し出します。次に、Stmt2 は、見つ

かった行の BASE テーブル db2.dataset2 の y 列の値を、db1.dataset1 から読み取られた y 値と同じになるように更新します。

コード例: 2 つのデータベースを使用してテーブルの値を更新する

```
libname db1 odbc user=XXXX pw=XXXX dsn=exadat;
libname db2 './base';

proc ds2;
data _null_;
  declare double x y;
  method run();
    dcl package sqlstmt stmt1('select x,y from db1.dataset1');
    dcl package sqlstmt stmt2('update db2.dataset2 set y=? where x=?',
      [y x]);

    stmt1.execute();
    stmt1.bindResults([x y]);
    do while (stmt1.fetch() = 0);
      stmt2.execute();
    end;
  end;
enddata;
run;
quit;
```

例: FedSQL ステートメントをハッシュパッケージに保存する

例の概要: FedSQL ステートメントをハッシュインスタンスに保存する

次の例は、ハッシュパッケージを使用して FedSQL ステートメントのセットを管理する方法を示しています。FedSQL ステートメントのセットは動的に割り当てられ、ハッシュパッケージに保存されます。ハッシュパッケージは、ハッシュパッケージから適切な FedSQL ステートメントを取得するために使用される整数キーによってインデックスが付けられます。

コード例: FedSQL ステートメントをハッシュパッケージに保存する

```
proc ds2;
/* Create 5 tables testdata1...testdata5.
 * Each table has 3 double columns (x,y,z) and no rows. */
data _null_;
  method init();
    declare int i;
    declare int rc;

    do i = 1 to 5;
      rc = sqlexec('create table testdata' || i ||
        '(x double, y double, z double)');
      if (rc ne 0) then put 'TEST FAILED';
    end;
  end;
enddata;
run;
quit;

proc ds2;
data _null_;

  /* Create an SQLstmt reference.
   * Note: does NOT create an sqlstmt instance. */
  declare package sqlstmt s;
  /* Create a hash instance. */
  declare package hash h();

  /* Hash key: sqlstmts are accessed by index 1..5. */
  declare int  i;
  /* Variables to bind to sqlstmt parameters. */
  declare double u v w;


  /* Create 5 sqlstmts to insert data in tables
   * testdata1...testdata5. Store the sqlstmts in hash
   * table h. */

  method init();
    declare int rc;

    h.definekey('i');    /* Key is index 1..5. */
    h.definedata('s');   /* Data is an sqlstmt reference. */
    h.definedone();

    /* Dynamically create 5 sqlstmt instances and add
     * each sqlstmt to hash table. */
    do i = 1 to 5;
```

```
    /* Dynamcially create an sqlstmt.
     * Variables [u v w] are bound to the parameters of
     *the sqlstmt. */
    s = _new_ sqlstmt('insert into testdata'
      || i || ' values (?,?,?)', [u v w]);

    /* Add the sqlstmt to hash h with key i. */
    rc = h.add();
    if (rc ne 0) then put 'TEST FAILED';
  end;
end;


/* Retrieve the sqlstmts from the hash table and execute
 * the sqlstmts to insert rows in the tables. */

method run();
  declare int j;
  declare int rc;

  /* For each of 5 data tables testdata1...testdata5, */
  /* insert 9 rows. */
  do j = 1 to 9;
    do i = 1 to 5;

      /* Find the sqlstmt in hash by index i. */
      rc = h.find();
      if (rc ne 0) then put 'TEST FAILED';

      /* Set values to insert into the table testdata i. */
      u = i;       /* data for 1st column (x) */
      v = -j;      /* data for 2nd column (y) */
      w = i*10+j;  /* data for 3rd column (z) */

      /* Execute the sqlstmt to insert the row. */
      rc = s.execute();
      if (rc ne 0) then put 'TEST FAILED';

      /* Explicitly close the result set to free resources.
       * If not explicitly closed, result set would be
       * automatically closed at next execute of the sqlstmt. */
      rc = s.closeResults();
      if (rc ne 0) then put 'TEST FAILED';

    end;
  end;
end;


/* Explicitly delete each sqlstmt instance.  Note if not
 * explicitly deleted, the sqlstmt instances would automatically
 * be deleted when the sqlstmt variables referencing the instances
 * were deleted. */

method term();
  declare int rc;
```

```
    do i = 1 to 5;
      rc = h.find();
      if (rc ne 0) then put 'TEST FAILED';
      s.delete();
    end;
  end;
enddata;
run;
quit;


proc print data=testdata1; quit;
proc print data=testdata2; quit;
proc print data=testdata3; quit;
proc print data=testdata4; quit;
proc print data=testdata5; quit;
```

出力例: FedSQL ステートメントをハッシュパッケージに保存する

アウトプット A22.2　テストデータテーブル

Obs	X	Y	Z
1	1	-1	11
2	1	-2	12
3	1	-3	13
4	1	-4	14
5	1	-5	15
6	1	-6	16
7	1	-7	17
8	1	-8	18
9	1	-9	19

Obs	X	Y	Z
1	2	-1	21
2	2	-2	22
3	2	-3	23
4	2	-4	24
5	2	-5	25
6	2	-6	26
7	2	-7	27
8	2	-8	28
9	2	-9	29

Obs	X	Y	Z
1	3	-1	31
2	3	-2	32
3	3	-3	33
4	3	-4	34
5	3	-5	35
6	3	-6	36
7	3	-7	37
8	3	-8	38
9	3	-9	39

Obs	X	Y	Z
1	4	-1	41
2	4	-2	42
3	4	-3	43
4	4	-4	44
5	4	-5	45
6	4	-6	46
7	4	-7	47
8	4	-8	48
9	4	-9	49

Obs	X	Y	Z
1	5	-1	51
2	5	-2	52
3	5	-3	53
4	5	-4	54
5	5	-5	55
6	5	-6	56
7	5	-7	57
8	5	-8	58
9	5	-9	59

例: MySQL 8 CHAR()データから末尾のスペースをフィルタリングする

例の概要: MySQL 8 CHAR()データから末尾のスペースをフィルタリングする

DS2 は、CHAR()値の末尾のスペースを 処理するために ANSI SQL 規則を使用します。MySQL 8 は、デフォルトで末尾のスペースを 重要な値として扱います。この違いは、DS2 で FedSQL ステートメントを使用するときに実行される文字比較に干渉します。この例は、TRIM 関数を使用して、SQLSTMT パッケージインスタンスの末尾のスペースをトリムする 方法を示しています。パッケージインスタンスは、2 つのテーブルから文字値をフィルタリングし、両方のテーブルに共通の値を含む 3 番目の一時テーブルを 作成します。

注: このタスクは、MySQL 8 を変更して、比較に照合順序を使用し、ANSI SQL 規則に従って末尾のスペースを 処理するようにした場合は必要ではありません。

コード例: MySQL 8 CHAR()データから末尾のスペースをフィルタリングする

```
/* Assign a libref to a MySQL 8 server */
libname mylib mysql server=sql8svr user=myuserid pw=mypwd db=database1;

/* Create two tables, Class and Names, in the database. In table Class, create two columns
called Name1 and ID and specify values for 5 rows. */
proc ds2;
data mylib.class (overwrite=yes);
 dcl char(8) name1;
 dcl char(30) id;
 method init();
 name1='JAMES'; id='10000000000000000000000000000000'; output;
 name1='GEORGE '; id='98765432109876543210987654321 0'; output;
 name1='JANET'; id='33'; output;
 name1='PAUL';id='44'; output;
 name1='JOHN'; id='12345678901234567890123456789 0'; output;
 end;
 enddata;
 run;
```

```
/* In table Names, create one column called Name that has four rows.
Include values from the Name1 column in table Class in the rows. */
data mylib.names (overwrite=yes);
 dcl char(8) name;
 method init();
 name='GEORGE  '; output;
 name='PAUL '; output;
 name='JOHN '; output;
 name='Ringo'; output;
 end;
 enddata;
run;
quit;

/* Create a temporary table that includes an ID column and defines
an instance of the SQLSMT package that submits a FedSQL SELECT statement
to retrieve the ID and Name1 values from table Class. Include a WHERE clause
that compares values in the Name1 column to trimmed instances of variable ?.
Specify a Name column as a constructor argument.  */
proc ds2;
data _null_;
 dcl decimal(30) id;
 dcl package sqlstmt fsqlclass('select id from mylib.class where name1= trim(?)', [name]);

/* Define a return code variable. */
 method run();
  dcl int rc;

/* Read table Names into memory. */
  set mylib.names;

/* Sort the values in the Name column. */
  by name;

/* Execute the Fsqlclass package instance on the in-memory data to apply the TRIM
function
 and WHERE clause on column Name. Capture any error codes. */
  rc=fsqlclass.execute();
  if (rc) then put 'ERROR: Error executing';

/* Bind the ID column to the results returned by Fsqlclass,
capturing any return codes. */
  rc =fsqlclass.bindResults([id]);
  if (rc) then put 'ERROR: Error binding results';

/* Fetch the results and print them to the SAS log. */
  rc = fsqlclass.fetch();
  if (rc = 0) then put name= id=;
 end;
 enddata;
 run;
 quit;
```

出力例: MySQL 8 CHAR()データから末尾のスペースをフィルタリングする

```
name=GEORGE   id=98765432109876543210987654321O
name=JOHN     id=12345678901234567890123456789O
name=PAUL     id=44
```

例: HTTP および JSON パッケージを使用して REST 形式のファイルを抽出および解析する

例の概要: HTTP および JSON パッケージを使用して REST 形式のファイルを抽出および解析する

この例では、HTTP および JSON パッケージを使用して、REST 形式のファイルから情報を抽出します。

コード例: HTTP および JSON パッケージを使用して REST 形式のファイルを抽出および解析する

```
/* DS2 program that uses a REST-based API.    */
/* Uses http package for API calls           */
/* and the JSON package to parse the result.  */

proc ds2;
thread work.json / overwrite = yes;
   /* Global package references            */
   dcl package json j();

   /* Keeping these variables for output    */
   dcl double timest having format datetime20.;
   dcl varchar(50) name;
```

```
dcl int lat lng bikes free;

/* these are temp variables */
dcl varchar(65534) character set utf8 response;
dcl int rc;
drop response rc;

method parseMessages();
 dcl int tokenType parseFlags;
 dcl nvarchar(128) token;
 rc=0;
 * iterate over all message entries;
 do while (rc=0);
  j.getNextToken( rc, token, tokenType, parseFlags);

    if (token eq 'timestamp') then
   do;
    j.getNextToken( rc, token, tokenType, parseFlags);
    timest=inputn(token,'anydtdtm26.');
   end;

  if (token eq 'name') then
   do;
    j.getNextToken( rc, token, tokenType, parseFlags);
    name=token;
   end;

  if (token eq 'lat') then
   do;
    j.getNextToken( rc, token, tokenType, parseFlags);
    lat=token;
   end;

  if (token eq 'lng') then
   do;
    j.getNextToken( rc, token, tokenType, parseFlags);
    lng=token;
   end;

  if (token eq 'bikes') then
   do;
    j.getNextToken( rc, token, tokenType, parseFlags);
    bikes=token;
   end;

   if (token eq 'free') then
   do;
    j.getNextToken( rc, token, tokenType, parseFlags);
    free=token;
    output;
   end;
 end;
 return;
end;

method init();
```

```
    dcl package http webQuery();
    dcl int rc tokenType parseFlags;
    dcl nvarchar(128) token;
    dcl integer i rc;

    /* create a GET call to the API               */
    /* 'sas_programming' covers all SAS programming */
    /* topics from communities                    */

    webQuery.createGetMethod(
      'http://api.citybik.es/capital-bikeshare.json');

    /* execute the GET                            */

    webQuery.executeMethod();

    /* retrieve the response body as a string     */

    webQuery.getResponseBodyAsString(response, rc);
    put response;
    rc = j.createParser( response );
    do while (rc = 0);
     j.getNextToken( rc, token, tokenType, parseFlags);
     if (token = '{') then
       parseMessages();
    end;
   end;

  method term();
    rc = j.destroyParser();
  end;
endthread;

data bikes (overwrite=yes);
declare thread work.json json;
method run();
  set from json threads=3;
end;
enddata;
run;
quit;
```

出力例: HTTP および JSON パッケージを使用して REST 形式のファイルを抽出および解析する

SAS ログに書き込まれる行の一部を次に示します。

```
[
 {
   "bikes": 10,
   "name": "Jones Branch & Westbranch Dr",
   "idx": 0,
   "lat":
38931911,
   "timestamp": "2017-09-13T18:21:46.986000Z",
   "lng": -77219261,
   "id": 0,

  "free": 3,
   "number": 430
 },
 {
   "bikes": 7,
   "name": "Town Center Pkwy &
Bowman Towne Dr",
   "idx": 1,
   "lat": 38962524,
   "timestamp":
"2017-09-13T18:21:46.979000Z",
   "lng": -77361902,
   "id": 1,
   "free": 8,

"number": 434
 },
 {
   "bikes": 17,
   "name": "Potomac & M St NW",
   "idx": 2,

"lat": 38905368,
   "timestamp": "2017-09-13T18:21:47.384000Z",
   "lng": -77065149,

"id": 2,
   "free": 1,
   "number": 473
 },
 {
   "bikes": 7,
   "name": "22nd St &
Constitution Ave NW",
   "idx": 3,
   "lat": 38892441,
   "timestamp":
"2017-09-13T18:21:46.931000Z",
   "lng": -77048947,
   "id": 3,
   "free": 15,

"number": 443
 },
 {
   "bikes": 2,
   "name": "18th & Eads St.",
   "idx": 4,

"lat": 38857250,
   "timestamp": "2017-09-13T18:21:47.160000Z",
   "lng": -77053320,

"id": 4,
   "free": 8,
   "number": 2
 },
```

例: HTTP から JSON テキストを読み取り、SAS データセットを作成する

例の概要: HTTP から JSON テキストを読み取り、SAS データセットを作成する

この例では、HTTP パッケージを使用して、http://httpbin.org/post から顧客が提供した JSON テキストを読み取ります。次に、この例では JSON パッケージを使用して、応答を SAS データセットに解析します。

コード例: HTTP から JSON テキストを読み取り、SAS データセットを作成する

```
proc ds2;
 package myhttp /overwrite=yes;
  method post( nvarchar(8192) url,
               nvarchar(67108864) payload,
               in_out nvarchar respbody,
               in_out int hstat, in_out int rc );
   dcl package http h();
   dcl package logger logr( 'App.TableServices.d2pkg.HTTP' );
   dcl nvarchar(8192) respHdrs;
   rc = h.createPostMethod( url );
   if rc ne 0 then goto Exit;
   rc = h.setRequestContentType( 'application/json' );
   if rc ne 0 then goto Exit;
   rc = h.addRequestHeader( 'Accept', 'application/json' );
   if rc ne 0 then goto Exit;
   rc = h.setRequestBodyAsString( payload );
   if rc ne 0 then goto Exit;
   rc = h.executeMethod();
   if rc ne 0 then goto Exit;
   hstat = h.getStatusCode();
   h.getResponseHeadersAsString( respHdrs, rc );
   logr.log( 3, '--- Beginning of response headers ---' );
   logr.log( 3, respHdrs );
   logr.log( 3, '--- End of response headers ----------' );
   if hstat lt 400 then h.getResponseBodyAsString( respbody, rc );
   else respbody = '';
   Exit:
```

```
   h.delete();
 end;
endpackage;
run;
quit;

proc ds2;
 ds2_options sas;

 data work.Shots(keep=(ShotId text PName DPO _count));
  dcl package json j();
  dcl int ShotId;
  dcl varchar(256) text PName;
  dcl int DPO _count;

  method init();
   dcl package myhttp p();
   dcl nvarchar(10000) body;
   dcl nvarchar(100000) response;
   dcl nvarchar(256) token;
   dcl int rc hstat tokenType parseFlags;
   rc = 0; hstat = 0;
   body =
      '{"_count":882, "Date":"2016-02-11",
        "Shots":[{"ShotId":14162,"text":"E","PName":"NG",
                  "DPO":1455209,"_count":98},
                 {"ShotId":14163,"text":"EN","PName":"NGG",
                  "DPO":1455210,"_count":784} ],
        "Defs": [{"DT":"P","osure":"SHEET","rioType":"EUR",
                  "DId":"e2938c74","_count":98},
                 {"DT":"N","osure":"VITY","SCat":"NGE",
                  "rioType":"EUR","DId":"faaa8b91","_count":784}],
        "VIds": [{"VId":"e2938c74"}, {"VId":"faaa8b91"} ] }';

   *Bounce the JSON payload off of a test server.;
   p.post( 'http://httpbin.org/post', body, response, hstat, rc );
   if ( hstat ~= 200 ) then do;
      put rc= hstat=;
      put 'Error: JSON payload did not bouce off of the server.';
      return;
   end;
   put response=;
   rc = j.createParser( response );

   * Look for a JSON object in the response.;
   do while ( rc < 101 );
      j.getNextToken( rc, token, tokenType, parseFlags );
      *put token=;
      if ( j.isLabel(tokenType, parseFlags) and
           (token EQ 'json') ) then
         goto LookForShots;
   end;
   put 'Error: no "json" in response.';
   return;

  LookForShots:
```

```
do while ( rc < 101 );
   j.getNextToken( rc, token, tokenType, parseFlags );
   *put token=;
   if ( j.isLabel(tokenType, parseFlags) and
        (token EQ 'Shots') ) then
      goto ShotsArray;
end;
put 'Error: "Shots" not found.';
return;

ShotsArray:
j.getNextToken( rc, token, tokenType, parseFlags );
if ( not j.isLeftBracket( tokenType ) ) then do;
   put 'Error: Expecting Shots array.';
   return;
end;

*put '----- Shots array ----------';
j.getNextToken( rc, token, tokenType, parseFlags );
*put token=;
do while ( j.isLeftBrace( tokenType ) );
   *put '-------OBJECT---------';
   j.getNextToken( rc, token, tokenType, parseFlags );
   *put token=;
   do while ( not j.isRightBrace( tokenType ) );
      if ( token EQ 'ShotId' ) then do;
         j.getNextToken( rc, token, tokenType, parseFlags );
         ShotId = token;
      end;
      else if ( token EQ 'text' ) then do;
         j.getNextToken( rc, token, tokenType, parseFlags );
         text = token;
      end;
      else if ( token EQ 'PName' ) then do;
         j.getNextToken( rc, token, tokenType, parseFlags );
         PName = token;
      end;
      else if ( token EQ 'DPO' ) then do;
         j.getNextToken( rc, token, tokenType, parseFlags );
         DPO = token;
      end;
      else if ( token EQ '_count' ) then do;
         j.getNextToken( rc, token, tokenType, parseFlags );
         _count = token;
      end;
      else do;
         put 'Error: Unexpected column.';
         return;
      end;
      j.getNextToken( rc, token, tokenType, parseFlags );

   end; * End of column loop.;
   output;
   j.getNextToken( rc, token, tokenType, parseFlags );

end; * End of row loop.;
```

```
   rc = j.destroyParser();
 end; * End method init;
 enddata;
run;
quit; * End of PROC DS2 step;

proc print data=work.Shots;
run;
```

出力例: HTTP から JSON テキストを読み取り、SAS データセットを作成する

The SAS System

Obs	ShotId	text	PName	DPO	_count
1	14162	E	NG	1455209	98
2	14163	EN	NGG	1455210	784