THE POWER TO KNOW.

SAS® 9.3 関数とCALLルーチン: リファレンス

SAS® ドキュメント

このマニュアルの正式な書籍名: SAS Institute Inc. 2011. *SAS® 9.3 関数と CALL ルーチン: リファレンス*. Cary, NC: SAS Institute Inc.

SAS® 9.3 関数と CALL ルーチン: リファレンス

ISBN 978–1–60764–901–4 (電子書籍)
ISBN 978–1–60764–901–4

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

ISBN 978–1–60764–901–4
初刷, 2011 July

ISBN 978–1–60764–901–4
初刷, 2011 July

SAS® Publishing provides a complete selection of books and electronic products to help customers use SAS software to its fullest potential. For more information about our e-books, e-learning products, CDs, and hard-copy books, visit the SAS Publishing Web site at **support.sas.com/publishing** or call 1-800-727-3228.

目次

本書について

SAS 言語の構文規則

SAS 言語の構文規則の概要

SAS では、SAS 言語要素の構文ドキュメントに共通の規則を使用しています。これらの規則により、SAS 構文の構成要素を簡単に識別できます。規則は、次の項目に分類されます。

- 構文の構成要素
- スタイル規則
- 特殊文字
- SAS ライブラリと外部ファイルの参照

構文のコンポーネント

言語要素の多くでは、その構文の構成要素はキーワードと引数から構成されます。 キーワードのみ必要な言語要素もあります。 また、キーワードに等号(=)が続く言語要素もあります。

キーワード
プログラムの作成時に使用する SAS 言語要素名です。キーワードはリテラルであり、通常、構文の先頭の単語です。CALL ルーチンでは、最初の 2 つの単語がキーワードです。

次の SAS 構文の例では、構文の最初の単語がキーワードです。

CHAR (*string, position*)

CALL RANBIN (*seed, n, p, x*);

ALTER (*alter-password*)

BEST *w.*

REMOVE <*data-set-name*>

次の例では、CALL ルーチンの最初の 2 つの単語がキーワードです。

CALL RANBIN(*seed*, *n*, *p*, *x*)

引数なしで 1 つのキーワードから構成される SAS ステートメント構文もあります。

DO;
... SAS code ...

END;

2 つのキーワード値のいずれか 1 つの指定が必要なシステムオプションもあります。

DUPLEX | **NODUPLEX**

引数
数値定数、文字定数、変数、式のいずれかです。引数は、キーワードに続くか、キーワードの後ろの等号に続きます。SAS では、引数を使用して、言語要素を処理します。引数が必須の場合もオプションの場合もあります。構文では、オプションの引数にはかぎかっこが付けられます。

次の例では、*string* と *position* がキーワード CHAR に続きます。これらの引数は、CHAR 関数の必須引数です。

CHAR (*string, position*)

引数ごとに値が指定されます。次の例の SAS コードでは、引数 *string* の値として'summer'、引数 *position* の値として 4 が指定されています。

```
x=char('summer', 4);
```

次の例では、*string* と *substring* は必須引数ですが、*modifiers* と *startpos* はオプションの引数です。

FIND(*string, substring* <,*modifiers*> <,*startpos*>

注: 通常、SAS ドキュメントのサンプルコードは、小文字の固定幅フォントを使用して表記されます。コードの作成には、大文字も、小文字も、大文字と小文字の両方も使用できます。

スタイル規則

SAS 構文の説明に使用されるスタイル規則には、大文字太字、大文字、斜体の規則も含まれます。

大文字太字
関数名やステートメント名などの SAS キーワードを示します。次の例では、キーワード ERROR の表記には大文字太字が使用されています。

ERROR<*message*>;

大文字
リテラルの引数を示します。

次の CMPMODEL=システムオプションの例では、BOTH、CATALOG、XML がリテラルです。

CMPMODEL = BOTH | CATALOG | XML

斜体
ユーザー指定の引数または値を示します。斜体表記の項目は、ユーザー指定値であり、次のいずれかを表します。

- 非リテラルの引数。次の LINK ステートメントの例では、引数 *label* はユーザー指定値であるため、斜体で表記されています。

  LINK *label*;

- 引数に割り当てられる非リテラル値。

  次の FORMAT ステートメントの例では、引数 DEFAULT に変数の *default-format* が割り当てられます。

  FORMAT = *variable-1* <, ..., *variable-nformat*><DEFAULT = *default-format*>;

斜体表記の項目は、選択可能な引数リストの総称でもあります(*attribute-list* など)。複数の斜体表記の項目が使用される場合、項目は *item-1, ..., item-n* という形式で表記されます。

特殊文字

SAS 言語要素の構文には、次の特殊文字も使用されます。

=
等号は、一部の言語要素(システムオプションなど)のリテラル値を示します。

次の MAPS システムオプションの例では、等号は MAPS の値を設定します。

MAPS = *location-of-maps*

< >
かぎかっこはオプションの引数を示します。かぎかっこ付きでない引数は必須引数です。

次の CAT 関数の例では、少なくとも項目が 1 つ必要です。

CAT (*item-1 <, ..., item-n>*)

|
縦棒は、値グループから 1 つの値を選択できることを示します。縦棒で区切られている値は、相互排他です。

次の CMPMODEL=システムオプションの例では、属性を 1 つのみ選択できます。

CMPMODEL = BOTH | CATALOG | XML

...
省略記号は、省略記号に続く引数や引数グループの繰り返しを示します。省略記号とその後の引数にかぎかっこが付けられている場合、その引数はオプションです。

次の CAT 関数の例では、省略記号はオプションの項目を複数指定できることを示しています。

CAT (*item-1 <, ..., item-n>*)

'*value*' or "*value*"
単一引用符や二重引用符付きの引数は、その値も単一引用符または二重引用符を付ける必要があることを示します。

次の FOOTNOTE ステートメントの例では、引数 *text* には引用符が付けられています。

FOOTNOTE <*n*> <*ods-format-options* '*text*' | "*text*">;

;
セミコロンは、ステートメントまたは CALL ルーチンの終わりを示します。

次の例では、それぞれのステートメントはセミコロンで終了しています。

```
data namegame; length color name $8; color = 'black'; name = 'jack'; game = trim(color)
|| name; run;
```

SAS ライブラリと外部ファイルへの参照

多くの SAS ステートメントなどの言語要素では、SAS ライブラリと外部ファイルを参照します。論理名(ライブラリ参照名またはファイル参照名)から参照を作

成するのか、引用符付きの物理ファイル名を使用するかを選択できます。 論理名を使用する場合、通常、関連付けに SAS ステートメント(LIBNAME または FILENAME)を使用するのか、動作環境のコントロール言語を使用するのかを選択します。 複数の方法を使用して、SAS ライブラリと外部ファイルを参照できます。動作環境によっては使用できない方法があります。

SAS ドキュメントでは、外部ファイルを使用する例には斜体のフレーズ *file-specification* を使用します。 また、SAS ライブラリを使用する例には斜体フレーズ *SAS-library* を使用します。 *SAS-library* は引用符付きであることに注意してください。

infile *file-specification* obs = 100;
libname *libref 'SAS-library'*;

SAS 9.3 関数と CALL ルーチンの新機能

概要

SAS 関数と CALL ルーチンは、別のドキュメントとしてパブリッシュされるようになりました。現在 *SAS Language Reference: Dictionary* には含まれていません。詳細については、“SAS Language Reference: Dictionary の変更” (xi ページ)を参照してください。

DATA ステップで Web サービスを呼び出す機能は新機能です。この機能用に 6 つの新しい SOAPxxx 関数が追加されました。さらに、その他いくつかの新しい関数が追加され、既存の関数が拡張されています。

新しい関数と CALL ルーチン

新しく追加された関数と CALL ルーチンを次に示します。

CALL RANCOMB (p. 211)
引数の値を置換し、*n* 個の値のうち *k* 個のランダムな組み合わせを返します。

EFFRATE (p. 384)
実効年利を返します。

MVALID (p. 659)
SAS メンバ名として使用する文字列の有効性を確認します。

NOMRATE (p. 666)
名目年利を返します。

SAVINGS (p. 822)
変動金利を使用して定期預金の残高を返します。

SOAPWEB (p. 841)
基本 Web 認証を使用して Web サービスを呼び出します。認証情報は引数で指定します。

SOAPWEBMETA (p. 843)
基本 Web 認証を使用して Web サービスを呼び出します。認証ドメインの認証情報はメタデータから取得されます。

SOAPWIPSERVICE (p. 845)
WS セキュリティ認証を使用して SAS 登録サービスを呼び出します。認証情報は引数で指定します。

SOAPWIPSRS (p. 847)
WS セキュリティ認証を使用して SAS 登録 Web サービスを呼び出します。認証情報は引数で指定します。セキュリティトークンサービスの検索方法を確認するためにレジストリサービスが直接呼び出されます。

SOAPWS (p. 849)
WS セキュリティ認証を使用して Web サービスを呼び出します。認証情報は引数で指定します。

SOAPWSMETA (p. 851)
WS セキュリティ認証を使用して Web サービスを呼び出します。認証ドメインの認証情報はメタデータから取得されます。

SQUANTILE (p. 856)
右側確率(SDF)を指定した場合に分布から分位点を返します。

SYSEXIST (p. 877)
動作環境変数が存在するかどうかを返します。

TIMEVALUE (p. 888)
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。

既存の関数の拡張

既存の関数は、次のように拡張されました。

- GENPOISSON 分布と TWEEDIE 分布が次の関数に追加されました。
  - CDF
  - PDF
  - SDF
  - LOGCDF
  - LOGPDF
  - LOGSDF
  - QUANTILE
- 新しい引数 *seasonality* が INTCYCLE、INTINDEX および INTSEAS 関数に追加されました。*seasonality* 引数では、日付と時間の周期をより柔軟に操作できます。詳細については、“INTCYCLE 関数” (552 ページ)、“INTINDEX 関数” (561 ページ)、および“INTSEAS 関数” (575 ページ)を参照してください。
- 年齢を計算する新しいオプションが YRDIF 関数に追加されました。詳細については、“YRDIF 関数” (957 ページ)を参照してください。
- SAS セッションエンコーディングと UTF-8 エンコーディングの説明が URLDECODE 関数と URLENCODE 関数に追加されました。詳細については、“URLDECODE 関数” (903 ページ)および“URLENCODE 関数” (905 ページ)を参照してください。
- GETOPTION 関数では、次のオプションを使用できます。
  - DEFAULTVALUE オプションは、システムオプションのデフォルトの出荷値を取得します。この値を使用して、システムオプションをデフォルトにリセットできます。
  - HEXVALUE オプションは、システムオプション値を 16 進値で返します。

- LOGNUMBERFORMAT オプションは、システムオプションの数値を返します。使用される句読点は、言語ロケールに依存します。
- STARTUPVALUE オプションは、コマンドラインまたは構成ファイルのいずれかで SAS の起動に使用されたシステムオプション値を返します。

既存の関数のドキュメント

次の 5 つの関数のドキュメントは、*SAS/ETS User's Guide* から *SAS 関数と CALL ルーチン: リファレンス*に移動されました。

CUMIPMT (p. 341)
開始日と終了日の間に支払われたローンの累積利息を返します。

CUMPRINC (p. 342)
開始日と終了日の間に支払われたローンの累積元本を返します。

IPMT (p. 584)
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。

PMT (p. 724)
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定期的支払いを返します。

PPMT (p. 727)
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。

SAS Language Reference: Dictionary の変更

9.3 より前は、このドキュメントは *SAS Language Reference: Dictionary* に含まれていました。9.3 以降では、*SAS Language Reference: Dictionary* は 7 つのドキュメントに分割されています。

- *SAS データセットオプション: リファレンス*
- *SAS 出力形式と入力形式: リファレンス*
- *SAS Functions and CALL Routines: Reference*
- *SAS ステートメント: リファレンス*
- *SAS システムオプション: リファレンス*
- *SAS コンポーネントオブジェクト: リファレンス* (ハッシュ、ハッシュ反復子および Java オブジェクトのドキュメントを含む)
- *Base SAS Utilities: リファレンス* (SAS DATA ステップデバッガおよび SAS ユーティリティマクロ%DS2CSV を含む)

推奨資料

このタイトルに関連した推奨される参考資料のリストを次に示します。

- *Base SAS Glossary*
- *Base SAS プロシジャガイド*
- *UNIX 版 SAS*
- *Windows 版 SAS*
- *z/OS 版 SAS*
- *SAS データセットオプション: リファレンス*
- *SAS 出力形式と入力形式: リファレンス*
- *SAS 言語リファレンス: 解説編*
- *SAS Metadata LIBNAME Engine: User's Guide*
- *SAS 各国語サポート(NLS): リファレンスガイド*
- *SAS Output Delivery System: ユーザーガイド*
- *SAS Scalable Performance Data Engine: リファレンス*
- *SAS ステートメント: リファレンス*
- *SAS システムオプション: リファレンス*

推奨される SAS Press の参考資料を次に示します。

- *An Array of Challenges - Test Your SAS Skills*
- *Cody's Data Cleaning Techniques Using SAS*
- *Combining and Modifying SAS Data Sets: Examples*
- *Debugging SAS Programs: A Handbook of Tools and Techniques*
- *SAS Functions by Example*
- *SAS Guide to Report Writing: Examples*
- *Health Care Data and SAS*
- *The Little SAS Book: A Primer*
- *Output Delivery System: The Basics and Beyond*
- *SAS Programming by Example*
- *Quick Results with the Output Delivery System*

- *Step-by-Step Programming with Base SAS Software*
- *Using the SAS Windowing Environment: A Quick Tutorial*
- *The SAS Workbook*
- *SAS XML LIBNAME Engine: User's Guide*

SAS の刊行物の総一覧については、support.sas.com/bookstore にてご確認ください。必要な書籍についてのご質問は、下記までお寄せください。

SAS Publishing Sales
SAS Campus Drive
Cary, NC 27513-2414
電話: 1-800-727-3228
ファクシミリ: 1-919-677-8166
電子メール: sasbook@sas.com
Web アドレス: support.sas.com/bookstore

1 章

SAS 関数と CALL ルーチン

関数と CALL ルーチンについて

関数の定義

SAS 関数は SAS プログラミング言語を構成する要素で、引数を受け取り、算術演算やその他の演算を実行し、値を返します。関数では、結果として数値または文字を返すことができます。返される値は割り当てステートメントや式の他の場所で使用できます。SAS には多数の関数が用意されていますが、独自の関数も作成できます。

Base SAS ソフトウェアでは、DATA ステップのプログラミングステートメント、WHERE 式、マクロ言語ステートメント、PROC REPORT、Structured Query Language(SQL)の中で SAS 関数を使用できます。

統計プロシジャの一部でも SAS 関数を使用します。さらに、DATA ステップで使用できる関数を提供する他の SAS ソフトウェア製品もあります。これらの関数の詳細については、個別の SAS ソフトウェア製品に関するドキュメントを参照してください。

CALL ルーチンの定義

CALL ルーチンは変数値を変更したり、その他のシステム関数を実行したりします。CALL ルーチンは関数に似ていますが、割り当てステートメントや式の中で使用できない点で関数と異なります。

SAS の CALL ルーチンは、すべて CALL ステートメントで起動します。つまり、CALL ステートメント内のキーワード CALL の後でルーチン名を指定する必要があります。

構文

関数の構文

関数の構文は次のいずれかの形式です。

function-name (*argument-1*<, …*argument-n*>)

function-name (OF *variable-list*)

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

function-name
関数名を指定します。

引数
変数名、定数、または SAS 式(他の関数を含む)を指定します。SAS で使用できる引数の数と種類は個別の関数と合わ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: 引数の値が無効の場合(たとえば、欠損している、または定められた範囲外である)、 SAS は引数が無効であることを示すメモを口グに書き込み、_ERROR_
を 1 に設定して、結果を欠損値に設定します。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));
- dollars=int(cash);
- if sum(cash,credit)>1000 then put 'Goal reached';

variable-list
個別の変数名を含む SAS 変数リスト(どの形式でも可)を使用します。変数リストを複数指定する場合、空白で区切るか、カンマと追加の 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);

サンプル: 次に 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{*}
現在定義済みの配列名を指定します。配列にサブスクリプトとしてアスタリスクを付けて指定すると、配列の各要素が個別の引数として扱われます。

OF 演算子は一時配列を受け入れるよう拡張されました。ほとんどの SAS 関数で通常の変数配列と同様に一時配列を OF リストで使用できますが、いくつかの制限があります。

参照項目: 制限のリストについては、“一時配列で OF 演算子を使用する” (5 ページ)を参照してください。

CALL ルーチンの構文

CALL ルーチンの構文は次のいずれかの形式です。

CALL *routine-name* (*argument-1*<, *...argument-n*>);

CALL *routine-name* (OF *variable-list*);

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

routine-name
SAS の CALL ルーチン名を指定します。

引数
変数名、定数、または SAS 式、外部モジュール名、配列参照、関数を指定します。引数が複数のときはカンマで区切ります。使用できる引数の数と種類は、個別の CALL ルーチンと合わせて辞書セクションに記載しています。次に例を示します。

```
•call prxsubstr(prx,string,position);

•call prxchange('/old/new',1+k,trim(string),result,length);

•call set(dsid);

•call ranbin(Seed_1,n,p,X1);

•call label(abc[j],lab);

•call cats(result,'abc',123);
```

variable-list
変数名を含む SAS 変数リスト(どの形式でも可)を使用します。変数リストを複数指定する場合、空白で区切るか、カンマと追加の OF で区切ります。

```
•call cats(inventory, of y1-y15, of z1-z15);

•call catt(of item17-item23 pack17-pack23);
```

関数と CALL ルーチンの使用

関数の引数に影響する制限

引数の値が無効の場合、SAS は引数が無効であることを示すメモまたはエラーメッセージをログに書き込み、結果を欠損値に設定します。関数の引数に関する一般的な制限は次のとおりです。

- 関数の中には、一定の範囲内で引数を指定する必要があるものがあります。たとえば、LOG 関数の引数は 0 より大きい必要があります。
- 数値引数に欠損値があると、関数の多くは SAS ログにメモを書き込んで欠損値を返します。例外としては、一部の記述統計量関数と財務関数があります。

- 関数の中には、EXP 関数のように、受け入れる引数の範囲がプラットフォーム依存のものがあります。

一時配列で OF 演算子を使用する

一時配列では OF 演算子を使用できます。OF 演算子を使用すると、数が変化するパラメータを含む引数をとるほとんどの関数に一時配列を渡せます。一部の関数では、通常の変数配列で OF リストの一時配列を使用するのと同様に、一時配列の OF リストを使用できます。

一時配列の使用には、いくつかの制限があります。これらの制限は、例の後にリストで掲載しています。

一時配列の使用方法の例を次に示します。

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

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

ログ 1.1　一時配列の使用例のログ出力

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

一時配列の OF リストは次の制限を受けます。

- 配列インデックスとして使用できません。
- 通常の変数配列と同様、パラメータ数が OF リストの要素数と一致する関数で使用できます。
- 数が変化するパラメータをとることができる関数で使用できます。
- DIF、LAG、SUBSTR、LENGTH、TRIM、MISSING 関数では使用できません。また、VLENGTH 関数などの変数情報関数では一切使用できません。

対象変数の特性

文字関数には、デフォルトの長さが 200 バイトの対象変数を結果として生成する関数があります。数値の対象変数はデフォルトの長さが 8 バイトです。デフォルトの対象変数の長さが適用されない文字関数を次の表に示します。これらの関数では、返される引数の長さが第 1 引数の長さから取得されます。

表 1.1 第 1 引数の長さによって返される引数が決まる関数

	関数
COMPBL	RIGHT
COMPRESS	STRIP
DEQUOTE	SUBSTR
INPUTC	SUBSTRN
LEFT	TRANSLATE
LOWCASE	TRIM
PUTC	TRIMN
REVERSE	UPCASE

対象変数の長さが割り当てられていない場合の対象変数の長さを次の関数リストに示します。

BYTE
対象変数に割り当てられるデフォルトの長さは 1 です。

INPUT
入力形式の幅によって対象変数の長さが決まります。

PUT
形式の幅によって対象変数の長さが決まります。

VTYPE
対象変数に割り当てられるデフォルトの長さは 1 です。

VTYPEX
対象変数に割り当てられるデフォルトの長さは 1 です。

記述統計量関数の注

SAS には記述統計量を返す関数が用意されています。これらの関数の多くは MEANS プロシジャと UNIVARIATE プロシジャによって生成される統計量に対応します。各統計量の計算手法は *Base SAS プロシジャガイド*の初等統計量プロシジャのセクションで説明します。SAS は引数の非欠損値の記述統計量を計算します。

財務関数の注

財務関数の種類

SAS には財務計算を行う関数グループが用意されています。関数は次の種類に分類されます。

表 1.2 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修正デュレーションを計算します。
	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。
パラメータ計算	COMPOUND	複利パラメータを計算します。
	MORT	割賦返済パラメータを計算します。
内部利益率	INTRR、IRR	内部利益率を計算します。
正味現在価値および正味将来価値	NETPV、NPV	正味現在価値および正味将来価値を計算します。
	SAVING	ある期間預金した場合の将来価値を計算します。
減価償却	DACCxx	指定した期間までの減価償却累積額を計算します。
	DEPxxx	ある 1 期間の減価償却額を計算します。
価格	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。
	BLACKPLPRC、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価格とプット価格を計算します。

価格関数の使用

ある金融商品の理論的な市場価額(価格)を計算するのに使うのが価格モデルです。この計算された価額を時価評価(MTM)額と呼びます。通常、価格関数は次の形式をとります。

price = *function*(*rf*1, *rf*2, *rf*3, ...)

価格関数の *rf1*、*rf2* および *rf3* は、利率や外国為替レートなどのリスクファクタです。MTM 額の計算に使用するリスクファクタの具体的な値がベースケース値です。ベースケース値のセットがベースケースの市場状態です。

MTM 額が決まったら、リスクファクタのベースケース値(*rf1*、*rf2* および *rf3*)を使って次のタスクを実行できます。

- ベースケース値を特定の値に設定してシナリオ分析を実行
- ベースケース値をさまざまな値に設定して利益/損失曲線分析および利益/損失表面分析を実行
- ベースケース値を他の値に自動設定し、感度を計算(リスクファクタのデルタ値およびガンマ値を計算)
- ベースケース値を攪乱させ、起こりうる市場状態を多数作り出して、ありうる将来の価格を多数計算し、シミュレーション分析を実行。モンテカルロ分析については、数式モデルおよびコピュラ法を使ってリスクファクタの値を計算

価格関数とその説明の一覧については、“財務関数の種類” (6 ページ)を参照してください。

マクロ関数内で DATA ステップ関数を使用する

マクロ関数%SYSFUNC および%QSYSFUNC は DATA ステップ関数のほとんどを呼び出し、マクロ機能でテキストを生成できます。%SYSFUNC と%QSYSFUNC には違いが 1 つあります。%QSYSFUNC は特殊文字と二ーモニックをマスクしますが、%SYSFUNC はマスクしません。%QSYSFUNC 関数と%SYSFUNC 関数の詳細については、*SAS マクロ言語: リファレンス*を参照してください。

%SYSFUNC の引数は、次の例に示すように、1 つの DATA ステップ関数と任意指定の形式です。

```
%sysfunc(date(),worddate.)
%sysfunc(attrn(&dsid,NOBS))
```

DATA ステップ関数は%SYSFUNC 内でネストできません。ただし、DATA ステップ関数を呼び出す%SYSFUNC 関数はネストできます。次に例を示します。

```
%sysfunc(compress(%sysfunc(getoption(sasautos)),
%str(%)%(%')));
```

%SYSFUNC に含まれる DATA ステップ関数の引数はすべてカンマで区切る必要があります。OF を先頭とする引数リストは使用できません。

%SYSFUNC はマクロ関数のため、DATA ステップ関数の場合とは異なり、文字値を引用符で囲む必要はありません。たとえば、OPEN 関数を単独で使用するときは引数を引用符で囲みますが、%SYSFUNC 内で使用するときは引用符を必要としません。

```
dsid=open(″sasuser.houses″,″i″);
dsid=open(″&mydata″,″&mode″);
```

```
%let dsid=%sysfunc(open(sasuser.houses,i));
%let dsid=%sysfunc(open(&mydata,&mode));
```

CALL ルーチンと%SYSCALL マクロステートメントを使用する

%SYSCALL マクロステートメントが CALL ルーチンを起動するとき、引数であるマクロ変数の値はそれぞれ解決されずに取得されて、CALL ルーチンに渡されます。CALL ルーチンが終了すると、各引数の値が対応するマクロ変数に書き戻されます。%SYSCALL で エラーが発生すると、CALL ルーチンはマクロ変数の値を更新せずに実行を停止し、エラーメッセージをログに書き込みます。

%SYSCALL が CALL ルーチンを起動するとき、引数の値は解決されずに CALL ルーチンに渡されます。未解決の引数の値はマクロ引用関数で引用されている可能性があり、デルタ文字を含んでいる可能性があります。引用形式の引数の値は、文字値の比較時に予測できない結果を招く可能性があります。CALL ルーチンの一部は、%SYSCALL からの呼び出し時に引数の引用を解除し、引用が解除された値を返します。引数の引用を解除する必要がない CALL ルーチンもあります。%SYSCALL からの呼び出し時に引数の引用を解除する CALL ルーチンのリストを次に示します。

- “CALL COMPCOST ルーチン” (162 ページ)
- “LEXCOMB 関数” (607 ページ)
- “LEXPERK 関数” (611 ページ)
- “CALL LEXPERM ルーチン” (183 ページ)
- “CALL PRXCHANGE ルーチン” (194 ページ)
- “CALL PRXNEXT ルーチン” (199 ページ)
- “CALL PRXSUBSTR ルーチン” (204 ページ)
- “CALL SCAN ルーチン” (233 ページ)
- “CALL SORTC ルーチン” (245 ページ)
- “CALL STDIZE ルーチン” (247 ページ)
- “CALL SYSTEM ルーチン” (254 ページ)

これに対し、%SYSCALL は CALL ルーチンを起動し、デルタ文字を含む未解決の値を返します。%SYSFUNC は関数を起動し、デルタ文字を含まない解決済みの値を返します。詳細については、“Macro Quoting” in Chapter 7 of *SAS Macro Language: Reference*、“%SYSCALL Statement” in *SAS Macro Language: Reference* および “%SYSFUNC and %QSYSFUNC Functions” in *SAS Macro Language: Reference* を参照してください。

関数を使用したファイルの操作

SAS では、関数やステートメントのいずれを使用するかによってファイルの操作方法が異なります。FOPEN、FGET および FCLOSE などの関数を使えば、INFILE、INPUT および PUT などのステートメントを使うよりもデータを検証、操作する自由度が高くなります。

外部ファイルを使うとき、FOPEN 関数はファイルデータバッファ(FDB)と呼ばれるバッファを割り当て、外部ファイルを開いて読み込みや更新を行います。FREAD 関数は外部ファイルからレコードを読み込み、データを FDB にコピーします。続いて、FGET 関数がデータを DATA ステップ変数に移動します。DATA ステップ内のステートメントや他の関数でこの関数が返す値を確認し、データを

引き続き処理する方法を決定します。レコードの処理後、FWRITE 関数が FDB の内容を外部ファイルに書き込むと、FCLOSE 関数はファイルを閉じます。

SAS データセットを使うときは、OPEN 関数でデータセットを開きます。FETCH 関数と FETCHOBS 関数は開いている SAS データセットからオブザベーションを DDV(データセットデータデータベクトル)内に読み込みます。続いて、GETVARC 関数と GETVARN 関数がデータを DATA ステップ変数に移動します。DATA ステップ内のステートメントや他の関数でこれらの関数が返す値を確認し、データを引き続き処理する方法を決定します。データの処理後、CLOSE 関数でそのデータセットを閉じます。

関数と CALL ルーチンの完全なリストについては、“カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)を参照してください。 詳細と例については、本書の辞書セクションを参照してください。

関数の SBCS、DBCS、MBCS 文字セットとの互換性

概要

SAS 文字列関数と CALL ルーチンは国際化で使用されるレベル番号によって分類されます。I18N とは国際化の略称であり、プログラムを変更することなく異なる言語と口ケールに適合する文字列関数を示します。

I18N では、次の 3 つのレベルで使用できる文字セットを識別しています。

- “I18N レベル 0” (10 ページ)
- “I18N レベル 1” (10 ページ)
- “I18N レベル 2” (10 ページ)

関数の互換性の詳細については、“SAS 文字列関数の国際化の互換性” (*SAS 各国語サポート(NLS): リファレンスガイド* 10 章)を参照してください。

I18N レベル 0

I18N レベル 0 の関数は 1 バイト文字セット(SBCS)でのみ使用するように設計されています。

I18N レベル 1

英語以外の言語を使用する場合、可能な限り I18N レベル 1 の関数の使用は避けてください。I18N レベル 1 の関数は、特定の環境下では 2 バイト文字セット(DBCS)または複数バイト文字セット(MBCS)エンコーディングを使用すると正常に動作しない場合があります。

I18N レベル 2

I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

乱数関数と乱数 CALL ルーチンの使用

乱数関数の種類

SAS には 2 種類の乱数関数が用意されています。最新の乱数関数は RAND 関数です。この関数は、松本と西村(1998)によって開発された Mersenne-Twister 擬似乱数ジェネレータ(RNG)を応用しています。この RNG は $2^{19937} - 1$ の非常に長い周期を持ち、高品質な統計的特性を有しています (周期は疑似乱数列が繰り返されるまでの生成数)。

RAND 関数は単一のシードで開始します。ただし、プロセスの状態は単一のシードでは取得できません。つまり、ジェネレータを停止した後、停止点からの再開はできません。ストリームの最初から始まる数値列を生成するには、STREAMINIT 関数を使用します。 詳細については、“RAND 関数” (784 ページ) の詳細セクションを参照してください。

従来からの乱数ジェネレータには、UNIFORM、NORMAL、RANUNI、RANNOR の他、RAN で始まる関数があります。これらの関数では周期が $2^{31} - 2$ 以下しかありません。擬似乱数ストリームは単一のシードで開始し、プロセスの状態を新しいシードに取得できます。つまり、ジェネレータを停止した後、適切なシードを対応する CALL ルーチンに与えることでジェネレータを停止点から再開できます。ストリームの途中から始まる数値列を生成するには、乱数関数を使用します。

シード値

乱数関数と CALL ルーチンは、*シード*と呼ばれる初期開始点から疑似乱数のストリームを生成します。シードはユーザーが指定するか、コンピュータのクロックから取得します。シードは 2^{31}–1(2,147,483,647)より小さい負でない整数値である必要があります。正のシードを使用すると、同じ DATA ステップを使用することで乱数のストリームを常に再現できます。ゼロをシードに使用すると、コンピュータクロックによってストリームが初期化され、乱数のストリームは再現できなくなります。

関数の乱数ストリーム生成方法の理解

DATA ステップで乱数のシングルストリームを生成する

このセクションの DATA ステップで、乱数関数のプロパティをいくつか説明します。関数を呼び出す DATA ステップは、各ステップとも初回呼び出しの初期シードが 7 となっているため、それぞれシード値 7 に基づく単一の疑似乱数ストリームを生成します。一部の DATA ステップでは、各種の方法でシード値を変更します。関数呼び出しが 1 つしかないステップもあれば、複数の関数呼び出しが行われるステップもあります。これらの DATA ステップは、どれもシードを変更しません。関数呼び出しに関連するシードは、最初の乱数関数の初回実行に使われるシードのみです。関数で独立したストリームを作成する方法はなく(作成するには CALL ルーチンを使用します)、新しい DATA ステップを開始するしか、関数による乱数ストリームを再開する方法はありません。

次の例では複数の DATA ステップを実行します。

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

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

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

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

/* This DATA step calls the RANNOR function and produces a */
/* single stream of random numbers based on a seed value of 7. */
```

```
data e (drop = i);
do i = 1 to 6;
e = rannor (7); output;
e = .; output;
end;
run;

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

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

このプログラムの結果出力を次に示します。

画面 1.1 *単一の乱数ストリームを生成した結果*

Single Random Number Streams

Obs	a	b	c	d		e
1	0.29474	0.29474	0.29474	0.29474		0.39464
2	0.79062	0.79062	0.79062	0.79062		
3	0.79877	0.79877	0.79877	0.26928	n	0.26928
4	0.81579	0.81579	0.81579			
5	0.45122	0.45122	0.45122	0.45122		0.27475
6	0.78494	0.78494	0.78494	0.78494		
7	0.80085	0.80085	0.80085	-0.11729	n	-0.11729
8	0.72184	0.72184	0.72184			
9	0.34856	0.34856	0.34856	0.34856		-1.41879
10	0.46597	0.46597	0.46597	0.46597		
11	0.73523	0.73523	0.73523	-0.39033	n	-0.39033
12	0.66709	0.66709	0.66709			

出力データセット A、B および C の疑似乱数ストリームは同一です。出力データセット D のストリームには、RANUNI 関数と RANNOR 関数への呼び出しが併用されています。オブザベーション 1、2、5、6、9、10 では、RANUNI から返される値が先のストリームの値と完全に一致しています。"n"フラグ付きのオブザベ

ーション 3、7 および 11 には、RANNOR 関数から返された値が含まれています。関数呼び出しを併用しても、疑似乱数ストリームの生成には影響しません。結果はすべて、一様に分布した値の単一ストリームに基づいています。値の一部は、RANNOR など、他の関数によって変換されてから返された値です。RANNOR 関数の結果は、RANUNI への 2 回の内部呼び出しによって生成されたものです。出力データセット D を作成する DATA ステップは、次のステップを 3 回実行し、12 のオブザベーションを作成します。

- RANUNI への呼び出し
- RANUNI への呼び出し
- RANNOR への呼び出し(RANUNI への内部呼び出しを 2 回実行)
- RANUNI への 2 回目の内部呼び出しを補正するためのスキップ行

データセット E を作成する DATA ステップでは、RANNOR が 6 回呼び出されます。呼び出し時に毎回 1 行をスキップしますが、これは RANNOR への呼び出しごとに実行される 2 回の RANUNI への内部呼び出しを補正するためです。データセット D を作成する DATA ステップで RANNOR から返される 3 つの値は、データセット E の対応する値と一致しています。

%SYSFUNC マクロで乱数のシングルストリームを生成する

%SYSFUNC を使用し、マクロ言語を介して RANUNI 関数を呼び出すと、疑似乱数ストリームが 1 つ作成されます。SAS を終了し、新しい SAS セッションを開始しない限りはシード値を変更できません。%SYSFUNC マクロは初回の起動時のみ、データセット A、B および C を生成した DATA ステップと同一の疑似乱数ストリームを生成します。後続のマクロ呼び出しでは、単一ストリームの続きを生成します。

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

%ran;
```

SAS は次の出力をログに書き込みます。

ログ 1.2 %SYSFUNC マクロを使った実行の結果

```
10 %macro ran;
11 %do i = 1 %to 12;
12 %let x = %sysfunc (ranuni (7));
13 %put &x;
14 %end;
15 %mend;
16 %ran;
0.29473798875451
0.79062100955779
0.79877014262544
0.81579051763554
0.45121804506109
0.78494144826426
0.80085421204606
0.72184205973606
0.34855818345609
0.46596586120592
0.73522999404707
0.66709365028287
```

乱数関数と乱数 CALL ルーチンのシード値の比較

乱数関数と CALL ルーチンは、それぞれ特定の統計的分布から疑似乱数を生成します。乱数関数はいずれも、整数定数、または整数定数を含む変数で表されるシード値を必要とします。CALL ルーチンはいずれもシード値を含む変数を呼び出します。また、すべての CALL ルーチンに生成された疑似乱数を含む変数が必要です。

シード変数は、関数または CALL ルーチンの初回実行前に初期化する必要があります。関数の毎回の実行後、現在のシードは内部で更新されますが、シード引数の値は変化しません。ただし、CALL ルーチンの毎回の反復後、シード変数は次の疑似乱数を生成するストリーム内の現在のシードを含みます。関数ではシード値を制御できません。そのため、初期化後の疑似乱数を制御できません。

NORMAL 関数と UNIFORM 関数(それぞれ RANNOR 関数と RANUNI 関数と同等)を除き、各乱数関数と同名の CALL ルーチンが SAS に用意されています。CALL ルーチンを使用すると、シード値をさらに高度に制御できます。

乱数 CALL ルーチンで複数のシードから複数のストリームを生成する

乱数 CALL ルーチンとストリームの概要

乱数 CALL ルーチンを使用すると、単一の DATA ステップで複数の疑似乱数ストリームを生成できます。各シード変数の初期化に異なるシード値を指定すると、生成される疑似乱数ストリームは計算量的には独立しますが、シード値を注意深く選択しなければ統計的に独立しません。

注: 複数のシードを使えば複数のストリームを作成できますが、これを実行することはお勧めしません。ストリームの作成は 1 つにとどめるのが安全です。複数のストリームを作成すると、ストリームが長くなるにつれて、ストリームが重複する可能性が増します。

次の 2 つの例では、最悪のシナリオとなるようなシードを故意に選択しています。これらの例では、複数のシードを使って複数のストリームを作成する方法を

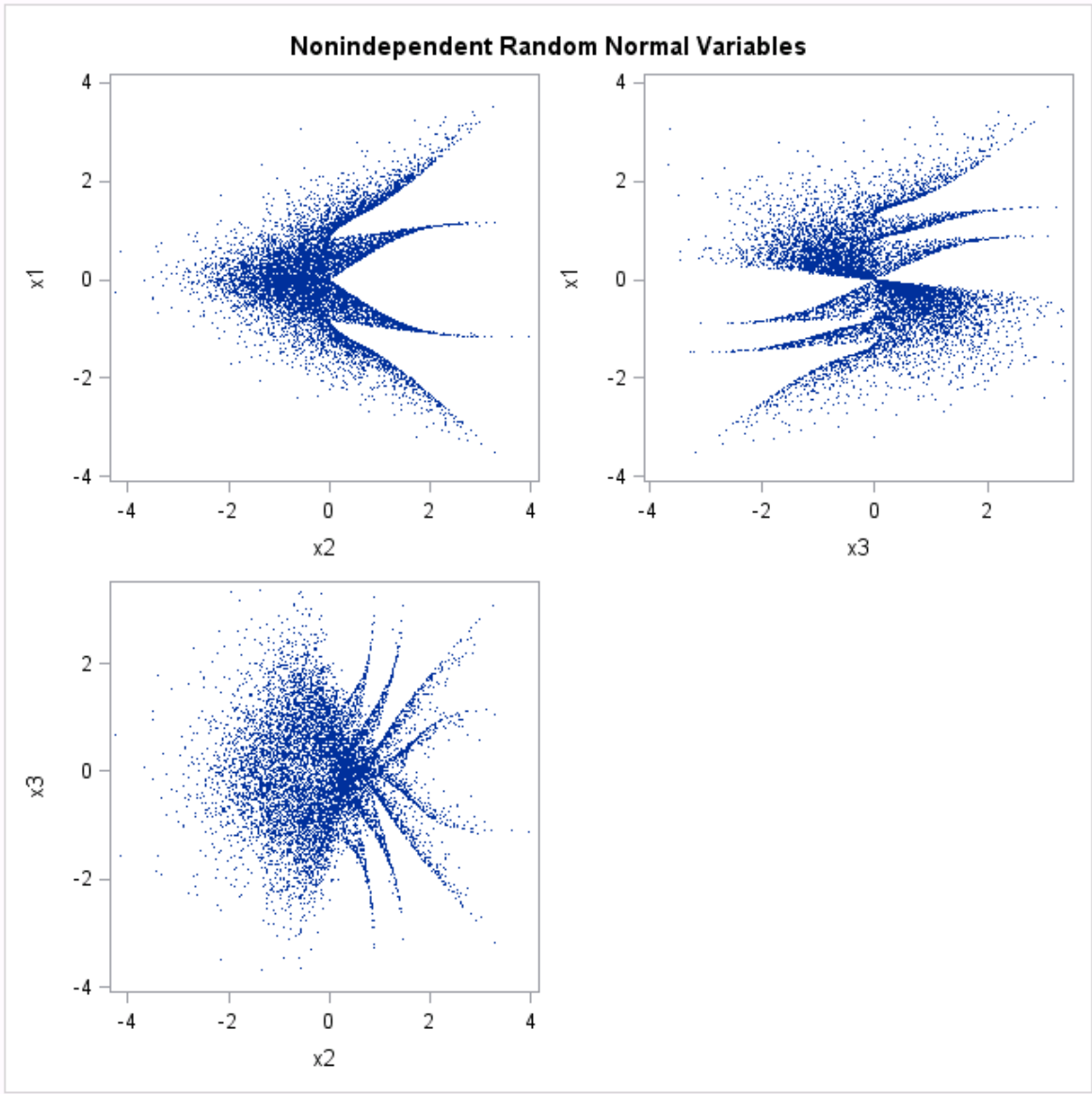
示しています。これは非推奨の方法ですが、乱数 CALL ルーチンで複数のシードを使えます。

例 1: 複数のストリームを作成するために複数のシードを使用する

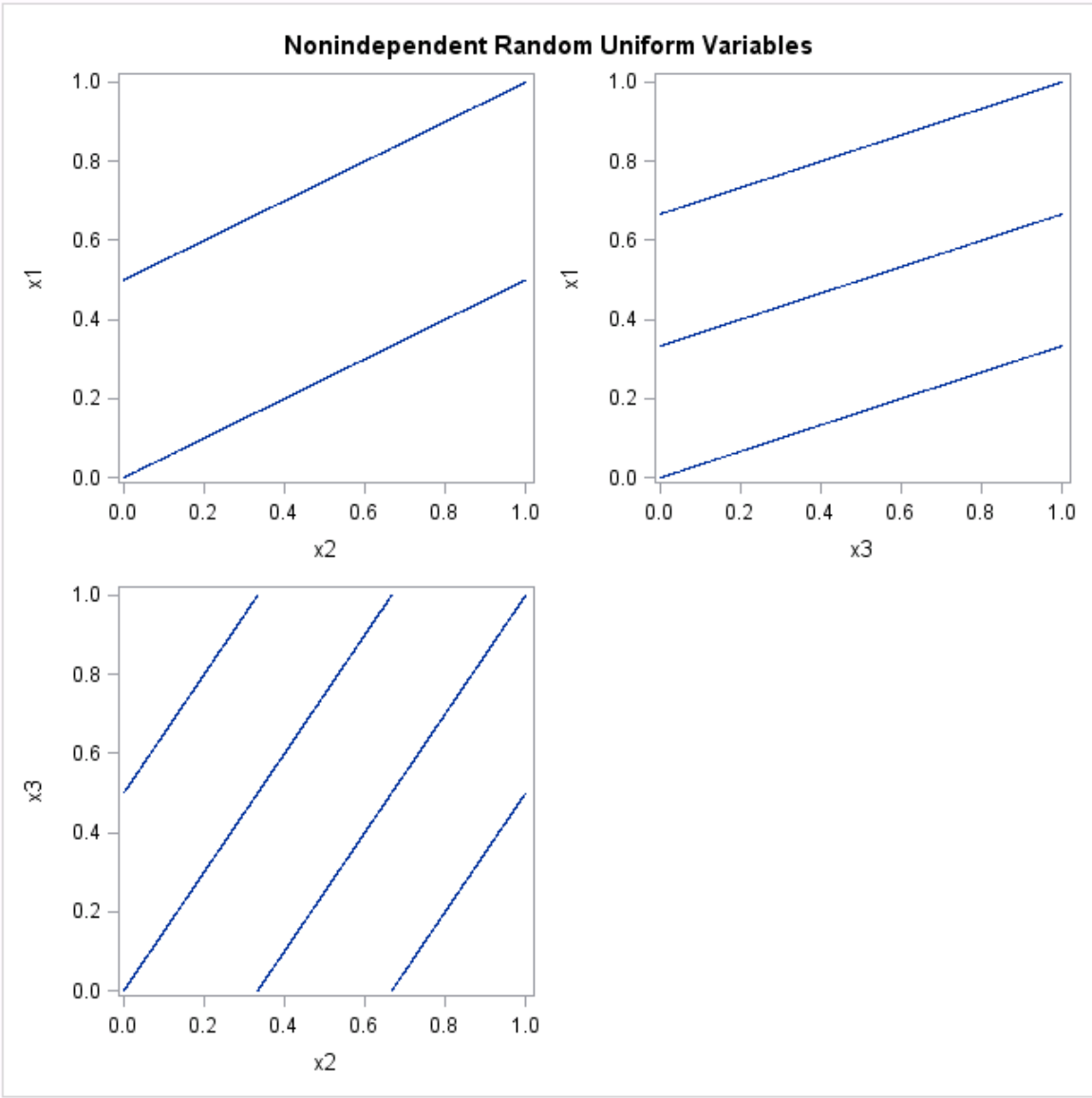
この例は、乱数 CALL ルーチンを使用し、複数のシードから疑似乱数的に分布した値の複数ストリームを生成できることを示しています。最初の DATA ステップでは、正規分布の変数から 3 つのデータセットを作成します。2 番目の DATA ステップでは、一様分布の変数を作成します。SGSCATTER プロシジャ(*SAS ODS Graphics: プロシジャガイド*を参照)は、2 種類の分布のそれぞれで、各変数ペア間にどのような関係があるかを示すために使用しています。

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

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

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

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

画面 1.2 複数のシードから作成した複数のストリーム: 独立していないランダムな正規変数



画面 1.3 複数のシードから作成した複数のストリーム: 独立していないランダムな一様変数



1 番目のプロット(画面 1.2 (17 ページ))では、正規変数が直線的な相関関係にはないように見えますが、明らかに独立していません。2 番目のプロット(画面 1.3 (18 ページ))では、一様変数が明確に相関関係にあることがわかります。この部類の乱数ジェネレータでは、ストリームが独立したものとなる保証は一切ありません。

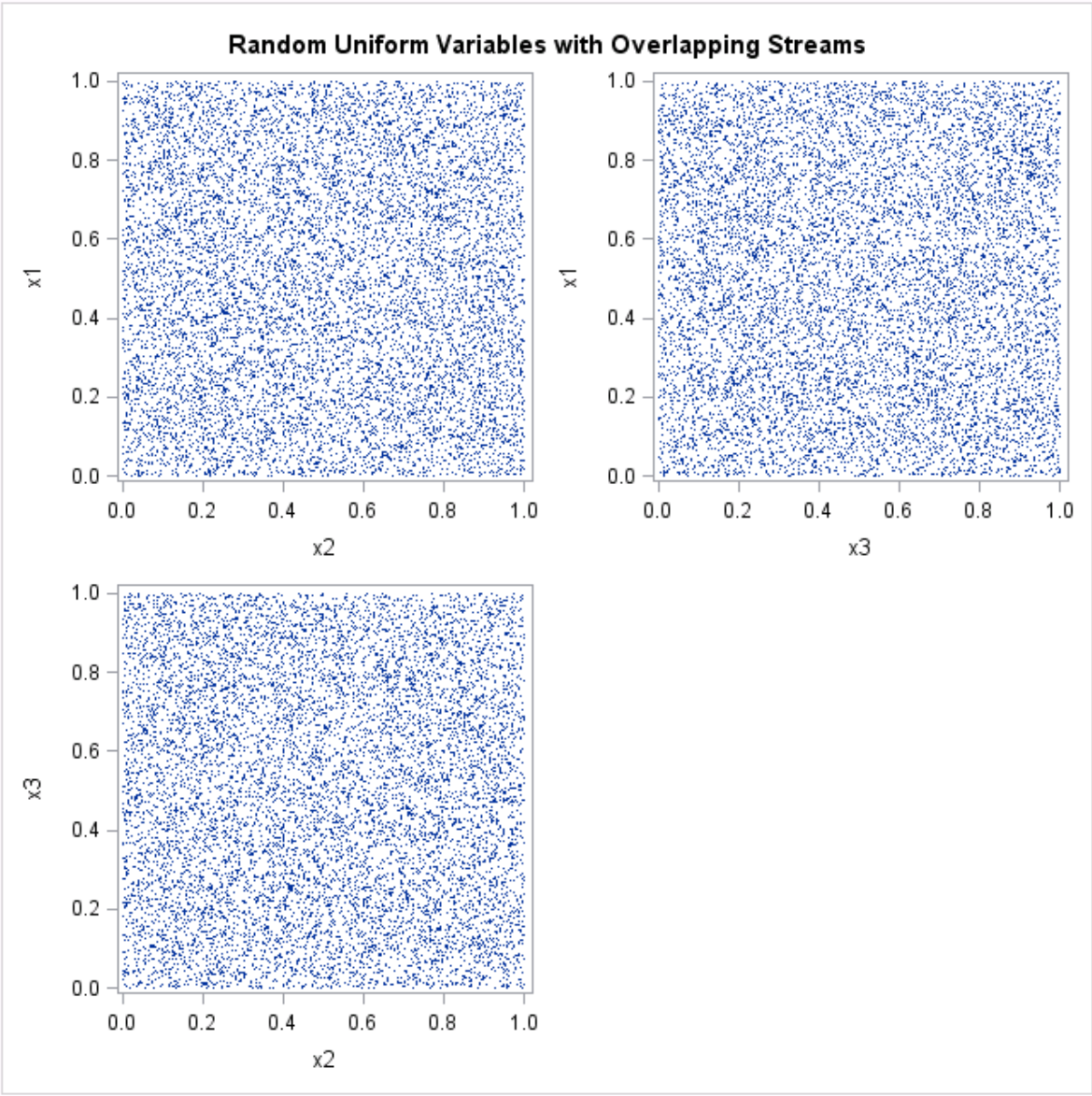
例 2: CALL RANUNI ルーチンで異なるシードを使用する

次の例では、3 つの異なるシードと CALL RANUNI ルーチンを使用して複数のストリームを作成しています。

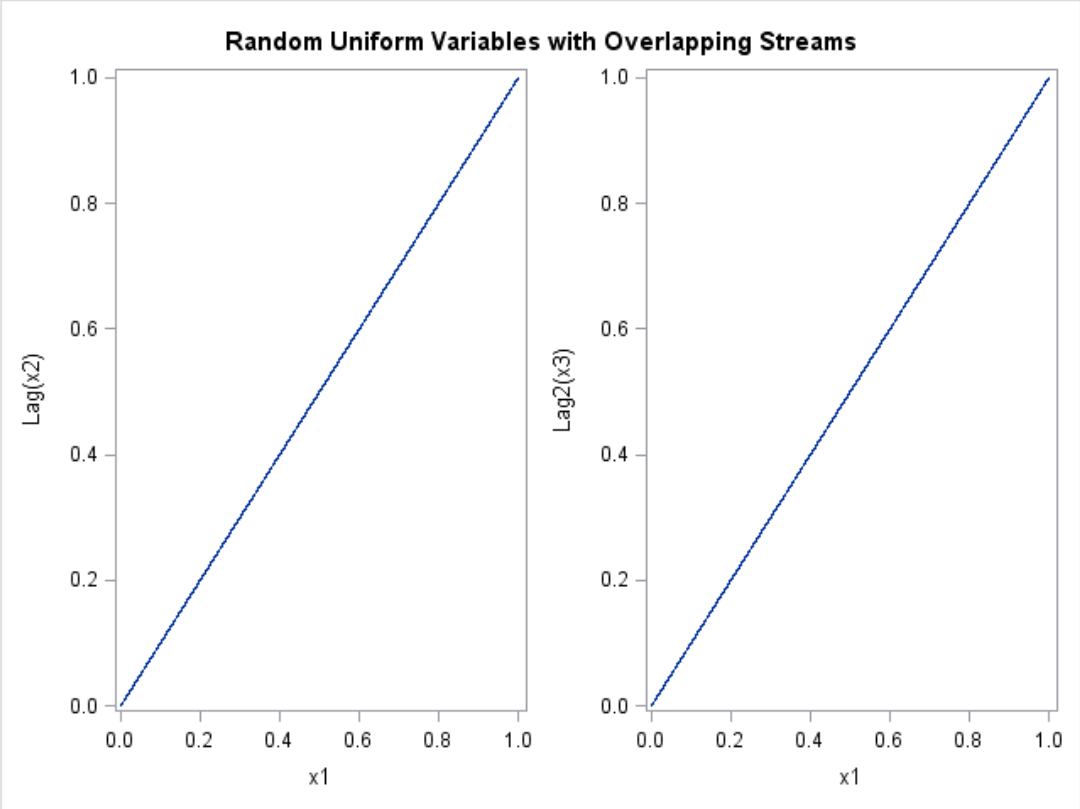
```
data uniform(drop=i);
seed1 = 255793849;
seed2 =1408147117;
seed3 = 961782675;
```

```
do i=1 to 10000;
call ranuni(seed1, x1);
call ranuni(seed2, x2);
call ranuni(seed3, x3);
i2 = lag(x2);
i3 = lag2(x3);
output;
end;
label i2='Lag(x2)' i3='Lag2(x3)';
run;

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

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

proc print noobs data=uniform(obs=10);
run;
```

画面 1.4 *CALL RANUNI で異なるシードを使用する: 重複するストリームを持つランダムな一様分布変数: プロット 1*



画面 1.5 *CALL RANUNI で異なるシードを使用する: 重複するストリームを持つランダムな一様分布変数: プロット 2*



画面 1.6 重複するストリームを持つランダムな一様分布変数

Random Uniform Variables with Overlapping Streams

seed1	seed2	seed3	x1	x2	x3	i2	i3
1408147117	961782675	383001085	0.65572	0.44786	0.17835	.	.
961782675	383001085	1989090982	0.44786	0.17835	0.92624	0.44786	.
383001085	1989090982	1375749095	0.17835	0.92624	0.64063	0.17835	0.17835
1989090982	1375749095	89319994	0.92624	0.64063	0.04159	0.92624	0.92624
1375749095	89319994	1345897251	0.64063	0.04159	0.62673	0.64063	0.64063
89319994	1345897251	561406336	0.04159	0.62673	0.26143	0.04159	0.04159
1345897251	561406336	1333490358	0.62673	0.26143	0.62095	0.62673	0.62673
561406336	1333490358	963442111	0.26143	0.62095	0.44864	0.26143	0.26143
1333490358	963442111	1557707418	0.62095	0.44864	0.72536	0.62095	0.62095
963442111	1557707418	137842443	0.44864	0.72536	0.06419	0.44864	0.44864

1 番目のプロット(画面 1.4 (20 ページ))は、変数が統計的に独立した、期待どおりの結果を示しています。ただし、2 番目のプロット(画面 1.5 (21 ページ))と最初の 10 のオブザベーションのリストは、2 つのストリームでほぼ完全な重複があることを示しています。x1 の最後の 9999 個の値は x2 の最初の 9999 個の値と一致しており、x1 の最後の 9998 個の値は x3 の最初の 9998 個の値と一致しています。言い換えれば、x1 と lag(x2)に加え、x1 と lag2(x3)の間の非欠損部分に完全な一致が存在します。1 番目のプロットのように、一見ストリームが独立しているようでも重複の可能性があります。ストリームの用途によっては、これは望ましくありません。

実際には、個別のシードをランダムに選んで小さいストリームを複数作成すれば、最初の 2 つの例に示したような問題にはおそらく遭遇しません。 画面 1.5 (21 ページ) では、最悪のシナリオとなるようなシードを故意に選択しています。

ストリームの作成は 1 つにとどめるのが安全です。複数のストリームを作成すると、ストリームが長くなるにつれて、ストリームが重複する可能性が増します。

乱数関数で単一のシードから複数の変数を生成する

関数とストリームの概要

プログラムで関数を使用する場合は、DATA ステップ内で複数のシードを指定して複数の疑似乱数ストリームは生成できません。

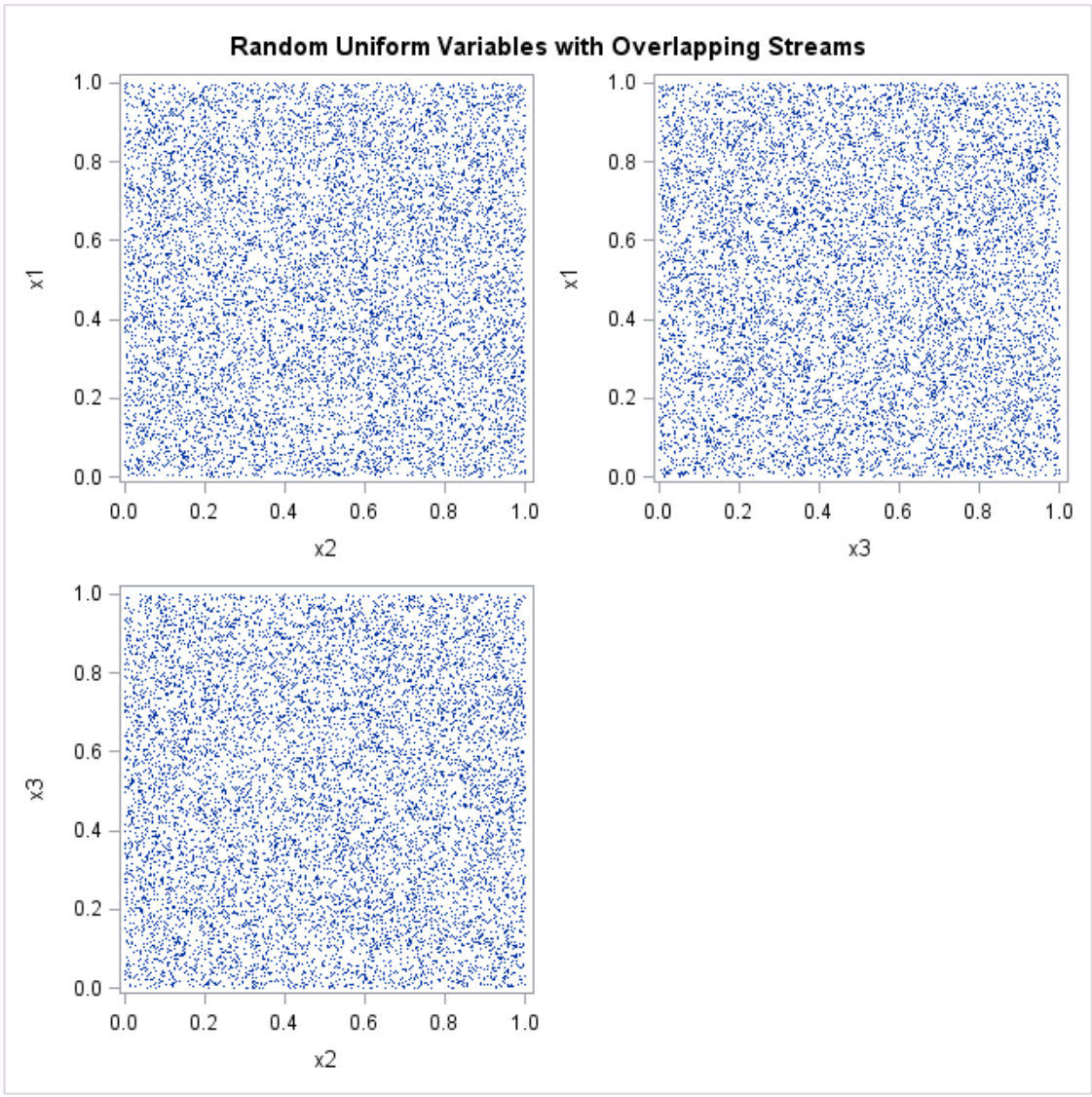
RANUNI 関数を使用し、単一のシードで同一のストリームから複数の変数を作成する最も安全な方法を次の例に示します。

例: 重複するストリームを持つランダムな一様分布変数を生成する

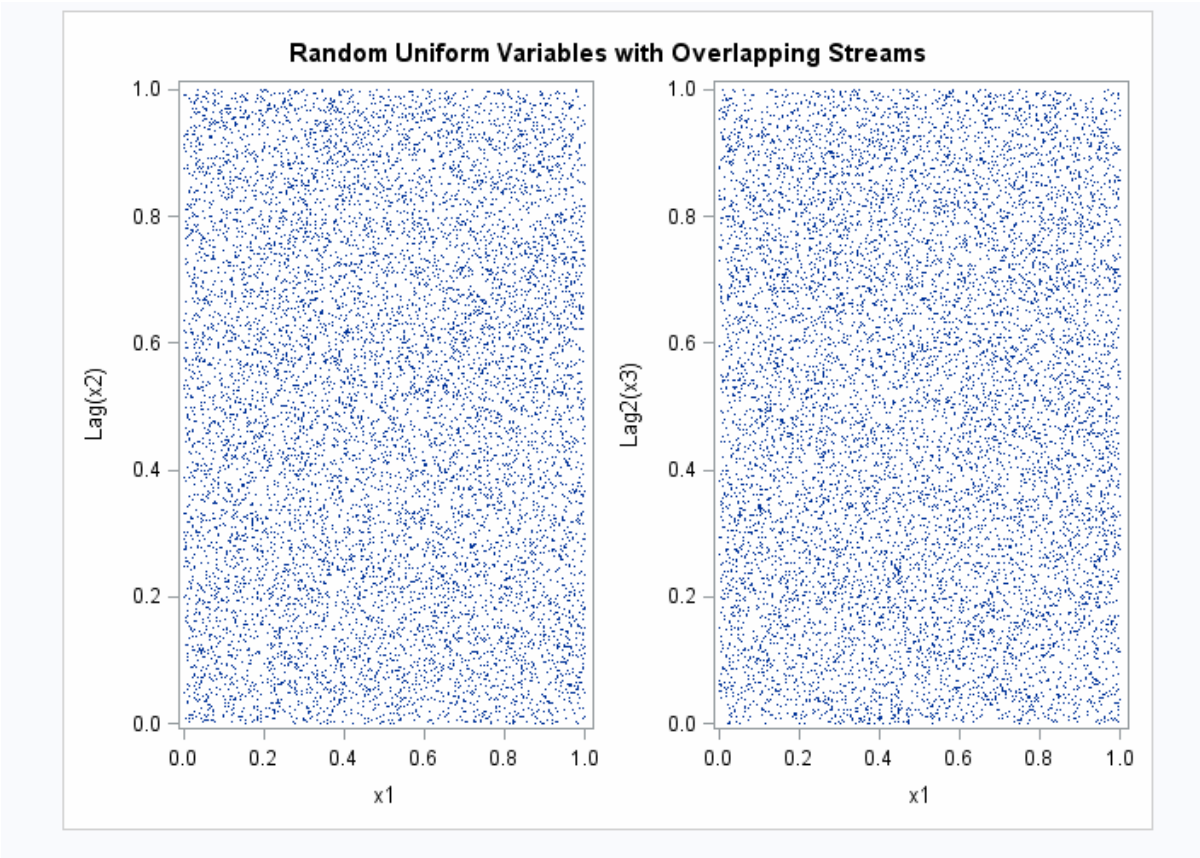
次の例では、重複するストリームを持つランダムな一様分布変数を作成するのに RANUNI 関数を使用しています。この例は、RANUNI 関数を使用して複数の変数を作成する最も安全な方法を示しています。すべての変数が単一のシードで同一のストリームから作成されます。

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

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

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

画面 1.7 *重複するストリームを持つランダムな一様分布変数: プロット 1*



画面 1.8 重複するストリームを持つランダムな一様分布変数: プロット 2



“例: 重複するストリームを持つランダムな一様分布変数を生成する” (23 ページ),では、変数が独立しているように見えます。ただし、このプログラミング手法でも全般的に問題なく機能するとは限りません。乱数関数と CALL ルーチンの周期は $2^{31} - 2$(およそ 21 億)以下でしかありません。この限界に達するとストリームは繰り返されます。複雑なシミュレーションをこなす現代のコンピュータなら、ストリーム全体をわずか数分で簡単に使い果たしてしまいます。

RAND 関数を代替として使用する

ランダムな一様分布変数を生成する手法としてより好ましいのは、複数ストリームが許可されない RAND 関数を使用する方法です。RAND 関数の周期は $2^{19937} - 1$ です。少なくとも 21 世紀初頭のコンピュータには、この限界は到達できません。$2^{19937} - 1$ という数は、およそ 10^{6000}(1 の後にゼロが 6000 個)です。比較として、SAS を実行するコンピュータのほとんどが 8 バイトで表すことのできる最大値はおよそ 10^{307} です。

開発された最も新しい乱数関数の RAND 関数では、複数のストリームを使用できません。RAND 関数は乱数 CALL ルーチンから異なるアルゴリズムを使用することで、複数のシードを使った複数のストリームを作成します。RAND プロセスの状態は単一のシードでは取得できないため、ジェネレータを停止した後、停止点からの再開はできません。したがって、RAND 関数で許可されるのは単一の数列ストリームのみですが、RANUNI 関数と同様に、この関数を使用して複数のストリームを作成できます。

乱数 CALL ルーチンを効果的に使用する

ストリームを開始、停止および再開する

RANUNI のストリームが使い果たされない限り、通常は単一のストリームを開始、停止するのに乱数 CALL ルーチンを使用します。たとえば、SAS で反復を実行し、停止して結果を評価し、ストリームの停止点から再開したい場合を考えます。次の例ではこの原理を説明しています。

例: ストリームを開始、停止および再開する

この例では、5 つの数値のストリームを生成、停止および再開してから同じストリームからさらに 5 つの数値を生成し、結果を組み合わせて比較用の完全なストリームを生成します。最初の DATA ステップでは、次のステップで開始シードに使用するために、乱数シードの状態をマクロ変数シードに格納しています。この例の個別ストリームの出力は、完全なストリームに一致します。

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

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

data all;
set u1 u2;
z = ranuni(104);
run;

proc print label;
title 'Random Uniform Variables with Overlapping Streams';
label x = 'Separate Streams' z = 'Single Stream';
run;
```

画面 1.9 ストリームを開始、停止および再開する

Random Uniform Variables with Overlapping Streams

Obs	Separate Streams	Single Stream
1	0.23611	0.23611
2	0.88923	0.88923
3	0.58173	0.58173
4	0.97746	0.97746
5	0.84667	0.84667
6	0.80484	0.80484
7	0.46983	0.46983
8	0.29594	0.29594
9	0.17858	0.17858
10	0.92292	0.92292

CALL ルーチン内の場合と関数内の場合のシード変更の比較

例: CALL ルーチン内と関数内のシード変更

シードの変更に CALL ルーチンを使用する場合の結果は、関数を使用してシードを変更する場合と異なります。この違いの例を次に示します。

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

proc print data = seeds;
title 'Random Uniform Variables with Overlapping Streams';
id i;
run;
```

画面 1.10 CALL ルーチン内と関数内のシード変更

Random Uniform Variables with Overlapping Streams

i	Seed1	Seed2	Seed3	X1	X2	X3
1	507036483	507036483	104	0.23611	0.23611	0.23611
2	1909599212	1909599212	104	0.88923	0.88923	0.88923
3	1249251009	1249251009	104	0.58173	0.58173	0.58173
4	2099077474	2099077474	104	0.97746	0.97746	0.97746
5	1818205895	17	17	0.84667	0.84667	0.84667
6	1728390132	310018657	17	0.80484	0.14436	0.80484
7	1008960848	1055505749	17	0.46983	0.49151	0.46983
8	635524535	1711572821	17	0.29594	0.79701	0.29594
9	383494893	879989345	17	0.17858	0.40978	0.17858
10	1981958542	1432895200	17	0.92292	0.66724	0.92292

CALL RANUNI ステートメントで i=5 のときに Seed2 を変更すると、X2 のストリームは X1 のストリームから強制的に逸脱させられます。ただし、RANUNI 関数で Seed3 を変更した場合には影響はありません。何も変更がなかったかのように X3 のストリームが続行するため、X1 と X3 のストリームは同一です。

SYSRANDOM マクロ変数と SYSRANEND マクロ変数を使用した乱数ストリームの作成

SYSRANDOM マクロ変数と SYSRANEND マクロ変数の概要

SAS には乱数ストリームを使用するプロシジャが多数あります。たとえば、FREQ、GLM、MCMC、OPTEX、PLAN などです。これらのプロシジャは、SAS DATA ステップで使用するのと同じ乱数関数と CALL ルーチンを使用します。SAS プロシジャでは、乱数ストリームを初期化するシードを指定するのに SEED=オプションを使用します。

乱数シードを使用する SAS プロシジャは、SYSRANDOM と SYSRANEND の 2 つのマクロ変数を作成します。これらのマクロ変数を使用すると、別のプロシジャで再現できる乱数ストリームを作成できます。

SYSRANDOM マクロ変数

SYSRANDOM マクロ変数は、直近のプロシジャの乱数シードを格納します。このマクロ変数は、SEED=オプションで指定されている整数に対応します。FREQ、GLM、MI、MCMC、OPTEX、PHREG、PLAN など、多数のプロシジャに SEED=オプションがあります。SEED=オプションで指定する整数は、乱数ストリームの

開始に使用します。正のシードを指定すると、指定したとおりに使用されます。SEED=オプションでシードを指定しない場合、またはシードがゼロ以下の場合は、プロシジャでクロックタイムからシードが生成されます。SYSRANDOM マクロ変数を使用すれば、直接指定したシードと内部生成されたシードのどちらでも復元できます。

SYSRANEND マクロ変数

SYSRANEND マクロ変数は、プログラムの次のステップで使用できるようにシードを格納します。プロシジャが完了したときの乱数プロセスの状態を、このシードが取得する場合があります。プログラムに複数のステップを含めて、各プロシジャで明示的にシードを指定せずに乱数列を制御できます。シミュレーションを 1 つのシードで開始し、後続のすべてのプロシジャで SYSRANEND マクロ変数を使ってシードを指定できます。また、同一のストリームを継続する乱数ジェネレータ(RNG)の停止と再開にも SYSRANEND マクロ変数を使用できる場合があります。

SAS プロシジャには 2 種類の RNG があります。RANUNI 関数で使用されている従来からの RNG は、疑似乱数ストリームを単一のシードで開始し、プロセスの状態を新しいシードに取得できます。GLM、GLIMMIX、MI、OPTEX、PLAN やその他のプロシジャでは、この従来からの RNG が使用されます。プロシジャが終了すると、SYSRANEND に格納されている値が新しいシードとなります。ジェネレータを停止した後、SYSRANEND マクロ変数を使用して停止点からジェネレータを再開できます。

MCMC、GENMOD、LIFEREG、PHREG など、その他のプロシジャは新しい Mersenne-Twister RNG を使用します。この RNG は RAND 関数でも使用されており、単一のシードを通じてストリームの状態を伝えません。プロシジャの中には、1 つの RNG を複数の計算に使用するものもあれば、他の計算には他の RNG を使用するものあります。これらのプロシジャから SYSRANEND マクロ変数を使用すれば、一連のプロシジャ実行を再現できるようになります。ただし、ランダムストリームは、単一の長いプロシジャ実行時のストリームとは同じにはなりません。

例: 結果を再現する

シードを復元し、復元したシードを一連の結果を再現する方法の例を次に示します。MCMC プロシジャは事後分布からサンプルを生成します。次のステートメントは線形回帰モデルから事後サンプルを作成します。

```
title 'Bayesian Linear Regression';

proc mcmc data=sashelp.class seed=0 outpost=out1;
parms beta0 0 beta1 0;
prior beta0 beta1 ~ normal(mean=0, var=1e6);
mu=beta0 + beta1*height;
model weight ~ n(mu, var=137);
run;
```

SEED=0 が指定されたため、乱数シードがクロックタイムから自動的に生成されます。このシードは SYSRANDOM マクロ変数に格納されます。値は%PUT ステートメントを使って表示できます。

```
%put sysrandom=&sysrandom;
```

次のステップは、前のステップと同一のシードで同一の結果を作成します。

```
proc mcmc data=sashelp.class seed=&sysrandom outpost=out2;
parms beta0 0 beta1 0;
prior beta0 beta1 ~ normal(mean=0, var=1e6);
mu=beta0 + beta1*height;
model weight ~ n(mu, var=137);
run;
```

次のステップをサブミットすると、PROC MCMC の 2 回の実行が同一のサンプルを作成していることを確認できます。

```
proc compare data=out1 compare=out2;
run;
```

例: 再現可能な乱数ストリームを作成する

PLAN プロシジャは完全要因実験計画を作成してランダム化します。OPTEX プロシジャは最適な実験計画の計画点候補一式を検索します。両方のプロシジャに SEED=オプションがあります。SYSRANEND マクロ変数を使用して、一連のステップを再現可能にする例を次に示します。

```
proc plan seed=17;
factors x1=4 x2=4 x3=2 x4=2 x5=3 x6=3 x7=2 x8=2 / noprint;
output out=cand;
run;
quit;

%put sysranend=&sysranend;

proc optex data=cand seed=&sysranend;
class x1-x8;
model x1-x8;
generate n=26 iter=10 method=m_federov;
output out=des;
run;
quit;
```

PROC OPTEX を複数回呼び出し、効率(計画がどの程度有効かを示す指標)が 98% を超える計画を検出したときに停止できます。SYSRANDOM マクロ変数と SYSRANEND マクロ変数を使用してこれを実現できます。次のステートメントは PROC OPTEX を呼び出して 100 の計画を作成し、最良の *D* 効率を出力します。

```
proc plan seed=17;
factors x1=4 x2=4 x3=2 x4=2 x5=3 x6=3 x7=2 x8=2 / noprint;
output out=cand;
run;
quit;

%macro design;
ods listing close;
%do %until(%sysevalf(&eff > 98));
proc optex data=Cand seed=&sysranend;
class x1-x8;
model x1-x8;
generate n=26 iter=100 keep=1 method=m_federov;
ods output efficiencies=e1;
run;
quit;
```

```
data _null_;
set e1;
call symputx('eff', dcriterion, 'L');
run;
%end;
ods listing;

proc optex data=Cand seed=&sysrandom;
class x1-x8;
model x1-x8;
generate n=26 iter=100 keep=1 method=m_federov;
output out=des;
run;
quit;
%mend;

%design;
```

D 効率はマクロ変数に格納されます。また、*D* 効率が 98 より大きいときには反復が停止します。最後のステップのシードは最終結果を再現し、表示するために使用されます。

日付間隔と時間間隔

日付間隔と時間間隔の定義

間隔とは、日、月または時間などの経過期間内で SAS が計測する測定単位です。SAS では、日付と時間の間隔をカレンダ上またはクロック上の固定点に基づいて決定します。間隔計算の開始点は、デフォルトで開始値が入る期間の開始時点となります。これは、指定された実際の開始値とは異なる場合があります。たとえば、2 つの日付間の月数を数えるのに INTCK 関数を使用する場合、開始値の日付に指定した日付が実際にその月の何日であるかにはかかわらず、SAS は開始値を該当月の初日として扱います。

間隔名と SAS 日付

SAS 日付値で使用する特定の間隔名もあれば、SAS 時間値および SAS 日時値で使用する間隔名もあります。SAS 日付値で使用する間隔名には YEAR、SEMIYEAR、QTR、MONTH、SEMIMONTH、TENDAY、WEEK、WEEKDAY、DAY があります。SAS 時間値および SAS 日時値で使用する間隔名には HOUR、MINUTE および SECOND があります。

SAS 日付値で使用する間隔名には、'DT'を接頭辞として付加し、SAS 日時値で使用する間隔名を作成できます。間隔名 DTYEAR、DTSEMIYEAR、DTQTR、DTMONTH、DTSEMIMONTH、DTTENDAY、DTWEEK、DTWEEKDAY、DTDAY、は SAS 時間値または SAS 日時値で使用します。

乗数とシフト間隔を使用した日時の増分

各種の経過時間を数えるための日付、時間および日時の間隔が SAS には用意されています。乗数とシフトインデックスを使用することで間隔の倍数を作成し、開始点をシフトしてより複雑な間隔指定を作成できます。

間隔名の一般的な形式を次に示します。

name<multiplier><.shift-index>

multiplier と *shift–index* の各引数は省略可能で、デフォルトは 1 です。たとえば、YEAR、YEAR1、YEAR.1、YEAR1.1 は、すべてが同様に 1 月から始まる通常のカレンダ年を指定する方法です。*multiplier* と *shift-index* に他の値を指定すると、1 年で異なる時期に始まる複数の間隔を作成できます。たとえば、WEEK6.11 という間隔指定は、第 2 水曜日を開始時点とする 6 週間隔です。

詳細については、“単一単位の間隔” (*SAS 言語リファレンス: 解説編* 7 章)、“複数単位の間隔” (*SAS 言語リファレンス: 解説編* 7 章)、および “間隔のシフト” (*SAS 言語リファレンス: 解説編* 7 章)を参照してください。

よく使用される時間間隔

年または日にネストできない時間間隔は、SAS 日付値または SAS 日時値ゼロと相対的に調整されます。SAS では、シフトされていない間隔の起点として、1960 年 1 月 1 日を参照時間に任意で定めています。シフトされている間隔については、1960 年 1 月 1 日との相対で定義されます。

たとえば、MONTH13 は 1960 年 1 月 1 日、1961 年 2 月 1 日、1962 年 3 月 1 日と続く間隔と、基準日 1960 年 1 月 1 日以前の 1958 年 12 月 1 日、1957 年 11 月 1 日と続く間隔を定義します。

もう 1 つの例として、WEEK6.13 という間隔指定は、第 2 金曜日を開始時点とする 6 週間隔を定義します。相対的な調整を 1960 年 1 月 1 日を含む期間に対して行うというこの規則によって、6 週間隔の第 2 金曜日がどの日付に対応するかが決定します。

よく使用される時間間隔を次の表に示します。

表 1.3 *よく使用される間隔と省略可能な乗数およびシフトインデックス*

間隔	説明
DAY3	3 日間隔
WEEK	日曜日を開始時点とする週間隔
WEEK.7	土曜日を開始時点とする週間隔
WEEK6.13	第 2 金曜日を開始時点とする 6 週間隔
WEEK2	第 1 日曜日を開始時点とする隔週間隔
WEEK1.1	WEEK と同じ
WEEK.2	月曜日を開始時点とする週間隔
WEEK6.3	第 1 火曜日を開始時点とする 6 週間隔
WEEK6.11	第 2 水曜日を開始時点とする 6 週間隔
WEEK4	第 1 日曜日を開始時点とする 4 週間隔
WEEKDAY	土曜日と日曜日を週末、5 日間を就業日とする週

間隔	説明
WEEKDAY1W	日曜日を週末日、6 日間を就業日とする週
WEEKDAY35W	火曜日と木曜日を週末日とし、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)
WEEKDAY17W	WEEKDAY と同じ
WEEKDAY67W	金曜日と土曜日を週末日とする 5 日間の週
WEEKDAY3.2	土曜日と日曜日を週末日とする 3 平日間隔(間隔は 1960 年 1 月 1 日を基準として調整されます。同一年にネストされる間隔については、1960 年 1 月 1 日にさかのぼって調整の必要はありません)
TENDAY4.2	TENDAY 単位の 2 番目を開始時点とする 40 日単位
SEMIMONTH2.2	月の 16 日目から翌月の 15 日目までの間隔
MONTH2.2	2 月-3 月、4 月-5 月、6 月-7 月、8 月-9 月、10 月-11 月、12 月-1 月(翌年)
MONTH2	1 月-2 月、3 月-4 月、5 月-6 月、7 月-8 月、9 月-10 月、11 月-12 月
QTR3.2	1960 年 2 月 1 日、1960 年 11 月 1 日、1961 年 8 月 1 日、1962 年 5 月 1 日と続く 9 か月間隔
SEMIYEAR.3	6 か月間隔、3 月-8 月および 9 月-2 月
YEAR.10	10 月に開始する会計年度
YEAR2.7	偶数年の 7 月を開始時点とする隔年間隔
YEAR2.19	奇数年の 7 月を開始時点とする隔年間隔
YEAR4.11	うるう年の 11 月を開始時点とする 4 年間隔(米国の大統領選挙の周期)
YEAR4.35	うるう年間の偶数年の 11 月を開始時点とする 4 年間隔(米国の中間選挙の周期)
DTMONTH13	1960 年 1 月 1 日午前零時を開始時点とする 13 か月間隔。1957 年 11 月 1 日、1958 年 12 月 1 日、1960 年 1 月 1 日、1961 年 2 月 1 日、1962 年 3 月 1 日など
HOUR8.7	午前 6 時、午後 2 時および午後 10 時を開始時点とする 8 時間間隔 (就業シフトに使えます)

interval の有効な値の完全なリストについては、表 7.3, “日時関数で使用される間隔,” (*SAS 言語リファレンス: 解説編*)を参照してください。

販売カレンダの間隔: ISO 8601 準拠

小売業界では、1 年のカレンダを 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 番目および 3 番目の数字は、それぞれ 1 番目、2 番目および 3 番目の期間に何週間あるかを指定します。

これらの形式から作成される間隔は、INTCINDEX、INTCK、INTCYCLE、INTFIT、INTFMT、INTGET、INTINDEX、INTNX、INTSEAS、INTSHIFT のいずれの関数でも使用できます。

詳細については、“小売りカレンダの間隔: ISO 8601 遵守” (*SAS 言語リファレンス: 解説編* 7 章)を参照してください。

カスタム時間間隔

カスタム時間間隔を使用する理由

標準の時間間隔(たとえば、QTR、MONTH および WEEK など)は、必ずしもデータに適合しません。さらに、標準の間隔で計測された時系列の中には、データ中にギャップが含まれているものがあります。たとえば、会計月を毎月 10 日に開始するとします。この場合、MONTH 間隔は毎月 1 日に始まるため、MONTH 間隔を使うのは適切ではありません。カスタム間隔を使用し、ビジネスとの親和性が高い周期でデータをモデリングし、圧縮によってデータ中のギャップを削除します。間隔は昇順で記述する必要があります。間隔間にギャップは存在できません。また、間隔は重複できません。

もう 1 つの例として、夜間は営業しないビジネスのデータを 1 時間ごとに収集するとします。この場合、DTHOUR 間隔を使用するとデータにギャップが生じ、標準の時系列分析で問題が発生する原因となる場合があります。また、日付間の営業日数を、祝日と週末を除いて数えるとします。ただし、祝日が数えられるのは、INTCK 関数を WEEKDAY 間隔と使用するときです。このような場合にカスタム間隔の使用が有効です。

SAS プログラムでカスタム時間間隔を使用する

カスタム間隔は SAS プログラム内のデータセットで定義できます。カスタム間隔を使用するには、各間隔で 2 つの操作を実行する必要があります。

1. INTERVALDS=システムオプションを OPTIONS ステートメントで使用し、データセット名とカスタム間隔名を関連付けます。

   INTERVALDS=システムオプションの引数の例を次に示します。ここでは、データセット StoreHoursDS とカスタム間隔 StoreHours を関連付けています。

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

   詳細については、“INTERVALDS= System Option” in *SAS System Options: Reference* を参照してください。

2. カスタム間隔を説明するデータセットを作成します。

   データセットには Begin 変数が含まれている必要があります。End 変数と Season 変数を含めることもできます。SAS プログラムで FORMAT ステートメントを使用し、Begin 変数と関連付ける必要があります。Begin 変数データに一致する SAS 日付、日時または数値形式をこの変数で指定します。End 変数が存在する場合は、これも FORMAT ステートメントに含めます。SAS 日付または SAS 日時形式でない数値形式は、値がオブザベーション番号を示し

ます。End 変数が存在しない場合は、次のオブザベーションの Begin 値より 1 少ない値が各オブザベーションの End の暗黙値となります。

カスタム間隔データセットの期間には、時系列でのバックキャスティングやフォーキャスティングなどの計算に加え、時系列の範囲外にまたがって行う可能性のあるその他の処理に必要な日付または時間が含まれている必要があります。

上記の手順に従ってカスタム間隔を定義した後は、標準時間間隔を指定できる SAS プロシジャと関数でカスタム間隔を指定します。

例 1: ビジネス展開している店舗の営業時間を INTNX 関数で作成する

次の DATA ステップは、月曜日から金曜日までの午前 9 時から午後 6 時と、土曜日の午前 9 時から午後 1 時までを営業時間とする、あるビジネスの StoreHoursDS データセットを作成します。ここでは"INTNX 関数" (567 ページ), を使用し、指定した時間間隔で日付値、時間値または日時値を増分し、日付値、時間値または日時値を返します。この例では、StoreHours が間隔、StoreHoursDS がユーザー指定の祝日を含むデータセットです。

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

proc print data=StoreHoursDS (obs=18);
run;
```

次の出力は、このカスタム間隔データセットの最初の 18 のオブザベーションを示しています。

画面 1.11 *店舗営業時間のカスタム間隔*

Store Hours Custom Interval

Obs	begin	end
1	01JAN09:09:00:00	01JAN09:09:59:59
2	01JAN09:10:00:00	01JAN09:10:59:59
3	01JAN09:11:00:00	01JAN09:11:59:59
4	01JAN09:12:00:00	01JAN09:12:59:59
5	01JAN09:13:00:00	01JAN09:13:59:59
6	01JAN09:14:00:00	01JAN09:14:59:59
7	01JAN09:15:00:00	01JAN09:15:59:59
8	01JAN09:16:00:00	01JAN09:16:59:59
9	01JAN09:17:00:00	01JAN09:17:59:59
10	02JAN09:09:00:00	02JAN09:09:59:59
11	02JAN09:10:00:00	02JAN09:10:59:59
12	02JAN09:11:00:00	02JAN09:11:59:59
13	02JAN09:12:00:00	02JAN09:12:59:59
14	02JAN09:13:00:00	02JAN09:13:59:59
15	02JAN09:14:00:00	02JAN09:14:59:59
16	02JAN09:15:00:00	02JAN09:15:59:59
17	02JAN09:16:00:00	02JAN09:16:59:59
18	02JAN09:17:00:00	02JAN09:17:59:59

例 2: 会計月のカスタム間隔を INTNX 関数で作成する

次の DATA ステップは、カスタム間隔 FiscalMonth を定義する FMDS データセットを作成します。このカスタム間隔は、会計月が毎月 10 日に始まるビジネスに対応します。INTNX 関数の SAME アライメントオプションで、INTNX 関数で生成される月の日付が Start 変数の日付と同じになるように指定します。MONTH 関数は Begin 変数の月を Season 変数に割り当て、月単位の季節性を指定します。

```
options intervalds=(FiscalMonth=FMDS);
data FMDS(keep=begin season);
start = '10JAN1999'd;
stop = '10JAN2001'd;
nmonths = intck('month',start,stop);
do i=0 to nmonths;
begin = intnx('month',start,i,'S');
season = month(begin);
output;
end;
format begin date9.;
run;
```

```
proc print data=FMDS;
title 'Fiscal Month Data';
run;
```

画面 1.12 *会計月のデータ*

Fiscal Month Data

Obs	begin	season
1	10JAN1999	1
2	10FEB1999	2
3	10MAR1999	3
4	10APR1999	4
5	10MAY1999	5
6	10JUN1999	6
7	10JUL1999	7
8	10AUG1999	8
9	10SEP1999	9
10	10OCT1999	10
11	10NOV1999	11
12	10DEC1999	12
13	10JAN2000	1
14	10FEB2000	2
15	10MAR2000	3
16	10APR2000	4
17	10MAY2000	5
18	10JUN2000	6
19	10JUL2000	7
20	10AUG2000	8
21	10SEP2000	9
22	10OCT2000	10
23	10NOV2000	11
24	10DEC2000	12
25	10JAN2001	1

カスタム間隔の FiscalMonth と標準の間隔の違いは次の例で確認できます。このプログラムの出力は、データの累計方法を比較します。FiscalMonth 間隔では、前月に始まる間隔に月の最初の 9 日間の値が累計されています。標準の MONTH 間隔では、カレンダ月に従って月の最初の 9 日間の値が累計されています。

```
data sales(keep=date sales);
do date = '01JAN2000'd to '31DEC2000'd;
```

```
month = MONTH(date);
dayofmonth = DAY(date);
sales = 0;
if (dayofmonth lt 10) then sales= month/9;
output;
end;
format date monyy.;
run;

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

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

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

title 'Standard Monthly Data and Fiscal Month Data';

proc print data=compare;
run;
```

画面 1.13 標準の月次データと会計月データの比較

Standard Monthly Data and Fiscal Month Data

Obs	date	FM_sales	SM_sales
1	10DEC1999	1	.
2	01JAN2000	.	1
3	10JAN2000	2	.
4	01FEB2000	.	2
5	10FEB2000	3	.
6	01MAR2000	.	3
7	10MAR2000	4	.
8	01APR2000	.	4
9	10APR2000	5	.
10	01MAY2000	.	5
11	10MAY2000	6	.
12	01JUN2000	.	6
13	10JUN2000	7	.
14	01JUL2000	.	7
15	10JUL2000	8	.
16	01AUG2000	.	8
17	10AUG2000	9	.
18	01SEP2000	.	9
19	10SEP2000	10	.
20	01OCT2000	.	10
21	10OCT2000	11	.
22	01NOV2000	.	11
23	10NOV2000	12	.
24	01DEC2000	.	12
25	10DEC2000	0	.

例 3: INTCK 関数でカスタム間隔を使用する

次の例では、INTCK 関数でカスタム間隔を使用し、営業日を数える場合に祝日を除いています。

```
options intervalds=(BankingDays=BankDayDS);
data BankDayDS(keep=begin);
start = '15DEC1998'd;
stop = '15JAN2002'd;
nwkdays = intck('weekday',start,stop);
do i = 0 to nwkdays;
begin = intnx('weekday',start,i);
year = year(begin);
if begin ne holiday('NEWYEAR',year) and
```

```
begin ne holiday('MLK',year) and
begin ne holiday('USPRESIDENTS',year) and
begin ne holiday('MEMORIAL',year) and
begin ne holiday('USINDEPENDENCE',year) and
begin ne holiday('LABOR',year) and
begin ne holiday('COLUMBUS',year) and
begin ne holiday('VETERANS',year) and
begin ne holiday('THANKSGIVING',year) and
begin ne holiday('CHRISTMAS',year) then
output;
end;
format begin date9.;
run;
data CountDays;
start = '01JAN1999'd;
stop = '31DEC2001'd;
ActualDays = intck('DAYS',start,stop);
Weekdays = intck('WEEKDAYS',start,stop);
BankDays = intck('BankingDays',start,stop);
format start stop date9.;
run;

title 'Methods of Counting Days';

proc print data=CountDays;
run;
```

画面 1.14 *銀行営業日のカスタム間隔*

Methods of Counting Days

Obs	start	stop	ActualDays	Weekdays	BankDays
1	01JAN1999	31DEC2001	1095	781	757

カスタム間隔名のベストプラクティス

カスタム間隔名を作成するときのベストプラクティスを次の各項目に示します。

- カスタム間隔名が既存の SAS 間隔名と競合しないようにします。たとえば、仮に BASE という SAS 間隔名がある場合は、次の形式をカスタム間隔名に使用しないようにします。
  - BASE
  - BASE*m*
  - BASE*m.n*
  - DTBASE
  - DTBASE*m*
  - DTBASE*m.n*

次のパラグラフでは変数について説明します。

m
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。たとえば、間隔 YEAR2 は 2 年の期間の間隔、つまり隔年です。

n
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。たとえば、YEAR.3 を指定すると 1 年の期間がシフトされて、毎年カレンダ年の 3 月 1 日に開始し、翌年の 2 月末に終了します。

CUSTBASE のようなカスタム間隔を定義すると、*m.n* を使用できます。

これらの規則があるため、カスタム間隔名の先頭に DT を付けたり、カスタム間隔名の末尾に数字を付けたりしないでください。

- カスタム間隔を安定して動作させるには、次のいずれかの形式を使う必要があります。

  date-format(開始値および終了値に使用)
  SAS 日付値で使用する間隔を指定します。

  datetime-format(開始値および終了値に使用)
  SAS 日時値で使用する間隔を指定します。

  number-format(開始値および終了値に使用)
  SAS 日時値で使用するオブザベーション番号を指定します。

- 開始値と終了値は同じ種類である必要があります。つまり、両方の値を日付値、日時値またはオブザベーション番号とします。
- カスタム間隔の計算は、最初の開始値より前、または最後の終了値より後で実行できません。開始変数のみを使用すると、計算できる最後の終了値は、最後の開始値である-1 です。時系列をフォーキャストまたはバックキャストする場合は、フォーキャスト値とバックキャスト値の時間定義が必要です。
- CUSTBASE*m*.2 は、定義上、最初の間隔の開始がデータセットの開始よりも前(– (*m*– 2)番目のオブザベーション)となるため、最初に日付値を求めるための開始期間を計算できません。たとえば、CUSTBASE4.2 という間隔があるとします。最初の間隔は最初のオブザベーションよりも前に開始します。

  ```
  OBS
  -2 Start of partial CUSTBASE4.2 interval observation: -(4-2) = -2.
  -1
  0
  1 End of partial CUSTBASE4.2 interval observation: This is the first
  observation in the data set.
  2 Start of first complete CUSTBASE4.2 interval.
  3
  4
  5 End of first complete CUSTBASE4.2 interval.
  6 Start of 2nd CUSTBASE4.2 interval.
  ```

  INTNX 関数を実行すると結果には OBS –2 に関連付けられる日付を返す必要がありますが、その日付は存在しません。

  INTNX('CUSTBASE4.2', *date-at-obs1*, 0, 'B');

- 季節性インデックスを定義するカスタム間隔データセットには *season* と名付けた変数を含めます。この結果は、**INTINDEX (*'interval*, *date*)*;と似た結果になります。***

次の例では、データセットはカスタム間隔 CUSTWEEK と関連付けられています。

```
Obs begin season
1 27DEC59 52
2 03JAN60 1
3 10JAN60 2
4 17JAN60 3
5 24JAN60 4
6 31JAN60 5
```

カスタム間隔関数を使用したときの結果例を次に示します。

INTINDEX ('CUSTWEEK', '*03JAN60'D*);
値 1 を返します。

INTSEAS ('CUSTWEEK');
季節で最大の値である値 52 を返します。

INTCYCLE ('CUSTWEEK');
CUSTWEEK52 を返します。これは CUSTBASE*max*(*season*)です。

INTCINDEX ('CUSTWEEK', '27DEC59'D);
値 1 を返します。

INTCINDEX('CUSTWEEK', '03JAN60'D)
値 2 を返します。

季節が *season* の前の値より小さくなると、新しい周期が始まります。

- 季節性は、season1、season2、season3 などと、季節が識別されると生じます。すべての季節が season1 と識別されると、季節性は存在しません。また、トリビアルな季節性と呼ばれる季節性も存在しません。

  トリビアルな季節性を利用できるのは、CUSTBASE*m* の形式の間隔のみです。*season* がデータセットに含まれていない場合、トリビアルな季節性が有効です。

- 開始変数の形式がデータセットに含まれている場合、INTFMT ('CUSTBASE', 'l')または INTFMT ('CUSTBASE', 's')によって生成されるメッセージが表示されます。データセットで指定されている形式に基づく形式を勧めるメッセージが表示されます。

- **INTSHIFT ('CUSTBASE');**または **INTSHIFT ('CUSTBASE*ms*');** を実行すると、CUSTBASE の値が返されます。

- 間隔 CUSTBASE、CUSTBASE*m* および CUSTBASE*m.s* は、INTNX、INTCK および INTTEST と使用すると期待通りに機能します。

Perl 正規表現(PRX)を使用したパターン照合

パターン照合の定義

パターン照合を使うと、文字列から複数の一致パターンを一度に検索し、抽出できます。また、文字列中で複数の置換を一度に行うこともできます。これらを実行するには、DATA ステップで PRX の関数と CALL ルーチンを使用します。

たとえば、ある文字列の出現を複数検索し、それらの文字列を他の文字列と置換できます。また、ソースファイルから文字列を検索し、一致する位置を返すことができます。ファイルの中で重複している単語も検索できます。

Perl 正規表現(PRX)の関数と CALL ルーチンの定義

Perl 正規表現(PRX)の関数と CALL ルーチンとは、文字列を解析するパターン照合言語に Perl の修正バージョンを使用する関数グループと CALL ルーチンを指します。次のタスクを実行できます。

- 文字列内で文字パターンを検索
- 文字列から部分文字列を抽出
- テキストを検索して他のテキストに置換
- Web ログやその他のテキストデータなど、大量のテキストを解析

Perl 正規表現は関数と CALL ルーチンの文字列一致カテゴリを構成します。これらの関数と CALL ルーチンの簡単な説明については、本書の辞書セクションのカテゴリ別の関数と CALL ルーチンを参照してください。 .

DATA ステップで Perl 正規表現を使用する利点

DATA ステップで Perl 正規表現を使えば、テキストの検索と置換のオプションが強化されます。Perl 正規表現を使用すると、次のタスクを実行できます。

- データの検証
- テキストの置換
- 文字列から部分文字列を抽出

正規表現を使わずに SAS プログラムを記述し、Perl 正規表現を使用したときと同じ結果を得ることは可能です。ただし、正規表現を使わないコードでは、文字列中の文字位置を扱い、文字列の一部を操作するために、より多くの関数呼び出しが必要となります。

Perl 正規表現は、これらのステップのすべてではないものの、ほとんどを 1 つの表現にまとめます。完成されたコードはより工ラーが起きにくく、保守が簡単で読みやすいものとなります。

DATA ステップで Perl 正規表現を使用する

Perl 正規表現の構文

Perl 正規表現のコンポーネント

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。メタ文字を使用すると、SAS は特別なアクションを実行します。これには、強制的に特定の位置から照合を開始したり、特定の文字セットとの照合を実行したりするなどのアクションがあります。デフォルトの区切り文字はフォワードスラッシュのペアです。メタ文字と、それらのメタ文字に一致する値を次の 2 つの例に示します。

- メタ文字\d を使用すると、0–9 間の数字に一致します。

- /\dt/を使用すると、文字列“Raleigh, NC 27506”中の数字に一致します。

PRX のメタ文字のリストは、“Perl 正規表現(PRX)のメタ文字テーブル” (973 ページ)にあります。 メタ文字の完全なリストについては、Perl のドキュメントを参照してください。

文字列で一致を検索するための基本的な構文

ソース文字列中で一致した値の位置を検索するには、PRXMATCH 関数を使用します。PRXMATCH の一般的な形式を次に示します。

/search-string/source-string/

次の例では、PRXMATCH 関数を使用して *search-string* の位置を *source-string* の中から検索します。

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

world は文字列 *Hello world!*の 7 番目の位置に出現するため、PRXMATCH の結果は値 7 です。

テキストの検索と置換の基本的な構文

テキストの検索と置換の基本的な構文は次の形式です。

s/*regular-expression/replacement-string/*

次の例では、PRXCHANGE 関数を使って置換を行う方法を示しています。

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

引数

s
　置換に使用するメタ文字を指定します。

world
　正規表現を指定します。

planet
　world を置換する値を指定指定します。

1
　1 回の一致検出で検索を終了することを指定します。

Hello World!
　検索するソース文字列を指定します。

置換の結果は **Hello planet** となります。

テキストの検索と置換の基本的な構文を使用するもう 1 つの例

もう 1 つの例で、PRXCHANGE 関数を使用して値 *Jones, Fred* を *Fred Jones* に変更します。

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

この例では、Perl 正規表現は **s/(¥w+), (¥w+)/$2 $1** です。一致の検索回数は–1 です。ソース文字列は'Jones, Fred'です。値–1 は、ソースの末尾に到達するまで一致したパターンを置換し続けることを指定します。

この Perl 正規表現は次の要素に分けられます。

s
　置換の正規表現を指定します。

(\w+)
1 つ以上の単語文字(英数字とアンダースコア)に一致します。かっこは、キャプチャバッファ 1 に値を格納することを示します。

,<空白>
カンマと空白 1 つに一致します。

(\w+)
1 つ以上の単語文字(英数字とアンダースコア)に一致します。かっこは、キャプチャバッファ 2 に値を格納することを示します。

/
正規表現と置換文字列の間の区切り文字列です。

$2
キャプチャバッファ 2 の値を置換する置換文字列部分です。この例ではカンマの後の単語で、結果に置換文字列を配置します。

<空白>
結果に空白を配置します。

$1
キャプチャバッファ 1 を結果に配置します。この例では、カンマの前の単語です。

テキストの置換

次の例では\u メタ文字と\L メタ文字を使用し、MCLAUREN の 2 番目の文字を小文字に置換します。

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

SAS は次の出力を ログに書き込みます。

```
x=McLAUREN
```

例 1: データの検証

文字列内に文字パターンが存在するかを検証できます。たとえば、文字列に正しい形式の電話番号が含まれているかどうかを調べて確認できます。この種のテストをデータ検証と呼びます。

電話番号のリストを検証する例を次に示します。有効な電話番号の形式は、**(XXX) XXX-XXXX** または **XXX-XXX-XXXX** のいずれかです。

```
data _null_; 1
if _N_ = 1 then
do;
paren = "¥([2-9]¥d¥d¥) ?[2-9]¥d¥d-¥d¥d¥d¥d"; 2
dash = "[2-9]¥d¥d-[2-9]¥d¥d-¥d¥d¥d¥d"; 3
expression = "/(" || paren || ")|(" || dash || ")/"; 4
retain re;
re = prxparse(expression); 5
if missing(re) then 6
do;
putlog "ERROR: Invalid expression " expression; 7
stop;
```

```
end;
end;

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

if ^prxmatch(re, home) then 8
putlog "NOTE: Invalid home phone number for " first last home;

if ^prxmatch(re, business) then 9
putlog "NOTE: Invalid business phone number for " first last business;

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

次の項目は、上記の DATA ステップで番号が付けられている行に対応しています。

1 DATA ステップを作成します。

2 (XXX)XXX-XXXX に一致する電話番号を識別する Perl 正規表現を記述し、変数 PAREN を割り当てて結果を保持します。次の構文要素を使用して Perl 正規表現を記述します。

\(	市外局番の開始かっこに一致します。
[2–9]	市外局番の最初の番号である数字 2–9 に一致します。
\d	市外局番の 2 番目の番号である数字に一致します。
\d	市外局番の 3 番目の番号である数字に一致します。
\)	市外局番の閉じかっこに一致します。
<space>?	直前の部分表現である空白に 0 回または 1 回一致します。Perl 正規表現では、空白に意味があります。これらは検索対象のテキストの空白に一致します。この例のように、疑問符の直前に空白があると、電話番号のこの位置の空白ゼロ個または 1 個に一致するパターンとなります。

3 XXX-XXX-XXXX に一致する電話番号を識別する Perl 正規表現を記述し、変数 DASH を割り当てて結果を保持します。

4 (XXX)XXX-XXXX と XXX—XXX—XXXX に一致する正規表現を連結する Perl 正規表現を記述します。連結することで、両方の電話番号形式を 1 つの正規表現で検索できるようになります。

PAREN と DASH の正規表現はかっこ内に配置されています。PAREN と DASH の間にあるメタ文字の棒(|)は、コンパイラに対していずれかのパターンとの一致を指示します。パターン全体を囲むスラッシュは、正規表現の開始位置と終了位置をコンパイラに伝えます。

5 Perl 正規表現を PRXPARSE に渡し、表現をコンパイルします。PRXPARSE はコンパイルされたパターンに対して値を返します。他の Perl 正規表現の関

数と CALL ルーチンで値を使用し、コンパイルされた Perl 正規表現を使った処理を SAS で実行できます。

6 MISSING 関数で正規表現が正常にコンパイルされたかどうかを確認します。

7 PUTLOG ステートメントを使用し、正規表現がコンパイルされなかった場合はエラーメッセージを SAS ログに書き込みます。

8 有効な自宅電話番号を検索します。PRXMATCH は PRXPARSE からの値を検索テキストと合わせて使用し、検索テキストで正規表現が検出された位置を返します。自宅電話番号との一致がない場合、PUTLOG ステートメントは SAS ログにメモを書き込みます。

9 有効な会社電話番号を検索します。PRXMATCH は PRXPARSE からの値を検索テキストと合わせて使用し、検索テキストで正規表現が検出された位置を返します。会社電話番号との一致がない場合、PUTLOG ステートメントは SAS ログにメモを書き込みます。

ログ 1.3 データの検証からの出力

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

例 2: テキストの一致と置換

この例では、Perl 正規表現を使用して一致を検索し、一致した文字を他の文字に置換します。PRXPARSE で正規表現をコンパイルした後、PRXCHANGE で一致を検索して置換を実行します。ここでは、小なり記号のすべての出現を<に置換します。これはテキストを HTML に変換するときに一般的に行われる置換処理です。

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

次の項目は、上記の DATA ステップで番号が付けられている行に対応しています。

1 DATA ステップを作成します。

2 SAS 変数を作成しないで、入力バッファに入力データレコードを入れます。

3 PRXCHANGE ルーチンを呼び出し、パターンによる置換を実行します。正規表現の形式は **s/regular-expression/replacement-text/**です。 正規表現の前の **s** は、これが置換正規表現であることを示します。–1 は PRXCHANGE に渡される特別な値で、置換できるものはすべて置換することを示します。

4 PUT ステートメントの_INFILE_
オプションを使用し、現在の出力行をログに書き込みます。

5 入力ファイルを識別します。

ログ 1.4 テキストの置換からの出力

```
x + y < 15
x < 10 < y
y < 11
```

PRXCHANGE に正規表現を渡して結果を返すことができるため、PROC SQL クエリから PRXCHANGE を呼び出せます。次のクエリは、前述の例と同じ文字置換を行う列を作成します。クエリは入力テーブルから **text_lines** を読み込み、列 **line** のテキストを置換し、結果を **html_line** という名前の列に配置します。

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

例 3: 文字列から部分文字列を抽出する

Perl 正規表現を使用すれば、文字列からテキストを検索し、簡単に抽出できます。この例では、DATA ステップでノースカロライナ州の会社電話番号のサブセットを作成します。このプログラムは市外局番を抽出し、ノースカロライナ州の市外局番のリストとの照合を行います。

```
data _null_; 1
if _N_ = 1 then
do;
paren = "¥((¥d¥d¥d)¥) ?¥d¥d¥d-¥d¥d¥d¥d"; 2
dash = "(¥d¥d¥d)-¥d¥d¥d-¥d¥d¥d¥d"; 3
regexp = "/(" || paren || ")|(" || dash || ")/"; 4
retain re;
re = prxparse(regexp); 5
if missing(re) then 6
do;
putlog "ERROR: Invalid regexp " regexp; 7
stop;
end;

retain areacode_re;
areacode_re = prxparse("/828|336|704|910|919|252/"); 8
if missing(areacode_re) then
do;
putlog "ERROR: Invalid area code regexp";
stop;
end;
end;

length first last home business $ 25;
length areacode $ 3;
input first last home business;
```

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

if prxmatch(re, business) then 9
do;
which_format = prxparen(re); 10
call prxposn(re, which_format, pos, len); 11
areacode = substr(business, pos, len);
if prxmatch(areacode_re, areacode) then 12
put "In North Carolina: " first last business;
end;
else
putlog "NOTE: Invalid business phone number for " first last business;
datalines;
Jerome Johnson (919)319-1677 (919)846-2198
Romeo Montague 800-899-2164 360-973-6201
Imani Rashid (508)852-2146 (508)366-9821
Palinor Kent 704-782-4673 704-782-3199
Ruby Archuleta 905-384-2839 905-328-3892
Takei Ito 704-298-2145 704-298-4738
Tom Joad 515-372-4829 515-389-2838
;
```

1 DATA ステップを作成します。

2 (XXX)XXX-XXXX に一致する電話番号を識別する Perl 正規表現を記述し、変数 PAREN を割り当てて結果を保持します。次の構文要素を使用して Perl 正規表現を記述します。

\(市外局番の開始かっこに一致します。開始かっこは部分一致の開始を表します。

[2–9] 数字 2–9 に一致します。

\d 市外局番の 2 番目の番号である数字に一致します。

\d 市外局番の 3 番目の番号である数字に一致します。.

\) 市外局番の閉じかっこに一致します。閉じかっこは部分一致の終了を表します。

? 直前の部分表現である空白に 0 回または 1 回一致します。Perl 正規表現では、空白に意味があります。これらは検索対象のテキストの空白に一致します。この例のように、疑問符の直前に空白があると、電話番号のこの位置の空白ゼロ個または 1 個に一致するパターンとなります。

3 XXX-XXX-XXXX に一致する電話番号を識別する Perl 正規表現を記述し、変数 DASH を割り当てて結果を保持します。

4 (XXX)XXX-XXXX と XXX—XXX—XXXX に一致する正規表現を連結する Perl 正規表現を記述します。連結することで、両方の電話番号形式を 1 つの正規表現で検索できるようになります。

PAREN と DASH の正規表現はかっこ内に配置されています。PAREN と DASH の間にあるメタ文字の棒(|)は、コンパイラに対していずれかのパターンとの一致を指示します。パターン全体を囲むスラッシュは、正規表現の開始位置と終了位置をコンパイラに伝えます。

5 Perl 正規表現を PRXPARSE に渡し、表現をコンパイルします。PRXPARSE はコンパイルされたパターンに対して値を返します。他の Perl 正規表現の関

数と CALL ルーチンで値を使用し、コンパイルされた Perl 正規表現を使った処理を SAS で実行できます。

6 MISSING 関数で正規表現がエラーなしにコンパイルされたかどうかを確認します。

7 PUTLOG ステートメントを使用し、正規表現がコンパイルされなかった場合はエラーメッセージを SAS ログに書き込みます。

8 有効なノースカロライナ州の市外局番を文字列から検索する Perl 正規表現をコンパイルします。

9 有効な会社電話番号を検索します。

10 PRXPAREN 関数でどちらの部分一致を使うかを判定します。PRXPAREN は一致した最後の部分一致を返します。市外局番が(XXX)形式に一致すると、PRXPAREN は値 2 を返します。市外局番が XXX 形式に一致すると、PRXPAREN は値 4 を返します。

11 PRXPOSN ルーチンを呼び出し、部分一致の位置と長さを取得します。

12 PRXPOSN ルーチンを呼び出し、部分一致の位置と長さを取得します。

ログ 1.5 文字列からの部分文字列抽出の出力

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

例 4: 文字列から部分文字列を抽出するもう 1 つの例

この例では、位置変数と長さ変数のかわりに、PRXPOSN 関数を元の検索テキストに渡します。PRXPOSN は一致したテキストを返します。

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

input first last phone & 16.;
if prxmatch(re, phone) then do; 4
area_code = prxposn(re, 1, phone); 5
if area_code ^in ("828"
"336"
"704"
"910"
"919"
"252") then
putlog "NOTE: Not in North Carolina: "
first last phone; 6
end;
datalines; 7
Thomas Archer (919)319-1677
Lucy Mallory (800)899-2164
Tom Joad (508)852-2146
Laurie Jorgensen (252)352-7583
```

```
;
run;
```

次の項目は、上記の DATA ステップで番号が付けられている行に対応しています。

1 DATA ステップを作成します。

2 これが最初のレコードの場合、*re* の値を検索します。

3 パターン照合を行う Perl 正規表現を記述します。次の構文要素を使用して Perl 正規表現を記述します。

/	正規表現の開始区切り文字です。
\(	次の文字入力を文字またはリテラルに指定します。
(	部分一致の開始を表します。
[2–9]	数字 2–9 に一致し、市外局番の最初の番号を識別します。
\d	市外局番の 2 番目の番号である数字に一致します。
\d	市外局番の 3 番目の番号である数字に一致します。
\)	市外局番の閉じかっこに一致します。閉じかっこは部分一致の終了を表します。
?	直前の部分表現である空白に 0 回または 1 回一致します。Perl 正規表現では、空白に意味があります。空白は検索対象のテキストの空白に一致します。この例のように、疑問符の直前に空白があると、電話番号のこの位置の空白ゼロ個または 1 個に一致するパターンとなります。
||	連結演算子です。
[2–9]	数字 2–9 に一致し、7 桁の電話番号の最初の番号を識別します。
\d	7 桁の電話番号の 2 番目の番号である数字に一致します。
\d	7 桁の電話番号の 3 番目の番号である数字に一致します。
–	市外局番の後の電話番号の最初の 3 つの数字と最後の 4 つの数字の間のハイフンです。
\d	7 桁の電話番号の 4 番目の番号である数字に一致します。
\d	7 桁の電話番号の 5 番目の番号である数字に一致します。
\d	7 桁の電話番号の 6 番目の番号である数字に一致します。
\d	7 桁の電話番号の 7 番目の番号である数字に一致します。
/	正規表現の終了区切り文字です。

4 文字列が開始する位置を返します。

5 市外局番が開始する位置を識別します。

6 リストから市外局番を検索します。市外局番が有効なノースカロライナ州の番号でない場合、PUTLOG ステートメントを使用して SAS ログにメモを書き込みます。

7 入力ファイルを識別します。

ログ 1.6 文字列からの部分文字列抽出の出力

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

SAS ログに Perl デバッグ出力を書き込む

DATA ステップは CALL PRXDEBUG ルーチンでのデバッグをサポートします。CALL PRXDEBUG を使用して、SAS ログに送られる Perl デバッグ出力をオンまたはオフにできます。

次の例では、SAS ログに Perl デバッグ出力を書き込みます。

```
data _null_;

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

/* CALL PRXDEBUG(0) turns off Perl debug output. */
call prxdebug(0);
run;
```

SAS は次の出力をログに書き込みます。

ログ 1.7 SAS デバッグ出力

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

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

Perl デバッグ出力の詳細については、"CALL PRXDEBUG ルーチン" (196 ページ)を参照してください。

Perl Artistic ライセンスの遵守

Perl 正規表現は、SAS® 9 以降でサポートされています。

PRX 関数は、正規表現のコンパイルと照合に Perl 5.6.1 の修正バージョンを使用します。SAS で利用できるよう、Perl はライブラリにコンパイルされています。このライブラリは SAS® 9 に付属しています。Perl 5.6.1 ファイルのオリジナル版と修正版は、**テクニカルサポート Web サイト**から ZIP ファイルで自由に入手可能です。この ZIP ファイルは Perl Artistic ライセンスの遵守を目的として提供するものであり、PRX 関数の使用には必要ありません。修正が加えられているファイルについては、各ファイルの最上部のコメントブロックに、変更内容と変更がいつ行われたかが記載されています。実行形式には、Perl 標準でない名前が付けられています。Perl の標準バージョンは、Perl の Web サイトから入手できます。

PRX 関数から利用できるのは Perl 正規表現のみです。Perl 言語のその他の部分は利用できません。修正バージョンの Perl 正規表現は、次の項目をサポートしません。

- Perl 変数(キャプチャバッファ変数$1 - $n を除きます。これらはサポート対象です)。
- 正規表現オプションの/c と/g。また、置換の/e オプション。
- SAS 9.0 での正規表現オプションの/o。(SAS 9.1 以降ではサポート対象です)。

- \N{name}構文を使用する名前付き文字。
- メタ文字の\pP、\PP および\X。
- 正規表現内での、(?{code})、(??{code})および(?p{code})の構文を含む Perl コードの実行。
- Unicode パターン照合。
- ?PATTERN?の使用。?は通常の正規表現の開始区切り文字と終了区切り文字として扱われます。
- メタ文字の\G。
- パターンと置換テキストの間の Perl コメント。たとえば、s{regexp} # perl comment {replacement}はサポートされません。
- m/\\\/を使用したバックスラッシュの一致。バックスラッシュに一致させるには m/\\/を使用します。

Web アプリケーション用の Base SAS 関数

Base SAS ソフトウェアには、Web 関連コンテンツを操作する 4 つの関数が用意されています。HTMLENCODE と URLENCODE はエンコード済み文字列を返します。HTMLDECODE と URLDECODE はデコード済み文字列を返します。

2 章

SAS 関数と CALL ルーチンの辞書

他の SAS ドキュメントに記載されている SAS 関数と CALL ルーチン

関数と Call ルーチンは、関連する題材とともに次のドキュメントにも記載されています。

- SAS Companion for Windows
- SAS Companion for z/OS
- Data Quality Server Reference
- Logging: Configuration and Programming Reference
- SAS Macro Language Reference
- National Language Support: Reference Guide

カテゴリ別の SAS 関数と CALL ルーチン

SAS 関数と CALL ルーチンのカテゴリは次のとおりです。

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。
配列	配列の情報を返します。
論理ビット演算関数	引数のビットごとの論理演算結果を返します。
文字関数	文字データに基づく情報を返します。
文字列マッチング	Perl 正規表現からの情報を返します。
組み合わせ関数	組み合わせと順列を生成します。
日付と時間	時間間隔を含む日付値と時間値を返します。
記述統計	平均値、中央値および標準偏差などの統計値を返します。
距離	測地距離を返します。
外部ファイル	外部ファイルに関連付けられた情報を返します。
外部ルーチン	文字または数値を返すか、リターンコードなしでルーチンを呼び出します。
財務関数	利息、定期的支払い、減価償却、株式取引のヨーロピアンオプションの価格などの財務値を計算します。
双曲線関数	正弦、余弦、正接などの双曲線計算を実行します。
マクロ	マクロ変数に値を割り当て、マクロ変数の値を返し、マクロ変数がグローバルスコープかローカルスコープかを判断し、マクロ変数が存在するかどうかを識別します。
数学関数	階乗、絶対値、ファジー比較、対数などの数値計算を実行します。
数値	式の真、偽、欠損に基づいて数値を返すか、インストールされている SAS のバージョンのソフトウェアイメージが存在するかどうかを判断します。
確率	$\chi 2$ 乗分布または Poisson 分布などからの確率計算を返します。
分位点	特定の分布からの分位点を返します。
乱数	特定の分布からのランダム変量を返します。
SAS ファイル I/O 関数	SAS ファイルの情報を返します。
検索	文字または数値を検索します。
並べ替え	文字引数または数値引数の値を並べ替えます。
特殊関数	メモリアドレスを返して保存し、メモリに値を直接書き込み、プログラムの実行を中断し、実行用の動作環境のコマンドを

	サブミットし、SAS システムまたはグラフィックオプションの値を返し、実行時に出力形式と入力形式を指定し、システムリターンコードを返し、UUID を返し、製品がライセンスされているかどうかを判断し、SAS 処理のその他の情報を返します。
州コード/郵便番号	郵便番号、FIPS コード、州および市区町村名、郵便番号間の測地距離を返します。
三角関数	正弦、余弦、正接などの三角関数値を返します。
切り捨て関数	多くの場合ファジーまたはゼロファジーを使用して、切り捨てた数値を返します。
変数制御	変数のラベルを割り当て、SAS データセットを DATA ステップまたはマクロ変数にリンクし、変数名を割り当てます。
変数情報	名前、種類、長さ、入力形式名、ラベル、その他の変数情報を返します。
Web サービス	Web サービスまたは SAS 登録 Web サービスを呼び出します。
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。

次の表に、カテゴリ別の SAS 関数と CALL ルーチンを示します。

カテゴリ	言語要素	説明
SAS ファイル I/O 関数	ATTRC 関数 (p. 127)	SAS データセットの文字属性の値を返します。
	ATTRN 関数 (p. 129)	SAS データセットの数値属性の値を返します。
	CEXIST 関数 (p. 290)	SAS カタログまたは SAS カタログエントリの存在を確認します。
	CLOSE 関数 (p. 297)	SAS データセットを閉じます。
	CUROBS 関数 (p. 343)	現在のオブザベーションのオブザベーション番号を返します。
	DROPNOTE 関数 (p. 380)	SAS データセットまたは外部ファイルからメモマーカーを削除します。
	DSNAME 関数 (p. 381)	データセット識別子と関連付けられた SAS データセット名を返します。
	ENVLEN 関数 (p. 385)	環境変数の長さを返します。
	EXIST 関数 (p. 389)	SAS ライブラリメンバの存在を確認します。
	FETCH 関数 (p. 398)	SAS データセットから次の非削除対象のオブザベーションをデータセットデータベクトル(DDV)に読み込みます。
	FETCHOBS 関数 (p. 399)	SAS データセットから指定したオブザベーションをデータセットデータベクトル(DDV)に読み込みます。
	GETVARC 関数 (p. 510)	SAS データセットの文字変数の値を返します。

カテゴリ	言語要素	説明
	GETVARN 関数 (p. 511)	SAS データセットの数値変数の値を返します。
	IORCMSG 関数 (p. 583)	_IORC_の書式化されたエラーメッセージを返します。
	LIBNAME 関数 (p. 618)	SAS ライブラリのライブラリ参照名の割り当てを設定または取り消します。
	LIBREF 関数 (p. 620)	ライブラリ参照名が割り当てられていることを確認します。
	NOTE 関数 (p. 674)	SAS データセットの現在のオブザベーションのオブザベーション ID を返します。
	OPEN 関数 (p. 697)	SAS データセットを開きます。
	PATHNAME 関数 (p. 700)	外部ファイルや SAS ライブラリの物理名または空白を返します。
	POINT 関数 (p. 725)	NOTE 関数で識別されたオブザベーションを検索します。
	RENAME 関数 (p. 803)	SAS ライブラリのメンバ、SAS カタログ内のエントリ、外部ファイル、ディレクトリのいずれかの名前を変更します。
	REWIND 関数 (p. 807)	データセットポインタを SAS データセットの先頭に配置します。
	SYSEXIST 関数 (p. 877)	環境内に動作環境変数が存在するかどうかを示す値を返します。
	SYSMSG 関数 (p. 880)	最後のデータセットまたは外部ファイル関数の処理のエラーまたは警告メッセージを返します。
	SYSRC 関数 (p. 884)	システムエラー番号を返します。
	VARFMT 関数 (p. 908)	SAS データセット変数に割り当てられた出力形式を返します。
	VARINFMT 関数 (p. 910)	SAS データセット変数に割り当てられた入力形式を返します。
	VARLABEL 関数 (p. 911)	SAS データセット変数に割り当てられたラベルを返します。
	VARLEN 関数 (p. 912)	SAS データセット変数の長さを返します。
	VARNAME 関数 (p. 913)	SAS データセット変数の名前を返します。
	VARNUM 関数 (p. 914)	SAS データセット内の変数の位置番号を返します。
	VARTYPE 関数 (p. 917)	SAS データセット変数のデータの種類を返します。
Special	GETOPTION Function (p. 503)	Returns the value of a SAS system or graphics option.

カテゴリ	言語要素	説明
Web サービス	SOAPWEB 関数 (p. 841)	基本 Web 認証を使用して Web サービスを呼び出します。認証情報は引数で指定します。
	SOAPWEBMETA 関数 (p. 843)	基本 Web 認証を使用して Web サービスを呼び出します。認証ドメインの認証情報はメタデータから取得されます。
	SOAPWIPSERVICE 関数 (p. 845)	WS セキュリティ認証を使用して SAS に登録されている Web サービスを呼び出します。認証情報は引数で指定します。
	SOAPWIPSRS 関数 (p. 847)	WS セキュリティ認証を使用して SAS に登録されている Web サービスを呼び出します。認証情報は引数で指定します。
	SOAPWS 関数 (p. 849)	WS セキュリティ認証を使用して Web サービスを呼び出します。認証情報は引数で指定します。
	SOAPWSMETA 関数 (p. 851)	WS セキュリティ認証を使用して Web サービスを呼び出します。指定された認証ドメインの認証情報はメタデータから取得します。
Web ツール	HTMLDECODE 関数 (p. 523)	HTML 数値文字参照または HTML 文字実体参照を含む文字列をデコードし、デコードされた文字列を返します。
	HTMLENCODE 関数 (p. 524)	HTML 文字実体参照を使用して文字をエンコードし、エンコードされた文字列を返します。
	URLDECODE 関数 (p. 903)	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. 905)	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。
外部ファイル	DCLOSE 関数 (p. 354)	DOPEN 関数によって開かれたディレクトリを閉じます。
	DCREATE 関数 (p. 356)	新しい外部ディレクトリの完全なパス名を返します。
	DINFO 関数 (p. 371)	ディレクトリの情報を返します。
	DNUM 関数 (p. 374)	ディレクトリ内のメンバ数を返します。
	DOPEN 関数 (p. 375)	ディレクトリを開き、ディレクトリ識別子の値を返します。
	DOPTNAME 関数 (p. 377)	ディレクトリ属性情報を返します。
	DOPTNUM 関数 (p. 378)	ディレクトリで使用可能な情報項目数を返します。
	DREAD 関数 (p. 379)	ディレクトリのメンバ名を返します。
	DROPNOTE 関数 (p. 380)	SAS データセットまたは外部ファイルからメモマーカーを削除します。
	FAPPEND 関数 (p. 393)	現在のレコードを外部ファイルの最後に追加します。

カテゴリ	言語要素	説明
	FCLOSE 関数 (p. 394)	外部ファイル、ディレクトリまたはディレクトリメンバを閉じます。
	FCOL 関数 (p. 395)	ファイルデータバッファ(FDB)内の現在の列の位置を返します。
	FDELETE 関数 (p. 396)	外部ファイルまたは空のディレクトリを削除します。
	FEXIST 関数 (p. 400)	ファイル参照名に関連付けられている外部ファイルが存在するかどうかを確認します。
	FGET 関数 (p. 401)	ファイルデータバッファ(FDB)からデータを変数にコピーします。
	FILEEXIST 関数 (p. 403)	物理名で外部ファイルが存在するかどうかを確認します。
	FILENAME 関数 (p. 404)	ファイル参照名を外部ファイル、ディレクトリまたは出力デバイスに割り当てるか、または割り当てを取り消します。
	FILEREF 関数 (p. 407)	現在の SAS セッションにファイル参照名が割り当てられているかどうかを確認します。
	FINFO 関数 (p. 463)	ファイル情報項目の値を返します。
	FNOTE 関数 (p. 474)	最後に読み込まれたレコードを特定し、FPOINT 関数で使用できる値を返します。
	FOPEN 関数 (p. 476)	外部ファイルを開いて、ファイル識別子の値を返します。
	FOPTNAME 関数 (p. 478)	ファイルに関する情報項目の名前を返します。
	FOPTNUM 関数 (p. 480)	外部ファイルで使用できる情報項目の数を返します。
	FPOINT 関数 (p. 481)	次に読み込むレコードに読み込みポインタを配置します。
	FPOS 関数 (p. 482)	ファイルデータバッファ(FDB)の列ポインタの位置を設定します。
	FPUT 関数 (p. 484)	外部ファイルのファイルデータバッファ(FDB)にデータを移動します。開始位置は FDB の現在の列位置になります。
	FREAD 関数 (p. 486)	外部ファイルのレコードをファイルデータバッファ(FDB)に読み込みます。
	FREWIND 関数 (p. 487)	ファイルの先頭にファイルポインタを配置します。
	FRLEN 関数 (p. 488)	最後に読み込まれたレコードのサイズを返します。出力のためにファイルが開かれている場合は、現在のレコードサイズを返します。
	FSEP 関数 (p. 489)	FGET 関数のトークン区切り文字を設定します。
	FWRITE 関数 (p. 491)	外部ファイルにレコードを書き込みます。

カテゴリ	言語要素	説明
	MOPEN 関数 (p. 653)	ディレクトリ ID とメンバ名でファイルを開き、ファイル ID または 0 のいずれかを返します。
	PATHNAME 関数 (p. 700)	外部ファイルや SAS ライブラリの物理名または空白を返します。
	RENAME 関数 (p. 803)	SAS ライブラリのメンバ、SAS カタログ内のエントリ、外部ファイル、ディレクトリのいずれかの名前を変更します。
	SYSMSG 関数 (p. 880)	最後のデータセットまたは外部ファイル関数の処理のエラーまたは警告メッセージを返します。
	SYSRC 関数 (p. 884)	システムエラー番号を返します。
外部ルーチン	CALL MODULE ルーチン (p. 189)	外部ルーチンをリターンコードなしで呼び出します。
	MODULEC 関数 (p. 649)	外部ルーチンを呼び出し、文字値を返します。
	MODULEN 関数 (p. 650)	外部ルーチンを呼び出し、数値を返します。
確率	CDF 関数 (p. 273)	累積確率分布の値を返します。
	LOGCDF 関数 (p. 624)	左累積分布関数の対数を返します。
	LOGPDF 関数 (p. 626)	確率密度(質量)関数の対数を返します。
	LOGSDF 関数 (p. 628)	生存関数の対数を返します。
	PDF 関数 (p. 703)	確率密度(質量)分布の値を返します。
	POISSON 関数 (p. 726)	Poisson 分布の確率を返します。
	PROBBETA 関数 (p. 728)	ベータ分布の確率を返します。
	PROBBNML 関数 (p. 729)	二項分布の確率を返します。
	PROBBNRM 関数 (p. 730)	2 変量正規分布の確率を返します。
	PROBCHI 関数 (p. 731)	$\chi 2$ 乗分布の確率を返します。
	PROBF 関数 (p. 732)	F 分布の確率を返します。
	PROBGAM 関数 (p. 733)	ガンマ分布の確率を返します。
	PROBHYPR 関数 (p. 734)	超幾何分布の確率を返します。
	PROBMC 関数 (p. 737)	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。
	PROBNEGB 関数 (p. 749)	負数二項分布の確率を返します。
	PROBNORM 関数 (p. 750)	標準正規分布の確率を返します。

カテゴリ	言語要素	説明
	PROBT 関数 (p. 751)	t 分布の確率を返します。
	SDF 関数 (p. 832)	生存関数を返します。
記述統計	CMISS 関数 (p. 297)	欠損引数の数を数えます。
	CSS 関数 (p. 340)	修正済み平方和を返します。
	CV 関数 (p. 343)	変動係数を返します。
	EUCLID 関数 (p. 387)	非欠損値引数のユークリッドノルムを返します。
	GEOMEAN 関数 (p. 501)	幾何平均を返します。
	GEOMEANZ 関数 (p. 502)	ゼロファジーを使用して、幾何平均を返します。
	HARMEAN 関数 (p. 515)	調和平均を返します。
	HARMEANZ 関数 (p. 516)	ゼロファジーを使用して、調和平均を返します。
	IQR 関数 (p. 585)	四分位範囲を返します。
	KURTOSIS 関数 (p. 589)	尖度を返します。
	LARGEST 関数 (p. 597)	k 番目に大きい非欠損値を返します。
	LPNORM 関数 (p. 632)	第 2 引数および後続の非欠損引数の Lp ノルムを返します。
	MAD 関数 (p. 633)	中央値からの平均絶対偏差を返します。
	MAX 関数 (p. 639)	最大値を返します。
	MEAN 関数 (p. 642)	算術平均(平均)を返します。
	MEDIAN 関数 (p. 643)	中央値を返します。
	MIN 関数 (p. 643)	最小値を返します。
	MISSING 関数 (p. 645)	引数が欠損値を含むかどうかの結果を表す数値を返します。
	N 関数 (p. 662)	数値の非欠損値の数を返します。
	NMISS 関数 (p. 666)	欠損数値の数を返します。
	ORDINAL 関数 (p. 699)	k 番目に小さい欠損値および非欠損値を返します。
	PCTL 関数 (p. 702)	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. 796)	非欠損値の範囲を返します。
	RMS 関数 (p. 809)	非欠損引数の 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. 838)	非欠損引数の歪度を返します。
	SMALLEST 関数 (p. 840)	k 番目に小さい非欠損値を返します。
	STD 関数 (p. 858)	非欠損引数の標準偏差を返します。
	STDERR 関数 (p. 859)	非欠損引数の平均の標準誤差を返します。
	SUM 関数 (p. 873)	非欠損引数の合計を返します。
	SUMABS 関数 (p. 874)	非欠損引数の絶対値の合計を返します。
	USS 関数 (p. 906)	非欠損引数の無修正平方和を返します。
	VAR 関数 (p. 908)	非欠損引数の分散を返します。
距離	GEODIST 関数 (p. 499)	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. 962)	2 つの郵便番号が示す場所間の測地距離を返します。
切り捨て関数	CEIL 関数 (p. 288)	予期しない浮動小数点の結果が生じないようにファジー処理して、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。
	CEILZ 関数 (p. 289)	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. 470)	予期しない浮動小数点の結果が生じないようにファジー処理して、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。
	FLOORZ 関数 (p. 471)	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. 490)	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. 543)	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値を返します。
	INTZ 関数 (p. 582)	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. 810)	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. 816)	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. 819)	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. 901)	数値を指定したバイト数で切り捨てます。

カテゴリ	言語要素	説明
組み合わせ関数	ALLCOMB 関数 (p. 94)	n 個の変数値から一度に k 個の変数値を取得する場合のすべての組み合わせを変化量の小さい順に生成します。
	ALLPERM 関数 (p. 96)	複数の変数の値のすべての順列を変化量の小さい順に生成します。
	CALL ALLCOMB ルーチン (p. 148)	n 個の変数値から一度に k 個の変数値を取得する場合のすべての組み合わせを変化量の小さい順に生成します。
	CALL ALLCOMBI ルーチン (p. 151)	n 個のオブジェクトを同時に k 個使用するときのインデックスのすべての組み合わせを、変化量の小さい順に生成します。
	CALL ALLPERM ルーチン (p. 153)	複数の変数の値のすべての順列を変化量の小さい順に生成します。
	CALL GRAYCODE ルーチン (p. 165)	n 個の項目のすべてのサブセットを変化量の小さい順に生成します。
	CALL LEXCOMB ルーチン (p. 174)	n 個の変数を同時に k 個使用するときの、重複しない非欠損値のすべての組み合わせを辞書式順序で生成します。
	CALL LEXCOMBI ルーチン (p. 177)	n 個のオブジェクトを同時に k 個使用するときのインデックスのすべての組み合わせを、辞書式順序で生成します。
	CALL LEXPERK ルーチン (p. 179)	n 個の変数から一度に k 個を取り出した非欠損値の重複しないすべての順列を辞書式順序で生成します。
	CALL LEXPERM ルーチン (p. 183)	複数の変数の非欠損値の重複しないすべての順列を辞書式順序で生成します。
	CALL RANCOMB ルーチン (p. 211)	引数の値を置換し、n 個の値のうち k 個のランダムな組み合わせを返します。
	CALL RANPERK ルーチン (p. 220)	引数の値を置換し、n 個の値のうち k 個のランダムな順列を返します。
	CALL RANPERM ルーチン (p. 222)	引数の値をランダムに置換します。
	COMB 関数 (p. 303)	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。
	GRAYCODE 関数 (p. 512)	n 個の項目のすべてのサブセットを変化量の小さい順に生成します。
	LCOMB 関数 (p. 600)	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。
	LEXCOMB 関数 (p. 607)	n 個の変数から一度に k 個を取り出した非欠損値の、重複しないすべての組み合わせを辞書式順序で生成します。
	LEXCOMBI 関数 (p. 610)	n 個のオブジェクトを同時に k 個使用するときのインデックスのすべての組み合わせを、辞書式順序で生成します。

カテゴリ	言語要素	説明
	LEXPERK 関数 (p. 611)	n 個の変数から一度に k 個を取り出した非欠損値の、重複しないすべての順列を辞書式順序で生成します。
	LEXPERM 関数 (p. 614)	複数の変数の非欠損値の重複しないすべての順列を辞書式順序で生成します。
	LFACT 関数 (p. 616)	FACT(階乗)関数の対数を計算します。
	LPERM 関数 (p. 631)	PERM 関数の対数を計算します。これは、要素数 r を含むオプションを使用した n 個のオブジェクトの順列数の対数です。
	PERM 関数 (p. 723)	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。
検索	WHICHC 関数 (p. 953)	第 1 引数に等しい文字値を検索し、最初に一致した値のインデックスを返します。
	WHICHN 関数 (p. 954)	第 1 引数に等しい数値を検索し、最初に一致した値のインデックスを返します。
財務関数	BLACKCLPRC 関数 (p. 136)	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. 138)	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. 140)	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. 142)	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. 319)	複利パラメータを返します。
	CONVX 関数 (p. 328)	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. 328)	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. 341)	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. 342)	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。
	DACCDB 関数 (p. 344)	定率法による減価償却累積額を返します。
	DACCDBSL 関数 (p. 345)	定率法による減価償却累積額と、定額法による原価償却への変換を返します。
	DACCSL 関数 (p. 346)	定額法による減価償却累積額を返します。

カテゴリ	言語要素	説明
	DACCSYD 関数 (p. 347)	年次級数和法による減価償却累積額を返します。
	DACCTAB 関数 (p. 347)	指定テーブルから減価償却累積額を返します。
	DEPDB 関数 (p. 357)	定率法による減価償却を返します。
	DEPDBSL 関数 (p. 357)	定額法による減価償却に変換する定率法を返します。
	DEPSL 関数 (p. 358)	定額法による減価償却を返します。
	DEPSYD 関数 (p. 359)	年次級数和法による減価償却を返します。
	DEPTAB 関数 (p. 360)	指定テーブルから減価償却を返します。
	DUR 関数 (p. 382)	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. 383)	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. 384)	有効な年利を返します。
	FINANCE 関数 (p. 408)	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。
	GARKHCLPRC 関数 (p. 494)	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. 496)	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. 574)	内部利益率を分数で返します。
	IPMT 関数 (p. 584)	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. 586)	内部利益率をパーセントで返します。
	MARGRCLPRC 関数 (p. 634)	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. 636)	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. 656)	割賦返済パラメータを返します。
	NETPV 関数 (p. 663)	現在正味価値をパーセントで返します。
	NOMRATE 関数 (p. 666)	名目年利を返します。
	NPV 関数 (p. 692)	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. 724)	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	言語要素	説明
	PPMT 関数 (p. 727)	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. 776)	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. 821)	定期預金の将来価値を返します。
	SAVINGS 関数 (p. 822)	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. 888)	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. 956)	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. 121)	逆余弦を返します。
	ARSIN 関数 (p. 122)	逆正弦を返します。
	ATAN 関数 (p. 125)	逆正接を返します。
	ATAN2 関数 (p. 126)	2 つの数値変数の比率の逆正接を返します。
	COS 関数 (p. 330)	余弦(コサイン)を返します。
	SIN 関数 (p. 837)	正弦を返します。
	TAN 関数 (p. 885)	正接を返します。
算術	DIVIDE 関数 (p. 373)	ODS 出力の特殊欠損値を処理する除算の結果を返します。
州コード/郵便番号	FIPNAME 関数 (p. 466)	2 桁の FIPS コードを大文字の州名に変換します。
	FIPNAMEL 関数 (p. 467)	2 桁の FIPS コードを大文字小文字混在の州名に変換します。
	FIPSTATE 関数 (p. 468)	2 桁の FIPS コードを 2 文字の州コードに変換します。
	STFIPS 関数 (p. 860)	州の郵便コードを FIPS コードに変換します。
	STNAME 関数 (p. 861)	州の郵便コードを大文字の州名に変換します。
	STNAMEL 関数 (p. 862)	州の郵便コードを大文字と小文字を使った州名に変換します。
	ZIPCITY 関数 (p. 960)	都市名と、郵便番号に対応する 2 文字の郵便コードを返します。
	ZIPCITYDISTANCE 関数 (p. 962)	2 つの郵便番号が示す場所間の測地距離を返します。
	ZIPFIPS 関数 (p. 963)	郵便番号を 2 桁の FIPS コードに変換します。

カテゴリ	言語要素	説明
	ZIPNAME 関数 (p. 965)	郵便番号を大文字の州名に変換します。
	ZIPNAMEL 関数 (p. 966)	郵便番号を大文字小文字混在の州名に変換します。
	ZIPSTATE 関数 (p. 968)	郵便番号を 2 桁の州の郵便コードに変換します。
数学関数	ABS 関数 (p. 91)	絶対値を返します。
	AIRY 関数 (p. 93)	Airy 関数の値を返します。
	BETA 関数 (p. 134)	beta 関数の値を返します。
	CALL LOGISTIC ルーチン (p. 187)	ロジスティック関数を各引数に適用します。
	CALL SOFTMAX ルーチン (p. 244)	softmax 値を返します。
	CALL STDIZE ルーチン (p. 247)	1 つ以上の変数の値を標準化します。
	CALL TANH ルーチン (p. 255)	双曲線正接を返します。
	CNONCT 関数 (p. 298)	χ2 乗分布の非心度パラメータを返します。
	COALESCE 関数 (p. 300)	数値の引数のリストからの最初の非欠損値を返します。
	COMPFUZZ 関数 (p. 309)	2 つの数値をファジー比較します。
	CONSTANT 関数 (p. 324)	マシン定数および数学定数を計算します。
	DAIRY 関数 (p. 348)	AIRY 関数の導関数を返します。
	DEVIANCE 関数 (p. 363)	確率分布に基づくデビアンスを返します。
	DIGAMMA 関数 (p. 369)	ディガンマ関数の値を返します。
	ERF 関数 (p. 386)	(正規)誤差関数の値を返します。
	ERFC 関数 (p. 387)	相補(正規)誤差関数の値を返します。
	EXP 関数 (p. 391)	指数関数の値を返します。
	FACT 関数 (p. 392)	階乗を計算します。
	FNONCT 関数 (p. 472)	F 分布の非心度パラメータの値を返します。
	GAMMA 関数 (p. 493)	ガンマ関数の値を返します。
	GCD 関数 (p. 498)	1 つ以上の整数の最大公約数を返します。

カテゴリ	言語要素	説明
	IBESSEL 関数 (p. 526)	変形ベッセル関数の値を返します。
	JBESSEL 関数 (p. 587)	ベッセル関数の値を返します。
	LCM 関数 (p. 599)	最小公倍数を返します。
	LGAMMA 関数 (p. 617)	ガンマ関数の自然対数を返します。
	LOG 関数 (p. 621)	自然(底 e)対数を返します。
	LOG1PX 関数 (p. 621)	1 に引数を加えた値の対数を返します。
	LOG10 関数 (p. 623)	底 10 の対数を返します。
	LOG2 関数 (p. 623)	底 2 の対数を返します。
	LOGBETA 関数 (p. 624)	ベータ関数の対数を返します。
	MOD 関数 (p. 646)	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. 650)	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。
	MSPLINT 関数 (p. 657)	単調性維持スプライン補間の縦座標を返します。
	SIGN 関数 (p. 836)	値の符合を返します。
	SQRT 関数 (p. 856)	値の平方根を返します。
	TNONCT 関数 (p. 890)	スチューデントの t 分布から非心度パラメータの値を返します。
	TRIGAMMA 関数 (p. 898)	トリガンマ関数の値を返します。
数値	IFN 関数 (p. 529)	式の真、偽、欠損に基づいて数値を返します。
	MODEXIST 関数 (p. 648)	インストールされている SAS のバージョンにソフトウェアイメージが存在するかどうかを判断します。
双曲線関数	ARCOSH 関数 (p. 121)	逆双曲線余弦を返します。
	ARSINH 関数 (p. 123)	逆双曲線正弦を返します。
	ARTANH 関数 (p. 124)	逆双曲線正接を返します。
	COSH 関数 (p. 330)	双曲線余弦を返します。
	SINH 関数 (p. 837)	双曲線正弦を返します。
	TANH 関数 (p. 886)	双曲線正接を返します。

カテゴリ	言語要素	説明
特殊関数	ADDR 関数 (p. 91)	32 ビットプラットフォームの変数のメモリアドレスを返します。
	ADDRLONG 関数 (p. 92)	32 ビットおよび 64 ビットプラットフォームの変数のメモリアドレスを返します。
	CALL POKE ルーチン (p. 192)	32 ビットプラットフォームのメモリに値を直接書き込みます。
	CALL POKELONG ルーチン (p. 193)	32 ビットおよび 64 ビットのプラットフォームのメモリに値を直接書き込みます。
	CALL SLEEP ルーチン (p. 243)	指定した期間、この CALL ルーチンを呼び出すプログラムの実行を中断します。
	CALL SYSTEM ルーチン (p. 254)	実行の動作環境コマンドをサブミットします。
	DIF 関数 (p. 368)	引数とその n 番目のラグの差分を返します。
	INPUT 関数 (p. 538)	指定した入力形式を使用して SAS が式を変換するときに生成された値を返します。
	INPUTC 関数 (p. 540)	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. 541)	実行時に数値の入力形式を指定できるようにします。
	LAG 関数 (p. 590)	キューから値を返します。
	PEEK 関数 (p. 718)	32 ビットプラットフォームのメモリアドレスの内容を数値変数に保存します。
	PEEKC 関数 (p. 719)	32 ビットプラットフォームのメモリアドレスの内容を文字変数に保存します。
	PEEKCLONG 関数 (p. 720)	32 ビットおよび 64 ビットプラットフォームのメモリアドレスの内容を文字変数に保存します。
	PEEKLONG 関数 (p. 722)	32 ビットおよび 64 ビットプラットフォームのメモリアドレスの内容を数値変数に保存します。
	PTRLONGADD 関数 (p. 770)	ポインタアドレスを 32 ビットおよび 64 ビットのプラットフォームの文字変数として返します。
	PUT 関数 (p. 770)	指定した出力形式を使用して値を返します。
	PUTC 関数 (p. 772)	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. 774)	実行時に数値の出力形式を指定できるようにします。
	SLEEP 関数 (p. 839)	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。

カテゴリ	言語要素	説明
	SYSEXIST 関数 (p. 877)	環境内に動作環境変数が存在するかどうかを示す値を返します。
	SYSGET 関数 (p. 878)	指定した動作環境変数の値を返します。
	SYSPARM 関数 (p. 880)	システムパラメータ文字列を返します。
	SYSPROCESSID 関数 (p. 881)	現在のプロセスのプロセス ID を返します。
	SYSPROCESSNAME 関数 (p. 882)	指定プロセス ID に関連付けられているプロセス名、または現在のプロセスの名前を返します。
	SYSPROD 関数 (p. 883)	製品のライセンスがあるかどうかを判断します。
	SYSTEM 関数 (p. 884)	SAS セッションで動作環境コマンドを発行し、システムリターンコードを返します。
	UUIDGEN 関数 (p. 907)	UUID (Universal Unique Identifier)の短い形式またはバイナリ形式を返します。
並べ替え	CALL SORTC ルーチン (p. 245)	文字引数の値を並べ替えます。
	CALL SORTN ルーチン (p. 246)	数値引数の値を並べ替えます。
配列	DIM 関数 (p. 370)	配列にある要素数を返します。
	HBOUND 関数 (p. 517)	配列の上限を返します。
	LBOUND 関数 (p. 598)	配列の下限を返します。
日付と時間	CALL IS8601_CONVERT ルーチン (p. 169)	ISO 8601 規格の間隔を日時値とデュレーション値に変換します。または、日時値とデュレーション値を ISO 8601 規格の間隔に変換します。
	DATDIF 関数 (p. 349)	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. 351)	SAS 日付値として現在の日付を返します。
	DATEJUL 関数 (p. 352)	ユリウス暦の日付を SAS 日付値に変換します。
	DATEPART 関数 (p. 352)	SAS 日時値から日付を抽出します。
	DATETIME 関数 (p. 353)	SAS 日時値として現在の日時を返します。
	DAY 関数 (p. 354)	SAS 日付値として月の日を返します。
	DHMS 関数 (p. 366)	日、時、分、秒の値から SAS 日時値を返します。
	HMS 関数 (p. 519)	SAS 時間値(時、分および秒)を返します。

カテゴリ	言語要素	説明
	HOLIDAY 関数 (p. 519)	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. 522)	SAS 時間値または SAS 日時値の時間を返します。
	INTCINDEX 関数 (p. 544)	周期インデックスを返します。この関数には、日付、時間または日時の間隔と値を指定します。
	INTCK 関数 (p. 546)	2 つの日付、時間または日時の値の間にある指定した種類の間隔の境界数を返します。
	INTCYCLE 関数 (p. 552)	次に高い季節周期での期間(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。
	INTFIT 関数 (p. 555)	2 つの日付に基づく時間間隔を返します。
	INTFMT 関数 (p. 558)	推奨 SAS 出力形式を返します。この関数には、日付、時間または日時の間隔を指定します。
	INTGET 関数 (p. 560)	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. 561)	季節インデックスを返します。この関数には、日付、時間または日時の間隔と値を指定します。
	INTNX 関数 (p. 567)	指定した時間間隔で日付、時間または日時の値を増分し、日付、時間または日時の値を返します。
	INTSEAS 関数 (p. 575)	季節周期の長さを返します。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. 578)	ベース間隔に対応するシフト間隔を返します。
	INTTEST 関数 (p. 580)	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。
	JULDATE 関数 (p. 587)	SAS 日付値からユリウス暦の日付を返します。
	JULDATE7 関数 (p. 588)	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. 641)	月、日および年の値から SAS 日付値を返します。
	MINUTE 関数 (p. 644)	SAS 時間または日付値から分を返します。
	MONTH 関数 (p. 652)	SAS 日付値から月を返します。
	NWKDOM 関数 (p. 695)	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. 777)	SAS 日付値から年の四半期を返します。
	SECOND 関数 (p. 835)	SAS 時間値または SAS 日時値の秒数を返します。
	TIME 関数 (p. 887)	数値の SAS 時間値として現在時刻を返します。

カテゴリ	言語要素	説明
	TIMEPART 関数 (p. 887)	SAS 日時値から時間値を抽出します。
	TODAY 関数 (p. 891)	数値の SAS 日付値として現在の日付を返します。
	WEEK 関数 (p. 948)	週番号の値を返します。
	WEEKDAY 関数 (p. 952)	SAS 日付値から、曜日に対応する整数を返します。
	YEAR 関数 (p. 955)	SAS 日付値から年を返します。
	YRDIF 関数 (p. 957)	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。
	YYQ 関数 (p. 959)	年と四半期の値から SAS 日付値を返します。
分位点	BETAINV 関数 (p. 135)	ベータ分布から分位点を返します。
	CINV 関数 (p. 295)	χ2 乗分布から分位点を返します。
	FINV 関数 (p. 465)	F 分布から分位点を返します。
	GAMINV 関数 (p. 492)	ガンマ分布から分位点を返します。
	PROBIT 関数 (p. 736)	標準正規分布の分位点を返します。
	QUANTILE 関数 (p. 778)	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. 856)	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. 889)	t 分布から分位点を返します。
変数情報	CALL VNEXT ルーチン (p. 257)	DATA ステップで使用される変数の名前、種類および長さを返します。
	VARRAY 関数 (p. 915)	指定した名前が配列かどうかを示す値を返します。
	VARRAYX 関数 (p. 916)	指定した引数の値が配列かどうかを示す値を返します。
	VFORMAT 関数 (p. 919)	指定した変数に関連付けられた出力形式を返します。
	VFORMATD 関数 (p. 920)	指定した変数に関連付けられた出力形式の 10 進値を返します。
	VFORMATDX 関数 (p. 921)	指定した引数の値に関連付けられた出力形式の 10 進値を返します。
	VFORMATN 関数 (p. 922)	指定した変数に関連付けられた出力形式名を返します。
	VFORMATNX 関数 (p. 923)	指定した引数の値に関連付けられた出力形式名を返します。
	VFORMATW 関数 (p. 924)	指定した変数に関連付けられた出力形式の幅を返します。

カテゴリ	言語要素	説明
	VFORMATWX 関数 (p. 925)	指定した引数の値に関連付けられた出力形式の幅を返します。
	VFORMATX 関数 (p. 926)	指定した引数の値に関連付けられた出力形式を返します。
	VINARRAY 関数 (p. 927)	指定した変数が配列のメンバかどうかを示す値を返します。
	VINARRAYX 関数 (p. 928)	指定した引数の値が配列のメンバかどうかを示す値を返します。
	VINFORMAT 関数 (p. 929)	指定した変数に関連付けられた入力形式を返します。
	VINFORMATD 関数 (p. 930)	指定した変数に関連付けられた入力形式の 10 進値を返します。
	VINFORMATDX 関数 (p. 931)	指定した変数の値に関連付けられた入力形式の 10 進値を返します。
	VINFORMATN 関数 (p. 932)	指定した変数に関連付けられた入力形式名を返します。
	VINFORMATNX 関数 (p. 933)	指定した引数の値に関連付けられた入力形式名を返します。
	VINFORMATW 関数 (p. 934)	指定した変数に関連付けられた入力形式の幅を返します。
	VINFORMATWX 関数 (p. 935)	指定した引数の値に関連付けられた入力形式の幅を返します。
	VINFORMATX 関数 (p. 936)	指定した引数の値に関連付けられた入力形式を返します。
	VLABEL 関数 (p. 937)	指定した変数に関連付けられているラベルを返します。
	VLABELX 関数 (p. 938)	指定した引数の値に関連付けられたラベルを返します。
	VLENGTH 関数 (p. 939)	指定した変数のコンパイル時(配分された)サイズを返します。
	VLENGTHX 関数 (p. 940)	引数の値と同じ名前の変数のコンパイル時(配分された)サイズを返します。
	VNAME 関数 (p. 941)	指定した変数の名前を返します。
	VNAMEX 関数 (p. 942)	変数名として指定した引数の値を検証します。
	VTYPE 関数 (p. 943)	指定した変数の種類(文字または数値)を返します。
	VTYPEX 関数 (p. 944)	指定した引数の値の種類(文字または数値)を返します。
	VVALUE 関数 (p. 945)	指定する変数に関連付けられている出力形式を適用した値を返します。
	VVALUEX 関数 (p. 947)	指定する引数に関連付けられている出力形式を適用した値を返します。

カテゴリ	言語要素	説明
変数制御	CALL LABEL ルーチン (p. 172)	指定した文字変数に変数ラベルを割り当てます。
	CALL SET ルーチン (p. 242)	SAS データセット変数を名前とデータ型が同じ DATA ステップまたはマクロ変数にリンクします。
	CALL VNAME ルーチン (p. 256)	指定した変数の値として変数名を割り当てます。
マクロ	CALL EXECUTE ルーチン (p. 165)	引数を解決し、実行用に解決された値を次のステップの境界で発行します。
	CALL SYMPUT ルーチン (p. 252)	マクロ変数に DATA ステップ情報を割り当てます。
	CALL SYMPUTX ルーチン (p. 252)	マクロ変数に値を割り当て、先頭と末尾の空白の両方を削除します。
	RESOLVE 関数 (p. 806)	マクロ機能によって処理された後の引数の解決値を返します。
	SYMEXIST 関数 (p. 875)	マクロ変数が存在するかどうかを返します。
	SYMGET 関数 (p. 876)	DATA ステップの実行時にマクロ変数の値を返します。
	SYMGLOBL 関数 (p. 876)	DATA ステップの実行時にマクロ変数が DATA ステップのグローバルスコープにあるかどうかを返します。
	SYMLOCAL 関数 (p. 877)	DATA ステップの実行時にマクロ変数が DATA ステップのローカルスコープにあるかどうかを返します。
文字関数	ANYALNUM 関数 (p. 98)	文字列から英数字を検索し、最初に検索された文字の位置を返します。
	ANYALPHA 関数 (p. 100)	文字列から英字を検索し、最初に検索された文字の位置を返します。
	ANYCNTRL 関数 (p. 102)	文字列からコントロール文字を検索し、最初に検索された文字の位置を返します。
	ANYDIGIT 関数 (p. 103)	文字列から数字を検索し、最初に検索された数字の位置を返します。
	ANYFIRST 関数 (p. 105)	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位置を返します。
	ANYGRAPH 関数 (p. 106)	文字列からグラフィカル文字を検索し、最初に検索された文字の位置を返します。
	ANYLOWER 関数 (p. 109)	文字列から小文字を検索し、最初に検索された文字の位置を返します。

カテゴリ	言語要素	説明
	ANYNAME 関数 (p. 110)	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位置を返します。
	ANYPRINT 関数 (p. 112)	文字列から印刷可能な文字を検索し、最初に検索された文字の位置を返します。
	ANYPUNCT 関数 (p. 114)	文字列から句読文字を検索し、最初に検索された文字の位置を返します。
	ANYSPACE 関数 (p. 116)	文字列から空白文字(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)を検索し、最初に検索された文字の位置を返します。
	ANYUPPER 関数 (p. 118)	文字列から大文字を検索し、最初に検索された文字の位置を返します。
	ANYXDIGIT 関数 (p. 119)	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位置を返します。
	BYTE 関数 (p. 147)	ASCII 照合順序または EBCDIC 照合順序の 1 文字を返します。
	CALL CATS ルーチン (p. 157)	先頭と末尾の空白を削除して、連結文字列を返します。
	CALL CATT ルーチン (p. 158)	末尾の空白を削除して、連結文字列を返します。
	CALL CATX ルーチン (p. 160)	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。
	CALL COMPCOST ルーチン (p. 162)	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。
	CALL MISSING ルーチン (p. 188)	欠損値を指定した文字変数または数値変数に割り当てます。
	CALL SCAN ルーチン (p. 233)	文字列から n 番目の単語の位置と長さを返します。
	CAT 関数 (p. 259)	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。
	CATQ 関数 (p. 261)	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. 265)	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. 267)	末尾の空白を削除して、連結文字列を返します。
	CATX 関数 (p. 270)	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。
	CHAR 関数 (p. 292)	文字列の指定した位置の 1 文字を返します。

カテゴリ	言語要素	説明
	CHOOSEC 関数 (p. 293)	引数のリストからの選択結果を表す文字値を返します。
	CHOOSEN 関数 (p. 294)	引数のリストからの選択結果を表す数値を返します。
	COALESCEC 関数 (p. 301)	文字引数のリストからの最初の非欠損値を返します。
	COLLATE 関数 (p. 302)	ASCII 照合順序または EBCDIC 照合順序の文字列を返します。
	COMPARE 関数 (p. 304)	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. 308)	文字列内のワード間にある複数の空白を取り除きます。
	COMPGED 関数 (p. 311)	2 つの文字列間の一般化編集距離を返します。
	COMPLEV 関数 (p. 317)	2 つの文字列間の Levenshtein の編集距離を返します。
	COMPRESS 関数 (p. 321)	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. 331)	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. 333)	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. 337)	文字列中のワード数を数えます。
	DEQUOTE 関数 (p. 361)	引用符で始まる文字列から一致する引用符を削除し、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. 448)	文字列内の特定の部分文字列を検索します。
	FINDC 関数 (p. 450)	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. 457)	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。
	FIRST 関数 (p. 469)	文字列の最初の文字を返します。
	IFC 関数 (p. 527)	式の真、偽、欠損に基づいて文字値を返します。
	INDEX 関数 (p. 531)	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. 533)	文字式から指定した文字を検索し、その文字の位置を返します。
	INDEXW 関数 (p. 534)	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. 601)	文字列を左詰めにします。

カテゴリ	言語要素	説明
	LENGTH 関数 (p. 602)	末尾の空白を除いた文字列の長さを返します。文字列が空白の場合には、1 を返します。
	LENGTHC 関数 (p. 603)	末尾の空白を含めた文字列の長さを返します。
	LENGTHM 関数 (p. 604)	文字列に割り当てられたメモリの量を返します(バイト単位)。
	LENGTHN 関数 (p. 606)	末尾の空白を除いた文字列の長さを返します。
	LOWCASE 関数 (p. 630)	引数のすべての文字を小文字に変換します。
	MD5 関数 (p. 639)	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。
	MISSING 関数 (p. 645)	引数が欠損値を含むかどうかの結果を表す数値を返します。
	MVALID 関数 (p. 659)	SAS メンバ名として使用する文字列の有効性を確認します。
	NLITERAL 関数 (p. 664)	指定した文字列を SAS 名前リテラルに変換します。
	NOTALNUM 関数 (p. 668)	文字列から英数字でない文字を検索し、最初に検索された文字の位置を返します。
	NOTALPHA 関数 (p. 669)	文字列から英字でない文字を検索し、最初に検索された文字の位置を返します。
	NOTCNTRL 関数 (p. 671)	文字列から制御文字でない文字を検索し、最初に検索された文字の位置を返します。
	NOTDIGIT 関数 (p. 673)	文字列から数字でない文字を検索し、最初に検索された文字の位置を返します。
	NOTFIRST 関数 (p. 676)	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、最初に検索された文字の位置を返します。
	NOTGRAPH 関数 (p. 678)	文字列からグラフィカル文字でない文字を検索し、最初に検索された文字の位置を返します。
	NOTLOWER 関数 (p. 680)	文字列から小文字でない文字を検索し、最初に検索された文字の位置を返します。
	NOTNAME 関数 (p. 681)	VALIDVARNAME=V7 の SAS 変数名として無効な文字を文字列から検索し、最初に検索された文字の位置を返します。
	NOTPRINT 関数 (p. 683)	文字列から印刷不可文字を検索し、最初に検索された文字の位置を返します。
	NOTPUNCT 関数 (p. 684)	文字列から句読文字でない文字を検索し、最初に検索された文字の位置を返します。

カテゴリ	言語要素	説明
	NOTSPACE 関数 (p. 686)	文字列から空白文字(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)でない文字を検索し、最初に検索された文字の位置を返します。
	NOTUPPER 関数 (p. 688)	文字列から大文字でない文字を検索し、最初に検索された文字の位置を返します。
	NOTXDIGIT 関数 (p. 690)	文字列から 16 進数字でない文字を検索し、最初に検索された文字の位置を返します。
	NVALID 関数 (p. 692)	SAS 変数名として使用する文字列の有効性を確認します。
	PROPCASE 関数 (p. 752)	引数のすべての単語を適切に大文字と小文字に変換します。
	QUOTE 関数 (p. 780)	文字値に二重引用符を付加します。
	RANK 関数 (p. 797)	ASCII 照合順序または EBCDIC 照合順序による文字の位置を返します。
	REPEAT 関数 (p. 805)	最初の引数を n+1 回繰り返した文字値を返します。
	REVERSE 関数 (p. 806)	文字列を逆にします。
	RIGHT 関数 (p. 808)	文字式を右揃えにします。
	SCAN 関数 (p. 824)	文字列から n 番目の単語を返します。
	SOUNDEX 関数 (p. 852)	文字列をエンコードして検索しやすくします。
	SPEDIS 関数 (p. 853)	2 つの単語の一致尤度を調べて、2 単語間のスペルの違いをコストで表します。
	STRIP 関数 (p. 863)	先頭と末尾の空白を削除して文字列を返します。
	SUBPAD 関数 (p. 865)	必要に応じて空白埋め込みを使用し、指定した長さの部分文字列を返します。
	擬似 SUBSTR 関数(割り当ての左辺に用いた場合) (p. 866)	文字値の内容を置換します。
	SUBSTR 関数 (p. 867)	引数から部分文字列を抽出します。
	SUBSTRN 関数 (p. 869)	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. 892)	文字列に含まれる特定の文字を置き換えます。
	TRANSTRN 関数 (p. 893)	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削除します。
	TRANWRD 関数 (p. 895)	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. 899)	文字列から末尾の空白を取り除きます。文字列が欠損値の場合は、1 つの空白を返します。

カテゴリ	言語要素	説明
	TRIMN 関数 (p. 900)	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. 902)	引数内のすべての文字を大文字に変換します。
	VERIFY 関数 (p. 918)	他の文字列に存在しない文字の最初の出現位置を返します。
文字列マッチング	CALL PRXCHANGE ルーチン (p. 194)	パターンマッチングの置換を実行します。
	CALL PRXDEBUG ルーチン (p. 196)	DATA ステップで Perl 正規表現を有効にし、デバッグ出力を SAS ログに送信します。
	CALL PRXFREE ルーチン (p. 198)	Perl 正規表現に割り当てられたメモリを解放します。
	CALL PRXNEXT ルーチン (p. 199)	パターンに一致し、1 つの文字列内で複数の一致が繰り返される部分文字列の位置と長さを返します。
	CALL PRXPOSN ルーチン (p. 201)	キャプチャバッファの開始位置と長さを返します。
	CALL PRXSUBSTR ルーチン (p. 204)	パターンに一致する部分文字列の位置と長さを返します。
	PRXCHANGE 関数 (p. 754)	パターンマッチングの置換を実行します。
	PRXMATCH 関数 (p. 759)	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. 763)	パターン内に一致が存在する場合の最後のかっこの一致を返します。
	PRXPARSE 関数 (p. 765)	文字値のパターン照合に使用できる Perl 正規表現(PRX)をコンパイルします。
	PRXPOSN 関数 (p. 767)	キャプチャバッファの値が含まれる文字列を返します。
乱数	CALL RANBIN ルーチン (p. 206)	二項分布からランダム変量を返します。
	CALL RANCAU ルーチン (p. 208)	Cauchy 分布からランダム変量を返します。
	CALL RANEXP ルーチン (p. 213)	指数分布からランダム変量を返します。
	CALL RANGAM ルーチン (p. 215)	ガンマ分布からランダム変量を返します。
	CALL RANNOR ルーチン (p. 218)	正規分布からランダム変量を返します。

カテゴリ	言語要素	説明
	CALL RANPOI ルーチン (p. 224)	ポアソン分布からランダム変量を返します。
	CALL RANTBL ルーチン (p. 226)	テーブル形式の確率分布からランダム変量を返します。
	CALL RANTRI ルーチン (p. 229)	三角分布からランダム変量を返します。
	CALL RANUNI ルーチン (p. 231)	一様分布からランダム変量を返します。
	CALL STREAMINIT ルーチン (p. 250)	後続の RAND 関数による乱数生成に使用するシード値を指定します。
	NORMAL 関数 (p. 668)	正規(ガウス)分布からランダム変量を返します。
	RANBIN 関数 (p. 782)	二項分布からランダム変量を返します。
	RANCAU 関数 (p. 783)	Cauchy 分布からランダム変量を返します。
	RAND 関数 (p. 784)	指定する分布から乱数を生成します。
	RANEXP 関数 (p. 794)	指数分布からランダム変量を返します。
	RANGAM 関数 (p. 795)	ガンマ分布からランダム変量を返します。
	RANNOR 関数 (p. 798)	正規分布からランダム変量を返します。
	RANPOI 関数 (p. 799)	ポアソン分布からランダム変量を返します。
	RANTBL 関数 (p. 800)	テーブル形式の確率分布からランダム変量を返します。
	RANTRI 関数 (p. 801)	三角分布からランダム変量を返します。
	RANUNI 関数 (p. 802)	一様分布からランダム変量を返します。
	UNIFORM 関数 (p. 902)	一様分布からランダム変量を返します。
論理ビット演算関数	BAND 関数 (p. 133)	2 つの引数のビットごとの論理積を返します。
	BLSHIFT 関数 (p. 143)	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. 144)	引数のビットごとの論理否定を返します。
	BOR 関数 (p. 145)	2 つの引数のビットごとの論理和を返します。
	BRSHIFT 関数 (p. 145)	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. 146)	2 つの引数のビットごとの排他的論理和を返します。

ディクショナリ

ABS 関数

絶対値を返します。

カテゴリ: 数学関数

構文

ABS (*argument*)

必須引数

引数
数値の定数、変数または式を指定します。

詳細

ABS 関数は、引数と同じ大きさの正の数を返します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=abs(2.4);	2.4
x=abs(-3);	3

ADDR 関数

32 ビットプラットフォームの変数のメモリアドレスを返します。

カテゴリ: 特殊関数

制限事項: 32 ビットプラットフォームでのみ使用します。

構文

ADDR(*variable*)

必須引数

変数
変数名を指定します。

詳細

戻り値は数値です。変数の保存場所は実行ごとに変わる可能性があるため、ADDR で返される値も異なる場合があります。通常、ADDR 関数は PEEK 関数と PEEKC 関数および CALL POKE ルーチンを組み合わせて使用されます。

ADDR 関数は、64 ビットプラットフォームでは使用きません。使用しようとすると、この制限が適用されることを示すメッセージが SAS によって口グに書き込まれます。従来のアプリケーションで ADDR を使用している場合、アプリケーションを変更して、かわりに ADDRLONG を使用します。ADDRLONG は、32 ビットと 64 ビットの両方のプラットフォームで使用できます。

比較

ADDR 関数は、32 ビットプラットフォームの変数のメモリアドレスを返します。ADDRLONG は、32 ビットおよび 64 ビットプラットフォームの変数のメモリアドレスを返します。

注: ADDRLONG は 32 ビットと 64 ビットの両方のプラットフォームで使用できるため、SAS では ADDR のかわりに ADDRLONG を使用することをお勧めします。

サンプル

次の例では、変数 FIRST が保存されているアドレスを返します。

```
data numlist;
first=3;
x=addr(first);
run;
```

関連項目:

関数:

- “PEEK 関数” (718 ページ)
- “PEEKC 関数” (719 ページ)
- “ADDRLONG 関数” (92 ページ)

CALL ルーチン:

- “CALL POKE ルーチン” (192 ページ)

ADDRLONG 関数

32 ビットおよび 64 ビットプラットフォームの変数のメモリアドレスを返します。

カテゴリ: 特殊関数

構文

ADDRLONG(*variable*)

必須引数

変数
変数を指定します。

詳細

戻り値は、アドレスのバイナリ表現を含む文字列です。この値を表示するには、$HEX*w*.形式を使用して、バイナリ値を同等の 16 進値に変換します。 変数に結果を保存する場合、移植性を確保するために 8 文字以上の文字変数に保存する必要があります。長さをまだ定義していない変数に結果を割り当てる場合、その変数の長さは 20 文字になります。

サンプル

次の例では、変数 ITEM のポインタアドレスを返して、値の出力形式を設定します。

```
data characterlist;
item=6345;
x=addrlong(item);
put x $hex16.;
run;
```

次の行が SAS ログに書き込まれます。

```
480063B020202020
```

AIRY 関数

Airy 関数の値を返します。

カテゴリ: 数学関数

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。

詳細

AIRY 関数は、Airy 関数の値を返します (リファレンス (971 ページ)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

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

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=airy(2.0);	0.0349241304
x=airy(−2.0);	0.2274074282

ALLCOMB 関数

n 個の変数値から一度に *k* 個の変数値を取得する場合のすべての組み合わせを変化量の小さい順に生成します。

カテゴリ: 組み合わせ関数

制限事項: ALLCOMB 関数は、%SYSFUNC マクロを使用する場合には実行できません。

構文

ALLCOMB(*count*, *k*, *variable-1, ... , variable-n*)

必須引数

count
ループで 1 ~ 組み合わせ数の値を割り当てる整数変数を指定します。

k
各組み合わせの項目数を示す 1 ~ *n*(1 と n を含む)の整数の定数、変数または式を指定します。

変数
すべての数値変数または同じ長さのすべての文字変数を指定します。これらの変数の値は置換されます。

制限事項: 33 個以内で項目を指定します。33 個を超える項目の組み合わせを検出する必要がある場合は、CALL ALLCOMBI ルーチンを使用します。

要件 ALLCOMB 関数を実行する前にこれらの変数を初期化します。

ヒント: ALLCOMB を実行すると、最初の *k* 個の変数に 1 つの組み合わせの値が含まれます。

詳細

ALLCOMB 関数をループで使用して、ALLCOMB の第 1 引数で各整数値(1 ~ 組み合わせ数)を受け入れます。*k* は定数です。組み合わせ数は、COMB 関数を使用して計算できます。最初の実行で、引数の種類と長さに不整合がないか確認されます。後続の各実行で、2 つの変数の値が交換されます。

ALLCOMB 関数では、次のアクションが実行されます。

- ALLCOMB は、最初の実行で 0 を返します。
- *variable-i* と *variable-j* の値が交換される場合(*i*<*j*)、ALLCOMB は *i* を返します。
- すべての組み合わせがすでに生成されていたために値が交換されなかった場合、ALLCOMB は–1 を返します。

ALLCOMB 関数を実行する場合に第 1 引数の順序が違っていると、その結果は役に立ちません。具体的には、変数を初期化した直後に第 1 引数に *j* を指定して ALLCOMB 関数を実行しても、*j*番目の組み合わせは取得できません(*j* が 1 の場合を除く)。*j*番目の組み合わせを取得するには、第 1 引数で 1 ~ *j* の値をそのままの順序で取得して、ALLCOMB を *j* 回実行する必要があります。

比較

SAS には、組み合わせを生成する 4 つの関数または CALL ルーチンがあります。

- ALLCOMB: N 個の変数の *値(欠損値または非欠損値)*の *考えられる*すべての組み合わせを生成します。値は数値または文字値になります。各組み合わせは、前の組み合わせに基づいて形成されます(1 つの値を削除して別の値を挿入)。
- LEXCOMB: 複数の変数の *非欠損値*の *重複しない*すべての組み合わせを生成します。値は数値または文字値になります。組み合わせは、辞書式順序で生成されます。
- ALLCOMBI: N 個の項目の *インデックス*のすべての組み合わせを生成します。*インデックス*は 1 ~ N の整数です。各組み合わせは、前の組み合わせに基づいて形成されます(1 つのインデックスを削除して別のインデックスを挿入)。
- LEXCOMBI: N 個の項目の *インデックス*のすべての組み合わせを生成します。*インデックス*は 1 ~ N の整数です。組み合わせは、辞書式順序で生成されます。

ALLCOMBI は最も速い関数および CALL ルーチンです。最も遅いのは LEXCOMB です。

サンプル

ALLCOMB 関数の例を次に示します。

```
data _null_;
array x[5] $3 ('ant' 'bee' 'cat' 'dog' 'ewe');
n=dim(x);
k=3;
ncomb=comb(n,k);
do j=1 to ncomb+1;
rc=allcomb(j, k, of x[*]);
put j 5. +3 x1-x3 +3 rc=;
end;
run;
```

SAS は次の出力を口グに書き込みます。

```
 1 ant bee cat rc=0
2 ant bee ewe rc=3
3 ant bee dog rc=3
4 ant cat dog rc=2
```

```
5 ant cat ewe rc=3
6 ant dog ewe rc=2
7 bee dog ewe rc=1
8 bee dog cat rc=3
9 bee ewe cat rc=2
10 dog ewe cat rc=1
11 dog ewe cat rc=-1
```

関連項目:

CALL ルーチン:

- “CALL ALLCOMB ルーチン” (148 ページ)

ALLPERM 関数

複数の変数の値のすべての順列を変化量の小さい順に生成します。

カテゴリ: 組み合わせ関数

構文

ALLPERM(*count*, *variable-1* <,*variable-2* …>)

必須引数

count
1 ~ 順列数の範囲の整数値で変数を指定します。

変数
すべての数値変数または同じ長さのすべての文字変数を指定します。これらの変数の値は置換されます。

制限事項: 18 個以内で変数を指定します。

要件 ALLPERM 関数を実行する前にこれらの変数を初期化します。

詳細

基本

ALLPERM 関数をループで使用して、ALLPERM の第 1 引数で各整数値(1 ~ 順列数)を受け入れます。最初の実行で、引数の種類と長さに不整合がないか確認されます。後続の各実行で、2 つの連続変数の値が交換されます。

注: 順列数は、PERM 関数を使用して計算できます。詳細については、“PERM 関数” (723 ページ)を参照してください。ALLPERM 関数では、次の値が返されます。

- 0 (*count*=1 の場合)
- J (*variable-J* および *variable-K* の値が交換される場合(K=J+1))
- –1 (*count*>N!の場合)

ALLPERM 関数を使用する場合に第 1 引数の順序が違っていると、その結果は役に立ちません。たとえば、変数を初期化した直後に第 1 引数に K を指定して ALLPERM 関数を実行しても、K 番目の順列の結果は得られません(K が 1 の場合

を除く)。K 番目の順列を取得するには、第 1 引数で 1～K の値をそのままの順序で取得して、ALLPERM 関数を K 回実行する必要があります。

ALLPERM は、一部の変数に同じ値や欠損値が含まれている場合でも常に N!個の順列を生成します。 同じ値があるときに重複しない順列のみを生成する場合や、順列から欠損値を除外する場合は、かわりに LEXPERM 関数を使用します。

注: ALLPERM 関数は、%SYSFUNC マクロを使用する場合には実行できません。

比較

SAS には、すべての順列を生成する 3 つの関数または CALL ルーチンがあります。

- ALLPERM: 複数の変数の値(*欠損値または非欠損値*)の *考えられる*すべての順列を生成します。各順列は、前の順列に基づいて形成されます(2 つの連続する値を交換)。
- LEXPERM: 複数の変数の *非欠損*値の *重複しない*すべての順列を生成します。順列は、辞書式順序で生成されます。
- LEXPERK: N 個の変数の *非欠損*値から K 個の *重複しない*すべての順列を生成します。順列は、辞書式順序で生成されます。

ALLPERM は最も速い関数および CALL ルーチンです。最も遅いのは LEXPERK です。

サンプル

次の例では、ALLPERM 関数を使用して特定の値の順列を生成します。

```
data _null_;
array x [4] $3 ('ant' 'bee' 'cat' 'dog');
n=dim(x);
nfact=fact(n);
do i=1 to nfact+1;
change=allperm(i, of x[*]);
put i 5. +2 change +2 x[*];
end;
run;
```

SAS は次の出力を口グに書き込みます。

```
 1 0 ant bee cat dog
2 3 ant bee dog cat
3 2 ant dog bee cat
4 1 dog ant bee cat
5 3 dog ant cat bee
6 1 ant dog cat bee
7 2 ant cat dog bee
8 3 ant cat bee dog
9 1 cat ant bee dog
10 3 cat ant dog bee
11 2 cat dog ant bee
12 1 dog cat ant bee
13 3 dog cat bee ant
14 1 cat dog bee ant
15 2 cat bee dog ant
16 3 cat bee ant dog
17 1 bee cat ant dog
```

```
18 3 bee cat dog ant
19 2 bee dog cat ant
20 1 dog bee cat ant
21 3 dog bee ant cat
22 1 bee dog ant cat
23 2 bee ant dog cat
24 3 bee ant cat dog
25 -1 bee ant cat dog
```

関連項目:

関数と CALL ルーチン:

- “CALL ALLPERM ルーチン” (153 ページ)
- “LEXPERM 関数” (614 ページ)
- “CALL RANPERK ルーチン” (220 ページ)
- “CALL RANPERM ルーチン” (222 ページ)

ANYALNUM 関数

文字列から英数字を検索し、最初に検索された文字の位置を返します。

カテゴリ: 文字関数

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

ANYALNUM(*string* <,*start*>)

必須引数

string
検索する文字の定数、変数または式を指定します。

オプション引数

start
検索を開始する位置を指定し、検索の方向を指定する整数です(省略可能)。

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。

サンプル

サンプル 1: 左から右への文字列のスキャン

次の例では、ANYALNUM 関数を使用して文字列から左から右の順に英数字を検索します。

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

次の行が SAS ログに書き込まれます。

```
 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
That's all
```

サンプル 2: 右から左への文字列のスキャン

次の例では、ANYALNUM 関数を使用して文字列から右から左の順に英数字を検索します。

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

次の行が SAS ログに書き込まれます。

```
 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
That's all
```

関連項目:

関数:

- “NOTALNUM 関数” (668 ページ)

ANYALPHA 関数

文字列から英字を検索し、最初に検索された文字の位置を返します。

カテゴリ: 文字関数

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

ANYALPHA(*string* <,*start*>)

必須引数

string
検索する文字の定数、変数または式です。

オプション引数

start
検索を開始する位置を指定し、検索の方向を指定する整数です(省略可能)。

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、文字列から英字を検索します。NOTALPHA 関数は、文字列から英字でない文字を検索します。

サンプル

サンプル 1: 文字列からの英字の検索

次の例では、ANYALPHA 関数を使用して文字列から左から右の順に英字を検索します。

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

次の行が SAS ログに書き込まれます。

```
j=1 c=N
j=2 c=e
j=3 c=x
j=4 c=t
```

```
j=9 c=n
j=16 c=E
That's all
```

サンプル 2: ANYALPHA 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、ANYALPHA 関数によって識別される制御文字が表示されます。

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

proc print data=test;
run;
```

関連項目:

関数:

- “NOTALPHA 関数” (669 ページ)

ANYCNTRL 関数

文字列からコントロール文字を検索し、最初に検索された文字の位置を返します。

カテゴリ: 文字関数

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

ANYCNTRL(*string* <,*start*>)

必須引数

string
検索する文字の定数、変数または式です。

オプション引数

start
検索を開始する位置を指定し、検索の方向を指定する整数です(省略可能)。

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。

ANYCNTRL 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

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。

サンプル

次のプログラムを実行すると、ANYCNTRL 関数によって識別される制御文字が表示されます。

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

proc print data=test;
run;
```

関連項目:

関数:

- “NOTCNTRL 関数” (671 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位置を返します。

カテゴリ: 文字関数

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

ANYDIGIT(*string* <,*start*>)

必須引数

string
検索する文字の定数、変数または式です。

オプション引数

start
検索を開始する位置を指定し、検索の方向を指定する整数です(省略可能)。

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。

サンプル

次の例では、ANYDIGIT 関数を使用して数字を検索します。

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

```
put +3 j= c=;
end;
end;
run;
```

次の行が SAS ログに書き込まれます。

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

関連項目:

関数:

- “NOTDIGIT 関数” (673 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位置を返します。

カテゴリ: 文字関数

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

ANYFIRST(*string* <,*start*>)

必須引数

string
検索する文字の定数、変数または式です。

オプション引数

start
検索を開始する位置を指定し、検索の方向を指定する整数です(省略可能)。

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
)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、AN

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。

サンプル

次の例では、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。

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

次の行が SAS ログに書き込まれます。

```
 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
That's all
```

関連項目:

関数:

- “NOTFIRST 関数” (676 ページ)

ANYGRAPH 関数

文字列からグラフィカル文字を検索し、最初に検索された文字の位置を返します。

カテゴリ: 文字関数

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

ANYGRAPH(*string* <,*start*>)

必須引数

string
検索する文字の定数、変数または式です。

オプション引数

start
検索を開始する位置を指定し、検索の方向を指定する整数です(省略可能)。

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。

サンプル

サンプル 1: 文字列からのグラフィカル文字の検索

次の例では、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。

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

次の行が SAS ログに書き込まれます。

```
 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=;
That's all
```

サンプル 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すると、ANYGRAPH 関数によって識別される制御文字が表示されます。

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

proc print data=test;
run;
```

関連項目:

関数:

- “NOTGRAPH 関数” (678 ページ)

ANYLOWER 関数

文字列から小文字を検索し、最初に検索された文字の位置を返します。

カテゴリ: 文字関数

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

ANYLOWER(*string* <,*start*>)

必須引数

string
検索する文字の定数、変数または式です。

オプション引数

start
検索を開始する位置を指定し、検索の方向を指定する整数です(省略可能)。

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。

サンプル

次の例では、ANYLOWER 関数を使用して文字列から小文字を検索します。

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

次の行が SAS ログに書き込まれます。

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

関連項目:

関数:

- "NOTLOWER 関数" (680 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位置を返します。

カテゴリ: 文字関数

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

ANYNAME(*string* <,*start*>)

必須引数

string
検索する文字の定数、変数または式です。

オプション引数

start
検索を開始する位置を指定し、検索の方向を指定する整数です(省略可能)。

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
)、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、AN

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。

サンプル

次の例では、ANYNAME 関数を使用して、VALIDVARNAME=V7 の SAS 変数名で有効な文字を文字列から検索します。

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

次の行が SAS ログに書き込まれます。

```
 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
That's all
```

関連項目:

関数:

- “NOTNAME 関数” (681 ページ)

ANYPRINT 関数

文字列から印刷可能な文字を検索し、最初に検索された文字の位置を返します。

カテゴリ: 文字関数

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

ANYPRINT(*string* <,*start*>)

必須引数

string
検索する文字の定数、変数または式です。

オプション引数

start
検索を開始する位置を指定し、検索の方向を指定する整数です(省略可能)。

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、文字列から印刷可能文字でない文字を検索します。

サンプル

サンプル 1: 文字列からの印刷可能文字の検索

次の例では、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。

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

次の行が SAS ログに書き込まれます。

```
 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=;
That's all
```

サンプル 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すると、ANYPRINT 関数によって識別される制御文字が表示されます。

```
data test;
do dec=0 to 255;
```

```
byte=byte(dec);
hex=put(dec,hex2.);
anyprint=anyprint(byte);
output;
end;

proc print data=test;
run;
```

関連項目:

関数:

- “NOTPRINT 関数” (683 ページ)

ANYPUNCT 関数

文字列から句読文字を検索し、最初に検索された文字の位置を返します。

カテゴリ: 文字関数

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

ANYPUNCT(*string* <,*start*>)

必須引数

string
検索する文字の定数、変数または式です。

オプション引数

start
検索を開始する位置を指定し、検索の方向を指定する整数です(省略可能)。

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。

サンプル

サンプル 1: 文字列からの句読文字の検索

次の例では、ANYPUNCT 関数を使用して文字列から句読文字を検索します。

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

次の行が SAS ログに書き込まれます。

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

サンプル 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。

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

proc print data=test;
run;
```

関連項目:

関数:

- “NOTPUNCT 関数” (684 ページ)

ANYSPACE 関数

文字列から空白文字(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)を検索し、最初に検索された文字の位置を返します。

カテゴリ: 文字関数

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

ANYSPACE(*string* <,*start*>)

必須引数

string
検索する文字の定数、変数または式です。

オプション引数

start
検索を開始する位置を指定し、検索の方向を指定する整数です(省略可能)。

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。

サンプル

サンプル 1: 文字列からの空白文字の検索

次の例では、ANYSPACE 関数を使用して文字列から空白文字を検索します。

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

次の行が SAS ログに書き込まれます。

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

サンプル 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。

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

proc print data=test;
run;
```

関連項目:

関数:

- “NOTSPACE 関数” (686 ページ)

ANYUPPER 関数

文字列から大文字を検索し、最初に検索された文字の位置を返します。

カテゴリ: 文字関数

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

ANYUPPER(*string* <,*start*>)

必須引数

string
検索する文字の定数、変数または式です。

オプション引数

start
検索を開始する位置を指定し、検索の方向を指定する整数です(省略可能)。

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。

サンプル

次の例では、ANYUPPER 関数を使用して文字列から大文字を検索します。

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

次の行が SAS ログに書き込まれます。

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

関連項目:

関数:

- “NOTUPPER 関数” (688 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位置を返します。

カテゴリ: 文字関数

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

ANYXDIGIT(*string* <,*start*>)

必須引数

string
検索する文字の定数、変数または式です。

オプション引数

start
検索を開始する位置を指定し、検索の方向を指定する整数です(省略可能)。

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。

サンプル

次の例では、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。

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

次の行が SAS ログに書き込まれます。

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

関連項目:

関数:

- “NOTXDIGIT 関数” (690 ページ)

ARCOS 関数

逆余弦を返します。

カテゴリ: 三角関数

構文

ARCOS (*argument*)

必須引数

引数

数値の定数、変数または式を指定します。

範囲: −1 ~ 1

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。

サンプル

SAS ステートメント	結果
x=arcos(1);	0
x=arcos(0);	1.5707963268
x=arcos(-0.5);	2.0943951024

ARCOSH 関数

逆双曲線余弦を返します。

カテゴリ: 双曲線関数

構文

ARCOSH(*x*)

必須引数

x

数値の定数、変数または式を指定します。

範囲: $x >= 1$

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($x >= 1$)で数学的に定義されます。

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

サンプル

次の例では、逆双曲線余弦を計算します。

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

SAS は次の出力を口グに書き込みます。

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

関連項目:

関数:

- “COSH 関数” (330 ページ)
- “SINH 関数” (837 ページ)
- “TANH 関数” (886 ページ)
- “ARSINH 関数” (123 ページ)
- “ARTANH 関数” (124 ページ)

ARSIN 関数

逆正弦を返します。

カテゴリ: 三角関数

構文

ARSIN (*argument*)

必須引数

引数
数値の定数、変数または式を指定します。

範囲: −1 ~ 1

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。

サンプル

SAS ステートメント	結果
x=arsin(0);	0
x=arsin(1);	1.5707963268
x=arsin(-0.5);	-0.523598776

ARSINH 関数

逆双曲線正弦を返します。

カテゴリ: 双曲線関数

構文

ARSINH(*x*)

必須引数

x
数値の定数、変数または式を指定します。

範囲: −∞< x <∞

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。-∞ < *x* < ∞

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

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。

サンプル

次の例では、逆双曲線正弦を計算します。

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

SAS は次の出力を口グに書き込みます。

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

関連項目:

関数:

- “COSH 関数” (330 ページ)
- “SINH 関数” (837 ページ)
- “TANH 関数” (886 ページ)
- “ARCOSH 関数” (121 ページ)
- “ARTANH 関数” (124 ページ)

ARTANH 関数

逆双曲線正接を返します。

カテゴリ: 双曲線関数

構文

ARTANH(*x*)

必須引数

x
数値の定数、変数または式を指定します。
範囲: $-1 < x < 1$

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(–1 < x< 1)で数学的に定義されます。$ARTANH(x) = \frac{1}{2} log\left(\frac{1 + x}{1 - x}\right)$

サンプル

次の例では、逆双曲線正接を計算します。

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

SAS は次の出力を口グに書き込みます。

```
x=0.5493061443
```

関連項目:

関数:

- “COSH 関数” (330 ページ)
- “SINH 関数” (837 ページ)

- “TANH 関数” (886 ページ)
- “ARCOSH 関数” (121 ページ)
- “ARSINH 関数” (123 ページ)

ATAN 関数

逆正接を返します。

カテゴリ: 三角関数

構文

ATAN (*argument*)

必須引数

引数
数値の定数、変数または式を指定します。

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 $-\pi/2$ ~ $\pi/2$です。引数がない場合、ATAN は欠損値を返します。

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。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=atan(0);	0
x=atan(1);	0.7853981634
x=atan(-9.0);	-1.460139106

関連項目:

関数

- “ATAN2 関数” (126 ページ)

ATAN2 関数

2 つの数値変数の比率の逆正接を返します。

カテゴリ: 三角関数

構文

ATAN2(*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。

argument-2
数値の定数、変数または式を指定します。

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、*argument-1*/*argument-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。ATAN2 は、ラジアンで結果を返します。値の範囲は $-\pi$ ~ π です。ATAN2 のいずれかの引数がない場合、ATAN2 は欠損値を返します。

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。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
a=atan2(-1, 0.5);	-1.107148718
b=atan2(6,8);	0.6435011088
c=atan2(5,-3);	2.1112158271

関連項目:

関数:

- “ATAN 関数” (125 ページ)

ATTRC 関数

SAS データセットの文字属性の値を返します。

カテゴリ: SAS ファイル I/O 関数

構文

ATTRC(*data-set-id*,*attr-name*)

必須引数

data-set-id
OPEN 関数が返すデータセット識別子を指定します。

attr-name
SAS データセットの属性の名前です。*attr-name* の値が無効な場合、欠損値が返されます。次のリストに、SAS データセットの属性名とその値を示します。

CHARSET
データセットを作成したコンピュータの文字セットの値を返します。

空の文字列	データセットが並べ替えられていません
ASCII	ASCII 文字セット
EBCDIC	EBCDIC 文字セット
ANSI	OS/2 ANSI 標準の ASCII 文字セット
OEM	OS/2 OEM コード形式

ENCRYPT
SAS データセットが暗号化されているかどうかに応じて'YES'または'NO'を返します。

ENGINE
データセットへのアクセスに使用するエンジンの名前を返します。

LABEL
データセットに割り当てられているラベルを返します。

LIB
データセットがある SAS ライブラリのライブラリ参照名を返します。

MEM
SAS データセット名を返します。

MODE
SAS データセットを開いている次のようなモードを返します。

I	INPUT モード: ランダムアクセスができます(エンジンでサポートされている場合)。それ以外の場合は、デフォルトの IN モードになります。
IN	INPUT モード: オブザベーションを順次読み込みます(オブザベーションの再アクセスができます)。
IS	INPUT モード: オブザベーションを順次読み込みます(オブザベーションの再アクセスはできません)。

N NEW モード: 新しいデータセットを作成します。

U UPDATE モード: ランダムアクセスができます(エンジンでサポートされている場合)。それ以外の場合は、デフォルトの UN モードになります。

UN UPDATE モード: オブザベーションを順次読み込みます(オブザベーションの再アクセスができます)。

US UPDATE モード: オブザベーションを順次読み込みます(オブザベーションの再アクセスはできません)。

V UTILITY モード: データセットに関連付けられている変数の属性およびインデックスを変更できます。

MTYPE
SAS ライブラリメンバの種類を返します。

SORTEDBY
データセットが並べ替えられていない場合、空の文字列を返します。それ以外の場合は、標準の BY ステートメント形式で BY 変数の名前を返します。

SORTLVL
データセットの並べ替え方法を示す値を返します。

空の文字列 データセットが並べ替えられていません。

WEAK データセットの並べ替え順序がユーザーによって確立されています(SORTEDBY データセットオプションなどを使用)。システムでその正当性を検証できないため、オブザベーションの順序には依存できません。

STRONG データセットの並べ替え順序がソフトウェアによって確立されています(PROC SORT や CONTENTS プロシジャの OUT=オプションなどを使用)。

SORTSEQ
データセットがネイティブコンピュータで並べ替えられている場合や、並べ替えの照合順序が動作環境のデフォルトである場合、空の文字列を返します。それ以外の場合は、ファイルの並べ替えに使用する代替照合順序の名前を返します。

TYPE
SAS データセットの種類を返します。

サンプル

サンプル 1: INPUT SEQUENTIAL モードに関するメッセージの SAS ログへの書き込み

この例では、SAS データセットが INPUT SEQUENTIAL モードで開いていない場合にメッセージを生成します。メッセージは次のように SAS ログに書き込まれます。

```
%let mode=%sysfunc(attrc(&dsid,MODE));
%if &mode ne IS %then
%put Data set has not been opened in INPUT SEQUENTIAL mode.;
```

サンプル 2: データセットが並べ替えられているかどうかのテスト

この例では、データセットが並べ替えられているかどうかをテストして、その結果を SAS ログに書き込みます。

```
data _null_;
dsid=open("sasdata.sortcars","i");
charset=attrc(dsid,"CHARSET");
if charset = "" then
put "Data set has not been sorted.";
else put "Data set sorted with " charset
"character set.";
rc=close(dsid);
run;
```

関連項目:

関数

- “ATTRN 関数” (129 ページ)
- “OPEN 関数” (697 ページ)

ATTRN 関数

SAS データセットの数値属性の値を返します。

カテゴリ: SAS ファイル I/O 関数

構文

ATTRN(*data-set-id*,*attr-name*)

必須引数

data-set-id
OPEN 関数が返すデータセット識別子を指定します。

attr-name
数値が返される SAS データセットの属性の名前です。*attr-name* の値が無効な場合、欠損値が返されます。次のリストに、SAS データセットの属性名とその値を示します。

ALTERPW
データセットの変更時にパスワードが要求されるかどうかを示します。

1 データセットは変更保護されています。

0 データセットは変更保護されていません。

ANOBS
オブザベーション数がエンジンで認識されているかどうかを示します。

1 オブザベーション数がエンジンで認識されています。

0 オブザベーション数がエンジンで認識されていません。

ANY
データセットにオブザベーションまたは変数があるかどうかを示します。

-1 データセットにオブザベーションまたは変数がありません。

0 データセットにオブザベーションがありません。

1 データセットにオブザベーションと変数があります。

別名: VAROBS

ARAND
ランダムアクセスがエンジンでサポートされているかどうかを示します。

1 ランダムアクセスがエンジンでサポートされています。

0 ランダムアクセスがエンジンでサポートされていません。

別名: RANDOM

ARWU
エンジンでファイルを操作できるかどうかを示します。

1 エンジンは読み取り専用ではありません。エンジンで SAS ファイルを作成または更新できます。

0 エンジンは読み取り専用です。

AUDIT
監査ファイルへの記録が有効になっているかどうかを示します。

1 記録が有効になっています。

0 記録が中断されています。

AUDIT_DATA
更新後のレコードイメージが保存されているかどうかを示します。

1 更新後のレコードイメージが保存されています。

0 更新後のレコードイメージが保存されていません。

AUDIT_BEFORE
更新前のレコードイメージが保存されているかどうかを示します。

1 更新前のレコードイメージが保存されています。

0 更新前のレコードイメージが保存されていません。

AUDIT_ERROR
失敗した更新後のレコードイメージが保存されているかどうかを示します。

1 失敗した更新後のレコードイメージが保存されています。

0 失敗した更新後のレコードイメージが保存されていません。

CRDTE
データセットの作成日を示します。戻り値は、作成日に対応する内部 SAS 日時値です。

ヒント: DATETIME.形式を使用してこの値を表示します。

ICONST
SAS データセットの一貫性制約の存在に関する情報を返します。

0 一貫性制約はありません。

1 1 つ以上の一般一貫性制約があります。

2 1 つ以上の参照一貫性制約があります。

3 1 つ以上の一般一貫性制約と 1 つ以上の参照一貫性制約の両方があります。

INDEX
データセットでインデックスがサポートされているかどうかを示します。

1 インデックスがサポートされています。

0 インデックスがサポートされていません。

ISINDEX
データセットにインデックスが付けられているかどうかを示します。

1 データセットに 1 つ以上のインデックスが存在します。

0 データセットにインデックスは付けられていません。

ISSUBSET
データセットがサブセットかどうかを示します。

1 1 つ以上の WHERE 句がアクティブになっています。

0 アクティブになっている WHERE 句はありません。

LRECL
論理レコード長を示します。

LRID
レコード ID の長さを示します。

MAXGEN
最大世代数を示します。

MAXRC
アプリケーションでリターンコードが確認されるかどうかを示します。

1 アプリケーションでリターンコードが確認されます。

0 アプリケーションでリターンコードが確認されません。

MODTE
データセットの最終変更日時を示します。戻り値は、内部 SAS 日時値です。

ヒント: DATETIME.形式を使用してこの値を表示します。

NDEL
削除対象としてマークされたデータセットのオブザベーション数を示します。

NEXTGEN
生成する次の世代番号を示します。

NLOBS
論理オブザベーション(削除対象としてマークされていないオブザベーション)の数を示します。アクティブな WHERE 句はこの数に影響しません。

-1 オブザベーション数は使用できません。

NLOBSF
各オブザベーションを強制的に読み込み、FIRSTOBS システムオプション、OBS システムオプション、WHERE 句を考慮して、論理オブザベーション(削除対象としてマークされていないオブザベーション)の数を示します。

ヒント: NLOBSF を ATTRN に渡すには、WHERE 句に一致するデータセットの各オブザベーションをエンジンで読み込む必要があります。ファイルの種類やサイズに基づいて、これらのオブザベーションを読み込むのは、時間のかかるプロセスです。

NOBS
物理オブザベーション(削除対象としてマークされているオブザベーションを含む)の数を示します。アクティブな WHERE 句はこの数に影響しません。

-1 オブザベーション数は使用できません。

NVARS
データセットの変数の数を示します。

PW
データセットのアクセス時にパスワードが要求されるかどうかを示します。

1 データセットは保護されています。

0 データセットは保護されていません。

RADIX
オブザベーション番号(基数によるアドレス指定)でアクセスできるかどうかを示します。

1 オブザベーション番号でアクセスできます。

0 オブザベーション番号でアクセスできません。

注: TAPE エンジンでアクセスするデータセットは、オブザベーション番号ではアクセスできませんが、インデックスでアドレス指定できます。

READPW
データセットの読み込み時にパスワードが要求されるかどうかを示します。

1 データセットは読み込み保護されています。

0 データセットは読み込み保護されていません。

TAPE
データセットテープのステータスを示します。

1 データセットはシーケンシャルファイルです。

0 データセットはシーケンシャルファイルではありません。

WHSTMT
アクティブな WHERE 句を示します。

0 アクティブになっている WHERE 句はありません。

1 永続的な WHERE 句がアクティブです。

2 一時的な WHERE 句がアクティブです。

3 永続的な WHERE 句と一時的な WHERE 句の両方がアクティブです。

WRITEPW
データセットへの書き込み時にパスワードが要求されるかどうかを示します。

1 データセットは書き込み保護されています。

0 データセットは書き込み保護されていません。

サンプル

サンプル 1: アクティブな WHERE 句の確認

この例では、WHERE 句がデータセットに対して現在アクティブかどうかを確認します。

```
%let iswhere=%sysfunc(attrn(&dsid,whstmt));
%if &iswhere %then
%put A WHERE clause is currently active.;
```

サンプル 2: インデックスが付けられているデータセットの確認

この例では、データセットにインデックスが付けらているどうかを確認します。

```
data _null_;
dsid=open("mydata");
isindex=attrn(dsid,"isindex");
if isindex then put "data set is indexed";
else put "data set is not indexed";
run;
```

サンプル 3: データセットのパスワード保護の確認

この例では、データセットがパスワードで保護されているかどうかを確認します。

```
data _null_;
dsid=open("mydata");
pw=attrn(dsid,"pw");
if pw then put "data set is protected";
run;
```

関連項目:

関数:

- “ATTRC 関数” (127 ページ)
- “OPEN 関数” (697 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: 論理ビット演算関数

構文

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

必須引数

argument-1, argument-2
数値の定数、変数または式を指定します。

範囲: 0 ~ (2^{32})-1(両端を含む)

詳細

いずれかの引数に欠損値が含まれている場合、この関数は欠損値を返して_ERROR_を 1 に設定します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=band(0Fx,05x); put x=hex.;	x=00000005

BETA 関数

beta 関数の値を返します。

カテゴリ: 数学関数

構文

BETA(*a*,*b*)

必須引数

a
第 1 形状パラメータです(*a*>0)。

b
第 2 形状パラメータです(*b*>0)。

詳細

BETA 関数は、式

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

で数学的に求められます(*a*>0、*b*>0)。以下に注意する必要があります。

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

$\Gamma(.)$ は gamma 関数です。

式を計算できない場合、BETA は欠損値を返します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=beta(5,3);	0.9523809524e-2

関連項目:

関数:

- “LOGBETA 関数” (624 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

構文

BETAINV (*p*,*a*,*b*)

必須引数

p
数値の確率です。
範囲: $0 \le p \le 1$

a
数値の形状パラメータです。
範囲: $a > 0$

b
数値の形状パラメータです。
範囲: $b > 0$

詳細

BETAINV 関数は、ベータ分布(形状パラメータは 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。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=betainv(0.001,2,4);	0.0101017879

関連項目:

関数:

- “QUANTILE 関数” (778 ページ)

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: 財務関数

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。

t
満期までの時間を指定する非欠損値。

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。

r
現時点から *t* までの無リスク金利を指定する正の非欠損分数。
要件 *t* の単位として同期間の *r* の値を指定します。

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。
要件 *t* の単位として同期間の *sigma* の値を指定します。

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。

$$CALL = \varepsilon^{-rt}\left(FN\left(d_1\right) - EN\left(d_2\right)\right)$$

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
期間 *t* の無リスク金利を指定します。

t
失効日までの時間を指定します。

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

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

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。

$$CALL = \max((F - E), 0)$$

価格の基礎については、価格関数の使用 (8 ページ)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値を返します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
	----+----1----+---2--
a=blackclprc(1000, .5, 950, 4, 2); put a;	65.335687119
b=blackclprc(850, 2.5, 125, 3, 1); put b;	0.012649067
c=blackclprc(7500, .9, 950, 3, 2); put c;	17.880939441

SAS ステートメント	結果
d=blackclprc(5000, -.5, 237, 3, 2); put d;	0

関連項目:

関数:

- “BLACKPTPRC 関数” (138 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: 財務関数

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。

t
満期までの時間を指定する非欠損値。

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。

r
現時点から *t* までの無リスク金利を指定する正の非欠損分数。
要件 *t* の単位として同期間の *r* の値を指定します。

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。
要件 *t* の単位として同期間の *sigma* の値を指定します。

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。

$$PUT = CALL + \varepsilon^{-rt}(E - F)$$

引数

E
オプションの権利行使価格を指定します。

r
期間 *t* の無リスク金利を指定します。

t
失効日までの時間を指定します。

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。

$PUT = \max((E - F), 0)$

価格の基礎については、価格関数の使用 (8 ページ)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値を返します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
	----+----1----+---2--
a=blackptprc(1000, .5, 950, 4, 2); put a;	72.102451281
b=blackptprc(850, 2.5, 125, 3, 1); put b;	0.4136352354
c=blackptprc(7500, .9, 950, 3, 2); put c;	458.07704789
d=blackptprc(5000, -.5, 237, 3, 2); put d;	0

関連項目:

関数:

- “BLACKCLPRC 関数” (136 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: 財務関数

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。

t
満期までの時間を指定する非欠損値。

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。

r
期間 *t* までの無リスク金利を指定する正の非欠損分数。
要件 *t* の単位として同期間の *r* の値を指定します。

sigma
原資産のボラティリティを指定する正の非欠損分数。
要件 *t* の単位として同期間の *sigma* の値を指定します。

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。

$$CALL = SN(d_1) - EN(d_2)\varepsilon^{-rt}$$

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
失効日までの時間を指定します。

r
期間 *t* の無リスク金利を指定します。

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。

$$CALL = \max((S - E), 0)$$

価格の基礎については、価格関数の使用 (8 ページ)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値を返します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
	----+----1----+---2--
a=blkshclprc(1000, .5, 950, 4, 2); put a;	831.05008469
b=blkshclprc(850, 2.5, 125, 3, 1); put b;	124.53035232
c=blkshclprc(7500, .9, 950, 3, 2); put c;	719.40891129
d=blkshclprc(5000, -.5, 237, 3, 2); put d;	0

関連項目:

関数:

- “BLKSHPTPRC 関数” (142 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: 財務関数

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。

t
満期までの時間を指定する非欠損値。

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。

r
期間 *t* までの無リスク金利を指定する正の非欠損分数。
要件 *t* の単位として同期間の *r* の値を指定します。

sigma
原資産のボラティリティを指定する正の非欠損分数。
要件 *t* の単位として同期間の *sigma* の値を指定します。

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。

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

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
失効日までの時間を指定します。

r
期間 *t* の無リスク金利を指定します。

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。

PUT = max ((*E* - *S*), 0)

価格の基礎については、価格関数の使用 (8 ページ)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値を返します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
	----+----1----+----2--
a=blkshptprc(1000, .5, 950, 4, 2); put a;	16.385367922
b=blkshptprc(850, 1.2, 125, 3, 1); put b;	1.426971358
c=blkshptprc(7500, .9, 950, 3, 2); put c;	273.45025684
d=blkshptprc(5000, -.5, 237, 3, 2); put d;	0

関連項目:

関数:

- “BLKSHCLPRC 関数” (140 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: 論理ビット演算関数

構文

BLSHIFT(*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: 0 ~ (2^{32})-1(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: 0 から 31(0 と 31 を含む)

詳細

いずれかの引数に欠損値が含まれている場合、この関数は欠損値を返して_ERROR_を 1 に設定します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=blshift(07x,2); put x=hex.;	x=0000001C

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: 論理ビット演算関数

構文

BNOT(*argument*)

必須引数

引数
数値の定数、変数または式を指定します。
範囲: 0 ~ (2^{32})-1(両端を含む)

詳細

引数が欠損値を含む場合、関数は欠損値を返して_ERROR_を 1 に設定します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=bnot(0F000000Fx); put x=hex.;	x=0FFFFFF0

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: 論理ビット演算関数

構文

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

必須引数

argument-1, argument-2
数値の定数、変数または式を指定します。

範囲: 0 ~ (2^{32})-1(両端を含む)

詳細

いずれかの引数に欠損値が含まれている場合、この関数は欠損値を返して_ERROR_を 1 に設定します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=bor(01x,0F4x); put x=hex.;	x=000000F5

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: 論理ビット演算関数

構文

BRSHIFT(*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: 0 ~ (2^{32})-1(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: 0 から 31(0 と 31 を含む)

詳細

いずれかの引数に欠損値が含まれている場合、この関数は欠損値を返して_ERROR_を 1 に設定します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=brshift(01Cx,2); put x=hex.;	x=00000007

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: 論理ビット演算関数

構文

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

必須引数

argument-1, argument-2
数値の定数、変数または式を指定します。
範囲: 0 ~ (2^{32})-1(両端を含む)

詳細

いずれかの引数に欠損値が含まれている場合、この関数は欠損値を返して_ERROR_を 1 に設定します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=bxor(03x,01x); put x=hex.;	x=00000002

BYTE 関数

ASCII 照合順序または EBCDIC 照合順序の 1 文字を返します。

カテゴリ: 文字関数

制限事項: I18N レベル 0 の関数は 1 バイト文字セット(SBCS)でのみ使用するように設計されています。

参照項目: “BYTE Function: UNIX” in *SAS Companion for UNIX Environments*

“BYTE Function: Windows” in *SAS Companion for Windows*

構文

BYTE (*n*)

必須引数

n

特定の ASCII または EBCDIC 文字を表す整数を指定します。

範囲: 0-255

詳細

返される変数の長さ

DATA ステップで、まだ長さが割り当てられていない変数に BYTE 関数から値が返される場合、その変数には長さ 1 が割り当てられます。

ASCII 照合順序および EBCDIC 照合順序

EBCDIC 照合順序では、*n* は 0 から 255 までの値になります。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。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果	
	ASCII	EBCDIC
	----+----1----+----2	----+----1----+----2

SAS ステートメント	結果	
x=byte(80); put x;	P	&

関連項目:

関数:

- “COLLATE 関数” (302 ページ)
- “RANK 関数” (797 ページ)

CALL ALLCOMB ルーチン

n 個の変数値から一度に *k* 個の変数値を取得する場合のすべての組み合わせを変化量の小さい順に生成します。

カテゴリ: 組み合わせ関数

構文

CALL ALLCOMB(*count*, *k*, *variable-1*, …,*variable-n*);

必須引数

count
整数の変数を指定します。変数には 1 ~ 組み合わせ数までがループで割り当てられます。

k
各組み合わせの項目数を示す 1 ~ *n*(1 と n を含む)の整数の定数、変数または式を指定します。

変数
すべての数値変数または同じ長さのすべての文字変数を指定します。これらの変数の値は置換されます。

制限事項: 33 個以内で項目を指定します。33 個を超える項目の組み合わせを検出する必要がある場合は、CALL ALLCOMBI ルーチンを使用します。

要件 ALLCOMB ルーチンを呼び出す前にこれらの変数を初期化します。

ヒント: ALLCOMB ルーチンを呼び出すと、最初の *k* 個の変数に 1 つの組み合わせの値が含まれます。

詳細

CALL ALLCOMB の処理

CALL ALLCOMB ルーチンをループで使用して、CALL ALLCOMB の第 1 引数で各整数値(1 ~ 組み合わせ数)を受け入れます。*k* は定数です。組み合わせ数は、COMB 関数を使用して計算できます。最初の呼び出しで、引数の種類と長さに不整合がないか確認されます。後続の各呼び出しでは、2 つの変数の値が交換されます。

ALLPER ルーチンを呼び出す場合に第 1 引数の順序が違っていると、その結果は役に立ちません。特に、変数を初期化した直後に第 1 引数を *j* として ALLCOMB を呼び出すと、*j*番目の組み合わせは取得されません(*j* が 1 である場合を除く)。*j* 番目の組み合わせを取得するには、第 1 引数に 1 ~ *j* の値をそのままの順序で取得して、ALLCOMB を *j* 回呼び出す必要があります。

マクロと CALL ALLCOMB ルーチンを使用する

ALLCOMB ルーチンは、%SYSCALL マクロを使って呼び出せます。この場合、*変数*の引数を同一の種類または長さとする必要はありません。%SYSCALL で引数が数値であることが識別されると、%SYSCALL は戻り値の形式を調整します。

CALL ALLCOMB ルーチンの実行中に工ラーが発生すると、次の両方の値が設定されます。

- &SYSERR には 4 よりも大きい値が割り当てられます。
- &SYSINFO には-100 よりも小さい値が割り当てられます。

エラーがない場合、&SYSERR はゼロに設定されて、&SYSINFO は次のいずれかの値に設定されます。

- 0 (*count*=1 の場合)
- *j(variable-j* および *variable-k* の値が交換されていて、*j*<*k* の場合)
- –1(重複しないすべての組み合わせが生成済みのために値が交換されなかった場合)

比較

SAS には、組み合わせを生成する 4 つの関数または CALL ルーチンがあります。

- ALLCOMB: *n* 個の変数の *値(欠損値または非欠損値)*の *考えられる*すべての組み合わせを生成します。値は数値または文字値になります。各組み合わせは、前の組み合わせに基づいて形成されます(1 つの値を削除して別の値を挿入)。
- LEXCOMB: 複数の変数の *非欠損値*の *重複しない*すべての組み合わせを生成します。値は数値または文字値になります。組み合わせは、辞書式順序で生成されます。
- ALLCOMBI は、*n* 個の項目の *インデックス*のすべての組み合わせを生成します。*インデックス*は、1 ~ *n* の整数です。各組み合わせは、前の組み合わせに基づいて形成されます(1 つのインデックスを削除して別のインデックスを挿入)。
- LEXCOMBI は、*n* 個の項目の *インデックス*のすべての組み合わせを生成します。*インデックス*は、1 ~ *n* の整数です。組み合わせは、辞書式順序で生成されます。

ALLCOMBI は最も速い関数および CALL ルーチンです。最も遅いのは LEXCOMB です。

サンプル

サンプル 1: DATA ステップで CALL ALLCOMB を使用する

CALL ALLCOMB ルーチンを DATA ステップで使用している例を次に示します。

```
data _null_;
array x[5] $3 ('ant' 'bee' 'cat' 'dog' 'ewe');
```

```
n=dim(x);
k=3;
ncomb=comb(n,k);
do j=1 to ncomb+1;
call allcomb(j, k, of x[*]);
put j 5. +3 x1-x3;
end;
run;
```

SAS は次の出力を口グに書き込みます。

```
 1 ant bee cat
2 ant bee ewe
3 ant bee dog
4 ant cat dog
5 ant cat ewe
6 ant dog ewe
7 bee dog ewe
8 bee dog cat
9 bee ewe cat
10 dog ewe cat
11 dog ewe cat
```

サンプル 2: マクロと CALL ALLCOMB を使用し、リターンコードを表示する

CALL ALLCOMB ルーチンをマクロで使用している例を次に示します。出力には%SYSINFO マクロ用の値が含まれます。

```
%macro test;
%let x1=ant;
%let x2=-.1234;
%let x3=1e10;
%let x4=hippopotamus;
%let x5=zebra;
%let k=2;
%let ncomb=%sysfunc(comb(5,&k));
%do j=1 %to &ncomb+1;
%syscall allcomb(j, k, x1, x2, x3, x4, x5);
%let jfmt=%qsysfunc(putn(&j,5.));
%let pad=%qsysfunc(repeat(%str(),30-%length(&x1 &x2)));
%put &jfmt: &x1 &x2 &pad sysinfo=&sysinfo;
%end;
%mend;

%test
```

SAS は次の出力を口グに書き込みます。

```
 1: ant -0.1234 sysinfo=0
2: ant zebra sysinfo=2
3: ant hippopotamus sysinfo=2
4: ant 10000000000 sysinfo=2
5: -0.1234 10000000000 sysinfo=1
6: -0.1234 zebra sysinfo=2
7: -0.1234 hippopotamus sysinfo=2
8: 10000000000 hippopotamus sysinfo=1
9: 10000000000 zebra sysinfo=2
```

10: hippopotamus zebra sysinfo=1
11: hippopotamus zebra sysinfo=-1

関連項目:

関数:

- “ALLCOMB 関数” (94 ページ)

CALL ALLCOMBI ルーチン

n 個のオブジェクトを同時に *k* 個使用するときのインデックスのすべての組み合わせを、変化量の小さい順に生成します。

カテゴリ: 組み合わせ関数

構文

CALL ALLCOMBI(*N*, *K*, *index-1*, …, *index-K*, <, *index-added*, *index-removed*>);

必須引数

N
オブジェクトの合計数を指定する数値の定数、変数または式です。

K
各組み合わせのオブジェクト数を指定する数値の定数、変数または式です。

index
返される組み合わせでオブジェクトのインデックスが入る数値変数です。インデックスは 1 ~ N(1 と N を含む)の整数です。

ヒント: *index-1* が欠損またはゼロの場合、ALLCOMBI は **index-K=K** までのインデックスを **index-1=1** に初期化します。それ以外の場合は、ALLCOMBI は組み合わせからインデックスの 1 つを削除し、他のインデックスを追加して新しい組み合わせを作ります。

オプション引数

index-added
追加したインデックスの値を返すために ALLCOMBI が使う数値変数です。

index-removed
削除したインデックスの値を返すために ALLCOMBI が使う数値変数です。

詳細

CALL ALLCOMBI の処理

初めて ALLCOMBI を呼び出すときは、先に次のタスクのいずれかを実行します。

- *index-1* をゼロまたは欠損値に設定します。
- *index-1* ~ *index-K* を 1 ~ N(1 と N を含む)の重複しない整数に初期化します。

N 個のオブジェクトを同時に K 個使用するときの組み合わせ数は、COMB(*N*, *K*)で計算できます。N 個のオブジェクトを同時に K 個使用するときのすべての組み合わせを生成するには、COMB(*N*, *K*)回実行するループで ALLCOMBI を呼び出します。

マクロと CALL ALLCOMBI ルーチンを使用する

%SYSCALL を使ったマクロ処理から ALLCOMBI を呼び出す場合は、すべての引数を初期化して数値にする必要があります。&SYSCALL は戻り値の形式を調整します。

CALL ALLCOMBI ルーチンの実行中にエラーが発生すると、次の両方の値が設定されます。

- &SYSERR には 4 よりも大きい値が割り当てられます。
- &SYSINFO には-100 よりも小さい値が割り当てられます。

エラーがない場合、&SYSERR および&SYSINFO はゼロに設定されます。

比較

CALL ALLCOMBI ルーチンは、N 個のオブジェクトを同時に K 個使用するときのすべてのインデックスの組み合わせを、変化量の小さい順に生成します。CALL ALLCOMB ルーチンは、N 個のオブジェクトを同時に K 個使用するときのすべての値の組み合わせを、変化量の小さい順に生成します。

サンプル

サンプル 1: DATA ステップで CALL ALLCOMBI を使用する

CALL ALLCOMBI ルーチンを DATA ステップで使用している例を次に示します。

```
data _null_;
array x[5] $3 ('ant' 'bee' 'cat' 'dog' 'ewe');
array c[3] $3;
array i[3];
n=dim(x);
k=dim(i);
i[1]=0;
ncomb=comb(n,k); /* The one extra call goes back */
do j=1 to ncomb+1; /* to the first combination. */
call allcombi(n, k, of i[*], add, remove);
do h=1 to k;
c[h]=x[i[h]];
end;
put @4 j= @10 'i= ' i[*] +3 'c= ' c[*] +3 add= remove=;
end;
run;
```

SAS は次の出力をログに書き込みます。

```
 j=1 i= 1 2 3 c= ant bee cat add=0 remove=0
j=2 i= 1 3 4 c= ant cat dog add=4 remove=2
j=3 i= 2 3 4 c= bee cat dog add=2 remove=1
j=4 i= 1 2 4 c= ant bee dog add=1 remove=3
j=5 i= 1 4 5 c= ant dog ewe add=5 remove=2
j=6 i= 2 4 5 c= bee dog ewe add=2 remove=1
j=7 i= 3 4 5 c= cat dog ewe add=3 remove=2
```

```
j=8 i= 1 3 5 c= ant cat ewe add=1 remove=4
j=9 i= 2 3 5 c= bee cat ewe add=2 remove=1
j=10 i= 1 2 5 c= ant bee ewe add=1 remove=3
j=11 i= 1 2 3 c= ant bee cat add=3 remove=5
```

サンプル 2: マクロと CALL ALLCOMBI を使用する

CALL ALLCOMBI ルーチンをマクロで使用している例を次に示します。

```
%macro test;
%let x1=0;
%let x2=0;
%let x3=0;
%let add=0;
%let remove=0;
%let n=5;
%let k=3;
%let ncomb=%sysfunc(comb(&n,&k));
%do j=1 %to &ncomb;
%syscall allcombi(n,k,x1,x2,x3,add,remove);
%let jfmt=%qsysfunc(putn(&j,5.));
%put &jfmt: &x1 &x2 &x3 add=&add remove=&remove;
%end;
%mend;
%test
```

SAS は次の出力を口グに書き込みます。

```
 1: 1 2 3 add=0 remove=0
2: 1 3 4 add=4 remove=2
3: 2 3 4 add=2 remove=1
4: 1 2 4 add=1 remove=3
5: 1 4 5 add=5 remove=2
6: 2 4 5 add=2 remove=1
7: 3 4 5 add=3 remove=2
8: 1 3 5 add=1 remove=4
9: 2 3 5 add=2 remove=1
10: 1 2 5 add=1 remove=3
```

関連項目:

CALL ルーチン:

- “CALL ALLCOMB ルーチン” (148 ページ)

CALL ALLPERM ルーチン

複数の変数の値のすべての順列を変化量の小さい順に生成します。

カテゴリ: 組み合わせ関数

構文

CALL ALLPERM(*count*, *variable–1*<, *variable–2 ...*>);

必須引数

count
1 ~ 順列数の範囲の整数の変数を指定します。

変数
すべての数値変数または同じ長さのすべての文字変数を指定します。これらの変数の値は置換されます。

制限事項: 18 個以内で変数を指定します。

要件 ALLPERM ルーチンを呼び出す前にこれらの変数を初期化します。

詳細

CALL ALLPERM の処理

CALL ALLPERM ルーチンをループで使用して、CALL ALLPERM の第 1 引数で各整数値(1 ~ 順列)を受け入れます。最初の呼び出しで、引数の種類と長さに不整合がないか確認されます。後続の呼び出しで、2 つの連続変数の値が交換されます。

注: 順列数は、PERM 関数を使用して計算できます。詳細については、PERM 関数 (723 ページ)を参照してください。

ALLPER ルーチンを呼び出す場合に第 1 引数の順序が違っていると、その結果は役に立ちません。特に、変数を初期化した直後に第 1 引数に K を指定して ALLPERM ルーチンを呼び出しても、K *番目*の順列の結果は得られません(K が 1 である場合を除く)。K *番目*の順列を取得するには、第 1 引数を 1 ~ K の値をそのままの順序で取得して、ALLPERM を K 回呼び出す必要があります。

ALLPERM は、一部の変数に同じ値や欠損値が含まれている場合でも常に N!個の順列を生成します。 同じ値があるときに重複しない順列のみを生成する場合や、順列から欠損値を除外する場合は、かわりに LEXPERM 関数を使用します。

マクロと CALL ALLPERM ルーチンを使用する

ALLPERM ルーチンは、%SYSCALL マクロを使って呼び出せます。この場合、*変数*の引数を同一の種類または長さとする必要はありません。%SYSCALL で引数が数値であることが識別されると、%SYSCALL は戻り値の形式を調整します。

CALL ALLPERM ルーチンの実行中にエラーが発生すると、次の両方の値が設定されます。

- &SYSERR には 4 よりも大きい値が割り当てられます。
- &SYSINFO には-100 よりも小さい値が割り当てられます。

エラーがない場合、&SYSERR はゼロに設定されて、&SYSINFO は次のいずれかの値に設定されます。

- 0 (*count*=1 の場合)
- J(1<*count*<=N!、かつ *variable-J* および *variable-K* の値が交換されていて、J+1=K の場合)
- -1 (*count*>N!の場合)

比較

SAS には、すべての順列を生成する 3 つの関数または CALL ルーチンがあります。

- ALLPERM: 複数の変数の値(*欠損値または非欠損値*)の *考えられる*すべての順列を生成します。各順列は、前の順列に基づいて形成されます(2 つの連続する値を交換)。
- LEXPERM: 複数の変数の *非欠損*値の *重複しない*すべての順列を生成します。順列は、辞書式順序で生成されます。
- LEXPERK: N 個の変数の *非欠損*値から K 個の *重複しない*すべての順列を生成します。順列は、辞書式順序で生成されます。

ALLPERM は最も速い関数および CALL ルーチンです。最も遅いのは LEXPERK です。

サンプル

サンプル 1: DATA ステップで CALL ALLPERM を使用する

次の例は、CALL ALLPERM ルーチンを使って指定した値の順列を生成します。

```
data _null_;
array x [4] $3 ('ant' 'bee' 'cat' 'dog');
n=dim(x);
nfact=fact(n);
do i=1 to nfact;
call allperm(i, of x[*]);
put i 5. +2 x[*];
end;
run;
```

SAS は次の出力を口グに書き込みます。

```
 1 ant bee cat dog
2 ant bee dog cat
3 ant dog bee cat
4 dog ant bee cat
5 dog ant cat bee
6 ant dog cat bee
7 ant cat dog bee
8 ant cat bee dog
9 cat ant bee dog
10 cat ant dog bee
11 cat dog ant bee
12 dog cat ant bee
13 dog cat bee ant
14 cat dog bee ant
15 cat bee dog ant
16 cat bee ant dog
17 bee cat ant dog
18 bee cat dog ant
19 bee dog cat ant
20 dog bee cat ant
21 dog bee ant cat
22 bee dog ant cat
23 bee ant dog cat
24 bee ant cat dog
```

サンプル 2: マクロと CALL ALLPERM を使用する

CALL ALLPERM ルーチンをマクロで使用している例を次に示します。出力には%SYSINFO マクロ用の値が含まれます。

```
%macro test;
%let x1=ant;
%let x2=-.1234;
%let x3=1e10;
%let x4=hippopotamus;
%let nperm=%sysfunc(perm(4));
%do j=1 %to &nperm+1;
%syscall allperm(j, x1, x2, x3, x4);
%let jfmt=%qsysfunc(putn(&j,5.));
%put &jfmt: &x1 &x2 &x3 &x4 sysinfo=&sysinfo;
%end;
%mend;

%test;
```

SAS は次の出力を口グに書き込みます。

```
 1: ant -0.1234 10000000000 hippopotamus sysinfo=0
2: ant -0.1234 hippopotamus 10000000000 sysinfo=3
3: ant hippopotamus -0.1234 10000000000 sysinfo=2
4: hippopotamus ant -0.1234 10000000000 sysinfo=1
5: hippopotamus ant 10000000000 -0.1234 sysinfo=3
6: ant hippopotamus 10000000000 -0.1234 sysinfo=1
7: ant 10000000000 hippopotamus -0.1234 sysinfo=2
8: ant 10000000000 -0.1234 hippopotamus sysinfo=3
9: 10000000000 ant -0.1234 hippopotamus sysinfo=1
10: 10000000000 ant hippopotamus -0.1234 sysinfo=3
11: 10000000000 hippopotamus ant -0.1234 sysinfo=2
12: hippopotamus 10000000000 ant -0.1234 sysinfo=1
13: hippopotamus 10000000000 -0.1234 ant sysinfo=3
14: 10000000000 hippopotamus -0.1234 ant sysinfo=1
15: 10000000000 -0.1234 hippopotamus ant sysinfo=2
16: 10000000000 -0.1234 ant hippopotamus sysinfo=3
17: -0.1234 10000000000 ant hippopotamus sysinfo=1
18: -0.1234 10000000000 hippopotamus ant sysinfo=3
19: -0.1234 hippopotamus 10000000000 ant sysinfo=2
20: hippopotamus -0.1234 10000000000 ant sysinfo=1
21: hippopotamus -0.1234 ant 10000000000 sysinfo=3
22: -0.1234 hippopotamus ant 10000000000 sysinfo=1
23: -0.1234 ant hippopotamus 10000000000 sysinfo=2
24: -0.1234 ant 10000000000 hippopotamus sysinfo=3
25: -0.1234 ant 10000000000 hippopotamus sysinfo=-1
```

関連項目:

関数:

- “LEXPERM 関数” (614 ページ)
- “ALLPERM 関数” (96 ページ)

CALL ルーチン:

- “CALL RANPERK ルーチン” (220 ページ)

- “CALL RANPERM ルーチン” (222 ページ)

CALL CATS ルーチン

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: 文字関数

構文

CALL CATS(*result* <, *item-1*, …, *item-n*>);

必須引数

result
文字変数を指定します。

制限事項: CALL CATS ルーチンが *result* の有効な引数として受け付けるのは文字変数のみです。これらの引数は CALL CATS で更新できないため、定数または SAS 式は使用しないでください。

オプション引数

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、BEST*w* 書式の文字列に値が変換されます。この場合、ログにメモは記録されません。

詳細

CALL CATS ルーチンは第 1 引数 *result* の結果を返します。*result* に続く引数の値がルーチンで追加されます。結果全体を格納するには *result* の長さが足りない場合、次が実行されます。

- 結果が切り捨てら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を示すメモがログに出力されて、切り捨ての原因となった引数のリストが表示されます(SQL 句内または WHERE 句内を除く)。
- DATA ステップ内で _ERROR_ を 1 に設定します(WHERE 句内を除く)。

CALL CATS ルーチンは、数値を BEST*w* 書式で書式設定したあと、先頭と末尾の空白を数値引数から削除します。

比較

通常、CALL CATS、CALL CATT、CALL CATX ルーチンは連結演算子(||)を TRIM および LEFT 関数と使用するステートメントと同等です。ただし、CALL CATS、CALL CATT、CALL CATX ルーチンの方が、TRIM および LEFT を使うよりも高速です。

CALL CATS、CALL CATT、CALL CATX と同等のステートメントを次の表に示します。X1 ~ X4 の変数は文字変数を指定し、SP は空白やカンマなどの区切り文字を示します。

CALL ルーチン	同等のステートメント
CALL CATS(OF X1-X4);	X1=TRIM(LEFT(X1))||TRIM(LEFT(X2))||TRIM(LEFT(X3))|| TRIM(LEFT(X4));
CALL CATT(OF X1-X4);	X1=TRIM(X1)||TRIM(X2)||TRIM(X3)||TRIM(X4);
CALL CATX(SP, OF X1-X4); *	X1=TRIM(LEFT(X1))||SP||TRIM(LEFT(X2))||SP|| TRIM(LEFT(X3))||SP||TRIM(LEFT(X4));

注: いずれかの引数が空白の場合、CALL CATX によって出力される結果は連結コードの出力するコードとわずかに異なります。この場合、CALL CATX は対応する区切り文字を省略します。たとえば、**CALL CATX(″+″,″X″,″ ″, ″Z″,″ ″);** は **X+Z** を出力します。

サンプル

CALL CATS ルーチンが文字列を連結する方法を次の例に示します。

```
data _null_;
length answer $ 36;
x='Athens is t ';
y=' he Olym ';
z=' pic site for 2004. ';
call cats(answer,x,y,z);
put answer;
run;
```

次の行が SAS ログに書き込まれます。

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7
Athens is the Olympic site for 2004.
```

関連項目:

関数:

- “CAT 関数” (259 ページ)
- “CATQ 関数” (261 ページ)
- “CATS 関数” (265 ページ)
- “CATT 関数” (267 ページ)
- “CATX 関数” (270 ページ)

CALL ルーチン:

- “CALL CATX ルーチン” (160 ページ)
- “CALL CATT ルーチン” (158 ページ)

CALL CATT ルーチン

末尾の空白を削除して、連結文字列を返します。

カテゴリ: 文字関数

構文

CALL CATT(*result* <, *item-1*, … *item-n*>);

必須引数

result
文字変数を指定します。

制限事項: CALL CATT ルーチンが *result* の有効な引数として受け付けるのは文字変数のみです。これらの引数は CALL CATT で更新できないため、定数または SAS 式は使用しないでください。

オプション引数

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、BEST*w* 書式の文字列に値が変換されます。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

CALL CATT ルーチンは第 1 引数 *result* の結果を返します。*result* に続く引数の値がルーチンで追加されます。結果全体を格納するには *result* の長さが足りない場合、次が実行されます。

- 結果が切り捨てら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を示すメモがログに出力されて、切り捨ての原因となった引数のリストが表示されます(SQL 句内または WHERE 句内を除く)。
- DATA ステップ内で_ERROR_を 1 に設定します(WHERE 句内を除く)。

CALL CATT ルーチンは、数値を BEST*w* 書式で書式設定したあと、先頭と末尾の空白を数値引数から削除します。

比較

通常、CALL CATS、CALL CATT、CALL CATX ルーチンは連結演算子(||)を TRIM および LEFT 関数と使用するステートメントと同等です。ただし、CALL CATS、CALL CATT、CALL CATX ルーチンの方が、TRIM および LEFT を使うよりも高速です。

CALL CATS、CALL CATT、CALL CATX と同等のステートメントを次の表に示します。X1 ~ X4 の変数は文字変数を指定し、SP は空白やカンマなどの区切り文字を示します。

CALL ルーチン	同等のステートメント
CALL CATS(OF X1-X4);	X1=TRIM(LEFT(X1))||TRIM(LEFT(X2))||TRIM(LEFT(X3))||TRIM(LEFT(X4));
CALL CATT(OF X1-X4);	X1=TRIM(X1)||TRIM(X2)||TRIM(X3)||TRIM(X4);
CALL CATX(SP, OF X1-X4); *	X1=TRIM(LEFT(X1))||SP||TRIM(LEFT(X2))||SP||TRIM(LEFT(X3))||SP||TRIM(LEFT(X4));

注: いずれかの引数が空白の場合、CALL CATX によって出力される結果は連結コードの出力するコードとわずかに異なります。この場合、CALL CATX は対応する区切り文字を省略します。たとえば、CALL CATX(″+″,″X″,″ ″, ″Z″,″ ″); は X+Z を出力します。

サンプル

CALL CATT ルーチンが文字列を連結する方法を次の例に示します。

```
data _null_;
length answer $ 36;
x='London is t ';
y='he Olym ';
z='pic site for 2012. ';
call catt(answer,x,y,z);
put answer;
run;
```

次の行が SAS ログに書き込まれます。

```
----+----1----+----2----+----3----+----4
London is the Olympic site for 2012.
```

関連項目:

関数:

- “CAT 関数” (259 ページ)
- “CATQ 関数” (261 ページ)
- “CATS 関数” (265 ページ)
- “CATT 関数” (267 ページ)
- “CATX 関数” (270 ページ)

CALL ルーチン:

- “CALL CATX ルーチン” (160 ページ)
- “CALL CATS ルーチン” (157 ページ)

CALL CATX ルーチン

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: 文字関数

構文

CALL CATX(*delimiter*, *result* <, *item-1*, … *item-n*>);

必須引数

delimiter
連結文字列の間で区切り文字として使用する文字列を指定します。

result
文字変数を指定します。

制限事項: CALL CATX ルーチンが *result* の有効な引数として受け付けるのは文字変数のみです。これらの引数は CALL CATX で更新できないため、定数または SAS 式は使用しないでください。

オプション引数

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、BEST*w* 書式の文字列に値が変換されます。この場合、ログにメモは記録されません。

詳細

CALL CATX ルーチンは第 2 引数 *result* の結果を返します。*result* に続く引数の値がルーチンで追加されます。結果全体を格納するには *result* の長さが足りない場合、次が実行されます。

- 結果が切り捨てら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を示すメモがログに出力されて、切り捨ての原因となった引数のリストが表示されます(SQL 句内または WHERE 句内を除く)。
- DATA ステップ内で _ERROR_ を 1 に設定します(WHERE 句内を除く)。

CALL CATX ルーチンは、数値を BEST*w* 書式で書式設定したあと、先頭と末尾の空白を数値引数から削除します。

比較

通常、CALL CATS、CALL CATT、CALL CATX ルーチンは連結演算子(||)を TRIM および LEFT 関数と使用するステートメントと同等です。ただし、CALL CATS、CALL CATT、CALL CATX ルーチンの方が、TRIM および LEFT を使うよりも高速です。

CALL CATS、CALL CATT、CALL CATX と同等のステートメントを次の表に示します。X1 ~ X4 の変数は文字変数を指定し、SP は空白やカンマなどの区切り文字を示します。

CALL ルーチン	同等のステートメント
CALL CATS(OF X1-X4);	X1=TRIM(LEFT(X1))||TRIM(LEFT(X2))||TRIM(LEFT(X3))|| TRIM(LEFT(X4));
CALL CATT(OF X1-X4);	X1=TRIM(X1)||TRIM(X2)||TRIM(X3)||TRIM(X4);
CALL CATX(SP, OF X1-X4); *	X1=TRIM(LEFT(X1))||SP||TRIM(LEFT(X2))||SP|| TRIM(LEFT(X3))||SP||TRIM(LEFT(X4));

注: いずれかの引数が空白の場合、CALL CATX によって出力される結果は連結コードが出力するコードとわずかに異なります。この場合、CALL CATX は

対応する区切り文字を省略します。たとえば、CALL CATX(″+″,newvar,″X″,″ ″,″Z″,″ ″);は X+Z を出力します。

サンプル

CALL CATX ルーチンが文字列を連結する方法を次の例に示します。

```
data _null_;
length answer $ 50;
separator='%%$%%';
x='Athens is t ';
y='he Olym ';
z=' pic site for 2004. ';
call catx(separator,answer,x,y,z);
put answer;
run;
```

次の行が SAS ログに書き込まれます。

```
----+----1----+----2----+----3----+----4----+----5
Athens is t%%$%%he Olym%%$%%pic site for 2004.
```

関連項目:

関数:

- “CAT 関数” (259 ページ)
- “CATQ 関数” (261 ページ)
- “CATS 関数” (265 ページ)
- “CATT 関数” (267 ページ)
- “CATX 関数” (270 ページ)

CALL ルーチン:

- “CALL CATS ルーチン” (157 ページ)
- “CALL CATT ルーチン” (158 ページ)

CALL COMPCOST ルーチン

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。

カテゴリ: 文字関数

制限事項: COMPGED 関数と合わせて使用します。

操作: %SYSCALL マクロステートメントで呼び出されると、CALL COMPCOST の引数から引用符が削除されます。詳細については、“CALL ルーチンと%SYSCALL マクロステートメントを使用する” (9 ページ)を参照してください。

構文

CALL COMPCOST(*operation-1*, *value-1* <,*operation-2*, *value-2* …>);

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を確認するには、“一般化編集距離の計算” (312 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。演算の有効な値と、デフォルトの演算のコストを次の表に示します。

操作	デフォルトコスト
APPEND=	very large
BLANK=	very large
DELETE=	100
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 ルーチンの呼び出し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程に十分大きいコストを示します。

プログラムから COMPCOST ルーチンが呼び出された後、指定コストは COMPCOST ルーチンがプログラムから再度呼び出されるまで、または COMPCOST への呼び出しを含むステップが終了するまで有効です。

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。

サンプル

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する例を次に示します。

```
options pageno=1 nodate linesize=80 pagesize=60;
data test;
length String $8 Operation $40;
if _n_ = 1 then call compcost('insert=',10,'DEL=',11,'r=', 12);
input String Operation;
GED=compged(string, 'baboon');
datalines;
baboon match
xbaboon insert
babon delete
baXoon replace
;
proc print data=test label;
label GED='Generalized Edit Distance';
var String Operation GED;
run;
```

結果出力を次に示します。

画面 2.1 *演算に基づく一般化編集距離*

The SAS System

Obs	String	Operation	Generalized Edit Distance
1	baboon	match	0
2	xbaboon	insert	10
3	babon	delete	11
4	baXoon	replace	12

関連項目:

関数:

- “COMPGED 関数” (311 ページ)
- “COMPARE 関数” (304 ページ)
- “COMPLEV 関数” (317 ページ)

CALL EXECUTE ルーチン

引数を解決し、実行用に解決された値を次のステップの境界で発行します。

カテゴリ: マクロ

構文

CALL EXECUTE(*argument*);

必須引数

引数

マクロの起動または SAS ステートメントを生成する文字列式または定数を指定します。*Argument* には次を指定できます。

- 引用符で囲まれた文字列。
- DATA ステップの文字変数の名前。DATA ステップの変数は引用符で囲まないでください。
- DATA ステップが解決する文字列式。マクロのテキスト式または SAS ステートメントに解決されます。

詳細

argument がマクロの起動に解決されると、マクロはただちに実行されます。マクロの実行中、DATA ステップの実行は一時停止します。*argument* が SAS ステートメントに解決されるか、マクロの実行によって SAS ステートメントが生成された場合、CALL EXECUTE ルーチンを含む DATA ステップが完了すると、ステートメントが実行されます。CALL EXECUTE の詳細については、*SAS マクロ言語: リファレンス*を参照してください。

CALL GRAYCODE ルーチン

n 個の項目のすべてのサブセットを変化量の小さい順に生成します。

カテゴリ: 組み合わせ関数

構文

CALL GRAYCODE(*k*, *numeric-variable-1*, ..., *numeric-variable-n*);

CALL GRAYCODE(*k*, *character-variable* <, *n*<, *in-out*>>);

必須引数

k
数値変数を指定します。CALL GRAYCODE ルーチンを実行する前に、*k* を次のいずれかの値に初期化します。

- 負の数値。CALL GRAYCODE でサブセットが初期化されて空になります。
- 初期セットの項目数。*numeric-variable-1* から *numeric-variable-n*、または *character-variable* で指定します。0 から N(0 と N を含む)の整数値にする必要があります。

CALL GRAYCODE の実行時に *k* の値が更新されます。サブセットの項目数が値として返されます。

numeric-variable
0 または 1 を値とする数値変数を指定します。CALL GRAYCODE の実行時に更新されます。*numeric-variable-j* の値が 1 の場合、*j*番目の項目がサブセットにあることを表します。*numeric-variable-j* の値が 0 の場合、*j*番目の項目がサブセットにないことを表します。

CALL GRAYCODE を実行する前に負の値を *k* に割り当てる場合は、CALL GRAYCODE を実行する前に *numeric-variable-1* から *numeric-variable-n* を初期化する必要はありません。ただし、非初期化変数に関するメモを非表示にする場合は除きます。

CALL GRAYCODE を実行する前に 0 と *n* の間(0 と n を含む)の値を *k* に割り当てる場合は、CALL GRAYCODE を実行する前に *numeric-variable-1* から *numeric-variable-n* を 1 の *k* の値および 0 の *n-k* の値に初期化する必要があります。

character-variable
長さが少なくとも *n* 文字の文字変数を指定します。先頭の *n* 文字でどの項目がサブセットにあるかを示します。デフォルトでは、*j*番目の位置にある"I"は *j*番目の項目がサブセットにあり、*j*番目の位置にある"O"は *j*番目の項目がサブセットにないことを示します。これらの 2 文字は *in-out* 引数の指定で変更できます。

CALL GRAYCODE を実行する前に負の値を *k* に割り当てる場合は、CALL GRAYCODE を実行する前に *character-variable* を初期化する必要はありません。ただし、非初期化変数に関するメモを非表示にする場合は除きます。

CALL GRAYCODE を実行する前に 0 と *n* の間(0 と n を含む)の値を *k* に割り当てる場合は、CALL GRAYCODE を実行する前に *character-variable* を項目がサブセットにあることを示す *k* 文字と、項目がサブセットないことを示す *k-k* に初期化する必要があります。

オプション引数

n
数値の定数、変数または式を指定します。デフォルトでは、*n* は *character-variable* の長さです。

in-out
文字定数、変数または式を指定します。デフォルト値は"IO"です。1 番目の文字は項目がサブセットにあることを示すのに使用されます。2 番目の文字は項目がサブセットにないことを示すのに使用されます。

詳細

DATA ステップで CALL GRAYCODE を使用する

k に負の値を指定して CALL GRAYCODE ルーチンを実行すると、サブセットは初期化されて空になります。

k に 0 と *n* の間(0 と n を含む)の整数値を指定して CALL GRAYCODE を実行すると、1 つの項目がサブセットに追加されるか、サブセットから削除されて、*k* の値がサブセット内の項目数と同値に更新されます。

n 個の項目のすべてのサブセットを生成するには、*k* を負の値に初期化し、CALL GRAYCODE を 2**n 回、ループで繰り返して実行します。空でないサブセットから開始するには、*k* をサブセット内の項目数に初期化し、その他の引数を希望の初期サブセットを指定するように初期化して、CALL GRAYCODE を 2**n-1 回、ループで繰り返して実行します。CALL GRAYCODE で生成される一連のサブセットは循環するため、どのサブセットで開始してもかまいません。

マクロと CALL GRAYCODE ルーチンを使用する

GRAYCODE ルーチンは、%SYSCALL マクロを使って呼び出せます。CALL GRAYCODE ルーチンを DATA ステップで使用するときと、マクロで使用するときの間にはいくつかの違いがあります。マクロでの使用方法を次に示します。

- すべての引数を空でない値に初期化する必要があります。
- *character-variable* 引数を使用する場合、空ではなく、少なくとも *n* 文字の数値以外の文字列に初期化する必要があります。
- *in-out* 引数を使用する場合は、空白、数値、小数点、プラス記号、マイナス記号以外の 2 文字に初期化する必要があります。

%SYSCALL で引数の種類が誤っていると判別された場合、または引数の種類を判別できない場合は、&SYSERR および&SYSINFO *が設定されません*。

それ以外の場合に CALL GRAYCOD ルーチンの実行中に工ラーが発生すると、次の両方の値が設定されます。

- &SYSERR には 4 よりも大きい値が割り当てられます。
- &SYSINFO には-100 よりも小さい値が割り当てられます。

エラーがない場合、&SYSERR はゼロに設定されて、&SYSINFO は次のいずれかの値に設定されます。

- 0(入力の *k* の値が負の場合)
- サブセットに追加、またはサブセットから削除された項目のインデックス(入力の *k* の値が負ではない有効な整数の場合)

サンプル

サンプル 1: CALL GRAYCODE で文字変数と正の初期値の k を使用する

CALL GRAYCODE ルーチンを使用して変化量の小さい順にサブセットを生成する例を次に示します。

```
data _null_;
x='++++';
n=length(x);
k=countc(x, '+');
put ' 1' +3 k= +2 x=;
nsubs=2**n;
```

```
do i=2 to nsubs;
call graycode(k, x, n, '+-');
put i 5. +3 k= +2 x=;
end;
run;
```

SAS は次の出力を口グに書き込みます。

```
 1 k=4 x=++++
2 k=3 x=-+++
3 k=2 x=-+-+
4 k=3 x=++-+
5 k=2 x=+--+
6 k=1 x=---+
7 k=0 x=----
8 k=1 x=+---
9 k=2 x=++--
10 k=1 x=-+--
11 k=2 x=-++-
12 k=3 x=+++-
13 k=2 x=+-+-
14 k=1 x=--+-
15 k=2 x=--++
16 k=3 x=+-++
```

サンプル 2: 数値変数と k (負) を指定した%SYSCALL を使用する

数値変数を持つ%SYSCALL マク口で変化量の小さい順にサブセットを生成する例を次に示します。

```
%macro test;
%let n=3;
%let x1=.;
%let x2=.;
%let x3=.;
%let k=-1;
%let nsubs=%eval(2**&n + 1);
%put nsubs=&nsubs k=&k x: &x1 &x2 &x3;
%do j=1 %to &nsubs;
%syscall graycode(k, x1, x2, x3);
%put &j: k=&k x: &x1 &x2 &x3 sysinfo=&sysinfo;
%end;
%mend;
%test;
```

SAS は次の出力を口グに書き込みます。

```
nsubs=9 k=-1 x: . . .
1: k=0 x: 0 0 0 sysinfo=0
2: k=1 x: 1 0 0 sysinfo=1
3: k=2 x: 1 1 0 sysinfo=2
4: k=1 x: 0 1 0 sysinfo=1
5: k=2 x: 0 1 1 sysinfo=3
6: k=3 x: 1 1 1 sysinfo=1
7: k=2 x: 1 0 1 sysinfo=2
8: k=1 x: 0 0 1 sysinfo=1
9: k=0 x: 0 0 0 sysinfo=3
```

サンプル 3: 文字変数と k (負) を指定した%SYSCALL を使用する

文字変数を持つ%SYSCALL マクロで変化量の小さい順にサブセットを生成する例を次に示します。

```
%macro test(n);
%*** Initialize the character variable to a
sufficiently long nonblank, nonnumeric value. ;
%let x=%sysfunc(repeat(_, &n-1));
%let k=-1;
%let nsubs=%eval(2**&n + 1);
%put nsubs=&nsubs k=&k x="&x";
%do j=1 %to &nsubs;
%syscall graycode(k, x, n);
%put &j: k=&k x="&x" sysinfo=&sysinfo;
%end;
%mend;
%test(3);
```

SAS は次の出力をログに書き込みます。

```
nsubs=9 k=-1 x="__"
1: k=0 x="OOO" sysinfo=0
2: k=1 x="IOO" sysinfo=1
3: k=2 x="IIO" sysinfo=2
4: k=1 x="OIO" sysinfo=1
5: k=2 x="OII" sysinfo=3
6: k=3 x="III" sysinfo=1
7: k=2 x="IOI" sysinfo=2
8: k=1 x="OOI" sysinfo=1
9: k=0 x="OOO" sysinfo=3
```

関連項目:

関数:

- “GRAYCODE 関数” (512 ページ)

CALL IS8601_CONVERT ルーチン

ISO 8601 規格の間隔を日時値とデュレーション値に変換します。または、日時値とデュレーション値を ISO 8601 規格の間隔に変換します。

カテゴリ: 日付と時間

構文

CALL IS8601_CONVERT(*convert-from*, *convert-to*, <*from-variables*> , <*to-variables*>, <*date-time-replacements*>

必須引数

convert-from
変換元が間隔、日時値とデュレーション値、またはデュレーション値のいずれであるかを表すキーワードを一重引用符で囲んで指定します。*convert-from* は次のいずれかの値を取ることができます。

'intvl' 変換元の値が間隔値であることを指定します。

'dt/du' 変換元の値が日時/デュレーション値であることを指定します。

'du/dt' 変換元の値がデュレーション/日時値であることを指定します。

'dt/dt' 変換元の値が日時/日時値であることを指定します。

'du' 変換元の値がデュレーション値であることを指定します。

convert-to
変換の結果を示すキーワードを一重引用符で囲んで指定します。*convert-to* は次のいずれかの値を取ることができます。

'intvl' 間隔値の作成を指定します。

'dt'/du' 日時/デュレーションの間隔の作成を指定します。

'du/dt' デュレーション/日時の間隔の作成を指定します。

'dt/dt' 日時/日時の間隔の作成を指定します。

'du' デュレーションの作成を指定します。

'start' 間隔値の開始日時またはデュレーションの開始の作成を指定します。

'end' 間隔値の終了日時またはデュレーションの終了の作成を指定します。

オプション引数

from-variable
変換元の値を含む 1 つまたは 2 つの変数を指定します。間隔値の変数を 1 つと、日時値およびデュレーション値に 1 つずつ(計 2 つ)の変数を指定します。日時値とデュレーション値は間隔のコンポーネントで、1 番目の値は間隔の開始値を示し、2 番目の値は間隔の終了値を示します。

要件:
整数変数は、少なくとも 16 バイトの文字変数である必要があります。文字変数の値は$N8601B 入力形式または$N8601E 入力形式で読み取られて判別されます。または、CALL ISO8601_CONVERT ルーチンを起動して返される整数値です。

日時値は、SAS の日時値であるか、$N8601B 入力形式、$N8601E 入力形式、または CALL ISO8601_ CONVERT ルーチンの起動で読み取られる8 バイトの文字値である必要があります。

デュレーション値は、デュレーションを秒数で表す数値であるか、$N8601B 入力形式、$N8601E 入力形式、または CALL ISO8601_ CONVERT ルーチンの起動で値が読み取られて判別される8 バイトの文字値である必要が

to-variable
変換値を含む 1 つまたは 2 つの変数を指定します。間隔値の変数を 1 つと、日時値およびデュレーション値に 1 つずつ(計 2 つ)の変数を指定します。

要件 間隔値は、長さが 16 バイト以上の文字変数である必要があります。

ヒント: 日時変数およびデュレーション変数は数値または文字のいずれかです。数値の精度を維持するには、数値変数の長さは少なくとも 8 文字である必要があります。日時およびデュレーションの文字変数は、少なくとも 16 バイトである必要があります。変数長よりも短い値には空白文字が埋め込まれます。

date-time-replacements

月、日または時間のコンポーネントが間隔値、日時値、デュレーション値で省略されているときに使用する日付または時間のコンポーネント値を指定します。*date-time-replacements* はカンマ区切りで指定する一連の数字で、順に年、月、日、時、分、秒を表します。*date-time-replacements* のコンポーネントは除外できます。ただし、後方にあるコンポーネントから順序に(秒、分、時、日、月)しか除外できません。代入値が指定されていない場合は、デフォルト値を使って変換が行われます。

デフォルト: 除外時の日付コンポーネントおよび時間コンポーネントのデフォルト値は次のとおりです。

1 月
1 日
0 時
0 分
0 秒

要件 年コンポーネントは日時値またはデュレーション値の一部である必要があります。したがって、*date-time-replacements* には使えません。*date-time-replacements* では、年のプレースホルダとしてカンマを使用する必要があります。たとえば、置換値文字列 *,9,4,,2,'* で 1 番目のカンマは年の値のプレースホルダです。

サンプル

この DATA ステップでは、ISO8601_
CONVERT 関数を使って次のタスクを実行します。

- 日時値およびデュレーション値を使った間隔の作成
- CALL
  ISO8601_
  CONVERT ルーチンを使って作成した間隔からの日時値およびデュレーション値の作成
- 日時値で除外された日付コンポーネントおよび時間コンポーネントに置換値を使った、日時値およびデュレーション値からの間隔の作成

読みやすいように、数値変数の末尾は N、文字列変数の末尾は C となっています。

```
data _null_;
/** declare variable length and type **/
/** Character datetime and duration values must be at least **/
/** 16 characters. In order not to lose precision, the **/
/** numeric datetime value has a length of 8. **/
length dtN duN 8 dtC duC $16 intervalC $32;
/** assign a numeric datetime value and a **/
/** character duration value. **/
dtN='15Sep2008:09:00:00'dt;
duC=input('P2y3m4dT5h6m7s', $n8601b.);
put dtN=;
put duC=;
/** Create an interval from a datetime and duration value **/
/** and format it using the ISO 8601 extended notation for **/
/** character values. **/
call is8601_convert('dt/du', 'intvl', dtN, duC, intervalC);
put '** Character interval created from datetime and duration values **/';
put intervalC $n8601e.;
```

```
put ' ';
/** Create numeric datetime and duration values from an interval **/
/** and format it using the ISO 8601 extended notation for **/
/** numeric values. **/

call is8601_convert('intvl', 'dt/du', intervalC, dtN, duN);
put '** Character datetime and duration created from an interval **/';
put dtN=;
put duN=;
put ' ';
/** assign a new datetime value with omitted components **/
dtC=input('2009---15T10:-:-', $n8601b.);
put '** This datetime is a character value. **';
put dtC $n8601h.;
put ' ';
/** Create an interval by reading in a datetime value **/
/** with omitted date and time components. Use replacement **/
/** values for the month, minutes, and seconds. **/

call is8601_convert('du/dt', 'intvl', duC, dtC, intervalC,,7,,,35,45);
put '** Interval created using a datetime with omitted values, **';
put '** inserting replacement values for month (7), minute (35) **';
put '** seconds (45). **';
put intervalC $n8601e.;
put ' ';
run;
```

SAS ログには次の出力が表示されます。

```
dtN=1537088400
duC=0002304050607FFC
** Character interval created from datetime and duration values **/
2008-09-15T09:00:00.000/P2Y3M4DT5H6M7S
** Character datetime and duration created from an interval **/
dtN=1537088400
duN=71211967
** This datetime is a character value. **
2009---15T10:-:-
** Interval created using a datetime with omitted values, **
** inserting replacement values for month (7), minute (35) **
** seconds (45). **
P2Y3M4DT5H6M7S/2009-07-15T10:35:45
NOTE: DATA statement used (Total process time):
real time 0.04 seconds
cpu time 0.03 seconds
```

CALL LABEL ルーチン

指定した文字変数に変数ラベルを割り当てます。

カテゴリ: 変数制御

構文

CALL LABEL(*variable-1*,*variable-2*);

必須引数

variable-1
SAS 変数を指定します。*variable-1* にラベルがない場合は、変数名が *variable-2* の値として割り当てられます。

variable-2
SAS 文字変数を指定します。変数ラベルの最大長は 256 文字です。したがって、変数ラベルが切り捨てられないようにするには、*variable-2* の長さは少なくとも 256 文字とする必要があります。

注: *variable-1* のラベルの長さがわかっている場合は、その長さを *variable-2* に設定すると、スペースを節約できます。

詳細

CALL LABEL ルーチンは *variable-1* 変数のラベルを文字変数 *variable-2* に割り当てます。

サンプル: 例

この例では、CALL LABEL ルーチンに配列参照を使って実行し、データセット OLD のすべての変数のラベルを、データセット NEW の変数 LAB の値として割り当てます。

```
data new;
set old;
/* lab is not in either array */
length lab $256;
/* all character variables in old */
array abc{*} _character_;
/* all numeric variables in old */
array def{*} _numeric_;
do i=1 to dim(abc);
/* get label of character variable */
call label(abc{i},lab);
/* write label to an observation */
output;
end;
do j=1 to dim(def);
/* get label of numeric variable */
call label(def{j},lab);
/* write label to an observation */
output;
end;
stop;
keep lab;
run;
```

関連項目:

関数:

- “VLABEL 関数” (937 ページ)

CALL LEXCOMB ルーチン

n 個の変数を同時に *k* 個使用するときの、重複しない非欠損値のすべての組み合わせを辞書式順序で生成します。

カテゴリ: 組み合わせ関数

操作: %SYSCALL マクロステートメントで呼び出されると、CALL LEXCOMB の引数から引用符が削除されます。詳細については、CALL ルーチンと%SYSCALL マクロステートメントを使用する (9 ページ)を参照してください。

構文

CALL LEXCOMB(*count*, *k*, *variable-1*, …, *variable-n*);

必須引数

count
整数値を指定します。値には 1 ~ 組み合わせ数までがループで割り当てられます。

k
各組み合わせの項目数を示す 1 ~ *n*(1 と n を含む)の整数の定数、変数または式を指定します。

変数
すべての数値変数または同じ長さのすべての文字変数を指定します。これらの変数の値は置換されます。

要件 LEXCOMB ルーチンを呼び出す前にこれらの変数を初期化します。

ヒント: LEXCOMB ルーチンを呼び出すと、最初の *k* 個の変数に 1 つの組み合わせの値が含まれます。

詳細

基本

CALL LEXCOMB ルーチンはループで使用します。CALL LEXCOMB の第 1 引数は 1 から変数の非欠損値の重複しない組み合わせ数までの各整数値を受け付けます。このループから LEXCOMB を呼び出すときは、*k* は毎回同じ値です。

組み合わせ数

すべての変数に等しくない非欠損値が含まれる場合、組み合わせ数は COMB(*n*,*k*) です。欠損値を含む変数の数が *m* 個ですべての非欠損値が等しくない場合、欠損値は組み合わせから除外されるため、LEXCOMB は COMB(*n*-*m*,*k*)個の組み合わせを生成します。

一部の変数に等しい値が含まれる場合、正確な組み合わせ数を計算することは困難です。正確な組み合わせ数を計算できない場合は、CALL LEXCOMB ルーチンのかわりに LEXCOMB 関数を使用します。

CALL LEXCOMB の処理

LEXCOMB ルーチンを初めて呼び出すと、次のアクションが実行されます。

- 引数の種類と長さに不整合がないか確認が行われます。

- *m* の欠損値が最後の *m* 引数に割り当てられます。
- *n-m* の非欠損値が *count* に続く最初の *n-m* 引数に昇順で割り当てられます。

後続の呼び出しでは、最後の組み合わせまで(最後の組み合わせを含む)、非欠損値の重複しない次の組み合わせが辞書式順序で生成されます。

LEXCOMB ルーチンを呼び出す場合に第 1 引数の順序が違っていると、その結果は役に立ちません。特に、変数を初期化した直後に第 1 引数を *j* として LEXCOMB を呼び出すと、*j*番目の組み合わせは取得されません(*j* が 1 である場合を除く)。*j*番目の組み合わせを取得するには、第 1 引数で 1 ~ *j* の値をそのままの順序で取得して、LEXCOMB ルーチンを *j* 回呼び出す必要があります。

マクロと CALL LEXCOMB ルーチンを使用する

LEXCOMB ルーチンは、%SYSCALL マクロを使って呼び出せます。この場合、*変数*の引数を同一の長さにする必要はありませんが、同じ種類にする必要があります。%SYSCALL で引数が数値であることが識別されると、%SYSCALL は戻り値の形式を調整します。

CALL LEXCOMB ルーチンの実行中にエラーが発生すると、次の両方の値が設定されます。

- &SYSERR には 4 よりも大きい値が割り当てられます。
- &SYSINFO には-100 よりも小さい値が割り当てられます。

エラーがない場合、&SYSERR はゼロに設定されて、&SYSINFO は次のいずれかの値に設定されます。

- 1(*count*=1 で、少なくとも 1 つの変数が非欠損値を持つ場合)
- 1(*variable-1* の値が変更された場合)
- *j*(*variable-1* から *variable-i* は変更されず、*j*=*i*+1 で、*variable-j* が変更された場合)
- –1(重複しないすべての組み合わせが生成済みの場合)

比較

CALL LEXCOMB ルーチンは、*n* 個の変数を同時に *k* 個使用するときの、重複しない非欠損値のすべての組み合わせを辞書式順序で生成します。CALL ALLCOMB ルーチンは、*n* 個の変数を同時に *k* 個使用するときのすべての値の組み合わせを、変化量の小さい順に生成します。

サンプル

サンプル 1: DATA ステップで CALL LEXCOMB を使用する

LEXCOMB ルーチンを呼び出して辞書式順序で重複しない組み合わせを生成する例を次に示します。

```
data _null_;
array x[5] $3 ('ant' 'bee' 'cat' 'dog' 'ewe');
n=dim(x);
k=3;
ncomb=comb(n,k);
do j=1 to ncomb;
call lexcomb(j, k, of x[*]);
put j 5. +3 x1-x3;
```

```
end;
run;
```

SAS は次の出力を口グに書き込みます。

```
 1 ant bee cat
2 ant bee dog
3 ant bee ewe
4 ant cat dog
5 ant cat ewe
6 ant dog ewe
7 bee cat dog
8 bee cat ewe
9 bee dog ewe
10 cat dog ewe
```

サンプル 2: マクロと CALL LEXCOMB を使用する

CALL LEXCOMB ルーチンをマクロで使用している例を次に示します。出力には%SYSINFO マクロ用の値が含まれます。

```
%macro test;
%let x1=ant;
%let x2=baboon;
%let x3=baboon;
%let x4=hippopotamus;
%let x5=zebra;
%let k=2;
%let ncomb=%sysfunc(comb(5,&k));
%do j=1 %to &ncomb;
%syscall lexcomb(j, k, x1, x2, x3, x4, x5);
%let jfmt=%qsysfunc(putn(&j, 5. ));
%let pad=%qsysfunc(repeat(%str( ), 20-%length(&x1 &x2)));
%put &jfmt: &x1 &x2 &pad sysinfo=&sysinfo;
%if &sysinfo < 0 %then %let j=%eval(&ncomb+1);
%end;
%mend;
%test
```

SAS は次の出力を口グに書き込みます。

```
 1: ant baboon sysinfo=1
2: ant hippopotamus sysinfo=2
3: ant zebra sysinfo=2
4: baboon baboon sysinfo=1
5: baboon hippopotamus sysinfo=2
6: baboon zebra sysinfo=2
7: hippopotamus zebra sysinfo=1
8: hippopotamus zebra sysinfo=-1
```

関連項目:

関数:

- “LEXCOMB 関数” (607 ページ)

CALL ルーチン:

- “CALL ALLCOMB ルーチン” (148 ページ)

CALL LEXCOMBI ルーチン

n 個のオブジェクトを同時に *k* 個使用するときのインデックスのすべての組み合わせを、辞書式順序で生成します。

カテゴリ: 組み合わせ関数

構文

CALL LEXCOMBI(*n*, *k*, *index-1*, …, *index-k*);

必須引数

n
オブジェクトの合計数を指定する数値の定数、変数または式です。

k
各組み合わせのオブジェクト数を指定する数値の定数、変数または式です。

index
返される組み合わせでオブジェクトのインデックスが入る数値変数です。インデックスは 1 ~ *n*(1 と n を含む)の整数です。

ヒント: *index-1* が欠損またはゼロの場合、CALL LEXCOMBI ルーチンは index-k=k までのインデックスを index-1=1 に初期化します。それ以外の場合は、CALL LEXCOMBI は組み合わせからインデックスの 1 つを削除し、他のインデックスを追加して新しい組み合わせを作ります。

詳細

CALL LEXCOMBI の処理

初めて LEXCOMBI ルーチンを呼び出すときは、先に次のタスクのいずれかを実行します。

- *index-1* をゼロまたは欠損値に設定します。
- *index-1* から *index-k* を 1 ~ *n*(1 と n を含む)の重複しない整数に初期化します。

n 個のオブジェクトを同時に *k* 個使用するときの組み合わせ数は、COMB(*n*,*k*)で計算できます。*n* 個のオブジェクトを同時に *k* 個使用するときのすべての組み合わせを生成するには、COMB(*n*,*k*)回実行するループで LEXCOMBI を呼び出します。

マクロと CALL LEXCOMBI ルーチンを使用する

%SYSCALL を使ったマクロ処理から LEXCOMBI ルーチンを呼び出す場合は、すべての引数を初期化して数値にする必要があります。%SYSCALL は戻り値を再フォーマットします。

CALL LEXCOMBI ルーチンの実行中にエラーが発生すると、次の両方の値が設定されます。

- &SYSERR には 4 よりも大きい値が割り当てられます。
- &SYSINFO には-100 よりも小さい値が割り当てられます。

エラーがない場合、&SYSERR はゼロに設定されて、&SYSINFO は次のいずれかの値に設定されます。

- 1(*variable-1* の値が変更された場合)
- *j*(*variable-1* から *variable-i* は変更されず、*j=i*+1 で、*variable-j* が変更された場合)
- –1(重複しないすべての組み合わせが生成済みの場合)

比較

CALL LEXCOMBI ルーチンは、*n* 個のオブジェクトを同時に *k* 個使用するときのすべてのインデックスの組み合わせを、辞書式順序で生成します。CALL ALLCOMBI ルーチンは、*n* 個のオブジェクトを同時に *k* 個使用するときのすべてのインデックスの組み合わせを、変化量の小さい順に生成します。

サンプル

サンプル 1: DATA ステップで CALL LEXCOMBI ルーチンを使用する

CALL LEXCOMBI ルーチンを使用して辞書式順序で重複しないインデックスの組み合わせを生成する例を次に示します。

```
data _null_;
array x[5] $3 ('ant' 'bee' 'cat' 'dog' 'ewe');
array c[3] $3;
array i[3];
n=dim(x);
k=dim(i);
i[1]=0;
ncomb=comb(n,k);
do j=1 to ncomb;
call lexcombi(n, k, of i[*]);
do h=1 to k;
c[h]=x[i[h]];
end;
put @4 j= @10 'i= ' i[*] +3 'c= ' c[*];
end;
run;
```

SAS は次の出力をログに書き込みます。

```
 j=1 i= 1 2 3 c= ant bee cat
j=2 i= 1 2 4 c= ant bee dog
j=3 i= 1 2 5 c= ant bee ewe
j=4 i= 1 3 4 c= ant cat dog
j=5 i= 1 3 5 c= ant cat ewe
j=6 i= 1 4 5 c= ant dog ewe
j=7 i= 2 3 4 c= bee cat dog
j=8 i= 2 3 5 c= bee cat ewe
j=9 i= 2 4 5 c= bee dog ewe
j=10 i= 3 4 5 c= cat dog ewe
```

サンプル 2: マクロと CALL LEXCOMBI ルーチンを使用し、リターンコードを表示する

次の例では、CALL LEXCOMBI ルーチンをマクロで使用します。出力には%SYSINFO マクロ用の値が含まれます。

```
%macro test;
%let x1=0;
```

```
%let x2=0;
%let x3=0;
%let n=5;
%let k=3;
%let ncomb=%sysfunc(comb(&n,&k));
%do j=1 %to &ncomb+1;
%syscall lexcombi(n,k,x1,x2,x3);
%let jfmt=%qsysfunc(putn(&j,5.));
%let pad=%qsysfunc(repeat(%str(),6-%length(&x1 &x2 &x3)));
%put &jfmt: &x1 &x2 &x3 &pad sysinfo=&sysinfo;
%end;
%mend;
%test
```

SAS は次の出力をログに書き込みます。

```
 1: 1 2 3 sysinfo=1
2: 1 2 4 sysinfo=3
3: 1 2 5 sysinfo=3
4: 1 3 4 sysinfo=2
5: 1 3 5 sysinfo=3
6: 1 4 5 sysinfo=2
7: 2 3 4 sysinfo=1
8: 2 3 5 sysinfo=3
9: 2 4 5 sysinfo=2
10: 3 4 5 sysinfo=1
11: 3 4 5 sysinfo=-1
```

関連項目:

CALL ルーチン:

- “CALL LEXCOMB ルーチン” (174 ページ)
- “CALL ALLCOMBI ルーチン” (151 ページ)

CALL LEXPERK ルーチン

n 個の変数から一度に *k* 個を取り出した非欠損値の重複しないすべての順列を辞書式順序で生成します。

カテゴリ: 組み合わせ関数

操作: %SYSCALL マクロステートメントで呼び出されると、CALL LEXPERK の引数から引用符が削除されます。詳細については、CALL ルーチンと%SYSCALL マクロステートメントを使用する (9 ページ)を参照してください。

構文

CALL LEXPERK(*count*, *k*, *variable-1*, …, *variable-n*);

必須引数

count
ループで 1 ~ 順列数までの値を割り当てる整数変数を指定します。

k
各順列の項目数を指定する、1 ~ *n*(1 と n を含む)の間の整数定数、変数または式を指定します。

変数
すべての数値変数または同じ長さのすべての文字変数を指定します。これらの変数の値は置換されます。

要件 LEXPERK ルーチンを呼び出す前にこれらの変数を初期化します。

ヒント: LEXPERK を呼び出すと、最初の *k* 個の変数に 1 つの順列の値が含まれます。

詳細

基本

CALL LEXPERK ルーチンはループで使用します。CALL LEXPERK の第 1 引数は 1 から変数の非欠損値の重複しない *k* 個の順列数までの各整数を受け付けます。このループ内の LEXPERK の各呼び出しで、*k* の値は同じである必要があります。

順列数

すべての変数に等しくない非欠損値が含まれる場合、順列数は PERM(,*k*)個です。欠損値を含む変数の数が *m* 個ですべての非欠損値が等しくない場合、欠損値は順列から除外されるため、CALL LEXPERK では PERM(*n-m*,*k*)個の順列を生成します。一部の変数に等しい値が含まれる場合、正確な順列数を計算することは困難です。正確な順列数を計算できない場合は、CALL LEXPERK ルーチンのかわりに LEXPERK 関数を使用します。

CALL LEXPERK 処理

LEXPERK ルーチンを初めて呼び出すと、次のアクションが実行されます。

- 引数の種類と長さに不整合がないか確認が行われます。
- *m* の欠損値が最後の *m* 引数に割り当てられます。
- *n-m* の非欠損値が *count* に続く最初の *n-m* 引数に昇順で割り当てられます。

後続の呼び出しでは、最後の順列まで(最後の順列を含む)、*k* 個の非欠損値の次の重複しない順列が辞書式順序で生成されます。

LEXPERK ルーチンを呼び出す場合に第 1 引数が順序が違っていると、その結果は役に立ちません。特に、変数を初期化した直後に第 1 引数が *j* の LEXPERK ルーチンを呼び出した場合、$j^{番目}$の順列は取得されません(*j* が 1 の場合を除く)。$j^{番目}$の順列を取得するには、第 1 引数に 1 ~ *j* の値をそのままの順序で取得して、LEXPERK を *j* 回呼び出す必要があります。

マクロと CALL LEXPERK ルーチンを使用する

%SYSCALL マクロを使用するときに LEXPERK ルーチンを呼び出せます。この場合、*変数*の引数を同一の長さにする必要はありませんが、同じ種類にする必要があります。%SYSCALL で引数が数値であることが識別されると、%SYSCALL は戻り値の形式を調整します。

CALL LEXPERK ルーチンの実行中にエラーが発生すると、次の両方の値が設定されます。

- &SYSERR には 4 よりも大きい値が割り当てられます。
- &SYSINFO には-100 よりも小さい値が割り当てられます。

エラーがない場合、&SYSERR はゼロに設定されて、&SYSINFO は次のいずれかの値に設定されます。

- 1(*count*=1 で 1 つ以上の変数に非欠損値が含まれる場合)
- 1(*count*>1 で *variable-1* の値が変更された場合)
- *j*(*count*>1 で *variable-1* から *variable-i* は変更されず、*variable-j*(*j*=*i*+1)が変更された場合)
- –1(重複しないすべての順列がすでに生成済みの場合)

比較

CALL LEXPERK ルーチンは、*n* 個の変数から一度に *k* 個を取り出した非欠損値の重複しないすべての順列を辞書式順序で生成します。CALL ALLPERM ルーチンは、複数の変数の値のすべての順列を変化量の小さい順に生成します。

サンプル

サンプル 1: DATA ステップで CALL LEXPERK を使用する

CALL LEXPERK ルーチンの例を次に示します。

```
data _null_;
array x[5] $3 ('V' 'W' 'X' 'Y' 'Z');
n=dim(x);
k=3;
nperm=perm(n,k);
do j=1 to nperm;
call lexperk(j, k, of x[*]);
put j 5. +3 x1-x3;
end;
run;
```

SAS は次の出力を口グに書き込みます。

```
 1 V W X
2 V W Y
3 V W Z
4 V X W
5 V X Y
6 V X Z
7 V Y W
8 V Y X
9 V Y Z
10 V Z W
11 V Z X
12 V Z Y
13 W V X
14 W V Y
15 W V Z
16 W X V
17 W X Y
18 W X Z
19 W Y V
20 W Y X
21 W Y Z
22 W Z V
```

```
23 W Z X
24 W Z Y
25 X V W
26 X V Y
27 X V Z
28 X W V
29 X W Y
30 X W Z
31 X Y V
32 X Y W
33 X Y Z
34 X Z V
35 X Z W
36 X Z Y
37 Y V W
38 Y V X
39 Y V Z
40 Y W V
41 Y W X
42 Y W Z
43 Y X V
44 Y X W
45 Y X Z
46 Y Z V
47 Y Z W
48 Y Z X
49 Z V W
50 Z V X
51 Z V Y
52 Z W V
53 Z W X
54 Z W Y
55 Z X V
56 Z X W
57 Z X Y
58 Z Y V
59 Z Y W
60 Z Y X
```

サンプル 2: マクロと CALL LEXPERK を使用する

マクロで使用される CALL LEXPERK ルーチンの例を次に示します。出力には%SYSINFO マクロ用の値が含まれます。

```
%macro test;
%let x1=ant;
%let x2=baboon;
%let x3=baboon;
%let x4=hippopotamus;
%let x5=zebra;
%let k=2;
%let nperk=%sysfunc(perm(5,&k));
%do j=1 %to &nperk;
%syscall lexperk(j, k, x1, x2, x3, x4, x5);
%let jfmt=%qsysfunc(putn(&j,5.));
%let pad=%qsysfunc(repeat(%str(),20-%length(&x1 &x2)));
%put &jfmt: &x1 &x2 &pad sysinfo=&sysinfo;
```

```
%if &sysinfo<0 %then %let j=%eval(&nperk+1);
%end;
%mend;
%test
```

SAS は次の出力をログに書き込みます。

```
 1: ant baboon sysinfo=1
2: ant hippopotamus sysinfo=2
3: ant zebra sysinfo=2
4: baboon ant sysinfo=1
5: baboon baboon sysinfo=2
6: baboon hippopotamus sysinfo=2
7: baboon zebra sysinfo=2
8: hippopotamus ant sysinfo=1
9: hippopotamus baboon sysinfo=2
10: hippopotamus zebra sysinfo=2
11: zebra ant sysinfo=1
12: zebra baboon sysinfo=2
13: zebra hippopotamus sysinfo=2
14: zebra hippopotamus sysinfo=-1
```

関連項目:

関数:

- “LEXPERM 関数” (614 ページ)

CALL ルーチン:

- “CALL ALLPERM ルーチン” (153 ページ)
- “CALL RANPERK ルーチン” (220 ページ)
- “CALL RANPERM ルーチン” (222 ページ)

CALL LEXPERM ルーチン

複数の変数の非欠損値の重複しないすべての順列を辞書式順序で生成します。

カテゴリ: 組み合わせ関数

操作: %SYSCALL マクロステートメントで呼び出されると、CALL LEXPERM の引数から引用符が削除されます。詳細については、CALL ルーチンと%SYSCALL マクロステートメントを使用する (9 ページ)を参照してください。

構文

CALL LEXPERM(*count*, *variable-1* <, …, *variable-N*>);

必須引数

count

1 ~ 順列数の範囲の整数値が含まれる数値変数を指定します。

変数

すべての数値変数または同じ長さのすべての文字変数を指定します。これらの変数の値は、LEXPERM によって置換されます。

要件 LEXPERM ルーチンを呼び出す前にこれらの変数を初期化します。

詳細

重複しない順列数の決定

これらの変数は、その後に示された式で使用するために定義されます。

N
置換される変数の数(引数の数から 1 を引いた数)を指定します。

M
置換される変数間の欠損値の数を指定します。

d
引数間の重複しない非欠損値の数を指定します。

N_i
i=1 ~ i=d までの場合、N_i には i *番目*の重複しない値のインスタンスを指定します。

引数の非欠損値の重複しない順列数は、次のように表されます。

$$P = \frac{(N_1 + N_2 + \ldots + N_d)!}{N_1!\,N_2!\ldots N_d!} <= N!$$

CALL LEXPERM 処理

引数 *count* が 1 ~ P の各整数値を受け入れるループで、CALL LEXPERM ルーチンを使用します。CALL LEXPERM で 0 より小さい値が返されたときにループを終了する場合、P を計算する必要はありません。

1=*count*<P の場合、次のアクションが実行されます。

- 引数の種類と長さに不整合がないか確認が行われます。
- M 個の欠損値が最後の M 個の引数に割り当てられます。
- *count* の後の最初の N-M 個の引数に N-M 個の非欠損値が昇順で割り当てられます。
- CALL LEXPERM は 1 を返します。

1<*count*<=P の場合、次のアクションが実行されます。

- 次の非欠損値の重複しない順列が辞書式順序で生成されます。
- *variable-1* から *variable-I* は変更されず *variable-J* (J=I+1)が変更された場合、CALL LEXPERM は J を返します。

count>P の場合、CALL LEXPERM は–1 を返します。

第 1 引数が順序どおりでない CALL LEXPERM ルーチンが実行された場合、結果は役に立たない可能性があります。特に、変数を初期化した直後に第 1 引数が K の CALL LEXPERM を実行した場合、K *番目*の順列は取得されません(K が 1 の場合を除く)。K *番目*の順列を取得するには、第 1 引数に 1 ~ K の値を正しい順序で渡す CALL LEXPERM を K 回実行する必要があります。

マクロと CALL LEXPERM ルーチンを使用する

%SYSCALL マクロを使用するときに LEXPERM ルーチンを呼び出せます。この場合、*variable* 引数は同じ長さにする必要はありませんが、同じ種類にする必要があります。%SYSCALL で引数が数値であることが識別されると、%SYSCALL は戻り値の形式を調整します。

CALL LEXPERM ルーチンの実行中にエラーが発生すると、次の両方の値が設定されます。

- &SYSERR には 4 よりも大きい値が割り当てられます。
- &SYSINFO には-100 よりも小さい値が割り当てられます。

エラーがない場合、&SYSERR はゼロに設定されて、&SYSINFO は次のいずれかの値に設定されます。

- 1(1=*count*<P の場合)
- 1(1<*count*<=P で *variable-1* の値が変更された場合)
- J(1<*count*<=P で *variable-1* から *variable-I* は変更されず、*variable-J*(J=I+1)が変更された場合)
- -1(*count*>P の場合)

比較

SAS には、すべての順列を生成する 3 つの関数または CALL ルーチンがあります。

- ALLPERM: 複数の変数の値(*欠損値または非欠損値*)の *考えられる*すべての順列を生成します。各順列は、前の順列に基づいて形成されます(2 つの連続する値を交換)。
- LEXPERM: 複数の変数の *非欠損*値の *重複しない*すべての順列を生成します。順列は、辞書式順序で生成されます。
- LEXPERK: N 個の変数の *非欠損*値から K 個の *重複しない*すべての順列を生成します。順列は、辞書式順序で生成されます。

ALLPERM は最も速い関数および CALL ルーチンです。最も遅いのは LEXPERK です。

サンプル

サンプル 1: DATA ステップで CALL LEXPERM を使用する

次の例では、複数の変数の非欠損値の重複しないすべての順列を辞書式順序で生成する DATA ステップを使用します。

```
data _null_;
array x[4] $3 ('ant' 'bee' 'cat' 'dog');
n=dim(x);
nfact=fact(n);
do i=1 to nfact;
call lexperm(i, of x[*]);
put i 5. +2 x[*];
end;
run;
```

SAS は次の出力を口グに書き込みます。

```
 1 ant bee cat dog
2 ant bee dog cat
3 ant cat bee dog
4 ant cat dog bee
5 ant dog bee cat
6 ant dog cat bee
7 bee ant cat dog
8 bee ant dog cat
9 bee cat ant dog
10 bee cat dog ant
11 bee dog ant cat
12 bee dog cat ant
13 cat ant bee dog
14 cat ant dog bee
15 cat bee ant dog
16 cat bee dog ant
17 cat dog ant bee
18 cat dog bee ant
19 dog ant bee cat
20 dog ant cat bee
21 dog bee ant cat
22 dog bee cat ant
23 dog cat ant bee
24 dog cat bee ant
```

サンプル 2: マクロと CALL LEXPERM を使用する

マクロで使用される CALL LEXPERM ルーチンの例を次に示します。出力には%SYSINFO マクロ用の値が含まれます。

```
%macro test;
%let x1=ant;
%let x2=baboon;
%let x3=baboon;
%let x4=hippopotamus;
%let n=4;
%let nperm=%sysfunc(perm(4));
%do j=1 %to &nperm;
%syscall lexperm(j,x1,x2,x3,x4);
%let jfmt=%qsysfunc(putn(&j,5.));
%put &jfmt: &x1 &x2 &x3 &x4 sysinfo=&sysinfo;
%if &sysinfo<0 %then %let j=%eval(&nperm+1);
%end;
%mend;

%test;
```

SAS は次の出力を口グに書き込みます。

```
 1: ant baboon baboon hippopotamus sysinfo=1
2: ant baboon hippopotamus baboon sysinfo=3
3: ant hippopotamus baboon baboon sysinfo=2
4: baboon ant baboon hippopotamus sysinfo=1
5: baboon ant hippopotamus baboon sysinfo=3
6: baboon baboon ant hippopotamus sysinfo=2
7: baboon baboon hippopotamus ant sysinfo=3
8: baboon hippopotamus ant baboon sysinfo=2
9: baboon hippopotamus baboon ant sysinfo=3
```

```
10: hippopotamus ant baboon baboon sysinfo=1
11: hippopotamus baboon ant baboon sysinfo=2
12: hippopotamus baboon baboon ant sysinfo=3
13: hippopotamus baboon baboon ant sysinfo=-1
```

関連項目:

関数:

- “LEXPERM 関数” (614 ページ)
- “LEXPERK 関数” (611 ページ)

CALL ルーチン:

- “CALL ALLPERM ルーチン” (153 ページ)
- “CALL RANPERK ルーチン” (220 ページ)
- “CALL RANPERM ルーチン” (222 ページ)

CALL LOGISTIC ルーチン

ロジスティック関数を各引数に適用します。

カテゴリ: 数学関数

構文

CALL LOGISTIC(*argument*<, *argument*, ...>);

必須引数

引数

数値変数です。

制限事項: CALL LOGISTIC ルーチンで有効な引数は変数のみです。これらの引数は CALL ルーチンで更新できないため、定数または SAS 式は使用しないでください。

詳細

CALL LOGISTIC ルーチンは各引数をその引数の口ジスティック値で置き換えます。たとえば、x_j に置き換えられます。

$$\frac{\varepsilon^{x_j}}{1+\varepsilon^{x_j}}$$

欠損値を含む引数が 1 つでもある場合、CALL LOGISTIC は欠損値をすべての引数で返します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果

x=0.5;
y=-0.5;
call logistic(x,y);
put x= y=;
	x=0.6224593312 y=0.3775406688

CALL MISSING ルーチン

欠損値を指定した文字変数または数値変数に割り当てます。

カテゴリ: 文字関数

構文

CALL MISSING(*varname1*<, *varname2*, ...>);

必須引数

varname
SAS 文字変数または数値変数の名前を指定します。

詳細

CALL MISSING ルーチンは、通常の数値の欠損値(.)を引数リストの各数値変数に割り当てます。

CALL MISSING ルーチンは、文字の欠損値(空白)を引数リストの各文字変数に割り当てます。現在の文字変数の長さが最大長と同じ場合、現在の長さは変更されません。それ以外の場合は、現在の長さが 1 に設定されます。

変数リストには、文字変数および数値変数を混在させることができます。

比較

MISSING 機能は引数に欠損値がないかを確認しますが、引数の値は変更しません。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果

prod='shoes';
invty=7498;
sales=23759;
call missing(sales);
put prod= invty= sales=;
	prod=shoes invty=7498 sales=.

SAS ステートメント	結果

prod='shoes';
invty=7498;
sales=23759;
call missing(prod,invty);
put prod= invty= sales=;
	prod= invty=. sales=23759

prod='shoes';
invty=7498;
sales=23759;
call missing(of _all_);
put prod= invty= sales=;
	prod= invty=. sales=.

関連項目:

関数:

- “MISSING 関数” (645 ページ)
- “DATA ステップでの設定方法” (*SAS 言語リファレンス: 解説編* 5 章)

CALL MODULE ルーチン

外部ルーチンをリターンコードなしで呼び出します。

カテゴリ: 外部ルーチン

構文

CALL MODULE(<*cntl-string*,> *module-name*<,*argument-1, ..., argument-n*>);

必須引数

module-name
使用する外部モジュールの名前です。

オプション引数

cntl-string
コントロール文字列です(省略可能)。最初の文字はアスタリスク(*)で、その後に次の文字の任意の組み合わせを指定する必要があります。

I CALL MODULE ルーチンのすべての引数を 16 進数で表示します。このオプションは、無効な引数や属性テーブルによって発生した問題を診断するために使用できます。I オプションを指定すると、E オプションが暗黙的に指定されます。

E エラーメッセージの詳細を出力します。E オプション(またはこのオプションより優先される I オプション)を使用しない場合、CALL MODULE ルーチンでは“無効な関数の引数”というエラーメッセージのみが生成されます。通常、このメッセージはエラーの原因を特定するのに不十分です。E オプションはプロダクション環境で役立ち、I オプションは開発環境またはデバッグ環境で役立ちます。

H CALL MODULE ルーチンの構文、属性ファイル形式および推奨される SAS の出力形式と入力形式に関する、簡単なヘルプ情報を表示します。

引数

要求されたルーチンに渡す 1 つ以上の引数です。

注意:

正しい引数と属性を使用してください。 無効な引数または属性を使用した場合、SAS システムだけでなくオペレーティングシステムにも障害が発生する可能性があります。

詳細

CALL MODULE ルーチンは、指定した引数を使用して外部ライブラリ内にあるルーチン *module-name* を実行します。

CALL MODULE は、引数の情報および個別のファイルで定義したルーチンの説明と引数の属性テーブルを使用してパラメータリストを作成します。属性テーブルはシーケンシャルテキストファイルで、CALL MODULE ルーチンで呼び出せるルーチンの説明が含まれます。このテーブルは、CALL MODULE がパラメータリストを作成して外部ルーチンに渡すときに、指定された引数を解釈する方法を定義することを目的としています。属性テーブルには、呼び出す各外部ルーチンの説明と、そのルーチンに関連付けられた各引数の説明が含まれる必要があります。

CALL MODULE を呼び出す前に、属性テーブルを含む外部ファイルを示すように SASCBTBL のファイル参照名を定義する必要があります。ファイルの作成時にファイルに任意の名前を付けることができます。これにより、SAS の変数と出力形式を CALL MODULE の引数として使用し、これらの引数が外部ルーチンに渡される前に適切に変換されます。このファイル参照名を定義しない場合、CALL MODULE は引数を変更せずに要求されたルーチンを呼び出します。

注意:

定義された属性テーブルなしで CALL MODULE ルーチンを使用した場合、SAS システムに障害が発生したり、コンピュータのリセットが必要になる可能性があります。 呼び出すすべての外部関数に対して属性テーブルを使用する必要があります。

比較

2 つの CALL ルーチンと 4 つの関数で同一の構文が共有されます。

- MODULEN および MODULEC 関数はそれぞれ数値と文字を返し、ルーチン CALL MODULE は値を返しません。
- CALL MODULEI ルーチンと関数 MODULEIC および MODULEIN では、ベクトルおよび行列引数が許可されます。これらはスカラ値を返します。CALL MODULEI、MODULEIC、MODULEIN は、IML プロシジャのみから呼び出せます。

サンプル

サンプル 1: CALL MODULE ルーチンを使用する

この例では、**xyz** ルーチンを呼び出します。次の属性テーブルを使用します。

```
routine xyz minarg=2 maxarg=2;
arg 1 input num byvalue format=ib4.;
arg 2 output char format=$char10.;
```

xyz 関数を呼び出すサンプル SAS コードを次に示します。

```
data _null_;
call module('xyz',1,x);
run;
```

サンプル 2: IML プロシジャで MODULEIN 関数を使用する

この例では、Windows プラットフォームで TRYMOD.DLL モジュールから **changi** ルーチンを呼び出します。次の属性テーブルを使用します。

```
routine changi module=trymod returns=long;
arg 1 input num format=ib4. byvalue;
arg 2 update num format=ib4.;
```

次の PROC IML コードは **changi** 関数を呼び出します。

```
proc iml;
x1=J(4,5,0);
do i=1 to 4;
do j=1 to 5;
x1[i,j]=i*10+j+3;
end;
end;
y1=x1;
x2=x1;
y2=y1;
rc=modulein('*i','changi',6,x2);
```

サンプル 3: MODULEN 関数を使用する

この例では、**Beep** ルーチンを呼び出します。このルーチンは、Windows プラットフォームでの KERNEL32 ダイナミックリンクライブラリの Win32 API に含まれています。次の属性テーブルを使用します。

```
routine Beep
minarg=2
maxarg=2
stackpop=called
callseq=byvalue
module=kernel32;
arg 1 num format=pib4.;
arg 2 num format=pib4.;
```

属性テーブルファイルに'myatttbl.dat'という名前を付けるとします。**Beep** 関数を呼び出すサンプル SAS コードを次に示します。

```
filename sascbtbl 'myatttbl.dat';
data _null_;
rc=modulen("*e","Beep",1380,1000);
run;
```

前のコードによって、コンピュータのスピーカーからビープ音がなります。

関連項目:

関数:

- “MODULEC 関数” (649 ページ)
- “MODULEN 関数” (650 ページ)

CALL POKE ルーチン

32 ビットプラットフォームのメモリに値を直接書き込みます。

カテゴリ: 特殊関数

制限事項: 32 ビットプラットフォームでのみ使用します。

構文

CALL POKE(*source*,*pointer*<,*length*><,*floating-point*>);

必須引数

ソース
メモリに書き込む値が含まれる定数、変数または式を指定します。

pointer
CALL POKE ルーチンで変更するデータの仮想アドレスが含まれる数値式を指定します。

オプション引数

長さ
source から *pointer* によって示されたアドレスに書き込むバイト数が含まれる数値の定数、変数または式を指定します。*length* を省略した場合、CALL POKE ルーチンで実行されるアクションは *source* が文字値か数値かによって異なります。

- *source* が文字値の場合、CALL POKE ルーチンは指定されたメモリの場所に *source* の値全体をコピーします。
- *source* が数値の場合、CALL POKE ルーチンは *source* を長整数型に変換し、ポインタを構成するバイト数をメモリに書き込みます。

z/OS 固有
z/OS では、ポインタの長さは状況に応じて 3 または 4 バイトです。

floating-point
source の値を浮動小数点数として保存するように指定します。*floating-point* の値はどのような数値でもかまいません。

ヒント: *floating-point* 引数を使用しない場合、*source* は整数値として保存されます。

詳細

注意:
CALL POKE ルーチンは、特定のケースで経験豊富なプログラマのみが使用するために用意されています。 このルーチンを使用する場合は、プログラミングと入力に細心の注意を払ってください。メモリに直接書き込むと、重大な問題が発生する可能性があります。このルーチンは、その時点でアクティブな SAS セッションまたはソフトウェアの別の部分で不可欠な要素が破壊されることを防ぐ通常の保護を省略します。

指定したメモリの場所にアクセスできない場合、CALL POKE ルーチンは"無効な引数"エラーを返します。

64 ビットのプラットフォームでは CALL POKE ルーチンは使用できません。使用しようとすると、この制限が適用されることを示すメッセージが SAS によってログに書き込まれます。CALL POKE を使用する従来のアプリケーションがある場合、かわりに CALL POKELONG を使用するようにアプリケーションを変更します。CALL POKELONG は 32 ビットと 64 ビット両方のプラットフォームで使用できます。

第 4 引数を使用した場合、値は浮動小数点数として保存されます。第 4 引数を使用しない場合、整数値として保存されます。

関連項目:

関数:

- “ADDR 関数” (91 ページ)
- “PEEK 関数” (718 ページ)
- “PEEKC 関数” (719 ページ)

CALL ルーチン:

- “CALL POKELONG ルーチン” (193 ページ)

CALL POKELONG ルーチン

32 ビットおよび 64 ビットのプラットフォームのメモリに値を直接書き込みます。

カテゴリ: 特殊関数

構文

CALL POKELONG(*source*,*pointer*<,*length*><,*floating-point*>)

必須引数

ソース
メモリに書き込む値が含まれる文字定数、変数または式を指定します。

pointer
CALL POKELONG ルーチンで変更するデータの仮想アドレスが含まれる文字列を指定します。

オプション引数

長さ
source から *pointer* によって示されたアドレスに書き込むバイト数が含まれる数値の SAS 式を指定します。*length* を省略した場合、CALL POKELONG ルーチンは指定されたメモリの場所に *source* の値全体をコピーします。

floating-point
source の値を浮動小数点数として保存するように指定します。*floating-point* の値はどのような数値でもかまいません。

ヒント: *floating-point* 引数を使用しない場合、*source* は整数値として保存されます。

詳細

注意:

CALL POKELONG ルーチンは、特定のケースで経験豊富なプログラマのみが使用するために用意されています。 このルーチンを使用する場合は、プログラミングと入力に細心の注意を払ってください。*メモリに直接書き込むと、重大な問題が発生する可能性があります。* このルーチンは、その時点でアクティブな SAS セッションまたはソフトウェアの別の部分で不可欠な要素が破壊されることを防ぐ通常の保護を省略します。

指定したメモリの場所にアクセスできない場合、CALL POKELONG ルーチンは"無効な引数"エラーを返します。

第 4 引数を使用した場合、値は浮動小数点数として保存されます。第 4 引数を使用しない場合、整数値として保存されます。

関連項目:

CALL ルーチン:

- “CALL POKE ルーチン” (192 ページ)

CALL PRXCHANGE ルーチン

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: PRXPARSE 関数とともに使用します。

操作: %SYSCALL マクロステートメントで呼び出されると、CALL PRXCHANGE の引数から引用符が削除されます。詳細については、CALL ルーチンと%SYSCALL マクロステートメントを使用する (9 ページ)を参照してください。

構文

CALL PRXCHANGE (*regular-expression-id*, *times*, *old-string* <, *new-string*<, *result-length* <, *truncation-value*<, *number-of-changes*>>>>);

必須引数

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。

時間
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。

ヒント: *times* の値が-1 の場合、一致するすべてのパターンが置換されます。

old-string
検索と置換を実行する文字式を指定します。

ヒント: *new-string* 引数を使用しない場合、*old-string* にすべての変更が加えられます。

オプション引数

new-string
old-string への変更の結果を挿入する文字変数を指定します。

ヒント: PRXCHANGE ルーチンの呼び出しで *new-string* 引数を使用した場合、*old-string* は変更されません。

result-length
結果にコピーされる文字数の戻り値が含まれる数値変数です。

ヒント: *old-string* の値の末尾の空白は *new-string* にコピーされないため、*result-length* の長さには含まれません。

truncation-value
変更処理の結果に応じて、0 または 1 の戻り値が含まれる数値変数です。

0 置換の結果全体の長さが *new-string* の長さ以下の場合。

1 置換の結果全体の長さが *new-string* の長さを超える場合。

number-of-changes
実行された置換の合計数の戻り値が含まれる数値変数です。*new-string* に挿入されるときに結果が切り捨てられた場合、*number-of-changes* の値は変更されません。

詳細

CALL PRXCHANGE ルーチンはパターンを照合して置換します。*times* の値が-1 の場合、可能な限り多くの回数の置換が実行されます。

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 (42 ページ)を参照してください。

比較

CALL PRXCHANGE ルーチンは PRXCHANGE 関数と似ていますが、CALL ルーチンはパターンマッチングの値を引数の戻り値としてではなくそのパラメータの 1 つとして返す点が異なります。

Perl 正規表現(PRX)の関数と CALL ルーチンは、連携してパターンと一致する文字列を操作します。これらの関数および CALL ルーチンのリストと概要説明については、“カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の文字列マッチングカテゴリを参照してください。

サンプル

次の例では、すべての cat、rat、bat を値 TREE で置き換えます。

```
data _null_;
/* Use a pattern to replace all occurrences of cat, */
/* rat, or bat with the value TREE. */
length text $ 46;
RegularExpressionId = prxparse('s/[crb]at/tree/');
text = 'The woods have a bat, cat, bat, and a rat!';
/* Use CALL PRXCHANGE to perform the search and replace. */
/* Because the argument times has a value of -1, the */
/* replacement is performed as many times as possible. */
```

```
call prxchange(RegularExpressionId, -1, text);
put text;
run;
```

SAS は次の行をログに書き込みます。

```
The woods have a tree, tree, tree, and a tree!
```

関連項目:

関数:

- “PRXCHANGE 関数” (754 ページ)
- “PRXPAREN 関数” (763 ページ)
- “PRXMATCH 関数” (759 ページ)
- “PRXPARSE 関数” (765 ページ)
- “PRXPOSN 関数” (767 ページ)

CALL ルーチン:

- “CALL PRXDEBUG ルーチン” (196 ページ)
- “CALL PRXFREE ルーチン” (198 ページ)
- “CALL PRXNEXT ルーチン” (199 ページ)
- “CALL PRXPOSN ルーチン” (201 ページ)
- “CALL PRXSUBSTR ルーチン” (204 ページ)

CALL PRXDEBUG ルーチン

DATA ステップで Perl 正規表現を有効にし、デバッグ出力を SAS ログに送信します。

カテゴリ: 文字列マッチング

制限事項: CALL PRXCHANGE、CALL PRXFREE、CALL PRXNEXT、CALL PRXPOSN、CALL PRXSUBSTR、PRXPARSE、PRXPAREN、PRXMATCH 関数および CALL ルーチンとともに使用します。PRXPARSE 関数は DBCS とは互換性がありません。

構文

CALL PRXDEBUG (*on-off*);

必須引数

on-off
数値の定数、変数または式を指定します。*on-off* の値が 0 ではなく正の値の場合、デバッグはオンになります。*on-off* の値が 0 の場合、デバッグはオフになります。

詳細

CALL PRXDEBUG ルーチンは、Perl 正規表現のコンパイル方法、およびパターンが文字値に一致した場合に実行する操作に関する情報を提供します。

特定の Perl 正規表現関数呼び出しのデバッグ出力を確認する場合、プログラム内で複数回デバッグをオンおよびオフにできます。

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 (42 ページ)を参照してください。

比較

Perl 正規表現(PRX)の関数と CALL ルーチンは、連携してパターンと一致する文字列を操作します。これらの関数および CALL ルーチンのリストと概要説明については、“カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の文字列マッチングカテゴリを参照してください。

サンプル

次の例では、デバッグ出力を生成します。

```
data _null_;
/* Turn the debugging option on. */
call prxdebug(1);
putlog 'PRXPARSE: ';
re = prxparse('/[bc]d(ef*g)+h[ij]k$/');
putlog 'PRXMATCH: ';
pos = prxmatch(re, 'abcdefg_gh_');
/* Turn the debugging option off. */
call prxdebug(0);
run;
```

次の行が SAS ログに書き込まれます。

ログ 2.1 *CALL PRXDEBUG による SAS ログの結果*

```
PRXPARSE: Compiling REx '[bc]d(ef*g)+h[ij]k$' 1 size 41 first at 1 2 rarest char g at 0 5 rarest char d at 0 1:
ANYOF[bc](10) 3 10: EXACT <d>(12) 12: CURLYX[0] {1,32767}(26) 14: OPEN1(16) 16: EXACT <e>(18) 18:
STAR(21) 19: EXACT <f>(0) 21: EXACT <g>(23) 23: CLOSE1(25) 25: WHILEM[1/1](0) 26: NOTHING(27) 27: EXACT
<h>(29) 29: ANYOF[ij](38) 38: EXACT <k>(40) 40: EOL(41) 41: END(0) anchored 'de' at 1 floating 'gh' at
3..2147483647 (checking floating) 4 stclass 'ANYOF[bc]' minlen 7 6 PRXMATCH: Guessing start of match, REx
'[bc]d(ef*g)+h[ij]k$' against 'abcdefg_gh_'... Did not find floating substr 'gh'... Match rejected by optimizer
```

次の項目は、前述の SAS ログで番号が付けられた行に対応します。

1 この行には Perl 正規表現のプリコンパイル済みの形式が表示されます。

2 size は Perl 正規表現のコンパイル済み形式の値(任意の単位)を示します。41 は照合する最初のノードのラベル ID です。

3 この行から正規表現のコンパイル済み形式でのプログラムノードのリストが開始します。

4 これら 2 つの行にはオプティマイザ情報が表示されます。上述の例では、オフセット 1 では部分文字列 **de**、オフセット 3 から無限大までは部分文字列 **gh** が一致に含まれる必要があることをオプティマイザが検出しています。パターンの一致をすばやく除外するため、Perl は部分文字列 **de** を確認する前に部分文字列 **gh** を確認します。

   オプティマイザは、文字クラス(行 5)、7 文字以上(行 6)の一致が *first* ID (行 2)で開始する情報を使用する可能性があります。

関連項目:

関数:

- “PRXCHANGE 関数” (754 ページ)
- “PRXPAREN 関数” (763 ページ)
- “PRXMATCH 関数” (759 ページ)
- “PRXPARSE 関数” (765 ページ)
- “PRXPOSN 関数” (767 ページ)

CALL ルーチン:

- “CALL PRXCHANGE ルーチン” (194 ページ)
- “CALL PRXFREE ルーチン” (198 ページ)
- “CALL PRXNEXT ルーチン” (199 ページ)
- “CALL PRXPOSN ルーチン” (201 ページ)
- “CALL PRXSUBSTR ルーチン” (204 ページ)

CALL PRXFREE ルーチン

Perl 正規表現に割り当てられたメモリを解放します。

カテゴリ: 文字列マッチング

制限事項: PRXPARSE 関数とともに使用します。

構文

CALL PRXFREE (*regular-expression-id*);

必須引数

regular-expression-id
PRXPARSE 関数によって返される ID 番号の値が含まれる数値変数を指定します。PRXFREE ルーチンの呼び出しが エラーなしで行われた場合、*regular-expression-id* は欠損値として設定されます。

詳細

CALL PRXFREE ルーチンは、Perl 正規表現に割り当てられた不要なリソースを解放します。

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 (42 ページ)を参照してください。

比較

Perl 正規表現(PRX)の関数と CALL ルーチンは、連携してパターンと一致する文字列を操作します。これらの関数および CALL ルーチンのリストと概要説明については、“カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の文字列マッチングカテゴリを参照してください。

関連項目:

関数:

- “PRXCHANGE 関数” (754 ページ)
- “PRXPAREN 関数” (763 ページ)
- “PRXPAREN 関数” (763 ページ)
- “PRXPARSE 関数” (765 ページ)
- “PRXPOSN 関数” (767 ページ)

CALL ルーチン:

- “CALL PRXCHANGE ルーチン” (194 ページ)
- “CALL PRXDEBUG ルーチン” (196 ページ)
- “CALL PRXNEXT ルーチン” (199 ページ)
- “CALL PRXPOSN ルーチン” (201 ページ)
- “CALL PRXSUBSTR ルーチン” (204 ページ)
- “CALL PRXCHANGE ルーチン” (194 ページ)

CALL PRXNEXT ルーチン

パターンに一致し、1 つの文字列内で複数の一致が繰り返される部分文字列の位置と長さを返します。

カテゴリ: 文字列マッチング

制限事項: PRXPARSE 関数とともに使用します。

操作: %SYSCALL マクロステートメントで呼び出されると、CALL PRXNEXT の引数から引用符が削除されます。詳細については、CALL ルーチンと%SYSCALL マクロステートメントを使用する (9 ページ)を参照してください。

構文

CALL PRXNEXT (*regular-expression-id*, *start*, *stop*, *source*, *position*, *length*);

必須引数

regular-expression-id
PRXPARSE 関数によって返される ID 番号の値が含まれる数値変数を指定します。

start
source でパターンマッチングを開始する位置を指定する数値変数です。一致に成功した場合、CALL PRXNEXT は *position* + MAX(1, *length*)の値を返します。一致に成功しなかった場合、*start* の値は変更されません。

停止
source で使用する最後の文字を指定する数値の定数、変数または式です。*stop* が-1 の場合、最後の文字は *source* の最後の空白値以外の文字です。

ソース
検索する文字定数、変数または式を指定します。

position
パターンが開始される *source* の位置の戻り値が含まれる数値変数です。一致が見つからない場合、CALL PRXNEXT は 0 を返します。

長さ
パターンに一致する文字列の長さの戻り値が含まれる数値変数です。一致が見つからない場合、CALL PRXNEXT は 0 を返します。

詳細

CALL PRXNEXT ルーチンは、パターンを使用して変数 *source* を検索します。*source* 内の *start* から *stop* の間にあるパターンの一致の位置と長さを返します。*start* パラメータの値は一致に続く次の文字の位置に更新されるため、CALL PRXNEXT では連続して複数回パターンに一致する文字列を検索できます。

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 (42 ページ)を参照してください。

比較

Perl 正規表現(PRX)の関数と CALL ルーチンは、連携してパターンと一致する文字列を操作します。これらの関数および CALL ルーチンのリストと概要説明については、“カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の文字列マッチングカテゴリを参照してください。

サンプル

次の例では、テキスト文字列内からすべての cat、rat、bat を検索します。

```
data _null_;
ExpressionID = prxparse('/[crb]at/');
text = 'The woods have a bat, cat, and a rat!';
start = 1;
stop = length(text);
/* Use PRXNEXT to find the first instance of the pattern, */
/* then use DO WHILE to find all further instances. */
/* PRXNEXT changes the start parameter so that searching */
/* begins again after the last match. */
call prxnext(ExpressionID, start, stop, text, position, length);
do while (position > 0);
found = substr(text, position, length);
put found= position= length=;
call prxnext(ExpressionID, start, stop, text, position, length);
end;
run;
```

次の行が SAS ログに書き込まれます。

```
 found=bat position=18 length=3
found=cat position=23 length=3
found=rat position=34 length=3
```

関連項目:

関数:

- “PRXCHANGE 関数” (754 ページ)
- “PRXPAREN 関数” (763 ページ)

- “PRXMATCH 関数” (759 ページ)
- “PRXPARSE 関数” (765 ページ)
- “PRXPOSN 関数” (767 ページ)

CALL ルーチン:

- “CALL PRXCHANGE ルーチン” (194 ページ)
- “CALL PRXDEBUG ルーチン” (196 ページ)
- “CALL PRXFREE ルーチン” (198 ページ)
- “CALL PRXPOSN ルーチン” (201 ページ)
- “CALL PRXSUBSTR ルーチン” (204 ページ)

CALL PRXPOSN ルーチン

キャプチャバッファの開始位置と長さを返します。

カテゴリ: 文字列マッチング

制限事項: PRXPARSE 関数とともに使用します。

構文

CALL PRXPOSN (*regular-expression-id*, *capture-buffer*, *start* <, *length*>);

必須引数

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。

capture-buffer
開始位置と長さを取得するキャプチャバッファを識別する値が含まれる数値の定数、変数または式です。

- *capture-buffer* の値が 0 の場合、CALL PRXPOSN は一致全体の開始位置と長さを返します。
- *capture-buffer* の値が 1 ~ 開始かっこ数の間の場合、CALL PRXPOSN はそのキャプチャバッファの開始位置と長さを返します。
- *capture-buffer* の値が開始かっこ数よりも大きい場合、CALL PRXPOSN は開始位置と長さとして欠損値を返します。

start
キャプチャバッファが見つかった位置の戻り値が含まれる数値変数です。

- *capture-buffer* の値が見つからない場合、CALL PRXPOSN は開始位置としてゼロ値を返します。
- *capture-buffer* の値がパターン内の開始かっこ数よりも大きい場合、CALL PRXPOSN は開始位置として欠損値を返します。

オプション引数

長さ

前の一致したパターンの長さの戻り値が含まれる数値変数です。

- パターンの一致が見つからない場合、CALL PRXPOSN は長さとしてゼロ値を返します。
- *capture-buffer* の値がパターン内の開始かっこ数よりも大きい場合、CALL PRXPOSN は長さとして欠損値を返します。

詳細

CALL PRXPOSN ルーチンは、PRXMATCH、PRXSUBSTR、PRXCHANGE、PRXNEXT のいずれかの結果を使用してキャプチャバッファを返します。CALL PRXPOSN ルーチン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。

キャプチャバッファは一致の一部としてかっこで囲まれ、正規表現で指定されます。CALL PRXPOSN はキャプチャバッファのテキストを直接返しません。テキストを返すには、SUBSTR 関数を呼び出す必要があります。

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 (42 ページ)を参照してください。

比較

CALL PRXPOSN ルーチンは PRXPOSN 関数と似ていますが、CALL PRXPOSN はキャプチャバッファ自体ではなくキャプチャバッファの位置と長さを返す点が異なります。

Perl 正規表現(PRX)の関数と CALL ルーチンは、連携してパターンと一致する文字列を操作します。これらの関数および CALL ルーチンのリストと概要説明については、“カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の文字列マッチングカテゴリを参照してください。

サンプル

サンプル 1: 一致内の部分一致の検索

次の例では、正規表現を検索し、PRXPOSN ルーチンを呼び出して 3 つの部分一致の位置と長さを特定します。

```
data _null_;
patternID = prxparse('/(¥d¥d):(¥d¥d)(am|pm)/');
text = 'The time is 09:56am.';
if prxmatch(patternID, text) then do;
call prxposn(patternID, 1, position, length);
hour = substr(text, position, length);
call prxposn(patternID, 2, position, length);
minute = substr(text, position, length);
call prxposn(patternID, 3, position, length);
ampm = substr(text, position, length);
put hour= minute= ampm=;
put text=;
end;
run;
```

SAS は次の行をログに書き込みます。

```
hour=09 minute=56 ampm=am
text=The time is 09:56am.
```

サンプル 2: 時間データの解析

次の例では、時間データを解析して結果を SAS ログに書き込みます。

```
data _null_;
if _N_ = 1 then
do;
retain patternID;
pattern = "/(¥d+):(¥d¥d)(?:¥.(¥d+))?/";
patternID = prxparse(pattern);
end;

array match[3] $ 8;
input minsec $80.;
position = prxmatch(patternID, minsec);
if position ^= 0 then
do;
do i = 1 to prxparen(patternID);
call prxposn(patternID, i, start, length);
if start ^= 0 then
match[i] = substr(minsec, start, length);
end;
put match[1] "minutes, " match[2] "seconds" @;
if ^missing(match[3]) then
put ", " match[3] "milliseconds";
end;
datalines;
14:56.456
45:32
;
```

SAS は次の行をログに書き込みます。

```
 14 minutes, 56 seconds, 456 milliseconds
45 minutes, 32 seconds
```

関連項目:

関数:

- “PRXCHANGE 関数” (754 ページ)
- “PRXPAREN 関数” (763 ページ)
- “PRXMATCH 関数” (759 ページ)
- “PRXPARSE 関数” (765 ページ)
- “PRXPOSN 関数” (767 ページ)

CALL ルーチン:

- “CALL PRXCHANGE ルーチン” (194 ページ)
- “CALL PRXDEBUG ルーチン” (196 ページ)
- “CALL PRXFREE ルーチン” (198 ページ)

- “CALL PRXNEXT ルーチン” (199 ページ)
- “CALL PRXSUBSTR ルーチン” (204 ページ)

CALL PRXSUBSTR ルーチン

パターンに一致する部分文字列の位置と長さを返します。

カテゴリ: 文字列マッチング

制限事項: PRXPARSE 関数とともに使用します。

操作: %SYSCALL マクロステートメントで呼び出されると、CALL PRXSUBSTR の引数から引用符が削除されます。詳細については、CALL ルーチンと%SYSCALL マクロステートメントを使用する (9 ページ)を参照してください。

構文

CALL PRXSUBSTR (*regular-expression-id*, *source*, *position* <, *length*>);

必須引数

regular-expression-id
PRXPARSE 関数によって返される ID 番号の値が含まれる数値変数を指定します。

ソース
検索する文字定数、変数または式を指定します。

position
パターンが開始される *source* の位置の戻り値が含まれる数値変数です。一致が見つからない場合、CALL PRXSUBSTR は 0 を返します。

オプション引数

長さ
パターンに一致する部分文字列の長さの戻り値が含まれる数値変数です。一致が見つからない場合、CALL PRXSUBSTR は 0 を返します。

詳細

CALL PRXSUBSTR ルーチンは PRXPARSE からパターンを使用して変数 *source* を検索し、文字列の開始位置を返します。指定されている場合は一致する文字列の長さも返します。デフォルトでは、特定の位置から始まる複数の文字にパターンが一致する場合、CALL PRXSUBSTR では最も長い一致が選択されます。

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 (42 ページ)を参照してください。

比較

CALL PRXSUBSTR は、PRXMATCH と同じマッチングを実行しますが、さらに、CALL PRXSUBSTR では *length* 引数を使用して、一致に関するより多くの情報を取得できます。

Perl 正規表現(PRX)の関数と CALL ルーチンは、連携してパターンと一致する文字列を操作します。これらの関数および CALL ルーチンのリストと概要説明に

ついては、“カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の文字列マッチングカテゴリを参照してください。

サンプル

サンプル 1: 部分文字列の位置と長さの検索

次の例では、文字列から部分文字列を検索し、文字列内の部分文字列の位置と長さを返します。

```
data _null_;
/* Use PRXPARSE to compile the Perl regular expression. */
patternID = prxparse('/world/');
/* Use PRXSUBSTR to find the position and length of the string. */
call prxsubstr(patternID, 'Hello world!', position, length);
put position= length=;
run;
```

次の行が SAS ログに書き込まれます。

```
position=7 length=5
```

サンプル 2: 部分文字列内の一致の検索

次の例では、avenue、drive、road を含む住所を検索し、見つかったテキストを抽出します。

```
data _null_;
if _N_ = 1 then
do;
retain ExpressionID;
/* The i option specifies a case insensitive search. */
pattern = "/ave|avenue|dr|drive|rd|road/i";
ExpressionID = prxparse(pattern);
end;
input street $80.;
call prxsubstr(ExpressionID, street, position, length);
if position ^= 0 then
do;
match = substr(street, position, length);
put match:$QUOTE. "found in " street:$QUOTE.;
end;
datalines;
153 First Street
6789 64th Ave
4 Moritz Road
7493 Wilkes Place
;
run;
```

次の行が SAS ログに書き込まれます。

```
"Ave" found in "6789 64th Ave"
"Road" found in "4 Moritz Road"
```

関連項目:

関数:

- “PRXCHANGE 関数” (754 ページ)
- “PRXPAREN 関数” (763 ページ)
- “PRXMATCH 関数” (759 ページ)
- “PRXPARSE 関数” (765 ページ)
- “PRXPOSN 関数” (767 ページ)

CALL ルーチン:

- “CALL PRXCHANGE ルーチン” (194 ページ)
- “CALL PRXDEBUG ルーチン” (196 ページ)
- “CALL PRXFREE ルーチン” (198 ページ)
- “CALL PRXNEXT ルーチン” (199 ページ)
- “CALL PRXPOSN ルーチン” (201 ページ)

CALL RANBIN ルーチン

二項分布からランダム変量を返します。

カテゴリ: 乱数

構文

CALL RANBIN(*seed*,*n*,*p*,*x*);

必須引数

シード

シード値です。CALL RANBIN が実行されるたびに、*seed* の新しい値が返されます。

範囲: $seed < 2^{31} - 1$

注: $seed \leq 0$ の場合、シードストリームの初期化に時刻が使用されます。

参照項目: シード値の詳細については、シード値 (11 ページ)および乱数関数と乱数 CALL ルーチンのシード値の比較 (15 ページ)を参照

n

整数の独立した Bernoulli 試行数です。

範囲: $n > 0$

p

数値の成功率パラメータです。

範囲: $0<p<1$

x

数値の SAS 変数です。CALL RANBIN が実行されるたびに、ランダム変量 *x* の新しい値が返されます。

詳細

CALL RANBIN ルーチンは *seed* を更新し、平均が *np* で分散が *np*(1–*p*)の二項分布から生成された変量 *x* を返します。$n \leq 50$、$np \leq 5$、$n(1-p) \leq 5$ の場合、SAS

は RANUNI 一様変量に適用される逆変換手法を使用します。*n*>50、*np*>5、*n(1–p)*>5 の場合、SAS は二項分布の正規近似を使用します。この場合、RANUNI 一様変量の Box-Muller 変換が使用されます。

シードを調整することで、同一または後続の DATA ステップの一部またはすべてのオブザベーションで変量のストリームへの一致または不一致を強制できます。

乱数 CALL ルーチンの効果的な使用方法に関する説明と例については、ストリームの開始、停止および再開 (26 ページ)を参照してください。

比較

CALL RANBIN ルーチンは、シードおよび乱数ストリームを RANBIN 関数よりも高度に制御できます。

サンプル

次の例では、CALL RANBIN ルーチンを使用します。

```
data u1 (keep = x);
seed = 104;
do i = 1 to 5;
call ranbin(seed, 2000, 0.2 ,x);
output;
end;
call symputx('seed', seed);
run;

data u2 (keep = x);
seed = &seed;
do i = 1 to 5;
call ranbin(seed, 2000, 0.2 ,x);
output;
end;
run;

data all;
set u1 u2;
z = ranbin(104, 2000, 0.2);
run;

proc print label;
label x = 'Separate Streams' z = 'Single Stream';
run;
```

画面 2.2 CALL RANBIN ルーチンからの出力

The SAS System

Obs	Separate Streams	Single Stream
1	423	423
2	418	418
3	403	403
4	394	394
5	429	429
6	369	369
7	413	413
8	417	417
9	400	400
10	383	383

関連項目:

関数:

- “RAND 関数” (784 ページ)
- “RANBIN 関数” (782 ページ)

CALL RANCAU ルーチン

Cauchy 分布からランダム変量を返します。

カテゴリ: 乱数

構文

CALL RANCAU(*seed*,*x*);

必須引数

シード

シード値です。CALL RANCAU が実行されるたびに、*seed* の新しい値が返されます。

範囲: $seed < 2^{31} - 1$

注: $seed \leq 0$ の場合、シードストリームの初期化に時刻が使用されます。

参照項目: シード値の詳細については、シード値 (11 ページ)および乱数関数と乱数 CALL ルーチンのシード値の比較 (15 ページ)を参照

x

数値の SAS 変数です。CALL RANCAU が実行されるたびに、ランダム変量 *x* の新しい値が返されます。

詳細

CALL RANCAU ルーチンは *seed* を更新し、位置パラメータが 0、尺度パラメータが 1 の Cauchy 分布から生成された変量 *x* を返します。

シードを調整することで、同一または後続の DATA ステップの一部またはすべてのオブザベーションで変量のストリームへの一致または不一致を強制できます。

RANUNI 一様変量に適用される受容-棄却手法が使用されます。*u* および *v* が独立した一様(−1/2、1/2)変数で $u^2+v^2 \leq 1/4$ の場合、*u*/*v* は Cauchy 変量です。

乱数 CALL ルーチンの効果的な使用方法に関する説明と例については、ストリームの開始、停止および再開 (26 ページ)を参照してください。

比較

CALL RANCAU ルーチンは、シードおよび乱数ストリームを RANCAU 関数よりも高度に制御できます。

サンプル

この例では、CALL RANCAU ルーチンを使用します。

```
data case;
retain Seed_1 Seed_2 Seed_3 45;
do i=1 to 10;
call rancau(Seed_1,X1);
call rancau(Seed_2,X2);
X3=rancau(Seed_3);
if i=5 then
do;
Seed_2=18;
Seed_3=18;
end;
output;
end;
run;

proc print;
id i;
var Seed_1-Seed_3 X1-X3;
run;
```

画面 2.3 CALL RANCAU ルーチンからの出力

The SAS System

i	Seed_1	Seed_2	Seed_3	X1	X2	X3
1	1404437564	1404437564	45	-1.14736	-1.14736	-1.14736
2	1326029789	1326029789	45	-0.23735	-0.23735	-0.23735
3	1988843719	1988843719	45	-0.15474	-0.15474	-0.15474
4	1233028129	1233028129	45	4.97935	4.97935	4.97935
5	50049159	18	18	0.20402	0.20402	0.20402
6	802575599	991271755	18	3.43645	4.44427	3.43645
7	1233458739	1437043694	18	6.32808	-1.79200	6.32808
8	52428589	959908645	18	0.18815	-1.67610	0.18815
9	1216356463	1225034217	18	0.80689	3.88391	0.80689
10	1711885541	425626811	18	0.92971	-1.31309	0.92971

CALL RANCAU ルーチンの別の例を次に示します。

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

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

data all;
set u1 u2;
z = rancau(104);
run;

proc print label;
label x = 'Separate Streams' z = 'Single Stream';
run;
```

画面 2.4 CALL RANCAU ルーチンからの出力

The SAS System

Obs	Separate Streams	Single Stream
1	-0.6780	-0.6780
2	0.1712	0.1712
3	1.1372	1.1372
4	0.1478	0.1478
5	16.6536	16.6536
6	0.0747	0.0747
7	-0.5872	-0.5872
8	1.4713	1.4713
9	0.1792	0.1792
10	-0.0473	-0.0473

関連項目:

関数:

- “RAND 関数” (784 ページ)
- “RANCAU 関数” (783 ページ)

CALL RANCOMB ルーチン

引数の値を置換し、*n* 個の値のうち *k* 個のランダムな組み合わせを返します。

カテゴリ: 組み合わせ関数

構文

CALL RANCOMB(*seed*, *k*, *variable-1* <, *variable–2, ...*>);

必須引数

シード
乱数シードが含まれる数値変数です。シードの詳細については、“シード値” (11 ページ)を参照してください。

k
ランダムな組み合わせに含める値の数です。

変数
同じ長さのすべての数値変数またはすべての文字変数を指定します。これら変数の *K* 個の値がランダムに置換されます。

詳細

基本

n 個の変数がある場合、CALL RANCOMB では最初の *k* 個の値を昇順で並べ替えて、*n* 個の値のうち *k* 個のランダムな組み合わせを作成する方法で変数の値を置換します。つまり、*n* 個の値のうち *k* 個のすべての $n!/(k!(n-k)!)$の組み合わせは、最初の *k* 個の値が返されることと等しく起こりえます。

CALL RANCOMB ルーチンの実行中にエラーが発生すると、次の両方の値が設定されます。

- &SYSERR には 4 よりも大きい値が割り当てられます。
- &SYSINFO には-100 よりも小さい値が割り当てられます。

エラーがない場合、&SYSERR および&SYSINFO はゼロに設定されます。

マクロと CALL RANCOMB を使用する

%SYSCALL マクロを使用するときに CALL RANCOMB ルーチンを呼び出せます。この場合、*変数*の引数を同一の種類または長さとする必要はありません。ただし、返される最初の *k* 個の値に文字と数値の両方が含まれる場合、それらの値は並べ替えられません。%SYSCALL で引数が数値であることが識別されると、%SYSCALL は戻り値の形式を調整します。

サンプル

サンプル 1: DATA ステップで CALL RANCOMB を使用する

CALL RANCOMB ルーチンを使用して、指定された値のランダムな組み合わせを生成する方法の例を次に示します。

```
data _null_;
array x x1-x5 (1 2 3 4 5);
seed = 1234567890123;
do n=1 to 10;
call rancomb(seed, 3, of x1-x5);
put seed= @20 ' x= ' x1-x3;
end;
run;
```

ログ 2.2 *DATA ステップでの CALL RANCOMB ルーチンの使用によるログ出力*

```
seed=1332351321 x= 2 4 5
seed=829042065 x= 1 3 4
seed=767738639 x= 2 3 5
seed=1280236105 x= 2 4 5
seed=670350431 x= 1 2 5
seed=1956939964 x= 2 3 4
seed=353939815 x= 1 3 4
seed=1996660805 x= 1 2 5
seed=1835940555 x= 2 4 5
seed=910897519 x= 2 3 4
```

サンプル 2: マクロと CALL RANCOMB を使用する

マクロで使用される CALL RANCOMB ルーチンの例を次に示します。

```
%macro test;
%let x1=ant;
%let x2=-.1234;
%let x3=1e10;
%let x4=hippopotamus;
%let x5=zebra;
%let k=3;
%let seed = 12345;
%do j=1 %to 10;
%syscall rancomb(seed, k, x1, x2, x3, x4, x5);
%put j=&j &x1 &x2 &x3;
%end;
%mend;

%test;
```

SAS は次の出力を口グに書き込みます。

```
j=1 -0.1234 hippopotamus zebra
j=2 hippopotamus -0.1234 10000000000
j=3 hippopotamus ant zebra
j=4 -0.1234 zebra ant
j=5 -0.1234 ant hippopotamus
j=6 10000000000 hippopotamus ant
j=7 10000000000 hippopotamus ant
j=8 ant 10000000000 -0.1234
j=9 zebra -0.1234 10000000000
j=10 zebra hippopotamus 10000000000
```

関連項目:

関数:

- “RAND 関数” (784 ページ)

CALL ルーチン:

- “CALL RANPERK ルーチン” (220 ページ)
- “CALL ALLPERM ルーチン” (153 ページ)
- “CALL RANPERM ルーチン” (222 ページ)

CALL RANEXP ルーチン

指数分布からランダム変量を返します。

カテゴリ: 乱数

構文

CALL RANEXP(*seed*,*x*);

必須引数

シード

シード値です。CALL RANEXP が実行されるたびに、*seed* の新しい値が返されます。

範囲: $seed < 2^{31} - 1$

注: $seed \leq 0$ の場合、シードストリームの初期化に時刻が使用されます。

参照項目: シード値の詳細については、シード値 (11 ページ)および乱数関数と乱数 CALL ルーチンのシード値の比較 (15 ページ)を参照

x

数値変数です。CALL RANEXP が実行されるたびに、ランダム変量 *x* の新しい値が返されます。

詳細

CALL RANEXP ルーチンは *seed* を更新し、パラメータが 1 の指数分布から生成された変量 *x* を返します。

シードを調整することで、同一または後続の DATA ステップの一部またはすべてのオブザベーションで変量のストリームへの一致または不一致を強制できます。

CALL RANEXP ルーチンは、RANUNI 一様変量に適用される逆変換手法を使用します。

乱数 CALL ルーチンの効果的な使用方法に関する説明と例については、ストリームの開始、停止および再開 (26 ページ)を参照してください。

比較

CALL RANEXP ルーチンは、シードおよび乱数ストリームを RANEXP 関数よりも高度に制御できます。

サンプル

この例では、CALL RANEXP ルーチンを使用します。

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

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

data all;
set u1 u2;
z = ranexp(104);
```

```
run;

proc print label;
label x = 'Separate Streams' z = 'Single Stream';
run;
```

画面 2.5 CALL RANEXP ルーチンからの出力

The SAS System

Obs	Separate Streams	Single Stream
1	1.44347	1.44347
2	0.11740	0.11740
3	0.54175	0.54175
4	0.02280	0.02280
5	0.16645	0.16645

関連項目:

関数:

- “RAND 関数” (784 ページ)
- “RANEXP 関数” (794 ページ)

CALL RANGAM ルーチン

ガンマ分布からランダム変量を返します。

カテゴリ: 乱数

構文

CALL RANGAM(*seed*,*a*,*x*);

必須引数

シード

シード値です。CALL RANGAM が実行されるたびに、*seed* の新しい値が返されます。

範囲: $seed < 2^{31} - 1$

注: $seed \leq 0$ の場合、シードストリームの初期化に時刻が使用されます。

参照項目: シード値の詳細については、シード値 (11 ページ)および乱数関数と乱数 CALL ルーチンのシード値の比較 (15 ページ)を参照

a

数値の形状パラメータです。

範囲: $a > 0$

x
数値変数です。CALL RANGAM が実行されるたびに、ランダム変量 *x* の新しい値が返されます。

詳細

CALL RANGAM ルーチンは *seed* を更新し、パラメータが *a* のガンマ分布から生成された変量 *x* を返します。

シードを調整することで、同一または後続の DATA ステップの一部またはすべてのオブザベーションで変量のストリームへの一致または不一致を強制できます。

a>1 の場合、Cheng による受容-棄却手法が使用されます(Cheng、1977)。$a \leq 1$ の場合、Fishman による受容-棄却手法が使用されます(Fishman、1978)。詳細については、“リファレンス” (971 ページ) を参照してください。

乱数 CALL ルーチンの効果的な使用方法に関する説明と例については、ストリームの開始、停止および再開 (26 ページ)を参照してください。

比較

CALL RANGAM ルーチンは、シードおよび乱数ストリームを RANGAM 関数よりも高度に制御できます。

サンプル

この例では、CALL RANGAM ルーチンを使用します。

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

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

data all;
set u1 u2;
z = rangam(104, 1);
run;

proc print label;
label x = 'Separate Streams' z = 'Single Stream';
run;
```

画面 2.6 CALL RANGAM ルーチンからの出力

The SAS System

Obs	Separate Streams	Single Stream
1	1.44347	1.44347
2	0.11740	0.11740
3	0.54175	0.54175
4	0.02280	0.02280
5	0.16645	0.16645

CALL RANGAM ルーチンを使用する別の例を次に示します。

```
data case;
retain Seed_1 Seed_2 Seed_3 45;
a=2;
do i=1 to 10;
call rangam(Seed_1,a,X1);
call rangam(Seed_2,a,X2);
X3=rangam(Seed_3,a);
if i=5 then
do;
Seed_2=18;
Seed_3=18;
end;
output;
end;
run;

proc print;
id i;
var Seed_1-Seed_3 X1-X3;
run;
```

画面 2.7 CALL RANGAM ルーチンからの出力

The SAS System

i	Seed_1	Seed_2	Seed_3	X1	X2	X3
1	1404437564	1404437564	45	1.30569	1.30569	1.30569
2	1326029789	1326029789	45	1.87514	1.87514	1.87514
3	1988843719	1988843719	45	1.71597	1.71597	1.71597
4	50049159	50049159	45	1.59304	1.59304	1.59304
5	802575599	18	18	0.43342	0.43342	0.43342
6	100573943	991271755	18	1.11812	1.32646	1.11812
7	1986749826	1437043694	18	0.68415	0.88806	0.68415
8	52428589	959908645	18	1.62296	2.46091	1.62296
9	1216356463	1225034217	18	2.26455	4.06596	2.26455
10	805366679	425626811	18	2.16723	6.94703	2.16723

i=5 の場合、CALL
RANGAM ステートメントで Seed_
2 を変更すると、X2 の変量のストリームが強制的にX1 の変量のストリームから逸脱します。ただ
3 を変更した場合には影響しません。

関連項目:

関数:

- “RAND 関数” (784 ページ)
- “RANGAM 関数” (795 ページ)

CALL RANNOR ルーチン

正規分布からランダム変量を返します。

カテゴリ: 乱数

構文

CALL RANNOR(*seed*,*x*);

必須引数

シード
シード値です。CALL RANNOR が実行されるたびに、*seed* の新しい値が返されます。

範囲: $seed < 2^{31} - 1$

注: *seed* ≤ 0 の場合、シードストリームの初期化に時刻が使用されます。

参照項目: シード値の詳細については、シード値 (11 ページ)および乱数関数と乱数 CALL ルーチンのシード値の比較 (15 ページ)を参照

x

数値変数です。CALL RANNOR が実行されるたびに、ランダム変量 *x* の新しい値が返されます。

詳細

CALL RANNOR ルーチンは *seed* を更新し、平均が 0 で分散が 1 の正規分布から生成された変量 *x* を返します。

シードを調整することで、同一または後続の DATA ステップの一部またはすべてのオブザベーションで変量のストリームへの一致または不一致を強制できます。

CALL RANNOR ルーチンは、RANUNI 一様変量の Box-Muller 変換を使用します。

乱数 CALL ルーチンの効果的な使用方法に関する説明と例については、ストリームの開始、停止および再開 (26 ページ)を参照してください。

比較

CALL RANNOR ルーチンは、シードおよび乱数ストリームを RANNOR 関数よりも高度に制御できます。

サンプル

この例では、CALL RANNOR ルーチンを使用します。

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

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

data all;
set u1 u2;
z = rannor(104);
run;

proc print label;
label x = 'Separate Streams' z = 'Single Stream';
run;
```

画面 2.8 CALL RANNOR ルーチンからの出力

The SAS System

Obs	Separate Streams	Single Stream
1	1.30390	1.30390
2	1.03049	1.03049
3	0.19491	0.19491
4	-0.34987	-0.34987
5	1.64273	1.64273
6	-1.75842	-1.75842
7	0.75080	0.75080
8	0.94375	0.94375
9	0.02436	0.02436
10	-0.97256	-0.97256

関連項目:

関数:

- “RAND 関数” (784 ページ)
- “RANNOR 関数” (798 ページ)

CALL RANPERK ルーチン

引数の値を置換し、*n* 個の値のうち *k* 個のランダムな順列を返します。

カテゴリ: 組み合わせ関数

構文

CALL RANEXP(*seed*, *k*, *variable-1* <, *variable–2, ...*>);

必須引数

シード

乱数シードが含まれる数値変数です。シードの詳細については、“シード値” (11 ページ)を参照してください。

k

ランダムな順列に含める値の数です。

変数

同じ長さのすべての数値変数またはすべての文字変数を指定します。これら変数の *K* 個の値がランダムに置換されます。

詳細

マクロと CALL RANPERK を使用する

%SYSCALL マクロを使用するときに RANPERK ルーチンを呼び出せます。この場合、*変数*の引数を同一の種類または長さとする必要はありません。
%SYSCALL で引数が数値であることが識別されると、%SYSCALL は戻り値の形式を調整します。

CALL RANPERK ルーチンの実行中にエラーが発生すると、次の両方の値が設定されます。

- &SYSERR には 4 よりも大きい値が割り当てられます。
- &SYSINFO には-100 よりも小さい値が割り当てられます。

エラーがない場合、&SYSERR および&SYSINFO はゼロに設定されます。

サンプル

サンプル 1: DATA ステップで CALL RANPERK を使用する

CALL RANPERK ルーチンを使用して、指定された値のランダムな順列を生成する方法の例を次に示します。

```
data _null_;
array x x1-x5 (1 2 3 4 5);
seed = 1234567890123;
do n=1 to 10;
call ranperk(seed, 3, of x1-x5);
put seed= @20 ' x= ' x1-x3;
end;
run;
```

ログ 2.3 *DATA ステップでの CALL RANPERK ルーチンの使用によるログ出力*

```
seed=1332351321 x= 5 4 2
seed=829042065 x= 4 1 3
seed=767738639 x= 5 1 2
seed=1280236105 x= 3 2 5
seed=670350431 x= 4 3 5
seed=1956939964 x= 3 1 2
seed=353939815 x= 4 2 1
seed=1996660805 x= 3 4 5
seed=1835940555 x= 5 1 4
seed=910897519 x= 5 1 2
```

サンプル 2: マクロと CALL RANPERK を使用する

マクロで使用される CALL RANPERK ルーチンの例を次に示します。

```
%macro test;
%let x1=ant;
%let x2=-.1234;
%let x3=1e10;
%let x4=hippopotamus;
%let x5=zebra;
%let k=3;
%let seed = 12345;
%do j=1 %to 10;
```

```
%syscall ranperk(seed, k, x1, x2, x3, x4, x5);
%put j=&j &x1 &x2 &x3;
%end;
%mend;

%test;
```

ログ 2.4 マクロでの CALL RANPERK ルーチンの使用によるログ出力

```
j=1 -0.1234 hippopotamus zebra
j=2 hippopotamus -0.1234 10000000000
j=3 hippopotamus ant zebra
j=4 -0.1234 zebra ant
j=5 -0.1234 ant hippopotamus
j=6 10000000000 hippopotamus ant
j=7 10000000000 hippopotamus ant
j=8 ant 10000000000 -0.1234
j=9 zebra -0.1234 10000000000
j=10 zebra hippopotamus 10000000000
```

関連項目:

関数:

- “RAND 関数” (784 ページ)

CALL ルーチン:

- “CALL ALLPERM ルーチン” (153 ページ)
- “CALL RANPERM ルーチン” (222 ページ)

CALL RANPERM ルーチン

引数の値をランダムに置換します。

カテゴリ: 組み合わせ関数

構文

CALL RANPERM(*seed*, *variable-1* <, *variable–2, ...*>);

必須引数

シード

乱数シードが含まれる数値変数です。シードの詳細については、“シード値” (11 ページ)を参照してください。

変数

同じ長さのすべての数値変数またはすべての文字変数を指定します。これらの変数の値がランダムに置換されます。

詳細

マクロと CALL RANPERM を使用する

%SYSCALL マクロを使用するときに RANPERM ルーチンを呼び出せます。この場合、*変数*の引数を同一の種類または長さとする必要はありません。%SYSCALL で引数が数値であることが識別されると、%SYSCALL は戻り値の形式を調整します。

CALL RANPERM ルーチンの実行中にエラーが発生すると、次の両方の値が設定されます。

- &SYSERR には 4 よりも大きい値が割り当てられます。
- &SYSINFO には-100 よりも小さい値が割り当てられます。

エラーがない場合、&SYSERR および&SYSINFO はゼロに設定されます。

サンプル

サンプル 1: DATA ステップで CALL RANPERM を使用する

CALL RANPERM ルーチンを使用して、指定された値のランダムな順列を生成する例を次に示します。

```
data _null_;
array x x1-x4 (1 2 3 4);
seed = 1234567890123;
do n=1 to 10;
call ranperm(seed, of x1-x4);
put seed= @20 ' x= ' x1-x4;
end;
run;
```

ログ 2.5　DATA ステップでの CALL RANPERM ルーチンの使用による出力

```
seed=1332351321 x= 1 3 2 4
seed=829042065 x= 3 4 2 1
seed=767738639 x= 4 2 3 1
seed=1280236105 x= 1 2 4 3
seed=670350431 x= 2 1 4 3
seed=1956939964 x= 2 4 3 1
seed=353939815 x= 4 1 2 3
seed=1996660805 x= 4 3 1 2
seed=1835940555 x= 4 3 2 1
seed=910897519 x= 3 2 1 4
```

サンプル 2: マクロと CALL RANPERM を使用する

%SYSCALL マクロで使用される CALL RANPERM ルーチンの例を次に示します。

```
%macro test;
%let x1=ant;
%let x2=-.1234;
%let x3=1e10;
%let x4=hippopotamus;
%let x5=zebra;
%let seed = 12345;
%do j=1 %to 10;
```

```
%syscall ranperm(seed, x1, x2, x3, x4, x5);
%put j=&j &x1 &x2 &x3;
%end;
%mend;

%test;
```

ログ 2.6 マクロでの CALL RANPERM ルーチンの使用による出力

```
j=1 zebra ant hippopotamus
j=2 10000000000 ant -0.1234
j=3 -0.1234 10000000000 ant
j=4 hippopotamus ant zebra
j=5 -0.1234 zebra 10000000000
j=6 -0.1234 hippopotamus ant
j=7 zebra ant -0.1234
j=8 -0.1234 hippopotamus ant
j=9 ant -0.1234 hippopotamus
j=10 -0.1234 zebra 10000000000
```

関連項目:

関数:

- “RAND 関数” (784 ページ)

CALL ルーチン:

- “CALL ALLPERM ルーチン” (153 ページ)
- “CALL RANPERK ルーチン” (220 ページ)

CALL RANPOI ルーチン

ポアソン分布からランダム変量を返します。

カテゴリ: 乱数

構文

CALL RANPOI(*seed*,*m*,*x*);

必須引数

シード

シード値です。CALL RANPOI を実行するたびに、*seed* の新しい値が返されます。

範囲: $seed < 2^{31} - 1$

注: $seed \leq 0$ の場合、シードストリームの初期化に時刻が使用されます。

参照項目: シード値の詳細については、“シード値” (11 ページ)および“乱数関数と乱数 CALL ルーチンのシード値の比較” (15 ページ)を参照してください。

m

数値の平均パラメータです。

範囲: $m \geq 0$

x
数値変数です。CALL RANPOI を実行するたびに、ランダム変量 *x* の新しい値が返されます。

詳細

CALL RANPOI ルーチンは *seed* を更新し、ポアソン分布から生成される平均値 *m* の変量 *x* を返します。

シードを調整することで、同一または後続の DATA ステップの一部またはすべてのオブザベーションで変量のストリームへの一致または不一致を強制できます。

$m < 85$ の場合、RANUNI 一様変量に適用される逆変換法が使用されます (Fishman, 1976; 次を参照: “リファレンス” (971 ページ).)。$m \geq 85$ の場合、Poisson ランダム変数の正規近似が使用されます。迅速に実行するために、内部変数は初期呼び出し(新しい各 *m*)でのみ計算されます。

乱数 CALL ルーチンの効果的な使用方法と例については、“ストリームを開始、停止および再開する” (26 ページ)を参照してください。

比較

CALL RANPOI ルーチンは、シードおよび乱数ストリームを RANPOI 関数よりも高度に制御できます。

サンプル

この例では、CALL RANPOI ルーチンを使用します。

```
data u1(keep=x);
seed = 104;
do i = 1 to 5;
call ranpoi(seed, 2000, x);
output;
end;
call symputx('seed', seed);
run;
data u2(keep=x);
seed = &seed;
do i = 1 to 5;
call ranpoi(seed, 2000, x);
output;
end;
run;
data all;
set u1 u2;
z = ranpoi(104, 2000);
run;
proc print label;
label x = 'Separate Streams' z = 'Single Stream';
run;
```

画面 2.9 CALL RANPOI ルーチンからの出力

The SAS System

Obs	Separate Streams	Single Stream
1	2058	2058
2	2046	2046
3	2009	2009
4	1984	1984
5	2073	2073
6	1921	1921
7	2034	2034
8	2042	2042
9	2001	2001
10	1957	1957

関連項目:

関数:

- “RAND 関数” (784 ページ)
- “RANPOI 関数” (799 ページ)

CALL RANTBL ルーチン

テーブル形式の確率分布からランダム変量を返します。

カテゴリ: 乱数

構文

CALL RANTBL(*seed*,p_1,...p_i,...p_n,*x*);

必須引数

シード

シード値です。CALL RANTBL を実行するたびに、*seed* の新しい値が返されます。

範囲: $seed < 2^{31} - 1$

注: $seed \leq 0$ の場合、シードストリームの初期化に時刻が使用されます。

参照項目: “シード値” (11 ページ) および “乱数関数と乱数 CALL ルーチンのシード値の比較” (15 ページ)

***p*_{*i*}**

数値の SAS 値です。

範囲: $0 < i \le n$ に対し $0 \le p_i \le 1$

x

数値の SAS 変数です。CALL RANTBL を実行するたびに、ランダム変量 *x* の新しい値が返されます。

詳細

CALL RANTBL ルーチンは *seed* を更新し、$p_1 \sim p_n$ *で定義された*確率質量関数から生成される変量 *x* を返します。

シードを調整することで、同一または後続の DATA ステップの一部またはすべてのオブザベーションで変量のストリームへの一致または不一致を強制できます。

RANUNI 一様変量に適用される逆変換法が使用されます。CALL RANTBL ルーチンは次のデータを返します。

$1 \quad with \ probability \ p_1$

$2 \quad with \ probability \ p_2$

.

.

.

$n \quad with \ probability \ p_n$

$n+1 \quad with \ probability \ 1 - \sum_{i=1}^{n} p_i \ if \ \sum_{i=1}^{n} p_i \le 1$

インデックス $j < n$ の場合、

$$\sum_{i=1}^{j} p_i \ge 1$$

RANTBL はインデックス *j* の発生確率が次と等しくなる、インデックス $1 \sim j$ のみを返します。

$$1 - \sum_{i=1}^{j-1} p_i$$

乱数 CALL ルーチンの効果的な使用方法と例については、“ストリームを開始、停止および再開する” (26 ページ)を参照してください。

比較

CALL RANTBL ルーチンは、シードおよび乱数ストリームを RANTBL 関数よりも高度に制御できます。

サンプル

この例では、CALL RANTBL ルーチンを使用します。

```
data u1(keep=x);
seed = 104;
do i = 1 to 5;
call rantbl(seed, .02, x);
output;
end;
call symputx('seed', seed);
run;
data u2(keep=x);
seed = &seed;
do i = 1 to 5;
call rantbl(seed, .02, x);
output;
end;
run;
data all;
set u1 u2;
z = rantbl(104, .02);
run;
proc print label;
label x = 'Separate Streams' z = 'Single Stream';
run;
```

画面 2.10 *CALL RANTBL ルーチンからの出力*

The SAS System

Obs	Separate Streams	Single Stream
1	2	2
2	2	2
3	2	2
4	2	2
5	2	2
6	2	2
7	2	2
8	2	2
9	2	2
10	2	2

関連項目:

関数:

- “RAND 関数” (784 ページ)
- “RANTBL 関数” (800 ページ)

CALL RANTRI ルーチン

三角分布からランダム変量を返します。

カテゴリ: 乱数

構文

CALL RANTRI(*seed*,*h*,*x*);

必須引数

シード

シード値です。CALL RANTRI を実行するたびに、*seed* の新しい値が返されます。

範囲: $seed < 2^{31} - 1$

注: $seed \leq 0$ の場合、シードストリームの初期化に時刻が使用されます。

参照項目: シード値の詳細については、“シード値” (11 ページ)および“乱数関数と乱数 CALL ルーチンのシード値の比較” (15 ページ)を参照してください。

h

数値の SAS 値です。

範囲: $0<h<1$

x

数値の SAS 変数です。CALL RANTRI を実行するたびに、ランダム変量 *x* の新しい値が返されます。

詳細

CALL RANTRI ルーチンは *seed* を更新し、分布のモーダル値である、パラメータ *h* を使用した間隔(0,1)の三角分布から生成される変量 *x* を返します。

シードを調整することで、同一または後続の DATA ステップの一部またはすべてのオブザベーションで変量のストリームへの一致または不一致を強制できます。

CALL RANTRI ルーチンでは、RANUNI 一様変量に適用される逆変換法を使用します。

乱数 CALL ルーチンの効果的な使用方法と例については、“ストリームを開始、停止および再開する” (26 ページ)を参照してください。

比較

CALL RANTRI ルーチンは、シードおよび乱数ストリームを RANTRI 関数よりも高度に制御できます。

サンプル

この例では、CALL RANTRI ルーチンを使用します。

```
data u1(keep=x);
seed = 104;
do i = 1 to 5;
call rantri(seed, .5, x);
output;
end;
call symputx('seed', seed);
run;
data u2(keep=x);
seed = &seed;
do i = 1 to 5;
call rantri(seed, .5, x);
output;
end;
run;
data all;
set u1 u2;
z = rantri(104, .5);
run;
proc print label;
label x = 'Separate Streams' z = 'Single Stream';
run;
```

画面 2.11 *CALL RANTRI ルーチンからの出力*

The SAS System

Obs	Separate Streams	Single Stream
1	0.34359	0.34359
2	0.76466	0.76466
3	0.54269	0.54269
4	0.89384	0.89384
5	0.72311	0.72311
6	0.68763	0.68763
7	0.48468	0.48468
8	0.38467	0.38467
9	0.29881	0.29881
10	0.80369	0.80369

関連項目:

関数:

- “RAND 関数” (784 ページ)
- “RANTRI 関数” (801 ページ)

CALL RANUNI ルーチン

一様分布からランダム変量を返します。

カテゴリ: 乱数

構文

CALL RANUNI(*seed*,*x*);

必須引数

シード

シード値です。CALL RANUNI を実行するたびに、*seed* の新しい値が返されます。

範囲: $seed < 2^{31} - 1$

ヒント: $seed \leq 0$ の場合、シードストリームの初期化に時刻が使用されます。

参照項目: シード値の詳細については、“シード値” (11 ページ)および“乱数関数と乱数 CALL ルーチンのシード値の比較” (15 ページ)を参照してください。

x

数値変数です。CALL RANUNI を実行するたびに、ランダム変量 *x* の新しい値が返されます。

詳細

CALL RANUNI ルーチンは *seed* を更新し、素数係数乗法型ジェネレータで係数 $2^{31}-1$ と乗数 397204094 を使用して、間隔(0,1)の一様分布から生成される変量 *x* を返します(Fishman and Moore 1982) (“リファレンス” (971 ページ)、 を参照)。

シードを調整することで、同一または後続の DATA ステップの一部またはすべてのオブザベーションで変量のストリームへの一致または不一致を強制できます。

乱数 CALL ルーチンの効果的な使用方法と例については、“ストリームを開始、停止および再開する” (26 ページ)を参照してください。

比較

CALL RANUNI ルーチンは、シードおよび乱数ストリームを RANUNI 関数よりも高度に制御できます。

サンプル: CALL RANUNI ルーチンを使用する

この例では、CALL RANUNI ルーチンを使用します。

```
data u1(keep=x);
seed = 104;
do i = 1 to 5;
call ranuni(seed, x);
```

```
output;
end;
call symputx('seed', seed);
run;
data u2(keep=x);
seed = &seed;
do i = 1 to 5;
call ranuni(seed, x);
output;
end;
run;
data all;
set u1 u2;
z = ranuni(104);
run;
proc print label;
label x = 'Separate Streams' z = 'Single Stream';
run;
```

画面 2.12 *CALL RANUNI ルーチンからの出力*

The SAS System

Obs	Separate Streams	Single Stream
1	0.23611	0.23611
2	0.88923	0.88923
3	0.58173	0.58173
4	0.97746	0.97746
5	0.84667	0.84667
6	0.80484	0.80484
7	0.46983	0.46983
8	0.29594	0.29594
9	0.17858	0.17858
10	0.92292	0.92292

関連項目:

関数:

- “RAND 関数” (784 ページ)
- “RANUNI 関数” (802 ページ)

CALL SCAN ルーチン

文字列から *n* 番目の単語の位置と長さを返します。

カテゴリ: 文字関数

操作: %SYSCALL マクロステートメントで呼び出されると、CALL SCAN の引数から引用符が削除されます。詳細については、“CALL ルーチンと%SYSCALL マクロステートメントを使用する” (9 ページ)を参照してください。

構文

CALL SCAN(<*string*>, *count*, *position*, *length* <, <*charlist*><, <*modifier(s)*>>>);

必須引数

count
CALL SCAN ルーチンが文字列内で選択する単語の番号を指定する整数値を使用するゼロ以外の数値の定数、変数または式です。たとえば、1 の値は 1 番目の単語、2 の値は 2 番目の単語を示します。次のルールが適用されます。

- *count* が正の場合、CALL SCAN は文字列の単語を左から右へ数えます。
- *count* が負の場合、CALL SCAN は文字列の単語を右から左へ数えます。

position
単語の位置を返す数値変数を指定します。*count* が文字列内の単語数を超えると、*position* で返される値はゼロになります。*count* がゼロまたは欠損している場合、*position* で返される値は欠損します。

長さ
単語の長さを返す数値変数を指定します。*count* が文字列内の単語数を超えると、*length* で返される値はゼロになります。*count* がゼロまたは欠損している場合、*length* で返される値は欠損します。

オプション引数

string
文字定数、変数または式を指定します。

charlist
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。このリストは、単語を区切る区切り文字として使用する文字を決定します。次のルールが適用されます。

- デフォルトでは、*charlist* のすべての文字が区切り文字として使用されます。
- *modifier* 引数で K 修飾子を指定すると、*charlist* に含まれていない文字がすべて区切り文字として使用されます。

ヒント: その他の修飾子を使用して *charlist* にさらに文字を追加できます。

modifier
空白以外の各文字によって CALL SCAN ルーチンのアクションが変更される、文字定数、変数または式を指定します。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。

a または A
: 文字のリストにアルファベット文字を追加します。

b または B
: *count* 引数の符号に関係なく、左から右ではなく、右から左へ逆行スキャンします。

c または C
: 文字のリストに制御文字を追加します。

d または D
: 文字のリストに数字を追加します。

f または F
: 文字のリストにアンダースコアと英文字(VALIDVARNAME=V7 を使用した SAS 変数名内の有効な最初の文字)を追加します。

g または 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 または H
: 文字のリストに水平タブを追加します。

i または I
: 大文字か小文字かは無視します。

k または 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 を指定すると、文字のリストに含まれている文字は区切り文字であるため、戻り値から除外されずにそのまま使用されます。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 または L
: 文字のリストに小文字を追加します。

m または 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 または 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 または O
: *charlist* 引数と *modifier* 引数を、CALL SCAN ルーチンが呼び出されるたびではなく、一度だけ処理します。DATA ステップで O 修飾子を使用すると、*charlist* 引数と *modifier* 引数が変更されないループで CALL SCAN を呼び出すときに、より迅速に実行できます。O 修飾子は SAS コード内の CALL SCAN ルーチンの各インスタンスに個別に適用され、CALL SCAN ルーチンのすべてのインスタンスで同じ区切り文字と修飾子が使用されるようにはなりません。

p または P
: 文字のリストに句読点を追加します。

q または 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、一致しない引用符が含まれている場合、左から右へのスキャンでは、右から左へのスキャンとは異なる単語が生成されます。

s または 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 または T	*string* 引数と *charlist* 引数から末尾の空白を取り除きます。両方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、CALL SCAN ルーチンで T 修飾子を使用するかわりに、TRIM 関数を使用します。
u または U	文字のリストに大文字を追加します。
w または W	文字のリストに印刷可能(書き込み可能)な文字を追加します。
x または X	文字のリストに 16 進文字を追加します。

ヒント: *modifier* 引数が文字定数の場合は引用符で囲みます。一組の引用符で複数の修飾子を指定します。*modifier* 引数は、文字変数または文字式としても表すことができます。

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。区切り文字は *charlist* 引数と *modifier* 引数で指定できます。

Q 修飾子を指定すると、引用符で囲まれた部分文字列内の区切り文字は無視されます。

CALL SCAN ルーチンでは、"単語"とは次の特性すべて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

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、CALL SCAN ルーチンでは、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。

ASCII 環境と EBCDIC 環境でデフォルトの区切り文字を使用する

CALL SCAN ルーチンで 4 つの引数のみを使用する場合、コンピュータで ASCII 文字または EBCDIC 文字が使われているかどうかによってデフォルトの区切り文字が異なります。

- コンピュータで ASCII 文字が使われている場合、デフォルトの区切り文字は次のとおりです。

  空白 ! $ % & ( ) * + , - . / ; < ^¦

  ^文字を含まない ASCII 環境の場合、CALL SCAN ルーチンではかわりに~文字を使用します。

- コンピュータで EBCDIC 文字が使われている場合、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* 引数で定義される文字のみになります。この場合、ASCII 環境

と EBCDIC 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、*charlist* 引数で明示的に指定された区切り文字のリストに追加します。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。

M 修飾子を指定して CALL SCAN ルーチンを使用する

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、次の条件を満たす場合、CALL SCAN ルーチンは正の位置と長さゼロを返します。

- 文字列が区切り文字で始まり、1 番目の単語を要求する。
- 文字列が区切り文字で終わり、最後の単語を要求する。
- 文字列に 2 つの連続する区切り文字が含まれており、2 つの区切り文字の間の単語を要求する。

カウントを文字列内の単語数より大きい絶対値に指定すると、CALL SCAN ルーチンはゼロの位置と長さを返します。

M 修飾子を指定せずに CALL SCAN ルーチンを使用する

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場合、CALL SCAN ルーチンは次のように動作します。

- 文字列の先頭または末尾の区切り文字を無視する
- 2 つ以上の連続する区切り文字を単一の区切り文字として扱う

文字列に区切り文字しか含まれていない、またはカウントを文字列内の単語数より大きい絶対値に指定すると、CALL SCAN ルーチンはゼロの位置と長さを返します。

単語を文字列として検索する

CALL SCAN ルーチンを呼び出した後に指定した単語を文字列として検索するには、*string* 引数、*position* 引数および *length* 引数を指定した SUBSTRN 関数を使用します。

```
substrn(string, position, length);
```

CALL SCAN はゼロの長さを返す可能性があるため、SUBSTR 関数を使用するとエラーが発生することがあります。

NULL 引数を使用する

CALL SCAN ルーチンでは、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。

サンプル

サンプル 1: 文字列内の単語をスキャンする

CALL SCAN ルーチンを使用して文字列内の単語の位置と長さを検索する方法の例を次に示します。

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

画面 2.13 *SAS 出力: 文字列内の単語をスキャンする*

The SAS System

Obs	i	position	length	Name
1	1	1	7	Picasso
2	2	9	8	Toulouse
3	3	18	7	Lautrec
4	4	26	6	Turner
5	5	33	4	"Van
6	6	38	5	Gogh"
7	7	44	9	Velazquez

サンプル 2: 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。注:

- カウントを負に指定すると、CALL SCAN ルーチン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、文字列内のすべての文字が区切り文字であるため、単語は検出されません。

```
data firstlast;
input String $60.;
call scan(string, 1, First_Pos, First_Length);
First_Word = substrn(string, First_Pos, First_Length);
call scan(string, -1, Last_Pos, Last_Length);
Last_Word = substrn(string, Last_Pos, Last_Length);
datalines4;
Jack and Jill
& Bob & Carol & Ted & Alice &
```

```
Leonardo
! $ % & ( ) * + , - . / ;
;;;;
proc print data=firstlast;
var First: Last:;
run;
```

画面 2.14 *文字列内の最初と最後の単語の検索結果*

The SAS System

Obs	First_Pos	First_Length	First_Word	Last_Pos	Last_Length	Last_Word
1	1	4	Jack	10	4	Jill
2	3	3	Bob	23	5	Alice
3	1	8	Leonardo	1	8	Leonardo
4	0	0		0	0	

サンプル 3: 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

単語が検出されなくなるまで文字列を左から右へスキャンする例を次に示します。M 修飾子が使用されていないため、CALL SCAN ルーチンは長さがゼロの単語は返しません。デフォルトの区切り文字に空白が含まれているため、CALL SCAN ルーチンはカウントが文字列内の単語数を超える場合にのみ、ゼロの位置または長さを返します。返された位置がゼロ以下の場合、ループを停止できます。ループを終了するには、エラーによって位置が欠損する場合に備えて、ゼロに対する厳密な等価比較を使用するかわりに不等比較を使用するほうが安全です (SAS では、欠損値は非欠損値より値が小さいとみなされます)。

```
data all;
length word $20;
drop string;
string = ' The quick brown fox jumps over the lazy dog. ';
do until(position <= 0);
count+1;
call scan(string, count, position, length);
word = substrn(string, position, length);
output;
end;
run;
proc print data=all noobs;
var count position length word;
run;
```

画面 2.15　M 修飾子を使用せずに文字列内のすべての単語を検索した結果

The SAS System

count	position	length	word
1	2	3	The
2	6	5	quick
3	12	5	brown
4	18	3	fox
5	22	5	jumps
6	28	4	over
7	33	3	the
8	37	4	lazy
9	42	3	dog
10	0	0	

サンプル 4: 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、CALL SCAN ルーチンは長さがゼロの単語を返します。

区切り文字と修飾子は CALL SCAN ルーチンのすべての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。

```
data comma;
length word $30;
string = ',leading, trailing,and multiple,,delimiters,,';
do until(position <= 0);
count + 1;
call scan(string, count, position, length, ',', 'mo');
word = substrn(string, position, length);
output;
end;
run;
proc print data=comma noobs;
var count position length word;
run;
```

画面 2.16　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した結果

The SAS System

count	position	length	word
1	1	0	
2	2	7	leading
3	10	10	trailing
4	21	12	and multiple
5	34	0	
6	35	10	delimiters
7	46	0	
8	47	0	
9	0	0	

サンプル 5: カンマ区切り値、引用符内の部分文字列および O 修飾子を使用する

O 修飾子と区切り文字としてカンマを指定した CALL SCAN ルーチンを使用する例を次に示します。

CALL SCAN ルーチン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。

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

画面 2.17 カンマ区切り値と引用符内の部分文字列の結果

The SAS System

count	position	length	word
1	1	7	He said
2	9	20	"She said, ""No!"""
3	30	11	not "Yes!"
4	0	0	

サンプル 6: 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

数字の部分文字列を検索する例を次に示します。*charlist* 引数が NULL であるため、文字のリストは最初は空になります。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。

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

画面 2.18 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した結果

The SAS System

count	position	length	digits
1	7	3	800
2	12	3	555
3	16	4	1234
4	0	0	

関連項目:

関数:

- “SCAN 関数” (824 ページ)
- “FINDW 関数” (457 ページ)
- “COUNTW 関数” (337 ページ)

CALL SET ルーチン

SAS データセット変数を名前とデータ型が同じ DATA ステップまたはマクロ変数にリンクします。

カテゴリ: 変数制御

構文

CALL SET(*data-set-id*);

必須引数

data-set-id
データセットを開くときに OPEN 関数によって割り当てられる ID。

詳細

SET を使用すると、関数を使用して SAS ファイルを読み込んだり操作したりするときに、変更または検証のために変数値にアクセスするために必要なコーディングを大幅に削減できます。CALL SET の後、SAS データセットから読み込みが実行されると、対応するマクロまたは DATA ステップ変数の値は一致する SAS データセット変数の値に設定されます。変数の長さが一致しない場合、値は必要に応じて切り捨てられるか、埋め込まれます。SET を使用しない場合、データセット変数とマクロまたは DATA ステップ変数の間で明示的に値を移動するには、GETVARC および GETVARN 関数を使用する必要があります。

一般的に、データセットとマクロおよび DATA ステップ変数をリンクするには、OPEN の直後に CALL SET を使用します。

サンプル: CALL SET ルーチンを使用する

この例では、CALL SET ルーチンを使用します。

- 次のステートメントは、オブザベーションをフェッチするときにマクロ変数 PRICE と STYLE の値を自動的に設定します。

```
%macro setvar;
%let dsid=%sysfunc(open(sasuser.houses,i));
/* No leading ampersand with %SYSCALL */
%syscall set(dsid);
%let rc=%sysfunc(fetchobs(&dsid,10));
%let rc=%sysfunc(close(&dsid));
%mend setvar;
%global price style;
%setvar
%put _global_;
```

- %PUT ステートメントは、SAS ログにこれらの行を書き込みます。

```
GLOBAL PRICE 127150
GLOBAL STYLE CONDO
```

- 次のステートメントは、SASUSER.HOUSES の最初の 10 件のオブザベーションの値を取得し、MYDATA に保存します。

```
data mydata;
/* create variables for assignment */
/*by CALL SET */
length style $8 sqfeet bedrooms baths 8
street $16 price 8;
drop rc dsid;
dsid=open("sasuser.houses","i");
call set (dsid);
do i=1 to 10;
rc=fetchobs(dsid,i);
output;
end;
run;
```

関連項目:

関数:

- “FETCH 関数” (398 ページ)
- “FETCHOBS 関数” (399 ページ)
- “GETVARC 関数” (510 ページ)
- “GETVARN 関数” (511 ページ)

CALL SLEEP ルーチン

指定した期間、この CALL ルーチンを呼び出すプログラムの実行を中断します。

カテゴリ: 特殊関数

参照項目: “CALL SLEEP Routine: UNIX” in *SAS Companion for UNIX Environments*
“CALL SLEEP Routine: z/OS” in *SAS Companion for z/OS*

構文

CALL SLEEP(*n* <, *unit*>);

必須引数

n
プログラムの実行を中断する時間の単位数を指定する数値定数。
範囲: $n \geq 0$

オプション引数

unit
n に適用される時間の単位(秒)を指定します。たとえば、1 は 1 秒、.001 は 1 ミリ秒、5 は 5 秒に相当します。

デフォルト: Windows PC 環境では 1、その他の環境では.001

詳細

CALL SLEEP ルーチンは指定した期間この CALL ルーチンを呼び出すプログラムの実行を中断します。プログラムには、DATA ステップ、マクロ、IML、SCL など、CALL ルーチンを呼び出せるものを指定できます。CALL SLEEP ルーチンの最大スリープ期間は 46 日です。

サンプル

サンプル 1: 指定した期間の実行中断

次に、DATA ステップ PAYROLL の実行を 1 分 10 秒間中断するように SAS に指示する例を示します。

```
data payroll;
call sleep(7000,.01);
...more SAS statements...
run;
```

サンプル 2: スリープ時間の計算に基づいた実行中断

次に、DATA ステップ BUDGET の実行を 2013 年 3 月 1 日午前 3 時まで中断するように SAS に指示する例を示します。SAS では、対象日と DATA ステップの実行開始日時に基づいて中断の長さが計算されます。

```
data budget;
sleeptime='01mar2013:03:00'dt-datetime();
call sleep(sleeptime,1);
...more SAS statements...;
run;
```

関連項目:

関数:

- “SLEEP 関数” (839 ページ)

CALL SOFTMAX ルーチン

softmax 値を返します。

カテゴリ: 数学関数

構文

CALL SOFTMAX(*argument*<,*argument*,…>);

必須引数

引数
数値です。

制限事項: CALL SOFTMAX ルーチンでは、有効な引数として変数のみを使用できます。これらの引数は CALL ルーチンで更新できないため、定数または SAS 式は使用しないでください。

詳細

CALL SOFTMAX ルーチンは、各引数をその引数の softmax 値で置き換えます。たとえば、x_j に置き換えられます。

$$\frac{\varepsilon^{x_j}}{\sum_{i=1}^{i=n} \varepsilon^{x_i}}$$

いずれかの引数に欠損値が含まれる場合、CALL SOFTMAX はすべての引数に欠損値を返します。正常に返された場合、すべての値の合計は 1 になります。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果

x=0.5;
y=−0.5;
z=1;
call softmax(x,y,z);
put x= y= z=;
	x=0.3314989604 y=0.1219516523 z=0.5465493873

CALL SORTC ルーチン

文字引数の値を並べ替えます。

カテゴリ: 並べ替え

操作: %SYSCALL マクロステートメントで呼び出されると、CALL SORTC の引数から引用符が削除されます。詳細については、“CALL ルーチンと%SYSCALL マクロステートメントを使用する” (9 ページ)を参照してください。

構文

CALL SORTC(*variable-1* <, …, *variable-n*>);

必須引数

変数
文字変数を指定します。

詳細

variable の値は、CALL SORTC ルーチンによって昇順に並べ替えられます。

比較

CALL SORTC ルーチンは文字変数に使用し、CALL SORTN ルーチンは数値変数に使用します。

サンプル

配列の文字変数を昇順に並べ替える例を次に示します。

```
data _null_;
array x(8) $10
('tweedledum' 'tweedledee' 'baboon' 'baby'
'humpty' 'dumpty' 'banana' 'babylon');
call sortc(of x(*));
put +3 x(*);
run;
```

SAS は次の出力をログに書き込みます。

```
 baboon baby babylon banana dumpty humpty tweedledee tweedledum
```

関連項目:

CALL ルーチン:

- “CALL SORTN ルーチン” (246 ページ)

CALL SORTN ルーチン

数値引数の値を並べ替えます。

カテゴリ: 並べ替え

構文

CALL SORTN(*variable-1* <, …, *variable-n*>);

必須引数

変数
数値変数を指定します。

詳細

variable の値は、CALL SORTN ルーチンによって昇順に並べ替えられます。

比較

CALL SORTN ルーチンは数値変数に使用し、CALL SORTC ルーチンは文字変数に使用します。

サンプル

次に、配列の数値変数を昇順に並べ替える例を示します。

```
data _null_;
array x(10) (0, ., .a, 1e-12, -1e-8, .z, -37, 123456789, 1e20, 42);
call sortn(of x(*));
put +3 x(*);
run;
```

SAS は次の出力をログに書き込みます。

```
. A Z -37 -1E-8 0 1E-12 42 123456789 1E20
```

関連項目:

CALL ルーチン:

- “CALL SORTC ルーチン” (245 ページ)

CALL STDIZE ルーチン

1 つ以上の変数の値を標準化します。

カテゴリ: 数学関数

操作: %SYSCALL マクロステートメントで呼び出されると、CALL STDIZE の引数から引用符が削除されます。詳細については、“CALL ルーチンと%SYSCALL マクロステートメントを使用する” (9 ページ)を参照してください。

構文

CALL STDIZE(<*option-1*,*option-2, …*,> *variable-1* <, *variable-2, …*>);

必須引数

変数

数値です。これらの値は、使用するメソッドに従って標準化されます。

オプション引数

option

大文字か小文字、またはその両方を使用した値の文字式を指定します。先頭と末尾の空白は無視されます。*option* には、次の 3 つのカテゴリがあります。

- standardization-options
- VARDEF-options
- miscellaneous-options

制限事項: 1 つの引数で複数のオプションは指定できないため、オプションごとに別々の引数を使用します。

ヒント: 文字式は末尾に等符号を付け、その後に数値の定数、変数または式である別の引数を続けることができます。

参照項目: 計算式およびその他の詳細については、*SAS/STAT 9.3 User's Guide* の PROC STDIZE を参照してください。CALL STDIZE で使用するオプションは、PROC STDIZE で使用するオプションと同じです。

standardization-options
変数の標準化に使用する場所メジャーとスケールメジャーの計算方法を指定します。次の標準化オプションを使用できます。

ABW= この後に、チューニング定数を指定する数値式である引数が続く必要があります。

AGK= この後に、クラスター内の分散の推定に含めるペアの比例を指定する数値式である引数が続く必要があります。

AHUBER= この後に、チューニング定数を指定する数値式である引数が続く必要があります。

AWAVE= この後に、チューニング定数を指定する数値式である引数が続く必要があります。

EUCLEN ユークリッドの長さを指定します。

IQR 四分位範囲を指定します。

L= この後に、L(p)またはミンコフスキー計量の計算で差が累乗される乗数を指定する 1 以上の値を指定した数値式である引数が続く必要があります。

MAD 中央値からの平均絶対偏差を指定します。

MAXABS 最大絶対値を指定します。

MEAN 算術平均(平均)を指定します。

MEDIAN ランクに従って順序付けられるデータセットの中間値を指定します。

MIDRANGE 範囲の中間点を指定します。

RANGE 値の範囲を指定します。

SPACING= この後に、スペーシングに含めるデータの比率を指定する数値式である引数が続く必要があります。

STD 標準偏差を指定します。

SUM 数値を加算して得られる結果を指定します。

USTD 無修正平方和に基づき、原点についての標準偏差を指定します。

VARDEF-options
分散の計算に使用する除数を指定します。VARDEF オプションには、次の値を指定できます。

DF 自由度を指定します。

N オブザベーション数を指定します。デフォルトは DF です。

miscellaneous-options
Miscellaneous オプションには、次の値を指定できます。

ADD= 標準化し、MULT=オプションの値で乗算した後に各値に追加する数値を指定する数値引数がこの後に続きます。デフォルト値は 0 です。

FUZZ= この後に、相対的あいまい度を指定する数値引数が続きます。

MISSING= この後に、欠損値のある変数に割り当てる値を指定する数値引数が続きます。

MULT= この後に、標準化後に各値を乗算する数値を指定する数値引数が続きます。デフォルト値は 1 です。

NORM 正規分布の標準偏差に対して一貫性が保たれるようにスケール推定量を正規化します。このオプションは、AGK=、IQR、MAD および SPACING=メソッドにのみ影響します。

PSTAT 場所メジャーとスケールメジャーの値を口グに書き込みます。

REPLACE 標準化されたデータの欠損値を値 0 で置き換えます(この値は標準化前の場所メジャーに対応します)。欠損値を他の値で置き換えるには、MISSING=オプションを参照してください。

SNORM 標準正規分布に対する期待値が約 1 となるようにスケール推定量を正規化します。このオプションは、SPACING=メソッドにのみ影響します。

詳細

CALL STDIZE ルーチンは、場所メジャーを減算し、スケールメジャーで除算することにより、数値変数である 1 つ以上の引数を変換します。さまざまな場所メジャーとスケールメジャーを使用できます。デフォルトの場所のオプションは MEAN、デフォルトのスケールのオプションは STD です。

さらに、標準化された各値を定数で乗算したり、定数を加算したりできます。最終出力値は、$result = add + mult * \left(\frac{(original - location)}{scale} \right)$ のようになります。

変数の説明を次に示します。

result
各変数に返される最終値を指定します。

add
加算する定数を指定します(ADD=オプション)。

mult
乗算する定数を指定します(MULT=オプション)。

original
オリジナル入力値を指定します。

location
場所メジャーを指定します。

scale
スケールメジャーを指定します。

欠損値は定数で置き換えることができます。MISSING=または REPLACE オプションを指定しない場合、欠損値がある変数は変更されません。ABW=、AHUBER=および AWAVE=メソッドの初期の推定法は MAD です。パーセント点は定義 5 を使用して計算されます。パーセント点計算の詳細については、*Base SAS プロシジャガイド*の SAS Elementary Statistics Procedures SAS Elementary Statistics Procedures を参照してください。

比較

CALL STDIZE ルーチンは、
SAS/STAT 製品の STDIZE プロシジャに似ています。ただし、CALL STDIZE ルーチンは主に SAS データセットの行の標準化に便利であるのに対し、STDIZE プロシジャは SAS データセットの列のみを標準化できます。詳細については、*SAS/STAT User's Guide* の PROC STDIZE を参照してください。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
retain x 1 y 2 z 3; call stdize(x,y,z); put x= y= z=;	x=-1 y=0 z=1
retain w 10 x 11 y 12 z 13; call stdize('iqr',w,x,y,z); put w= x= y= z=;	w=-0.75 x=-0.25 y=0.25 z=0.75
retain w . x 1 y 2 z 3; call stdize('range',w,x,y,z); put w= x= y= z=;	w=. x=0 y=0.5 z=1
retain w . x 1 y 2 z 3; call stdize('mult=',10,'missing=', -1,'range',w,x,y,z); put w= x= y= z=;	w=-1 x=0 y=5 z=10

CALL STREAMINIT ルーチン

後続の RAND 関数による乱数生成に使用するシード値を指定します。

カテゴリ: 乱数

構文

CALL STREAMINIT(*seed*);

必須引数

シード

整数のシード値です。

範囲: $seed < 2^{31} - 1$

ヒント: 正以外のシードを指定すると、CALL STREAMINIT は無視されます。後続の乱数生成は、システムクロックからシードを設定します。

詳細

乱数の再現可能なストリームを作成するには、RAND 乱数関数を呼び出す前に CALL STREAMINIT を指定します。CALL STREAMINIT ルーチンでシードを指定する前に RAND 関数を呼び出すと(または CALL STREAMINIT ルーチンで正以外のシード値を指定すると)、RAND 関数はシステムクロックへの呼び出しを使用してシードを設定します。各 DATA ステップでは、1 つの CALL STREAMINIT シードが受け入れられます。優先されるシード値は、最初の RAND 関数呼び出しが行われる前に指定されるシード値です。シード値の詳細については、“シード値” (11 ページ)を参照してください。

サンプル: 乱数の再現可能なストリームの作成

CALL STREAMINIT でシード値を指定し、RAND 関数で乱数の再現可能なストリームを作成する方法の例を次に示します。

```
data random;
call streaminit(123);
do i=1 to 10;
x1=rand('cauchy');
output;
end;
proc print data=random;
id i;
run;
```

画面 2.19 *CALL STREAMINIT でシード設定した数値文字列*

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

関連項目:

関数:

- “RAND 関数” (784 ページ)

CALL SYMPUT ルーチン

マクロ変数に DATA ステップ情報を割り当てます。

カテゴリ: マクロ

構文

CALL SYMPUT(*argument-1*,*argument-2*);

必須引数

argument-1
値に割り当てられるマクロ変数を識別する文字式を指定します。マクロ変数が存在しない場合は、ルーチンによって作成されます。

argument-2
割り当てられる値が含まれる文字定数、変数または式を指定します。

詳細

CALL SYMPUT ルーチンは、値が DATA ステップの情報であるマクロ変数を作成するか、既存のマクロ変数に DATA ステップ値を割り当てます。CALL SYMPUT の詳細については、*SAS マクロ言語: リファレンス*の"SYMPUT ルーチン"を参照してください。

関連項目:

関数:

- “SYMGET 関数” (876 ページ)

CALL SYMPUTX ルーチン

マクロ変数に値を割り当て、先頭と末尾の空白の両方を削除します。

カテゴリ: マクロ

構文

CALL SYMPUTX(*macro-variable*, *value* <,*symbol-table*>);

必須引数

macro-variable
次のいずれかの種類が表示されます。

- 引用符で囲まれた、SAS 名である文字列。
- 値が SAS 名の文字変数の名前。
- マクロ変数名を生成する文字式。この形式は、一連のマクロ変数を作成するのに便利です。

文字定数、変数または式。*name* の値から先頭と末尾の空白が削除され、その結果がマクロ変数の名前として使用されます。

value
文字または数値の定数、変数または式を指定します。*value* が数値の場合、値は BEST.出力形式を使用して文字列に変換され、SAS ログにメモは発行されません。先頭と末尾の空白が削除され、その結果の文字列がマクロ変数に割り当てられます。

オプション引数

symbol-table
文字定数、変数または式を指定します。*symbol-table* の値の大文字と小文字は区別されません。*symbol-table* の最初の空白以外の文字で、マクロ変数の保存先の記号テーブルを指定します。*symbol-table* の最初の非空白文字として有効な値は次のとおりです。

G　マクロ変数は、ローカル記号テーブルが存在する場合でも、グローバル記号テーブルに保存されます。

L　マクロ変数は、存在する最もローカルな記号テーブルに保存され、マクロの外側で使用する場合、このテーブルはグローバル記号テーブルになります。

F　マクロ変数が記号テーブルに存在する場合、CALL SYMPUTX はマクロ変数が存在する最もローカルな記号テーブルのバージョンを使用します。マクロ変数が存在しない場合、CALL SYMPUTX は最もローカルな記号テーブルに変数を保存します。

注: *symbol-table* を省略したり、*symbol-table* が空白の場合、CALL SYMPUTX は CALL SYMPUT ルーチンと同じ記号テーブルにマクロ変数を保存します。

詳細

CALL SYMPUTX は次の点を除き、CALL SYMPUT に似ています。

- 第 2 引数が数値のとき、CALL SYMPUTX は SAS ログにメモを書き込みません。ただし、CALL SYMPUT は、ログに数値が文字値に変換されたというメモを書き込みます。
- CALL SYMPUTX は、数値の第 2 引数を文字値に変換するとき、最大で 32 文字のフィールド幅を使用します。CALL SYMPUT は最大で 12 文字のフィールド幅を使用します。
- CALL SYMPUTX は両方の引数を左寄せし、末尾の空白を取り除きます。CALL SYMPUT は引数を左寄せせず、第 1 引数のみから末尾の空白を取り除きます。*name* の値の先頭に空白があると エラーが発生します。
- CALL SYMPUTX では、マクロ変数を保存する記号テーブルを指定できるのに対し、CALL SYMPUT では指定できません。

サンプル: CALL SYMPUTX を使用する

CALL SYMPUTX を使用した結果の例を次に示します。

```
data _null_;
call symputx(' items ', ' leading and trailing blanks removed ',
'lplace');
call symputx(' x ', 123.456);
run;
%put items=!&items!;
%put x=!&x!;
```

次の行が SAS ログに書き込まれます。

```
----+----1----+----2----+----3----+----4----+----5
items=!leading and trailing blanks removed!
x=!123.456!
```

関連項目:

関数:

- “SYMGET 関数” (876 ページ)

CALL ルーチン:

- “CALL SYMPUT ルーチン” (252 ページ)

CALL SYSTEM ルーチン

実行の動作環境コマンドをサブミットします。

カテゴリ: 特殊関数

操作: %SYSCALL マクロステートメントで呼び出されると、CALL SYSTEM の引数から引用符が削除されます。詳細については、“CALL ルーチンと%SYSCALL マクロステートメントを使用する” (9 ページ)を参照してください。

参照項目: “CALL SYSTEM Routine: Windows” in *SAS Companion for Windows*

“CALL SYSTEM Routine: UNIX” in *SAS Companion for UNIX Environments*

“CALL SYSTEM Routine: z/OS” in *SAS Companion for z/OS*

構文

CALL SYMPUT(*command*);

必須引数

command

引用符で囲まれたシステムコマンド(文字列)、値がシステムコマンドである式、または値が実行されるシステムコマンドである文字変数の名前のいずれかを指定します。

動作環境の情報

指定できる内容については、動作環境に関する SAS のドキュメントを参照してください。

制限事項: コマンドの長さは、末尾の空白を含めて 1024 文字以内にする必要があります。

詳細

CALL SYSTEM ルーチンの動作は、X コマンド、X ステートメントおよび SYSTEM 関数に似ています。条件付きで実行でき、式を引数として受け入れ、実行時に実行されるため、特定の状況で便利です。

関連項目:

関数:

- “SYSTEM 関数” (884 ページ)

CALL TANH ルーチン

双曲線正接を返します。

カテゴリ: 数学関数

構文

CALL TANH(*argument*<, *argument,...*>);

必須引数

引数
数値です。

制限事項: CALL TANH ルーチンでは、変数を有効な引数としてのみ使用できます。これらの引数は CALL ルーチンで更新できないため、定数または SAS 式は使用しないでください。

詳細

サブルーチン TANH は、各引数をその引数の双曲線正接で置き換えます。たとえば、x_j に置き換えられます。

$$tanh(x_j) = \frac{\varepsilon^{x_j} - \varepsilon^{-x_j}}{\varepsilon^{x_j} + \varepsilon^{-x_j}}$$

欠損値を含む引数が 1 つでもある場合、CALL TANH は欠損値をすべての引数で返します。

サンプル: 例

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果

x=0.5;
y=-0.5;
call tanh(x,y);
put x= y=;
	x=0.4621171573 y=-0.462117157

関連項目:

関数:

- “TANH 関数” (886 ページ)

CALL VNAME ルーチン

指定した変数の値として変数名を割り当てます。

カテゴリ: 変数制御

構文

CALL VNAME(*variable-1*,*variable-2*);

必須引数

variable-1
SAS 変数を指定します。

variable-2
SAS 文字変数を指定します。SAS 変数名には最大で 32 文字まで使用できるため、*variable-2* の長さを少なくとも 32 にする必要があります。

詳細

CALL VNAME ルーチンは、*variable-1* 変数の名前を *variable-2* 変数の値として割り当てます。

サンプル: CALL VNAME ルーチンを使用する

この例では、CALL VNAME ルーチンで配列参照を使用し、データセット OLD のすべての変数の名前を返します。

```
data new(keep=name);
set old;
/* all character variables in old */
array abc{*} _character_;
/* all numeric variables in old */
array def{*} _numeric_;
/* name is not in either array */
length name $32;
do i=1 to dim(abc);
/* get name of character variable */
call vname(abc{i},name);
```

```
/* write name to an observation */
output;
end;
do j=1 to dim(def);
/* get name of numeric variable */
call vname(def[j],name);
/* write name to an observation */
output;
end;
stop;
run;
```

関連項目:

関数:

- “VNAME 関数” (941 ページ)
- “VNAMEX 関数” (942 ページ)

CALL VNEXT ルーチン

DATA ステップで使用される変数の名前、種類および長さを返します。

カテゴリ: 変数情報

構文

CALL VNEXT(*varname* <,*vartype*<, *varlength*>>);

必須引数

varname

CALL VNEXT ルーチンで更新される文字変数です。次のルールが適用されます。

- *varname* の入力値が空白の場合、*varname* で返される値は DATA ステップの変数のリストにある最初の変数の名前になります。
- CALL VNEXT ルーチンを DATA ステップで初めて実行する場合、*varname* で返される値は DATA ステップの変数のリストにある最初の変数の名前になります。

前述の条件のどちらも存在しない場合、*varname* の入力値は無視されます。CALL VNEXT ルーチンを実行するたびに *varname* で返される値は、リストの次の変数の名前になります。

DATA ステップのすべての変数の名前が返されると、*varname* で返される値は空白になります。

オプション引数

vartype

入力値が無視される文字変数です。返される値は"N"または"C"です。次のルールが適用されます。

- *varname* で返される値が数値変数の名前の場合、*vartype* で返される値は "N"です。
- *varname* で返される値が文字変数の名前の場合、*vartype* で返される値は "C"です。
- *varname* で返される値が空白の場合、*vartype* で返される値も空白です。

varlength
数値変数です。*varlength* の入力値は無視されます。

返される値は、*varname* で名前が返される変数の長さです。*varname* で返される値が空白の場合、*varlength* で返される値はゼロです。

詳細

CALL VNEXT ルーチンで返される変数名には、_N_や_ERROR_
などの自動変数があります。DATA ステップにBY ステートメントが含まれている場合、CALL VNEXT で返される名前には FIRST.*variable* と LAST.*variable* の名前が含まれます。CALL VNEXT は、CALL VNEXT への引数として使用される変数の名前も返します。

注: CALL VNEXT で返される変数名の順序は、使用している SAS のリリースと動作環境によって異なる場合があります。

サンプル: CALL VNEXT ルーチンを使用する

CALL VNEXT ルーチンを使用した結果の例を次に示します。

```
data test;
x=1;
y='abc';
z=.;
length z 5;
run;
data attributes;
set test;
by x;
input a b $ c;
length name $32 type $3;
name=' ';
length=666;
do i=1 to 99 until(name=' ');
call vnext(name,type,length);
put i= name @40 type= length=;
end;
this_is_a_long_variable_name=0;
datalines;
1 q 3
;
```

ログ 2.7 CALL VNEXT ルーチンの SAS ログ出力の一部

```
i=1 x type=N length=8
i=2 y type=C length=3
i=3 z type=N length=5
i=4 FIRST.x type=N length=8
i=5 LAST.x type=N length=8
i=6 a type=N length=8
i=7 b type=C length=8
i=8 c type=N length=8
i=9 name type=C length=32
i=10 type type=C length=3
i=11 length type=N length=8
i=12 i type=N length=8
i=13 this_is_a_long_variable_name type=N length=8
i=14 _ERROR_ type=N length=8
i=15 _N_ type=N length=8
i=16 type= length=0
```

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: 文字関数

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

ヒント: DBCS に相当する関数は、*SAS 各国語サポート(NLS): リファレンスガイド*の KSTRCAT です。

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*. 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

DATA ステップで、まだ長さが割り当てられていない変数に CAT 関数から値が返される場合、その変数には 200 バイトの長さが設定されます。まだ長さが割り当てられていない変数に連結演算子(||)から値が返される場合、その変数の長さは結合されている値の長さの合計になります。

返される変数の長さ: 特殊なケース

CAT 関数は、変数または一時バッファに値を返します。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)

- 最大 32767 文字(WHERE 句以外の DATA ステップ)
- 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に返す場合、バッファの長さは呼び出し環境によって異なります。バッファの値は 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。
- 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。

CAT 関数は、BEST*w*. 形式で数値の出力形式を設定した後で先頭および末尾の空白を数値引数から削除します。

比較

CAT、CATS、CATT、CATX 関数の結果は、*一般的に*連結演算子(||)と TRIM および LEFT 関数の特定の組み合わせによって生成される結果と同じになります。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" (259 ページ)を参照してください。

CAT、CATS、CATT、CATX 関数は TRIM および LEFT 関数よりも速く、変数リストをサポートする呼び出し環境では、変数リストに対応する OF 構文で使用できます。

次の表に、CAT、CATS、CATT、CATX 関数と同等のコードを示します。X1 ~ X4 の変数は文字変数を指定し、SP は空白やカンマなどの区切り文字を示します。

関数	同等のコード
CAT(OF X1-X4)	X1||X2||X3||X4
CATS(OF X1-X4)	TRIM(LEFT(X1))||TRIM(LEFT(X2))||TRIM(LEFT(X3))|| TRIM(LEFT(X4))
CATT(OF X1-X4)	TRIM(X1)||TRIM(X2)||TRIM(X3)||TRIM(X4)
CATX(SP, OF X1-X4)	TRIM(LEFT(X1))||SP||TRIM(LEFT(X2))||SP|| TRIM(LEFT(X3))||SP||TRIM(LEFT(X4))

サンプル

次の例では、CAT 関数がどのように文字列を連結するのかを示します。

```
data _null_;
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 $char.;
run;
```

SAS は次の行をログに書き込みます。

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7
The 2012 Olympic Arts Festi val included works by D ale Chihuly.
```

関連項目:

関数:

- “CATQ 関数” (261 ページ)
- “CATS 関数” (265 ページ)
- “CATT 関数” (267 ページ)
- “CATX 関数” (270 ページ)

CALL ルーチン:

- “CALL CATS ルーチン” (157 ページ)
- “CALL CATT ルーチン” (158 ページ)
- “CALL CATX ルーチン” (160 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: 文字関数

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 または'
CATQ は、文字列に引用符を追加するときに一重引用符を使用します。

2 または”
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 または A
すべての項目引数に引用符を追加します。

b または 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 または C
区切り文字としてカンマを使用します。

d または D
指定の区切り文字引数があることを示します。

h または H
区切り文字として水平タブを使用します。

m または 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 または 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 または 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 または 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 または 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 または 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*. 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。
- 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。

CATQ が値を一時バッファに返す場合、バッファの長さは呼び出し環境によって異なります。バッファの値は 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。次のルールが適用されます。

- CATQ は、アンパサンド(&)やパーセント(%)記号を含む文字列、または一重引用符より二重引用符の方が多い文字列には、一重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。

サンプル

次の例では、CATQ 関数がどのように文字列を連結するのかを示します。

```
options ls=110;
data _null_;
result1=CATQ(' ',
'noblanks',
'one blank',
12345,
' lots of blanks ');
result2=CATQ('CS',
'Period (.) ',
'Ampersand (&) ',
'Comma (,) ',
'Double quotation marks (") ',
' Leading Blanks');
result3=CATQ('BCQT',
'Period (.) ',
'Ampersand (&) ',
'Comma (,) ',
```

```
'Double quotation marks (") ',
' Leading Blanks');
result4=CATQ('ADT',
'#=#',
'Period (.) ',
'Ampersand (&) ',
'Comma (,) ',
'Double quotation marks (") ',
' Leading Blanks');
result5=CATQ('N',
'ABC_123 ',
'123 ',
'ABC 123');
put (result1-result5) (=/);
run;
```

SAS は次の出力を口グに書き込みます。

```
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” (259 ページ)
- “CATS 関数” (265 ページ)
- “CATT 関数” (267 ページ)
- “CATX 関数” (270 ページ)

CALL ルーチン:

- “CALL CATS ルーチン” (157 ページ)
- “CALL CATT ルーチン” (158 ページ)
- “CALL CATX ルーチン” (160 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: 文字関数

制限事項: I18N レベル 0 の関数は 1 バイト文字セット(SBCS)でのみ使用するように設計されています。

構文

CATS(*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*. 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

返される変数の長さ
DATA ステップで、まだ長さが割り当てられていない変数に CATS 関数から値が返される場合、その変数には 200 バイトの長さが設定されます。まだ長さが割り当てられていない変数に連結演算子(||)から値が返される場合、その変数の長さは結合されている値の長さの合計になります。

返される変数の長さ: 特殊なケース
CATS 関数は、変数または一時バッファに値を返します。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)
- 最大 32767 文字(WHERE 句以外の DATA ステップ)
- 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に返す場合、バッファの長さは呼び出し環境によって異なります。バッファの値は 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。
- 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。

CATS 関数は、BEST*w*. 形式で数値の出力形式を設定した後で先頭および末尾の空白を数値引数から削除します。

比較

CAT、CATS、CATT、CATX 関数の結果は、*一般的に*連結演算子(||)と TRIM および LEFT 関数の特定の組み合わせによって生成される結果と同じになります。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" (266 ページ)を参照してください。

CAT、CATS、CATT、CATX 関数は TRIM および LEFT 関数よりも速く、変数リストをサポートする呼び出し環境では、変数リストに対応する OF 構文で使用できます。

次の表に、CAT、CATS、CATT、CATX 関数と同等のコードを示します。X1 ~ X4 の変数は文字変数を指定し、SP は空白やカンマなどの区切り文字を示します。

関数	同等のコード
CAT(OF X1-X4)	X1||X2||X3||X4
CATS(OF X1-X4)	TRIM(LEFT(X1))||TRIM(LEFT(X2))||TRIM(LEFT(X3))|| TRIM(LEFT(X4))
CATT(OF X1-X4)	TRIM(X1)||TRIM(X2)||TRIM(X3)||TRIM(X4)
CATX(SP, OF X1-X4)	TRIM(LEFT(X1))||SP||TRIM(LEFT(X2))||SP|| TRIM(LEFT(X3))||SP||TRIM(LEFT(X4))

サンプル

次の例では、CATS 関数がどのように文字列を連結するのかを示します。

```
data _null_;
x=' The Olym';
y='pic Arts Festi';
z=' val includes works by D ';
a='ale Chihuly.';
result=cats(x,y,z,a);
put result $char.;
run;
```

次の行が SAS ログに書き込まれます。

```
----+----1----+----2----+----3----+----4----+----5----+----6
The Olympic Arts Festival includes works by Dale Chihuly.
```

関連項目:

関数:

- “CAT 関数” (259 ページ)
- “CATQ 関数” (261 ページ)
- “CATT 関数” (267 ページ)
- “CATX 関数” (270 ページ)

CALL ルーチン:

- “CALL CATS ルーチン” (157 ページ)
- “CALL CATT ルーチン” (158 ページ)
- “CALL CATX ルーチン” (160 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: 文字関数

制限事項: I18N レベル 0 の関数は 1 バイト文字セット(SBCS)でのみ使用するように設計されています。

構文

CATT(*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*. 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

DATA ステップで、まだ長さが割り当てられていない変数に CATT 関数から値が返される場合、その変数には 200 バイトの長さが設定されます。まだ長さが割り当てられていない変数に連結演算子(||)から値が返される場合、その変数の長さは結合されている値の長さの合計になります。

返される変数の長さ: 特殊なケース

CATT 関数は、変数または一時バッファに値を返します。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)
- 最大 32767 文字(WHERE 句以外の DATA ステップ)
- 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に返す場合、バッファの長さは呼び出し環境によって異なります。バッファの値は 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。
- 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。

CATT 関数は、BEST*w*. 形式で数値の出力形式を設定した後で先頭および末尾の空白を数値引数から削除します。

比較

CAT、CATS、CATT、CATX 関数の結果は、*一般的に*連結演算子(||)と TRIM および LEFT 関数の特定の組み合わせによって生成される結果と同じになります。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” (268 ページ)を参照してください。

CAT、CATS、CATT、CATX 関数は TRIM および LEFT 関数よりも速く、変数リストをサポートする呼び出し環境では、変数リストに対応する OF 構文で使用できます。

次の表に、CAT、CATS、CATT、CATX 関数と同等のコードを示します。X1 ~ X4 の変数は文字変数を指定し、SP は空白やカンマなどの区切り文字を示します。

関数	同等のコード
CAT(OF X1-X4)	X1||X2||X3||X4
CATS(OF X1-X4)	TRIM(LEFT(X1))||TRIM(LEFT(X2))||TRIM(LEFT(X3))|| TRIM(LEFT(X4))
CATT(OF X1-X4)	TRIM(X1)||TRIM(X2)||TRIM(X3)||TRIM(X4)
CATX(SP, OF X1-X4)	TRIM(LEFT(X1))||SP||TRIM(LEFT(X2))||SP|| TRIM(LEFT(X3))||SP||TRIM(LEFT(X4))

サンプル

次の例では、CATT 関数がどのように文字列を連結するのかを示します。

```
data _null_;
x=' The Olym';
y='pic Arts Festi';
z=' val includes works by D ';
a='ale Chihuly.';
result=catt(x,y,z,a);
put result $char.;
run;
```

次の行が SAS ログに書き込まれます。

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7
The Olympic Arts Festi val includes works by Dale Chihuly.
```

関連項目:

関数:

- “CAT 関数” (259 ページ)
- “CATQ 関数” (261 ページ)
- “CATS 関数” (265 ページ)
- “CATX 関数” (270 ページ)

CALL ルーチン:

- “CALL CATS ルーチン” (157 ページ)
- “CALL CATT ルーチン” (158 ページ)

- “CALL CATX ルーチン” (160 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: 文字関数

制限事項: I18N レベル 0 の関数は 1 バイト文字セット(SBCS)でのみ使用するように設計されています。

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*. 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” (270 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

DATA ステップで、まだ長さが割り当てられていない変数に CATX 関数から値が返される場合、その変数には 200 バイトの長さが設定されます。まだ長さが割り当てられていない変数に連結演算子(||)から値が返される場合、その変数の長さは結合されている値の長さの合計になります。

返される変数の長さ: 特殊なケース

CATX 関数は、変数または一時バッファに値を返します。CATX 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)
- 最大 32767 文字(WHERE 句以外の DATA ステップ)
- 最大 65534 文字(CATX がマクロプロセッサから呼び出される場合)

CATX が値を一時バッファに返す場合、バッファの長さは呼び出し環境によって異なります。バッファの値は CATX 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。
- 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。

比較

CAT、CATS、CATT、CATX 関数の結果は、*一般的に*連結演算子(||)と TRIM および LEFT 関数の特定の組み合わせによって生成される結果と同じになります。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" (270 ページ)を参照してください。

CAT、CATS、CATT、CATX 関数は TRIM および LEFT 関数よりも速く、変数リストをサポートする呼び出し環境では、変数リストに対応する OF 構文で使用できます。

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。"サンプル 2: 欠損値のある文字列の連結" (272 ページ)を参照してください。

次の表に、CAT、CATS、CATT、CATX 関数と同等のコードを示します。X1 ~ X4 の変数は文字変数を指定し、SP は空白やカンマなどの区切り文字を示します。

関数	同等のコード
CAT(OF X1-X4)	X1||X2||X3||X4
CATS(OF X1-X4)	TRIM(LEFT(X1))||TRIM(LEFT(X2))||TRIM(LEFT(X3))|| TRIM(LEFT(X4))
CATT(OF X1-X4)	TRIM(X1)||TRIM(X2)||TRIM(X3)||TRIM(X4)
CATX(SP, OF X1-X4)	TRIM(LEFT(X1))||SP||TRIM(LEFT(X2))||SP|| TRIM(LEFT(X3))||SP||TRIM(LEFT(X4))

サンプル

サンプル 1: 欠損値のない文字列の連結

次の例では、CATX 関数がどのように欠損値のない文字列を連結するのかを示します。

```
data _null_;
separator='%%$%%';
x='The Olympic ';
```

```
y=' Arts Festival ';
z=' includes works by ';
a='Dale Chihuly.';
result=catx(separator,x,y,z,a);
put result $char.;
run;
```

次の行が SAS ログに書き込まれます。

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7
The Olympic%%$%%Arts Festival%%$%%includes works by%%$%%Dale Chihuly.
```

サンプル 2: 欠損値のある文字列の連結

次の例では、CATX 関数がどのように欠損値のある文字列を連結するのかを示します。

```
data one;
length x1-x4 $1;
input x1-x4;
datalines;
A B C D
E . F G
H . . J
;
run;
data two;
set one;
SP='^';
test1=catx(sp, of x1-x4);
test2=trim(left(x1)) || sp || trim(left(x2)) || sp || trim(left(x3)) || sp ||
trim(left(x4));
run;

proc print data=two;
run;
```

画面 2.20 *欠損値で CATX を使用する*

The SAS System

Obs	x1	x2	x3	x4	SP	test1	test2
1	A	B	C	D	^	A^B^C^D	A^B^C^D
2	E		F	G	^	E^F^G	E^ ^F^G
3	H			J	^	H^J	H^ ^ ^J

関連項目:

関数:

- “CAT 関数” (259 ページ)
- “CATQ 関数” (261 ページ)

- “CATS 関数” (265 ページ)
- “CATT 関数” (267 ページ)

CALL ルーチン:

- “CALL CATS ルーチン” (157 ページ)
- “CALL CATT ルーチン” (158 ページ)
- “CALL CATX ルーチン” (160 ページ)

CDF 関数

累積確率分布の値を返します。

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” (778 ページ)を参照してください。

構文

CDF (*distribution*,*quantile*<,*parm-1, ... ,parm-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
ベルヌーイ	BERNOULLI
ベータ	BETA
二項	BINOMIAL
コーシー	CAUCHY
χ^2 乗	CHISQUARE
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
ポアソン	POISSON
T	T
Tweedie (p. 284)	TWEEDIE
一様	UNIFORM
逆ガウス(Wald)	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。

等量分類
ランダム変数の値を指定する数値定数、変数または式です。

オプション引数

parm-1, … ,parm-k
特定の分布に適した*形状*、*位置*または*尺度*パラメータを指定する定数、変数または式です。

参照項目: これらのパラメータの詳細については、“詳細” (274 ページ)を参照してください。

詳細

CDF 関数は、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。

Bernoulli 分布

CDF('BERNOULLI',*x*,*p*)

引数

x
数値のランダム変数です。

p

数値の成功確率です。

範囲: $0 \le p \le 1$

詳細

ベルヌーイ分布の CDF 関数は、ベルヌーイ分布(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。式は次のとおりです。

$$CDF('BERN', x, p) = \begin{cases} 0 & x < 0 \\ 1 - p & 0 \le x < 1 \\ 1 & x \ge 1 \end{cases}$$

注: この分布には、*位置*または*尺度*パラメータはありません。

ベータ分布

CDF('BETA',*x*,*a*,*b*<,*l*,*r*>)

引数

x

数値のランダム変数です。

a

数値の形状パラメータです。

範囲: $a > 0$

b

数値の形状パラメータです。

範囲: $b > 0$

l

数値の左位置パラメータです。

デフォルト: 0

r

右位置パラメータです。

デフォルト: 1

範囲: $r > l$

詳細

ベータ分布の CDF 関数は、ベータ分布(形状パラメータは 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。

$$CDF('BETA', x, a, b, l, r) = \begin{cases} 0 & x \le l \\ \dfrac{1}{\beta(a, b)} \displaystyle\int_{l}^{x} \dfrac{(v - l)^{a-1}(r - v)^{b-1}}{(r - l)^{a+b-1}} dv & l < x \le r \\ 1 & x > r \end{cases}$$

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。

$$\Gamma(a) = \int_0^{\infty} x^{a-1} e^{-x} dx$$

二項分布

CDF('BINOMIAL',*m*,*p*,*n*)

引数

m

成功数を数える整数のランダム変数です。

範囲: $m = 0, 1, ...$

p

数値の成功確率です。

範囲: $0 \le p \le 1$

n

独立ベルヌーイ試行数を数える整数のパラメータです。

範囲: $n = 0, 1, ...$

詳細

二項分布の CDF 関数は、二項分布(パラメータは 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。式は次のとおりです。

$$CDF('BINOM', m, p, n) = \begin{cases} 0 & m < 0 \\ \sum_{j=0}^{m} \binom{N}{j} p^j (1-p)^{n-j} & 0 \le m \le n \\ 1 & m > n \end{cases}$$

注: 二項分布には、*位置*または*尺度*パラメータはありません。

Cauchy 分布

CDF('CAUCHY',*x*<,*θ*><,*λ*>)

引数

x

数値のランダム変数です。

θ

数値の位置パラメータです。

デフォルト: 0

λ

数値の尺度パラメータです。

デフォルト: 1

範囲: $\lambda > 0$

詳細

コーシー分布の CDF 関数は、コーシー分布(位置パラメータは θ、尺度パラメータは 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。式は次のとおりです。

$$CDF('CAUCHY', x, \theta, \lambda) = \frac{1}{2} + \frac{1}{\pi} tan^{-1}\left(\frac{x-\theta}{\lambda}\right)$$

カイ 2 乗分布

CDF('CHISQUARE',*x*,*df* <,*nc*>)

引数

x
数値のランダム変数です。

df
数値の自由度パラメータです。

範囲: $df > 0$

nc
任意の数値の非心度パラメータです。

範囲: $nc \geq 0$

詳細

χ2 乗分布の CDF 関数は、χ2 乗分布(自由度は *df*、非心度パラメータは *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 χ2 乗分布から値が返されます。次の式では、$v = df$ および $\lambda = nc$となります。次の式は、χ2 乗分布の CDF 関数を表しています。

$$CDF('CHISQ', x, v, \lambda) = \begin{cases} 0 & x < 0 \\ \sum_{j=0}^{\infty} e^{-\frac{\lambda}{2}} \frac{\left(\frac{\lambda}{2}\right)^j}{j!} P_c(x, v+2j) & x \geq 0 \end{cases}$$

この式では、$P_c(.,.)$は、心度 χ2 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。

$$P_g(y, b) = \frac{1}{\Gamma(b)} \int_0^y e^{-v} v^{b-1} dv$$

指数分布

CDF('EXPONENTIAL',*x* <,*λ*>)

引数

x
数値のランダム変数です。

λ
尺度パラメータです。

デフォルト: 1

範囲: $\lambda > 0$

詳細

指数分布の CDF 関数は、指数分布(尺度パラメータは 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。式は次のとおりです。

$$CDF('EXPO', x, \lambda) = \begin{cases} 0 & x < 0 \\ 1 - e^{-\frac{x}{\lambda}} & x \geq 0 \end{cases}$$

F 分布

CDF('F',*x*,*ndf*,*ddf* <,*nc*>)

引数

x
数値のランダム変数です。

ndf
数値の分子の自由度パラメータです。
範囲: $ndf > 0$

ddf
数値の分母の自由度パラメータです。
範囲: $ddf > 0$

nc
数値の非心度パラメータです。
範囲: $nc \geq 0$

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。次の式では、$v_1 = ndf$、$v_2 = ddf$ および $\lambda = nc$となります。次の式は、*F* 分布の CDF 関数を表しています。

$$CDF('F',\ x,\ v_1,\ v_2,\ \lambda) = \begin{cases} 0 & x < 0 \\ \sum_{j=0}^{\infty} e^{-\frac{\lambda}{2}} \frac{\left(\frac{\lambda}{2}\right)^j}{j!} P_F(x,\ v_1 + 2j,\ v_2) & x \geq 0 \end{cases}$$

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*パラメータはありません。

ガンマ分布

CDF('GAMMA',*x*,*a*<,*λ*>)

引数

x
数値のランダム変数です。

a
数値の形状パラメータです。
範囲: $a > 0$

λ
数値の尺度パラメータです。
デフォルト: 1
範囲: $\lambda > 0$

詳細

ガンマ分布の CDF 関数は、ガンマ分布(形状パラメータは a、尺度パラメータは 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。式は次のとおりです。

$$CDF('GAMMA', x, a, \lambda) = \begin{cases} 0 & x < 0 \\ \frac{1}{\lambda^a \Gamma(a)} \int_0^x v^{a-1} e^{-\frac{v}{\lambda}} dv & x \geq 0 \end{cases}$$

一般化 Poisson 分布

CDF ('GENPOISSON', x, θ, η)

引数

x

整数のランダム変数です。

θ

形状パラメータを指定します。

範囲: ≤5 かつ>0

η

形状パラメータを指定します。

範囲: ≥0 かつ<0.95

ヒント: η =0 の場合、平均と分散が θ のポアソン分布になります。η>0 の場合、平均は $\theta \div (1 - \eta)$ で、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。

$$f(x; \theta, \eta) = \theta(\theta + \eta x)^{x-1} e^{-\theta - \eta x} / x!, \quad x = 0, 1, 2, \ldots, \qquad \theta > 0,\ 0 \leq \eta < 1$$

η =0 の場合、一般化 Poisson 分布は、形状パラメータが θ の標準ポアソン分布になります。

幾何分布

CDF('GEOMETRIC',m,p)

引数

m

失敗数を示す数値のランダム変数です。

範囲: $m = 0, 1, \ldots$

p

数値の成功確率です。

範囲: $0 \leq p \leq 1$

詳細

幾何分布の CDF 関数は、幾何分布(パラメータは 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。式は次のとおりです。

$$CDF('GEOM', m, p) = \begin{cases} 0 & m < 0 \\ 1 - (1 - p)^{(m+1)} & m \geq 0 \end{cases}$$

注: この分布には、*位置*または*尺度*パラメータはありません。

超幾何分布

CDF('HYPER',*x*,*N*,*R*,*n* <,*o*>)

引数

x
整数のランダム変数です。

N
整数の母集団サイズパラメータです。
範囲: $N = 1, 2, ...$

R
対象カテゴリの整数の項目数です。
範囲: $R = 0, 1, ..., N$

n
整数のサンプルサイズパラメータです。
範囲: $n = 1, 2, ..., N$

o
任意の数値のオッズ比パラメータです。
範囲: $o > 0$

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。式は次のとおりです。

$$CDF('HYPER', x, N, R, n, o) = \begin{cases} 0 & x < max(0, R + n - N) \\ \dfrac{\sum_{i=0}^{x} \binom{R}{i}\binom{N-R}{n-i} o^i}{\sum_{j=max(0,R+n-N)}^{min(R,n)} \binom{R}{j}\binom{N-R}{n-j} o^j} & max(0, R + n - N) \le x \le min(R, n) \\ 1 & x > min(R, n) \end{cases}$$

Laplace 分布

CDF('LAPLACE',*x*<,*θ*,*λ*>)

引数

x
数値のランダム変数です。

θ
数値の位置パラメータです。
デフォルト: 0

λ
数値の尺度パラメータです。
デフォルト: 1
範囲: $\lambda > 0$

詳細

Laplace 分布の CDF 関数は、Laplace 分布(位置パラメータは θ、尺度パラメータは 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。式は次のとおりです。

$$CDF('LAPLACE', x, \theta, \lambda) = \begin{cases} \frac{1}{2} e^{\frac{(x-\theta)}{\lambda}} & x < \theta \\ 1 - \frac{1}{2} e^{\left(-\frac{(x-\theta)}{\lambda}\right)} & x \geq \theta \end{cases}$$

ロジスティック分布

CDF('LOGISTIC',*x*<,*θ*,*λ*>)

引数

x

数値のランダム変数です。

θ

数値の位置パラメータです。

デフォルト: 0

λ

数値の尺度パラメータです。

デフォルト: 1

範囲: λ > 0

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(位置パラメータは θ、尺度パラメータは 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。式は次のとおりです。

$$CDF('LOGISTIC', x, \theta, \lambda) = \frac{1}{1 + e^{\left(-\frac{x-\theta}{\lambda}\right)}}$$

対数正規分布

CDF('LOGNORMAL',*x*<,*θ*,*λ*>)

引数

x

数値のランダム変数です。

θ

数値の対数尺度パラメータを指定します (exp(θ)は尺度パラメータ)。

デフォルト: 0

λ

数値の形状パラメータを指定します。

デフォルト: 1

範囲: λ > 0

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は θ、形状パラメータは 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。式は次のとおりです。

$$CDF('LOGN',\ x,\ \theta,\ \lambda) = \begin{cases} 0 & x \le 0 \\ \frac{1}{\lambda\sqrt{2\pi}} \int_{-\infty}^{\log(x)} e^{\left(-\frac{(v-\theta)^2}{2\lambda^2}\right)} dv & x > 0 \end{cases}$$

負数二項分布

CDF('NEGBINOMIAL',*m*,*p*,*n*)

引数

m

失敗数を数える正の整数のランダム変数です。

範囲: $m = 0, 1, \ldots$

p

数値の成功確率です。

範囲: $0 \le p \le 1$

n

成功数を数える数値です。

範囲: $n > 0$

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。式は次のとおりです。

$$CDF('NEGB',\ m,\ p,\ n) = \begin{cases} 0 & m < 0 \\ p^n \sum_{j=0}^{m} \binom{n+j-1}{n-1} (1-p)^j & m \ge 0 \end{cases}$$

注: 負の二項分布には、*位置*または*尺度*パラメータはありません。

正規分布

CDF('NORMAL',*x*<,*θ*,*λ*>)

引数

x

数値のランダム変数です。

θ

数値の位置パラメータです。

デフォルト: 0

λ

数値の尺度パラメータです。

デフォルト: 1

範囲: $\lambda > 0$

詳細

正規分布の CDF 関数は、正規分布(位置パラメータは θ、尺度パラメータは 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。式は次のとおりです。

$$CDF('NORMAL',\ x,\ \theta,\ \lambda) = \frac{1}{\lambda\sqrt{2\pi}} \int_{-\infty}^{x} e^{\left(-\frac{(v-\theta)^2}{2\lambda^2}\right)} dv$$

正規混合分布

CDF('NORMALMIX',*x*,*n*,*p*,*m*,*s*)

引数

x

数値のランダム変数です。

n

混合の整数です。

範囲: $n = 1, 2, ...$

p

n 個の母集団 $p_1, p_2, \ldots, p_n$です。ここで、$\sum_{i=1}^{i=n} p_i = 1$ となります。

範囲: $p = 0, 1, ...$

m

n 個の平均 $m_1, m_2, \ldots, m_n$ です。

s

n 個の標準偏差 $s_1, s_2, \ldots, s_n$ です。

範囲: $s > 0$

詳細

正規混合分布の CDF 関数は、正規混合分布からオブザベーションが x 以下となる確率を返します。式は次のとおりです。

$$CDF('NORMALMIX', x, n, p, m, s) = \sum_{i=1}^{i=n} p_i \, CDF('NORMAL', x, m_i, s_i)$$

注: 正規混合分布には、*位置*または*尺度*パラメータはありません。

パレート分布

CDF('PARETO',*x*,*a*<,*k*>)

引数

x

数値のランダム変数です。

a

数値の形状パラメータです。

範囲: $a > 0$

k

数値の尺度パラメータです。

デフォルト: 1

範囲: $k > 0$

詳細

パレート分布の CDF 関数は、パレート分布(形状パラメータは a、尺度パラメータは 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。式は次のとおりです。

$$CDF('PARETO', x, a, k) = \begin{cases} 0 & x < k \\ 1 - \left(\frac{k}{x}\right)^a & x \geq k \end{cases}$$

Poisson 分布

CDF('POISSON',*n*,*m*)

引数

n

整数のランダム変数です。

範囲: $n = 0, 1, ...$

m

数値の平均パラメータです。

範囲: $m > 0$

詳細

ポアソン分布の CDF 関数は、ポアソン分布(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。式は次のとおりです。

$$CDF('POISSON', n, m) = \begin{cases} 0 & n < 0 \\ \sum_{i=0}^{n} e^{-m}\frac{m^i}{i!} & n \geq 0 \end{cases}$$

注: ポアソン分布には、*位置*または*尺度*パラメータはありません。

T 分布

CDF('T',*t*,*df* <,*nc*>)

引数

t

数値のランダム変数です。

df

数値の自由度パラメータです。

範囲: $df > 0$

nc

任意の数値の非心度パラメータです。

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。次の式では、$\nu = df$ および $\delta = nc$です。式は次のとおりです。

$$CDF('T', t, \nu, \delta) = \frac{1}{2^{(\nu/2-1)}\Gamma\left(\frac{\nu}{2}\right)}\int_0^{\infty} x^{\nu-1} e^{-\frac{1}{2}x^2} \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\frac{tx}{\sqrt{\nu}}} e^{-\frac{1}{2}(u-\delta)^2} du dx$$

注: *T* 分布には、*位置*または*尺度*パラメータはありません。

Tweedie 分布

CDF ('TWEEDIE', *y*, *p*<,μ, φ>)

引数

y

ランダム変数です。

範囲: $y \geq 0$

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
累乗パラメータです。
範囲: $p \geq 1$
注: この引数は必須です。

μ
平均です。
デフォルト: 1
範囲: μ>0

φ
分散パラメータです。
デフォルト: 1
範囲: φ>0

詳細

ツウィーディ分布の CDF 関数は、式 $variance = \phi \times \mu^{p}$に関する分散と平均を使用して拡張分散モデルを返します。

式は次のとおりです。

$$\int_0^y \frac{1}{y} \sum_{j=1}^{\infty} \left(\frac{y^{-j\alpha}(p-1)^{j\alpha}}{\phi^{j(1-\alpha)}(2-p)^{j} j! \, \Gamma(-j\alpha)} \right) e^{\left(\frac{1}{\phi} \left(y \frac{\mu^{1-p}-1}{1-p} - \frac{\mu^{2-p}-1}{2-p} \right) \right)} dy$$

前述の式には次の関係が適用されます。

$$\alpha = \frac{2-p}{1-p}$$

注: 計算されたツウィーディの確率の精度は、パラメータ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。

一様分布

CDF('UNIFORM',*x*<,*l*,*r*>)

引数

x
数値のランダム変数です。

l
数値の左位置パラメータです。
デフォルト: 0

r
数値の右位置パラメータです。
デフォルト: 1
範囲: $r > l$

詳細

一様分布の CDF 関数は、一様分布(左位置パラメータは l、右位置パラメータは 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。式は次のとおりです。

$$CDF('UNIFORM', x, l, r) = \begin{cases} 0 & x < l \\ \frac{x - l}{r - l} & l \le x < r \\ 1 & x \ge r \end{cases}$$

注: l と r のデフォルト値は、それぞれ 0 と 1 になります。

逆ガウス(Wald)分布

CDF('WALD',*x*,*λ*<,*μ*>)

CDF('IGAUSS',*x*,*λ*<,*μ*>)

引数

x

数値のランダム変数です。

λ

数値の形状パラメータです。

範囲: $\lambda > 0$

μ

平均です。

デフォルト: 1

範囲: $\mu > 0$

詳細

Wald 分布の CDF 関数は、Wald 分布(形状パラメータは 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。式は次のとおりです。

$$Fx(x) = \Phi\left\{\sqrt{\frac{\lambda}{x}}\left(\frac{x}{\mu} - 1\right)\right\} + e^{2\lambda/\mu}\Phi\left\{-\sqrt{\frac{\lambda}{x}}\left(\frac{x}{\mu} + 1\right)\right\}$$

この式では、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。

Weibull 分布

CDF('WEIBULL',*x*,*a*<,*λ*>)

引数

x

数値のランダム変数です。

a

数値の形状パラメータです。

範囲: $a > 0$

λ

数値の尺度パラメータです。

デフォルト: 1

範囲: $\lambda > 0$

詳細

Weibull 分布の CDF 関数は、Weibull 分布(形状パラメータは a、尺度パラメータは 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。式は次のとおりです。

$$CDF('WEIBULL', x, a, \lambda) = \begin{cases} 0 & x < 0 \\ 1 - e^{-\left(\frac{x}{\lambda}\right)^a} & x \geq 0 \end{cases}$$

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
y=cdf('BERN',0,.25);	0.75
y=cdf('BETA',0.2,3,4);	0.09888
y=cdf('BINOM',4,.5,10);	0.37695
y=cdf('CAUCHY',2);	0.85242
y=cdf('CHISQ',11.264,11);	0.57858
y=cdf('EXPO',1);	0.63212
y=cdf('F',3.32,2,3);	0.82639
y=cdf('GAMMA',1,3);	0.080301
y=cdf('GENPOISSON',9,1,.7);	0.906162963
y=cdf('HYPER',2,200,50,10);	0.52367
y=cdf('LAPLACE',1);	0.81606
y=cdf('LOGISTIC',1);	0.73106
y=cdf('LOGNORMAL',1);	0.5
y=cdf('NEGB',1,.5,2);	0.5
y=cdf('NORMAL',1.96);	0.97500
y=cdf('NORMALMIX',2.3,3,.33,.33,.34, .5,1.5,2.5,.79,1.6,4.3);	0.7181
y=cdf('PARETO',1,1);	0
y=cdf('POISSON',2,1);	0.91970
y=cdf('T',.9,5);	0.79531
y=cdf('TWEEDIE',.8,5);	0.5917629164
y=cdf('UNIFORM',0.25);	0.25

SAS ステートメント	結果
y=cdf('WALD',1,2);	0.62770
y=cdf('WEIBULL',1,2);	0.63212

関連項目:

関数:

- “LOGCDF 関数” (624 ページ)
- “LOGPDF 関数” (626 ページ)
- “LOGSDF 関数” (628 ページ)
- “PDF 関数” (703 ページ)
- “QUANTILE 関数” (778 ページ)
- “SDF 関数” (832 ページ)
- “SQUANTILE 関数” (856 ページ)

CEIL 関数

予期しない浮動小数点の結果が生じないようにファジー処理して、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: 切り捨て関数

構文

CEIL (*argument*)

必須引数

引数
数値の定数、変数または式を指定します。

詳細

引数が整数の 1E-12 以内の場合、関数はその整数を返します。

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 以内でも、CEIL 関数はその整数と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。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果

var1=2.1;
a=ceil(var1);
put a;
	3

b=ceil(-2.4); put b;	-2
c=ceil(1+1.e-11); put c;	2
d=ceil(-1+1.e-11); put d;	0
e=ceil(1+1.e-13); put e;	1
f=ceil(223.456); put f;	224
g=ceil(763); put g;	763
h=ceil(-223.456); put h;	-223

関連項目:

関数:

- “CEILZ 関数” (289 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: 切り捨て関数

構文

CEILZ (*argument*)

必須引数

引数
数値定数、変数または式です。

詳細

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 以内でも、CEIL 関数はその整数と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。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
a=ceilz(2.1); put a;	3
b=ceilz(−2.4); put b;	−2
c=ceilz(1+1.e−11); put c;	2
d=ceilz(−1+1.e−11); put d;	0
e=ceilz(1+1.e−13); put e;	2
f=ceilz(223.456); put f;	224
g=ceilz(763); put g;	763
h=ceilz(−223.456); put h;	−223

関連項目:

関数:

- “CEIL 関数” (288 ページ)
- “FLOOR 関数” (470 ページ)
- “FLOORZ 関数” (471 ページ)
- “INT 関数” (543 ページ)
- “INTZ 関数” (582 ページ)
- “ROUND 関数” (810 ページ)
- “ROUNDE 関数” (816 ページ)
- “ROUNDZ 関数” (819 ページ)

CEXIST 関数

SAS カタログまたは SAS カタログエントリの存在を確認します。

カテゴリ: SAS ファイル I/O 関数

構文

CEXIST(*entry*<,'U'>)

必須引数

entry
SAS カタログまたはカタログのエントリ名を指定する文字定数、変数または式です。*entry* の値が 1 または 2 レベルの名前の場合、カタログ名とみなされます。カタログ内にエントリが存在するかどうかをテストする場合は、3 または 4 レベルの名前を使用します。

オプション引数

'U'
更新のためにカタログを開くことができるかどうかをテストします。

詳細

CEXIST は、SAS カタログまたはカタログエントリが存在する場合は 1、存在しない場合は 0 を返します。

サンプル

サンプル 1: カタログエントリの存在確認

この例では、LIB.CAT1 のエントリ X.PROGRAM の存在を確認します。

```
data _null_;
if cexist("lib.cat1.x.program") then
put "Entry X.PROGRAM exists";
run;
```

サンプル 2: 更新のためにカタログを開くことができるかどうかの確認

この例では、カタログ LIB.CAT1 が存在していて、更新のために開くことができるかどうかをテストします。カタログが存在しない場合、SAS ログにメッセージが書き込まれます。マクロステートメントでは、文字列を引用符で囲みません。

```
%if %sysfunc(cexist(lib.cat1,u)) %then
%put The catalog LIB.CAT1 exists and can be opened for update.;
%else
%put %sysfunc(sysmsg());
```

関連項目:

関数:

- “EXIST 関数” (389 ページ)

CHAR 関数

文字列の指定した位置の 1 文字を返します。

カテゴリ: 文字関数

構文

CHAR(*string*, *position*)

必須引数

string
文字定数、変数または式を指定します。

position
返す文字の位置を指定する整数です。

詳細

DATA ステップでは、CHAR 関数の対象変数のデフォルトの長さは 1 です。

position に欠損値が含まれている場合、CHAR は長さ 0 の文字列を返します。それ以外の場合、CHAR は長さ 1 の文字列を返します。

position が 0 以下の場合や文字列の長さよりも大きい場合、CHAR は空白を返します。それ以外の場合、CHAR は文字列の指定した位置の文字を返します。

比較

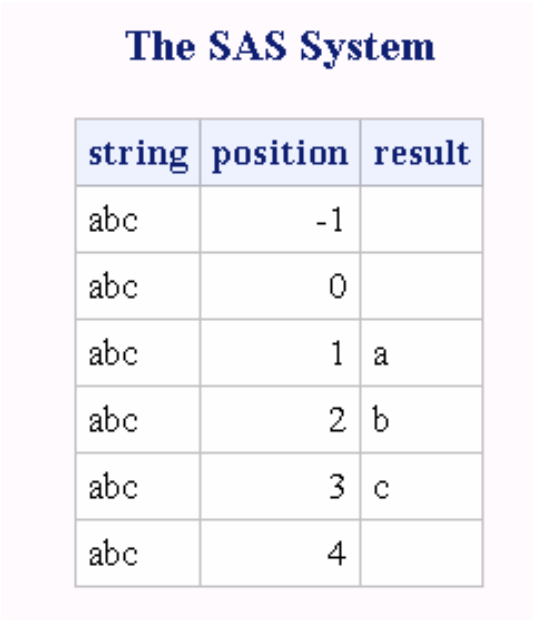
CHAR 関数は、SUBPAD(*string*, *position*, 1)と同じ結果を返します。結果は同じでも、対象変数のデフォルトの長さは異なります。

サンプル

次の例では、CHAR 関数を使用した結果を示します。

```
data test;
retain string "abc";
do position = -1 to 4;
result=char(string, position);
output;
end;
run;

proc print noobs data=test;
run;
```

画面 2.21 CHAR 関数からの出力



The SAS System

string	position	result
abc	-1	
abc	0	
abc	1	a
abc	2	b
abc	3	c
abc	4	

関連項目:

関数:

- “FIRST 関数” (469 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: 文字関数

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

CHOOSEC (*index-expression*, *selection-1* <,…*selection-n*>)

必須引数

index-expression
数値の定数、変数または式を指定します。

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。

詳細

返される変数の長さ

DATA ステップで、まだ長さが割り当てられていない変数に CHOOSEC 関数から値が返される場合、その変数には 200 バイトの長さが設定されます。

基本

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。

サンプル

次の例では、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。

```
data _null_;
Fruit=choosec(1,'apple','orange','pear','fig');
Color=choosec(3,'red','blue','green','yellow');
Planet=choosec(2,'Mars','Mercury','Uranus');
Sport=choosec(-3,'soccer','baseball','gymnastics','skiing');
put Fruit= Color= Planet= Sport=;
run;
```

SAS は次の行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” (294 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: 文字関数

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

CHOOSEN (*index-expression*, *selection-1* <,…*selection-n*>)

必須引数

index-expression
数値の定数、変数または式を指定します。

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。

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、CHOOSEC が文字値を返すのに対して CHOOSEN は数値を返す点が異なります。

サンプル

次の例では、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。

```
data _null_;
ItemNumber=choosen(5,100,50,3784,498,679);
Rank=choosen(-2,1,2,3,4,5);
Score=choosen(3,193,627,33,290,5);
Value=choosen(-5,-37,82985,-991,3,1014,-325,3,54,-618);
put ItemNumber= Rank= Score= Value=;
run;
```

SAS は次の行を口グに書き込みます。

```
ItemNumber=679 Rank=4 Score=33 Value=1014
```

関連項目:

関数:

- “CHOOSEC 関数” (293 ページ)

CINV 関数

χ2 乗分布から分位点を返します。

カテゴリ: 分位点

構文

CINV (*p*,*df* <,*nc*>)

必須引数

p
数値の確率です。
範囲: $0 \leq p < 1$

df
数値の自由度パラメータです。
範囲: $df > 0$

オプション引数

nc
数値の非心度パラメータです。
範囲: $nc \geq 0$

詳細

CINV 関数は、$\chi 2$ 乗分布(自由度は *df*、非心度パラメータは *nc*)から *p*番目の分位点を返します。$\chi 2$ 乗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。この関数では、整数以外の自由度パラメータ *df* を使用できます。

省略可能なパラメータ *nc* を指定していない場合や値が 0 の場合、心度 $\chi 2$ 乗分布から分位点が返されます。非心度パラメータ *nc* は、X が正規ランダム変数(平均は μ、分散は 1)の場合に、X^2 が非心度 $\chi 2$ 乗分布(df=1、$nc = \mu^2$)になるように定義されます。

注意:
nc の値が大きい場合、アルゴリズムが失敗する可能性があります。その場合、欠損値が返されます。

注: CINV は、PROBCHI 関数の逆数です。

サンプル

次に示す最初のステートメントでは、心度 $\chi 2$ 乗分布(自由度は 3)から 95番目のパーセント点をどのように見つけるのかを示します。2 番目のステートメントでは、非心度 $\chi 2$ 乗分布(自由度は 3.5、非心度パラメータは 4.5)から 95番目のパーセント点をどのように見つけるのかを示します。

SAS ステートメント	結果
q1=cinv(.95,3);	7.8147279033
a2=cinv(.95,3.5,4.5);	7.504582117

関連項目:

関数:

- “QUANTILE 関数” (778 ページ)

CLOSE 関数

SAS データセットを閉じます。

カテゴリ: SAS ファイル I/O 関数

構文

CLOSE(*data-set-id*)

必須引数

data-set-id
OPEN 関数が返すデータセット識別子を指定する数値変数です。

詳細

CLOSE は、処理が成功した場合にゼロ、成功しなかった場合に非ゼロ値を返します。アプリケーションで必要なくなるとすぐにすべての SAS データセットを閉じます。

注: DATA ステップ内で開かれているすべてのデータセットは、DATA ステップの終了時に自動的に閉じます。

サンプル

この例では、OPEN を使用して SAS データセット PAYROLL を開きます。この例では、データセットが正常に開いた場合(変数 PAYID の値が正になる)、CLOSE を使用してデータセットを閉じます。

```
%let payid=%sysfunc(open(payroll,is));
macro statements
%if &payid > 0 %then
%let rc=%sysfunc(close(&payid));
```

関連項目:

関数:

- “OPEN 関数” (697 ページ)

CMISS 関数

欠損引数の数を数えます。

カテゴリ: 記述統計

構文

CMISS(*argument-1* <, *argument-2,…*>)

必須引数

引数
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、その文字式は欠損として数えられます。

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。

比較

CMISS 関数は、引数を変換しません。NMISS 関数は、すべての引数を数値に変換します。

関連項目:

関数:

- “NMISS 関数” (666 ページ)
- “MISSING 関数” (645 ページ)

CNONCT 関数

$\chi 2$ 乗分布の非心度パラメータを返します。

カテゴリ: 数学関数

構文

CNONCT(*x*,*df*,*prob*)

必須引数

x
数値のランダム変数です。
範囲: $x \geq 0$

df
数値の自由度パラメータです。
範囲: $df > 0$

prob
確率です。
範囲: $0 < prob < 1$

詳細

CNONCT 関数は、非心度 $\chi 2$ 乗分布(パラメータは *x*、*df*、*nc*)から負でない非心度パラメータを返します。*prob* が心度 $\chi 2$ 乗分布(パラメータは *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。

$$P_c(x \mid \mathrm{df},\ nc) - prob = 0$$

前述の式には次の関係が適用されます。

$$P_c(x \mid \mathrm{df},\ \mathrm{nc}) = \varepsilon^{\frac{-nc}{2}} \sum_{j=0}^{\infty} \frac{\left(\frac{nc}{2}\right)^j}{j!} P_g\left(\frac{x}{2} \,\middle|\, \frac{df}{2} + j\right)$$

前述の式には次の関係が適用されます。

$$P_g(x \mid a)$$

次の式によって得られるガンマ分布の確率です。

$$P_g(x \mid a) = \frac{1}{\Gamma(a)} \int_0^x t^{a-1} \varepsilon^{-t} dt$$

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。

サンプル

```
data work;
x=2;
df=4;
do nc=1 to 3 by .5;
prob=probchi(x,df,nc);
ncc=cnonct(x,df,prob);
output;
end;
run;
proc print;
run;
```

画面 2.22 *χ2 乗分布の非心度パラメータの計算*

The SAS System

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

COALESCE 関数

数値の引数のリストからの最初の非欠損値を返します。

カテゴリ: 数学関数

構文

COALESCE(*argument-1*<..., *argument-n*>)

必須引数

引数
数値の定数、変数または式を指定します。

詳細

基本

COALESCE では、1 つ以上の数値引数を使用できます。COALESCE 関数は、リストされている順に各引数の値を確認し、最初の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。すべての引数の値が欠損値の場合、COALESCE 関数は欠損値を返します。

比較

COALESCE 関数は数値引数を検索するのに対し、COALESCEC 関数は文字引数を検索します。

サンプル

ステートメントとその結果を次に示します。

SAS ステートメント	結果
x = COALESCE(42, .);	42
y = COALESCE(.A, .B, .C);	.
z = COALESCE(., 7, ., ., 42);	7

関連項目:

関数:

- “COALESCEC 関数” (301 ページ)

COALESCEC 関数

文字引数のリストからの最初の非欠損値を返します。

カテゴリ: 文字関数

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

COALESCEC(*argument-1*<..., *argument-n*>)

必須引数

引数
文字定数、変数または式を指定します。

詳細

返される変数の長さ

DATA ステップで、まだ長さが割り当てられていない変数に COALESCEC 関数から値が返される場合、その変数には 200 バイトの長さが設定されます。

基本

COALESCEC では、1 つ以上の文字引数を使用できます。COALESCEC 関数は、リストされている順に各引数の値を確認し、最初の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 関数はその引数の値を返します。文字値は、長さがゼロの場合やすべての文字が空白の場合に欠損値とみなされます。すべての引数の値が欠損値の場合、COALESCEC 関数は長さがゼロの文字列を返します。

比較

COALESCEC 関数は文字引数を検索するのに対し、COALESCE 関数は数値引数を検索します。

サンプル

ステートメントとその結果を次に示します。

SAS ステートメント	結果
COALESCEC('', 'Hello')	Hello
COALESCEC ('', 'Goodbye', 'Hello')	Goodbye

関連項目:

関数:

• “COALESCE 関数” (300 ページ)

COLLATE 関数

ASCII 照合順序または EBCDIC 照合順序の文字列を返します。

カテゴリ: 文字関数

制限事項: I18N レベル 0 の関数は 1 バイト文字セット(SBCS)でのみ使用するように設計されています。

参照項目: “COLLATE Function: Windows” in *SAS Companion for Windows*

“COLLATE Function: UNIX” in *SAS Companion for UNIX Environments*

構文

COLLATE (*start-position*<,*end-position*>) | (*start-position*<,,*length*>)

必須引数

start-position

照合順序で返される最初の文字の位置を数値で指定します。

操作: *start-position* のみを指定すると、COLLATE は、その位置から照合順序の最後までの連続する文字または 255 文字(先に到達した方)を返します。

オプション引数

end-position

照合順序で返される最後の文字の位置を数値で指定します。

EBCDIC 照合順序では、*end-position* の最大値は 255 です。ASCII 照合順序では、0 ~ 127 の *end-position* の値に対応する文字が標準文字セットを表します。128 ~ 255 の *end-position* 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。

ヒント:

end-position は、*start-position* よりも大きくする必要があります。

end-position を指定すると、COLLATE は照合順序の *start-position* ~ *end-position* (両端を含む)間のすべての文字値を返します。

end-position を省略して *length* を使用する場合、*end-position* の位置にカンマを付けます。

長さ

照合順序内の文字数を指定します。

デフォルト: 200

ヒント: *end-position* を省略する場合、*length* を使用して結果の長さを明示的に指定します。

詳細

返される変数の長さ

DATA ステップで、まだ長さが割り当てられていない変数に COLLATE 関数から値が返される場合、その変数には 200 バイトの長さが設定されます。

基本

end-position と *length* の両方を指定すると、COLLATE は *length* を無視します。照合順序の残りの文字列よりも長い文字列を要求すると、COLLATE は照合順序の最後までの文字列を返します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
ASCII	----+----1----+---2--

x=collate(48,,10);
y=collate(48,57);
put @1 x @14 y;
	0123456789 0123456789

EBCDIC	

x=collate(240,,10);
y=collate(240,249);
put @1 x @14 y;
	0123456789 0123456789

関連項目:

関数:

- “BYTE 関数” (147 ページ)
- “RANK 関数” (797 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: 組み合わせ関数

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。

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: $r \leq n$

詳細

COMB 関数を数学的表現で表した式は次のとおりです。

$$COMB(n,r) = \binom{n}{r} = \frac{n!}{r!(n-r)!}$$

前述の式で、$n \geq 0$、$r \geq 0$ および $n \geq r$ 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。ある程度大きい値になると、COMB 関数を計算できない場合があります。

サンプル

ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=comb(5,1);	5

関連項目:

関数:

- “FACT 関数” (392 ページ)
- “PERM 関数” (723 ページ)
- “LCOMB 関数” (600 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: 文字関数

制限事項: I18N レベル 0 の関数は 1 バイト文字セット(SBCS)でのみ使用するように設計されています。

ヒント: DBCS に相当する関数は、*SAS 各国語サポート(NLS): リファレンスガイド*の KCOMPARE です。“DBCS の互換性” (305 ページ)も参照してください。

構文

COMPARE(*string–1*, *string-2*<,*modifiers*>)

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

modifier
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 または I	*string–1* および *string–2* の大文字と小文字を区別しません。
l または 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 または 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、SAS データセットまたは変数名に使用できない特殊文字(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。

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てから、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てから、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

DBCS に相当する関数は、*SAS 各国語サポート(NLS): リファレンスガイド*に記載されている KCOMPARE です。COMPARE 関数と KCOMPARE 関数には若干の違いがあります。両方の関数でさまざまな数の引数を使用できますが、第 3 引数の使用方法には互換性がありません。次の例では、構文の違いを示します。

COMPARE(*string-1*, *string-2* <, *modifiers*>)

KCOMPARE(*string-1* <, *position*<, *count*>> , *string-2*)

サンプル

サンプル 1: 2 つの文字列を比較するときの比較順序の理解

次の例では、COMPARE 関数を使用して 2 つの文字列を比較します。

```
data test;
infile datalines missover;
input string1 $char8. string2 $char8. modifiers $char8.;
result=compare(string1, string2, modifiers);
datalines;
1234567812345678
123 abc
abc abx
xyz abcdef
aBc abc
aBc AbC i
abc abc
abc abc l
abc abx
abc abx l
ABC 'abc'n
ABC 'abc'n n
'$12'n $12 n
'$12'n $12 nl
'$12'n $12 ln
;

proc print data=test;
run;
```

画面 2.23 COMPARE 関数を使用して 2 つの文字列を比較した結果

The SAS System

Obs	string1	string2	modifiers	result
1	12345678	12345678		0
2	123	abc		-1
3	abc	abx		-3
4	xyz	abcdef		1
5	aBc	abc		-2
6	aBc	AbC	i	0
7	abc	abc		-1
8	abc	abc	l	0
9	abc	abx		2
10	abc	abx	l	-3
11	ABC	'abc'n		1
12	ABC	'abc'n	n	0
13	'$12'n	$12	n	-1
14	'$12'n	$12	nl	1
15	'$12'n	$12	ln	0

サンプル 2: COMPARE 関数を使用した文字列の切り捨て

次の例では、: (コロン)修飾子を使用して文字列を切り捨てます。

```
data test2;
pad1=compare('abc','abc ');
pad2=compare('abc','abcdef ');
truncate1=compare('abc','abcdef',':');
truncate2=compare('abcdef','abc',':');
blank=compare('','abc', ':');
run;

proc print data=test2 noobs;
run;
```

画面 2.24 切り捨て修飾子を使用した結果

The SAS System

pad1	pad2	truncate1	truncate2	blank
0	-4	0	0	-1

関連項目:

関数:

- “COMPGED 関数” (311 ページ)
- “COMPLEV 関数” (317 ページ)

CALL ルーチン:

- “CALL COMPCOST ルーチン” (162 ページ)

COMPBL 関数

文字列内のワード間にある複数の空白を取り除きます。

カテゴリ: 文字関数

制限事項: I18N レベル 0 の関数は 1 バイト文字セット(SBCS)でのみ使用するように設計されています。

構文

COMPBL(*source*)

必須引数

ソース
圧縮する文字定数、変数または式を指定します。

詳細

返される変数の長さ

DATA ステップで、まだ長さが割り当てられていない変数に COMPBL 関数から値が返される場合、その変数の長さはデフォルトの第 1 引数の長さに設定されます。

基本

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。元の文字列から削除する対象の文字に空白を指定すると、COMPRESS 関数は元の文字列からすべての空白を削除します。一方、COMPBL 関数は複数の空白を 1 つの空白に圧縮しますが、1 つの空白には影響しません。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
----+----1----+---2--	
string='Hey Diddle Diddle'; string=compbl(string); put string;	Hey Diddle Diddle
string='125 E Main St'; length address $10; address=compbl(string); put address;	125 E Main

関連項目:

関数:

- “COMPRESS 関数” (321 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: 数学関数

構文

COMPFUZZ(*value1*, *value2* <, *fuzz*<, *scale*>>)

必須引数

value1
比較対象の 2 つの数値のうち 1 番目の数値を指定します。

value2
比較対象の 2 番目の数値を指定します。

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

scale
スケール係数を指定します。

デフォルト: MAX (ABS (value1), ABS (value2))

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(*value1* < *value2* - threshold の場合)

- 0(ABS(*value1* - *value2*) ≤ threshold の場合)
- 1(*value1* > *value2* + 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。

サンプル

浮動小数点計算では、数値を加算する順番によって累積値が異なる場合があります。x1 ~ x*n* 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。

```
data _null_;
x1 = -1/3;
x2 = 22/7;
x3 = -1234567891;
x4 = 1234567890;
/* Add the numbers in two different orders. */
sum1 = x1 + x2 + x3 + x4;
sum2 = x4 + x3 + x2 + x1;
diff = abs (sum1 - sum2);
put sum1= / sum2= / diff=;
/* Using only a fuzz value gives the wrong result. The fuzz value */
/* is 8 because there are four numbers in each sum, for a total of */
/* eight numbers. */
compfuzz = compfuzz (sum1, sum2, 8);
put "fuzz only (wrong): " compfuzz=;
/* Using a fuzz factor and a scale value gives the correct result. */
scale = abs(x1) + abs(x2) + abs(x3) + abs(x4);
compfuzz = compfuzz (sum1, sum2, 8, scale);
put "fuzz and scale (correct): " compfuzz=;
run;
```

SAS は次の出力をログに書き込みます。

ログ 2.8 COMPFUZZ 関数の SAS ログ(一部)

```
sum1=1.8095238095
sum2=1.8095238095
diff=5.543588E-11
fuzz only (wrong): compfuzz=-1
fuzz and scale (correct): compfuzz=0
```

関連項目:

関数:

- "FUZZ 関数" (490 ページ)
- "ROUND 関数" (810 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: 文字関数

制限事項: I18N レベル 0 の関数は 1 バイト文字セット(SBCS)でのみ使用するように設計されています。

構文

COMPGED(*string-1*, *string-2* <,*cutoff*> <,*modifiers*>)

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
数値定数、変数または式です。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* に小さい値を使うと、*string–1* および *string–2* の値が長い場合に COMPGED の効率が向上します。

modifiers
COMPGED 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 または I	*string–1* および *string–2* の大文字と小文字を区別しません。
l または 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 または N	n リテラル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* の大文字と小文字の区別をしません。

: (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。

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てから、n リテラルから引用符を削除します。
- "LN"は n リテラルから引用符を削除してから、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を使用します。ポインタを進める、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。

操作	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、ポインタを移動せずに 1 文字を出力文字列の末尾に追加します。

操作	デフォルトコスト単位	演算の内容
BLANK	10	次の操作のいずれかを行います。 • ポインタ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の位置を 1 つ進めます。 • ポインタ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、ポインタの位置を 1 つ進めます。 COMPCOST 関数で BLANK のコストがゼロ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。
DELETE	100	出力文字列を変更せずにポインタの位置を 1 つ進めます。
DOUBLE	20	ポインタ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の位置を 1 つ進めます。
FINSERT	200	ポインタが第 1 位置にあるとき、ポインタを移動せずに 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、ポインタの位置を 1 つ進めます。
INSERT	100	ポインタ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、ポインタの位置を 1 つ進めます。

操作	デフォルトコスト単位	演算の内容
PUNCTUATION	30	次の操作のいずれかを行います。 • ポインタ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の位置を 1 つ進めます。 • ポインタ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、ポインタの位置を 1 つ進めます。 COMPCOST 関数で PUNCTUATION のコストがゼロ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。
REPLACE	100	出力文字列の末尾に 1 文字を追加し、ポインタの位置を 1 つ進めます。
SINGLE	20	ポインタ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の位置を 1 つ進めます。
SWAP	20	入力文字列からポインタ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、CALL COMPCOST ルーチンを使用するか、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、CALL COMPCOST ルーチンを使って、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、CALL COMPCOST ルーチンおよび 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。

サンプル

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。

```
data test;
infile datalines missover;
input String1 $char8. +1 String2 $char8. +1 Operation $40.;
GED=compged(string1, string2);
datalines;
baboon baboon match
baXboon baboon insert
baoon baboon delete
baXoon baboon replace
baboonX baboon append
baboo baboon truncate
babboon baboon double
babon baboon single
baobon baboon swap
bab oon baboon blank
bab,oon baboon punctuation
bXaoon baboon insert+delete
bXaYoon baboon insert+replace
bXoon baboon delete+replace
Xbaboon baboon finsert
aboon baboon trick question: swap+delete
Xaboon baboon freplace
axoon baboon fdelete+replace
axoo baboon fdelete+replace+truncate
axon baboon fdelete+replace+single
baby baboon replace+truncate*2
balloon baboon replace+insert
;
```

```
proc print data=test label;
label GED='Generalized Edit Distance';
var String1 String2 GED Operation;
run;
```

画面 2.25 演算に基づく一般化編集距離

The SAS System

Obs	String1	String2	Generalized Edit Distance	Operation
1	baboon	baboon	0	match
2	baXboon	baboon	100	insert
3	baoon	baboon	100	delete
4	baXoon	baboon	100	replace
5	baboonX	baboon	50	append
6	baboo	baboon	10	truncate
7	babboon	baboon	20	double
8	babon	baboon	20	single
9	baobon	baboon	20	swap
10	bab oon	baboon	10	blank
11	bab,oon	baboon	30	punctuation
12	bXaoon	baboon	200	insert+delete
13	bXaYoon	baboon	200	insert+replace
14	bXoon	baboon	200	delete+replace
15	Xbaboon	baboon	200	finsert
16	aboon	baboon	120	trick question: swap+delete
17	Xaboon	baboon	200	freplace
18	axoon	baboon	300	fdelete+replace
19	axoo	baboon	310	fdelete+replace+truncate
20	axon	baboon	320	fdelete+replace+single
21	baby	baboon	120	replace+truncate*2
22	balloon	baboon	200	replace+insert

関連項目:

関数:

- “COMPARE 関数” (304 ページ)
- “COMPLEV 関数” (317 ページ)

CALL ルーチン:

- “CALL COMPCOST ルーチン” (162 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: 文字関数

制限事項: I18N レベル 0 の関数は 1 バイト文字セット(SBCS)でのみ使用するように設計されています。

構文

COMPLEV(*string-1*, *string-2* <,*cutoff*> <,*modifiers*>)

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。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* の値が長い場合に COMPLEV の効率が向上します。

modifiers
COMPLEV 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 または I	*string–1* および *string–2* の大文字と小文字を区別しません。
l または 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 または N	n リテラル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* の大文字と小文字の区別をしません。
: (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てから、n リテラルから引用符を削除します。
- "LN"は n リテラルから引用符を削除してから、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。

サンプル

次の例では、Levenshtein の編集距離を計算して 2 つの文字列を比較します。

```
data test;
infile datalines missover;
input string1 $char8. string2 $char8. modifiers $char8.;
result=complev(string1, string2, modifiers);
datalines;
1234567812345678
abc abxc
ac abc
aXc abc
aXbZc abc
aXYZc abc
WaXbYcZ abc
XYZ abcdef
aBc abc
aBc AbC i
abc abc
abc abc l
AxC 'abc'n
AxC 'abc'n n
;

proc print data=test;
run;
```

画面 2.26 Levenshtein の編集距離を計算して 2 つの文字列を比較した結果

The SAS System

Obs	string1	string2	modifiers	result
1	12345678	12345678		0
2	abc	abxc		1
3	ac	abc		1
4	aXc	abc		1
5	aXbZc	abc		2
6	aXYZc	abc		3
7	WaXbYcZ	abc		4
8	XYZ	abcdef		6
9	aBc	abc		1
10	aBc	AbC	i	0
11	abc	abc		2
12	abc	abc	l	0
13	AxC	'abc'n		6
14	AxC	'abc'n	n	1

関連項目:

関数:

- “COMPARE 関数” (304 ページ)
- “COMPGED 関数” (311 ページ)

CALL ルーチン:

- “CALL COMPCOST ルーチン” (162 ページ)

COMPOUND 関数

複利パラメータを返します。

カテゴリ: 財務関数

構文

COMPOUND(*a*,*f*,*r*,*n*)

必須引数

a

数値で初期の金額を指定します。

範囲: $a \geq 0$

f

数値で将来(*n* 期間の末日)の金額を指定します。

範囲: $f \geq 0$

r

数値で期間単位の金利を分数で指定します。

範囲: $r \geq 0$

n

整数で複利計算を行う期間数を指定します。

範囲: $n \geq 0$

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 から工ラーが返されます。

サンプル

$2000 の投資で名目年利 9 パーセント、30 か月複利運用したときの累計額は次の式で表します。

```
future=compound(2000,.,0.09/12,30);
```

返される値は 2502.54 です。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 パーセントは、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。

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: 文字関数

制限事項: I18N レベル 0 の関数は 1 バイト文字セット(SBCS)でのみ使用するように設計されています。

ヒント: DBCS に相当する関数は、KCOMPRESS です。

構文

COMPRESS(<*source*> <, *chars*> <, *modifiers*>)

オプション引数

ソース

文字定数、変数または式を指定します。これらから指定した文字が削除されます。

chars

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。

デフォルトでは、このリスト内の文字が *source* 引数から削除されます。第 3 引数に K 修飾子を指定すると、このリスト内の文字のみが結果に保持されます。

ヒント 第 3 引数に他の修飾子を使うことでリストに文字を追加できます。

ヒント 文字のリテラル文字列を引用符で囲みます。

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。修飾子として使用できる文字は次のとおりです。

a または A	文字のリストにアルファベット文字を追加します。
c または C	文字のリストに制御文字を追加します。
d または D	文字のリストに数字を追加します。
f または 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 または G	文字のリストにグラフィカル文字を追加します。
h または H	文字のリストに水平タブを追加します。
i または I	保持する、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 または K	リスト内の文字を削除するのではなく、保持します。
l または L	小文字を文字リストに追加します。
n または 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 または 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 または P　文字のリストに句読点を追加します。

s または 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 または 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 または U　大文字を文字リストに追加します。

w または W　印刷可能文字を文字リストに追加します。

x または X　文字のリストに 16 進文字を追加します。

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。

詳細

返される変数の長さ

DATA ステップで、まだ長さが割り当てられていない変数に COMPRESS 関数から値が返される場合、その変数の長さは第 1 引数の長さに設定されます。

基本

COMPRESS 関数には、NULL 引数を指定できます。NULL 引数は長さがゼロの文字列として扱われます。

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*	引数から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場合、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。
最初の 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(s)*	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。

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。たとえば、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、次の関数呼び出しのいずれかを使用します。

```
COMPRESS(source, "1234567890+-");
COMPRESS(source, "+-", "d");
```

サンプル

サンプル 1: ブランクの取り除き

SAS ステートメント	結果
	----+----1
a='AB C D '; b=compress(a); put b;	ABCD

サンプル 2: 小文字の取り除き

SAS ステートメント	結果
	----+----1----+----2----+----3
x='123-4567-8901 B 234-5678-9012 c'; y=compress(x,'ABCD','l'); put y;	123-4567-8901 234-5678-9012

サンプル 3: 空白文字の取り除き

SAS ステートメント	結果
	----+----1
x='1 2 3 4 5'; y=compress(x,,'s'); put y;	12345

サンプル 4: リストされた文字の保持

SAS ステートメント	結果
	----+----1
x='Math A English B Physics A'; y=compress(x,'ABCD','k'); put y;	ABA

サンプル 5: 文字列を圧縮して長さ 0 を返す

SAS ステートメント	結果
	----+----1
x=' '; l=lengthn(compress(x)); put l;	0

関連項目:

関数:

- “COMPBL 関数” (308 ページ)
- “LEFT 関数” (601 ページ)
- “TRIM 関数” (899 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: 数学関数

構文

CONSTANT(*constant*<, *parameter*>)

必須引数

定数

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。

説明	定数
自然数の底	'E'
Euler 定数	'EULER'
PI	'PI'
正確な整数	'EXACTINT' <,nbytes>
最大倍精度浮動小数	'BIG'
base を底とする BIG の対数	'LOGBIG' <,base>
BIG の平方根	'SQRTBIG'
最小倍精度浮動小数	'SMALL'

説明	定数
base を底とする SMALL の対数	'LOGSMALL' <,base>
SMALL の平方根	'SQRTSMALL'
マシン精度の定数	'MACEPS'
base を底とする MACEPS の対数	'LOGMACEPS' <,base>
MACEPS の平方根	'SQRTMACEPS'

オプション引数

パラメータ

任意の数値パラメータです。*constant* で指定する定数の中には、CONSTANT 関数の働きを変える任意の引数があります。

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 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。

範囲 $2 \le nbytes \le 8$

デフォルト 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

CONSTANT('BIG')

使用しているコンピュータで表現できる最大倍精度浮動小数点数(8 ビット)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。

制限 指定する *base* は 1+SQRTMACEPS の値より大きくする必要があります。

デフォルト 自然数の底、E。

base を底として、使用しているコンピュータで表現できる最大倍精度浮動小数点数(8 ビット)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は発生せず、問題なく累乗できます。

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は発生せず、問題なくで累乗できます。

BIG の平方根

CONSTANT('SQRTBIG')

使用しているコンピュータで表現できる最大倍精度浮動小数点数(8 ビット)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に問題なく平方根を求めることができます。

最小倍精度浮動小数

CONSTANT('SMALL')

使用しているコンピュータで表現できる最小倍精度浮動小数点数(8 ビット)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。

制限　指定する *base* は 1+SQRTMACEPS の値より大きくする必要があります。

デフォルト　自然数の底、E。

base を底として、使用しているコンピュータで表現できる最小倍精度浮動小数点数(8 ビット)の対数を返します。

CONSTANT('LOGSMALL', *base*)以下に累乗された指定の *base* は、累乗演算(**)を使って問題なく累乗できます。アンダーフローまたはゼロにはなりません。

浮動小数点数は、CONSTANT('LOGSMALL')以上任意の浮動小数点数で累乗関数 EXP を使って問題なくで累乗できます。アンダーフローまたはゼロにはなりません。

SMALL の平方根

CONSTANT('SQRTSMALL')

使用しているコンピュータで表現できる最小倍精度浮動小数点数(8 ビット)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、問題なく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、$\varepsilon = 2^{-j}$が、ある整数 j に対する最小倍精度浮動小数点数(8 ビット)として返されるため、$1 + \varepsilon > 1$ となります。

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。

制限　指定する *base* は 1+SQRTMACEPS の値より大きくする必要があります。

デフォルト　自然数の底、E。

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。

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: 財務関数

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。分数で表します。
範囲: $0 < y < 1$

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: $f > 0$

c(1), ... ,c(k)
キャッシュフローのリストを指定します。

詳細

CONVX 関数は次の値を返します。

$$C = \sum_{k=1}^{K} \frac{k(k+f)\dfrac{c(k)}{(1+y)^{\frac{k}{f}}}}{P(1+y)^2 f^2}$$

前述の式には次の関係が適用されます。

$$P = \sum_{k=1}^{K} \frac{c(k)}{(1+y)^{\frac{k}{f}}}$$

サンプル

```
data _null_;
c=convx(1/20,1,.33,.44,.55,.49,.50,.22,.4,.8,.01,.36,.2,.4);
put c;
run;
```

返される値は 42.3778 です。

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: 財務関数

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: $A > 0$

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: $0 \le c < 1$

n

期間当たりのクーポン数を指定します。

範囲: $n > 0$ の整数

K

クーポンの残数を指定します。

範囲: $K > 0$ の整数

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: $0 < k_0 \le \frac{1}{n}$

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: $y > 0$

詳細

CONVXP 関数は次の値を返します。

$$C = \frac{1}{n^2}\left(\frac{\sum_{k=1}^{K} t_k(t_k+1)\frac{c(k)}{\left(1+\frac{y}{n}\right)^{t_k}}}{P\left(1+\frac{y}{n}\right)^2}\right)$$

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。

$$P = \sum_{k=1}^{K} \frac{c(k)}{\left(1+\frac{y}{n}\right)^{t_k}}$$

サンプル

次の例では、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。

```
data _null_;
y=convxp(1000,.01,4,14,.33/2,.08);
put y;
run;
```

返される値は 11.729001987 です。

COS 関数

余弦(コサイン)を返します。

カテゴリ: 三角関数

構文

COS (*argument*)

必須引数

引数

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。*argument* が非常に大きく、**mod(argument,pi)**の誤差がおよそ小数点以下 3 桁より小さい場合、COS は欠損値を返します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=cos(0.5);	0.8775825619
x=cos(0);	1
x=cos(3.14159/3);	0.500000766

COSH 関数

双曲線余弦を返します。

カテゴリ: 双曲線関数

構文

COSH(*argument*)

必須引数

引数
数値の定数、変数または式を指定します。

詳細

COSH 関数は、次の式による引数の双曲線余弦(ハイパボリック コサイン)を返します。

$$\left(\varepsilon^{argument} + \varepsilon^{-argument}\right) / 2$$

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=cosh(0);	1
x=cosh(-5.0);	74.209948525
x=cosh(0.5);	1.1276259652

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: 文字関数

制限事項: 英語以外の言語を使用する場合、可能な限り I18N レベル 1 の関数の使用は避けてください。I18N レベル 1 の関数は、特定の環境下では 2 バイト文字セット(DBCS)または複数バイト文字セット(MBCS)エンコーディングを使用すると正常に動作しない場合があります。

ヒント: KCOUNT 関数 を DBCS 処理に使えますが、働きが異なります。詳細については、DBCS の互換性 (332 ページ)を参照してください。

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。
ヒント: 文字のリテラル文字列を引用符で囲みます。

substring
string の中で数える対象となる部分文字列を文字定数、変数または式で指定します。
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。次の修飾子(*modifiers*)には大文字と小文字のどちらでも使用できます。

i 数える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、COUNT は *substring* 内の文字に使用されているとおりに大文字と小文字を区別して文字を数えます。

t *string* および *substring* から末尾の空白を取り除きます。

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。

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 subsring 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 を返す可能性があります。

DBCS の互換性

KCOUNT 関数(*SAS 各国語サポート(NLS): リファレンスガイド*に記載されています)を DBCS 処理に使えますが、働きが異なります。

COUNT 関数の *substring* の値が 2 バイトよりも長い場合、COUNT 関数は DBCS 文字列を扱えます。次の例では、構文の違いを示します。

COUNT(*string*, *substring* <,*modifiers*>

KCOUNT(*string)*

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。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
xyz='This is a thistle? Yes, this is a thistle.'; howmanythis=count(xyz,'this'); put howmanythis;	3
xyz='This is a thistle? Yes, this is a thistle.'; howmanyis=count(xyz,'is'); put howmanyis;	6

SAS ステートメント	結果
howmanythis_i=count('This is a thistle? Yes, this is a thistle.' ,'this','i'); put howmanythis_i;	4
variable1='This is a thistle? Yes, this is a thistle.'; variable2='is '; variable3='i'; howmanyis_i=count(variable1,variable2,variable3); put howmanyis_i;	4

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;
	6

関連項目:

関数:

- “COUNTC 関数” (333 ページ)
- “COUNTW 関数” (337 ページ)
- “FIND 関数” (448 ページ)
- “INDEX 関数” (531 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: 文字関数

制限事項: 英語以外の言語を使用する場合、可能な限り I18N レベル 1 の関数の使用は避けてください。I18N レベル 1 の関数は、特定の環境下では 2 バイト文字セット(DBCS)または複数バイト文字セット(MBCS)エンコーディングを使用すると正常に動作しない場合があります。

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。

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 *modifier* 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。

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

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 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。

空白	無視されます。
a または A	文字のリストにアルファベット文字を追加します。
b または 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 または C	文字のリストに制御文字を追加します。
d または D	文字のリストに数字を追加します。
f または 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 または G	文字のリストにグラフィカル文字を追加します。
h または H	文字のリストに水平タブを追加します。
i または I	大文字と小文字を区別しません。
l または L	小文字を文字リストに追加します。
n または 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 または 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。以降の呼び出しで *charlist* または *modifier* 引数の値を変更する場合、COUNTC で変更が無視されることがあります。
p または P	文字のリストに句読点を追加します。
s または 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 または 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 または U　大文字を文字リストに追加します。

v または 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 または W　印刷可能文字を文字リストに追加します。

x または X　文字のリストに 16 進文字を追加します。

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。

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。

サンプル

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例を次に示します。

```
data test;
string = 'Baboons Eat Bananas ';
a = countc(string, 'a');
b = countc(string,'b');
b_i = countc(string,'b','i');
abc_i = countc(string,'abc','i');
/* Scan string for characters that are not "a", "b", */
/* and "c", ignore case, (and include blanks). */
abc_iv = countc(string,'abc','iv');
/* Scan string for characters that are not "a", "b", */
/* and "c", ignore case, and trim trailing blanks. */
abc_ivt = countc(string,'abc','ivt');
run;

options pageno=1 ls=80 nodate;
proc print data=test noobs;
run;
```

画面 2.27 COUNTC 関数で修飾子を使用する場合としない場合の出力

The SAS System

string	a	b	b_i	abc_i	abc_iv	abc_ivt
Baboons Eat Bananas	5	1	3	8	16	11

関連項目:

関数:

- “ANYALNUM 関数” (98 ページ)
- “ANYALPHA 関数” (100 ページ)
- “ANYCNTRL 関数” (102 ページ)
- “ANYDIGIT 関数” (103 ページ)
- “ANYGRAPH 関数” (106 ページ)
- “ANYLOWER 関数” (109 ページ)
- “ANYPRINT 関数” (112 ページ)
- “ANYPUNCT 関数” (114 ページ)
- “ANYSPACE 関数” (116 ページ)
- “ANYUPPER 関数” (118 ページ)
- “ANYXDIGIT 関数” (119 ページ)
- “NOTALNUM 関数” (668 ページ)
- “NOTALPHA 関数” (669 ページ)
- “NOTCNTRL 関数” (671 ページ)
- “NOTDIGIT 関数” (673 ページ)
- “NOTGRAPH 関数” (678 ページ)
- “NOTLOWER 関数” (680 ページ)
- “NOTPRINT 関数” (683 ページ)
- “NOTPUNCT 関数” (684 ページ)
- “NOTSPACE 関数” (686 ページ)
- “NOTUPPER 関数” (688 ページ)
- “NOTXDIGIT 関数” (690 ページ)
- “FINDC 関数” (450 ページ)
- “INDEXC 関数” (533 ページ)
- “VERIFY 関数” (918 ページ)

COUNTW 関数

文字列中のワード数を数えます。

カテゴリ: 文字関数

構文

COUNTW(<*string*> <, *chars*> <, *modifiers*>)

オプション引数

string
文字定数、変数または式を指定します。この中にあるワードを数えます。

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。リスト中の文字はワード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。

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 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。

空白	無視されます。
a または A	文字のリストにアルファベット文字を追加します。
b または B	左から右方向ではなく、右から左方向に数えます。右から左方向へ数えて変化があるのは、Q 修飾子を使っている場合で、かつ一組になっていない引用符が文字列中にある場合のみです。
c または C	文字のリストに制御文字を追加します。
d または D	文字のリストに数字を追加します。
f または 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 または G	文字のリストにグラフィカル文字を追加します。
h または H	文字のリストに水平タブを追加します。
i または I	大文字か小文字かは無視します。
k または 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。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 または L　小文字を文字リストに追加します。

m または 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 または N　数字、アンダースコアおよび英文字(VALIDVARNAME=V7 で SAS 変数名の先頭文字の次に使用できる文字)を文字リストに追加します。

o または O　*chars* 引数および *modifier* 引数を 1 回だけ処理します。COUNTW 関数の呼び出し時に毎回処理しません。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、*chars* 引数および *modifier* が変更されないループで COUNTW を呼び出すときに、より迅速に実行できます。

p または P　文字のリストに句読点を追加します。

q または 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、一致しない引用符が含まれる場合、左から右へのスキャンでは、右から左へのスキャンとは異なるワードが生成されます。

s または 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 または T　*string* 引数および *chars* 引数から末尾の空白を取り除きます。

u または U　大文字を文字リストに追加します。

w または W　印刷可能文字を文字リストに追加します。

x または X　文字のリストに 16 進文字を追加します。

詳細

"ワード"の定義

COUNTW 関数では、次の特性のいずれかを持つ部分文字列を"ワード"と呼びます。

- 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: "ワード"の定義は SCAN 関数および COUNTW.sgml 関数の両方に共通します。

区切り文字とは、ワードを区切るために指定できる数個の文字のいずれかを意味します。

ASCII 環境および EBCDIC 環境で COUNTW 関数を使用する

COUNTW 関数で引数を 2 つしか使用しない場合、ASCII 文字と EBCDIC 文字のいずれをコンピュータで使用しているかによってデフォルトの区切り文字が変わります。

- コンピュータで ASCII 文字が使われている場合、デフォルトの区切り文字は次のとおりです。

  空白 ! $ % & ( ) * + , - . / ; < ^ |

  ^文字のない ASCII 環境の場合、SCAN 関数はかわりに~文字を使用します。

- コンピュータで EBCDIC 文字が使われている場合、デフォルトの区切り文字は次のとおりです。

  空白 ! $ % & ( ) * + , - . / ; < ¬ | ¢

NULL 引数を使用する

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 修飾子を使用する

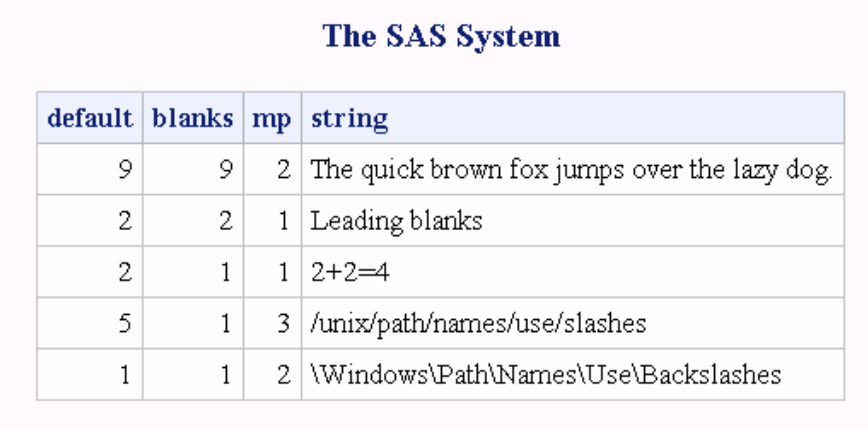
M 修飾子を使用しない場合、ワードには少なくとも 1 つの文字が含まれている必要があります。M 修飾子を使用する場合、ワードの長さをゼロとすることができます。この場合、ワード数は文字列中の区切り文字数に 1 を加算した数です。Q 修飾子を使用するときに引用符に囲まれている文字列中の区切り文字の数ではありません。

サンプル

COUNTW 関数に M 修飾子および P 修飾子を使う方法を次の例に示します。

```
data test;
length default blanks mp 8;
input string $char60.;
default = countw(string);
blanks = countw(string, ' ');
mp = countw(string, 'mp');
datalines;
The quick brown fox jumps over the lazy dog.
Leading blanks
2+2=4
/unix/path/names/use/slashes
¥Windows¥Path¥Names¥Use¥Backslashes
;
run;

proc print noobs data=test;
run;
```

画面 2.28 COUNTW 関数からの出力



The SAS System

default	blanks	mp	string
9	9	2	The quick brown fox jumps over the lazy dog.
2	2	1	Leading blanks
2	1	1	2+2=4
5	1	3	/unix/path/names/use/slashes
1	1	2	\Windows\Path\Names\Use\Backslashes

関連項目:

関数:

- “COUNT 関数” (331 ページ)
- “COUNTC 関数” (333 ページ)
- “FINDW 関数” (457 ページ)
- “SCAN 関数” (824 ページ)

CALL ルーチン:

- “CALL SCAN ルーチン” (233 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

構文

CSS(*argument-1*<,...*argument-n*>)

必須引数

引数

数値の定数、変数または式を指定します。少なくとも 1 つの非欠損引数が必要です。非欠損引数がない場合は、関数から欠損値が返されます。引数が 1 つ以上ある場合、引数リストには OF で始まる変数のリストを含められます。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x1=css(5,9,3,6);	18.75
x2=css(5,8,9,6,.);	10
x3=css(8,9,6,.);	4.6666666667
x4=css(of x1-x3);	101.11574074

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: 財務関数

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。

number-of-periods
支払期間の数を指定します。*number-of-periods* は正の整数値とする必要があります。

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。

オプション引数

start-period
計算の開始期間を指定します。

end-period
計算の終了期間を指定します。

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。*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。

サンプル

- 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。

```
TotalInterest = CUMIPMT(0.09/12, 360, 125000, 13, 24, 0);
```

計算によって 11,135.23 という値が返されます。

- 同じローンで最初の期間に支払う利息は次のように計算します。

```
first_period_interest = CUMIPMT(0.09/12, 360, 125000, 1, 1, 0);
```

計算によって 937.50 という値が返されます。

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: 財務関数

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。

number-of-periods
支払期間の数を指定します。*number-of-periods* は正の整数値とする必要があります。

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。

オプション引数

start-period
計算の開始期間を指定します。

end-period
計算の終了期間を指定します。

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。*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。

サンプル

- 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。

```
PrincipalYear2=CUMPRINC(0.09/12, 360, 125000, 12, 24, 0);
```

計算によって 934.107 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。

```
PrincipalYear2b = CUMPRINC(0.09/12, 360, 125000, 12, 24, 1);
```

計算によって 927.153 という値が返されます。

CUROBS 関数

現在のオブザベーションのオブザベーション番号を返します。

カテゴリ: SAS ファイル I/O 関数

要件 ネイティブライブラリエンジンを使ってアクセスする非圧縮の SAS データセットとの組み合わせでのみ、この関数を使用してください。

構文

CUROBS(*data-set-id*)

必須引数

data-set-id
OPEN 関数が返すデータセット識別子を指定する数値です。

詳細

使用するエンジンがオブザベーション番号をサポートしない場合、関数は欠損値を返します。

この関数を SAS ビューと使用する場合、関連する SAS データセット内のオブザベーション数ではなく、SAS ビュー内のオブザベーション数である相対オブザベーション番号を返します。

サンプル

FETCHOBS 関数を使用して、データセット MYDATA 中の 10 番目のオブザベーションを取得する例を次に示します。CUROBS から返される OBSNUM の値は 10 です。

```
%let dsid=%sysfunc(open(mydata,i));
%let rc=%sysfunc(fetchobs(&dsid,10));
%let obsnum=%sysfunc(curobs(&dsid));
```

関連項目:

関数:

- “FETCHOBS 関数” (399 ページ)
- “OPEN 関数” (697 ページ)

CV 関数

変動係数を返します。

カテゴリ: 記述統計

構文

CV(*argument-1*,*argument-2*<,...*argument-n*>)

必須引数

引数
数値の定数、変数または式を指定します。少なくとも 2 つの引数が必要です。引数リストには OF で始まる変数のリストを含められます。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x1=cv(5,9,3,6);	43.47826087
x2=cv(5,8,9,6,.);	26.082026548
x3=cv(8,9,6,.);	19.924242152
x4=cv(of x1-x3);	40.953539216

DACCDB 関数

定率法による減価償却累積額を返します。

カテゴリ: 財務関数

構文

DACCDB(*p*,*v*,*y*,*r*)

必須引数

p
計算対象期間を示す数値です。整数でない *p* 引数は、端数期間の前後で連続する 2 期間に償却が比例配分されます。

v
償却対象の資産の初期額を表す数値です。

y
資産の耐用期間を表す数値です。

範囲: $y > 0$

r
償却率を小数点値で表した数値です。

範囲: $r > 0$

詳細

DACCDB 関数は定率法による減価償却累積額を返します。式は次のとおりです。

$$DACCDB(p,\ v,\ y,\ r) = \begin{cases} 0 & p \le 0 \\ v\left(1 - \left(1 - \frac{r}{y}\right)^{int(p)}\right)\left(1 - (p - int(p))\frac{r}{y}\right) & p > 0 \end{cases}$$

int(*p*)は *p* の整数部分です。*p* 引数および *y* 引数には、期間を表す同じ単位を使って表す必要があります。*r* を 2 に設定すると倍額定率を得られます。

サンプル

償却対象の初期額が$1000 で、耐用期間が 15 年の資産があるとします。定率を 200 パーセントとする場合の最初の 10 年の償却額を表す式は次のとおりです。

```
a=daccdb(10,1000,15,2);
```

返される値は 760.93 です。第 1 引数および第 3 引数は年単位で表します。

DACCDBSL 関数

定率法による減価償却累積額と、定額法による原価償却への変換を返します。

カテゴリ: 財務関数

構文

DACCDBSL(*p*,*v*,*y*,*r*)

必須引数

p
計算対象期間を示す数値です。

v
償却対象の資産の初期額を表す数値です。

y
資産の耐用期間を表す整数です。
範囲: $y > 0$

r
償却率を分数で表した数値です。
範囲: $r > 0$

詳細

DACCDBSL 関数は定率法による減価償却累積額と、次のとおり定義される定額法による減価償却への変換を返します。

$$DACCDBSL\ (p,\ v,\ y,\ r) = \sum_{i=1}^{p} DEPDBSL\ (i,\ v,\ y,\ r)$$

定額法による減価償却に変換する定率法は、各期間で償却額が大きい方の償却方法(残存償却額に対する定率法または定額法)を選択します。*p* 引数および *y* 引数には、期間を表す同じ単位を使って表す必要があります。

サンプル

償却対象の初期額が$1,000 で、耐用期間が 10 年の資産があるとします。定率を 150%とする場合の 5 年目の減価償却累積額を表す式は次のとおりです。

```
y5=daccdbsl(5,1000,10,1.5);
```

返される値は 564.99 です。第 1 引数および第 3 引数は年単位で表します。

DACCSL 関数

定額法による減価償却累積額を返します。

カテゴリ: 財務関数

構文

DACCSL(*p*,*v*,*y*)

必須引数

p
計算対象期間を示す数値です。分数の *p* は、端数期間の前後で連続する 2 期間に償却が比例配分されます。

v
償却対象の資産の初期額を表す数値です。

y
資産の耐用期間を表す数値です。
範囲: $y > 0$

詳細

DACCSL 関数は定額法による減価償却累積額を返します。これは次の式で求められます。

$$DACCSL\ (p,\ v,\ y) = \begin{cases} 0 & p < 0 \\ v\left(\frac{p}{y}\right) & 0 \le p \le y \\ v & p > y \end{cases}$$

p 引数および *y* 引数には、期間を表す同じ単位を使って表す必要があります。

サンプル

01APR86 に取得された資産があり、減価償却対象の初期額が$1000 で、耐用期間が 10 年であるとします。この資産の 31DEC87 までの減価償却累積額は次のとおり表せます。

```
a=daccsl(1.75,1000,10);
```

返される値は 175.00 です。第 1 引数および第 3 引数は年単位で表します。

DACCSYD 関数

年次級数和法による減価償却累積額を返します。

カテゴリ: 財務関数

構文

DACCSYD(*p*,*v*,*y*)

必須引数

p
計算対象期間を示す数値です。整数でない *p* 引数は、端数期間の前後で連続する 2 期間に償却が比例配分されます。

v
償却対象の資産の初期額を表す数値です。

y
資産の耐用期間を表す数値です。

範囲: $y > 0$

詳細

DACCSYD 関数は年次級数和法による減価償却累積額を返します。式は次のとおりです。

$$DACCSYD(p,\ v,\ y) = \begin{cases} 0 & p < 0 \\ v\dfrac{int(p)\left(y - \dfrac{int(p) - 1}{2}\right) + (p - int(p))(y - int(p))}{int(y)\left(y - \dfrac{int(y) - 1}{2}\right) + (y - int(y))^2} & 0 \le p \le y \\ v & p > y \end{cases}$$

int(*y*)は *y* の整数部分を示します。*p* 引数および *y* 引数には、期間を表す同じ単位を使って表す必要があります。

サンプル

01OCT86 に取得された資産があり、減価償却対象の初期額が$1,000 で、耐用期間が 5 年であるとします。この資産の 01JAN88 までの減価償却累積額は次のとおり表せます。

```
y2=daccsyd(15/12,1000,5);
```

返される値は 400.00 です。第 1 引数および第 3 引数は年単位で表します。

DACCTAB 関数

指定テーブルから減価償却累積額を返します。

カテゴリ: 財務関数

構文

DACCTAB(*p*,*v*,*t1, . . . ,tn*)

必須引数

p
計算対象期間を示す数値です。整数でない *p* 引数は、端数期間の前後で連続する 2 期間に償却が比例配分されます。

v
償却対象の資産の初期額を表す数値です。

t1,t2, . . . ,tn
各期間の減価償却を *t1+t2+...tn* とする分数の数値です。≤ 1.

詳細

DACCTAB 関数はユーザー指定のテーブルによる減価償却累積額を返します。この関数の計算式は次のとおりです。

$$DACCTAB(p,\ v,\ t_1,\ t_2,\ \ldots,\ t_n) = \begin{cases} 0 & p \le 0 \\ v\left(t_1 + t_2 + \ldots + t_{int(p)} + (p - int(p))\,t_{int(p)+1}\right) & 0 < p < n \\ v & p \ge n \end{cases}$$

与えられた *p* について、引数 $t_1,\ t_2,\ \ldots,\ t_k$ のみを k=ceil(p)で指定する必要があります。

サンプル

償却対象の初期額が$1000 で、耐用期間が 5 年の資産があるとします。1 年目、2 年目、3 年目、4 年目、5 年目の各年の減価償却率を.15、.22、.21、.21、.21 とするテーブルによって償却した場合の 3 年目までの減価償却累積額は次のとおり表せます。

```
y3=dacctab(3,1000,.15,.22,.21,.21,.21);
```

返される値は 580.00 です。4 番目の償却率.21 と 5 番目の償却率.21 は計算に必要ないため、省略できます。

DAIRY 関数

AIRY 関数の導関数を返します。

カテゴリ: 数学関数

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 (971 ページ)のリストを参照)

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=dairy(2.0);	-0.053090384
x=dairy(-2.0);	0.6182590207

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

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。

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。

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* の有効な値は次のとおりです。

'30/360'
その月または年の実際のカレンダ日数に関わらず、1 か月を 30 日、1 年を 360 日として指定します。

月末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、またはその日付が 2 月 30 日のように無効な日付でない限り常に各月

の同じ日付に支払います。詳細については、“日数計算基準(30/360)の計算方法” (350 ページ)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日数を使用し、各年にその年の実際のカレンダ日数を使用します。

別名: *'Actual'*

'ACT/360'
各月に実際のカレンダ日数を使用し、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。

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。

$$Numberofdays = [(Y2 - Y1) * 360] + [(M2 - M1) * 30] + (D2 - D1)$$

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。

次のルールが適用されます。

- 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。

サンプル

次の例では、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。

```
data _null;
sdate='16oct78'd;
edate='16feb96'd;
actual=datdif(sdate, edate, 'act/act');
days360=datdif(sdate, edate, '30/360');
put actual= days360=;
run;
```

SAS ステートメント	結果
put actual=;	6332
put days360=;	6240

関連項目:

関数:

- “YRDIF 関数” (957 ページ)

リファレンス

Securities Industry Association 1994. *Standard Securities Calculation Methods - Fixed Income Securities Formulas for Analytic Measures*. 2 巻 New York, 米国: . 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

参照項目: “TODAY 関数” (891 ページ)

構文

DATE()

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

構文

DATEJUL(*julian-date*)

必須引数

julian-date
ユリウス暦の日付を表す SAS 数値式を指定します。SAS のユリウス日付は *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。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
Xstart=datejul(94365); put Xstart / Xstart date9.;	12783 31DEC1994
Xend=datejul(2001001); put Xend / Xend date9.;	14976 01JAN2001

関連項目:

関数:

- “JULDATE 関数” (587 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

構文

DATEPART(*datetime*)

必須引数

datetime
SAS 日時値を表す SAS 式を指定します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果

conn='01feb94:8:45'dt;
servdate=datepart(conn);
put servdate worddate.;
	February 1, 1994

関連項目:

関数:

- “DATETIME 関数” (353 ページ)
- “TIMEPART 関数” (887 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

構文

DATETIME()

サンプル

次の例では、1960 年 1 月 1 日から現在の時間までの秒数を表す SAS 値を返します。

```
when=datetime();
put when=;
```

関連項目:

関数:

- “DATE 関数” (351 ページ)
- “TIME 関数” (887 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

構文

DAY(*date*)

必須引数

date
SAS 日付値を表す SAS 式を指定します。

詳細

DAY 関数は、月の日を表す 1 ~ 31 の整数を生成します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果

now='05may97'd;
d=day(now);
put d;
	5

関連項目:

関数:

- “MONTH 関数” (652 ページ)
- “YEAR 関数” (955 ページ)

DCLOSE 関数

DOPEN 関数によって開かれたディレクトリを閉じます。

カテゴリ: 外部ファイル

構文

DCLOSE(*directory-id*)

必須引数

directory-id
DOPEN 関数によってディレクトリが開かれたときに割り当てられた識別子を指定する数値変数です。

詳細

DCLOSE は、操作に成功した場合は 0、失敗した場合は≠0 を返します。DCLOSE 関数は、DOPEN 関数によって以前に開かれたディレクトリを閉じます。DCLOSE は開かれたメンバも閉じます。

注: DATA ステップ内で開かれているすべてのディレクトリまたはメンバは、DATA ステップの終了時に自動的に閉じられます。

サンプル

サンプル 1: DCLOSE を使用してディレクトリを閉じる

次の例では、以前にファイル参照名 MYDIR が割り当てられているディレクトリを開き、メンバ数を返してからそのディレクトリを閉じます。

```
%macro memnum(filrf,path);
%let rc=%sysfunc(filename(filrf,&path));
%if %sysfunc(fileref(&filrf)) = 0 %then
%do;
/* Open the directory. */
%let did=%sysfunc(dopen(&filrf));
%put did=&did;
/* Get the member count. */
%let memcount=%sysfunc(dnum(&did));
%put &memcount members in &filrf.;
/* Close the directory. */
%let rc= %sysfunc(dclose(&did));
%end;
%else %put Invalid FILEREF;
%mend;
%memnum(MYDIR,physical-filename)
```

サンプル 2: DATA ステップ内で DCLOSE を使用する

次の例では、DATA ステップ内で DCLOSE 関数を使用します。

```
%let filrf=MYDIR;
data _null_;
rc=filename("&filrf","physical-filename");
if fileref("&filrf") = 0 then
do;
/* Open the directory. */
did=dopen("&filrf");
/* Get the member count. */
memcount=dnum(did);
put memcount "members in &filrf";
/* Close the directory. */
rc=dclose(did);
end;
else put "Invalid FILEREF";
run;
```

関連項目:

関数:

- “DOPEN 関数” (375 ページ)
- “FCLOSE 関数” (394 ページ)
- “FOPEN 関数” (476 ページ)
- “MOPEN 関数” (653 ページ)

DCREATE 関数

新しい外部ディレクトリの完全なパス名を返します。

カテゴリ: 外部ファイル

構文

DCREATE(*directory-name*<,*parent-directory*>)

必須引数

directory-name
作成するディレクトリの名前を指定する文字定数、変数または式です。この値にパス名を含めることはできません。

オプション引数

parent-directory
新しいディレクトリを作成するディレクトリの完全なパス名を含む文字定数、変数または式です。*parent-directory* の値を指定しない場合、現在のディレクトリが親ディレクトリになります。

詳細

DCREATE 関数では、動作環境でディレクトリを作成できます。ディレクトリを作成できない場合、DCREATE は空の文字列を返します。

サンプル

UNIX 動作環境で変数 DirectoryName に保存された名前を使用して新しいディレクトリを作成するには、次の形式を使用します。

```
NewDirectory=dcreate(DirectoryName,'/local/u/abcdef/');
```

Windows 動作環境で変数 DirectoryName に保存された名前を使用して新しいディレクトリを作成するには、次の形式を使用します。

```
NewDirectory=dcreate(DirectoryName,'d:¥testdir¥');
```

DEPDB 関数

定率法による減価償却を返します。

カテゴリ: 財務関数

構文

DEPDB(*p*,*v*,*y*,*r*)

必須引数

p
計算対象期間を示す数値です。整数でない *p* 引数は、端数期間の前後で連続する 2 期間に償却が比例配分されます。

v
償却対象の資産の初期額を表す数値です。

y
資産の耐用期間を表す数値です。
範囲: $y > 0$

r
償却率を分数で表した数値です。
範囲: $r \geq 0$

詳細

DEPDB 関数は、次の式で求められる、定率法を使用した減価償却を返します。

$$DEPDB(p, v, y, r) = DACCDB(p, v, y, r) - DACCDB(p - 1, v, y, r)$$

p 引数および *y* 引数には、期間を表す同じ単位を使って表す必要があります。*r* を 2 に設定すると倍額定率を得られます。

サンプル

資産の初期値は$1,000 で、耐用年数は 15 年です。200%定率法を使用して、10 年目の資産価値の減価償却を次のように表すことができます。

```
y10=depdb(10,1000,15,2);
```

返される値は 36.78 です。第 1 引数および第 3 引数は年単位で表します。

DEPDBSL 関数

定額法による減価償却に変換する定率法を返します。

カテゴリ: 財務関数

構文

DEPDBSL(*p*,*v*,*y*,*r*)

必須引数

p
計算を行う期間を表す整数です。

v
償却対象の資産の初期額を表す数値です。

y
資産の耐用期間を表す整数です。
範囲: $y > 0$

r
償却率を分数で表した数値です。
範囲: $r \geq 0$

詳細

DEPDBSL 関数は、次の式で求められる、定額法による減価償却に変換する定率法を使用して減価償却を返します。

$$DEPDBSL(p, v, y, r) = \begin{cases} 0 & p \leq 0 \\ v\frac{r}{y}\left(1 - \frac{r}{y}\right)^{p-1} & 0 < p \leq t \\ \frac{v\left(1 - \frac{r}{y}\right)t}{(y - t)} & t < p \leq y \\ 0 & p > y \end{cases}$$

前述の式には次の関係が適用されます。

$$t = int\left(y - \frac{y}{r} + 1\right)$$

int( )は数値引数の整数部分を示します。

p 引数および *y* 引数には、期間を表す同じ単位を使って表す必要があります。定額法による減価償却に変換する定率法は、各期間で償却額が大きい方の償却方法(残存償却額に対する定率法または定額法)を選択します。

サンプル

償却対象の初期額が$1,000 で、耐用期間が 10 年の資産があるとします。150%定率法を使用して、5 年目の資産価値の減価償却を次のように表すことができます。

```
y5=depdbsl(5,1000,10,1.5);
```

値 87.001041667 が返されます。第 1 引数および第 3 引数は年単位で表します。

DEPSL 関数

定額法による減価償却を返します。

カテゴリ: 財務関数

構文

DEPSL(*p*,*v*,*y*)

必須引数

p
計算対象期間を示す数値です。分数の *p* は、端数期間の前後で連続する 2 期間に償却が比例配分されます。

v
償却対象の資産の初期額を表す数値です。

y
資産の耐用期間を表す数値です。
範囲: $y > 0$

詳細

DEPSL 関数は、次の式で求められる、定額法による減価償却を返します。

$$DEPSL(p, v, y) = DACCSL(p, v, y) - DACCSL(p - 1, v, y)$$

p 引数および *y* 引数には、期間を表す同じ単位を使って表す必要があります。

サンプル

01APR86 に取得された資産があり、減価償却対象の初期額が$1,000 で、耐用期間が 10 年であるとします。1986 年の資産価値の減価償却は、次のように表されます。

```
d=depsl(9/12,1000,10);
```

返される値は 75.00 です。第 1 引数および第 3 引数は年単位で表します。

DEPSYD 関数

年次級数和法による減価償却を返します。

カテゴリ: 財務関数

構文

DEPSYD(*p*,*v*,*y*)

必須引数

p
計算対象期間を示す数値です。整数でない *p* 引数は、端数期間の前後で連続する 2 期間に償却が比例配分されます。

v
償却対象の資産の初期額を表す数値です。

y
減価償却期間数における資産の耐用年数を示す数値です。
範囲: $y > 0$

詳細

DEPSYD 関数は、次の式で求められる、年次級数和法による減価償却を返します。

$$DEPSYD(p, v, y) = DACCSYD(p, v, y) - DACCSYD(p - 1, v, y)$$

p 引数および *y* 引数には、期間を表す同じ単位を使って表す必要があります。

サンプル

01OCT86 に取得された資産があり、減価償却対象の初期額が$1,000 で、耐用期間が 5 年であるとします。1986 年および 1987 年の資産価値の減価償却は、次のように表されます。

```
y1=depsyd(3/12,1000,5);
y2=depsyd(15/12,1000,5);
```

返される値はそれぞれ 83.33 と 316.67 です。第 1 引数および第 3 引数は年単位で表します。

DEPTAB 関数

指定テーブルから減価償却を返します。

カテゴリ: 財務関数

構文

DEPTAB(*p*,*v*,*t1, . . . ,tn*)

必須引数

p
計算対象期間を示す数値です。整数でない *p* 引数は、端数期間の前後で連続する 2 期間に償却が比例配分されます。

v
償却対象の資産の初期額を表す数値です。

t1,t2, . . . ,tn
各期間の減価償却を *t1+t2+...tn* とする分数の数値です。≤ 1.

詳細

DEPTAB 関数は、指定したテーブルを使用して減価償却を返します。式は次のとおりです。

$$DEPTAB(p, v, t_1, t_2, ..., t_n) = DACCTAB(p, v, t_1, t_2, ..., t_n) - DACCTAB(p - 1, v, t_1, t_2, ..., t_n)$$

与えられた p について、引数 $t_1, t_2, \ldots, t_k$ のみを k=ceil(p)で指定する必要があります。

サンプル

償却対象の初期額が$1,000 で、耐用期間が 5 年の資産があるとします。1 年目、2 年目、3 年目、4 年目、5 年目の各年の減価償却率を.15、.22、.21、.21、.21 とするテーブルによって償却した場合の 3 年目までの減価償却は次のとおり表せます。

```
y3=deptab(3,1000,.15,.22,.21,.21,.21);
```

返される値は 210.00 です。4 番目の償却率.21 と 5 番目の償却率.21 は計算に必要ないため、省略できます。

DEQUOTE 関数

引用符で始まる文字列から一致する引用符を削除し、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: 文字関数

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

DEQUOTE(*string*)

必須引数

string
文字定数、変数または式を指定します。

詳細

返される変数の長さ

DATA ステップで、まだ長さが割り当てられていない変数に DEQUOTE 関数から値が返される場合、その変数に引数の長さが設定されます。

基本

DEQUOTE 関数で返される値は次のように決定されます。

- *string* の最初の文字が一重引用符または二重引用符でない場合、DEQUOTE は *string* をそのまま返します。
- *string* の最初の 2 文字が両方とも一重引用符または二重引用符で、3 番目の文字が異なる種類の引用符の場合、DEQUOTE は長さが 0 の結果を返します。
- *string* の最初の文字が一重引用符の場合、DEQUOTE 関数は結果からその一重引用符を削除します。次に *string* を左から右にスキャンし、他の一重引用符をさらに探します。それぞれの連続する一重引用符のペアは 1 つの一重引用符に変更されます。*string* 内に終了引用符のない最初の一重引用符は削除され、その引用符の右側にあるすべての文字も削除されます。

- *string* の最初の文字が二重引用符の場合、DEQUOTE 関数は結果からその二重引用符を削除します。次に *string* を左から右にスキャンし、他の二重引用符をさらに探します。それぞれの連続する二重引用符のペアは 1 つの二重引用符に変更されます。*string* 内に終了引用符のない最初の二重引用符は削除され、その引用符の右側にあるすべての文字も削除されます。

注: string が引用符で囲まれた定数の場合、それらの引用符は *string* 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。

サンプル

この例では、DATA ステップ内で DEQUOTE を使用します。

```
data test;
input string $60.;
result = dequote(string);
datalines;
No quotation marks, no change
No "leading" quotation marks, no change
"Matching double quotation marks are removed"
'Matching single quotation marks are removed'
"Paired ""quotation marks"" are reduced"
'Paired '' quotation marks '' are reduced'
"Single 'quotation marks' inside '' double'' quotation marks are unchanged"
'Double "quotation marks" inside ""single"" quotation marks are unchanged'
"No matching quotation mark, no problem
Don't remove this apostrophe
"Text after the matching quotation mark" is "deleted"
;
proc print noobs;
title 'Input Strings and Output Results from DEQUOTE';
run;
```

画面 2.29 DEQUOTE 関数を使用して一致する引用符を削除する

Input Strings and Output Results from DEQUOTE

string	result
No quotation marks, no change	No quotation marks, no change
No "leading" quotation marks, no change	No "leading" quotation marks, no change
"Matching double quotation marks are removed"	Matching double quotation marks are removed
'Matching single quotation marks are removed'	Matching single quotation marks are removed
"Paired ""quotation marks"" are reduced"	Paired "quotation marks" are reduced
'Paired " quotation marks " are reduced'	Paired ' quotation marks ' are reduced
"Single 'quotation marks' inside " double" quotation marks	Single 'quotation marks' inside " double" quotation marks
'Double "quotation marks" inside ""single"" quotation marks	Double "quotation marks" inside ""single"" quotation marks
"No matching quotation mark, no problem	No matching quotation mark, no problem
Don't remove this apostrophe	Don't remove this apostrophe
"Text after the matching quotation mark" is "deleted"	Text after the matching quotation mark

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: 数学関数

構文

DEVIANCE(*distribution*, *variable*, *shape-parameters*<,*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。有効な分布については、次の表を参照してください。

分布	引数
ベルヌーイ	'BERNOULLI' | 'BERN'
二項	'BINOMIAL' | 'BINO'
ガンマ	'GAMMA'

分布	引数
逆ガウス(Wald)	'IGAUSS' | 'WALD'
正規	'NORMAL' | 'GAUSSIAN'
ポアソン	'POISSON' | 'POIS'

変数
数値定数、変数または式です。

shape-parameter
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

変数
成功した場合は 1、失敗した場合は 0 の二値の数値ランダム変数です。

p
$\varepsilon \le p \le 1-\varepsilon$ 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 $0 \le p \le \varepsilon$ のすべての p 値は、ε で置換されます。間隔 $1-\varepsilon \le p \le 1$ のすべての p 値は、$1-\varepsilon$ で置換されます。

DEVIANCE 関数は、成功が 1 のランダム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。

$$DEVIANCE('BERN', variable, p, \varepsilon) = \begin{cases} -2\log(1-p) & x = 0 \\ -2\log(p) & x = 1 \\ . & otherwise \end{cases}$$

二項分布

DEVIANCE('BINO', *variable*, *μ*, *n* <, *ε*>)

引数

変数
成功数が含まれる数値ランダム変数です。

範囲: $0 \le variable \le 1$

μ
数値の平均パラメータです。

範囲: $n\varepsilon \le \mu \le n(1-\varepsilon)$

n
Bernoulli 試行数の整数パラメータです。

範囲: $n \geq 0$

ε

限界 μ に使用される任意の正の数値です。間隔 $0 \leq \mu \leq n\varepsilon$ のすべての μ 値は、$n\varepsilon$ で置換されます。間隔 $n(1-\varepsilon) \leq \mu \leq n$ 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。二項分布の DEVIANCE 関数は次の式で表されます。x はランダム変数です。

$$DEVIANCE('BINO', x, \mu, n) = \begin{cases} . & x < 0 \\ 2\left(x\log\left(\frac{x}{\mu}\right) + (n-x)\log\left(\frac{n-x}{n-\mu}\right)\right) & 0 \leq x \leq n \\ . & x > n \end{cases}$$

ガンマ分布

DEVIANCE('GAMMA', *variable*, μ <, ε>)

引数

変数

数値のランダム変数です。

範囲: $variable \geq \varepsilon$

μ

数値の平均パラメータです。

範囲: $\mu \geq \varepsilon$

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 $0 \leq variable \leq \varepsilon$ のすべての *variable* 値は、ε で置換されます。間隔 $0 \leq \mu \leq \varepsilon$ 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 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 はランダム変数です。

$$DEVIANCE('GAMMA', x, \mu) = \begin{cases} . & x < 0 \\ 2\left(-\log\left(\frac{x}{\mu}\right) + \frac{x-\mu}{\mu}\right) & x \geq \varepsilon,\ \mu \geq \varepsilon \end{cases}$$

逆ガウス(Wald)分布

DEVIANCE('IGAUSS' | 'WALD', *variable*, μ <, ε>)

引数

変数

数値のランダム変数です。

範囲: $variable \geq \varepsilon$

μ

数値の平均パラメータです。

範囲: $\mu \geq \varepsilon$

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 $0 \leq variable \leq \varepsilon$ のすべての *variable* 値は、ε で置換されます。間隔 $0 \leq \mu \leq \varepsilon$ 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 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 はランダム変数です。

$$DEVIANCE('IGAUSS', x, \mu) = \begin{cases} . & x < 0 \\ \dfrac{(x-\mu)^2}{\mu^2 x} & x \geq \varepsilon,\ \mu \geq \varepsilon \end{cases}$$

正規分布

DEVIANCE('NORMAL' | 'GAUSSIAN', *variable*, μ)

引数

変数
数値のランダム変数です。

μ
数値の平均パラメータです。

DEVIANCE 関数は、平均パラメータ 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 はランダム変数です。

$$DEVIANCE('NORMAL', x, \mu) = (x-\mu)^2$$

Poisson 分布

DEVIANCE('POISSON', *variable*, μ <, ε>)

引数

変数
数値のランダム変数です。

範囲: $variable \geq 0$

μ
数値の平均パラメータです。

範囲: $\mu \geq \varepsilon$

ε
限界 μ に使用される任意の正の数値です。間隔 $0 \leq \mu \leq \varepsilon$ 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 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 はランダム変数です。

$$DEVIANCE('POISSON', x, \mu) = \begin{cases} . & x < 0 \\ 2\left(x\log\left(\dfrac{x}{\mu}\right) - (x-\mu)\right) & x \geq 0,\ \mu \geq \varepsilon \end{cases}$$

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

構文

DHMS(*date*,*hour*,*minute*,*second*)

必須引数

date
SAS 日付値を表す SAS 式を指定します。

時
数値です。

分
数値です。

秒
数値です。

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。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
dtid=dhms('01jan03'd,15,30,15); put dtid; put dtid datetime.;	1357054215 01JAN03:15:30:15
dtid2=dhms('01jan03'd,15,30,61); put dtid2; put dtid2 datetime.;	1357054261 01JAN03:15:31:01
dtid3=dhms('01jan03'd,15,.5,15); put dtid3; put dtid3 datetime.;	1357052445 01JAN03:15:00:45

次の SAS ステートメントに、SAS 日付値と SAS 時間値を SAS 日時値に組み合わせる方法を示します。2003 年 4 月 2 日の 15:05:02 にこれらのステートメントを実行した場合、次の結果になります。

SAS ステートメント	結果

day=date();
time=time();
sasdt=dhms(day,0,0,time);
put sasdt datetime.;
	02APR03:15:05:02

関連項目:

関数:

- “HMS 関数” (519 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: 特殊関数

構文

DIF<*n*> (*argument*)

必須引数

引数
数値の定数、変数または式を指定します。

オプション引数

n
ラグ数を指定します。

詳細

DIF 関数、DIF1、DIF2、...、DIF100 は、引数とその *n* 番目のラグの最初の差分を返します。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。

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、2 番目の差分 DIF(DIF(X))とは異なります。

サンプル

この例では、LAG 関数と DIF 関数の違いを示します。

```
data two;
input X @@;
Z=lag(x);
D=dif(x);
datalines;
1 2 6 4 7
;
proc print data=two;
run;
```

画面 2.30 DIF 関数と LAG 関数の違い

The SAS System

Obs	X	Z	D
1	1	.	.
2	2	1	1
3	6	2	4
4	4	6	-2
5	7	4	3

関連項目:

関数:

- “LAG 関数” (590 ページ)

DIGAMMA 関数

ディガンマ関数の値を返します。

カテゴリ: 数学関数

構文

DIGAMMA(*argument*)

必須引数

引数

数値の定数、変数または式を指定します。

制限事項: 正でない整数は無効です。

詳細

DIGAMMA 関数は、次の式で求められる比率を返します。

$$\Psi(x) = \Gamma'(x) / \Gamma(x)$$

ここで、$\Gamma(.)$ 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。*argument*>0 の場合、DIGAMMA 関数は LGAMMA 関数の導関数です。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=digamma(1.0);	-0.577215665

DIM 関数

配列にある要素数を返します。

カテゴリ: 配列

構文

DIM<*n*> (*array-name*)

DIM(*array-name*,*bound-n*)

必須引数

array-name
同じ DATA ステップで以前に定義された配列の名前を指定します。この引数には定数、変数または式は使用できません。

bound-n
多次元配列にある要素数を確認するディメンションを指定する、数値定数、変数または式です。*n* が指定されていない場合のみ *bound-n* を使用します。

オプション引数

n
多次元配列にある要素数を確認するディメンションを指定します。*n* 値が指定されていない場合、DIM 関数は配列の最初のディメンションにある要素数を返します。

詳細

DIM 関数は、1 次元配列にある要素数、またはディメンションの下限が 1 の多次元配列内の指定したディメンションにある要素数を返します。配列処理で DIM を使用することで、配列の要素数を変更するたびに反復 DO グループの上限が変更されることを防ぎます。

比較

- DIM は、常に配列のディメンションにある要素の合計数を返します。
- HBOUND は、配列のディメンションでの上限のリテラル値を返します。

注: 配列のディメンションの下限値が 1 以外で、上限値が配列のディメンションにある要素の合計数以外の場合、この違いが重要です。

サンプル

サンプル 1: 1 次元配列

この例では、DIM は 5 の値を返します。そのため、SAS は DO ループでステートメントを 5 回繰り返します。

```
 array big{5} weight sex height state city;
do i=1 to dim(big);
more SAS statements;
end;
```

サンプル 2: 多次元配列

この例では、多次元配列に DIM 関数を指定する 2 つの方法を示します。SAS コードの例に続く表に示すように、いずれの方法でも DIM で同じ値が返されます。

```
    array mult{5,10,2} mult1-mult100;
```

構文	別の構文	値
DIM(MULT)	DIM(MULT,1)	5
DIM2(MULT)	DIM(MULT,2)	10
DIM3(MULT)	DIM(MULT,3)	2

関連項目:

関数:

- “HBOUND 関数” (517 ページ)
- “LBOUND 関数” (598 ページ)

ステートメント:

- “ARRAY ステートメント” (*SAS ステートメント: リファレンス*)
- “配列参照ステートメント” (*SAS ステートメント: リファレンス*)

その他のリファレンス:

- “配列処理” (*SAS 言語リファレンス: 解説編* 23 章)

DINFO 関数

ディレクトリの情報を返します。

カテゴリ: 外部ファイル

参照項目: “DINFO Function: Windows” in *SAS Companion for Windows*
“DINFO Function: UNIX” in *SAS Companion for UNIX Environments*
“DINFO Function: z/OS” in *SAS Companion for z/OS*

構文

DINFO(*directory-id*,*info-item*)

必須引数

directory-id
DOPEN 関数によってディレクトリが開かれたときに割り当てられた識別子を指定する数値変数です。

info-item
取得される情報項目を指定する文字定数、変数または式です。DINFO は、*info-item* 引数の値が無効な場合は空白を返します。使用可能な情報は、動作環境により異なります。

詳細

使用可能なシステム依存のディレクトリ情報項目の名前を確認するには、DOPTNAME 関数を使用します。使用可能なディレクトリ情報項目数を確認するには、DOPTNUM 関数を使用します。

動作環境の情報
DINFO は、システム依存のディレクトリパラメータ値を返します。システム依存のディレクトリパラメータの詳細については、動作環境に関する SAS のドキュメントを参照してください。

サンプル

サンプル 1: DINFO を使用してディレクトリの情報を返す

次の例では、ディレクトリ MYDIR を開き、使用可能なディレクトリ情報項目数を確認して最後の項目の値を取得します。

```
%let filrf=MYDIR;
%let rc=%sysfunc(filename(filrf,physical-name));
%let did=%sysfunc(dopen(&filrf));
%let numopts=%sysfunc(doptnum(&did));
%let foption=%sysfunc(doptname(&did,&numopts));
%let charval=%sysfunc(dinfo(&did,&foption));
%let rc=%sysfunc(dclose(&did));
```

サンプル 2: DATA ステップ内で DINFO を使用する

次の例では、各ディレクトリ情報項目の名前と値を含むデータセットを作成します。

```
data diropts;
length foption $ 12 charval $ 40;
keep foption charval;
rc=filename("mydir","physical-name");
did=dopen("mydir");
numopts=doptnum(did);
do i=1 to numopts;
foption=doptname(did,i);
charval=dinfo(did,foption);
output;
end;
run;
```

関連項目:

関数:

- “DOPEN 関数” (375 ページ)
- “DOPTNAME 関数” (377 ページ)
- “DOPTNUM 関数” (378 ページ)
- “FINFO 関数” (463 ページ)
- “FOPTNAME 関数” (478 ページ)
- “FOPTNUM 関数” (480 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

構文

DIVIDE(*x*, *y*)

必須引数

x
数値定数、変数または式です。

y
数値定数、変数または式です。

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。

画面 2.31 DIVIDE 関数によって返される値

y \ x	positive	zero	negative	.I	.M	._	other
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。

サンプル

次に、DIVIDE 関数を使用した結果の例を示します。

```
data _null_;
a = divide(1, 0);
put +3 a= '(infinity)';
b = divide(2, .I);
put +3 b=;
c = divide(.I, -1);
put +3 c= '(minus infinity)';
d = divide(constant('big'), constant('small'));
put +3 d= '(infinity because of overflow)';
run;
```

SAS は次の出力をログに書き込みます。

```
 a=I (infinity)
b=0
c=M (minus infinity)
d=I (infinity because of overflow)
```

DNUM 関数

ディレクトリ内のメンバ数を返します。

カテゴリ: 外部ファイル

構文

DNUM(*directory-id*)

必須引数

directory-id
DOPEN 関数によってディレクトリが開かれたときに割り当てられた識別子を指定する数値変数です。

詳細

DNUM を使用して、DREAD に渡すことができる最大メンバ数を確認します。

サンプル

サンプル 1: DNUM を使用してメンバ数を返す

次の例では、ディレクトリ MYDIR を開き、メンバ数を確認してディレクトリを閉じます。

```
%let filrf=MYDIR;
%let rc=%sysfunc(filename(filrf,physical-name));
%let did=%sysfunc(dopen(&filrf));
%let memcount=%sysfunc(dnum(&did));
%let rc=%sysfunc(dclose(&did));
```

サンプル 2: DATA ステップ内で DNUM を使用する

次の例では、MYDIR というディレクトリ内のメンバ数を返す DATA ステップを作成します。

```
data _null_;
rc=filename("mydir","physical-name");
did=dopen("mydir");
memcount=dnum(did);
rc=dclose(did);
run;
```

関連項目:

関数:

- “DOPEN 関数” (375 ページ)
- “DREAD 関数” (379 ページ)

DOPEN 関数

ディレクトリを開き、ディレクトリ識別子の値を返します。

カテゴリ: 外部ファイル

参照項目: “DOPEN Function: Windows” in *SAS Companion for Windows*
“DOPEN Function: UNIX” in *SAS Companion for UNIX Environments*
“DOPEN Function: z/OS” in *SAS Companion for z/OS*

構文

DOPEN(*fileref*)

必須引数

fileref
ディレクトリに割り当てられたファイル参照名を指定する文字定数、変数または式です。

制限事項: DOPEN を呼び出す前に、ファイル参照名をディレクトリに関連付ける必要があります。

詳細

DOPEN は、ディレクトリを開き、他の SAS 外部ファイルアクセス関数でそのディレクトリの識別に使用するディレクトリ識別子の値(0 より大きい値)を返します。ディレクトリを開けなかった場合、DOPEN は 0 を返します。SYSMSG 関数を呼び出すことでそのエラーメッセージを取得できます。開かれるディレクトリは、ファイル参照名で識別される必要があります。ファイル参照名を割り当てるには、FILENAME ステートメントまたは FILENAME 外部ファイルアクセス関数を使用します。一部の動作環境では、システムコマンドを使用してファイル参照名を割り当てることもできます。

マクロから DOPEN 関数を呼び出した場合、呼び出し結果はその結果がマクロ内で関数に渡される場合にのみ有効です。DATA ステップから DOPEN 関数を呼び出した場合、結果はその結果が同じ DATA ステップ内で関数に渡される場合にのみ有効です。

動作環境の情報
この関数の説明で使用される、SAS 外部ファイルアクセス関数に関連する用語*ディレクトリ*は、動作環境で管理されるファイルをグループ化した集合を指します。このようなグルーピングは、異なる動作環境ではディレクトリ、サブディレクトリ、フォルダ、MACLIB、分割データセットなど、さまざまな名前で呼ばれます。詳細については、動作環境に関する SAS のドキュメントを参照してください。

サンプル

サンプル 1: DOPEN を使用してディレクトリを開く

この例では、ファイル参照名 MYDIR をディレクトリに割り当てます。DOPEN を使用してディレクトリを開きます。DOPTNUM で使用可能なシステム依存のディレクトリ情報項目数を確認し、DCLOSE でディレクトリを閉じます。

```
%let filrf=MYDIR;
%let rc=%sysfunc(filename(filrf,physical-name));
%let did=%sysfunc(dopen(&filrf));
%let infocnt=%sysfunc(doptnum(&did));
%let rc=%sysfunc(dclose(&did));
```

サンプル 2: DATA ステップ内で DOPEN を使用する

この例では、DATA ステップ内で処理するディレクトリを開きます。

```
data _null_;
drop rc did;
rc=filename("mydir","physical-name");
did=dopen("mydir");
if did > 0 then do;
...more statements...
end;
else do;
```

```
msg=sysmsg();
put msg;
end;
run;
```

関連項目:

関数:

- “DCLOSE 関数” (354 ページ)
- “DOPTNUM 関数” (378 ページ)
- “FOPEN 関数” (476 ページ)
- “MOPEN 関数” (653 ページ)
- “SYSMSG 関数” (880 ページ)

DOPTNAME 関数

ディレクトリ属性情報を返します。

カテゴリ: 外部ファイル

参照項目: “DOPTNAME Function: Windows” in *SAS Companion for Windows*
“DOPTNAME Function: UNIX” in *SAS Companion for UNIX Environments*
“DOPTNAME Function: z/OS” in *SAS Companion for z/OS*

構文

DOPTNAME(*directory-id*,*nval*)

必須引数

directory-id
DOPEN 関数によってディレクトリが開かれたときに割り当てられた識別子を指定する数値変数です。

nval
オプションのシーケンス番号を指定する数値定数、変数または式です。

詳細

動作環境の情報
番号、名前およびディレクトリ情報の特性は、動作環境によって異なります。ディレクトリで使用可能なオプション数は、動作環境によって異なります。詳細については、動作環境に関する SAS のドキュメントを参照してください。

サンプル

サンプル 1: DOPTNAME を使用してディレクトリ属性情報を取得する

この例では、ファイル参照名 MYDIR のディレクトリを開き、すべてのシステム依存のディレクトリ情報項目を取得して SAS ログに書き込み、ディレクトリを閉じます。

```
%let filrf=mydir;
%let rc=%sysfunc(filename(filrf,physical-name));
%let did=%sysfunc(dopen(&filrf));
%let infocnt=%sysfunc(doptnum(&did));
%do j=1 %to &infocnt;
%let opt=%sysfunc(doptname(&did,&j));
%put Directory information=&opt;
%end;
%let rc=%sysfunc(dclose(&did));
```

サンプル 2: DATA ステップ内で DOPTNAME を使用する

次の例では、各ディレクトリ情報項目の名前と値を含むデータセットを作成します。

```
data diropts;
length optname $ 12 optval $ 40;
keep optname optval;
rc=filename("mydir","physical-name");
did=dopen("mydir");
numopts=doptnum(did);
do i=1 to numopts;
optname=doptname(did,i);
optval=dinfo(did,optname);
output;
end;
run;
```

関連項目:

関数:

- “DINFO 関数” (371 ページ)
- “DOPEN 関数” (375 ページ)
- “DOPTNUM 関数” (378 ページ)

DOPTNUM 関数

ディレクトリで使用可能な情報項目数を返します。

カテゴリ: 外部ファイル

参照項目: “DOPTNUM Function: Windows” in *SAS Companion for Windows*
“DOPTNUM Function: UNIX” in *SAS Companion for UNIX Environments*
“DOPTNUM Function: z/OS” in *SAS Companion for z/OS*

構文

DOPTNUM(*directory-id*)

必須引数

directory-id
DOPEN 関数によってディレクトリが開かれたときに割り当てられた識別子を指定する数値変数です。

詳細

動作環境の情報
番号、名前およびディレクトリ情報の特性は、動作環境によって異なります。ディレクトリで使用可能なオプション数は、動作環境によって異なります。詳細については、動作環境に関する SAS のドキュメントを参照してください。

サンプル

サンプル 1: 情報項目数を取得する

この例では、ディレクトリ MYDIR で使用可能なシステム依存のディレクトリ情報項目数を取得し、ディレクトリを閉じます。

```
%let filrf=mydir;
%let rc=%sysfunc(filename(filrf,physical-name));
%let did=%sysfunc(dopen(&filrf));
%let infocnt=%sysfunc(doptnum(&did));
%let rc=%sysfunc(dclose(&did));
```

サンプル 2: DATA ステップ内で DOPTNUM を使用する

この例では、MYDIR ディレクトリで使用可能なシステム依存の情報項目数を取得するデータセットを作成します。

```
data _null_;
rc=filename("mydir","physical-name");
did=dopen("mydir");
infocnt=doptnum(did);
rc=dclose(did);
run;
```

関連項目:

関数:

- “DINFO 関数” (371 ページ)
- “DOPEN 関数” (375 ページ)
- “DOPTNAME 関数” (377 ページ)

DREAD 関数

ディレクトリのメンバ名を返します。

カテゴリ: 外部ファイル

構文

DREAD(*directory-id*,*nval*)

必須引数

directory-id
DOPEN 関数によってディレクトリが開かれたときに割り当てられた識別子を指定する数値です。

nval
ディレクトリ内のメンバの通し番号を指定する数値定数、変数または式です。

詳細

DREAD は、エラーが発生した場合(*nval* が範囲外の場合など)は空白を返します。DNUM を使用して、DREAD に渡すことができる最大メンバ数を確認します。

サンプル

この例では、ファイル参照名 MYDIR で識別されるディレクトリを開き、メンバ数を取得してその数を変数 MEMCOUNT に代入します。次に最後のメンバ名を取得し、その名前を変数 LSTNAME に代入してディレクトリを閉じます。

```
%let filrf=mydir;
%let rc=%sysfunc(filename(filrf,physical-name));
%let did=%sysfunc(dopen(&filrf));
%let lstname=;
%let memcount=%sysfunc(dnum(&did));
%if &memcount > 0 %then
%let lstname=%sysfunc(dread(&did,&memcount));
%let rc=%sysfunc(dclose(&did));
```

関連項目:

関数:

- “DNUM 関数” (374 ページ)
- “DOPEN 関数” (375 ページ)

DROPNOTE 関数

SAS データセットまたは外部ファイルからメモマーカーを削除します。

カテゴリ: SAS ファイル I/O 関数
外部ファイル

構文

DROPNOTE(*data-set-id | file-id*,*note-id*)

必須引数

data-set-id* | *file-id
通常、OPEN 関数または FOPEN 関数によってデータセットまたは外部ファイルが開かれたときに割り当てられた識別子を指定する数値変数です。

note-id
NOTE 関数または FNOTE 関数によって割り当てられた識別子を指定する数値です。

詳細

DROPNOTE は、NOTE または FNOTE によって設定されたマーカーを削除します。成功した場合は 0、失敗した場合は≠0 を返します。

サンプル

この例では、SAS データセット MYDATA を開き、開始オブザベーションをフェッチしてデータセットの先頭にメモ ID を設定します。POINT を使用して開始オブザベーションに戻り、DROPNOTE を使用してメモ ID を削除します。

```
%let dsid=%sysfunc(open(mydata,i));
%let rc=%sysfunc(fetch(&dsid));
%let noteid=%sysfunc(note(&dsid));
more macro statements
%let rc=%sysfunc(point(&dsid,&noteid));
%let rc=%sysfunc(fetch(&dsid));
%let rc=%sysfunc(dropnote(&dsid,&noteid));
```

関連項目:

関数:

- “FETCH 関数” (398 ページ)
- “FNOTE 関数” (474 ページ)
- “FOPEN 関数” (476 ページ)
- “FPOINT 関数” (481 ページ)
- “NOTE 関数” (674 ページ)
- “OPEN 関数” (697 ページ)
- “POINT 関数” (725 ページ)

DSNAME 関数

データセット識別子と関連付けられた SAS データセット名を返します。

カテゴリ: SAS ファイル I/O 関数

構文

DSNAME(*data-set-id*)

必須引数

data-set-id
OPEN 関数によって返されたデータセット識別子を指定する数値変数です。

詳細

DSNAME は、データセット識別子と関連付けられた SAS データセット名を返すか、データセット識別子が無効な場合は空白を返します。

サンプル

この例では、変数 DSID と関連付けられた SAS データセットの名前を確認し、SAS ログにその名前を表示します。

```
%let dsid=%sysfunc(open(sasuser.houses,i));
%put The current open data set
is %sysfunc(dsname(&dsid)).;
```

関連項目:

関数:

- “OPEN 関数” (697 ページ)

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: 財務関数

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: $y > 0$

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: $f > 0$

c(1), ... ,c(k)
キャッシュフローのリストを指定します。

詳細

DUR 関数は次の値を返します。

$$C = \sum_{k=1}^{K} \frac{k\left(\frac{c(k)}{(1+y)^{\frac{k}{f}}}\right)}{(P(1+y)f)}$$

前述の式には次の関係が適用されます。

$$P = \sum_{k=1}^{K} \frac{c(k)}{(1+y)^{\frac{k}{f}}}$$

サンプル

```
data _null_;
d=dur(1/20,1,.33,.44,.55,.49,.50,.22,.4,.8,.01,.36,.2,.4);
put d;
run;
```

返される値は 5.28402 です。

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: 財務関数

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: $A > 0$

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: $0 \le c < 1$

n

期間当たりのクーポン数を指定します。

範囲: $n > 0$ の整数

K

クーポンの残数を指定します。

範囲: $K > 0$ の整数

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: $0 < k_0 \le 1/n$

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: $y > 0$

詳細

DURP 関数は次の値を返します。

$$D = \frac{1}{n}\frac{\sum_{k=1}^{K} t_k \frac{c(k)}{\left(1+\frac{y}{n}\right)^{t_k}}}{P\left(1+\frac{y}{n}\right)}$$

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。

$$P = \sum_{k=1}^{K} \frac{c(k)}{\left(1+\frac{y}{n}\right)^{t_k}}$$

サンプル

```
data _null_;
d=durp(1000,1/100,4,14,.33/2,.10);
put d;
run;
```

返される値は 3.26496 です。

EFFRATE 関数

有効な年利を返します。

カテゴリ: 財務関数

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。

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。

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。

サンプル

- 名目利率が 10%で利息が毎月複利計算される場合、対応する有効な利率は次のように表されます。

  ```
  effective_rate1 = EFFRATE('MONTH', 10);
  ```
- 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、対応する有効な利率は次のように表されます。

  ```
  effective-rate2 = EFFRATE('QUARTER', 10);
  ```

ENVLEN 関数

環境変数の長さを返します。

カテゴリ: SAS ファイル I/O 関数

構文

ENVLEN(*argument*)

必須引数

引数

オペレーティングシステム環境変数の名前が含まれる文字変数を指定します。*argument* は引用符で囲みます。

詳細

ENVLEN 関数は、オペレーティングシステム環境変数の値の長さを返します。環境変数が存在しない場合、SAS は–1 を返します。

動作環境の情報

argument の値は動作環境に固有です。

サンプル

次の例は、説明のみを目的としています。返される実際の値は、SAS がインストールされているコンピュータによって異なります。

SAS ステートメント	結果
/* Windows operating environment */ x=envlen("PATH"); put x;	309
/* UNIX operating environment */ y=envlen("PATH"); put y;	365
z=envlen("THIS IS NOT DEFINED"); put z;	-1

ERF 関数

(正規)誤差関数の値を返します。

カテゴリ: 数学関数

構文

ERF(*argument*)

必須引数

引数

数値の定数、変数または式を指定します。

詳細

ERF 関数は、次の式で求められる積分値を返します。

$$ERF(x) = \frac{2}{\sqrt{\pi}} \int_0^x \varepsilon^{-z^2} dz$$

サンプル

ERF 関数を使用して、平均値が 0 で標準偏差が 1 の正規分布に従うランダム変数の値が 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));
```

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
y=erf(1.0);	0.8427007929
y=erf(−1.0);	−0.842700793

ERFC 関数

相補(正規)誤差関数の値を返します。

カテゴリ: 数学関数

構文

ERFC(*argument*)

必須引数

引数
数値の定数、変数または式を指定します。

詳細

ERFC 関数は、ERF 関数(1 − ERF(*argument*))の補数を返します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=erfc(1.0);	0.1572992071
x=erfc(−1.0);	.8427007929

EUCLID 関数

非欠損値引数のユークリッドノルムを返します。

カテゴリ: 記述統計

構文

EUCLID(*value-1* <,*value-2 ...*>)

必須引数

value
数値の定数、変数または式を指定します。

詳細

すべての引数が欠損値の場合、結果は欠損値になります。それ以外の場合、非欠損値引数のユークリッドノルムを返します。

次の例では、*x*1, *x*2, ..., *xn* は非欠損値引数の値です。

$$EUCLID(x1,\ x2,\ ...,\ xn) = \sqrt{x1^2 + x2^2 + ... + xn^2}$$

サンプル

サンプル 1: 非欠損値引数のユークリッドノルムの計算

次の例では、非欠損値引数のユークリッドノルムを返します。

```
data _null_;
x=euclid(.,3,0,.q,-4);
put x=;
run;
```

SAS は次の出力を口グに書き込みます。

```
x=5
```

サンプル 2: 変数リスト使用時のユークリッドノルムの計算

次の例では、変数リストを使用してユークリッドノルムを計算します。

```
data _null_;
x1 = 1;
x2 = 3;
x3 = 4;
x4 = 3;
x5 = 1;
x = euclid(of x1-x5);
put x=;
run;
```

SAS は次の出力を口グに書き込みます。

```
x=6
```

関連項目:

関数:

- “RMS 関数” (809 ページ)

- “LPNORM 関数” (632 ページ)

EXIST 関数

SAS ライブラリメンバの存在を確認します。

カテゴリ: SAS ファイル I/O 関数

構文

EXIST(*member-name*<,*member-type*<, *generation*>>)

必須引数

member-name
SAS ライブラリメンバを指定する文字定数、変数または式です。*member-name* が空白または NULL 文字列の場合、EXIST はメンバ名として _LAST_ システム変数の値を使用します。

オプション引数

member-type
SAS ライブラリメンバの種類を指定する文字定数、変数または式です。一般的なメンバの種類には、ACCESS、CATALOG、DATA、VIEW などがあります。*member-type* を指定しない場合、メンバの種類は DATA とみなされます。

generation
存在を確認する SAS データセットの世代番号を指定する数値定数、変数または式です。*member-type* が DATA 以外の場合、*generation* は無視されます。

正の数は、その世代番号による履歴バージョンへの絶対参照です。負の数は、以前の最も新しいバージョンから最も古いバージョンまでの、基本バージョンに対する履歴バージョンへの相対参照です。たとえば、–1 は最も新しいバージョンまたは基本バージョンより 1 つ前のバージョンを示します。0 は相対世代番号として扱われます。

詳細

シーケンシャルライブラリを使用する場合、EXIST 関数の結果は未定義になります。シーケンシャルライブラリを使用*しない*場合、EXIST 関数は、ライブラリメンバが存在する場合は 1、*member-name* が存在しないか *member-type* が無効な場合は 0 を返します。

カタログ内のエントリの存在を確認するには、CEXIST 関数を使用します。

サンプル

サンプル 1: データセットの存在確認

この例では、データセットの存在を確認します。データセットが存在しない場合、この例ではログにメッセージを表示します。

```
%let dsname=sasuser.houses;
%macro opends(name);
%if %sysfunc(exist(&name)) %then
```

```
%let dsid=%sysfunc(open(&name,i));
%else %put Data set &name does not exist.;
%mend opends;
%opends(&dsname);
```

サンプル 2: データビューの存在確認

この例では、SAS ビュー TEST.MYVIEW の存在を確認します。ビューが存在しない場合、この例ではログにメッセージを表示します。

```
data _null_;
dsname="test.myview";
if (exist(dsname,"VIEW")) then
dsid=open(dsname,"i");
else put dsname 'does not exist.';
run;
```

サンプル 3: 世代データセットの存在確認

この例では、正の世代番号(絶対参照)を使用して世代データセットの存在を確認します。

```
data new(genmax=3);
x=1;
run;
data new;
x=99;
run;
data new;
x=100;
run;
data new;
x=101;
run;
data _null_;
test=exist('new', 'DATA', 4);
put test=;
test=exist('new', 'DATA', 3);
put test=;
test=exist('new', 'DATA', 2);
put test=;
test=exist('new', 'DATA', 1);
put test=;
run;
```

次の行が SAS ログに書き込まれます。

```
test=1
test=1
test=1
test=0
```

この例を次のように変更することで、負の数(相対参照)を使用して世代データセットの存在を確認できます。

```
data new2(genmax=3);
x=1;
run;
data new2;
x=99;
```

```
run;
data new2;
x=100;
run;
data new2;
x=101;
run;
data _null_;
test=exist('new2', 'DATA', 0);
put test=;
test=exist('new2', 'DATA', -1);
put test=;
test=exist('new2', 'DATA', -2);
put test=;
test=exist('new2', 'DATA', -3);
put test=;
test=exist('new2', 'DATA', -4);
put test=;
run;
```

次の行が SAS ログに書き込まれます。

```
test=1
test=1
test=1
test=0
test=0
```

関連項目:

関数:

- “CEXIST 関数” (290 ページ)
- “FEXIST 関数” (400 ページ)
- “FILEEXIST 関数” (403 ページ)

EXP 関数

指数関数の値を返します。

カテゴリ: 数学関数

構文

EXP(*argument*)

必須引数

引数

数値の定数、変数または式を指定します。詳細については、“SAS 式の定義” (*SAS 言語リファレンス: 解説編* 6 章)を参照してください。

詳細

EXP 関数は、定数 *e* (約 2.71828)に対する指定した引数の累乗を返します。この結果は、コンピュータの浮動小数点の最大値によって制限されます。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=exp(1.0);	2.7182818285
x=exp(0);	1

関連項目:

SAS Language Reference: Concepts

- “算術演算子” (*SAS 言語リファレンス: 解説編* 6 章)

FACT 関数

階乗を計算します。

カテゴリ: 数学関数

構文

FACT(*n*)

必須引数

n
数値定数、変数または式です。

詳細

FACT 関数の数学的表現は、次の式で表されます。

$$FACT(n) = n!$$

$n \geq 0$ 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。比較的大きい値の場合、FACT 関数を計算できないことがあります。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=fact(5);	120

関連項目:

関数:

- “COMB 関数” (303 ページ)
- “PERM 関数” (723 ページ)
- “LFACT 関数” (616 ページ)

FAPPEND 関数

現在のレコードを外部ファイルの最後に追加します。

カテゴリ: 外部ファイル

構文

FAPPEND(*file-id*<,*cc*>)

必須引数

file-id
通常、FOPEN 関数によってファイルが開かれたときに割り当てられた識別子を指定する数値変数です。

オプション引数

cc
改行制御文字を指定する文字定数、変数または式です。

空白	レコードが新しい行で開始されます。
0	この新しい行の前に 1 つの空白行が挿入されます。
-	この新しい行の前に 2 つの空白行が挿入されます。
1	行が新しいページで開始されます。
+	前の行を上書きします。
P	行がコンピュータのプロンプトになります。
=	行に改行制御情報が含まれます。
その他すべて	行レコードが新しい行で開始されます。

詳細

FAPPEND は、ファイルデータバッファ(FDB)に現在含まれているレコードを外部ファイルの最後に追加します。FAPPEND は、操作に成功した場合は 0、失敗した場合は≠0 を返します。

サンプル

この例では、ファイル参照名 MYFILE を外部ファイルに割り当ててそのファイルを開こうとします。ファイルが正常に開かれた場合、データをファイルデータバッファ内に移動し、レコードを追加してファイルを閉じます。マクロステートメントでは、文字列を引用符で囲む必要はありません。

```
%let filrf=myfile;
%let rc=%sysfunc(filename(filrf,
physical-filename));
%let fid=%sysfunc(fopen(&filrf,a));
%if &fid > 0 %then
%do;
%let rc=%sysfunc(fput(&fid,
Data for the new record));
%let rc=%sysfunc(fappend(&fid));
%let rc=%sysfunc(fclose(&fid));
%end;
%else
%do;
/* unsuccessful open processing */
%end;
```

関連項目:

関数:

- “DOPEN 関数” (375 ページ)
- “FCLOSE 関数” (394 ページ)
- “FGET 関数” (401 ページ)
- “FOPEN 関数” (476 ページ)
- “FPUT 関数” (484 ページ)
- “FWRITE 関数” (491 ページ)
- “MOPEN 関数” (653 ページ)

FCLOSE 関数

外部ファイル、ディレクトリまたはディレクトリメンバを閉じます。

カテゴリ: 外部ファイル

参照項目: “FCLOSE Function: z/OS” in *SAS Companion for z/OS*

構文

FCLOSE(*file-id*)

必須引数

file-id

通常、FOPEN 関数によってファイルが開かれたときに割り当てられた識別子を指定する数値変数です。

詳細

FCLOSE は、操作に成功した場合は 0、失敗した場合は≠0 を返します。ファイルを DATA ステップ内で開いた場合、そのファイルは DATA ステップの終了時に自動的に閉じられます。

動作環境の情報

一部の動作環境では、DATA ステップの最後に FCLOSE 関数を使用してファイルを閉じる必要があります。詳細については、動作環境に関する SAS のドキュメントを参照してください。

サンプル

この例では、ファイル参照名 MYFILE を外部ファイルに割り当ててそのファイルを開こうとします。ファイルが正常に開かれた場合は変数 FID が正の値で示され、プログラムによって最初のレコードが読み取られた後にファイルが閉じられ、ファイル参照名の割り当てが取り消されます。

```
%let filrf=myfile;
%let rc=%sysfunc(filename(filrf,
physical-filename));
%let fid=%sysfunc(fopen(&filrf));
%if &fid > 0 %then
%do;
%let rc=%sysfunc(fread(&fid));
%let rc=%sysfunc(fclose(&fid));
%end;
%else
%do;
%put %sysfunc(sysmsg());
%end;
%let rc=%sysfunc(filename(filrf));
```

関連項目:

関数:

- “DCLOSE 関数” (354 ページ)
- “DOPEN 関数” (375 ページ)
- “FOPEN 関数” (476 ページ)
- “FREAD 関数” (486 ページ)
- “MOPEN 関数” (653 ページ)

FCOL 関数

ファイルデータバッファ(FDB)内の現在の列の位置を返します。

カテゴリ: 外部ファイル

構文

FCOL(*file-id*)

必須引数

file-id
通常、FOPEN 関数によってファイルが開かれたときに割り当てられた識別子を指定する数値変数です。

詳細

ファイルデータバッファ(FDB)内のデータを操作するには、FCOL と FPOS を組み合わせて使用します。

サンプル

この例では、ファイル参照名 MYFILE を外部ファイルに割り当ててそのファイルを開こうとします。変数 FID の正の値が示すように、ファイルが正常に開かれると、位置 POS に相対する FDB にデータをさらに挿入し、レコードを書き込んだ後、ファイルが閉じられます。

```
%let filrf=myfile;
%let rc=%sysfunc(filename(filrf,
physical-filename));
%let fid=%sysfunc(fopen(&filrf,o));
%if (&fid > 0) %then
%do;
%let record=This is data for the record.;
%let rc=%sysfunc(fput(&fid,&record));
%let pos=%sysfunc(fcol(&fid));
%let rc=%sysfunc(fpos(&fid,%eval(&pos+1)));
%let rc=%sysfunc(fput(&fid,more data));
%let rc=%sysfunc(fwrite(&fid));
%let rc=%sysfunc(fclose(&fid));
%end;
%let rc=%sysfunc(filename(filrf));
```

外部ファイルに書き込まれる新しいレコードは次のとおりです。

```
This is data for the record. more data
```

関連項目:

関数:

- “FCLOSE 関数” (394 ページ)
- “FOPEN 関数” (476 ページ)
- “FPOS 関数” (482 ページ)
- “FPUT 関数” (484 ページ)
- “FWRITE 関数” (491 ページ)
- “MOPEN 関数” (653 ページ)

FDELETE 関数

外部ファイルまたは空のディレクトリを削除します。

カテゴリ: 外部ファイル

参照項目: “FDELETE Function: Windows” in *SAS Companion for Windows*
“FDELETE Function: UNIX” in *SAS Companion for UNIX Environments*
“FDELETE Function: z/OS” in *SAS Companion for z/OS*

構文

FDELETE(*fileref* | *directory*)

必須引数

fileref
外部ファイルに割り当てたファイル参照名を指定する文字定数、変数または式です。ファイル参照名は、FILENAME ステートメントまたは FILENAME 外部ファイルアクセス関数を使用して割り当てることができます。

制限事項: FDELETE で使用するファイル参照名は連結できません。

Windows 固有: 一部の動作環境では、環境変数で割り当てられたファイル参照名を指定できます。システムコマンドを使用してファイル参照名を割り当てることもできます。詳細については、動作環境に関する SAS のドキュメントを参照してください。

directory
削除する空のディレクトリを指定する文字定数、変数または式です。

制限事項: ディレクトリを削除する権限が必要です。

詳細

FDELETE は操作が成功した場合は 0、失敗した場合は≠0 を返します。

サンプル

サンプル 1: 外部ファイルの削除

この例では、変数 FNAME 内の外部ファイルのファイル参照名を生成します。次に、FDELETE を呼び出してファイルを削除してから、再度 FILENAME 関数を呼び出して参照ファイル名の割り当てを取り消します。

```
data _null_;
fname="tempfile";
rc=filename(fname,"physical-filename");
if rc = 0 and fexist(fname) then
rc=fdelete(fname);
rc=filename(fname);
run;
```

サンプル 2: ディレクトリの削除

この例では、FDELETE を使用して、書き込みアクセス権がある空のディレクトリを削除します。ディレクトリが空でない場合、任意の SYSMSG 関数はファイルを削除できないというエラーメッセージを返します。

```
filename testdir 'physical-filename';
data _null_;
rc=fdelete('testdir');
put rc=;
```

```
msg=sysmsg();
put msg=;
run;
```

関連項目:

関数:

- “FEXIST 関数” (400 ページ)
- “FILENAME 関数” (404 ページ)

ステートメント:

- “FILENAME ステートメント” (*SAS ステートメント: リファレンス*)

FETCH 関数

SAS データセットから次の非削除対象のオブザベーションをデータセットデータベクトル(DDV)に読み込みます。

カテゴリ: SAS ファイル I/O 関数

構文

FETCH(*data-set-id* <,'NOSET'>)

必須引数

data-set-id
OPEN 関数によって返されたデータセット識別子を指定する数値変数です。

オプション引数

'NOSET'
SET ルーチンが呼び出された場合でも、SAS データセットの変数値がマクロ変数または DATA ステップ変数に自動的に渡されないようにします。

詳細

FETCH は操作が成功した場合は 0、失敗した場合は≠0、データセットの終わりに達した場合は-1 を返します。FETCH は削除対象としてマークされたオブザベーションをスキップします。

SET ルーチンを以前に呼び出したことがある場合、データセット変数の値が DDV から対応する DATA ステップ変数またはマクロ変数に自動的に渡されます。フェッチした値が DATA ステップ変数またはマクロ変数に自動的にコピーされないようにこの動作を一時的に無効にするには、NOSET オプションを使用します。

サンプル

この例では、SAS データセット MYDATA から次のオブザベーションをフェッチします。データセットの終わりに達した場合や エラーが発生した場合、SYSMSG

は適切なメッセージを取得して SAS ログに書き込みます。マクロステートメントでは、文字列を引用符で囲む必要はありません。

```
%let dsid=%sysfunc(open(mydata,i)); %let rc=%sysfunc(fetch(&dsid)); %if &rc ne 0 %then %put %sysfunc(sysmsg()); %else %do; ...more m
```

関連項目:

関数:

- “FETCHOBS 関数” (399 ページ)
- “GETVARC 関数” (510 ページ)
- “GETVARN 関数” (511 ページ)

CALL ルーチン:

- “CALL SET ルーチン” (242 ページ)

FETCHOBS 関数

SAS データセットから指定したオブザベーションをデータセットデータベクトル(DDV)に読み込みます。

カテゴリ: SAS ファイル I/O 関数

構文

FETCHOBS(*data-set-id*,*obs-number*<,*options*>)

必須引数

data-set-id
OPEN 関数によって返されたデータセット識別子を指定する数値変数です。

obs-number
読み込むオブザベーションの番号を指定する数値定数、変数または式です。FETCHOBS は、ABS オプションを指定しないかぎり、オブザベーション値を相対的なオブザベーション番号として扱います。削除対象としてマークしたオブザベーションはスキップされるため、相対的なオブザベーション番号はディスク上の物理的なオブザベーション番号とは一致しない場合があります。WHERE 句がアクティブな場合、WHERE 条件を満たすオブザベーションのみが数えられます。

デフォルト: FETCHOBS は削除されたオブザベーションをスキップします。

オプション引数

オプション
空白で区切って 1 つ以上のオプションを指定する文字定数、変数または式です。

ABS	*obs-number* の値が絶対値であることを指定します。つまり、削除対象のオブザベーションも数えられます。
NOSET	SET ルーチンが呼び出された場合でも、SAS データセットの変数値が DATA ステップ変数またはマクロ変数に自動的に渡されないようにします。

詳細

FETCHOBS は操作が成功した場合は 0、失敗した場合は≠0、データセットの終わりに達した場合は-1 を返します。非ゼロのリターンコードに関連付けられたエラーメッセージを取得するには、SYSMSG 関数を使用します。SET ルーチンを以前に呼び出したことがある場合、データセット変数の値が DDV から対応する DATA ステップ変数またはマクロ変数に自動的に渡されます。この動作を一時的に無効にするには、NOSET オプションを使用します。

obs-number が 1 未満の場合、エラー条件が返されます。*obs-number* が、SAS データセット内のオブザベーション数より大きい場合、ファイル終了条件が返されます。

サンプル

この例では、SAS データセット MYDATA から 10 番目のオブザベーションをフェッチします。エラーが発生した場合、SYSMSG 関数はエラーメッセージを取得して SAS ログに書き込みます。マクロステートメントでは、文字列を引用符で囲む必要はありません。

```
%let rc = %sysfunc(fetchobs(&mydataid,10));
%if &rc = -1 %then
%put End of data set has been reached.;
%if &rc > 0 %then %put %sysfunc(sysmsg());
```

関連項目:

関数:

- “FETCH 関数” (398 ページ)
- “GETVARC 関数” (510 ページ)
- “GETVARN 関数” (511 ページ)

CALL ルーチン:

- “CALL SET ルーチン” (242 ページ)

FEXIST 関数

ファイル参照名に関連付けられている外部ファイルが存在するかどうかを確認します。

カテゴリ: 外部ファイル

参照項目: “FEXIST Function: Windows” in *SAS Companion for Windows*
“FEXIST Function: UNIX” in *SAS Companion for UNIX Environments*
“FEXIST Function: z/OS” in *SAS Companion for z/OS*

構文

FEXIST(*fileref*)

必須引数

fileref
外部ファイルに割り当てられたファイル参照名を指定する文字定数、変数または式です。

制限事項: *fileref* は以前に割り当てたものを指定する必要があります。

Windows 固有: 一部の動作環境では、環境変数で割り当てられたファイル参照名を指定できます。詳細については、動作環境に関する SAS のドキュメントを参照してください。

詳細

FEXIST は、*fileref* に関連付けられた外部ファイルが存在する場合は 1、ファイルが存在しない場合は 0 を返します。ファイル参照名は、FILENAME ステートメントまたは FILENAME 外部ファイルアクセス関数を使用して割り当てることができます。一部の動作環境では、システムコマンドを使用してファイル参照名を割り当てることもできます。

比較

FILEEXIST は物理名に基づいてファイルが存在するかどうかを確認します。

サンプル

この例では、外部ファイルが存在するかどうかを確認し、結果を SAS ログに書き込みます。

```
 %if %sysfunc(fexist(&fref)) %then
%put The file identified by the fileref
&fref exists.;
%else
%put %sysfunc(sysmsg());
```

関連項目:

関数:

- “EXIST 関数” (389 ページ)
- “FILEEXIST 関数” (403 ページ)
- “FILENAME 関数” (404 ページ)
- “FILEREF 関数” (407 ページ)

ステートメント:

- “FILENAME ステートメント” (*SAS ステートメント: リファレンス*)

FGET 関数

ファイルデータバッファ(FDB)からデータを変数にコピーします。

カテゴリ: 外部ファイル

構文

FGET(*file-id*,*variable*<,*length*>)

必須引数

file-id
通常、FOPEN 関数によってファイルが開かれたときに割り当てられた識別子を指定する数値変数です。

変数
DATA ステップの場合、データを保持する文字変数を指定します。マクロの場合、データを保持するマクロ変数を指定します。*variable* がマクロ変数で、存在しない場合は、作成されます。

オプション引数

長さ
FDB から取得する文字数を指定します。*length* を指定すると、指定した文字数のみ(またはバッファ内の残りの文字数が length より少ない場合は、その残りの文字数)が取得されます。*length* を省略すると、現在の列の位置から次の区切り文字まで、FDB 内のすべての文字が返されます。デフォルトの区切り文字は空白です。区切り文字は取得されません。

参照項目: 区切り文字の詳細については、“FSEP 関数” (489 ページ)を参照してください。

詳細

FGET は操作が成功した場合は 0、FDB の終わりに達した場合または使用可能なトークンがなくなった場合は-1 を返します。

FGET の実行後、列ポインタは FDB の次の読み込み位置へ移動します。

サンプル

この例では、ファイル参照名 MYFILE を外部ファイルに割り当ててそのファイルを開こうとします。ファイルが正常に開かれると、ファイルデータバッファに最初のレコードを読み込み、レコードの最初のトークンを取得し、このトークンを変数 MYSTRING に保存した後、ファイルが閉じられます。マクロステートメントでは、文字列を引用符で囲む必要はありません。

```
%let filrf=myfile;
%let rc=%sysfunc(filename(filrf,
physical-filename));
%let fid=%sysfunc(fopen(&filrf));
%if &fid > 0 %then
%do;
%let rc=%sysfunc(fread(&fid));
%let rc=%sysfunc(fget(&fid,mystring));
%put &mystring;
%let rc=%sysfunc(fclose(&fid));
%end;
%let rc=%sysfunc(filename(filrf));
```

関連項目:

関数:

- “FCLOSE 関数” (394 ページ)
- “FILENAME 関数” (404 ページ)
- “FOPEN 関数” (476 ページ)
- “FPOS 関数” (482 ページ)
- “FREAD 関数” (486 ページ)
- “FSEP 関数” (489 ページ)
- “MOPEN 関数” (653 ページ)

FILEEXIST 関数

物理名で外部ファイルが存在するかどうかを確認します。

カテゴリ: 外部ファイル

参照項目: “FILEEXIST Function: Windows” in *SAS Companion for Windows*
“FILEEXIST Function: UNIX” in *SAS Companion for UNIX Environments*
“FILEEXIST Function: z/OS” in *SAS Companion for z/OS*

構文

FILEEXIST(*file-name*)

必須引数

file-name
動作環境内の外部ファイルの完全修飾物理ファイル名を指定する文字定数、変数または式です。

詳細

FILEEXIST は、外部ファイルが存在する場合は 1、外部ファイルが存在しない場合は 0 を返します。*file-name* の物理名の指定は、動作環境によって異なります。

動作環境のユーティリティで物理ファイル名の一部が認識される場合がありますが、FILEEXIST では必ず完全修飾物理ファイル名を使用する必要があります。

サンプル

この例では、外部ファイルが存在するかどうかを確認します。ファイルが存在する場合、FILEEXIST はそのファイルを開きます。ファイルが存在しない場合、FILEEXIST は SAS ログにメッセージを表示します。マクロステートメントでは、文字列を引用符で囲む必要はありません。

```
%if %sysfunc(fileexist(&myfilerf)) %then
%let fid=%sysfunc(fopen(&myfilerf));
%else
%put The external file &myfilerf does not exist.;
```

関連項目:

関数:

- “EXIST 関数” (389 ページ)
- “FEXIST 関数” (400 ページ)
- “FILENAME 関数” (404 ページ)
- “FILEREF 関数” (407 ページ)
- “FOPEN 関数” (476 ページ)

FILENAME 関数

ファイル参照名を外部ファイル、ディレクトリまたは出力デバイスに割り当てるか、または割り当てを取り消します。

カテゴリ: 外部ファイル

参照項目: “FILENAME Function: Windows” in *SAS Companion for Windows*
“FILENAME Function: UNIX” in *SAS Companion for UNIX Environments*
“FILENAME Function: z/OS” in *SAS Companion for z/OS*

構文

FILENAME(*fileref* <,*file-name*> <,*device-type*> <,'*host-options*'> <,*dir-ref*>)

必須引数

fileref
外部ファイルに割り当てるファイル参照名を指定します。DATA ステップの場合、*fileref* には文字式、ファイル参照名を指定する引用符で囲まれた文字列、または値にファイル参照名を含む DATA ステップ変数を指定できます。マクロの場合(たとえば%SYSFUNC 関数)、*fileref* は外部ファイルに割り当てるファイル参照名を値に含むマクロ変数の名前(アンパサンドは含まない)になります。

要件 *fileref* が DATA ステップ変数の場合、長さは 8 文字以内にする必要があります。

ヒント: ファイル参照名が、値が空白の DATA ステップ文字変数で最大長が 8 文字の場合、または *fileref* で指定したマクロ変数の値が NULL の場合、ファイル参照名が生成され、それぞれ文字変数またはマクロ変数に割り当てられます。

オプション引数

file-name
外部ファイルを指定する文字定数、変数または式です。空白の *file-name* を指定すると、以前に割り当てたファイル参照名の割り当てが取り消されます。

device-type
デバイスの種類、またはファイル参照名が物理ファイルではない入出力デバイスや場所を指定する場合は使用するアクセス方法を指定する文字定数、変数または式です。

DISK
デバイスがディスクドライブであることを指定します。

別名: BASE

ヒント: ファイル参照名をディスク上のファイルに割り当てるとき、DISK を指定する必要はありません。

DUMMY
ファイルへの出力を破棄することを指定します。

ヒント: DUMMY を指定すると、テストに便利な場合があります。

GTERM
出力デバイスの種類が、グラフィックデータを受け取るグラフィックデバイスであることを示します。

PIPE
名前の付いていないパイプを指定します。

注: 一部の動作環境では、パイプはサポートされていません。

PLOTTER
バッファされていないグラフィック出力デバイスを指定します。

PRINTER
プリンタまたはプリンタスプールファイルを指定します。

TAPE
テープドライブを指定します。

TEMP
ファイル名が割り当てられている間だけ存在する一時ファイルを作成します。一時ファイルにアクセスするには論理名を使用する必要があり、この一時ファイルは論理名が存在する間のみ使用できます。

制限事項: 物理パス名は指定しないでください。指定すると、エラーが返されます。

ヒント: TEMP デバイスで操作されるファイルは、同じ属性を使用し、DISK ファイルと同様に動作させることができます。

TERMINAL
ユーザーのパーソナルコンピュータを指定します。

UPRINTER
Universal Printing プリンタ定義名を指定します。

動作環境: FILENAME 関数では、動作環境に固有のデバイスもサポートされます。詳細については、動作環境に関する SAS のドキュメントを参照してください。

'host-options'
ファイル属性や処理属性など、ホストに固有の詳細を指定します。詳細については、動作環境に関する SAS のドキュメントを参照してください。

dir-ref
外部ファイルが保存されているディレクトリまたは分割データセットに割り当てられたファイル参照名を指定します。

詳細

FILENAME は操作が成功した場合は 0、失敗した場合は≠0 を返します。ファイルまたはデバイスに関連付けられている名前は、*fileref*(ファイル参照名)と呼ばれます。外部ファイルおよびディレクトリを操作する他のシステム関数では、ファイルを物理ファイル名ではなくファイル参照名で識別する必要があります。ファイル参照名と物理ファイルの関連付けは、現在の SAS セッションの期間、ま

たは FILENAME を使用して関連付けを変更または廃止するまで有効です。ファイル参照名の割り当ては、FILENAME の *file-name* 引数で NULL 文字列を指定すると取り消すことができます。

動作環境の情報
この説明中の*ディレクトリ*という用語は、動作環境で管理されるファイルをグループ化した集合を指します。これらのグルーピングは、異なる動作環境ではディレクトリ、サブディレクトリ、フォルダ、MACLIB、分割データセットなど、さまざまな名前で呼ばれます。詳細については、動作環境に関する SAS のドキュメントを参照してください。

一部の動作環境では、システムコマンドを使用してファイル参照名を割り当てることもできます。動作環境によっては、SAS 以外で割り当てられたファイル参照名を FILENAME で変更したり、割り当てを取り消したりできないことがあります。

サンプル

サンプル 1: 外部ファイルにファイル参照名を割り当てる

この例では、ファイル参照名 MYFILE を外部ファイルに割り当てます。次に、ファイル参照名の割り当てを取り消します。マクロステートメントでは、文字列を引用符で囲む必要はありません。

```
%let filrf=myfile;
%let rc=%sysfunc(filename(filrf, physical-filename));
%if &rc ne 0 %then
%put %sysfunc(sysmsg());
%let rc=%sysfunc(filename(filrf));
```

サンプル 2: システム生成のファイル参照名を割り当てる

この例では、システムで生成されたファイル参照名を外部ファイルに割り当てます。ファイル参照名は、変数 FNAME に保存されます。マクロステートメントでは、文字列を引用符で囲む必要はありません。

```
%let rc=%sysfunc(filename(fname, physical-filename));
%if &rc %then
%put %sysfunc(sysmsg());
%else
%do;
more macro statements
%end;
```

サンプル 3: パイプファイルにファイル参照名を割り当てる

この例では、ディレクトリ/u/myid 内のファイルが表示される UNIX コマンド LS の出力を使用して、ファイル参照名 MYPIPE をパイプファイルに割り当てます。マクロステートメントでは、文字列を引用符で囲む必要はありません。

```
%let filrf=mypipe;
%let rc=%sysfunc(filename(filrf, %str(ls /u/myid), pipe));
```

関連項目:

関数:

- “FEXIST 関数” (400 ページ)

- “FILEEXIST 関数” (403 ページ)
- “FILEREF 関数” (407 ページ)
- “SYSMSG 関数” (880 ページ)

FILEREF 関数

現在の SAS セッションにファイル参照名が割り当てられているかどうかを確認します。

カテゴリ: 外部ファイル

参照項目: “FILEREF Function: Windows” in *SAS Companion for Windows*
“FILEREF Function: UNIX” in *SAS Companion for UNIX Environments*
“FILEREF Function: z/OS” in *SAS Companion for z/OS*

構文

FILEREF(*fileref*)

必須引数

fileref
検証するファイル参照名を指定する文字定数、変数または式です。
範囲: 1 ~ 8 文字

詳細

負のリターンコードは、ファイル参照名は存在するが、そのファイル参照名に関連付けられた物理ファイルが存在しないことを示します。正の値は、ファイル参照名が割り当てられていないことを示します。ゼロの値は、ファイル参照名と外部ファイルの両方が存在することを示します。

ファイル参照名は、FILENAME ステートメントまたは FILENAME 関数を使用して外部ファイルに割り当てることができます。

Windows 固有
一部の動作環境では、システムコマンドを使用してファイル参照名を割り当てることもできます。詳細については、動作環境に関する SAS のドキュメントを参照してください。

サンプル

サンプル 1: ファイル参照名が割り当てられていることの確認

この例では、ファイル参照名 MYFILE が現在外部ファイルに割り当てられているかどうかをテストします。ファイル参照名が現在割り当てられていない場合は、システムエラーメッセージが発行されます。

```
%if %sysfunc(fileref(myfile))>0 %then
%put MYFILE is not assigned;
```

サンプル 2: ファイル参照名とファイルの両方が存在することの確認

この例では、ファイル参照名とファイルの両方が存在するかどうかを判断するために、ゼロ値をテストします。

```
%if %sysfunc(fileref(myfile)) ne 0 %then
%put %sysfunc(sysmsg());
```

関連項目:

関数:

- “FEXIST 関数” (400 ページ)
- “FILEEXIST 関数” (403 ページ)
- “FILENAME 関数” (404 ページ)
- “SYSMSG 関数” (880 ページ)

ステートメント:

- “FILENAME ステートメント” (*SAS ステートメント: リファレンス*)

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: 財務関数

構文

FINANCE(*string-identifier*, *parm1*, *parm2*,…)

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” (412 ページ)	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” (412 ページ)	満期に利息を支払う証券の未払い利息を計算します。
“AMORDEGRC” (413 ページ)	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” (413 ページ)	各会計期間の減価償却を計算します。
“COUPDAYBS” (414 ページ)	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” (414 ページ)	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” (414 ページ)	決済日から次のクーポン日までの日数を計算します。
“COUPNCD” (415 ページ)	決済日後の次のクーポン日を計算します。
“COUPNUM” (415 ページ)	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” (416 ページ)	決済日前の前のクーポン日を計算します。
“CUMIPMT” (416 ページ)	2 つの期間の間に支払われる累積利息を計算します。
“CUMPRINC” (416 ページ)	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” (417 ページ)	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” (417 ページ)	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” (418 ページ)	証券の割引率を計算します。
“DOLLARDE” (418 ページ)	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” (418 ページ)	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” (419 ページ)	定期的に利息を支払う証券の年度期間を計算します。
“EFFECT” (419 ページ)	有効な年利を計算します。
“FV” (419 ページ)	投資の将来価値を計算します。
“FVSCHEDULE” (420 ページ)	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” (420 ページ)	完全投資済み証券の利率を計算します。
“IPMT” (420 ページ)	一定期間の投資の利払いを計算します。
“IRR” (421 ページ)	一連のキャッシュフローの内部利益率を計算します。
“ISPMT” (421 ページ)	投資の指定期間中に支払われる利息を計算します。
“MDURATION” (422 ページ)	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” (422 ページ)	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” (422 ページ)	年間の名目金利を計算します。
“NPER” (423 ページ)	投資の期間数を計算します。
“NPV” (423 ページ)	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” (423 ページ)	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” (424 ページ)	端数の開始期間で証券の利回りを計算します。
“ODDLPRICE” (425 ページ)	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” (425 ページ)	端数の最終期間で証券の利回りを計算します。
“PMT” (426 ページ)	年金の定期的支払いを計算します。
“PPMT” (426 ページ)	一定期間の投資の元本に対する支払いを計算します。
“PRICE” (427 ページ)	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” (427 ページ)	割引証券の額面$100 当たりの価格を計算します。
“PRICEMAT” (428 ページ)	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” (428 ページ)	投資の現在価値を計算します。
“RATE” (429 ページ)	年金期間当たりの利率を計算します。
“RECEIVED” (429 ページ)	完全投資済み証券の満期受け取り額を計算します。
“SLN” (429 ページ)	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” (430 ページ)	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” (430 ページ)	米国財務省短期証券の債券換算利回りを計算します。

string-identifier	説明
“TBILLPRICE” (430 ページ)	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。
“TBILLYIELD” (431 ページ)	米国財務省短期証券の利回りを計算します。
“VDB” (431 ページ)	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” (432 ページ)	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” (432 ページ)	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” (432 ページ)	定期的に利息を支払う証券の利回りを計算します。
“YIELDDISC” (433 ページ)	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” (433 ページ)	満期に利息を支払う証券の年間利回りを計算します。

parm
各 *string-identifier* に関連付けられたパラメータを指定します。次のパラメータを使用できます。

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。

詳細

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。

FINANCE('ACCRINT', *issue*, *first-interest*, *settlement*, *rate*, *par*, *frequency*, <*basis*>);

引数

issue
証券の発行日を指定します。

first-interest
証券の最初の利息日を指定します。

settlement
決済日を指定します。

rate
利率を指定します。

par
証券の額面価格を指定します。*par* を省略すると、**$1000** という値が使用されます。

度数
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。

basis
日数計算基準値を指定します(省略可能)。

説明:
“サンプル 1: 未払い利息を計算する: ACCRINT” (434 ページ)

ACCRINTM

満期に利息を支払う証券の未払い利息を計算します。

FINANCE('ACCRINTM', *issue*, *settlement*, *rate*, *par*, <*basis*>);

引数

issue
証券の発行日を指定します。

settlement
決済日を指定します。

rate
利率を指定します。

par
証券の額面価格を指定します。*par* を省略すると、**$1000** という値が使用されます。

basis
日数計算基準値を指定します(省略可能)。

説明:
"サンプル 2: 未払い利息を計算する: ACCRINTM" (434 ページ)

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。

FINANCE('AMORDEGRC', *cost*, *date-purchased*, *first-period*, *salvage*, *period*, *rate*, <*basis*>);

引数

コスト
資産の初期コストを指定します。

date-purchased
資産の購入日を指定します。

first-period
第 1 期間の最終日を指定します。

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。

period
減価償却期間を指定します。

rate
減価償却率を指定します。

basis
日数計算基準値を指定します(省略可能)。

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。

説明:
"サンプル 3: 減価償却を計算する: AMORDEGRC" (434 ページ)

AMORLINC

各会計期間の減価償却を計算します。

FINANCE('AMORLINC', *cost*, *date-purchased*, *first-period*, *salvage*, *period*, *rate*, <*basis*>);

引数

コスト
資産の初期コストを指定します。

date-purchased
資産の購入日を指定します。

first-period
第 1 期間の最終日を指定します。

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。

period
減価償却期間を指定します。

rate
減価償却率を指定します。

basis
日数計算基準値を指定します(省略可能)。

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。

説明:
“サンプル 4: 計算の説明: AMORLINC” (435 ページ)

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。

FINANCE('COUPDAYBS', *settlement*, *maturity*, *frequency*, <*basis*>);

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。

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。

度数
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。

basis
使用する日数計算基準の種類を指定します。

説明:
“サンプル 5: 計算の説明: COUPDAYBS” (435 ページ)

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、2011 年 5 月 23 日の場合は DATE(2011,5,23)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

COUPDAYS
決済日を含むクーポン期間の日数を計算します。

FINANCE('COUPDAYS', *settlement*, *maturity*, *frequency*, <*basis*>);

引数

settlement
決済日を指定します。

maturity
満期日を指定します。

度数
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。

basis
日数計算基準値を指定します(省略可能)。

説明:
“サンプル 6: 計算の説明: COUPDAYS” (435 ページ)

COUPDAYSNC
決済日から次のクーポン日までの日数を計算します。

FINANCE('COUPDAYSNC', *settlement*, *maturity*, *frequency*, <*basis*>);

引数

settlement
決済日を指定します。

maturity
満期日を指定します。

度数
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。

basis
日数計算基準値を指定します(省略可能)。

説明:
"サンプル 7: 計算の説明: COUPDAYSNC" (436 ページ)

COUPNCD

決済日後の次のクーポン日を計算します。

FINANCE('COUPNCD', *settlement*, *maturity*, *frequency*, <*basis*>);

引数

settlement
決済日を指定します。

maturity
満期日を指定します。

度数
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。

basis
日数計算基準値を指定します(省略可能)。

説明:
"サンプル 8: 計算の説明: COUPNCD" (436 ページ)

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。

FINANCE('COUPNUM', *settlement*, *maturity*, *frequency*, <*basis*>);

引数

settlement
決済日を指定します。

maturity
満期日を指定します。

度数
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。

basis
日数計算基準値を指定します(省略可能)。

説明:
“サンプル 9: 計算の説明: COUPNUM” (436 ページ)

COUPPCD
決済日前の前のクーポン日を計算します。

FINANCE('COUPPCD', *settlement*, *maturity*, *frequency*, <*basis*>);

引数

settlement
決済日を指定します。

maturity
満期日を指定します。

度数
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。

basis
日数計算基準値を指定します(省略可能)。

説明:
“サンプル 10: 計算の説明: COUPPCD” (436 ページ)

CUMIPMT
2 つの期間の間に支払われる累積利息を計算します。

FINANCE('CUMIPMT', *rate*, *nper*, *pv*, *start-period*, *end-period*, <*type*>);

引数

rate
利率を指定します。

nper
支払い期間の総数を指定します。

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。

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。

end-period
計算の最終期間を指定します。

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。

説明:
“サンプル 11: 計算の説明: CUMIPMT” (437 ページ)

CUMPRINC
2 つの期間の間に口ーンで支払われる累積元本を計算します。

FINANCE('CUMPRINC', *rate*, *nper*, *pv*, *start-period*, *end-period*, <*type*>);

引数

rate
利率を指定します。

nper
支払い期間の総数を指定します。

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。

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。

end-period
計算の最終期間を指定します。

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。

説明:
“サンプル 12: 計算の説明: CUMPRINC” (437 ページ)

DB

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。

FINANCE('DB', *cost*, *salvage*, *life*, *period*, <*month*>);

引数

コスト
資産の初期コストを指定します。

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。

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。

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。

月
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。

説明:
“サンプル 13: 計算の説明: DB” (437 ページ)

DDB

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。

FINANCE('DDB', *cost*, *salvage*, *life*, *period*, <*factor*>);

引数

コスト
資産の初期コストを指定します。

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。

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。

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。

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)。

説明:
"サンプル 14: 計算の説明: DDB" (437 ページ)

DISC
証券の割引率を計算します。

FINANCE('DISC', *settlement*, *maturity*, *pr*, *redemption*, <*basis*>);

引数

settlement
決済日を指定します。

maturity
満期日を指定します。

pr
額面$100 当たりの証券の価格を指定します。

redemption
満期受け取り額を指定します。

basis
日数計算基準値を指定します(省略可能)。

説明:
"サンプル 15: 計算の説明: DISC" (438 ページ)

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。

FINANCE('DOLLARDE', *fractionaldollar*, *fraction*);

引数

fractionaldollar
小数で表す数値を指定します。

fraction
分母に使用する整数を指定します。

説明:
"サンプル 16: 計算の説明: DOLLARDE" (438 ページ)

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。

FINANCE('DOLLARFR', *decimaldollar*, *fraction*);

引数

decimaldollar
小数を指定します。

fraction
分母に使用する整数を指定します。

説明:
“サンプル 17: 計算の説明: DOLLARFR” (438 ページ)

DURATION
定期的に利息を支払う証券の年度期間を計算します。

FINANCE('DURATION', *settlement*, *maturity*, *coupon*, *yld*, *frequency*, <*basis*>);

引数

settlement
決済日を指定します。

maturity
満期日を指定します。

coupon
証券の年間のクーポン率を指定します。

yld
証券の年間利回りを指定します。

度数
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。

basis
日数計算基準値を指定します(省略可能)。

説明:
“サンプル 18: 計算の説明: DURATION” (438 ページ)

EFFECT
有効な年利を計算します。

FINANCE('EFFECT', *nominalrate*, *npery*);

引数

nominalrate
名目金利を指定します。

npery
年間の複利計算期間数を指定します。

説明:
“サンプル 19: 計算の説明: EFFECT” (439 ページ)

FV
投資の将来価値を計算します。

FINANCE('FV', *rate*, *nper*, <*pmt*> , <*pv*>, <*type*>);

引数

rate
利率を指定します。

nper
支払い期間の総数を指定します。

pmt
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、*pmt* 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。*pmt* を省略する場合、*pv* 引数を含める必要があります。

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、*pmt* 引数を含める必要があります。

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。

説明:
"サンプル 20: 計算の説明: FV" (439 ページ)

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。

FINANCE('FVSCHEDULE', *principal*, *schedule1*, *schedule2*...);

引数

元本
現在価値を指定します。

schedule
適用する一連の利率を指定します。

説明:
"サンプル 21: 計算の説明: FVSCHEDULE" (439 ページ)

INTRATE
完全投資済み証券の利率を計算します。

FINANCE('INTRATE', *settlement*, *maturity*, *investment*, *redemption*, <*basis*>);

引数

settlement
決済日を指定します。

maturity
満期日を指定します。

investment
証券に投資される金額を指定します。

redemption
満期受け取り額を指定します。

basis
日数計算基準値を指定します(省略可能)。

説明:
"サンプル 22: 計算の説明: INTRATE" (439 ページ)

IPMT
指定期間の投資の利払いを計算します。

FINANCE('IPMT', *rate*, *period*, *nper*, *pv*, <*fv*> , <*type*>);

引数

rate
利率を指定します。

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。

nper
支払い期間の総数を指定します。

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。

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)。

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。

説明:
"サンプル 23: 計算の説明: IPMT" (440 ページ)

IRR

一連のキャッシュフローの内部利益率を計算します。

FINANCE('IRR', *value1*, *value2*, …, *value_n*);

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。

説明:
"サンプル 24: 計算の説明: IRR" (440 ページ)

ISPMT

投資の指定期間中に支払われる利息を計算します。

FINANCE ('ISPMT', *interest-rate*, *period*, *number-payments*, *PV*);

引数

interest-rate
投資の利率です。

period
利率を計算する期間です。*Period* は 1 ~ *number-payments* の間の値にする必要があります。

number-payments
年金の支払い回数です。

PV
ローン金額または支払いの現在価値です。

説明:
"サンプル 25: 計算の説明: ISPMT" (440 ページ)

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。

FINANCE('MDURATION', *settlement*, *maturity*, *coupon*, *yld*, *frequency*, <*basis*>);

引数

settlement
決済日を指定します。

maturity
満期日を指定します。

coupon
証券の年間のクーポン率を指定します。

yld
証券の年間利回りを指定します。

度数
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。

basis
日数計算基準値を指定します(省略可能)。

説明:
“サンプル 26: 計算の説明: MDURATION” (440 ページ)

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。

FINANCE('MIRR', *value1*, …, *value_n*, *financerate*, *reinvestrate*);

引数

値
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。*Values* 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。

financerate
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。

reinvestrate
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。

説明:
“サンプル 27: 計算の説明: MIRR” (441 ページ)

NOMINAL

年間の名目金利を計算します。

FINANCE('NOMINAL', *effectrate*, *npery*);

引数

effectrate
有効な利率を指定します。

npery
年間の複利計算期間数を指定します。

説明:
“サンプル 28: 計算の説明: NOMINAL” (441 ページ)

NPER
投資の期間数を計算します。

FINANCE('NPER', *rate*, *pmt* , *pv*, <*fv*> , <*type*>);

引数

rate
利率を指定します。

pmt
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、*pmt* 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。*pmt* を省略する場合、*pv* 引数を含める必要があります。

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、*pmt* 引数を含める必要があります。

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)。

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。

説明:
“サンプル 29: 計算の説明: NPER” (441 ページ)

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。

FINANCE('NPV', *rate*, *value−1* <,…value-n>);

引数

rate
利率を指定します。

value
一連のキャッシュフローを表します。

説明:
“サンプル 30: 計算の説明: NPV” (442 ページ)

ODDFPRICE
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('ODDFPRICE', *settlement*, *maturity*, *issue*, *first-coupon*, *rate*, *yld*, *redemption*, *frequency*, <*basis*>);

引数

settlement
決済日を指定します。

maturity
満期日を指定します。

issue
証券の発行日を指定します。

first-coupon
証券の開始クーポン日を指定します。

rate
利率を指定します。

yld
証券の年間利回りを指定します。

redemption
満期受け取り額を指定します。

度数
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。

basis
日数計算基準値を指定します(省略可能)。

説明:
"サンプル 31: 計算の説明: ODDFPRICE" (442 ページ)

ODDFYIELD

端数の開始期間で証券の利回りを計算します。

FINANCE('ODDFYIELD', *settlement*, *maturity*, *issue*, *first-coupon*, *rate*, *pr*, *redemption*, *frequency*, <*basis*>);

引数

settlement
決済日を指定します。

maturity
満期日を指定します。

issue
証券の発行日を指定します。

first-coupon
証券の開始クーポン日を指定します。

rate
利率を指定します。

pr
額面$100 当たりの証券の価格を指定します。

redemption
満期受け取り額を指定します。

度数
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。

basis
日数計算基準値を指定します(省略可能)。

説明:
“サンプル 32: 計算の説明: ODDFYIELD” (442 ページ)

ODDLPRICE
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('ODDLPRICE', *settlement*, *maturity*, *last_interest*, *rate*, *yld*, *redemption*, *frequency*, <*basis*>);

引数

settlement
決済日を指定します。

maturity
満期日を指定します。

last_interest
証券の最終クーポン日を指定します。

rate
利率を指定します。

yld
証券の年間利回りを指定します。

redemption
満期受け取り額を指定します。

度数
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。

basis
日数計算基準値を指定します(省略可能)。

説明:
“サンプル 33: 計算の説明: ODDLPRICE” (443 ページ)

ODDLYIELD
端数の最終期間で証券の利回りを計算します。

FINANCE('ODDLYIELD', *settlement*, *maturity*, *last_interest*, *rate*, *pr*, *redemption*, *frequency*, <*basis*>);

引数

settlement
決済日を指定します。

maturity
満期日を指定します。

last_interest
証券の最終クーポン日を指定します。

rate
利率を指定します。

pr
額面$100 当たりの証券の価格を指定します。

redemption
満期受け取り額を指定します。

度数
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。

basis
日数計算基準値を指定します(省略可能)。

説明:
“サンプル 34: 計算の説明: ODDLYIELD” (443 ページ)

PMT
年金の定期的支払いを計算します。

FINANCE('PMT', *rate*, *nper*, *pv*, <*fv*> , <*type*>);

引数

rate
利率を指定します。

nper
支払期間の数を指定します。

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。

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)。

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。

説明:
“サンプル 35: 計算の説明: PMT” (443 ページ)

PPMT
指定期間の投資の元本に対する支払いを計算します。

FINANCE('PPMT', *rate*, *per*, *nper*, *pv*, <*fv*> , <*type*>);

引数

rate
利率を指定します。

per
期間を指定します。

範囲: 1–nper

nper
支払期間の数を指定します。

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。

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)。

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。

説明:
"サンプル 36: 計算の説明: PPMT" (443 ページ)

PRICE

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('PRICE', *settlement*, *maturity*, *rate*, *yld*, *redemption*, *frequency*, <*basis*>);

引数

settlement
決済日を指定します。

maturity
満期日を指定します。

rate
利率を指定します。

yld
証券の年間利回りを指定します。

redemption
満期受け取り額を指定します。

度数
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。

basis
日数計算基準値を指定します(省略可能)。

説明:
"サンプル 37: 計算の説明: PRICE" (444 ページ)

PRICEDISC

額面$100 当たりの割引証券の価格を計算します。

FINANCE('PRICEDISC', *settlement*, *maturity*, *discount*, *redemption*, <*basis*>);

引数

settlement
決済日を指定します。

maturity
満期日を指定します。

discount
証券の割引率を指定します。

redemption
満期受け取り額を指定します。

basis
日数計算基準値を指定します(省略可能)。

説明:
“サンプル 38: 計算の説明: PRICEDISC” (444 ページ)

PRICEMAT

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('PRICEMAT', *settlement*, *maturity*, *issue*, *rate*, *yld*, <*basis*>);

引数

settlement
決済日を指定します。

maturity
満期日を指定します。

issue
証券の発行日を指定します。

rate
利率を指定します。

yld
証券の年間利回りを指定します。

basis
日数計算基準値を指定します(省略可能)。

説明:
“サンプル 39: 計算の説明: PRICEMAT” (444 ページ)

PV

投資の現在価値を計算します。

FINANCE('PV', *rate*, *nper*, *pmt* , <*fv*> , <*type*>);

引数

rate
利率を指定します。

nper
支払い期間の総数を指定します。

pmt
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、*pmt* 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。

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)。

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。

説明:
“サンプル 40: 計算の説明: PV” (445 ページ)

RATE
年金期間当たりの利率を計算します。

FINANCE('RATE', *nper*, *pmt* , *pv*, <*fv*> , <*type*>);

引数

nper
支払い期間の総数を指定します。

pmt
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、*pmt* 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。*pmt* を省略する場合、*pv* 引数を含める必要があります。

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。

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)。

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。

説明:
“サンプル 41: 計算の説明: RATE” (445 ページ)

RECEIVED
完全投資済み証券の満期受け取り額を計算します。

FINANCE('RECEIVED', *settlement*, *maturity*, *investment*, *discount*, <*basis*>);

引数

settlement
決済日を指定します。

maturity
満期日を指定します。

investment
証券に投資される金額を指定します。

discount
証券の割引率を指定します。

basis
日数計算基準値を指定します(省略可能)。

説明:
“サンプル 42: 計算の説明: RECEIVED” (445 ページ)

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。

FINANCE('SLN', *cost*, *salvage*, *life*);

引数

コスト
資産の初期コストを指定します。

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。

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。

説明:
"サンプル 43: 計算の説明: SLN" (445 ページ)

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。

FINANCE('SYD', *cost*, *salvage*, *life*, *period*);

引数

コスト
資産の初期コストを指定します。

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。

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。

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。

説明:
"サンプル 44: 計算の説明: SYD" (446 ページ)

TBILLEQ

米国財務省短期証券の債券換算利回りを計算します。

FINANCE('TBILLEQ', *settlement*, *maturity*, *discount*);

引数

settlement
決済日を指定します。

maturity
満期日を指定します。

discount
証券の割引率を指定します。

説明:
"サンプル 45: 計算の説明: TBILLEQ" (446 ページ)

TBILLPRICE

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('TBILLPRICE', *settlement*, *maturity*, *discount*);

引数

settlement
決済日を指定します。

maturity
満期日を指定します。

discount
証券の割引率を指定します。

説明:
“サンプル 46: 計算の説明: TBILLPRICE” (446 ページ)

TBILLYIELD
米国財務省短期証券の利回りを計算します。

FINANCE('TBILLYIELD', *settlement*, *maturity*, *pr*);

引数

settlement
決済日を指定します。

maturity
満期日を指定します。

pr
額面$100 当たりの証券の価格を指定します。

説明:
“サンプル 47: 計算の説明: TBILLYIELD” (446 ページ)

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。

FINANCE('VDB', *cost*, *salvage*, *life*, *start-period*, *end-period*, <*factor*> , <*noswitch*>);

引数

コスト
資産の初期コストを指定します。

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。

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。

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。

end-period
計算の最終期間を指定します。

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)。

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。

説明:
“サンプル 48: 計算の説明: VDB” (446 ページ)

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。

FINANCE('XIRR', *values*, *dates*, <*guess*>);

引数

値
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。

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。

guess
XIRR の結果に近い推測する数値を指定します(省略可能)。

説明:
"サンプル 49: 計算の説明: XIRR" (447 ページ)

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。

FINANCE('XNPV', *rate*, *values*, *dates*);

引数

rate
利率を指定します。

値
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。

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。

説明:
"サンプル 50: 計算の説明: XNPV" (447 ページ)

YIELD

定期的に利息を支払う証券の利回りを計算します。

FINANCE('YIELD', *settlement*, *maturity*, *rate*, *pr*, *redemption*, *frequency*, <*basis*>);

引数

settlement
決済日を指定します。

maturity
満期日を指定します。

rate
利率を指定します。

pr
額面$100 当たりの証券の価格を指定します。

redemption
満期受け取り額を指定します。

度数
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。

basis
日数計算基準値を指定します(省略可能)。

説明:
"サンプル 51: 計算の説明: YIELD" (447 ページ)

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。

FINANCE('YIELDDISC', *settlement*, *maturity*, *rate*, *pr*, *redemption*, <*basis*>);

引数

settlement
決済日を指定します。

maturity
満期日を指定します。

rate
利率を指定します。

pr
額面$100 当たりの証券の価格を指定します。

redemption
満期受け取り額を指定します。

basis
日数計算基準値を指定します(省略可能)。

説明:
"サンプル 52: 計算の説明: YIELDDISC" (448 ページ)

YIELDMAT
満期に利息を支払う証券の年間利回りを計算します。

FINANCE('YIELDMAT', *settlement*, *maturity*, *issue*, *rate*, *pr*, <*basis*>);

引数

settlement
決済日を指定します。

maturity
満期日を指定します。

issue
証券の発行日を指定します。

rate
利率を指定します。

pr
額面$100 当たりの証券の価格を指定します。

basis
日数計算基準値を指定します(省略可能)。

説明:
"サンプル 53: 計算の説明: YIELDMAT" (448 ページ)

サンプル

サンプル 1: 未払い利息を計算する: ACCRINT

次の例では、定期的に利息を支払う証券の未払い利息を計算します。

```
data _null_;
issue = mdy(2,27,1996);
firstinterest = mdy(8,31,1998);
settlement = mdy(5,1,1998);
rate = 0.1;
par = 1000;
frequency = 2;
basis = 1;
r = finance('accrint', issue, firstinterest,
settlement, rate, par, frequency, basis);
put r=;
run;
```

返される *r* の値は 217.39728 です。

サンプル 2: 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。

```
data _null_;
issue = mdy(2,28,1998);
maturity = mdy(8,31,1998);
rate = 0.1;
par = 1000;
basis = 0;
r = finance('accrintm', issue, maturity, rate, par, basis);
put r=;
run;
```

返される *r* の値は 50.555555556 です。

サンプル 3: 減価償却を計算する: AMORDEGRC

次の例では、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。

```
data _null_;
cost = 2400;
```

```
datepurchased = mdy(8,19,2008);
firstperiod = mdy(12,31,2008);
salvage = 300;
period = 1;
rate = 0.15;
basis = 1;
r = finance('amordegrc', cost, datepurchased,
firstperiod, salvage, period, rate, basis);
put r=;
run;
```

返される *r* の値は 776 です。

サンプル 4: 計算の説明: AMORLINC

次の例では、各会計期間の減価償却を計算します。

```
data _null_;
cost = 2400;
dp = mdy(9,30,1998);
fp = mdy(12,31,1998);
salvage = 245;
period = 0;
rate = 0.115;
basis = 0;
r = finance('amorlinc', cost, dp, fp, salvage,
period, rate, basis);
put r = ;
run;
```

返される *r* の値は 69 です。

サンプル 5: 計算の説明: COUPDAYBS

次の例では、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。

```
data _null_;
settlement = mdy(12,30,1994);
maturity = mdy(11,29,1997);
frequency = 4;
basis = 2;
r = finance('coupdaybs', settlement, maturity, frequency, basis);
put r = ;
run;
```

返される *r* の値は 31 です。

サンプル 6: 計算の説明: COUPDAYS

次の例では、決済日を含むクーポン期間の日数を計算します。

```
data _null_;
settlement = mdy(1,25,2007);
maturity = mdy(11,15,2008);
frequency = 2;
basis = 1;
r = finance('coupdays', settlement, maturity, frequency, basis);
put r = ;
run;
```

返される *r* の値は 181 です。

サンプル 7: 計算の説明: COUPDAYSNC

次の例では、決済日から次のクーポン日までの日数を計算します。

```
data _null_;
settlement = mdy(1,25,2007);
maturity = mdy(11,15,2008);
frequency = 2;
basis = 1;
r = finance('coupdaysnc', settlement, maturity, frequency, basis);
put r = ;
run;
```

返される *r* の値は 110 です。

サンプル 8: 計算の説明: COUPNCD

次の例では、決済日後の次のクーポン日を計算します。

```
data _null_;
settlement = mdy(1,25,2007);
maturity = mdy(11,15,2008);
frequency = 2;
basis = 1;
r = finance('coupncd', settlement, maturity, frequency, basis);
put r = date7.;
run;
```

返される *r* の値は 15MAY07 です。

注: r は数値の SAS 値で、DATE7 形式を使用して出力できます。

サンプル 9: 計算の説明: COUPNUM

次の例では、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。

```
data _null_;
settlement = mdy(1,25,2007);
maturity = mdy(11,15,2008);
frequency = 2;
basis = 1;
r = finance('coupnum', settlement, maturity, frequency, basis);
put r = ;
run;
```

返される *r* の値は 4 です。

サンプル 10: 計算の説明: COUPPCD

次の例では、決済日前の前のクーポン日を計算します。

```
data _null_;
settlement = mdy(1,25,2007);
maturity = mdy(11,15,2008);
frequency = 2;
basis = 1;
r = finance('couppcd', settlement, maturity, frequency, basis);
put settlement;
put maturity;
put r date7.;
run;
```

返される *r* の値は 11/15/2006 です。

サンプル 11: 計算の説明: CUMIPMT

次の例では、2 つの期間の間に支払われる累積利息を計算します。

```
data _null_;
rate = 0.09;
nper = 30;
pv = 125000;
startperiod = 13;
endperiod = 24;
type = 0;
r = finance('cumipmt', rate, nper, pv,
startperiod, endperiod, type);
put r = ;
run;
```

返される *r* の値は-94054.82033 です。

サンプル 12: 計算の説明: CUMPRINC

次の例では、2 つの期間の間に口ーンに支払われる累積元本を計算します。

```
data _null_;
rate = 0.09;
nper = 30;
pv = 125000;
startperiod = 13;
endperiod = 24;
type = 0;
r = finance('cumprinc', rate, nper, pv,
startperiod, endperiod, type);
put r = ;
run;
```

返される *r* の値は-94054.82033 です。

サンプル 13: 計算の説明: DB

次の例では、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。

```
data _null_;
cost = 1000000;
salvage = 100000;
life = 6;
period = 2;
month = 7;
r = finance('db', cost, salvage, life, period, month);
put r = ;
run;
```

返される *r* の値は 259639.41667 です。

サンプル 14: 計算の説明: DDB

次の例では、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。

```
data _null_;
cost = 2400;
```

```
salvage = 300;
life = 10*365;
period = 1;
factor = .;
r = finance('ddb', cost, salvage, life, period, factor);
put r = ;
run;
```

返される *r* の値は 1.3150684932 です。

サンプル 15: 計算の説明: DISC

次の例では、証券の割引率を計算します。

```
data _null_;
settlement = mdy(1,25,2007);
maturity = mdy(6,15,2007);
pr = 97.975;
redemption = 100;
basis = 1;
r = finance('disc', settlement, maturity, pr, redemption, basis);
put r = ;
run;
```

返される *r* の値は 0.052420213 です。

サンプル 16: 計算の説明: DOLLARDE

次の例では、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。

```
data _null_;
fractionaldollar = 1.125;
fraction = 16;
r = finance('dollarde', fractionaldollar, fraction);
put r = ;
run;
```

返される *r* の値は 1.78125 です。

サンプル 17: 計算の説明: DOLLARFR

次の例では、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。

```
data _null_;
decimaldollar = 1.125;
fraction = 16;
r = finance('dollarfr', decimaldollar, fraction);
put r = ;
run;
```

返される *r* の値は 1.02 です。分数形式では、*r* の値は $1\frac{2}{16}$ のように読みます。

サンプル 18: 計算の説明: DURATION

次の例では、定期的に利息を支払う証券の年度期間を計算します。

```
data _null_;
settlement = mdy(1,1,2008);
maturity = mdy(1,1,2016);
couponrate = 0.08;
yield = 0.09;
```

```
frequency = 2;
basis = 1;
r = finance('duration', settlement,
maturity, couponrate, yield, frequency, basis);
put r = ;
run;
```

返される *r* の値は 5.993775 です。

サンプル 19: 計算の説明: EFFECT

次の例では、有効な年利を計算します。

```
data _null_;
nominalrate = 0.0525;
npery = 4;
r = finance('effect', nominalrate, npery);
put r = ;
run;
```

返される *r* の値は 0.053543 です。

サンプル 20: 計算の説明: FV

次の例では、投資の将来価値を計算します。

```
data _null_;
rate = 0.06/12;
nper = 10;
pmt = -200;
pv = -500;
type = 1;
r = finance('fv', rate, nper, pmt, pv, type);
put r = ;
run;
```

返される *r* の値は 2581.4033741 です。

サンプル 21: 計算の説明: FVSCHEDULE

次の例では、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。

```
data _null_;
principal = 1;
r1 = 0.09;
r2 = 0.11;
r3 = 0.1;
r = finance('fvschedule', principal, r1, r2, r3);
put r = ;
run;
```

返される *r* の値は 1.33089 です。

サンプル 22: 計算の説明: INTRATE

次の例では、完全投資済み証券の利率を計算します。

```
data _null_;
settlement = mdy(2,15,2008);
maturity = mdy(5,15,2008);
investment = 1000000;
redemption = 1014420;
```

```
basis = 2;
r = finance('intrate', settlement, maturity,
investment, redemption, basis);
put r = ;
run;
```

返される *r* の値は 0.05768 です。

サンプル 23: 計算の説明: IPMT

次の例では、指定期間の投資の利払いを計算します。

```
data _null_;
rate = 0.1/12;
per = 2;
nper = 3;
pv = 100;
fv = .;
type = .;
r = finance('ipmt', rate, per, nper, pv, fv, type);
put r = ;
run;
```

返される *r* の値は-94054.82033 です。

サンプル 24: 計算の説明: IRR

次の例では、一連のキャッシュフローの内部利益率を計算します。

```
data _null_;
v1 = -70000;
v2 = 12000;
v3 = 15000;
v4 = 18000;
v5 = 21000;
v6 = 26000;
r = finance('irr', v1, v2, v3, v4, v5, v6);
put r = ;
run;
```

返される *r* の値は 0.086630948 です。

サンプル 25: 計算の説明: ISPMT

次の例では、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。

```
data ispmt;
interest=finance('ISPmt', 0.075/12, 8, 2*12, 5000);
put interest=;
run;
```

返される値は-20.83333333 です。

サンプル 26: 計算の説明: MDURATION

次の例では、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。

```
data _null_;
settlement = mdy(1,1,2008);
maturity = mdy(1,1,2016);
```

```
couponrate = 0.08;
yield = 0.09;
frequency = 2;
basis = 1;
r = finance('mduration', settlement, maturity,
couponrate, yield, frequency, basis);
put r = ;
run;
```

返される *r* の値は 5.7356698139 です。

サンプル 27: 計算の説明: MIRR

次の例では、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。

```
data _null_;
v1 = -1000;
v2 = 3000;
v3 = 4000;
v4 = 5000;
financerate = 0.08;
reinvestrate = 0.10;
r = finance('mirr', v1, v2, v3, v4, financerate, reinvestrate);
put r = ;
run;
```

返される *r* の値は 1.3531420172 です。

サンプル 28: 計算の説明: NOMINAL

次の例では、年間の名目金利を計算します。

```
data _null_;
effectrate = 0.08;
npery = 4;
r = finance('nominal', effectrate, npery);
put r = ;
run;
```

返される *r* の値は 0.0777061876 です。

サンプル 29: 計算の説明: NPER

次の例では、投資の期間数を計算します。

```
data _null_;
rate = 0.08;
pmt = 200;
pv = 1000;
fv = 0;
type = 0;
r = finance('nper', rate, pmt, pv, fv, type);
put r = ;
run;
```

返される *r* の値は-94054.82033 です。

サンプル 30: 計算の説明: NPV

次の例では、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。

```
data _null_;
rate = 0.08;
v1 = 200;
v2 = 1000;
v3 = 0.;
r = finance('npv', rate, v1, v2, v3);
put r = ;
run;
```

返される *r* の値は 1042.5240055 です。

サンプル 31: 計算の説明: ODDFPRICE

次の例では、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。

```
data _null_;
settlement = mdy(1,15,93);
maturity = mdy(1,1,98);
issue = mdy(1,1,93);
firstcoupon = mdy(7,1,94);
rate = 0.07;
yld = 0.06;
redemption = 100;
frequency = 2;
basis = 0;
r = finance('oddfprice',
settlement, maturity, issue, firstcoupon, rate, yld, redemption,
frequency, basis);
put r = ;
run;
```

返される *r* の値は 103.94103984 です。

サンプル 32: 計算の説明: ODDFYIELD

次の例では、端数の開始期間で利回りの利息を計算します。

```
data _null_;
settlement = mdy(1,15,93);
maturity = mdy(1,1,98);
issue = mdy(1,1,93);
firstcoupon = mdy(7,1,94);
rate = 0.07;
pr = 103.94103984;
redemption = 100;
frequency = 2;
basis = 0;
r = finance('oddfyield',
settlement, maturity, issue, firstcoupon, rate, pr, redemption,
frequency, basis);
put r = ;
run;
```

返される *r* の値は 0.06 です。

サンプル 33: 計算の説明: ODDLPRICE

次の例では、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。

```
data _null_;
settlement = mdy(2,7,2008);
maturity = mdy(6,15,2008);
lastinterest = mdy(10,15,2007);
rate = 0.0375;
yield = 0.0405;
redemption = 100;
frequency = 2;
basis = 0;
r = finance('oddlprice', settlement, maturity, lastinterest,
rate, yield, redemption, frequency, basis);
put r = ;
run;
```

返される *r* の値は 99.878286015 です。

サンプル 34: 計算の説明: ODDLYIELD

次の例では、端数の最終期間で証券の利回りを計算します。

```
data _null_;
settlement = mdy(2,7,2008);
maturity = mdy(6,15,2008);
lastinterest = mdy(10,15,2007);
rate = 0.0375;
pr = 99.878286015;
redemption = 100;
frequency = 2;
basis = 0;
r = finance('oddlyield', settlement, maturity, lastinterest,
rate, pr, redemption, frequency, basis);
put r = ;
run;
```

返される *r* の値は 0.0405 です。

サンプル 35: 計算の説明: PMT

次の例では、年金の定期的支払いを計算します。

```
data _null_;
rate = 0.08;
nper = 5;
pv = 91;
fv = 3;
type = 0;
r = finance('pmt', rate, nper, pv, fv, type);
put r = ;
run;
```

返される *r* の値は-23.30290673 です。

サンプル 36: 計算の説明: PPMT

次の例では、指定期間の投資の元本に対する支払いを計算します。

```
data _null_;
rate = 0.08;
```

```
per = 10;
nper = 10;
pv = 200000;
fv = 0;
type = 0;
r = finance('ppmt', rate, per, nper, pv, fv, type);
put r = ;
run;
```

返される *r* の値は-94054.82033 です。

サンプル 37: 計算の説明: PRICE

次の例では、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。

```
data _null_;
settlement = mdy(2,15,2008);
maturity = mdy(11,15,2017);
rate = 0.0575;
yield = 0.065;
redemption = 100;
frequency = 2;
basis = 0;
r = finance('price', settlement, maturity, rate, yield, redemption,
frequency, basis);
put r = ;
run;
```

返される *r* の値は 94.634361621 です。

サンプル 38: 計算の説明: PRICEDISC

次の例では、額面$100 当たりの割引証券の価格を計算します。

```
data _null_;
settlement = mdy(2,15,2008);
maturity = mdy(11,15,2017);
discount = 0.0525;
redemption = 100;
basis = 0;
r = finance('pricedisc', settlement, maturity, discount, redemption, basis);
put r = ;
run;
```

返される *r* の値は 48.8125 です。

サンプル 39: 計算の説明: PRICEMAT

次の例では、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。

```
data _null_;
settlement = mdy(2,15,2008);
maturity = mdy(4,13,2008);
issue = mdy(11,11,2007);
rate = 0.061;
yield = 0.061;
basis = 0;
r = finance('pricemat', settlement, maturity, issue, rate, yield, basis);
put r = ;
run;
```

返される *r* の値は 99.98449888 です。

サンプル 40: 計算の説明: PV

次の例では、投資の現在価値を計算します。

```
data _null_;
rate = 0.05;
nper = 10;
pmt = 1000;
fv = 200;
type = 0;
r = finance('pv', rate, nper, pmt, fv, type);
put r = ;
run;
```

返される *r* の値は-94054.82033 です。

サンプル 41: 計算の説明: RATE

次の例では、年金期間当たりの利率を計算します。

```
data _null_;
nper = 4;
pmt = -2481;
pv = 8000;
r = finance('rate', nper, pmt, pv);
put r = ;
run;
```

返される *r* の値は 0.0921476841 です。

サンプル 42: 計算の説明: RECEIVED

次の例では、完全投資済み証券の満期受け取り額を計算します。

```
data _null_;
settlement = mdy(2,15,2008);
maturity = mdy(5,15,2008);
investment = 1000000;
discount = 0.0575;
basis = 2;
r = finance('received', settlement, maturity, investment, discount, basis);
put r = ;
run;
```

返される *r* の値は 1014584.6544 です。

サンプル 43: 計算の説明: SLN

次の例では、ある期間の資産の定額法による減価償却を計算します。

```
data _null_;
cost = 2000;
salvage = 200;
life = 11;
r = finance('sln', cost, salvage, life);
put r = ;
run;
```

返される *r* の値は 163.63636364 です。

サンプル 44: 計算の説明: SYD

次の例では、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。

```
data _null_;
cost = 2000;
salvage = 200;
life = 11;
per = 1;
r = finance('syd', cost, salvage, life, per);
put r = ;
run;
```

返される *r* の値は 300 です。

サンプル 45: 計算の説明: TBILLEQ

次の例では、米国財務省短期証券の債券換算利回りを計算します。

```
data _null_;
settlement = mdy(3,31,2008);
maturity = mdy(6,1,2008);
discount = 0.0914;
r = finance('tbilleq', settlement, maturity, discount);
put r = ;
run;
```

返される *r* の値は 0.0941514936 です。

サンプル 46: 計算の説明: TBILLPRICE

次の例では、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。

```
data _null_;
settlement = mdy(3,31,2008);
maturity = mdy(6,1,2008);
discount = 0.09;
r = finance('tbillprice', settlement, maturity, discount);
put r = ;
run;
```

返される *r* の値は 98.45 です。

サンプル 47: 計算の説明: TBILLYIELD

次の例では、米国財務省短期証券の利回りを計算します。

```
data _null_;
settlement = mdy(3,31,2008);
maturity = mdy(6,1,2008);
pr = 98;
r = finance('tbillyield', settlement, maturity, pr);
put r = ;
run;
```

返される *r* の値は 0.1184990125 です。

サンプル 48: 計算の説明: VDB

次の例では、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。

```
data _null_;
cost = 2400;
salvage = 300;
life = 10;
startperiod = 0;
endperiod = 1;
factor = 1.5;
r = finance('vdb', cost, salvage, life, startperiod, endperiod, factor);
put r = ;
run;
```

返される *r* の値は 360 です。

サンプル 49: 計算の説明: XIRR

次の例では、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。

```
data _null_;
v1 = -10000; d1 = mdy(1,1,2008);
v2 = 2750; d2 = mdy(3,1,2008);
v3 = 4250; d3 = mdy(10,30,2008);
v4 = 3250; d4 = mdy(2,15,2009);
v5 = 2750; d5 = mdy(4,1,2009);
r = finance('xirr', v1, v2, v3, v4, v5, d1, d2, d3, d4, d5, 0.1);
put r = ;
run;
```

返される *r* の値は 0.3733625335 です。

サンプル 50: 計算の説明: XNPV

次の例では、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。

```
data _null_;
r = 0.09;
v1 = -10000; d1 = mdy(1,1,2008);
v2 = 2750; d2 = mdy(3,1,2008);
v3 = 4250; d3 = mdy(10,30,2008);
v4 = 3250; d4 = mdy(2,15,2009);
v5 = 2750; d5 = mdy(4,1,2009);
r = finance('xnpv', r, v1, v2, v3, v4, v5, d1, d2, d3, d4, d5);
put r = ;
run;
```

返される *r* の値は 2086.647602 です。

サンプル 51: 計算の説明: YIELD

次の例では、定期的に利息を支払う証券の利回りを計算します。

```
data _null_;
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 = ;
run;
```

返される *r* の値は 0.0650000069 です。

サンプル 52: 計算の説明: YIELDDISC

次の例では、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。

```
data _null_;
settlement = mdy(2,15,2008);
maturity = mdy(11,15,2016);
pr = 95.04287;
redemption = 100;
basis = 0;
r = finance('yielddisc', settlement, maturity, pr, redemption, basis);
put r = ;
run;
```

返される *r* の値は 0.0059607748 です。

サンプル 53: 計算の説明: YIELDMAT

次の例では、満期に利息を支払う証券の年間利回りを計算します。

```
data _null_;
settlement = mdy(3,15,2008);
maturity = mdy(11,3,2008);
issue = mdy(11,8,2007);
rate = 0.0625;
pr = 100.0123;
basis = 0;
r = finance('yieldmat', settlement, maturity, issue, rate, pr, basis);
put r = ;
run;
```

返される *r* の値は 0.0609543337 です。

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: 文字関数

制限事項: 英語以外の言語を使用する場合、可能な限り I18N レベル 1 の関数の使用は避けてください。I18N レベル 1 の関数は、特定の環境下では 2 バイト文字セット(DBCS)または複数バイト文字セット(MBCS)エンコーディングを使用すると正常に動作しない場合があります。

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* <,*modifiers*> <,*startpos*>)

FIND(*string*,*substring* <,*startpos*> <,*modifiers*>)

必須引数

string
部分文字列を検索する文字定数、変数または式を指定します。

ヒント: 文字のリテラル文字列を引用符で囲みます。

substring
string で検索する文字の部分文字列を指定する文字定数、変数または式です。

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。次の修飾子 (*modifiers*)には大文字と小文字のどちらでも使用できます。

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
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。

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を指定し、検索の方向を指定する整数値を持つ数値定数、変数または式です。

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。

startpos 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を決定します。検索方向は、*startpos* の符合で決まります。

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。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
whereisshe=find('She sells seashells? Yes, she does.','she '); put whereisshe;	27
variable1='She sells seashells? Yes, she does.'; variable2='she '; variable3='i'; whereisshe_i=find(variable1,variable2,variable3); put whereisshe_i;	1

expression1='She sells seashells? '||'Yes, she does.';
expression2=kscan('he or she',3)||' ';
expression3=trim('t ');
whereisshe_t=find(expression1,expression2,expression3);
put whereisshe_t;
	14

xyz='She sells seashells? Yes, she does.';
startposvar=22;
whereisshe_22=find(xyz,'she',startposvar);
put whereisshe_22;
	27

xyz='She sells seashells? Yes, she does.';
startposexp=1-23;
whereisShe_ineg22=find(xyz,'She','i',startposexp);
put whereisShe_ineg22;
	14

関連項目:

関数:

- “COUNT 関数” (331 ページ)
- “FINDC 関数” (450 ページ)
- “INDEX 関数” (531 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: 文字関数

制限事項: 英語以外の言語を使用する場合、可能な限り I18N レベル 1 の関数の使用は避けてください。I18N レベル 1 の関数は、特定の環境下では 2 バイト文字セット(DBCS)または複数バイト文字セット(MBCS)エンコーディングを使用すると正常に動作しない場合があります。

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*>)

FINDC(*string*, *charlist* <,*modifiers*>)

FINDC(*string*, *charlist*, *modifier(s)* <, *startpos*>)

FINDC(*string*, *charlist*, <*startpos*> , <*modifiers*)>

必須引数

string
検索する文字列を指定する文字定数、変数または式です。

ヒント: 文字のリテラル文字列を引用符で囲みます。

charlist
文字のリストを初期化する定数、変数または文字式です(省略可能)。*modifier* 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。

modifier
任意の文字定数、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。次の文字(大文字または小文字)を修飾子として使用できます。

空白
無視されます。

a
A
文字のリストにアルファベット文字を追加します。

b
B
startpos 引数の符合に関係なく、左から右ではなく、右から左に検索します。

c
C
文字のリストに制御文字を追加します。

d
D
文字のリストに数字を追加します。

f
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
G
文字のリストにグラフィカル文字を追加します。

h
H
文字のリストに水平タブを追加します。

i
I
検索時に大文字と小文字を区別しません。

k
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。

l
L
小文字を文字リストに追加します。

n
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
O
charlist 引数および *modifier* 引数を 1 回だけ処理します。FINDC 関数の呼び出し時に毎回処理しません。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、*charlist* 引数と *modifier* 引数が変更されないループで FINDC を呼び出すときに、より迅速に実行できます。

p
P
文字のリストに句読点を追加します。

s
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
T
string 引数と *charlist* 引数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、FINDC 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
U
大文字を文字リストに追加します。

w
W
印刷可能文字を文字リストに追加します。

x
X
文字のリストに 16 進文字を追加します。

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。

オプション引数

startpos
検索を開始する位置を指定し、検索の方向を指定する整数値を持つ任意の数値定数、変数または式です。

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を指定します。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* の符合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	*startpos* の位置から検索を開始して右方向に進みます。*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。
0 より小さい	*startpos* の位置から検索を開始して左方向に進みます。*startpos* が文字列の負の長さよりも小さい場合、文字列の最後から検索を開始します。
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 *modifier* 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 *modifier* 引数も *startpos* 引数もありません。

サンプル

サンプル 1: 文字列の文字の検索

この例では、文字列を検索して、検出された文字を返します。

```
data _null_;
string = 'Hi, ho!';
charlist = 'hi';
j = 0;
do until (j = 0);
j = findc(string, charlist, j+1);
if j = 0 then put +3 "That's all";
else do;
c = substr(string, j, 1);
put +3 j= c=;
end;
end;
run;
```

SAS は次の出力を口グに書き込みます。

```
 j=2 c=i
j=5 c=h
That's all
```

サンプル 2: 文字列の文字の検索(大文字と小文字を区別しない)

この例では、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。

```
data _null_;
string = 'Hi, ho!';
charlist = 'ho';
j = 0;
do until (j = 0);
j = findc(string, charlist, j+1, "i");
if j = 0 then put +3 "That's all";
else do;
c = substr(string, j, 1);
put +3 j= c=;
end;
end;
run;
```

SAS は次の出力を口グに書き込みます。

```
 j=1 c=H
j=5 c=h
j=6 c=o
That's all
```

サンプル 3: K 修飾子を使用した文字の検索

この例では、文字列を検索して、文字リストにない文字を返します。

```
data _null_;
string = 'Hi, ho!';
charlist = 'hi';
j = 0;
do until (j = 0);
j = findc(string, charlist, "k", j+1);
if j = 0 then put +3 "That's all";
else do;
```

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

SAS は次の出力を口グに書き込みます。

```
 j=1 c=H
j=3 c=,
j=4 c=
j=6 c=o
j=7 c=!
That's all
```

サンプル 4: h、i および空白文字の検索

この例では、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。

```
data _null_;
whereishi=0;
do until(whereishi=0);
whereishi=findc('Hi there, Ian!','hi ',whereishi+1);
if whereishi=0 then put "The End";
else do;
whatfound=substr('Hi there, Ian!',whereishi,1);
put whereishi= whatfound=;
end;
end;
run;
```

SAS は次の出力を口グに書き込みます。

```
whereishi=2 whatfound=i
whereishi=3 whatfound=
whereishi=5 whatfound=h
whereishi=10 whatfound=
The End
```

サンプル 5: 文字 h および i の検索(大文字と小文字を区別しない)

この例では、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。

```
data _null_;
whereishi_i=0;
do until(whereishi_i=0);
variable1='Hi there, Ian!';
variable2='hi';
variable3='i';
whereishi_i=findc(variable1,variable2,variable3,whereishi_i+1);
if whereishi_i=0 then put "The End";
else do;
whatfound=substr(variable1,whereishi_i,1);
put whereishi_i= whatfound=;
end;
end;
run;
```

SAS は次の出力を口グに書き込みます。

```
whereishi_i=1 whatfound=H
whereishi_i=2 whatfound=i
whereishi_i=5 whatfound=h
whereishi_i=11 whatfound=I
The End
```

サンプル 6: 文字 h および i の検索(末尾の空白を削除)

この例では、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。

```
data _null_;
whereishi_t=0;
do until(whereishi_t=0);
expression1='Hi there, '||'Ian!';
expression2=kscan('bye or hi',3)||' ';
expression3=trim('t ');
whereishi_t=findc(expression1,expression2,expression3,whereishi_t+1);
if whereishi_t=0 then put "The End";
else do;
whatfound=substr(expression1,whereishi_t,1);
put whereishi_t= whatfound=;
end;
end;
run;
```

SAS は次の行の出力を口グに書き込みます。

```
whereishi_t=2 whatfound=i
whereishi_t=5 whatfound=h
The End
```

サンプル 7: h、i、H、I を除くすべての文字の検索

この例では、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。

```
data _null_;
whereishi_iv=0;
do until(whereishi_iv=0);
xyz='Hi there, Ian!';
whereishi_iv=findc(xyz,'hi',whereishi_iv+1,'iv');
if whereishi_iv=0 then put "The End";
else do;
whatfound=substr(xyz,whereishi_iv,1);
put whereishi_iv= whatfound=;
end;
end;
run;
```

SAS は次の出力を口グに書き込みます。

```
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” (98 ページ)
- “ANYALPHA 関数” (100 ページ)
- “ANYCNTRL 関数” (102 ページ)
- “ANYDIGIT 関数” (103 ページ)
- “ANYGRAPH 関数” (106 ページ)
- “ANYLOWER 関数” (109 ページ)
- “ANYPRINT 関数” (112 ページ)
- “ANYPUNCT 関数” (114 ページ)
- “ANYSPACE 関数” (116 ページ)
- “ANYUPPER 関数” (118 ページ)
- “ANYXDIGIT 関数” (119 ページ)
- “COUNTC 関数” (333 ページ)
- “INDEXC 関数” (533 ページ)
- “VERIFY 関数” (918 ページ)
- “NOTALNUM 関数” (668 ページ)
- “NOTALPHA 関数” (669 ページ)
- “NOTCNTRL 関数” (671 ページ)
- “NOTDIGIT 関数” (673 ページ)
- “NOTGRAPH 関数” (678 ページ)
- “NOTLOWER 関数” (680 ページ)
- “NOTPRINT 関数” (683 ページ)
- “NOTPUNCT 関数” (684 ページ)
- “NOTSPACE 関数” (686 ページ)
- “NOTUPPER 関数” (688 ページ)
- “NOTXDIGIT 関数” (690 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: 文字関数

構文

FINDW(*string*, *word* <, *chars*>)

FINDW(*string*, *word*, *chars*, *modifiers*<, *startpos*>)

FINDW(*string*, *word*, *chars*, *startpos*< , *modifiers*>)

FINDW(*string*, *word*, *startpos*< , *chars*<, *modifiers*>>)

必須引数

string
検索する文字列を指定する文字定数、変数または式です。

word
検索する単語を指定する文字定数、変数または式です。

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。

startpos
検索を開始する位置を指定し、検索の方向を指定する整数値を持つ数値定数、変数または式です(省略可能)。

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 FINDW 関数のアクションを変更します。

ヒント *modifier* 引数を使用する場合、*chars* 引数の後に配置する必要があります。

次の文字を修飾子として使用できます。

空白
無視されます。

a
A
文字のリストにアルファベット文字を追加します。

b
B
startpos 引数の符合に関係なく、左から右ではなく、右から左にスキャンします。

c
C
文字のリストに制御文字を追加します。

d
D
文字のリストに数字を追加します。

e
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。不完全な単語は数えられません。

f
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
G
文字のリストにグラフィカル文字を追加します。

h
H
文字のリストに水平タブを追加します。

I
I
大文字か小文字かは無視します。

k
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。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
L
小文字を文字リストに追加します。

m
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
N
数字、アンダースコアおよび英文字(VALIDVARNAME=V7 で SAS 変数名の先頭文字の次に使用できる文字)を文字リストに追加します。

o
O
chars 引数および *modifier* 引数を 1 回だけ処理します。FINDW 関数の呼び出し時に毎回処理しません。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、*chars* 引数と *modifier* 引数が変更されないループで FINDW を呼び出すときに、より迅速に実行できます。

p
P
文字のリストに句読点を追加します。

q
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、一致しない引用符が含まれている場合、左から右へのスキャンでは、右から左へのスキャンとは異なる単語が生成されます。

r
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
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
T
string、*word*、*chars* 引数から末尾の空白を削除します。

u
U
大文字を文字リストに追加します。

w
W
印刷可能文字を文字リストに追加します。

x
X
文字のリストに 16 進文字を追加します。

詳細

"区切り文字"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。*chars* 引数、*modifier* 引数、またはその両方を使用して区切り文字を指定できます。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。

"ワード"の定義

"単語"は、次の両方の特性がある部分文字列です。

- 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* で単語内の位置を指定した場合、その単語は数えられません。
- 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 引数を指定する場合、*startpos* の絶対値で検索の開始位置を指定します。検索方向は、*startpos* の符合で指定します。

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て右方向に進みます。*startpos* が文字列の長さよりも大きい場合、FINDW 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て左方向に進みます。*startpos* が文字列の負の長さよりも小さい場合、文字列の最後から検索を開始します。
0	FINDW は値 0 を返します。

startpos 引数や B 修飾子を指定しない場合、FINDW は文字列の最初から開始して左から右に検索します。B 修飾子は指定するが、*startpos* 引数は使用しない場合、FINDW は文字列の最後から開始して右から左に検索します。

ASCII および EBCDIC 環境での FINDW 関数の使用

2 つの引数のみを指定して FINDW 関数を使用する場合、デフォルトの区切り文字は、コンピュータで使用している文字(ASCII または EBCDIC)によって異なります。

- コンピュータで ASCII 文字が使われている場合、デフォルトの区切り文字は次のとおりです。

  空白 ! $ % & ( ) * + , - . / ; < ^ |

  ^文字を含まない ASCII 環境の場合、FINDW 関数はかわりに~文字を使用します。

- コンピュータで EBCDIC 文字が使われている場合、デフォルトの区切り文字は次のとおりです。

  空白 ! $ % & ( ) * + , - . / ; < ¬ | ¢

NULL 引数を使用する

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。

サンプル

サンプル 1: 文字列からの単語の検索

次の例では、文字列から単語"she"を検索し、単語の開始位置を返します。

```
data _null_;
whereisshe=findw('She sells sea shells? Yes, she does.','she');
put whereisshe=;
run;
```

SAS は次の出力を口グに書き込みます。

```
whereisshe=28
```

サンプル 2: Chars および Startpos 引数を使用した文字列の検索

次の例では、単語"rain"が 2 回出現します。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* 引数では、区切り文字として空白文字が指定されています。

```
data _null_;
result = findw('At least 2.5 meters of rain falls in a rain forest.',
'rain', ' ', 25);
put result=;
run;
```

SAS は次の出力を口グに書き込みます。

```
result=40
```

サンプル 3: I 修飾子および Startpos 引数を使用した文字列の検索

次の例では、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。

```
data _null_;
string='Artists from around the country display their art at
an art festival.';
result=findw(string, 'Art',' ', 'i', 10);
put result=;
run;
```

SAS は次の出力を口グに書き込みます。

```
result=47
```

サンプル 4: E 修飾子を使用した文字列の検索

次の例では、E 修飾子を使用して、単語"art"の検索中にスキャンした完全な単語の数を返します。

```
data _null_;
string='Artists from around the country display their art at
an art festival.';
result=findw(string,'art',' ','E');
put result=;
run;
```

SAS は次の出力を口グに書き込みます。

```
result=8
```

サンプル 5: E 修飾子および Startpos 引数を使用した文字列の検索

次の例では、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。

```
data _null_;
string='Artists from around the country display their art at
an art festival.';
result=findw(string, 'art',' ','E',50);
put result=;
run;
```

SAS は次の出力を口グに書き込みます。

```
result=3
```

サンプル 6: 2 つの修飾子を使用した文字列の検索

次の例では、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。

```
data _null_;
string='The Great Himalayan National Park was created in 1984. Because
of its terrain and altitude, the park supports a diversity
of wildlife and vegetation.';
result=findw(string,'park',' ','I E');
put result=;
run;
```

SAS は次の出力を口グに書き込みます。

```
result=5
```

サンプル 7: R 修飾子を使用した文字列の検索

次の例では、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。

```
data _null_;
string='Artists from around the country display their art at
an art festival.';
word=' art ';
result=findw(string, word, ' ', 'R');
put result=;
run;
```

SAS は次の出力を口グに書き込みます。

```
result=47
```

関連項目:

関数:

- “COUNTW 関数” (337 ページ)
- “FIND 関数” (448 ページ)
- “FINDC 関数” (450 ページ)
- “INDEXW 関数” (534 ページ)
- “SCAN 関数” (824 ページ)

CALL ルーチン:

- “CALL SCAN ルーチン” (233 ページ)

FINFO 関数

ファイル情報項目の値を返します。

カテゴリ: 外部ファイル

参照項目: “FINFO Function: Windows” in *SAS Companion for Windows*
“FINFO Function: UNIX” in *SAS Companion for UNIX Environments*

"FINFO Function: z/OS" in *SAS Companion for z/OS*

構文

FINFO(*file-id*,*info-item*)

必須引数

file-id
(通常、FOPEN 関数で)ファイルを開いたときに割り当てられた識別子を指定する数値定数、変数または式です。

info-item
取得するファイル情報項目の名前を指定する文字定数、変数または式です。

詳細

FINFO は、外部ファイルのシステムに依存する情報項目の値を返します。*info-item* に指定した値が無効な場合、FINFO は空白を返します。

動作環境の情報
ファイルで使用できる情報は、動作環境やアクセス方式によって異なります。

比較

- FOPTNAME 関数は、使用可能なファイル情報項目の名前を決定します。
- FOPTNUM 関数は、システムに依存する使用可能な情報項目の数を決定します。

サンプル

この例では、SAS データセットの外部ファイルに関する情報項目を保存します。

```
data info;
length infoname infoval $60;
drop rc fid infonum i close;
rc=filename('abc','physical-filename');
fid=fopen('abc');
infonum=foptnum(fid);
do i=1 to infonum;
infoname=foptname(fid,i);
infoval=finfo(fid,infoname);
output;
end;
close=fclose(fid);
run;
```

関連項目:

関数:

- "FCLOSE 関数" (394 ページ)
- "FOPTNUM 関数" (480 ページ)
- "MOPEN 関数" (653 ページ)

FINV 関数

F 分布から分位点を返します。

カテゴリ: 分位点

構文

FINV (*p*, *ndf*, *ddf* <,*nc*>)

必須引数

p
数値の確率です。
範囲: $0 \leq p < 1$

ndf
数値の分子の自由度パラメータです。
範囲: $ndf > 0$

ddf
数値の分母の自由度パラメータです。
範囲: $ddf > 0$

オプション引数

nc
数値の非心度パラメータです(省略可能)。
範囲: $nc \geq 0$

詳細

FINV 関数は、*F* 分布(分子の自由度は *ndf*、分母の自由度は *ddf*、非心度パラメータは *nc*)から *p*番目の分位点を返します。*F* 分布のオブザベーションが分位点未満となる確率は *p* です。この関数では、整数以外の自由度パラメータ *ndf* および *ddf* を使用できます。

パラメータ *nc* (省略可能)を指定していない場合や値が 0 の場合、心度 *F* 分布から分位点が返されます。非心度パラメータ *nc* は、X および Y が正規ランダム変数(平均はそれぞれ μ と 0、分散は 1)の場合に、X^2 / Y^2 が非心度 *F* 分布($nc = \mu^2$)になるように定義されます。

注意:
nc の値が大きい場合､アルゴリズムが失敗する可能性があります。その場合、欠損値が返されます。

注: FINV は、PROBF 関数の逆数です。

サンプル

これらのステートメントは、心度 *F* 分布(自由度は 2 および 10)と非心度 *F* 分布(自由度は 2 および 10.3、非心度パラメータは 2)の 95番目の分位点の値を計算します。

SAS ステートメント	結果
q1=finv(.95,2,10);	4.1028210151
q2=finv(.95,2,10.3,2);	7.583766024

関連項目:

関数:

- “QUANTILE 関数” (778 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: 州コード/郵便番号

構文

FIPNAME(*expression*)

必須引数

式
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 FIPNAME 関数から値が返される場合、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、米国の Federal Information Processing Standards (FIPS)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、同じ引数をとりますが、異なる値を返します。FIPNAME は、大文字の州名を返します。FIPNAMEL は、大文字小文字混在の州名を返します。FIPSTATE は、2 文字の州コード(米国領の場合は世界中で利用されている GSA 地理コード)を大文字で返します。

サンプル

この例では、FIPNAME、FIPNAMEL、FIPSTATE を使用した場合の違いを示します。

SAS ステートメント	結果
x=fipname(37); put x;	NORTH CAROLINA

SAS ステートメント	結果
x=fipnamel(37); put x;	North Carolina
x=fipstate(37); put x;	NC

関連項目:

関数:

- “FIPNAMEL 関数” (467 ページ)
- “FIPSTATE 関数” (468 ページ)
- “STFIPS 関数” (860 ページ)
- “STNAME 関数” (861 ページ)
- “STNAMEL 関数” (862 ページ)

FIPNAMEL 関数

2 桁の FIPS コードを大文字小文字混在の州名に変換します。

カテゴリ: 州コード/郵便番号

構文

FIPNAMEL(*expression*)

必須引数

式
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 FIPNAMEL 関数から値が返される場合、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、米国の Federal Information Processing Standards (FIPS)コードを大文字小文字混在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、同じ引数をとりますが、異なる値を返します。FIPNAME は、大文字の州名を返します。FIPNAMEL は、大文字小文字混在の州名を返します。FIPSTATE は、2 文字の州コード(米国領の場合は世界中で利用されている GSA 地理コード)を大文字で返します。

サンプル

この例では、FIPNAME、FIPNAMEL、FIPSTATE を使用した場合の違いを示します。

SAS ステートメント	結果
x=fipname(37); put x;	NORTH CAROLINA
x=fipnamel(37); put x;	North Carolina
x=fipstate(37); put x;	NC

関連項目:

関数:

- “FIPNAME 関数” (466 ページ)
- “FIPSTATE 関数” (468 ページ)
- “STFIPS 関数” (860 ページ)
- “STNAME 関数” (861 ページ)
- “STNAMEL 関数” (862 ページ)

FIPSTATE 関数

2 桁の FIPS コードを 2 文字の州コードに変換します。

カテゴリ: 州コード/郵便番号

構文

FIPSTATE(*expression*)

必須引数

式
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、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、米国の Federal Information Processing Standards (FIPS)コードを大文字の 2 文字の州コード(米国領の場合は世界中で利用されている GSA 地理コード)に変換します。

比較

FIPNAME、FIPNAMEL および FIPSTATE 関数は、同じ引数をとりますが、異なる値を返します。FIPNAME は、大文字の州名を返します。FIPNAMEL は、大文字小文字混在の州名を返します。FIPSTATE は、2 文字の州コード(米国領の場合は世界中で利用されている GSA 地理コード)を大文字で返します。

サンプル

この例では、FIPNAME、FIPNAMEL、FIPSTATE を使用した場合の違いを示します。

SAS ステートメント	結果
x=fipname(37); put x;	NORTH CAROLINA
x=fipnamel(37); put x;	North Carolina
x=fipstate(37); put x;	NC

関連項目:

関数:

- “FIPNAME 関数” (466 ページ)
- “FIPNAMEL 関数” (467 ページ)
- “STFIPS 関数” (860 ページ)
- “STNAME 関数” (861 ページ)
- “STNAMEL 関数” (862 ページ)

FIRST 関数

文字列の最初の文字を返します。

カテゴリ: 文字関数

構文

FIRST(*string*)

必須引数

string
文字列を指定します。

詳細

DATA ステップでは、FIRST 関数の対象変数のデフォルトの長さは 1 です。

FIRST 関数は長さ 1 の文字列を返します。*string* の長さが 0 の場合、FIRST 関数は 1 つの空白を返します。

比較

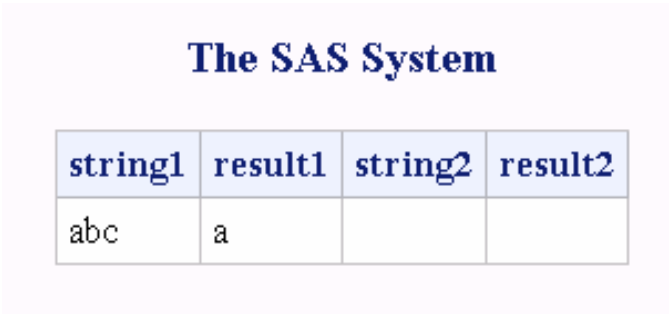
FIRST 関数は、CHAR(*string*, 1)および SUBPAD(*string*, 1, 1)と同じ結果を返します。結果は同じでも、対象変数のデフォルトの長さは異なります。

サンプル

次の例では、FIRST 関数を使用した結果を示します。

```
data test;
string1="abc";
result1=first(string1);
string2="";
result2=first(string2);
run;
proc print noobs data=test;
run;
```

画面 2.32 *FIRST 関数からの出力*



The SAS System

string1	result1	string2	result2
abc	a		

関連項目:

関数:

- “CHAR 関数” (292 ページ)

FLOOR 関数

予期しない浮動小数点の結果が生じないようにファジー処理して、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: 切り捨て関数

構文

FLOOR (*argument*)

必須引数

引数
数値の定数、変数または式を指定します。

詳細

引数が整数の 1E-12 以内の場合、関数はその整数を返します。

比較

FLOORZ 関数とは異なり、FLOOR 関数は結果をファジー処理します。引数が整数の 1E-12 以内でも、FLOOR 関数はその整数と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。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果

var1=2.1;
a=floor(var1);
put a;
	2

var2=-2.4;
b=floor(var2);
put b;
	-3

c=floor(-1.6); put c;	-2
d=floor(1.-1.e-13); put d;	1
e=floor(763); put e;	763
f=floor(-223.456); put f;	-224

関連項目:

関数:

- “FLOORZ 関数” (471 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: 切り捨て関数

構文

FLOORZ (*argument*)

必須引数

引数
数値定数、変数または式です。

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 以内でも、FLOOR 関数はその整数と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。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果

var1=2.1;
a=floorz(var1);
put a;
	2

var2=-2.4;
b=floorz(var2);
put b;
	-3

c=floorz(-1.6); put c;	-2

var6=(1.-1.e-13);
d=floorz(1-1.e-13);
put d;
	0

e=floorz(763); put e;	763
f=floorz(-223.456); put f;	-224

関連項目:

関数:

- “FLOOR 関数” (470 ページ)

FNONCT 関数

F 分布の非心度パラメータの値を返します。

カテゴリ: 数学関数

構文

FNONCT(*x*,*ndf*,*ddf*,*prob*)

必須引数

x
数値のランダム変数です。
範囲: $x \geq 0$

ndf
数値の分子の自由度パラメータです。
範囲: $ndf > 0$

ddf
数値の分母の自由度パラメータです。
範囲: $ddf > 0$

prob
確率です。
範囲: $0 < prob < 1$

詳細

FNONCT 関数は、非心度 *F* 分布(パラメータは *x*、*ndf*、*ddf*、*nc*)から負でない非心度パラメータを返します。*prob* が心度 *F* 分布(パラメータは *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。

$$P_f(x \mid ndf,\ ddf,\ nc) - prob = 0$$

前述の式には次の関係が適用されます。

$$P_f(x \mid ndf,\ ddf,\ nc) = \varepsilon^{\frac{-nc}{2}} \sum_{j=0}^{\infty} \frac{\left(\frac{nc}{2}\right)^j}{j!} I_{\frac{(ndf)x}{ddf+(ndf)x}}\left(\frac{ddf}{2} + j,\ \frac{ddf}{2}\right)$$

この式では、$I(\ldots)$は、次の式によって得られるベータ分布の確率です。

$$I_x(a,\ b) = \frac{\int_0^x t^{a-1}(1-t)^{b-1}\,dt}{\frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}}$$

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。

サンプル

```
data work;
x=2;
df=4;
ddf=5;
do nc=1 to 3 by .5;
prob=probf(x,df,ddf,nc);
```

```
ncc=fnonct(x,df,ddf,prob);
output;
end;
run;
proc print;
run;
```

画面 2.33 FNONCT の出力例

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

FNOTE 関数

最後に読み込まれたレコードを特定し、FPOINT 関数で使用できる値を返します。

カテゴリ: 外部ファイル

構文

FNOTE(*file-id*)

必須引数

file-id
(通常、FOPEN 関数で)ファイルを開いたときに割り当てられた識別子を指定する数値定数、変数または式です。

詳細

FNOTE はブックマークのように使用できます(ファイルの位置をマークし、FPOINT を使用してアプリケーションが後でその位置に戻れるようにします)。FNOTE から返される値は、FPOINT 関数が特定のレコードのファイルポインタを再配置するために必要になります。

FNOTE の各識別子に関連付けられたメモリを解放するには、DROPNOTE を使用します。

注: 新しいレコードが現在のレコードよりも長い場合、現在のレコードのかわりに新しいレコードを書き込むことはできません。

サンプル

この例では、ファイル参照名 MYFILE を外部ファイルに割り当ててそのファイルを開こうとします。ファイルが正常に開いた場合(変数 FID の値が正になる)、レコードを読み込み、3 番目に読み込まれたレコードの位置を変数 NOTE3 に保存します。その後、FPOINT を使用して NOTE3 にポインタを戻してファイルを更新します。レコードを更新したら、ファイルを閉じます。

```
%let
fref=MYFILE;
%let rc=%sysfunc(filename(fref,
physical-filename));
%let fid=%sysfunc(fopen(&fref,u));
%if &fid > 0 %then
%do;
%let rc=%sysfunc(fread(&fid));
/* Read second record. */
%let rc=%sysfunc(fread(&fid));
/* Read third record. */
%let rc=%sysfunc(fread(&fid));
/* Note position of third record. */
%let note3=%sysfunc(fnote(&fid));
/* Read fourth record. */
%let rc=%sysfunc(fread(&fid));
/* Read fifth record. */
%let rc=%sysfunc(fread(&fid));
/* Point to third record. */
%let rc=%sysfunc(fpoint(&fid,&note3));
/* Read third record. */
%let rc=%sysfunc(fread(&fid));
/* Copy new text to FDB. */
%let rc=%sysfunc(fput(&fid,New text));
/* Update third record */
/* with data in FDB. */
%let rc=%sysfunc(fwrite(&fid));
/* Close file. */
%let rc=%sysfunc(fclose(&fid));
%end;
%let rc=%sysfunc(filename(fref));
```

関連項目:

関数:

- “DROPNOTE 関数” (380 ページ)
- “FCLOSE 関数” (394 ページ)
- “FILENAME 関数” (404 ページ)
- “FOPEN 関数” (476 ページ)
- “FPOINT 関数” (481 ページ)
- “FPUT 関数” (484 ページ)
- “FREAD 関数” (486 ページ)
- “FREWIND 関数” (487 ページ)

- “FWRITE 関数” (491 ページ)
- “MOPEN 関数” (653 ページ)

FOPEN 関数

外部ファイルを開いて、ファイル識別子の値を返します。

カテゴリ: 外部ファイル

参照項目: “FOPEN Function: z/OS” in *SAS Companion for z/OS*

構文

FOPEN(*fileref* <,*open-mode*<,*record-length*<,*record-format*>>>)

必須引数

fileref
外部ファイルに割り当てられたファイル参照名を指定する文字定数、変数または式です。

ヒント: *fileref* が 8 文字よりも長い場合、8 文字になるように切り捨てられます。

オプション引数

open-mode
ファイルへのアクセスの種類を指定する文字定数、変数または式です。

A APPEND モード: 現在のファイルの最後の後に新しいレコードを書き込むことができます。

I INPUT モード: 読み取り専用です(デフォルト)。

O OUTPUT モード: FILENAME ステートメントまたは関数の動作環境オプションで指定したデフォルトの OPEN モードになります。動作環境オプションが指定されていない場合、ファイルの先頭に新しいレコードを書き込むことができます。

S Sequential Input モード: パイプや他のシーケンシャルデバイス(ハードウェアポートなど)に使用されます。

U UPDATE モード: 読み込みと書き込みの両方ができます。

デフォルト: I

record-length
ファイルの論理レコード長を指定する数値定数、変数または式です。ファイルの既存のレコード長を使用するには、0 の長さを指定するか、ここでは値を指定しません。

record-format
ファイルのレコード形式を指定する文字定数、変数または式です。既存のレコード形式を使用する場合は、ここでは値を指定しません。有効な値は、次のとおりです。

B データがバイナリデータとして解釈されます。

D デフォルトのレコード形式を使用します。

E　編集可能なレコード形式を使用します。

F　固定長レコードがファイルに含まれます。

P　動作環境に依存するレコード形式のプリンタ改行制御がファイルに含まれます。*注:* FBA または VBA レコード形式の z/OS データセットの場合、*record-format* 引数に'P'を指定します。

V　可変長レコードがファイルに含まれます。

注:
引数が無効な場合、FOPEN は0 を返し、対応するエラーメッセージのテキストをSYSMSG 関
ERROR_自動変数も設定されません。

詳細

注意:
OUTPUT モードを使用する場合は注意が必要です。 出力のために既存のファイルを開くと、現在のファイルの内容が警告なしで上書きされます。

FOPEN 関数は、読み込みまたは更新のために既存のファイルを開き、開いているファイルの識別に使用するファイル識別子の値を他の関数に返します。FOPEN 関数を呼び出す前にファイル参照名を既存のファイルに関連付ける必要があります。ファイルを開くことができなかった場合、FOPEN は 0 を返します。ファイル参照名は、FILENAME ステートメントまたは FILENAME 外部ファイルアクセス関数を使用して割り当てることができます。一部の動作環境では、システムコマンドを使用してファイル参照名を割り当てることもできます。

FOPEN 関数をマクロから呼び出す場合、マクロの関数に渡された呼び出しの結果のみが有効になります。FOPEN 関数を DATA ステップから呼び出す場合、同じ DATA ステップの関数に渡された結果のみが有効になります。

動作環境の情報
動作環境によっては FCLOSE を使用する必要がない場合もありますが、FOPEN を使用してファイルを開いた場合は DATA ステップの最後で FCLOSE 関数を使用することをお勧めします。FOPEN の詳細については、現在の動作環境向けの SAS ドキュメントを参照してください。

サンプル

サンプル 1: デフォルトを使用してファイルを開く
この例では、ファイル参照名 MYFILE を既存のファイルに割り当て、すべてのデフォルトを使用して入力のためにファイルを開きます。マクロステートメントでは、文字列を引用符で囲む必要はありません。

```
%let filrf=myfile;
%let rc=%sysfunc(filename(filrf,
physical-filename));
%let fid=%sysfunc(fopen(&filrf));
```

サンプル 2: デフォルトを使用しないでファイルを開く
この例では、デフォルトを使用しないで入力のためにファイルを開きます。マクロステートメントでは、文字列を引用符で囲む必要はありません。

```
%let fid=%sysfunc(fopen(file2,o,132,e));
```

サンプル 3: エラー処理

この例では、エラーの確認方法や SYSMSG 関数がエラーメッセージを書き込む方法を示します。

```
data _null_;
f=fopen('bad','?');
if not f then do;
m=sysmsg();
put m;
abort;
end;
... more SAS statements ...
run;
```

関連項目:

関数:

- “DOPEN 関数” (375 ページ)
- “FCLOSE 関数” (394 ページ)
- “FILENAME 関数” (404 ページ)
- “FILEREF 関数” (407 ページ)
- “MOPEN 関数” (653 ページ)
- “SYSMSG 関数” (880 ページ)

ステートメント:

- “FILENAME ステートメント” (*SAS ステートメント: リファレンス*)

FOPTNAME 関数

ファイルに関する情報項目の名前を返します。

カテゴリ: 外部ファイル

参照項目: “FOPTNAME Function: Windows” in *SAS Companion for Windows*
“FOPTNAME Function: UNIX” in *SAS Companion for UNIX Environments*
“FOPTNAME Function: z/OS” in *SAS Companion for z/OS*

構文

FOPTNAME(*file-id*,*nval*)

必須引数

file-id
通常、FOPEN 関数によってファイルが開かれたときに割り当てられた識別子を指定する数値変数です。

nval
情報項目数を指定する数値定数、変数または式です。

詳細

エラーが発生すると、FOPTNAME は空白を返します。

Windows 固有
使用可能な情報項目の数、値および種類は、動作環境やアクセス方式によって異なります。

サンプル

サンプル 1: ファイル情報項目の取得とログへの書き込み

この例では、システムに依存する使用可能なファイル情報項目を取得して、ログに書き込みます。

```
%let filrf=myfile;
%let rc=%sysfunc(filename(filrf,
physical-filename));
%let fid=%sysfunc(fopen(&filrf));
%let infonum=%sysfunc(foptnum(&fid));
%do j=1 %to &infonum;
%let name=%sysfunc(foptname(&fid,&j));
%let value=%sysfunc(finfo(&fid,&name));
%put File attribute &name equals &value;
%end;
%let rc=%sysfunc(fclose(&fid));
%let rc=%sysfunc(filename(filrf));
```

サンプル 2: ファイル属性の名前および値を含むデータセットの作成

この例では、使用可能なファイル属性の名前および値を含むデータセットを作成します。

```
data fileatt;
length name $ 20 value $ 40;
drop rc fid j infonum;
rc=filename("myfile","physical-filename");
fid=fopen("myfile");
infonum=foptnum(fid);
do j=1 to infonum;
name=foptname(fid,j);
value=finfo(fid,name);
put 'File attribute ' name
'has a value of ' value;
output;
end;
rc=filename("myfile");
run;
```

関連項目:

関数:

- “DINFO 関数” (371 ページ)
- “DOPTNAME 関数” (377 ページ)
- “DOPTNUM 関数” (378 ページ)

- “FCLOSE 関数” (394 ページ)
- “FILENAME 関数” (404 ページ)
- “FINFO 関数” (463 ページ)
- “FOPEN 関数” (476 ページ)
- “FOPTNUM 関数” (480 ページ)
- “MOPEN 関数” (653 ページ)

FOPTNUM 関数

外部ファイルで使用できる情報項目の数を返します。

カテゴリ: 外部ファイル

参照項目: “FOPTNUM Function: Windows” in *SAS Companion for Windows*
“FOPTNUM Function: UNIX” in *SAS Companion for UNIX Environments*
“FOPTNUM Function: z/OS” in *SAS Companion for z/OS*

構文

FOPTNUM(*file-id*)

必須引数

file-id
通常、FOPEN 関数によってファイルが開かれたときに割り当てられた識別子を指定する数値変数です。

詳細

Windows 固有
使用可能な情報項目の数、値および種類は、動作環境によって異なります。

比較

- FOPTNAME を使用して、特定の動作環境で使用できる項目の名前を決定します。
- FINFO を使用して、特定の情報項目の値を取得します。

サンプル

この例では、ファイル参照名 MYFILE を使用して外部ファイルを開き、システムに依存する使用可能なファイル情報項目の数を決定します。

```
%let fid=%sysfunc(fopen(myfile)); %let infonum=%sysfunc(foptnum(&fid));
```

関連項目:

関数:

- “DINFO 関数” (371 ページ)

- “DOPTNAME 関数” (377 ページ)
- “DOPTNUM 関数” (378 ページ)
- “FINFO 関数” (463 ページ)
- “FOPEN 関数” (476 ページ)
- “FOPTNAME 関数” (478 ページ)
- “MOPEN 関数” (653 ページ)

FPOINT 関数

次に読み込むレコードに読み込みポインタを配置します。

カテゴリ: 外部ファイル

構文

FPOINT(*file-id*,*note-id*)

必須引数

file-id
通常、FOPEN 関数によってファイルが開かれたときに割り当てられた識別子を指定する数値変数です。

note-id
FNOTE 関数で割り当てた識別子を指定します。

詳細

FPOINT は、操作が成功した場合は 0、失敗した場合は≠0 を返します。FPOINT は、次に読み込むレコードのみを決定します。次に書き込むレコードに影響はありません。更新のためにファイルを開くと、FWRITE は最後に読み込まれたレコードに書き込みます。

注: 新しいレコードが現在のレコードよりも長い場合、現在のレコードのかわりに新しいレコードを書き込むことはできません。

サンプル

この例では、ファイル参照名 MYFILE を外部ファイルに割り当ててそのファイルを開こうとします。ファイルが正常に開かれた場合、レコードを読み込み、NOTE3 を使用して 3 番目に読み込まれたレコードの位置を保存します。その後、NOTE3 にポインタを戻してファイルを更新し、ファイルを閉じます。

```
%let filrf=myfile;
%let rc=%sysfunc(filename(filrf,
physical-filename));
%let fid=%sysfunc(fopen(&filrf,u));
%if &fid > 0 %then
%do;
/* Read first record. */
%let rc=%sysfunc(fread(&fid));
/* Read second record. */
```

```
%let rc=%sysfunc(fread(&fid));
/* Read third record. */
%let rc=%sysfunc(fread(&fid));
/* Note position of third record. */
%let note3=%sysfunc(fnote(&fid));
/* Read fourth record. */
%let rc=%sysfunc(fread(&fid));
/* Read fifth record. */
%let rc=%sysfunc(fread(&fid));
/* Point to third record. */
%let rc=%sysfunc(fpoint(&fid,&note3));
/* Read third record. */
%let rc=%sysfunc(fread(&fid));
/* Copy new text to FDB. */
%let rc=%sysfunc(fput(&fid,New text));
/* Update third record */
/* with data in FDB. */
%let rc=%sysfunc(fwrite(&fid));
/* Close file. */
%let rc=%sysfunc(fclose(&fid));
%end;
%let rc=%sysfunc(filename(filrf));
```

関連項目:

関数:

- “DROPNOTE 関数” (380 ページ)
- “FCLOSE 関数” (394 ページ)
- “FILENAME 関数” (404 ページ)
- “FNOTE 関数” (474 ページ)
- “FOPEN 関数” (476 ページ)
- “FPUT 関数” (484 ページ)
- “FREAD 関数” (486 ページ)
- “FREWIND 関数” (487 ページ)
- “FWRITE 関数” (491 ページ)
- “MOPEN 関数” (653 ページ)

FPOS 関数

ファイルデータバッファ(FDB)の列ポインタの位置を設定します。

カテゴリ: 外部ファイル

構文

FPOS(*file-id*,*nval*)

必須引数

file-id
通常、FOPEN 関数によってファイルが開かれたときに割り当てられた識別子を指定する数値変数です。

nval
ポインタを設定する列を指定する数値定数、変数または式です。

詳細

FPOS は、操作が成功した場合は 0、失敗した場合は≠0 を返します。出力モードでファイルを開く場合、指定した位置が現在のレコードの最後を超えると、レコードのサイズが適切に増加します。ただし、固定ブロックまたは VBA ファイルの場合、レコードの最後を超える列位置を指定しても、レコードサイズは変更されず、テキスト文字列はファイルに書き込まれません。

更新モードでファイルを開く場合、指定した位置が現在のレコードの最後を超えていなければ、SAS はレコードをファイルに書き込みます。指定した位置が現在のレコードの最後を超えている場合、SAS はエラーメッセージを返し、新しいレコードを書き込みません。

```
ERROR: Cannot increase record length in update mode.
```

注: FOPEN 関数で更新モードを使用する場合、FWRITE 関数を実行する前に FREAD を実行する必要があります。

サンプル

この例では、ファイル参照名 MYFILE を既存のファイルに割り当てて、更新モードでファイルを開こうとします。変数 FID の正の値が示すように、ファイルが正常に開かれた場合、SAS はレコードを読み込み、ファイルのバッファの列 12 にデータを配置します。生成されるレコード長が元のレコード長以下の場合、SAS はレコードを書き込み、ファイルを閉じます。生成されるレコード長が元のレコード長よりも大きい場合、SAS はエラーメッセージをログに書き込みます。

```
%macro ptest;
%let filrf=myfile;
%let rc=%sysfunc(filename(filrf,test.txt));
%let fid=%sysfunc(fopen(&filrf,o));
%let rc=%sysfunc(fread(&fid));
%put &fid;
%if (&fid > 0) %then
%do;
%let dataline=This is some data.;
/* Position at column 12 in the FDB. */
%let rc=%sysfunc(fpos(&fid,12));
%put &rc one;
/* Put the data in the FDB. */
%let rc=%sysfunc(fput(&fid,&dataline));
%put &rc two;
%if (&rc ne 0) %then
%do;
%put %sysfunc(sysmsg());
%end;
%else %do;
/* Write the record. */
%let rc=%sysfunc(fwrite(&fid));
```

```
%if (&rc ne 0) %then
%do;
%put write fails &rc;
%end;
%end;
/* Close the file. */
%let rc=%sysfunc(fclose(&fid));
%end;
%let rc=%sysfunc(filename(filrf));
%mend;
%ptest;
```

ログ 2.9 *FPOS 関数からの出力*

```
1
0 one
0 two
```

関連項目:

関数:

- “FCLOSE 関数” (394 ページ)
- “FCOL 関数” (395 ページ)
- “FILENAME 関数” (404 ページ)
- “FOPEN 関数” (476 ページ)
- “FPUT 関数” (484 ページ)
- “FWRITE 関数” (491 ページ)
- “MOPEN 関数” (653 ページ)

FPUT 関数

外部ファイルのファイルデータバッファ(FDB)にデータを移動します。開始位置は FDB の現在の列位置になります。

カテゴリ: 外部ファイル

構文

FPUT(*file-id*,*cval*)

必須引数

file-id
通常、FOPEN 関数によってファイルが開かれたときに割り当てられた識別子を指定する数値変数です。

cval
ファイルデータを指定する文字定数、変数または式です。

詳細

FPUT は、操作が成功した場合は 0、失敗した場合は≠0 を返します。FDB に移動するバイト数は、変数の長さによって決まります。その後、列ポインタの値が増えて、新しいテキストの最後を 1 つ超えた位置になります。

注: 新しいレコードが現在のレコードよりも長い場合、現在のレコードのかわりに新しいレコードを書き込むことはできません。

サンプル

この例では、ファイル参照名 MYFILE を既存のファイルに割り当てて、APPEND モードでファイルを開こうとします。変数 FID の正の値が示すように、ファイルが正常に開かれた場合、FPUT を使用して FDB にデータを移動し、FWRITE を使用してレコードを追加した後、ファイルを閉じます。マクロステートメントでは、文字列を引用符で囲む必要はありません。

```
%macro ptest;
%let filrf=myfile;
%let rc=%sysfunc(filename(filrf,test.txt));
%let fid=%sysfunc(fopen(&filrf,a));
%if &fid > 0 %then
%do;
%let rc=%sysfunc(fread(&fid));
%let mystring=This is some data.;
%let rc=%sysfunc(fput(&fid,&mystring));
%let rc=%sysfunc(fwrite(&fid));
%let rc=%sysfunc(fclose(&fid));
%end;
%else
%put %sysfunc(sysmsg());
%let rc=%sysfunc(filename(filrf));
%put return code = &rc;
%mend;
%ptest;
```

SAS は次の出力をログに書き込みます。

```
return code = 0
```

関連項目:

関数:

- “FCLOSE 関数” (394 ページ)
- “FILENAME 関数” (404 ページ)
- “FNOTE 関数” (474 ページ)
- “FOPEN 関数” (476 ページ)
- “FPOINT 関数” (481 ページ)
- “FPOS 関数” (482 ページ)
- “FWRITE 関数” (491 ページ)
- “MOPEN 関数” (653 ページ)
- “SYSMSG 関数” (880 ページ)

FREAD 関数

外部ファイルのレコードをファイルデータバッファ(FDB)に読み込みます。

カテゴリ: 外部ファイル

構文

FREAD(*file-id*)

必須引数

file-id
通常、FOPEN 関数によってファイルが開かれたときに割り当てられた識別子を指定する数値変数です。

詳細

FREAD は、操作が成功した場合は 0、失敗した場合は≠0 を返します。ファイルポインタの位置は読み込み処理の完了後に自動的に更新されるため、一連の FREAD 関数でファイルレコードを連続して読み込むことができます。

明示的にファイルポインタを配置するには、FNOTE、FPOINT および FREWIND を使用します。

サンプル

この例では、ファイル参照名 MYFILE を外部ファイルに割り当ててそのファイルを開こうとします。ファイルが正常に開いた場合、すべてのファイルのレコードがログにリストされます。

```
%let filrf=myfile;
%let rc=%sysfunc(filename(filrf,
physical-filename));
%let fid=%sysfunc(fopen(&filrf));
%if &fid > 0 %then
%do %while(%sysfunc(fread(&fid)) = 0);
%let rc=%sysfunc(fget(&fid,c,200));
%put &c;
%end;
%let rc=%sysfunc(fclose(&fid));
%let rc=%sysfunc(filename(filrf));
```

関連項目:

関数:

- “FCLOSE 関数” (394 ページ)
- “FGET 関数” (401 ページ)
- “FILENAME 関数” (404 ページ)
- “FNOTE 関数” (474 ページ)
- “FOPEN 関数” (476 ページ)

- “FREWIND 関数” (487 ページ)
- “FREWIND 関数” (487 ページ)
- “MOPEN 関数” (653 ページ)

FREWIND 関数

ファイルの先頭にファイルポインタを配置します。

カテゴリ: 外部ファイル

構文

FREWIND(*file-id*)

必須引数

file-id
通常、FOPEN 関数によってファイルが開かれたときに割り当てられた識別子を指定する数値変数です。

詳細

FREWIND は、操作が成功した場合は 0、失敗した場合は≠0 を返します。
FREWIND は、シーケンシャルアクセスで開いているファイルには影響しません。

サンプル

この例では、ファイル参照名 MYFILE を外部ファイルに割り当てます。ファイルを開き、ファイルの最後に到達するまでレコードを読み込みます。その後、FREWIND 関数はファイルの先頭にポインタを再配置します。最初のレコードが再度読み込まれてファイルデータバッファ(FDB)に保存されます。最初のトークンが取得されてマクロ変数 VAL に保存されます。

```
%let filrf=myfile;
%let rc=%sysfunc(filename(filrf,
physical-filename));
%let fid=%sysfunc(fopen(&filrf));
%let rc=0;
%do %while (&rc ne -1);
/* Read a record. */
%let rc=%sysfunc(fread(&fid));
%end;
/* Reposition pointer to beginning of file. */
%if &rc = -1 %then
%do;
%let rc=%sysfunc(frewind(&fid));
/* Read first record. */
%let rc=%sysfunc(fread(&fid));
/* Read first token */
/* into macro variable VAL. */
%let rc=%sysfunc(fget(&fid,val));
%put val=&val;
```

```
%end;
%else
%put Error on fread=%sysfunc(sysmsg());
%let rc=%sysfunc(fclose(&fid));
%let rc=%sysfunc(filename(filrf));
```

関連項目:

関数:

- “FCLOSE 関数” (394 ページ)
- “FGET 関数” (401 ページ)
- “FILENAME 関数” (404 ページ)
- “FOPEN 関数” (476 ページ)
- “FREAD 関数” (486 ページ)
- “MOPEN 関数” (653 ページ)
- “SYSMSG 関数” (880 ページ)

FRLEN 関数

最後に読み込まれたレコードのサイズを返します。出力のためにファイルが開かれている場合は、現在のレコードサイズを返します。

カテゴリ: 外部ファイル

構文

FRLEN(*file-id*)

必須引数

file-id
通常、FOPEN 関数によってファイルが開かれたときに割り当てられた識別子を指定する数値変数です。

サンプル

この例では、ファイル参照名 MYFILE で識別されるファイルを開きます。外部ファイルの最小レコード長および最大レコード長を決定し、結果を口グに書き込みます。

```
%let fid=%sysfunc(fopen(myfile));
%let min=0;
%let max=0;
%if (%sysfunc(fread(&fid)) = 0) %then
%do;
%let min=%sysfunc(frlen(&fid));
%let max=&min;
%do %while(%sysfunc(fread(&fid)) = 0);
%let reclen=%sysfunc(frlen(&fid));
```

```
%if (&reclen > &max) %then
%let max=&reclen;
%if (&reclen < &min) %then
%let min=&reclen;
%end;
%end;
%let rc=%sysfunc(fclose(&fid));
%put max=&max min=&min;
```

関連項目:

関数:

- “FCLOSE 関数” (394 ページ)
- “FOPEN 関数” (476 ページ)
- “FREAD 関数” (486 ページ)
- “MOPEN 関数” (653 ページ)

FSEP 関数

FGET 関数のトークン区切り文字を設定します。

カテゴリ: 外部ファイル

構文

FSEP(*file-id*,*characters*<,'x' | 'X'>)

必須引数

file-id
通常、FOPEN 関数によってファイルが開かれたときに割り当てられた識別子を指定する数値変数です。

character
ファイルデータバッファ(FDB)の項目を区切る 1 つ以上の区切り文字を指定する文字定数、変数または式です。リストされる各文字が区切り文字になります。つまり、*character* が#@の場合、#または@で項目が区切られます。複数の連続する区切り文字(@#@など)は、1 つの区切り文字として扱われます。

デフォルト: 空白

オプション引数

'x' | 'X'
文字の区切り文字が 16 進値であることを示します。

制限事項:
この引数で有効な値は、'x'および'X'のみです。その他のすべての値では エラーが発生します。

'x'または'X'を第 3 引数として渡す場合、有効な 16 進文字列を第 2 引数 *character* として渡す必要があります。そうしないと、関数が失敗します。有効な 16 進文字列は、0–9 の偶数および A–F の文字です。

ヒント: マクロステートメントを使用する場合、x または X を囲む引用符は必要ありません。

詳細

FSEP は、操作が成功した場合は 0、失敗した場合は≠0 を返します。

サンプル

外部ファイルのデータは次のような形式になります。

```
John J. Doe,Male,25,Weight Lifter
Pat O'Neal,Female,22,Gymnast
```

各フィールドはカンマで区切られています。

この例では、カンマを区切り文字として使用して、ファイル参照名 MYFILE で識別されるファイルを読み込み、NAME、GENDER、AGE、WORK の値を SAS ログに書き込みます。マクロステートメントでは文字列を引用符で囲みませんが、関数の引数のリテラルカンマはマクロの引用符関数(%STR など)で囲む必要があります。

```
%let fid=%sysfunc(fopen(myfile));
%let rc=%sysfunc(fsep(&fid,%str(,)));
%do %while(%sysfunc(fread(&fid)) = 0);
%let rc=%sysfunc(fget(&fid,name));
%let rc=%sysfunc(fget(&fid,gender));
%let rc=%sysfunc(fget(&fid,age));
%let rc=%sysfunc(fget(&fid,work));
%put name=%bquote(&name) gender=&gender
age=&age work=&work;
%end;
%let rc=%sysfunc(fclose(&fid));
```

関連項目:

関数:

- “FCLOSE 関数” (394 ページ)
- “FGET 関数” (401 ページ)
- “FOPEN 関数” (476 ページ)
- “FREAD 関数” (486 ページ)
- “MOPEN 関数” (653 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: 切り捨て関数

構文

FUZZ(*argument*)

必須引数

引数
数値の定数、変数または式を指定します。

詳細

引数が整数の 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、その引数を返します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果

var1=5.9999999999999;
x=fuzz(var1);
put x 16.14
	6.000000000000000

x=fuzz(5.99999999); put x 16.14;	5.999999990000000

FWRITE 関数

外部ファイルにレコードを書き込みます。

カテゴリ: 外部ファイル

構文

FWRITE(*file-id*<,*cc*>)

必須引数

file-id
通常、FOPEN 関数によってファイルが開かれたときに割り当てられた識別子を指定する数値変数です。

オプション引数

cc
改行制御文字を指定する文字定数、変数または式です。

空白	新しい行でレコードを開始します。
0	新しい行の前に 1 つの空白行をスキップします。
-	新しい行の前に 2 つの空白行をスキップします。
1	新しいページで行を開始します。
+	前の行を上書きします。

P	コンピュータプロンプトとして行を解釈します。
=	改行制御情報として行を解釈します。
その他すべて	新しい行で行レコードを開始します。

詳細

FWRITE は、操作が成功した場合は 0、失敗した場合は≠0 を返します。FWRITE は、ファイルデータバッファ(FDB)のテキストを外部ファイルに移動します。改行制御文字を使用するには、FOPEN でレコード形式として **P** (印刷形式)を指定してファイルを開く必要があります。

注: 更新モードを使用する場合、FWRITE を実行する前に FREAD を実行する必要があります。新しいレコードが現在のレコードよりも長い場合、現在のレコードのかわりに新しいレコードを書き込むことはできません。

サンプル

この例では、ファイル参照名 MYFILE を外部ファイルに割り当ててそのファイルを開こうとします。ファイルが正常に開いた場合、2 つの空白行をスキップして外部ファイルに 1 ~ 50 の番号を書き込みます。マクロステートメントでは、文字列を引用符で囲む必要はありません。

```
%let filrf=myfile;
%let rc=%sysfunc(filename(filrf,
physical-filename));
%let fid=%sysfunc(fopen(&filrf,o,0,P));
%do i=1 %to 50;
%let rc=%sysfunc(fput(&fid,
%sysfunc(putn(&i,2.))));
%if (%sysfunc(fwrite(&fid,-)) ne 0) %then
%put %sysfunc(sysmsg());
%end;
%let rc=%sysfunc(fclose(&fid));
```

関連項目:

関数:

- “FAPPEND 関数” (393 ページ)
- “FCLOSE 関数” (394 ページ)
- “FGET 関数” (401 ページ)
- “FILENAME 関数” (404 ページ)
- “FOPEN 関数” (476 ページ)
- “FPUT 関数” (484 ページ)
- “SYSMSG 関数” (880 ページ)

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

構文

GAMINV(*p*,*a*)

必須引数

p
数値の確率です。
範囲: $0 \leq p < 1$

a
数値の形状パラメータです。
範囲: $a > 0$

詳細

GAMINV 関数は、ガンマ分布(形状パラメータは *a*)から *p*番目の分位点を返します。ガンマ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: GAMINV は、PROBGAM 関数の逆数です。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
q1=gaminv(0.5,9);	8.6689511844
q2=gaminv(0.1,2.1);	0.5841932369

関連項目:

関数:

- “QUANTILE 関数” (778 ページ)

GAMMA 関数

ガンマ関数の値を返します。

カテゴリ: 数学関数

構文

GAMMA(*argument*)

必須引数

引数
数値の定数、変数または式を指定します。
制限事項: 正でない整数は無効です。

詳細

GAMMA 関数は、次の式で得られる積分を返します。

$$GAMMA(x) = \int_0^{\infty} t^{x-1} \varepsilon^{-t}\, dt.$$

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。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=gamma(6);	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: 財務関数

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。

t
満期までの時間を指定する非欠損値。

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。

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。

***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。

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 *t* の単位として同期間の *sigma* の値を指定します。

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。

$$CALL = SN(d_1)\left(\varepsilon^{-R_f t}\right) - EN(d_2)\left(\varepsilon^{-R_d t}\right)$$

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
失効日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。

$$CALL = \max((S - E), 0)$$

価格設定の基本については、“価格関数の使用” (8 ページ)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値を返します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
	----+----1----+---2--
a=garkhclprc(1000, .5, 950, 4, 4, 2); put a;	65.335687119
b=garkhclprc(850, 1.2, 125, 5, 3, 1); put b;	1.9002767538
c=garkhclprc(7500, .9, 950, 3, 2, 2); put c;	69.328647279
d=garkhclprc(5000, -.5, 237, 3, 3, 2); put d;	0

関連項目:

関数:

- “GARKHPTPRC 関数” (496 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: 財務関数

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。

t
満期までの時間を指定する非欠損値。

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。

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。

***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。

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 *t* の単位として同期間の *sigma* の値を指定します。

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。

$$PUT = CALL - S\left(\varepsilon^{-R_f t}\right) + E\left(\varepsilon^{-R_d t}\right)$$

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
失効日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。

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

価格設定の基本については、“価格関数の使用” (8 ページ)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値を返します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
	----+----1----+---2--
a=garkhptprc(1000, .5, 950, 4, 4, 2);	72.102451281
b=garkhptprc(850, 1.2, 125, 5, 3, 1);	0.5917507981
c=garkhptprc(7500, .9, 950, 3, 2, 2);	416.33604902
d=garkhptprc(5000, -.5, 237, 3, 3, 2);	0

関連項目:

関数:

- “GARKHCLPRC 関数” (494 ページ)

GCD 関数

1 つ以上の整数の最大公約数を返します。

カテゴリ: 数学関数

構文

GCD(*x1, x2, x3, ..., xn*)

必須引数

x
整数値の数値定数、変数または式を指定します。

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。

いずれかの引数がない場合、戻り値は欠損値になります。

サンプル

次の例では、整数 10 と 15 の最大公約数を返します。

```
data _null_;
x=gcd(10, 15);
put x=;
run;
```

SAS は次の出力を ログに書き込みます。

```
x=5
```

関連項目:

関数:

- “LCM 関数” (599 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

構文

GEODIST(*latitude-1*, *longitude-1*, *latitude-2*, *longitude-2* <,*options*>)

必須引数

緯度

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 ~ –90 にする必要があります。値をラジアンで表す場合、pi/2 ~ –pi/2 にする必要があります。

経度

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、180 ~ –180 にする必要があります。値をラジアンで表す場合、pi ~ –pi にする必要があります。

オプション引数

option

次のいずれかの文字を含む文字定数、変数または式を指定します。

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。

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。

サンプル

サンプル 1: 測地距離(キロメートル単位)の計算

次の例では、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。

```
data _null_;
distance=geodist(30.68, -88.25, 35.43, -82.55);
put 'Distance= ' distance 'kilometers';
run;
```

SAS は次の出力をログに書き込みます。

```
Distance= 748.6529147 kilometers
```

サンプル 2: 測地距離(マイル単位)の計算

次の例では、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。

```
data _null_;
distance=geodist(30.68, -88.25, 35.43, -82.55, 'M');
put 'Distance = ' distance 'miles';
run;
```

SAS は次の出力をログに書き込みます。

```
Distance = 465.29081088 miles
```

サンプル 3: 度数単位の入力を使用した測地距離の計算

次の例では、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。

```
data _null_;
input lat1 long1 lat2 long2;
Distance = geodist(lat1,long1,lat2,long2,'DM');
put 'Distance = ' Distance 'miles';
datalines;
35.2 -78.1 37.6 -79.8
;
run;
```

SAS は次の出力をログに書き込みます。

```
Distance = 190.72474282 miles
```

サンプル 4: ラジアン単位の入力を使用した測地距離の計算

次の例では、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。

```
data _null_;
input lat1 long1 lat2 long2;
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';
datalines;
35.2 -78.1 37.6 -79.8
;
run;
```

SAS は次の出力を ログに書き込みます。

```
Distance= 190.72474282 miles
```

リファレンス

Vincenty, T. “Direct and Inverse Solutions of Geodesics on the Ellipsoid with Application of Nested Equations.” 1975. *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

構文

GEOMEAN(*argument*<,*argument*,…>)

必須引数

引数

負でない数値定数、変数または式です。

ヒント: 引数リストには OF で始まる変数のリストを含められます。

詳細

いずれかの引数が負の場合、結果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
ERROR_
が1 に設定されます。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。すべての引数が欠

n は、非欠損値の引数の数で、$x_1, x_2, \ldots, x_n$ は、それらの引数の値です。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、幾何平均は次のようになります。

$$\exp\left(\frac{(\log(x_1) + \log(x_2) + \ldots + \log(x_n))}{n}\right)$$

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。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x1=geomean(1,2,2,4);	2
x2=geomean(.,2,4,8);	4
x3=geomean(of x1-x2);	2.8284271247

関連項目:

関数:

- “GEOMEANZ 関数” (502 ページ)
- “HARMEAN 関数” (515 ページ)
- “HARMEANZ 関数” (516 ページ)
- “MEAN 関数” (642 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

構文

GEOMEANZ(*argument*<,*argument*,…>)

必須引数

引数

負でない数値定数、変数または式です。

ヒント: 引数リストには OF で始まる変数のリストを含められます。

詳細

いずれかの引数が負の場合、結果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
ERROR_
が1 に設定されます。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。すべての引数が欠

n は、非欠損値の引数の数で、$x_1, x_2, \ldots, x_n$ は、それらの引数の値です。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、幾何平均は次のようになります。

$$\exp\left(\frac{(\log(x_1) + \log(x_2) + \ldots + \log(x_n))}{n}\right)$$

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。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x1=geomeanz(1,2,2,4);	2
x2=geomeanz(.,2,4,8);	4
x3=geomeanz(of x1-x2);	2.8284271247

関連項目:

関数:

- “GEOMEAN 関数” (501 ページ)
- “HARMEAN 関数” (515 ページ)
- “HARMEANZ 関数” (516 ページ)
- “MEAN 関数” (642 ページ)

GETOPTION Function

Returns the value of a SAS system or graphics option.

Category: Special

Syntax

GETOPTION(*option-name*<, *return-value-option*><*return-value-formatting-options*>)

Required Argument

option-name
is a character constant, variable, or expression that specifies the name of the system option.

Tips:
Do not put an equal sign after the name. For example, write PAGESIZE= as PAGESIZE.

SAS options that are passwords, such as EMAILPW and METAPASS, return the value **xxxxxxxx**, and not the actual password.

Return Value Options

DEFAULTVALUE
returns the default option value.

Restriction: DEFAULTVALUE is valid only for SAS system options. SAS issues a warning message when the DEFAULTVALUE option is specified and *option-name* is a graphics option.

HOWSCOPE
returns a character string that specifies the scope of an option.

Restriction: HOWSCOPE is valid only for SAS system options. SAS issues a warning message when the HOWSCOPE option is specified and *option-name* is a graphics option.

HOWSET
returns a character string that specifies how an option value was set.

Restriction: HOWSET is valid only for SAS system options. SAS issues a warning message when the HOWSET option is specified and *option-name* is a graphics option.

STARTUPVALUE
returns the system option value that was used to start SAS either on the command line or in a configuration file.

Restriction: STARTUPVALUE is valid only for SAS system options. SAS issues a warning message when the STARTUPVALUE option is specified and *option-name* is a graphics option.

Return Value Formatting Options

CM
reports graphic units of measure in centimeters.

Restriction: CM is valid only for graphics options and the following SAS system options: BOTTOMMARGIN, TOPMARGIN, RIGHTMARGIN, and LEFTMARGIN. SAS writes a note to the log when the CM option is specified and *option-name* is not a graphics option or an option that specifies a margin value.

EXPAND
for options that contain environment variables, returns the option value with the value of the environment variable.

Restrictions:

Variable expansion is valid only in the Windows and UNIX operating environments.

EXPAND is valid only for character system option values. EXPAND is ignored if *option-name* has an option type of Boolean, such as CENTER or NOCENTER, or if the value of the option is numeric.

Note: SAS issues a note when EXPAND is specified for Boolean options and for options that have numeric values. SAS issues a warning when EXPAND is specified and the option is a graphics option.

Tip: By default, some option values are displayed with expanded variable values. Other options require the EXPAND option in the PROC OPTIONS statement. Use the DEFINE option in the PROC OPTIONS statement to determine whether an option value expands variables by default or if the EXPAND option is required. If the output from PROC OPTIONS DEFINE shows the following information, you must use the EXPAND option to expand variable values:

```
Expansion: Environment variables, within the option value, are not expanded
```

KEYEXPAND

for options that contain environment variables, returns the value in the format **option-name=value**.

Restriction: KEYEXPAND is valid only for character system option values. SAS issues an error message when the KEYEXPAND option is specified and *option-name* is a graphics option. KEYEXPAND is ignored if *option-name* has an option type of Boolean, such as CENTER or NOCENTER, or if the value of the option is numeric.

KEYWORD

returns option values in a **option-name=value** format that would be suitable for direct use in the SAS OPTIONS or GOPTIONS global statements.

Restrictions:

KEYWORD is not valid when it is used with the HEXVALUE, EXPAND, KEYEXPAND, or LOGNUMBERFORMAT options. SAS writes a note to the log when the GETOPTION function contains conflicting options.

KEYWORD is valid only for character or numeric system option values. KEYWORD is ignored for system options whose option type is Boolean, such as CENTER or NOCENTER. SAS issues an error message when the KEYWORD option is specified and *option-name* is a graphics option.

Note: For a system option with a null value, the GETOPTION function returns a value of ' ' (single quotation marks with a blank space between them). An example is EMAILID=' '.

HEXVALUE

returns the option value as a hexadecimal value.

Restriction: HEXVALUE is valid only for character or numeric system option values. If HEXVALUE is specified for system options whose option type is Boolean, such as CENTER or NOCENTER, or if *option-name* is a graphics option, SAS issues an error message.

IN

reports graphic units of measure in inches.

Restriction: IN is valid only for graphics options and the following SAS system options: BOTTOMMARGIN, TOPMARGIN, RIGHTMARGIN, and LEFTMARGIN. SAS writes a note to the log when the IN option is specified and *option-name* is not a graphics option or an option that specifies a margin value.

LOGNUMBERFORMAT

formats SAS system option values using locale-specific punctuation.

Restriction: Do not use LOGNUMBERFORMAT if the returned value is used to set an option value by using the OPTIONS statement. The OPTIONS statement does not accept commas in numeric values.

Examples

Example 1: Using GETOPTION to Save and Restore the YEARCUTOFF Option

This example saves the value of the YEARCUTOFF option, processes SAS statements based on the value of the YEARCUTOFF option, and then resets the value to 1920 if it is not already 1920.

```
/* Save the value of the YEARCUTOFF system option */
%let cutoff=%sysfunc(getoption(yearcutoff,keyword));

data ages;
  if getoption('yearcutoff') = '1920' then
     do;
       ...more SAS statements...
     end;
     else do;
           ...more SAS statements...
           /* Reset YEARCUTOFF   */
           options &cutoff;
         end;
run;
```

Example 2: Using GETOPTION to Obtain Different Reporting Options

This example defines a macro to illustrate the use of the GETOPTION function to obtain the value of system and graphics options by using different reporting options.

```
%macro showopts;
  %put MAPS= %sysfunc(
     getoption(MAPS));
  %put MAPSEXPANDED= %sysfunc(
     getoption(MAPS, EXPAND));
  %put PAGESIZE= %sysfunc(
     getoption(PAGESIZE));
  %put PAGESIZESETBY= %sysfunc(
     getoption(PAGESIZE, HOWSET));
  %put PAGESIZESCOPE= %sysfunc(
     getoption(PAGESIZE, HOWSCOPE));
  %put PS= %sysfunc(
     getoption(PS));
  %put LS= %sysfunc(
     getoption(LS));
  %put PS(keyword form)= %sysfunc(
     getoption(PS,keyword));
  %put LS(keyword form)= %sysfunc(
     getoption(LS,keyword));
  %put FORMCHAR= %sysfunc(
     getoption(FORMCHAR));
  %put HSIZE= %sysfunc(
```

```
    getoption(HSIZE));
  %put VSIZE= %sysfunc(
    getoption(VSIZE));
  %put HSIZE(in/keyword form)= %sysfunc(
    getoption(HSIZE,in,keyword));
  %put HSIZE(cm/keyword form)= %sysfunc(
    getoption(HSIZE,cm,keyword));
  %put VSIZE(in/keyword form)= %sysfunc(
    getoption(VSIZE,in,keyword));
  %put HSIZE(cm/keyword form)= %sysfunc(
    getoption(VSIZE,cm,keyword));
%mend;
goptions VSIZE=8.5 in HSIZE=11 in;
options PAGESIZE=67;
%showopts
```

The following is the SAS log:

```
NOTE: PROCEDURE PRINTTO used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds

6    %macro showopts;
7      %put MAPS= %sysfunc(
8          getoption(MAPS));
9      %put MAPSEXPANDED= %sysfunc(
10         getoption(MAPS, EXPAND));
11     %put PAGESIZE= %sysfunc(
12         getoption(PAGESIZE));
13     %put PAGESIZESETBY= %sysfunc(
14         getoption(PAGESIZE, HOWSET));
15     %put PAGESIZESCOPE= %sysfunc(
16         getoption(PAGESIZE, HOWSCOPE));
17     %put PS= %sysfunc(
18         getoption(PS));
19     %put LS= %sysfunc(
20         getoption(LS));
21     %put PS(keyword form)= %sysfunc(
22         getoption(PS,keyword));
23     %put LS(keyword form)= %sysfunc(
24         getoption(LS,keyword));
25     %put FORMCHAR= %sysfunc(
26         getoption(FORMCHAR));
27     %put HSIZE= %sysfunc(
28         getoption(HSIZE));
29     %put VSIZE= %sysfunc(
30         getoption(VSIZE));
31     %put HSIZE(in/keyword form)= %sysfunc(
32         getoption(HSIZE,in,keyword));
33     %put HSIZE(cm/keyword form)= %sysfunc(
34         getoption(HSIZE,cm,keyword));
35     %put VSIZE(in/keyword form)= %sysfunc(
36         getoption(VSIZE,in,keyword));
37     %put HSIZE(cm/keyword form)= %sysfunc(
38         getoption(VSIZE,cm,keyword));
39   %mend;
40   goptions VSIZE=8.5 in HSIZE=11 in;
41   options PAGESIZE=67;
42   %showopts
MAPS= ("!sasroot¥maps-path¥en¥maps")
MAPSEXPANDED= ("C:¥maps-path¥en¥maps")
PAGESIZE= 67
PAGESIZESETBY= Options Statement
PAGESIZESCOPE= Line Mode Process
PS= 67
LS= 78
PS(keyword form)= PS=67
LS(keyword form)= LS=78
FORMCHAR= ,ƒ„…†‡ˆ‰Š‹Œ+=|-/¥<>*
HSIZE= 11.0000 in
VSIZE= 8.5000 in
HSIZE(in/keyword form)= HSIZE=11.0000 in
HSIZE(cm/keyword form)= HSIZE=27.9400 cm
VSIZE(in/keyword form)= VSIZE=8.5000 in
HSIZE(cm/keyword form)= VSIZE=21.5900 cm
43   proc printto; run;
```

Example 3: Returning Default and Start-up Values

This example changes the value of the PAPERSIZE system option to a specific value, the PAPERSIZE option default value, and to the value that was assigned to the PAPERSIZE option when SAS started.

```
/* Check the value of papersize before we change it.       */
/* The initial value is A4 as this value was used when      */
/* SAS started.                                            */

  %put %sysfunc(getoption(papersize,keyword));

/* Change the PAPERSIZE value and check the change.        */

  options papersize="600x800 Pixels";

  %put %sysfunc(getoption(papersize,keyword));

/* Change PAPERSIZE back to the default value and check it.  */
/* RESULT:  LETTER                                          */

  %let defsize = %sysfunc(getoption(papersize,keyword,defaultvalue)) ;
  options &defsize; run;
  %put %sysfunc(getoption(papersize,keyword));

/* Change the value to the startup value and check it.      */
/* RESULT:  A4                                              */

  %let defsize = %sysfunc(getoption(papersize,keyword,startupvalue)) ;
  options &defsize; run;
  %put %sysfunc(getoption(papersize,keyword));
```

The SAS log displays the following lines:

```
22   /* Check the value of papersize before we change it.       */
23   /* The initial value is A4 as this value was used when      */
24   /* SAS started.                                            */
25
26      %put %sysfunc(getoption(papersize,keyword));
PAPERSIZE=A4
27
28   /* Change the PAPERSIZE value and check the change.        */
29
30      options papersize="600x800 Pixels";
31
32      %put %sysfunc(getoption(papersize,keyword));
PAPERSIZE=600X800 PIXELS
33
34   /* Change PAPERSIZE back to the default value and check it.  */
35   /* RESULT:  LETTER                                          */
36
37      %let defsize = %sysfunc(getoption(papersize,keyword,defaultvalue)) ;
38      options &defsize; run;
39      %put %sysfunc(getoption(papersize,keyword));
PAPERSIZE=LETTER
40
41   /* Change the value to the startup value and check it.      */
42   /* RESULT:  A4                                              */
43
44      %let defsize = %sysfunc(getoption(papersize,keyword,startupvalue)) ;
45      options &defsize; run;
46      %put %sysfunc(getoption(papersize,keyword));
PAPERSIZE=A4
```

Note: The default settings for the PAGESIZE= and the LINESIZE= options depend on the mode that you use to run SAS.

GETVARC 関数

SAS データセットの文字変数の値を返します。

カテゴリ: SAS ファイル I/O 関数

構文

GETVARC(*data-set-id*,*var-num*)

必須引数

data-set-id
OPEN 関数が返すデータセット識別子を指定する数値定数、変数または式です。

var-num
データセットデータベクトル(DDV)の変数の数を指定する数値定数、変数または式です。

ヒント:
この値は、VARNUM 関数を使用して取得できます。
CONTENTS プロシジャを使用すると、この値は変数の横にリストされます。

詳細

VARNUM を使用して、SAS データセットの変数の数を取得します。VARNUM は、次の例のようにネストしたり、変数に割り当てて第 2 引数として渡したりできます。GETVARC は、データセットデータベクトル(DDV)の現在のオブザベーションからマクロまたは DATA ステップの変数に文字変数値を読み込みます。

サンプル

- この例では、SASUSER.HOUSES データセットを開き、10 番目のオブザベーション全体を取得します。開いているデータセットのデータセット識別子の値は、マクロ変数 MYDATAID に保存されます。この例では、VARNUM をネストして DDV の変数の位置を返し、文字変数 STYLE の値を読み込みます。

```
%let mydataid=%sysfunc(open
(sasuser.houses,i));
%let rc=%sysfunc(fetchobs(&mydataid,10));
%let style=%sysfunc(getvarc(&mydataid,
%sysfunc(varnum
(&mydataid,STYLE))));
%let rc=%sysfunc(close(&mydataid));
```

- この例では、第 2 引数として渡すことができる変数に VARNUM を割り当てます。この例では、オブザベーション 10 のデータをフェッチします。

```
%let namenum=%sysfunc(varnum(&mydataid,NAME));
%let rc=%sysfunc(fetchobs(&mydataid,10));
%let user=%sysfunc(getvarc
(&mydataid,&namenum));
```

関連項目:

関数:

- “FETCH 関数” (398 ページ)
- “FETCHOBS 関数” (399 ページ)
- “GETVARN 関数” (511 ページ)
- “VARNUM 関数” (914 ページ)

GETVARN 関数

SAS データセットの数値変数の値を返します。

カテゴリ: SAS ファイル I/O 関数

構文

GETVARN(*data-set-id*,*var-num*)

必須引数

data-set-id
OPEN 関数が返すデータセット識別子を指定する数値定数、変数または式です。

var-num
データセットデータベクトル(DDV)の変数の数を指定する数値定数、変数または式です。

ヒント:
この値は、VARNUM 関数を使用して取得できます。
CONTENTS プロシジャを使用すると、この値は変数の横にリストされます。

詳細

VARNUM を使用して、SAS データセットの変数の数を取得します。VARNUM は、"例"セクションのようにネストしたり、変数に割り当てて第 2 引数として渡したりできます。GETVARN は、データセットデータベクトル(DDV)の現在のオブザベーションからマクロまたは DATA ステップの変数に数値変数値を読み込みます。

サンプル

- この例では、SAS データセットの 10 番目のオブザベーション全体を取得します。データセットは、OPEN を使用して事前に開いておく必要があります。開いているデータセットのデータセット識別子の値は、変数 MYDATAID に保存されます。この例では、VARNUM をネストし、開いている SAS データセットの 10 番目のオブザベーションから数値変数 PRICE の値を読み込みます。

```
%let rc=%sysfunc(fetchobs(&mydataid,10));
%let price=%sysfunc(getvarn(&mydataid,
```

```
%sysfunc(varnum
(&mydataid,price))));
```

- この例では、第 2 引数として渡すことができる変数に VARNUM を割り当てます。この例では、オブザベーション 10 のデータをフェッチします。

```
%let pricenum=%sysfunc(varnum
(&mydataid,price));
%let rc=%sysfunc(fetchobs(&mydataid,10));
%let price=%sysfunc(getvarn
(&mydataid,&pricenum));
```

関連項目:

関数:

- “FETCH 関数” (398 ページ)
- “FETCHOBS 関数” (399 ページ)
- “GETVARC 関数” (510 ページ)
- “VARNUM 関数” (914 ページ)

GRAYCODE 関数

n 個の項目のすべてのサブセットを変化量の小さい順に生成します。

カテゴリ: 組み合わせ関数

制限事項: GRAYCODE 関数は、%SYSFUNC マクロを使用する場合には実行できません。

構文

GRAYCODE(*k*, *numeric-variable-1*, ..., *numeric-variable-n*)

GRAYCODE(*k*, *character-variable* <, *n*<, *in-out*>>)

必須引数

k

数値変数を指定します。GRAYCODE 関数を実行する前に *k* を次のいずれかの値に初期化します。

- 負の数(GRAYCODE によってサブセットが空に初期化されます)
- *numeric-variable-1* ~ *numeric-variable-n* または *character-variable* で指定する初期セットの項目数(0 ~ *n* (0 と n を含む)の整数値である必要があります)

k の値は、GRAYCODE の実行時に更新されます。サブセットの項目数が値として返されます。

numeric-variable

値 0 または値 1 の数値変数を指定します。この値は GRAYCODE の実行時に更新されます。*numeric-variable-j* の値が 1 の場合、*j*番目の項目がサブセットにあることを表します。*numeric-variable-j* の値が 0 の場合、*j*番目の項目がサブセットにないことを表します。

GRAYCODE を実行する前に負の値を *k* に割り当てる場合、GRAYCODE を実行する前に *numeric-variable-1* ~ *numeric-variable-n* を初期化する必要はありません。ただし、非初期化変数に関するメモを非表示にする場合は除きます。

GRAYCODE を実行する前に 0 ~ *n* (0 と n を含む)の値を *k* に割り当てる場合、*numeric-variable-1* ~ *numeric-variable-n* を *k* 個の値 1 と *n-k* 個の値 0 に初期化する必要があります。

character-variable
長さが少なくとも *n* 文字の文字変数を指定します。先頭の *n* 文字でどの項目がサブセットにあるかを示します。デフォルトでは、*j*番目の位置にある"I"は *j*番目の項目がサブセットにあり、*j*番目の位置にある"O"は *j*番目の項目がサブセットにないことを示します。これらの 2 文字は *in-out* 引数の指定で変更できます。

GRAYCODE を実行する前に負の値を *k* に割り当てる場合、GRAYCODE を実行する前に *character-variable* を初期化する必要はありません。ただし、非初期化変数に関するメモを非表示にする場合は除きます。

GRAYCODE を実行する前に 0 ~ *n* (0 と n を含む)の値を *k* に割り当てる場合、*character-variable* を、項目がサブセットに含まれていることを示す *k* 個の文字と項目がサブセットに含まれていないことを示す *n-k* 個の文字に初期化する必要があります。

オプション引数

n
数値の定数、変数または式を指定します。デフォルトでは、*n* は *character-variable* の長さです。

in-out
文字定数、変数または式を指定します。デフォルト値は"IO"です。1 番目の文字は項目がサブセットにあることを示すのに使用されます。2 番目の文字は項目がサブセットにないことを示すのに使用されます。

詳細

k に負の値を割り当てて GRAYCODE を実行すると、サブセットが空に初期化されます。GRAYCODE 関数はゼロを返します。

k に 0 ~ *n* (0 と n を含む)の整数値を割り当てて GRAYCODE を実行すると、サブセットの項目が 1 つ追加または削除され、サブセットの項目数と同じになるように *k* の値が更新されます。サブセットの *j*番目の項目が追加または削除されると、GRAYCODE 関数は *j* を返します。

n 項目のすべてのサブセットを生成するには、*k* を負の値に初期化して、2**n* 回繰り返すループで GRAYCODE を実行します。空でないサブセットで開始するには、*k* をサブセットの項目数に初期化して、目的の初期サブセットを示すように他の引数を初期化し、2**n*-1 回繰り返すループで GRAYCODE を実行します。GRAYCODE で生成される一連のサブセットは循環するため、どのサブセットで開始してもかまいません。

サンプル

サンプル 1: 数値変数(n=4)と k (初期値は負)を使用する

次のプログラムでは、数値変数を使用して変更の少ない順にサブセットを生成します。

```
data _null_;
array x[4];
n=dim(x);
k=-1;
nsubs=2**n;
do i=1 to nsubs;
rc=graycode(k, of x[*]);
put i 5. +3 k= ' x=' x[*] +3 rc=;
end;
run;
```

SAS は次の出力を口グに書き込みます。

```
 1 k=0 x=0 0 0 0 rc=0
2 k=1 x=1 0 0 0 rc=1
3 k=2 x=1 1 0 0 rc=2
4 k=1 x=0 1 0 0 rc=1
5 k=2 x=0 1 1 0 rc=3
6 k=3 x=1 1 1 0 rc=1
7 k=2 x=1 0 1 0 rc=2
8 k=1 x=0 0 1 0 rc=1
9 k=2 x=0 0 1 1 rc=4
10 k=3 x=1 0 1 1 rc=1
11 k=4 x=1 1 1 1 rc=2
12 k=3 x=0 1 1 1 rc=1
13 k=2 x=0 1 0 1 rc=3
14 k=3 x=1 1 0 1 rc=1
15 k=2 x=1 0 0 1 rc=2
16 k=1 x=0 0 0 1 rc=1
```

サンプル 2: 文字変数と k (初期値は正) を使用する

次の例では、文字変数を使用して変更の少ない順にサブセットを生成します。

```
data _null_;
x='++++';
n=length(x);
k=countc(x, '+');
put ' 1' +3 k= +2 x=;
nsubs=2**n;
do i=2 to nsubs;
rc=graycode(k, x, n, '+-');
put i 5. +3 k= +2 x= +3 rc=;
end;
run;
```

SAS は次の出力を口グに書き込みます。

```
 1 k=4 x=++++ 2 k=3 x=-+++ rc=1 3 k=2 x=-+-+ rc=3 4 k=3 x=+++-+ rc=1 5 k=2 x=+---+ rc=2 6 k=1 x=----+ rc=1 7 k=0 x=----- rc=4 8
```

関連項目:

CALL ルーチン:

- “CALL GRAYCODE ルーチン” (165 ページ)

HARMEAN 関数

調和平均を返します。

カテゴリ: 記述統計

構文

HARMEAN(*argument*<,*argument*,…>)

必須引数

引数

負でない数値定数、変数または式です。

ヒント: 引数リストには OF で始まる変数のリストを含められます。

詳細

いずれかの引数が負の場合、結果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
ERROR_
が 1 に設定されます。すべての引数が欠損値の場合、結果は欠損値になります。それ以外の場合、結

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 は、非欠損値の引数の数で、$x_1, x_2, \ldots, x_n$ は、それらの引数の値です。調和平均は次のようになります。

$$\frac{n}{\frac{1}{x_1} + \frac{1}{x_2} + \ldots + \frac{1}{x_n}}$$

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。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x1=harmean(1,2,4,4);	2

SAS ステートメント	結果
x2=harmean(.,4,12,24);	8
x3=harmean(of x1-x2);	3.2

関連項目:

関数:

- “GEOMEAN 関数” (501 ページ)
- “GEOMEANZ 関数” (502 ページ)
- “HARMEANZ 関数” (516 ページ)
- “MEAN 関数” (642 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

構文

HARMEANZ(*argument*<,*argument*,…>)

必須引数

引数

負でない数値定数、変数または式です。

ヒント: 引数リストには OF で始まる変数のリストを含められます。

詳細

いずれかの引数が負の場合、結果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
ERROR_
が 1 に設定されます。すべての引数が欠損値の場合、結果は欠損値になります。それ以外の場合、結

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 は、非欠損値の引数の数で、$x_1, x_2, \ldots, x_n$ は、それらの引数の値です。調和平均は次のようになります。

$$\frac{n}{\frac{1}{x_1} + \frac{1}{x_2} + \ldots + \frac{1}{x_n}}$$

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。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x1=harmeanz(1,2,4,4);	2
x2=harmeanz(.,4,12,24);	8
x3=harmeanz(of x1-x2);	3.2

関連項目:

関数:

- “GEOMEAN 関数” (501 ページ)
- “GEOMEANZ 関数” (502 ページ)
- “HARMEAN 関数” (515 ページ)
- “MEAN 関数” (642 ページ)

HBOUND 関数

配列の上限を返します。

カテゴリ: 配列

構文

HBOUND<*n*> (*array-name*)

HBOUND(*array-name*,*bound-n*)

必須引数

array-name
同じ DATA ステップで以前に定義した配列の名前です。

bound-n
上限を調べる次元を指定する数値定数、変数または式です。*bound-n* は、*n* を指定していない場合にのみ使用します。

オプション引数

n
上限を調べる次元を指定する整数定数です。*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。

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。配列処理で HBOUND を使用すると、配列の上限を変更するたびに

DO 反復グループの上限を変更しないで済みます。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。

比較

- HBOUND は、配列のディメンションでの上限のリテラル値を返します。
- DIM は、常に配列のディメンションにある要素の合計数を返します。

  注: 配列のディメンションの下限値が 1 以外で、上限値が配列のディメンションにある要素の合計数以外の場合、この違いが重要です。

サンプル

サンプル 1: 1 次元配列

この例では、HBOUND は次元の上限(値 5)を返します。そのため、SAS は DO ループでステートメントを 5 回繰り返します。

```
 array big{5} weight sex height state city;
do i=1 to hbound(big5);
more SAS statements;
end;
```

サンプル 2: 多次元配列

この例では、多次元配列で HBOUND 関数を指定する 2 つの方法を示します。どちらの方法でも HBOUND は同じ値を返します(SAS コード例の後にある表を参照)。

```
array mult{2:6,4:13,2} mult1-mult100;
```

構文	別の構文	値
HBOUND(MULT)	HBOUND(MULT,1)	6
HBOUND2(MULT)	HBOUND(MULT,2)	13
HBOUND3(MULT)	HBOUND(MULT,3)	2

関連項目:

関数:

- “DIM 関数” (370 ページ)
- “LBOUND 関数” (598 ページ)

ステートメント:

- “ARRAY ステートメント” (*SAS ステートメント: リファレンス*)
- “配列参照ステートメント” (*SAS ステートメント: リファレンス*)
- “配列処理” (*SAS 言語リファレンス: 解説編* 23 章)

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

構文

HMS(*hour*,*minute*,*second*)

必須引数

時
数値です。

分
数値です。

秒
数値です。

詳細

HMS 関数は、SAS 時間値を表す正の数値を返します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
hrid=hms(12,45,10); put hrid / hrid time.;	 45910 12:45:10

関連項目:

関数:

- “DHMS 関数” (366 ページ)
- “HOUR 関数” (522 ページ)
- “MINUTE 関数” (644 ページ)
- “SECOND 関数” (835 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

構文

HOLIDAY('*holiday*', *year*)

必須引数

'*holiday*'
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 2.1 *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	カナダ独立記念日の祝日	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
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)

祝日の値	説明	祝日の日付
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 ~ 金曜日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 ~ 日曜日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

年
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。

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、表 2.1 (520 ページ)を参照)。

多くの祝日の定義は時代とともに変わりますが、現在の祝日の定義が過去および将来の年に無限に拡張されます。多くの祝日の現在の表現は、1971 年またはその前後から始まっています。

HOLIDAY 関数は、SAS 日付値を返します。SAS 日付値をカレンダ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', 2007);**は、**NWKDOM(4, 5, 11, 2007);**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', 2007);**は、**MDY(12, 25, 2007);**と同じ値を返します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
thanks = holiday('thanksgiving', 2007); format thanks date9.; put thanks;	22NOV2007

SAS ステートメント	結果
boxing = holiday('boxing', 2007); format boxing date9.; put boxing;	26DEC2007
easter = holiday('easter', 2007); format easter date9.; put easter;	08APR2007
canada = holiday('canada', 2007); format canada date9.; put canada;	01JUL2007
fathers = holiday('fathers', 2007); format fathers date9.; put fathers;	17JUN2007
valentines = holiday('valentines', 2007); format valentines date9.; put valentines;	14FEB2007
victoria = holiday('victoria', 2007); format victoria date9.; put victoria;	21MAY2007

関連項目:

関数:

- “NWKDOM 関数” (695 ページ)
- “MDY 関数” (641 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

構文

HOUR(*time* | *datetime*)

必須引数

time
SAS 時間値を指定する数値定数、変数または式です。

datetime
SAS 日時値を指定する数値定数、変数または式です。

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。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果

now='1:30't;
h=hour(now);
put h;
	1

関連項目:

関数:

- “SECOND 関数” (835 ページ)

HTMLDECODE 関数

HTML 数値文字参照または HTML 文字実体参照を含む文字列をデコードし、デコードされた文字列を返します。

カテゴリ: Web ツール

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

HTMLDECODE(*expression*)

必須引数

式
文字定数、変数または式を指定します。

詳細

HTMLDECODE 関数は、次の文字エンティティ参照を認識します。

文字エンティティ参照	デコードされた文字
&amp;	&
&lt;	<
&gt;	>
&quot;	"

文字エンティティ参照	デコードされた文字
'	'

認識されないエンティティ(&<name>;)は、変更されずにそのまま出力文字列に残ります。

HTMLDECODE 関数は、次の形式の数値エンティティ参照を認識します。

&#*nnn*;
nnn は、1 桁以上の 10 進数を示します。

&#X*nnn*;
nnn は、1 桁以上の 16 進数を示します。

注: 現在の SAS セッションエンコーディングで表すことができない数値文字参照はデコードされません。参照は変更されずに出力文字列にコピーされます。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x1=htmldecode('not a <tag>');	not a <tag>
x2=htmldecode('&');	'&'
x3=htmldecode ('ABC');	'ABC'

関連項目:

関数:

- “HTMLENCODE 関数” (524 ページ)

HTMLENCODE 関数

HTML 文字実体参照を使用して文字をエンコードし、エンコードされた文字列を返します。

カテゴリ: Web ツール

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

HTMLENCODE(*expression*, <*options*>)

必須引数

式

文字定数、変数または式を指定します。デフォルトでは、より大きい(>)、より小さい(<)およびアンパサンド(&)文字は、それぞれ>、<および&としてエンコードされます。SAS 9 でのみ、この動作を *options* 引数で変更できます。

注: エンコードされた文字列が出力文字列よりも長くなる場合があります。出力変数を定義するときに長さを追加することを考慮する必要があります。エンコードされた文字列の長さが、定義されている最大長を超えると、出力文字列が切り捨てられる可能性があります。

オプション引数

オプション

エンコードする文字の種類を指定する文字定数、変数または式です。複数のオプションを使用する場合、空白で各オプションを区切ります。 次のオプションを使用できます。

オプション	文字関数	文字エンティティ参照	説明
amp	&	&amp;	HTMLENCODE 関数は、デフォルトでこれらの文字をエンコードします。これらの文字のみをエンコードする場合、options 引数を指定する必要はありません。ただし、options 引数の値を指定するとデフォルトが上書きされるため、エンコードするすべての文字のオプションを明示的に指定する必要があります。
gt	>	&gt;	
lt	<	&lt;	
apos	'	&apos;	このオプションは、HTML または XML タグ属性で使用するテキストのアポストロフィ(')文字をエンコードする場合に使用します。
quot	"	&quot;	このオプションは、HTML または XML タグ属性で使用するテキストの二重引用符(")文字をエンコードする場合に使用します。
7bit	7 ビット ASCII エンコーディングで表されない文字	&#x*nnn*; (Unicode)	*nnn* は、1 桁以上の 16 進数です。これらの文字をエンコードして、7 ビット ASCII エンコーディングのみをサポートする通信パス(ftp や電子メールなど)で簡単に転送できる HTML または XML を作成します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
htmlencode("John's test <tag>")	John's test <tag>
htmlencode("John's test <tag>",'apos')	John's test <tag>
htmlencode('John "Jon" Smith <tag>','quot')	John "Jon" Smith <tag>
htmlencode("'A&B&C'",'amp lt gt apos')	'A&B&C'
htmlencode('80'x, '7bit') ('80'x は西ヨーロッパロケールのユーロ記号)	€ (20AC はユーロ記号の Unicode コードポイント)

関連項目:

関数:

- "HTMLDECODE 関数" (523 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: 数学関数

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: $nu \geq 0$

x
数値の定数、変数または式を指定します。
範囲: $x \geq 0$

kode
負でない整数を指定する数値定数、変数または式です。

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。

$$\varepsilon^{-x} I_{nm}(x)$$

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=ibessel(2,2,0);	0.6889484477
x=ibessel(2,2,1);	0.0932390333

IFC 関数

式の真、偽、欠損に基づいて文字値を返します。

カテゴリ: 文字関数

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

IFC(*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 の場合に返される文字定数、変数または式を指定します。

value-returned-when-false
logical-expression の値が false の場合に返される文字定数、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文字定数、変数または式を指定します。

詳細

返される変数の長さ

DATA ステップで、まだ長さが割り当てられていない変数に IFC 関数から値が返される場合、その変数には 200 バイトの長さが設定されます。

基本

IFC 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。

IFC 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)、IFC は第 2 引数の値を返します。*logical-expression* が欠損値の場合、IFC は第 4 引数の値を返します(第 4 引数がある場合)。それ以外の場合、*logical-expression* が false であれば、IFC は第 3 引数の値を返します。

IFC 関数は DATA ステップの式で役立ちます。IF/THEN/ELSE 構文を使用しづらいまたは使用できない WHERE 句や他の式ではさらに便利です。

比較

IFC 関数は IFN 関数と類似していますが、IFN が数値を返すのに対して IFC は文字値を返す点が異なります。

サンプル

次の例では、IFC は式 **grade>80** を評価して、チームの複数のメンバのパフォーマンスを決定するロジックを実装します。結果は SAS ログに書き込まれます。

```
data _null_;
input name $ grade;
performance = ifc(grade>80, 'Pass ', 'Needs Improvement');
put name= performance=;
datalines;
John 74
Kareem 89
Kati 100
Maria 92
;
run;
```

ログ 2.10 *SAS ログの一部: IFC 関数*

```
name=John performance=Needs Improvement
name=Kareem performance=Pass
name=Kati performance=Pass
name=Maria performance=Pass
```

この例では、IF/THEN/ELSE 構文を使用して IFC 関数と同じ出力を生成します。結果は SAS ログに書き込まれます。

```
data _null_;
input name $ grade;
if grade>80 then performance='Pass ';
else performance = 'Needs Improvement';
put name= performance=;
datalines;
John 74
Sam 89
Kati 100
Maria 92
;
run;
```

ログ 2.11 SAS ログの一部: IF/THEN/ELSE 構文

```
name=John performance=Needs Improvement
name=Sam performance=Pass
name=Kati performance=Pass
name=Maria performance=Pass
```

関連項目:

関数:

- “IFN 関数” (529 ページ)

IFN 関数

式の真、偽、欠損に基づいて数値を返します。

カテゴリ: 数値

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

IFN(*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 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 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 は第 2 引数の値を返します。*logical-expression* が欠損値の場合、IFN は第 4 引数の値を返します(第 4 引数がある場合)。それ以外の場合、*logical-expression* が false であれば、IFN は第 3 引数の値を返します。

IFN 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 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 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 は数値を返す点が異なります。

サンプル

サンプル 1: IFN 関数を使用した手数料の計算

次の例では、IFN 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(TotalSales > 10000, TotalSales*.05, TotalSales*.02);
put commission=;
datalines;
25000
10000
500
10300
;
run;
```

SAS は次の出力を口グに書き込みます。

```
commission=1250
commission=200
commission=10
commission=515
```

サンプル 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
;
run;
```

SAS は次の出力を口グに書き込みます。

```
commission=1250
commission=200
commission=10
commission=515
```

サンプル 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;
```

画面 2.34 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” (527 ページ)

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: 文字関数

制限事項: I18N レベル 0 の関数は 1 バイト文字セット(SBCS)でのみ使用するように設計されています。

ヒント: DBCS に相当する関数は、*SAS 各国語サポート(NLS): リファレンスガイド*の KINDEX です。“DBCS の互換性” (532 ページ)を参照してください。

構文

INDEX(*source*,*excerpt*)

必須引数

ソース
検索する文字の定数、変数または式を指定します。

excerpt
source で検索する文字の文字列を指定する文字定数、変数または式です。

ヒント:
文字のリテラル文字列を引用符で囲みます。

先頭および末尾の空白はどちらも *excerpt* 引数の一部としてみなされます。末尾の空白を削除するには、INDEX 関数内の *excerpt* 変数に TRIM 関数を含めます。

詳細

基本

INDEX 関数は、*source* を左から右へ、*excerpt* で指定される文字列が最初に現れる個所を検索し、*source* 内での文字列の最初の文字の位置を返します。*source* 内に文字列が見つからない場合、INDEX は値 0 を返します。文字列が複数回現れる場合、INDEX は、最初の出現個所の位置のみを返します。

DBCS の互換性

DBCS に相当する関数は、*SAS 各国語サポート(NLS): リファレンスガイド*に記載されている KINDEX です。ただし、末尾の空白の処理方法に若干の違いがあります。KINDEX では、第 2 引数の複数の空白が第 1 引数の 1 つの空白と一致します。次の例では、2 つの関数の違いを示します。

```
index('ABC,DE F(X=Y)',' ') => 0
kindex('ABC,DE F(X=Y)',' ') => 7
```

サンプル

サンプル 1: ソース文字列の変数位置の検出

次の例では、*source* 内での *excerpt* 引数の最初の位置を検出します。

```
data _null_;
a = 'ABC.DEF(X=Y)';
b = 'X=Y';
x = index(a,b);
put x=;
run;
```

SAS は次の出力を口グに書き込みます。

```
x=9
```

サンプル 2: INDEX 関数と TRIM 関数を併用する場合の末尾の空白の削除

次の例では、TRIM 関数を使わずに INDEX 関数を使用する場合の結果を示します。TRIM 関数を使わずに INDEX を使用する場合、先頭および末尾の空白は *excerpt* 引数の一部としてみなされます。TRIM 関数と INDEX を併用する場合、次の例のように TRIM は *excerpt* 引数から末尾の空白を削除します。TRIM 関数は、INDEX 関数内で使用されます。

```
options nodate nostimer ls=78 ps=60;
data _null_;
length a b $14;
a='ABC.DEF (X=Y)';
b='X=Y';
q=index(a,b);
w=index(a,trim(b));
put q= w=;
run;
```

SAS は次の出力を口グに書き込みます。

```
q=0 w=10
```

関連項目:

関数:

- “FIND 関数” (448 ページ)
- “INDEXC 関数” (533 ページ)
- “INDEXW 関数” (534 ページ)

INDEXC 関数

文字式から指定した文字を検索し、その文字の位置を返します。

カテゴリ: 文字関数

制限事項: I18N レベル 0 の関数は 1 バイト文字セット(SBCS)でのみ使用するように設計されています。

ヒント: DBCS に相当する関数は、*SAS 各国語サポート(NLS): リファレンスガイド*の KINDEXC です。

構文

INDEXC(*source*,*excerpt-1*<,… *excerpt-n*>)

必須引数

ソース
検索する文字の定数、変数または式を指定します。

excerpt
source で検索する文字定数、変数または式を指定します。
ヒント: 複数の excerpt を指定する場合は、カンマで区切ります。

詳細

INDEXC 関数は、*source* を左から右へ、excerpt 内に含まれるいずれかの文字が最初に現れる個所を検索し、*source* 内でのその文字の位置を返します。*excerpt-1* ~ *excerpt-n* の文字がいずれも *source* 内で見つからなかった場合、INDEXC は値 0 を返します。

比較

INDEXC 関数は、文字列に含まれる個々の文字が最初に現れる個所を検索するのに対し、INDEX 関数は、文字列が部分文字列として最初に現れる個所を検索します。FINDC 関数には、さらに多くのオプションがあります。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
a='ABC.DEP (X2=Y1)'; x=indexc(a,'0123',';()=.'); put x;	4
b='have a good day'; x=indexc(b,'pleasant','very'); put x;	2

関連項目:

関数:

- “FINDC 関数” (450 ページ)
- “INDEX 関数” (531 ページ)
- “INDEXW 関数” (534 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: 文字関数

制限事項: I18N レベル 0 の関数は 1 バイト文字セット(SBCS)でのみ使用するように設計されています。

構文

INDEXW(*source*,*excerpt*<,*delimiters*>)

必須引数

ソース
検索する文字の定数、変数または式を指定します。

excerpt
source で検索する文字定数、変数または式を指定します。SAS は、*excerpt* から先頭および末尾の区切り文字を削除します。

オプション引数

delimiter
INDEXW が文字列の区切り文字として使用する文字を含む文字定数、変数または式を指定します。デフォルトの区切り文字は空白文字です。

詳細

INDEXW 関数は、*source* を左から右へ、*excerpt* が最初に現れる個所を検索し、*source* 内での部分文字列の最初の文字の位置を返します。*source* 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。文字列が複数回現れる場合、INDEXW は、最初の出現個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、*source* の最初および *source* の最後になります。代替区切り文字を使用する場合、INDEXW はテキストの最後を終了データとして認識しません。

第 2 引数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。

- *source* と *excerpt* 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。
- *excerpt* に空白のみが含まれている場合や長さが 0 の場合、*source* 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。FINDW 関数には、さらに多くのオプションがあります。

サンプル

サンプル 1: SAS の例の表

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
s='asdf adog dog'; p='dog '; x=indexw(s,p); put x;	11
s='abcdef x=y'; p='def'; x=indexw(s,p); put x;	0

SAS ステートメント	結果
x=″abc,def@ xyz″; abc=indexw(x, ″ abc ″, ″@″); put abc;	0
x=″abc,def@ xyz″; comma=indexw(x, ″,″, ″@″); put comma;	0
x='abc,def% xyz'; def=indexw(x, 'def', '%,'); put def;	5
x=″abc,def@ xyz″; at=indexw(x, ″@″, ″@″); put at;	0
x=″abc,def@ xyz″; xyz=indexw(x, ″ xyz″, ″@″); put xyz;	9
c=indexw(trimn(' '), ' ');	1
g=indexw(' x y ', trimn(' '));	0

サンプル 2: 区切り文字としてセミコロン(;)を使用する

次の例では、SAS プログラムでどのようにセミコロン区切り文字が使用されるのかを示します。このプログラムでは、CATX 関数も呼び出されます。CATX の各呼び出しの後で適切な位置にセミコロン区切り文字を配置し、INDEXW 関数の第 2 引数から空白を削除する必要があります。これを行わないと、検索は成功しません。

```
data temp;
infile datalines;
input name $12.;
datalines;
abcdef
abcdef
;
run;
data temp2;
set temp;
format name_list $1024.;
retain name_list ' ';
exists=indexw(name_list, trim(name), ';');
if exists=0 then do
name_list=catx(';', name_list, name)||';' ;
name_count +1;
put '----------------------------------';
put exists= ;
put name_list= ;
put name_count= ;
end;
run;
```

ログ 2.12 区切り文字としてセミコロンを使用した場合の出力

```
--------------------------------
exists=0
name_list=abcdef;name_count=1
```

この例では、CATX が最初に呼び出されるときに *name_list* が空白で、*name* の値が'abcdef'になっています。CATX は、セミコロンを追加せずに'abcdef'を返します。ただし、INDEXW が 2 回目に呼び出される場合、*name_list* の値は'abcdef'と 1018 (1024–6)個の空白、*name* の値は'abcdef'と 6 個の空白になります。INDEXW の第 3 引数はセミコロン(;)であるため、空白は有効となり、単語の境界を表しません。そのため、第 1 引数で第 2 引数は見つかりません。

例に空白がなければ、INDEXW の動作がわかりやすくなります。次の例では、第 1 引数に完全な単語 ABCDE が見つからないため、*x* の値として 0 が想定されます。

```
x = indexw('ABCDEF;XYZ', 'ABCDE', ';');
```

ゼロ以外の結果を返す第 2 引数の値は ABCDEF と XYZ だけです。

サンプル 3: 区切り文字として空白を使用する

次の例では、区切り文字として空白を使用します。

```
data temp;
infile datalines;
input name $12.;
datalines;
abcdef
abcdef
;
run;
data temp2;
set temp;
format name_list $1024.;
retain name_list ' ';
exists=indexw(name_list, name, ' ');
if exists=0 then do
name_list=catx(' ', name_list, name) ;
name_count +1;
put '--------------------------------';
put exists= ;
put name_list= ;
put name_count= ;
end;
run;
```

ログ 2.13 区切り文字として空白を使用した場合の出力

```
--------------------------------
exists=0
name_list=abcdefname_count=1
```

関連項目:

関数:

- “FINDW 関数” (457 ページ)
- “INDEX 関数” (531 ページ)
- “INDEXC 関数” (533 ページ)

INPUT 関数

指定した入力形式を使用して SAS が式を変換するときに生成された値を返します。

カテゴリ: 特殊関数

構文

INPUT(*source*,<? | ??>,*informat*.)

必須引数

ソース

特定の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。

?または??

無効なデータ値が読み取られた場合にエラーメッセージと入力行の両方を非表示にする、任意の疑問符(?)と二重疑問符(??)修飾子を指定します。? 修飾子は、無効なデータメッセージを非表示にします。??
修飾子は、無効なデータメッセージを非表示にし、さらに無効なデータが読み取られたときに自
ERROR_が 1 に設定されることを防ぎます。

informat.

ソースに適用する SAS 入力形式です。この引数は、入力形式名の後にピリオドを追加する必要があり、文字定数、変数および式は使用できません。

詳細

INPUT 関数が長さの割り当てられていない変数に文字値を返した場合、デフォルトでは変数の長さは入力形式の幅によって決定されます。

INPUT 関数では、指定した入力形式を使用して *source* の値を変換できます。入力形式によって、結果が数値であるか文字であるかが決定します。INPUT を使用して、文字値を数値または他の文字値に変換できます。

比較

INPUT 関数は、指定した入力形式を使用して SAS 式が変換されるときに生成された値を返します。その値を変数に格納するには、割り当てステートメントを使用する必要があります。INPUT ステートメントでは、入力形式を使用してデータ値を読み取ります。その値の変数への格納は任意です。

INPUT 関数では、名前の後にピリオドと任意の小数点指定を追加して入力形式を指定する必要があります。INPUTC 関数と INPUTN 関数では、文字定数、変数または式として入力形式を指定できます。

サンプル

サンプル 1: 文字値を数値に変換する

この例では、INPUT 関数を使用して文字値を数値に変換し、その値を別の変数に格納します。COMMA9.入力形式によって SALE 変数の値が読み取られ、カンマが取り除かれます。この結果の値 2115353 が FMTSALE に格納されます。

```
 data testin;
input sale $9.;
fmtsale=input(sale,comma9.);
datalines;
2,115,353
;
```

サンプル 2: PUT 関数と INPUT 関数を使用する

この例では、PUT で数値を文字列として返します。値 122591 が CHARDATE 変数に割り当てられます。INPUT で SAS 日付入力形式を使用して、文字列の値を SAS 日付値として返します。値 11681 が SASDATE 変数に格納されます。

```
 numdate=122591;
chardate=put(numdate,z6.);
sasdate=input(chardate,mmddyy6.);
```

サンプル 3: エラーメッセージを非表示にする

この例では、疑問符(?)修飾子を使用して、データエラーが検出された場合に無効なデータエラーメ
ERROR_が 1 に設定され、入力データ行が SAS ログに書き込まれます。

```
 y=input(x,? 3.1);
```

二重疑問符(??)修飾子はエラーメッセージと入力行を非表示にし、無効なデータが読み取られたと
ERROR_
が 1 に設定されることを防ぐため、次の 2 つの例は同じ結果になります。

- **y=input(x,?? 2.);**
- **y=input(x,? 2.); _error_=0;**

関連項目:

関数:

- “INPUTC 関数” (540 ページ)
- “INPUTN 関数” (541 ページ)
- “PUT 関数” (770 ページ)
- “PUTC 関数” (772 ページ)
- “PUTN 関数” (774 ページ)

ステートメント:

- “INPUT ステートメント” (*SAS ステートメント: リファレンス*)

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: 特殊関数

構文

INPUTC(*source*, *informat*<,*w*>)

必須引数

ソース
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。

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。

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。

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。INPUT 関数はコンパイル時に入力形式を指定するため、INPUT 関数を使用した方が高速です。

サンプル

この例では、文字の入力形式を指定する方法を示します。この例の PROC FORMAT ステップでは、このステップで同時に作成する 3 つの入力形式のいずれかの名前で変数値 1、2 および 3 を書式化する、出力形式 TYPEFMT.を作成します。入力形式には、質問の種類に応じて、異なる単語として"positive"、"negative"および"neutral"の応答が格納されます。PROC FORMAT で出力形式と入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、現在の質問の種類に適切な入力形式の値が含まれる変数 RESPINF を作成します。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。INPUTC 関数は、質問の種類と適切な入力形式に基づいて WORD の値を割り当てます。

```
 proc format;
value typefmt 1='$groupx'
2='$groupy'
3='$groupz';
invalue $groupx 'positive'='agree'
```

```
'negative'='disagree'
'neutral'='notsure';
invalue $groupy 'positive'='accept'
'negative'='reject'
'neutral'='possible';
invalue $groupz 'positive'='pass'
'negative'='fail'
'neutral'='retest';
run;
data answers;
input type response $;
respinformat = put(type, typefmt.);
word = inputc(response, respinformat);
datalines;
1 positive
1 negative
1 neutral
2 positive
2 negative
2 neutral
3 positive
3 negative
3 neutral
;
```

開始オブザベーションの WORD の値は **agree** です。最終オブザベーションの WORD の値は **retest** です。

関連項目:

関数:

- “INPUT 関数” (538 ページ)
- “INPUTN 関数” (541 ページ)
- “PUT 関数” (770 ページ)
- “PUTC 関数” (772 ページ)
- “PUTN 関数” (774 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: 特殊関数

構文

INPUTN(*source*, *informat*<,*w*<,*d*>>)

必須引数

ソース
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。

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。

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。

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。

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。INPUT 関数はコンパイル時に入力形式を指定するため、INPUT 関数を使用した方が高速です。

サンプル

この例では、数値の入力形式を指定する方法を示します。この例の PROC FORMAT ステップでは、SAS 日付入力形式の名前で変数値 1 および 2 を書式化する、出力形式 READDATE.を作成します。DATA ステップで、2 つの異なるソース(変数 SOURCE の値で指定)からの生データを使用して SAS データセットを作成します。各ソースには日付が異なって指定されています。レコードの読み取り後、DATA ステップで SOURCE の値を使用して、日付の読み取りに適切な入力形式の値が含まれる変数 DATEINF を作成します。また、SAS 日付の値が含まれる新しい変数 NEWDATE も DATA ステップで作成します。INPUTN 関数で、オブザベーションのソースと適切な入力形式に基づいて NEWDATE の値を割り当てます。

```
 proc format;
value readdate 1='date7.'
2='mmddyy8.';
run;
options yearcutoff=1920;
data fixdates (drop=start dateinformat);
length jobdesc $12;
input source id lname $ jobdesc $ start $;
dateinformat=put(source, readdate.);
newdate = inputn(start, dateinformat);
datalines;
1 1604 Ziminski writer 09aug90
1 2010 Clavell editor 26jan95
2 1833 Rivera writer 10/25/92
2 2222 Barnes proofreader 3/26/98
;
```

関連項目:

関数:

- “INPUT 関数” (538 ページ)

- “INPUTC 関数” (540 ページ)
- “PUT 関数” (770 ページ)
- “PUTC 関数” (772 ページ)
- “PUTN 関数” (774 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値を返します。

カテゴリ: 切り捨て関数

構文

INT(*argument*)

必須引数

引数
数値の定数、変数または式を指定します。

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。引数の値が整数の 1E-12 内にある場合、関数はその整数を返します。*argument* 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。*argument* 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。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果

var1=2.1;
x=int(var1);
put x;
	2

var2=-2.4;
y=int(var2);
put y;
	-2

a=int(1+1.e-11); put a;	1

SAS ステートメント	結果
b=int(-1.6); put b;	-1

関連項目:

関数:

- “CEIL 関数” (288 ページ)
- “FLOOR 関数” (470 ページ)
- “INTZ 関数” (582 ページ)

INTCINDEX 関数

周期インデックスを返します。この関数には、日付、時間または日時の間隔と値を指定します。

カテゴリ: 日付と時間

構文

INTCINDEX(*間隔*<<*multiple*.<*shift-index*>>> , *date-time-value*)

必須引数

間隔

WEEK、MONTH または QTR などの間隔名が含まれる文字定数、変数または式を指定します。*Interval* は、大文字または小文字で表示できます。*interval* に使用可能な値のリストについては、 表 7.3, “日時関数で使用される間隔,” (*SAS 言語リファレンス: 解説編*) を参照してください。

ヒント *interval* が文字定数の場合、値を引用符で囲みます。

ヒント *interval* に有効な値は、*date-time-value* が日付、時間または日時の値のいずれであるかによって異なります。

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。間隔名の一般的な形式を次に示します。

interval<*multiple.shift-index*>

間隔名の 3 つの部分は次のとおりです。

間隔

基本間隔の種類の名前を指定します。たとえば、YEAR で年間隔を指定します。

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。たとえば、間隔 YEAR2 は 2 年の期間の間隔、つまり隔年です。

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” (31 ページ)を参照してください。

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。たとえば、YEAR.3 で、各カレンダ年の 3 月 1 日に開始して翌年の 2 月末に終了するようにシフトされた年間隔を指定します。

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 は工ラーになります。

デフォルトのシフト期間が間隔と同じ場合、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 間隔はデフォルトでは MONTH の期間でシフトされるため、シフトインデックスで月間隔はシフトできません。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。

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" (31 ページ)を参照してください。

date-time-value
指定した間隔の期間を表す日付、時間または日時の値を指定します。

詳細

INTCINDEX 関数は、季節周期のインデックスを返します。この関数には、間隔と SAS 日付、時間または日時の値を指定します。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、季節周期(年)内の間隔(月)数は 12 です。WEEK は、DAY と等しい間隔の季節周期です。そのため、intcindex('day','01SEP78'd);では、1978 年 9 月 1 日はその年の 35 週目の 6 日目であるため 35 の値を返します。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" (31 ページ)を参照してください。

INTCINDEX 関数は、小売業界によるカレンダ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 言語リファレンス: 解説編*の"Retail Calendar Intervals: ISO 8601 Compliant"を参照してください。

比較

INTCINDEX 関数は周期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。

例 cycle_index = intcindex('day','04APR2005'd);では、INTCINDEX 関数は年間通算週を返します。例 index = intindex('day','04APR2005'd);では、INTINDEX 関数は曜日を返します。

例 cycle_index = intcindex('minute','01Sep78:00:00:00'dt);では、INTCINDEX 関数は時刻を返します。例 index = intindex('minute','01Sep78:00:00:00'dt);では、INTINDEX 関数は分を返します。

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。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
cycle_index1 = intcindex('day', '01SEP05'd); put cycle_index1;	35
cycle_index2 = intcindex('dtqtr', '23MAY2005:05:03:01'dt); put cycle_index2;	1
cycle_index3 = intcindex('tenday', '13DEC2005' d); put cycle_index3;	1
cycle_index4 = intcindex('minute', '23:13:02't); put cycle_index4;	24
var1 = 'semimonth'; cycle_index5 = intcindex(var1, '05MAY2005:10:54:03'dt); put cycle_index5;	1

関連項目:

関数:

- “INTINDEX 関数” (561 ページ)
- “INTCYCLE 関数” (552 ページ)
- “INTSEAS 関数” (575 ページ)

INTCK 関数

2 つの日付、時間または日時の値の間にある指定した種類の間隔の境界数を返します。

カテゴリ: 日付と時間

構文

INTCK(*間隔*<*multiple*> <*.shift-index*>, *start-date*, *end-date*, <'*method*'>)

INTCK(*custom-interval*, *start-date*, *end-date*, <'*method*'>)

必須引数

間隔

間隔名が含まれる文字定数、変数または式を指定します。*Interval* は、大文字または小文字で表示できます。*interval* に使用可能な値のリストについては、表 7.3, “日時関数で使用される間隔,” (*SAS 言語リファレンス: 解説編*) を参照してください。

間隔の種類(日付、日時または時間)は、*start-date* の値の種類と一致する必要があります。

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。間隔名の一般的な形式を次に示します。

interval<multiple.shift-index>
間隔名の 3 つの部分は次のとおりです。

間隔
基本間隔の種類の名前を指定します。たとえば、YEAR で年間隔を指定します。

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。たとえば、間隔 YEAR2 は 2 年の期間の間隔、つまり隔年です。

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" (31 ページ)を参照してください。

custom-interval
SAS データセットによって定義されるユーザー定義間隔を指定します。各オブザベーションには、2 つの変数 *begin* と *end* が含まれます。

要件 *custom-interval* 変数を使用する場合は、INTERVALDS システムオプションを使用する必要があります。

参照項目: カスタム間隔の詳細については、"詳細" (548 ページ)を参照してください。

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。たとえば、YEAR.3 で、各カレンダ年の 3 月 1 日に開始して翌年の 2 月末に終了するようにシフトされた年間隔を指定します。

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 は工エラーになります。

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します。そのため、シフトインデックスで月間隔はシフトできません。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。

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" (31 ページ)を参照してください。

start-date
開始 SAS 日付、時間または日時の値を表す SAS 式を指定します。

end-date
終了 SAS 日付、時間または日時の値を表す SAS 式を指定します。

オプション引数

'method'
離散手法と連続手法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
連続時間での測定を指定します。間隔は開始日に基づいてシフトされます。

連続手法は記念日の計算に役立ちます。たとえば、次のプログラムを実行することで結婚年数を計算できます。

```
data b;
 WeddingDay='14feb2000'd;
 Today=today();
 YearsMarried=INTCK('YEAR',WeddingDay,today(),'C');
 format WeddingDay Today date9.;
run;
proc print data=b;
run;
```

この結果は、WeddingDay=14FEB2000、Today=17NOV2010、YearsMarried=10です。

CONTINUOUS 手法の場合、2000 年 1 月 15 日～2000 年 2 月 15 日の月単位の間隔は 1 か月です。

別名: C または CONT

DISCRETE
離散時間での測定を指定します。離散手法は、間隔の境界(月末など)を数えます。

デフォルトの離散手法は、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 手法の場合、2000 年 1 月 31 日～2000 年 2 月 1 日の月単位の間隔は 1 か月です。

別名: D または DISC

デフォルト: DISCRETE

詳細

カレンダの間隔の計算

1 つの離散時間間隔内のすべての値は同等とみなされます。つまり、月間隔を指定した場合、2005 年 1 月 1 日と 2005 年 1 月 15 日は同等です。これら両方の日付は、2005 年 1 月 1 日に開始して 2005 年 1 月 31 日に終了する間隔を表します。間隔の開始日(2005 年 1 月 1 日)または間隔の終了日(2005 年 1 月 31 日)を使用して間隔を識別できます。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。

例 **intck('qtr','14JAN2005'd,'02SEP2005'd);**では、*start-date* ('14JAN2005'd)は 2005 年の第 1 四半期と同等です。*end-date* ('02SEP2005'd)は 2005 年の第 3 四半期と同等です。間隔数(*start-date* と *end-date* の間に次の間隔の開始点が含まれる回数)は 2 です。

デフォルトの離散手法を使用する 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。

- 関数 INTCK('MONTH','1jan1991'd,'31jan1991'd)では、2 つの日付が同月内に存在するため、0 を返します。
- 関数 INTCK('MONTH','31jan1991'd,'1feb1991'd)では、2 つの日付が 1 か月離れた別の月に存在するため、1 を返します。
- 関数 INTCK('MONTH','1feb1991'd,'31jan1991'd)では、1 番目の日付が 2 番目の日付よりも後の離散間隔内に存在するため、-1 を返します (1 番目の日付が 2 番目の日付よりも後で、2 つの日付が同じ離散間隔内に存在しない場合、INTCK は常に負の値を返します)。

離散手法を使用する場合、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、連続手法を使用します。

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” (31 ページ)を参照してください。

日付間隔と日時間隔

SAS 日時値で使用する必要のある間隔は、SAS 日時間隔です。日時間隔を形成するには、日付間隔に接頭辞"DT"を付加します。たとえば、MONTH は SAS 日付間隔で、DTMONTH は SAS 日時間隔です。同様に、YEAR は SAS 日付間隔で、DTYEAR は SAS 日時間隔です。次の例では、DTDAY 日時間隔を使用して 2011 年 8 月 1 日～2012 年 2 月 1 日の日数を返します。

```
data _null_;
days=intck('dtday', '01aug2011:00:10:48'dt, '01feb2012:00:10:48'dt);
put days=;
run;
```

SAS は次の出力を口グに書き込みます。

```
days=184
```

カスタム間隔

カスタム間隔は SAS データセットで定義します。データセットには *begin* 変数を含める必要があり、*end* 変数と *season* 変数を含めることもできます。各オブザベーションは、間隔の開始点が含まれる *begin* 変数、および存在する場合は間隔の終了点を含む *end* 変数で 1 つの間隔を表します。間隔は昇順で記述する必要があります。間隔間にギャップは存在できません。また、間隔は重複できません。

SAS システムオプション INTERVALDS=は、カスタム間隔を定義して間隔データセットを新しい間隔名に関連付けるために使用します。INTERVALDS=システムオプションを指定する方法の例を次に示します。

options intervalds=(*interval=libref.dataset-name*);

引数

間隔

間隔名を指定します。*interval* の値は、*libref.dataset-name* で命名されたデータセットです。

libref.dataset-name
ユーザーが指定した祝日を含むファイルのライブラリ参照名とデータセット名を指定します。

詳細については、“カスタム時間間隔” (34 ページ)を参照してください。

販売カレンダの間隔

小売業界では、1 年のカレンダを 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、“小売りカレンダの間隔: ISO 8601 遵守” (*SAS 言語リファレンス: 解説編* 7 章)を参照してください。

サンプル

サンプル 1: INTCK を使用した間隔の例

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
qtr=intck('qtr','10jan95'd,'01jul95'd); put qtr;	2
year=intck('year','31dec94'd, '01jan95'd); put year;	1
year=intck('year','01jan94'd, '31dec94'd); put year;	0
semi=intck('semiyear','01jan95'd, '01jan98'd); put semi;	6
weekvar=intck('week2.2','01jan97'd, '31mar97'd); put weekvar;	7
wdvar=intck('weekday7w','01jan97'd, '01feb97'd); put wdvar;	26

y='year';
date1='1sep1991'd;
date2='1sep2001'd;
newyears=intck(y,date1,date2);
put newyears;
	10

y=trim('year ');
date1='1sep1991'd + 300;
date2='1sep2001'd - 300;
newyears=intck(y,date1,date2);
put newyears;
	8

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、1994 年 12 月 31 日 ~ 1995 年 1 月 1 日の間に 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。これは、1994 年 1 月 1 日 ~ 1994 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、1995 年 1 月 1 日 ~ 1998 年 1 月 1 日の間に半年の間隔が 6 回含まれるため、SAS は 6 の値を返します (終了日が 1997 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、1997 年 1 月 1 日 ~ 1997 年 3 月 31 日の間に最初の月曜日から始まる 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 26 を返します。この例では 1997 年 1 月 1 日 ~ 1997 年 2 月 1 日の間に週末として土曜日のみを数えるため、この期間には 26 回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。最後の例では、引数に式が使用されています。

サンプル 2: 手法の比較例

method が異なる値の例を次に示します。

```
data a;
interval='month';
start='14FEB2000'd;
end='13MAR2000'd;
months_default=intck(interval, start, end);
months_discrete=intck(interval, start, end,'d');
months_continuous=intck(interval, start, end,'c');
output;

end='14MAR2000'd;
months_default=intck(interval, start, end);
months_discrete=intck(interval, start, end,'d');
months_continuous=intck(interval, start, end,'c');
output;

start='31JAN2000'd;
end='01FEB2000'd;
months_default=intck(interval, start, end);
months_discrete=intck(interval, start, end,'d');
months_continuous=intck(interval, start, end,'c');
output;
format start end date.;
run;

proc print data=a;
run;
```

画面 2.35 手法間の比較

The SAS System

Obs	interval	start	end	months_default	months_discrete	months_continuous
1	month	14FEB00	13MAR00	1	1	0
2	month	14FEB00	14MAR00	1	1	1
3	month	31JAN00	01FEB00	1	1	0

関連項目:

関数:

- “INTNX 関数” (567 ページ)

システムオプション:

- “INTERVALDS= System Option” in *SAS System Options: Reference*

INTCYCLE 関数

次に高い季節周期での期間(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

構文

INTCYCLE(*間隔* <<*multiple*.<*shift-index*>>> , <*seasonality*>)

必須引数

間隔

WEEK、MONTH または QTR などの間隔名が含まれる文字定数、変数または式を指定します。*Interval* は、大文字または小文字で表示できます。*interval* に使用可能な値のリストについては、 表 7.3, “日時関数で使用される間隔,” (*SAS 言語リファレンス: 解説編*)を参照してください。

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。間隔名の一般的な形式を次に示します。

interval<multiple.shift-index>

間隔名の 3 つの部分は次のとおりです。

間隔

基本間隔の種類の名前を指定します。たとえば、YEAR で年間隔を指定します。

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。たとえば、間隔 YEAR2 は 2 年の期間の間隔、つまり隔年です。

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” (31 ページ)を参照してください。

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。たとえば、YEAR.3 で、各カレンダ年の 3 月 1 日に開始して翌年の 2 月末に終了するようにシフトされた年間隔を指定します。

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。

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” (31 ページ)を参照してください。

オプション引数

季節性
数値を指定します。

この引数では、日付と時間の周期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。

サンプル: 次の例では、関数
```
INTCYCLE('MONTH', 3);
```
には *seasonality* 引数が指定されており、値 QTR が返されます。関数
```
INTCYCLE('MONTH');
```
には *seasonality* 引数が指定されておらず、値 YEAR が返されます。

詳細

基本

INTCYCLE 関数は、日付、時間または日時の間隔に応じて、季節周期の間隔を返します。たとえば、**INTCYCLE('MONTH');**では、1 月 ~ 12 月は 1 年周期を構成するため、値 YEAR を返します。**INTCYCLE('DAY');**では、日曜日 ~ 土曜日は 1 週間周期を構成するため、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” (31 ページ)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” (32 ページ)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” (31 ページ)を参照してください。

INTCYCLE 関数は、小売業界によるカレンダ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、“販売カレンダの間隔: ISO 8601 準拠” (34 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、次に高い季節周期での期間(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 User's Guide* を参照してください。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
cycle_year = intcycle('year'); put cycle_year;	YEAR
cycle_quarter = intcycle('qtr'); put cycle_quarter;	YEAR
cycle_3 = intcycle('month', 3); put cycle_3;	QTR
cycle_month = intcycle('month'); put cycle_month;	YEAR
cycle_weekday = intcycle('weekday'); put cycle_weekday;	WEEK
cycle_weekday2 = intcycle('weekday', 5); put cycle_weekday2;	WEEK
cycle_day = intcycle('day'); put cycle_day;	WEEK
cycle_day2 = intcycle('day', 10); put cycle_day2;	TENDAY
var1 = 'second'; cycle_second = intcycle(var1); put cycle_second;	DTMINUTE

関連項目:

関数:

- “INTSEAS 関数” (575 ページ)
- “INTINDEX 関数” (561 ページ)
- “INTCINDEX 関数” (544 ページ)

その他のリファレンス:

- *SAS/ETS User's Guide*

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

構文

INTFIT(*argument-1*, *argument-2*, '*type*')

必須引数

引数

SAS 日付値や日時値、またはオブザベーションを表す SAS 式を指定します。

ヒント: 日付値または日時値が使用できない場合、通常は引数としてオブザベーション番号が使用されます。

'*type*'

引数が SAS 日付値、日時値、オブザベーション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 *argument-1* と *argument-2* が日付値であることを指定します。

dt *argument-1* と *argument-2* が日時値であることを指定します。

obs *argument-1* と *argument-2* がオブザベーションであることを指定します。

詳細

INTFIT 関数は、間隔内に配置された 2 つの日付値、日時値またはオブザベーションに基づいて、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” (567 ページ)を参照してください。

INTFIT でオブザベーションの引数を使用する場合、オブザベーション番号を使用してその周期を確認できます。次の例では、INTFIT の最初の 2 つの引数はオブザベーション番号で、*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 9.3 User's Guide* を参照してください。

INTFIT 関数は、小売業界によるカレンダ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、“小売りカレンダの間隔: ISO 8601 遵守” (*SAS 言語リファレンス: 解説編* 7 章) を参照してください。

サンプル

サンプル 1: 2 つの日付に基づく間隔を確認する

2 つの日付に基づく間隔の例を次に示します。この例の *type* 引数は、入力を日付値として識別します。

```
data a;
length interval $20;
date1='01jan11'd;
do i=1 to 25;
date2=intnx('day', date1, i);
interval=intfit(date1, date2, 'd');
output;
end;
format date1 date2 date.;
run;
proc print data=a;
run;
```

画面 2.36 INTFIT 関数からの間隔の出力

The SAS System

Obs	interval	date1	i	date2
1	DAY	01JAN11	1	02JAN11
2	DAY2	01JAN11	2	03JAN11
3	DAY3.2	01JAN11	3	04JAN11
4	DAY4	01JAN11	4	05JAN11
5	DAY5.4	01JAN11	5	06JAN11
6	DAY6.5	01JAN11	6	07JAN11
7	WEEK.7	01JAN11	7	08JAN11
8	DAY8.5	01JAN11	8	09JAN11
9	DAY9.8	01JAN11	9	10JAN11
10	TENDAY	01JAN11	10	11JAN11
11	DAY11.6	01JAN11	11	12JAN11
12	DAY12.5	01JAN11	12	13JAN11
13	DAY13.13	01JAN11	13	14JAN11
14	WEEK2.14	01JAN11	14	15JAN11
15	SEMIMON	01JAN11	15	16JAN11
16	DAY16.5	01JAN11	16	17JAN11
17	DAY17.14	01JAN11	17	18JAN11
18	DAY18.17	01JAN11	18	19JAN11
19	DAY19.9	01JAN11	19	20JAN11
20	TENDAY2	01JAN11	20	21JAN11
21	WEEK3.7	01JAN11	21	22JAN11
22	DAY22.17	01JAN11	22	23JAN11
23	DAY23.22	01JAN11	23	24JAN11
24	DAY24.5	01JAN11	24	25JAN11
25	DAY25.4	01JAN11	25	26JAN11

この出力では、増分値が 1 日の場合、INTFIT 関数の結果が DAY であることを示しています。増分値が 2 日の場合、INTFIT 関数の結果は DAY2 です。増分値が

3 日の場合、INTFIT 関数の結果は DAY3.2(3 のシフトインデックス)です (2 つの入力日付が金曜日と月曜日の場合、結果は WEEKDAY です)。増分値が 7 日の場合、結果は WEEK です。

サンプル 2: 日付がオブザベーションとして識別される場合に 2 つの日付間に基づく間隔を確認する

2 つの日付に基づく間隔の例を次に示します。この例では、*type* 引数は入力をオブザベーションとして識別します。

```
data a;
length interval $20;
date1='01jan11'd;
do i=1 to 25;
date2=intnx('day', date1, i);
interval=intfit(date1, date2, 'obs');
output;
end;
format date1 date2 date.;
run;
proc print data=a;
run;
```

画面 2.37 *日付がオブザベーションとして識別された場合の INTFIT 関数からの間隔の出力*

The SAS System

Obs	interval	date1	i	date2
1	OBS	01JAN11	1	02JAN11
2	OBS2	01JAN11	2	03JAN11
3	OBS3.2	01JAN11	3	04JAN11
4	OBS4	01JAN11	4	05JAN11
5	OBS5.4	01JAN11	5	06JAN11
6	OBS6.5	01JAN11	6	07JAN11
7	OBS7.2	01JAN11	7	08JAN11
8	OBS8.5	01JAN11	8	09JAN11
9	OBS9.8	01JAN11	9	10JAN11
10	OBS10.9	01JAN11	10	11JAN11
11	OBS11.6	01JAN11	11	12JAN11
12	OBS12.5	01JAN11	12	13JAN11
13	OBS13.13	01JAN11	13	14JAN11
14	OBS14.9	01JAN11	14	15JAN11
15	OBS15.14	01JAN11	15	16JAN11
16	OBS16.5	01JAN11	16	17JAN11
17	OBS17.14	01JAN11	17	18JAN11
18	OBS18.17	01JAN11	18	19JAN11
19	OBS19.9	01JAN11	19	20JAN11
20	OBS20.9	01JAN11	20	21JAN11
21	OBS21.2	01JAN11	21	22JAN11
22	OBS22.17	01JAN11	22	23JAN11
23	OBS23.22	01JAN11	23	24JAN11
24	OBS24.5	01JAN11	24	25JAN11
25	OBS25.4	01JAN11	25	26JAN11

関連項目:

関数:

- “INTCK 関数” (546 ページ)
- “INTNX 関数” (567 ページ)

INTFMT 関数

推奨 SAS 出力形式を返します。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

構文

INTFMT(*間隔*<<*multiple*.<.*shift-index*>>> , '*size*')

必須引数

間隔

WEEK、MONTH または QTR などの間隔名が含まれる文字定数、変数または式を指定します。*Interval* は、大文字または小文字で表示できます。*interval* に使用可能な値のリストについては、*SAS 言語リファレンス: 解説編*の 表 7.3, “日時関数で使用される間隔,” (*SAS 言語リファレンス: 解説編*) を参照してください。

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。間隔名の一般的な形式を次に示します。

interval<*multiple.shift-index*>

間隔名の 3 つの部分は次のとおりです。

間隔

基本間隔の種類の名前を指定します。たとえば、YEAR で年間隔を指定します。

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。たとえば、間隔 YEAR2 は 2 年の期間の間隔、つまり隔年です。

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” (31 ページ)を参照してください。

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。たとえば、YEAR.3 で、各カレンダ年の 3 月 1 日に開始して翌年の 2 月末に終了するようにシフトされた年間隔を指定します。

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。

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" (31 ページ)を参照してください。

'size'

LONG または SHORT を指定します。出力形式に年の値が含まれる場合、LONG または L では 4 桁の年を使用する出力形式が指定されます。SHORT または S では、2 桁の年を使用する出力形式が指定されます。

詳細

INTFMT 関数は、日付、時間または日時の間隔に応じて推奨出力形式を返し、指定した間隔の時系列に関連付けられた時間 ID 値を表示します。SIZE の有効な値(LONG、L、SHORT、S のいずれか)によって、出力形式が SAS 日付値を参照するときに 2 桁の年を使用するか 4 桁の年を使用するかを指定します。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" (31 ページ)を参照してください。

INTFMT 関数は、小売業界によるカレンダ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 言語リファレンス: 解説編*の"Retail Calendar Intervals: ISO 8601 Compliant"を参照してください。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
fmt1 = intfmt('qtr', 's'); put fmt1;	YYQC4.
fmt2 = intfmt('qtr', 'l'); put fmt2;	YYQC6.
fmt3 = intfmt('month', 'l'); put fmt3;	MONYY7.
fmt4 = intfmt('week', 'short'); put fmt4;	WEEKDATX15.
fmt5 = intfmt('week3.2', 'l'); put fmt5;	WEEKDATX17.
fmt6 = intfmt('day', 'long'); put fmt6;	DATE9.
var1 = 'month2'; fmt7 = intfmt(var1, 'long'); put fmt7;	MONYY7.

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

構文

INTGET(*date-1*, *date-2*, *date-3*)

必須引数

date
SAS 日付値または日時値を指定します。

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～2000 年 3 月 3 日の日数は 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～2000 年 3 月 9 日の日数は 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～2000 年 3 月 1 日の月数は 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 です。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 間隔のリストについては、表 7.3, “日時関数で使用される間隔,” (*SAS 言語リファレンス: 解説編*) を参照してください。

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。

販売カレンダの間隔

INTGET 関数は、小売業界によるカレンダ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、“小売りカレンダの間隔: ISO 8601 遵守” (*SAS 言語リファレンス: 解説編* 7 章) を参照してください。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
interval=intget('01jan00'd,'01jan01'd,'01may01'd); put interval;	MONTH4
interval=intget('29feb80'd,'28feb82'd,'29feb84'd); put interval;	YEAR2.2
interval=intget('01feb80'd,'16feb80'd,'01mar80'd); put interval;	SEMIMONTH
interval=intget('2jan09'd,'2feb10'd,'2mar11'd); put interval;	MONTH13.4
interval=intget('10feb80'd,'19feb80'd,'28feb80'd); put interval;	DAY9.2
interval=intget('01apr2006:00:01:02'dt, '01apr2006:00:02:02'dt, '01apr2006:00:03:02'dt); put interval;	MINUTE

関連項目:

関数:

- “INTFIT 関数” (555 ページ)
- “INTNX 関数” (567 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、時間または日時の間隔と値を指定します。

カテゴリ: 日付と時間

構文

INTINDEX(*間隔*<<*multiple*.<*shift-index*>>> , *date-value*, <*seasonality*>)

必須引数

間隔

WEEK、MONTH または QTR などの間隔名が含まれる文字定数、変数または式を指定します。*Interval* は、大文字または小文字で表示できます。*interval* に使用可能な値のリストについては、 表 7.3, “日時関数で使用される間隔,” (*SAS 言語リファレンス: 解説編*) を参照してください。

ヒント *interval* が文字定数の場合、値を引用符で囲みます。

ヒント *interval* に有効な値は、*date-value* が日付、時間、日時の値のいずれであるかによって異なります。詳細については、“よく使用される時間間隔” (32 ページ) を参照してください。

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。間隔名の一般的な形式を次に示します。

interval<multiple.shift-index>

間隔名の 3 つの部分は次のとおりです。

間隔

基本間隔の種類の名前を指定します。たとえば、YEAR で年間隔を指定します。

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。たとえば、間隔 YEAR2 は 2 年の期間の間隔、つまり隔年です。

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” (31 ページ)を参照してください。

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。たとえば、YEAR.3 で、各カレンダ年の 3 月 1 日に開始して翌年の 2 月末に終了するようにシフトされた年間隔を指定します。

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。

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” (31 ページ)を参照してください。

date-value

指定した間隔の期間を表す日付、時間または日時の値を指定します。

オプション引数

季節性

数値または周期を指定します。

この引数では、日付と時間の周期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。

サンプル: 次の例では、関数

```
INTINDEX('MONTH', date, 3);
```

は次の関数と同じ結果になります。

```
INTINDEX('MONTH', date, 'QTR');
```

最初の例の *Seasonality* は数値(月数)で、2 番目の例の *seasonality* は周期(QTR)です。

詳細

INTINDEX 関数の間隔

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、時間または日時の値を指定します。季節インデックスは、指定した間隔の季節周期での日付、時間または日時の値の位置を表す数値です。たとえば、1 年周期には 12 か月があり 12 月はその年の 12 番目の月であるため、**intindex('month', '01DEC2000'd);**は 12 の値を返します。**intindex('qtr', '01JAN2000'd);**および **intindex('qtr', '31MAR2000'd);**では、両方のステートメントに 2000 年の第 1 四半期内の値が含まれるため、INTINDEX は同じ値を返します。ステートメント **intindex('day', '01DEC2000'd);**では、日単位のデータは週単位の周期に含まれ、2000 年 12 月 1 日は金曜日(6 番目の曜日)であるため、6 の値を返します。

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、時間または日時の値に適合する必要があります。たとえば、**intindex('month', '01DEC2000'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。次の例 **intindex('day', '01DEC2000'd);**では、週間隔には 7 日が含まれ、2000 年 12 月 1 日は金曜日(6 番目の曜日)であるため、6 の値を返します。DAY 間隔には SAS 日付値が必要です。

例 **intindex('qtr', '01JAN2000:00:00:00'dt);**では、QTR 間隔は日付に日時値ではなく SAS 日付値が必要なため、エラーを返します。例 **intindex('dtmonth', '01DEC2000:00:00:00'dt);**では、12 の値を返します。DTMONTH 間隔には日時値が必要です。

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” (31 ページ)を参照してください。

販売カレンダの間隔

INTINDEX 関数は、小売業界によるカレンダ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、“販売カレンダの間隔: ISO 8601 準拠” (34 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。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、時間または日時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 User's Guide* 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 関数は周期インデックスを返します。

例 index = intindex('day','04APR2005'd);では、INTINDEX 関数は曜日を返します。例 cycle_index = intcindex('day','04APR2005'd);では、INTCINDEX 関数は年間通算週を返します。

例 index = intindex('minute','01Sep78:00:00:00'dt);では、INTINDEX 関数は分を返します。例 cycle_index = intcindex('minute','01Sep78:00:00:00'dt);では、INTCINDEX 関数は時刻を返します。

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。

サンプル

サンプル 1: INTINDEX に 2 つの引数を使用した例

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
interval1 = intindex('qtr', '14AUG2005'd); put interval1;	3
interval2 = intindex('dtqtr','23DEC2005:15:09:19'dt); put interval2;	4
interval3 = intindex('hour', '09:05:15't); put interval3;	10
interval4 = intindex('month', '26FEB2005'd); put interval4;	2
interval5 = intindex('dtmonth', '28MAY2005:05:15:00'dt); put interval5;	5
interval6 = intindex('week', '09SEP2005'd); put interval6;	36
interval7 = intindex('tenday', '16APR2005'd); put interval7;	11

サンプル 2: 季節性の例

SAS では、デフォルトの季節周期を使用します。たとえば、月単位のデータは年単位の季節とみなされます。ただし、月単位のデータを半年単位の季節周期とすることもできます。この例では、デフォルトを使用するかわりに 3 番目の引数 *seasonality* を使用して、季節性を指定します。うるう年の処理方法も示しています。

```
data weekly;
*do year = 2000 to 2010;
year = 2004;
```

```
NewYear = HOLIDAY('NEWYEAR',year);
do i = -5 to 5;
date = INTNX('week',NewYear,i);
output;
end;
*end;
format date date.;
format NewYear date.;
run;

/* The standard leap week is the first week of year. */
/* An alternative method uses a third argument:leap week is week 53. */
title "Using a Third Argument to Control Weekly Seasonality";
data LeapWeekExample;
set weekly;
StandardIndex = INTINDEX('week',date);
IndexWithLeap = INTINDEX('week',date,53);
run;
proc print;
run;

/* Using a number and an interval can be equivalent for the third argument. */
title "Using the Third Argument as a Number or Cycle";
data Equiv3rdArg;
set sashelp.air(obs=12);
defaultSeasonal = INTINDEX('MONTH',date);
SeasonalArg12 = INTINDEX('MONTH',date,12);
SeasonalArgYear = INTINDEX('MONTH',date,'YEAR');
format date date.;
run;
proc print;
run;


/* Use the third argument for non-standard seasonality. */
title "Using the Third Argument for Non-Standard Seasonality";
data NonStandardSeasonal;
set sashelp.air(obs=24);
/* Standard Index - MONTH is Yearly Seasonal */
StandardIndex = INTINDEX('MONTH',date);
SemiYrIndex = INTINDEX('MONTH',date,'SEMIYR');
Index6 = INTINDEX('MONTH',date,6);
format date date.;
run;

proc print;
run;
```

画面 2.38 季節性の例からの出力

Using a Third Argument to Control Weekly Seasonality

Obs	year	NewYear	i	date	StandardIndex	IndexWithLeap
1	2004	01JAN04	-5	23NOV03	48	48
2	2004	01JAN04	-4	30NOV03	49	49
3	2004	01JAN04	-3	07DEC03	50	50
4	2004	01JAN04	-2	14DEC03	51	51
5	2004	01JAN04	-1	21DEC03	52	52
6	2004	01JAN04	0	28DEC03	1	53
7	2004	01JAN04	1	04JAN04	1	1
8	2004	01JAN04	2	11JAN04	2	2
9	2004	01JAN04	3	18JAN04	3	3
10	2004	01JAN04	4	25JAN04	4	4
11	2004	01JAN04	5	01FEB04	5	5

Using the Third Argument as a Number or Cycle

Obs	DATE	AIR	defaultSeasonal	SeasonalArg12	SeasonalArgYear
1	01JAN49	112	1	1	1
2	01FEB49	118	2	2	2
3	01MAR49	132	3	3	3
4	01APR49	129	4	4	4
5	01MAY49	121	5	5	5
6	01JUN49	135	6	6	6
7	01JUL49	148	7	7	7
8	01AUG49	148	8	8	8
9	01SEP49	136	9	9	9
10	01OCT49	119	10	10	10
11	01NOV49	104	11	11	11
12	01DEC49	118	12	12	12

Using the Third Argument for Non-Standard Seasonality

Obs	DATE	AIR	StandardIndex	SemiYrIndex	Index6
1	01JAN49	112	1	1	1
2	01FEB49	118	2	2	2
3	01MAR49	132	3	3	3
4	01APR49	129	4	4	4
5	01MAY49	121	5	5	5
6	01JUN49	135	6	6	6
7	01JUL49	148	7	1	1
8	01AUG49	148	8	2	2
9	01SEP49	136	9	3	3
10	01OCT49	119	10	4	4
11	01NOV49	104	11	5	5
12	01DEC49	118	12	6	6
13	01JAN50	115	1	1	1
14	01FEB50	126	2	2	2
15	01MAR50	141	3	3	3
16	01APR50	135	4	4	4
17	01MAY50	125	5	5	5
18	01JUN50	149	6	6	6
19	01JUL50	170	7	1	1
20	01AUG50	170	8	2	2
21	01SEP50	158	9	3	3
22	01OCT50	133	10	4	4
23	01NOV50	114	11	5	5
24	01DEC50	140	12	6	6

関連項目:

関数:

- “INTCINDEX 関数” (544 ページ)
- “INTSEAS 関数” (575 ページ)
- “INTCYCLE 関数” (552 ページ)

その他のリファレンス:

- *SAS/ETS User's Guide*

INTNX 関数

指定した時間間隔で日付、時間または日時の値を増分し、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

構文

INTNX(*間隔*<*multiple*> <.*shift-index*>, *start-from*, *increment*<, '*alignment*'>)

INTNX(*custom-interval*, *start-from*, *increment*<, '*alignment*'>)

必須引数

間隔

WEEK、SEMIYEAR、QTR、HOUR などの時間間隔が含まれる文字定数、変数または式を指定します。*Interval* は、大文字または小文字で表示できます。*interval* に使用可能な値のリストについては、*SAS 言語リファレンス: 解説編* の"Intervals Used with Date and Time Functions"の表を参照してください。

ヒント 間隔の種類(日付、日時または時間)は、*start-from* の値の種類と一致する必要があります。

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。間隔名の一般的な形式を次に示します。

interval<*multiple.shift-index*>

間隔名の 3 つの部分は次のとおりです。

間隔

基本間隔の種類の名前を指定します。たとえば、YEAR で年間隔を指定します。

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。たとえば、間隔 YEAR2 は 2 年の期間の間隔、つまり隔年です。

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" (31 ページ)を参照してください。

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。たとえば、YEAR.3 で、各カレンダ年の 3 月 1 日に開始して翌年の 2 月末に終了するようにシフトされた年間隔を指定します。

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します。そのため、シフトインデックスで月間隔はシフトできません。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、偶数月の 1 日目に開始する 2 か月の期間を指定します。

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" (31 ページ)を参照してください。

start-from

開始点を識別する SAS 日付、時間または日時の値を表す SAS 式を指定します。

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。

オプション引数

'*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: 詳細については、“SAME 配置” (570 ページ)を参照してください。

デフォルト: BEGINNING

参照項目: 詳細については、“SAS 日付出力の間隔内の位置合わせ” (570 ページ)を参照してください。

custom-interval
定義する間隔を指定します。

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 (SAS 日付値をカレンダ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)。2003 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。

```
x=intnx('week', '17oct03'd, 6);
put x date9.;
```

INTNX は値 23NOV2003 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” (31 ページ)を参照してください。

SAS 日付出力の間隔内の位置合わせ

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日付を返します。同じカレンダ日付は、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。有効なシフト期間については、 表 7.3, “日時関数で使用される間隔,” (*SAS 言語リファレンス: 解説編*) 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、 表 7.3, “日時関数で使用される間隔,” (*SAS 言語リファレンス: 解説編*) 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。

```
intnx('week', '15mar2000'd, 1, 'same'); returns 22MAR2000
intnx('dtweek', '15mar2000:8:45'dt, 1, 'same'); returns 22MAR00:08:45:00
intnx('year', '15mar2000'd, 5, 'same'); returns 15MAR2005
```

日付を調整する

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。

```
intnx('month', '15mar2000'd, 5, 'same'); returns 15AUG2000
intnx('year', '29feb2000'd, 2, 'same'); returns 28FEB2002
intnx('month', '31aug2001'd, 1, 'same'); returns 30SEP2001
intnx('year', '01mar1999'd, 1, 'same'); returns 01MAR2000 (the first day of the
third month of the year)
```

例 **intnx('year', '29feb2000'd, 2);**では、INTNX 関数は値 01JAN2002 を返します。この値は、開始日の年(2000)から 2 年後の最初の日付です。

例 **intnx('year', '29feb2000'd, 2, 'same');**では、INTNX 関数は値 28FEB2002 を返します。この場合、開始日は 2000 年で、結果となる年は 2 年後(2002)、月は同月(2 月)、日付は 28 日が 2002 年 2 月 29 日に最も近いため、これらの値になります。

カスタム間隔

カスタム間隔は SAS データセットで定義します。データセットには *begin* 変数を含める必要があり、*end* 変数と *season* 変数を含めることもできます。各オブザベーションは、間隔の開始点が含まれる *begin* 変数、および存在する場合は間隔の終了点を含む *end* 変数で 1 つの間隔を表します。間隔は昇順で記述する必要があります。間隔間にギャップは存在できません。また、間隔は重複できません。

SAS システムオプション INTERVALDS=は、カスタム間隔を定義して間隔データセットを新しい間隔名に関連付けるために使用します。INTERVALDS=システムオプションを指定する方法の例を次に示します。

options intervalds=(*interval*=*libref.dataset-name*);

引数

間隔
間隔名を指定します。*interval* の値は、*libref.dataset-name* で命名されたデータセットです。

libref.dataset-name
ユーザーが指定した祝日を含むファイルのライブラリ参照名とデータセット名を指定します。

詳細については、"カスタム時間間隔" (34 ページ)を参照してください。

販売カレンダの間隔

小売業界では、1 年のカレンダを 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、"小売りカレンダの間隔: ISO 8601 遵守" (*SAS 言語リファレンス: 解説編* 7 章) を参照してください。

サンプル

サンプル 1

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
yr=intnx('year','05feb94'd,3); put yr / yr date7.;	13515 01JAN97
x=intnx('month','05jan95'd,0); put x / x date7.;	12784 01JAN95
next=intnx('semiyear','01jan97'd,1); put next / next date7.;	13696 01JUL97

SAS ステートメント	結果
past=intnx('month2','01aug96'd,-1); put past / past date7.;	13270 01MAY96
sm=intnx('semimonth2.2','01apr97'd,4); put sm / sm date7.;	13711 16JUL97

x='month';
date='1jun1990'd;
nextmon=intnx(x,date,1);
put nextmon / nextmon date7.;
	11139
	01JUL90

x1='month ';
x2=trim(x1);
date='1jun1990'd - 100;
nextmonth=intnx(x2,date,1);
put nextmonth / nextmonth date7.;
	11017
	01MAR90

任意の *alignment* 引数を使用して日付を進める例を次に示します。

SAS ステートメント	結果
date1=intnx('month','01jan95'd,5,'beginning'); put date1 / date1 date7.;	12935 01JUN95
date2=intnx('month','01jan95'd,5,'middle'); put date2 / date2 date7.;	12949 15JUN95
date3=intnx('month','01jan95'd,5,'end'); put date3 / date3 date7.;	12964 30JUN95
date4=intnx('month','01jan95'd,5,'sameday'); put date4 / date4 date7.;	12935 01JUN95
date5=intnx('month','15mar2000'd,5,'same'); put date5 / date5 date9.;	14837 15AUG2000

interval='month';
date='1sep2001'd;
align='m';
date4=intnx(interval,date,2,align);
put date4 / date4 date7.;
	15294
	15NOV01

x1='month ';
x2=trim(x1);
date='1sep2001'd + 90;
date5=intnx(x2,date,2,'m');
put date5 / date5 date7.;
	15356
	16JAN02

サンプル 2: カスタム間隔の使用例

次の例では、INTNX 関数の *custom-interval* 形式を使用して、指定した時間間隔で日付、時間または日時の値を増分します。

```
options intervalds=(weekdaycust=dstest);
data dstest;
format begin end date9.;
begin='01jan2008'd; end='01jan2008'd; output;
begin='02jan2008'd; end='02jan2008'd; output;
begin='03jan2008'd; end='03jan2008'd; output;
begin='04jan2008'd; end='06jan2008'd; output;
begin='07jan2008'd; end='07jan2008'd; output;
begin='08jan2008'd; end='08jan2008'd; output;
begin='09jan2008'd; end='09jan2008'd; output;
begin='10jan2008'd; end='10jan2008'd; output;
begin='11jan2008'd; end='13jan2008'd; output;
begin='14jan2008'd; end='14jan2008'd; output;
begin='15jan2008'd; end='15jan2008'd; output;
run;

data _null_;
format start date9. endcustom date9.;
start='01jan2008'd;
do i=0 to 9;
endcustom=intnx('weekdaycust', start, i);
put endcustom;
end;
run;
```

SAS は次の出力をログに書き込みます。

```
01JAN2008
02JAN2008
03JAN2008
04JAN2008
07JAN2008
08JAN2008
09JAN2008
10JAN2008
11JAN2008
14JAN2008
```

関連項目:

関数:

- “INTCK 関数” (546 ページ)
- “INTSHIFT 関数” (578 ページ)

システムオプション:

- “INTERVALDS= System Option” in *SAS System Options: Reference*

INTRR 関数

内部利益率を分数で返します。

カテゴリ: 財務関数

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

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。

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 (“NETPV 関数” (663 ページ) を参照)。内部利益率は次の式で求められます。

$$r = \begin{cases} \frac{1}{x^{freq}} - 1 & freq > 0 \\ -log_{\varepsilon}(x) & freq = 0 \end{cases}$$

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。

サンプル

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表されます。

```
rate=intrr(1,-400,100,200,300);
```

返される値は 0.19438 です。

関連項目:

関数:

- “IRR 関数” (586 ページ)

INTSEAS 関数

季節周期の長さを返します。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

構文

INTSEAS(*間隔*<<*multiple*.<*shift-index*>>> <*seasonality*>)

必須引数

間隔

WEEK、MONTH または QTR などの間隔名が含まれる文字定数、変数または式を指定します。*Interval* は、大文字または小文字で表示できます。*interval* に使用可能な値のリストについては、 表 7.3, “日時関数で使用される間隔,” (*SAS 言語リファレンス: 解説編*) を参照してください。

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。間隔名の一般的な形式を次に示します。

interval<*multiple.shift-index*>

間隔名の 3 つの部分は次のとおりです。

間隔

基本間隔の種類の名前を指定します。たとえば、YEAR で年間隔を指定します。

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。たとえば、間隔 YEAR2 は 2 年の期間の間隔、つまり隔年です。

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” (31 ページ)を参照してください。

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。たとえば、YEAR.3 で、各カレンダ年の 3 月 1 日に開始して翌年の 2 月末に終了するようにシフトされた年間隔を指定します。

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 は工ラーになります。

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。

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" (31 ページ)を参照してください。

オプション引数

季節性
数値または周期を指定します。

この引数では、日付と時間の周期をより柔軟に操作できます。53 週の年がある場合、例 **INTSEAS('WEEK', 53);**のように *seasonality* の値に 53 を使用することで、容易に季節性を決定できます。デフォルトでは、**INTSEAS('WEEK');**は 52 です。

サンプル: 関数
INTSEAS('*interval*, *seasonality*);
は、*seasonality* に数値を指定した場合は数を返します。関数
INTSEAS('MONTH', 'QTR');
は、QTR 周期を指定した場合は 3 の値を返します。

詳細

基本
INTSEAS 関数は、季節周期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、季節周期(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、季節周期内の間隔数は 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、季節周期の間隔数は 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" (31 ページ)を参照してください。

販売カレンダの間隔

小売業界では、1 年のカレンダを 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、“小売りカレンダの間隔: ISO 8601 遵守” (*SAS 言語リファレンス: 解説編* 7 章) 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周期の長さを返します。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 User's Guide* を参照してください。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
cycle_years = intseas('year'); put cycle_years;	1
cycle_smiyears = intseas('semiyear'); put cycle_smiyears;	2
cycle_quarters = intseas('quarter'); put cycle_quarters;	4
cycle_number = intseas('month', 'qtr'); put cycle_number;	3
cycle_months = intseas('month'); put cycle_months;	12
cycle_smimonths = intseas('semimonth'); put cycle_smimonths;	24
cycle_tendays = intseas('tenday'); put cycle_tendays;	36
cycle_weeks = intseas('week'); put cycle_weeks;	52
cycle_wkdays = intseas('weekday'); put cycle_wkdays;	5
cycle_hours = intseas('hour'); put cycle_hours;	24

SAS ステートメント	結果
cycle_minutes = intseas('minute'); put cycle_minutes;	60
cycle_month2 = intseas('month2.2'); put cycle_month2;	6
cycle_week2 = intseas('week2'); put cycle_week2;	26
var1 = 'month4.3'; cycle_var1 = intseas(var1); put cycle_var1;	3
cycle_day1 = intseas('day1'); put cycle_day1;	7

関連項目:

関数:

- “INTCYCLE 関数” (552 ページ)
- “INTINDEX 関数” (561 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

構文

INTSHIFT(*間隔* <<*multiple*.<*shift-index*>>>)

必須引数

間隔

WEEK、SEMIYEAR、QTR、HOUR などの時間間隔が含まれる文字定数、変数または式を指定します。*Interval* は、大文字または小文字で表示できます。*interval* に使用可能な値のリストについては、 表 7.3, “日時関数で使用される間隔,” (*SAS 言語リファレンス: 解説編*) を参照してください。

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。間隔名の一般的な形式を次に示します。

interval<multiple.shift-index>

間隔名の 3 つの部分は次のとおりです。

間隔
基本間隔の種類の名前を指定します。たとえば、YEAR で年間隔を指定します。

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。たとえば、間隔 YEAR2 は 2 年の期間の間隔、つまり隔年です。

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” (31 ページ)を参照してください。

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。たとえば、YEAR.3 で、各カレンダ年の 3 月 1 日に開始して翌年の 2 月末に終了するようにシフトされた年間隔を指定します。

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。

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” (31 ページ)を参照してください。

詳細

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、“小売りカレンダの間隔: ISO 8601 遵守” (*SAS 言語リファレンス: 解説編* 7 章) を参照してください。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
shift1 = intshift('year'); put shift1;	MONTH
shift2 = intshift('dtyear'); put shift2;	DTMONTH
shift3 = intshift('minute'); put shift3;	DTMINUTE

SAS ステートメント	結果
interval = 'weekdays'; shift4 = intshift(interval); put shift4;	WEEKDAY
shift5 = intshift('weekday5.4'); put shift5;	WEEKDAY
shift6 = intshift('qtr'); put shift6;	MONTH
shift7 = intshift('dttenday'); put shift7;	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

構文

INTTEST(*間隔*<<*multiple*.<*shift-index*>>>)

必須引数

間隔

WEEK、MONTH または QTR などの間隔名が含まれる文字定数、変数または式を指定します。*Interval* は、大文字または小文字で表示できます。*interval* に使用可能な値のリストについては、 表 7.3, “日時関数で使用される間隔,” (*SAS 言語リファレンス: 解説編*) を参照してください。

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。間隔名の一般的な形式を次に示します。

interval<*multiple.shift-index*>

間隔名の 3 つの部分は次のとおりです。

間隔

基本間隔の種類の名前を指定します。たとえば、YEAR で年間隔を指定します。

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。たとえば、YEAR2 は 2 年の期間で構成されます。

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” (31 ページ)を参照してください。

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。たとえば、YEAR.3 では、各カレンダ

年の 3 月 1 日に開始して翌年の 2 月末に終了するようにシフトされた年間隔を指定します。

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 は無効です。

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。

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" (31 ページ)を参照してください。

詳細

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 言語リファレンス: 解説編*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、"小売りカレンダの間隔: ISO 8601 遵守" (*SAS 言語リファレンス: 解説編* 7 章) を参照してください。

サンプル

次の例では、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。

SAS ステートメント	結果
test1 = inttest('month'); put test1;	1
test2 = inttest('week6.13'); put test2;	1
test3 = inttest('tenday'); put test3;	1
test4 = inttest('twoweeks'); put test4;	0
var1 = 'hour2.2'; test5 = inttest(var1); put test5;	1

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: 切り捨て関数

構文

INTZ (*argument*)

必須引数

引数
数値定数、変数または式です。

詳細

次のルールが適用されます。

- 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。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果

var1=2.1;
a=intz(var1);
put a;
	2

var2=-2.4;
b=intz(var2);
put b;
	-2

var3=1+1.e-11;
c=intz(var3);
put c;
	1

f=intz(-1.6); put f;	-1

関連項目:

関数:

- “CEIL 関数” (288 ページ)
- “CEILZ 関数” (289 ページ)
- “FLOOR 関数” (470 ページ)
- “FLOORZ 関数” (471 ページ)
- “INT 関数” (543 ページ)
- “ROUND 関数” (810 ページ)
- “ROUNDZ 関数” (819 ページ)

IORCMSG 関数

_IORC_の書式化されたエラーメッセージを返します。

カテゴリ: SAS ファイル I/O 関数

構文

IORCMSG()

詳細

まだ長さが割り当てられていない変数に IORCMSG 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 が割り当てられます。

IORCMSG 関数は、自動変数_IORC_
の現在値に関連付けられた、書式化されたエラーメッセージを返します。_
IORC_
変数は、MODIFY ステートメントを使用した場合、またはSET ステートメントにKEY=オプション
IORC_
変数の値は内部値で、SYSRC 自動呼出しマクロと組み合わせて読み取られます。_
IORC_
に特定の値を設定しようとすると、予期しない結果になる可能性があります。

サンプル

次のプログラムでは、銀行口座と現在の銀行預金残高を含む、更新されたマスタファイルにオブザ
IORC_変数をクエリし、_IORC_
値が予期しない値の場合は書式化されたエラーメッセージを返します。

```
libname bank 'SAS-library';
data bank.master(index=(AccountNum));
infile 'external-file-1';
format balance dollar8.;
input @ 1 AccountNum $ 1-3 @ 5 balance 5-9;
run;
data bank.trans(index=(AccountNum));
infile 'external-file-2';
format deposit dollar8.;
```

```
input @ 1 AccountNum $ 1-3 @ 5 deposit 5-9;
run;
data bank.master;
set bank.trans;
modify bank.master key=AccountNum;
if (_IORC_ EQ %sysrc(_SOK)) then
do;
balance=balance+deposit;
replace;
end;
else
if (_IORC_ = %sysrc(_DSENOM)) then
do;
balance=deposit;
output;
_error_=0;
end;
else
do;
errmsg=IORCMSG();
put 'Unknown error condition:'
errmsg;
end;
run;
```

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: 財務関数

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。

period
利息の支払いを計算する支払い期間を指定します。*period* は、*number-of-periods* の値以下の正の整数値で指定する必要があります。

number-of-periods
支払期間の数を指定します。*number-of-periods* は正の整数値とする必要があります。

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。

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。

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。

サンプル

$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。

```
InterestPaid1 = IPMT(0.1/12, 1, 36, 8000);
```

計算によって 66.67 という値が返されます。

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。

- `InterestPaid2 = IPMT(0.1/12, 1, 36, 8000, 0, 1);`

  計算によって 0.0 という値が返されます。

- `InterestPaid3 = IPMT(0.1, 3, 3, 8000);`

  計算によって 292.447 という値が返されます。

- `InterestPaid4 = IPMT(0.09/12, 359, 360, 125000, 0, 1);`

  計算によって 7.4314473 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

構文

IQR(*value-1* <,*value-2*...>)

必須引数

value
四分位範囲を計算する数値定数、変数または式を指定します。

詳細

すべての引数が欠損値の場合、結果は欠損値になります。それ以外の場合、非欠損値の四分位範囲を返します。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*を参照してください。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
iqr=iqr(2,4,1,3,999999); put iqr;	2

関連項目:

関数:

- “MAD 関数” (633 ページ)
- “PCTL 関数” (702 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: 財務関数

構文

IRR(*freqc0,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.

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。

詳細

IRR 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。

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。

関連項目:

関数:

- “INTRR 関数” (574 ページ)

JBESSEL 関数

ベッセル関数の値を返します。

カテゴリ: 数学関数

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)。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=jbessel(2,2);	0.3528340286

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

構文

JULDATE(*date*)

必須引数

date
SAS 日付値を指定します。

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(1～365、またはうるう年の場合は 1～366)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る可能性があり(末尾が 00 の年の場合)、1 桁になる可能性もあります(年の末尾が 01～09 の場合)。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。たとえば、YEARCUTOFF=1920 の場合、JULDATE では 1997 年 1 月 1 日には 97001、1878 年 12 月 31 には 1878365 を返します。

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。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
julian=juldate('31dec99'd);	99365
julian=juldate('01jan2099'd);	2099001

関連項目:

関数:

- “DATEJUL 関数” (352 ページ)
- “JULDATE7 関数” (588 ページ)

システムオプション:

- “YEARCUTOFF=システムオプションの使用” (*SAS 言語リファレンス: 解説編* 7 章)

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

構文

JULDATE7(*date*)

必須引数

date
SAS 日付値を指定します。

詳細

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。

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。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
julian=juldate7('31dec96'd);	1996366
julian=juldate7('01jan2099'd);	2099001

関連項目:

関数:

- “JULDATE 関数” (587 ページ)

KURTOSIS 関数

尖度を返します。

カテゴリ: 記述統計

構文

KURTOSIS(*argument-1*,*argument-2*,*argument-3*,*argument-4*<,…,*argument-n*>)

必須引数

引数
数値の定数、変数または式を指定します。

詳細

少なくとも4 つの非欠損引数が必要です。非欠損引数がない場合は、関数から欠損値が返されます。ERROR_を 1 に設定します。

引数リストには OF で始まる変数のリストを含められます。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x1=kurtosis(5,9,3,6);	0.928
x2=kurtosis(5,8,9,6,.);	-3.3
x3=kurtosis(8,9,6,1);	1.5
x4=kurtosis(8,1,6,1);	-4.483379501
x5=kurtosis(of x1-x4);	-5.065692754

LAG 関数

キューから値を返します。

カテゴリ: 特殊関数

構文

LAG<*n*> (*argument*)

必須引数

引数
数値または文字の定数、変数、式のいずれかを指定します。

オプション引数

n
ラグ値の数を指定します。

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、変数にはデフォルトで長さ 200 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てそのキューに以前格納されていた値を返します。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。

LAG*n* の引数が配列名の場合、配列内の各変数に対して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。

サンプル

サンプル 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。

```
data one;
input x @@;
y=lag1(x);
z=lag2(x);
datalines;
1 2 3 4 5 6
;
proc print data=one;
title 'LAG Output';
run;
```

画面 2.39 *2 つのラグ値の生成からの出力*

LAG Output

Obs	x	y	z
1	1	.	.
2	2	1	.
3	3	2	1
4	4	3	2
5	5	4	3
6	6	5	4

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)。

サンプル 2: BY グループで複数のラグ値を生成する

各 BY グループ内で最大 3 つのラグ値を生成する方法の例を次に示します。

```
/*************************************************************************/
/* This program generates up to three lagged values. By increasing the */
/* size of the array and the number of assignment statements that use */
/* the LAGn functions, you can generate as many lagged values as needed. */
```

```
/*****************************************************************************/
/* Create starting data. */
data old;
input start end;
datalines;
1 1
1 2
1 3
1 4
1 5
1 6
1 7
2 1
2 2
3 1
3 2
3 3
3 4
3 5
;
data new(drop=i count);
set old;
by start;
/* Create and assign values to three new variables. Use ENDLAG1- */
/* ENDLAG3 to store lagged values of END, from the most recent to the */
/* third preceding value. */
array x(*) endlag1-endlag3;
endlag1=lag1(end);
endlag2=lag2(end);
endlag3=lag3(end);
/* Reset COUNT at the start of each new BY-Group */
if first.start then count=1;
/* On each iteration, set to missing array elements */
/* that have not yet received a lagged value for the */
/* current BY-Group. Increase count by 1. */
do i=count to dim(x);
x(i)=.;
end;
count + 1;
run;
proc print;
run;
```

画面 2.40　3 つのラグ値の生成からの出力

The SAS System

Obs	start	end	endlag1	endlag2	endlag3
1	1	1	.	.	.
2	1	2	1	.	.
3	1	3	2	1	.
4	1	4	3	2	1
5	1	5	4	3	2
6	1	6	5	4	3
7	1	7	6	5	4
8	2	1	.	.	.
9	2	2	1	.	.
10	3	1	.	.	.
11	3	2	1	.	.
12	3	3	2	1	.
13	3	4	3	2	1
14	3	5	4	3	2

サンプル 3: 変数の移動平均を計算する

データセット内の変数の移動平均を計算する例を次に示します。

```
/* Title: Compute the moving average of a variable
Goal: Compute the moving average of a variable through the entire data set,
of the last n observations and of the last n observations within a
BY-group.
Input:
*/
data x;
do x=1 to 10;
output;
end;
run;
/* Compute the moving average of the entire data set. */
data avg;
retain s 0;
set x;
s=s+x;
a=s/_n_;
run;
proc print;
run;
/* Compute the moving average of the last 5 observations. */
%let n = 5;
data avg (drop=s);
retain s;
set x;
s = sum (s, x, -lag&n(x)) ;
a = s / min(_n_, &n);
run;
```

```
proc print;
run;
/* Compute the moving average within a BY-group of last n observations.
For the first n-1 observations within the BY-group, the moving average
is set to missing. */
data ds1;
do patient='A','B','C';
do month=1 to 7;
num=int(ranuni(0)*10);
output;
end;
end;
run;
proc sort;
by patient;
%let n = 4;
data ds2;
set ds1;
by patient;
retain num_sum 0;
if first.patient then do;
count=0;
num_sum=0;
end;
count+1;
last&n=lag&n(num);
if count gt &n then num_sum=sum(num_sum,num,-last&n);
else num_sum=sum(num_sum,num);
if count ge &n then mov_aver=num_sum/&n;
else mov_aver=.;
run;
proc print;
run;
```

画面 2.41 *変数の移動平均の計算からの出力*

The SAS System

Obs	s	x	a
1	1	1	1.0
2	3	2	1.5
3	6	3	2.0
4	10	4	2.5
5	15	5	3.0
6	21	6	3.5
7	28	7	4.0
8	36	8	4.5
9	45	9	5.0
10	55	10	5.5

The SAS System

Obs	x	a
1	1	1.0
2	2	1.5
3	3	2.0
4	4	2.5
5	5	3.0
6	6	4.0
7	7	5.0
8	8	6.0
9	9	7.0
10	10	8.0

The SAS System

Obs	patient	month	num	num_sum	count	last4	mov_aver
1	A	1	9	9	1	.	.
2	A	2	0	9	2	.	.
3	A	3	1	10	3	.	.
4	A	4	6	16	4	.	4.00
5	A	5	3	10	5	9	2.50
6	A	6	9	19	6	0	4.75
7	A	7	5	23	7	1	5.75
8	B	1	7	7	1	6	.
9	B	2	8	15	2	3	.
10	B	3	8	23	3	9	.
11	B	4	1	24	4	5	6.00
12	B	5	0	17	5	7	4.25
13	B	6	6	15	6	8	3.75
14	B	7	4	11	7	8	2.75
15	C	1	5	5	1	1	.
16	C	2	7	12	2	0	.
17	C	3	1	13	3	6	.
18	C	4	8	21	4	4	5.25
19	C	5	0	16	5	5	4.00
20	C	6	0	9	6	7	2.25
21	C	7	6	14	7	1	3.50

サンプル 4: フィボナッチ数列を生成する

次の例では、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。

```
data _null_;
put 'Fibonacci Sequence';
n=1;
```

```
f=1;
put n= f=;
do n=2 to 10;
f=sum(f,lag(f));
put n= f=;
end;
run;
```

SAS は次の出力を口グに書き込みます。

```
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
```

サンプル 5: LAG 関数の引数に式を使用する

次のプログラムでは、*argument* の値に式を使用し、X、Y および Z の値を含むデータセットを作成します。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。

```
data one;
input X @@;
Y=lag1(x+10);
Z=lag2(x);
datalines;
1 2 3 4 5 6
;
proc print;
title 'Lag Output: Using an Expression';
run;
```

画面 2.42 *式を使用した LAG 関数からの出力*

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

関連項目:

関数:

- “DIF 関数” (368 ページ)

LARGEST 関数

k 番目に大きい非欠損値を返します。

カテゴリ: 記述統計

構文

LARGEST (*k*, *value-1* <, *value-2*...>)

必須引数

k
返す値を指定する数値定数、変数または式です。

value
処理する数値定数、変数または式の値を指定します。

詳細

k
が欠損値、0 未満、またはvalue の数よりも大きい値の場合、結果は欠損値になり、_ERROR_が 1 に設定されます。または、*k*
が非欠損値の value の数よりも大きい場合、結果は欠損値になりますが、_ERROR_は 1 に設定されません。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果

k=1;
largest1=largest(k, 456, 789, .Q, 123);
put largest1;
	789

k=2;
largest2=largest(k, 456, 789, .Q, 123);
put largest2;
	456

k=3;
largest3=largest(k, 456, 789, .Q, 123);
put largest3;
	123

k=4;
largest4=largest(k, 456, 789, .Q, 123);
put largest4;
	.

関連項目:

関数:

- “ORDINAL 関数” (699 ページ)
- “PCTL 関数” (702 ページ)
- “SMALLEST 関数” (840 ページ)

LBOUND 関数

配列の下限を返します。

カテゴリ: 配列

構文

LBOUND<*n*> (*array-name*)

LBOUND(*array-name*,*bound-n*)

必須引数

array-name
同じ DATA ステップで以前に定義した配列の名前です。

bound-n
下限を確認するディメンションを指定する数値定数、変数または式です。*bound-n* は、*n* を指定していない場合にのみ使用します。

オプション引数

n
下限を確認するディメンションを指定する整数定数です。*n* 値が指定されていない場合、LBOUND 関数は配列の最初のディメンションの下限を返します。

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ディメンションの下限を返します。配列処理で LBOUND を使用することで、配列の限界を変更するたびに反復 DO グループの下限が変更されることを防ぎます。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。

サンプル

サンプル 1: 1 次元配列

この例では、LBOUND はディメンションの下限である 2 の値を返します。SAS は DO ループでステートメントを 5 回繰り返します。

```
 array big{2:6} weight sex height state city;
do i=lbound(big) to hbound(big);
```

```
...more SAS statements...;
end;
```

サンプル 2: 多次元配列

この例では、多次元配列に LBOUND 関数を指定する 2 つの方法を示します。SAS コードの例に続く表に示すように、いずれの方法でも LBOUND で同じ値が返されます。

```
array mult{2:6,4:13,2} mult1-mult100;
```

構文	別の構文	値
LBOUND(MULT)	LBOUND(MULT,1)	2
LBOUND2(MULT)	LBOUND(MULT,2)	4
LBOUND3(MULT)	LBOUND(MULT,3)	1

関連項目:

関数:

- “DIM 関数” (370 ページ)
- “HBOUND 関数” (517 ページ)

ステートメント:

- “ARRAY ステートメント” (*SAS ステートメント: リファレンス*)
- “配列参照ステートメント” (*SAS ステートメント: リファレンス*)
- “配列処理関係の用語” (*SAS 言語リファレンス: 解説編* 23 章)

LCM 関数

最小公倍数を返します。

カテゴリ: 数学関数

構文

LCM(*x1*,*x2*,*x3*, …, *xn*)

必須引数

x
整数値の数値定数、変数または式を指定します。

詳細

LCM (最小公倍数)関数は、数字のセットのすべてのメンバで完全に割り切れる最小の倍数を返します。たとえば、12 と 18 の最小公倍数は 36 です。

いずれかの引数がない場合、戻り値は欠損値になります。

サンプル

次の例では、整数 10 と 15 で完全に割り切れる最小の倍数を返します。

```
data _null_;
x=lcm(10,15);
put x=;
run;
```

SAS は次の出力をログに書き込みます。

```
x=30
```

関連項目:

関数:

- “GCD 関数” (498 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: 組み合わせ関数

構文

LCOMB(*n*,*r*)

必須引数

n
サンプルの選択元となる要素の合計数を表す負でない整数です。

r
選択される要素数を表す負でない整数です。
制限事項: $r \leq n$

比較

LCOMB 関数は、COMB 関数の対数を計算します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=lcomb(5000,500); put x;	1621.4411361

SAS ステートメント	結果
y=lcomb(100,10); put y;	30.482323362

関連項目:

関数:

- “COMB 関数” (303 ページ)

LEFT 関数

文字列を左詰めにします。

カテゴリ: 文字関数

制限事項: I18N レベル 0 の関数は 1 バイト文字セット(SBCS)でのみ使用するように設計されています。

ヒント: DBCS に相当する関数は、KLEFT です。“DBCS の互換性” (601 ページ)を参照してください。

構文

LEFT(*argument*)

必須引数

引数
文字定数、変数または式を指定します。

詳細

基本

DATA ステップで、まだ長さが割り当てられていない変数に LEFT 関数から値が返される場合、その変数に引数の長さが割り当てられます。

LEFT は、先頭の空白を値の末尾に移動した引数を返します。引数の長さは変わりません。

DBCS の互換性

LEFT 関数は、文字列を左詰めにします。ほとんどの場合、LEFT 関数を使用できます。アプリケーションを ASCII 環境で実行できる場合、またはアプリケーションで文字列を操作しない場合、KLEFT 関数ではなく LEFT 関数を使用します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
	----+----1----+
a=' DUE DATE'; b=left(a); put b;	DUE DATE

関連項目:

関数:

- “COMPRESS 関数” (321 ページ)
- “RIGHT 関数” (808 ページ)
- “STRIP 関数” (863 ページ)
- “TRIM 関数” (899 ページ)

LENGTH 関数

末尾の空白を除いた文字列の長さを返します。文字列が空白の場合には、1 を返します。

カテゴリ: 文字関数

制限事項: I18N レベル 0 の関数は 1 バイト文字セット(SBCS)でのみ使用するように設計されています。

ヒント: DBCS に相当する関数は、KLENGTH です。
LENGTH 関数はバイト単位で長さを返し、KLENGTH 関数は文字ベースの単位で長さを返します。

構文

LENGTH(*string*)

必須引数

string
文字定数、変数または式を指定します。

詳細

LENGTH 関数は、*string* 内で最も右にある空白以外の文字の位置を表す整数を返します。*string* の値が空白の場合、LENGTH は値 1 を返します。*string* 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、数値は BEST12.出力形式を使用して自動的に右揃えの文字列に変換されます。この場合、LENGTH は値 12 を返し、SAS ログに数値が文字値に変換されたというメモを書き込みます。

比較

- LENGTH 関数と LENGTHN 関数は、空白以外の文字列に対し同じ値を返します。LENGTH は空白の文字列に値 1 を返すのに対し、LENGTHN は値 0 を返します。

- 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 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、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。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
len=length('ABCDEF'); put len;	6
len2=length(' '); put len2;	1

関連項目:

関数:

- “LENGTHC 関数” (603 ページ)
- “LENGTHM 関数” (604 ページ)
- “LENGTHN 関数” (606 ページ)

LENGTHC 関数

末尾の空白を含めた文字列の長さを返します。

カテゴリ: 文字関数

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

LENGTHC(*string*)

必須引数

string
文字定数、変数または式を指定します。

詳細

LENGTHC 関数は、*string* 内の文字数(空白と空白以外の両方を含む)を返します。*string* は数値定数、変数または式(初期化済みまたは未初期化)で、SAS は BEST12. 出力形式を使用して自動的に数値を右揃えの文字列に変換します。この場合、LENGTHC は値 12 を返し、SAS ログに数値が文字値に変換されたというメモを出力します。

比較

- LENGTHC 関数は末尾の空白を含む文字列の長さを返しますが、LENGTH 関数と LENGTHN 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 LENGTHN の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。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=lengthc('variable with trailing blanks '); put x;	32
length fixed $35; fixed='variable with trailing blanks '; x=lengthc(fixed); put x;	35

関連項目:

関数:

- “LENGTH 関数” (602 ページ)
- “LENGTHM 関数” (604 ページ)
- “LENGTHN 関数” (606 ページ)

LENGTHM 関数

文字列に割り当てられたメモリの量を返します(バイト単位)。

カテゴリ: 文字関数

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

LENGTHM(*string*)

必須引数

string
文字定数、変数または式を指定します。

詳細

LENGTHM 関数は、*string* に割り当てるメモリ量をバイト単位で表す整数を返します。*string* 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、数値は BEST12.出力形式を使用して自動的に右揃えの文字列に変換されます。この場合、LENGTHM は値 12 を返し、SAS ログに数値が文字値に変換されたというメモを書き込みます。

比較

LENGTHM 関数は文字列に割り当てるメモリ量をバイト単位で返すのに対し、LENGTH、LENGTHC および LENGTHN 関数は文字列の長さを返します。LENGTHM は必ず LENGTH、LENGTHC および LENGTHN が返す値以上の値を返します。

サンプル

サンプル 1: 文字式に割り当てるメモリ量を判断する

この例では、文字式の中間結果を保存するバッファに割り当てられたメモリの量(バイト単位)を判断します。式 CAT(x,y)の値の長さがどれくらいになるかは不明なため、最大で 32767 バイトのメモリが割り当てられます。

```
data _null_;
x='x';
y='y';
lc=lengthc(cat(x,y));
lm=lengthm(cat(x,y));
put lc= lm=;
run;
```

SAS は次の出力を ログに書き込みます。

```
lc=2 lm=32767
```

サンプル 2: 外部ファイルの変数に割り当てるメモリ量を判断する

この例では、外部ファイルから SAS ファイルに入力される変数に割り当てるメモリの量(バイト単位)を判断します。

```
data _null_;
file 'test.txt';
put 'trailing blanks ';
run;
data test;
infile 'test.txt';
input;
x=lengthm(_infile_);
put x;
run;
```

次の行が SAS ログに書き込まれます。

```
256
```

関連項目:

関数:

- “LENGTH 関数” (602 ページ)
- “LENGTHC 関数” (603 ページ)
- “LENGTHN 関数” (606 ページ)

LENGTHN 関数

末尾の空白を除いた文字列の長さを返します。

カテゴリ: 文字関数

制限事項: I18N レベル 0 の関数は 1 バイト文字セット(SBCS)でのみ使用するように設計されています。

構文

LENGTHN(*string*)

必須引数

string
文字定数、変数または式を指定します。

詳細

LENGTHN 関数は、*string* 内で最も右にある空白以外の文字の位置を表す整数を返します。*string* の値が空白の場合、LENGTHN は値 0 を返します。*string* 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、数値は BEST12.出力形式を使用して自動的に右揃えの文字列に変換されます。この場合、LENGTHN は値 12 を返し、SAS ログに数値が文字値に変換されたというメモを書き込みます。

比較

- LENGTHN 関数と LENGTH 関数は、空白以外の文字列に対し同じ値を返します。LENGTHN は空白の文字列に値 0 を返すのに対し、LENGTH は値 1 を返します。
- LENGTHN 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。LENGTHN は必ず LENGTHC が返す値以下の値を返します。
- LENGTHN 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、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。LENGTHN は必ず LENGTHM が返す値以下の値を返します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
len=lengthn('ABCDEF'); put len;	6
len2=lengthn(' '); put len2;	0

関連項目:

関数:

- “LENGTH 関数” (602 ページ)
- “LENGTHC 関数” (603 ページ)
- “LENGTHM 関数” (604 ページ)

LEXCOMB 関数

n 個の変数から一度に *k* 個を取り出した非欠損値の、重複しないすべての組み合わせを辞書式順序で生成します。

カテゴリ: 組み合わせ関数

制限事項: LEXCOMB 関数は、%SYSFUNC マクロを使用するときには実行できません。

構文

LEXCOMB(*count*, *k*, *variable-1*, …, *variable-n*)

必須引数

count
ループで 1 ~ 組み合わせ数の値を割り当てる整数変数を指定します。

k
各組み合わせの項目数を指定する、1 ~ *n*(1 と n を含む)の定数、変数または式です。

変数
すべての数値変数または同じ長さのすべての文字変数を指定します。これらの変数の値は置換されます。

要件 LEXCOMB 関数を実行する前にこれらの変数を初期化します。

ヒント: LEXCOMB 関数を実行すると、最初の *k* 個の変数には 1 つの組み合わせの値が含まれます。

詳細

基本

LEXCOMB の第 1 引数が 1 ~ 変数の非欠損値の重複しない組み合わせ数である各整数値をとるループで、LEXCOMB 関数を使用します。このループ内での LEXCOMB の各実行では、*k* は同じ値になります。

組み合わせ数

すべての変数に値が等しくない非欠損値が含まれる場合、組み合わせ数は COMB(*n*,*k*)になります。欠損値を含む変数の数が m 個で、すべての非欠損値が等しくない場合、欠損値は組み合わせから除外されるため、LEXCOMB は COMB(*n*-*m*,*k*)個の組み合わせを生成します。

一部の変数の値が等しい場合、正確な組み合わせ数を計算するのは難しくなりますが、COMB(*n*,*k*)が上限となります。LEXCOMB がゼロ未満の値を返すときにループが終了するのであれば、正確な組み合わせ数を計算する必要はありません。

LEXCOMB 処理

LEXCOMB 関数を初めて実行すると、次のアクションが実行されます。

- 引数の種類と長さに不整合がないか確認が行われます。
- m 個の欠損値は、最後の m 個の引数に割り当てられます。
- *n-m* 個の非欠損値が *count* に続く最初の *n-m* 個の引数に昇順で割り当てられます。
- LEXCOMB は 1 を返します。

後続の実行では、最後の組み合わせに達するまで、次のアクションが実行されます。

- 次の重複しない非欠損値の組み合わせが辞書式順序で生成されます。
- *variable-1* ~ *variable-i* は変更されず、*variable-j* が変更された場合(*j*=*i*+1)、LEXCOMB は *j* を返します。

すべての重複しない組み合わせを生成した後に LEXCOMB 関数を実行すると、LEXCOMB は–1 を返します。

LEXCOMB 関数を実行する場合に第 1 引数の順序が違っていると、その結果は役に立ちません。特に、変数を初期化した直後に第 1 引数 *j* で LEXCOMB 関数を実行すると、*j*番目の組み合わせは取得されません(*j* が 1 の場合を除く)。*j*番目の組み合わせを取得するには、第 1 引数で 1 ~ *j* の値をそのままの順序で取得して、LEXCOMB 関数を *j* 回実行する必要があります。

比較

LEXCOMB 関数は、*n* 個の変数から一度に *k* 個を取り出した非欠損値の重複しないすべての組み合わせを辞書式順序で生成します。ALLCOMB 関数は、n 個の変数値から一度に *k* 個の変数値を取得する場合のすべての組み合わせを、変化量の小さい順に生成します。

サンプル

サンプル 1: 重複しない組み合わせを辞書式順序で生成する

次の例では、LEXCOMB 関数を使用して重複しない組み合わせを辞書式順序で生成します。

```
data _null_;
array x[5] $3 ('ant' 'bee' 'cat' 'dog' 'ewe');
n=dim(x);
k=3;
ncomb=comb(n,k);
do j=1 to ncomb+1;
rc=lexcomb(j, k, of x[*]);
```

```
put j 5. +3 x1-x3 +3 rc=;
if rc<0 then leave;
end;
run;
```

SAS は次の出力を口グに書き込みます。

```
 1 ant bee cat rc=1
2 ant bee dog rc=3
3 ant bee ewe rc=3
4 ant cat dog rc=2
5 ant cat ewe rc=3
6 ant dog ewe rc=2
7 bee cat dog rc=1
8 bee cat ewe rc=3
9 bee dog ewe rc=2
10 cat dog ewe rc=1
11 cat dog ewe rc=-1
```

サンプル 2: 重複しない組み合わせを辞書式順序で生成する: 別の例

LEXCOMB 関数の別の使用例を次に示します。

```
data _null_;
array x[5] $3 ('X' 'Y' 'Z' 'Z' 'Y');
n=dim(x);
k=3;
ncomb=comb(n,k);
do j=1 to ncomb+1;
rc=lexcomb(j, k, of x[*]);
put j 5. +3 x1-x3 +3 rc=;
if rc<0 then leave;
end;
run;
```

SAS は次の出力を口グに書き込みます。

```
 1 X Y Y rc=1
2 X Y Z rc=3
3 X Z Z rc=2
4 Y Y Z rc=1
5 Y Z Z rc=2
6 Y Z Z rc=-1
```

関連項目:

関数:

- “ALLCOMB 関数” (94 ページ)

CALL ルーチン:

- “CALL LEXCOMB ルーチン” (174 ページ)

LEXCOMBI 関数

n 個のオブジェクトを同時に *k* 個使用するときのインデックスのすべての組み合わせを、辞書式順序で生成します。

カテゴリ: 組み合わせ関数

制限事項: LEXCOMBI 関数は、%SYSFUNC マクロを使用するときには実行できません。

構文

LEXCOMBI(*n*, *k*, *index-1*, …, *k*)

必須引数

n
オブジェクトの合計数を指定する数値の定数、変数または式です。

K
各組み合わせのオブジェクト数を指定する数値の定数、変数または式です。

index
返される組み合わせでオブジェクトのインデックスが入る数値変数です。インデックスは 1 ~ *n*(1 と n を含む)の整数です。

ヒント: *index-1* が欠損しているかゼロの場合、LEXCOMBI 関数はインデックスを index-1=1 ~ index-k=k に初期化します。それ以外の場合、LEXCOMBI は組み合わせからインデックスを 1 つ削除し、別のインデックスを追加して、新しい組み合わせを作成します。

詳細

LEXCOMBI 関数を初めて実行する前に、次のタスクのいずれかを実行します。

- *index-1* をゼロまたは欠損値に設定します。
- *index-1* から *index-k* を 1 ~ *n*(1 と n を含む)の重複しない整数に初期化します。

n 個のオブジェクトを同時に *k* 個使用するときの組み合わせ数は、COMB(*n*,*k*)で計算できます。*n* 個のオブジェクトを同時に *k* 個使用するときのすべての組み合わせを生成するには、COMB(*n*,*k*)回実行するループで LEXCOMBI 関数を実行します。

LEXCOMBI 関数で返される値は、どのインデックスが変更されたかを示します(ある場合)。*index-1* ~ *index-i* は変更されず、*index-j* が変更された場合(*j*=*i*+1)、LEXCOMBI は *i* を返します。辞書式順序の最後の組み合わせが生成された後に LEXCOMBI が呼び出されると、LEXCOMBI は–1 を返します。

比較

LEXCOMBI 関数は、*n* 個のオブジェクトを同時に *k* 個使用するときのすべてのインデックスの組み合わせを、辞書式順序で生成します。ALLCOMBI 関数は、*n* 個のオブジェクトを同時に *k* 個使用するときのすべてのインデックスの組み合わせを、変化量の小さい順に生成します。

サンプル

次の例では、LEXCOMBI 関数を使用してインデックスの組み合わせを辞書式順序で生成します。

```
data _null_;
array x[5] $3 ('ant' 'bee' 'cat' 'dog' 'ewe');
array c[3] $3;
array i[3];
n=dim(x);
k=dim(i);
i[1]=0;
ncomb=comb(n,k);
do j=1 to ncomb+1;
rc=lexcombi(n, k, of i[*]);
do h=1 to k;
c[h]=x[i[h]];
end;
put @4 j= @10 'i= ' i[*] +3 'c= ' c[*] +3 rc=;
end;
run;
```

SAS は次の出力をログに書き込みます。

```
 j=1 i= 1 2 3 c= ant bee cat rc=1
j=2 i= 1 2 4 c= ant bee dog rc=3
j=3 i= 1 2 5 c= ant bee ewe rc=3
j=4 i= 1 3 4 c= ant cat dog rc=2
j=5 i= 1 3 5 c= ant cat ewe rc=3
j=6 i= 1 4 5 c= ant dog ewe rc=2
j=7 i= 2 3 4 c= bee cat dog rc=1
j=8 i= 2 3 5 c= bee cat ewe rc=3
j=9 i= 2 4 5 c= bee dog ewe rc=2
j=10 i= 3 4 5 c= cat dog ewe rc=1
j=11 i= 3 4 5 c= cat dog ewe rc=-1
```

関連項目:

CALL ルーチン:

- “CALL LEXCOMBI ルーチン” (177 ページ)
- “CALL ALLCOMBI ルーチン” (151 ページ)

LEXPERK 関数

n 個の変数から一度に *k* 個を取り出した非欠損値の、重複しないすべての順列を辞書式順序で生成します。

カテゴリ: 組み合わせ関数

制限事項: LEXPERK 関数は、%SYSFUNC マクロを使用するときには実行できません。

構文

LEXPERK(*count*, *k*, *variable-1*, …, *variable-n*)

必須引数

count
1 ~ 順列数の範囲の整数の変数を指定します。

k
1 ~ *n*(1 と n を含む)の整数値を指定する数値定数、変数または式です。

変数
すべての数値変数または同じ長さのすべての文字変数を指定します。これらの変数の値は置換されます。

要件 LEXPERK 関数を実行する前にこれらの変数を初期化します。

ヒント: LEXPERK を実行すると、最初の *k* 個の変数には 1 つの順列の値が含まれます。

詳細

基本

LEXPERK の第 1 引数が 1 ~ 変数の *k* 個の非欠損値の重複しない順列数である各整数値をとるループで、LEXPERK 関数を使用します。このループ内での LEXPERK の各実行では、*k* は同じ値になります。

順列数

すべての変数に値の等しくない非欠損値が含まれる場合、順列数は PERM(*n*,*k*)になります。欠損値を含む変数の数が *m* 個で、すべての非欠損値が等しくない場合、欠損値は順列から除外されるため、LEXPERK 関数は PERM(*n*-*m*,*k*)個の順列を生成します。一部の変数の値が等しい場合、正確な順列数を計算するのは難しくなりますが、PERM(*n*,*k*)が上限となります。LEXPERK 関数がゼロ未満の値を返すときにループが終了するのであれば、正確な順列数を計算する必要はありません。

LEXPERK 処理

LEXPERK 関数を初めて実行すると、次のアクションが実行されます。

- 引数の種類と長さに不整合がないか確認が行われます。
- *m* の欠損値が最後の *m* 引数に割り当てられます。
- *n*-*m* 個の非欠損値が *count* に続く最初の *n*-*m* 個の引数に昇順で割り当てられます。
- LEXPERK は 1 を返します。

後続の実行では、最後の順列に達するまで、次のアクションが実行されます。

- *k* 個の非欠損値の重複しない次の組み合わせが辞書式順序で生成されます。
- *variable-1* ~ *variable-i* は変更されず、*variable-j* が変更された場合(*j*=*i*+1)、LEXPERK は *j* を返します。

すべての重複しない順列を生成した後に LEXPERK 関数を実行すると、LEXPERK は–1 を返します。

LEXPERK 関数を実行する場合に第 1 引数の順序が違っていると、その結果は役に立ちません。特に、変数を初期化した直後に第 1 引数 *j* で LEXPERK 関数を実行すると、*j*番目の順列は取得されません(*j* が 1 の場合を除く)。*j*番目の順列を取得するには、第 1 引数で 1 ~ *j* の値をそのままの順序で取得して、LEXPERK 関数を *j* 回実行する必要があります。

比較

LEXPERK 関数は、*n* 個の変数から一度に *k* 個を取り出した非欠損値の、重複しないすべての順列を辞書式順序で生成します。LEXPERK 関数は、*n* 個の変数の非欠損値の重複しないすべての順列を辞書式順序で生成します。ALLPERM 関数は、複数の変数の値のすべての順列を変化量の小さい順に生成します。

サンプル

LEXPERK 関数の例を次に示します。

```
data _null_;
array x[5] $3 ('X' 'Y' 'Z' 'Z' 'Y');
n=dim(x);
k=3;
nperm=perm(n,k);
do j=1 to nperm+1;
rc=lexperk(j, k, of x[*]);
put j 5. +3 x1-x3 +3 rc=;
if rc<0 then leave;
end;
run;
```

SAS は次の出力をログに書き込みます。

```
 1 X Y Y rc=1
2 X Y Z rc=3
3 X Z Y rc=2
4 X Z Z rc=3
5 Y X Y rc=1
6 Y X Z rc=3
7 Y Y X rc=2
8 Y Y Z rc=3
9 Y Z X rc=2
10 Y Z Y rc=3
11 Y Z Z rc=3
12 Z X Y rc=1
13 Z X Z rc=3
14 Z Y X rc=2
15 Z Y Y rc=3
16 Z Y Z rc=3
17 Z Z X rc=2
18 Z Z Y rc=3
19 Z Z Y rc=-1
```

関連項目:

関数:

- “ALLPERM 関数” (96 ページ)
- “LEXPERM 関数” (614 ページ)

CALL ルーチン:

- “CALL RANPERK ルーチン” (220 ページ)
- “CALL RANPERM ルーチン” (222 ページ)

LEXPERM 関数

複数の変数の非欠損値の重複しないすべての順列を辞書式順序で生成します。

カテゴリ: 組み合わせ関数

構文

LEXPERM(*count*, *variable-1* <, …, *variable-N*>)

必須引数

count
1 ~ 順列数の範囲の整数の変数を指定します。

変数
すべての数値変数または同じ長さのすべての文字変数を指定します。これらの変数の値は、LEXPERM によって置換されます。

要件 LEXPERM 関数を実行する前にこれらの変数を初期化します。

詳細

重複しない順列数の決定

後続の式で使用するために定義する変数を次に示します。

N
置換される変数の数(引数の数から 1 を引いた数)を指定します。

M
置換される変数間の欠損値の数を指定します。

d
引数間の重複しない非欠損値数を指定します。

N_i
i=1 ~ i=d で、N_i は i 番目の重複しない値のインスタンス数を指定します。

引数の非欠損値の重複しない順列数は次のように表します。

$$P = \frac{(N_1 + N_2 + \ldots + N_d)!}{N_1 ! \, N_2 ! \ldots N_d!} < = N!$$

注: LEXPERM 関数は、%SYSFUNC マクロを使用して実行できません。

LEXPERM 処理

引数 *count* が 1 ~ P の各整数値をとるループで LEXPERM 関数を使用します。LEXPERM 関数がゼロ未満の値を返すときにループが終了するのであれば、P を計算する必要はありません。

1=*count*<P の場合、次のアクションが実行されます。

- 引数の種類と長さに不整合がないか確認が行われます。
- M 個の欠損値が最後の M 個の引数に割り当てられます。

- N-M 個の非欠損値は、*count* に続く最初の N-M 個の引数に昇順で割り当てられます。
- LEXPERM は 1 を返します。

1<*count*<=P の場合、次のアクションが実行されます。

- 次の重複しない非欠損値の順列が辞書式順序で生成されます。
- *variable-1* ~ *variable-I* は変更されず、*variable-J* が変更された場合(J=I+1)、LEXPERM は J を返します。

count>P の場合、LEXPERM は–1 を返します。

LEXPERM 関数を実行する場合に第 1 引数の順序が間違っていると、その結果は役に立たない場合があります。特に、変数を初期化した直後に第 1 引数 K で LEXPERM 関数を実行すると、K 番目の順列は取得されません(K が 1 の場合を除く)。K 番目の順列を取得するには、第 1 引数で 1 ~ K の値をそのままの順序で取得して、LEXPERM を K 回実行する必要があります。

比較

SAS には、すべての順列を生成する 3 つの関数または CALL ルーチンがあります。

- ALLPERM: 複数の変数の値(*欠損値または非欠損値*)の *考えられる*すべての順列を生成します。各順列は、前の順列に基づいて形成されます(2 つの連続する値を交換)。
- LEXPERM: 複数の変数の *非欠損*値の *重複しない*すべての順列を生成します。順列は、辞書式順序で生成されます。
- LEXPERK: N 個の変数の *非欠損*値から K 個の *重複しない*すべての順列を生成します。順列は、辞書式順序で生成されます。

ALLPERM は最も速い関数および CALL ルーチンです。最も遅いのは LEXPERK です。

サンプル

LEXPERM 関数の例を次に示します。

```
data _null_;
array x[6] $1 ('X' 'Y' 'Z' ' ' 'Z' 'Y');
nfact=fact(dim(x));
put +3 nfact=;
do i=1 to nfact;
rc=lexperm(i, of x[*]);
put i 5. +2 rc= +2 x[*];
if rc<0 then leave;
end;
run;
```

SAS は次の出力を口グに書き込みます。

```
 nfact=720
1 rc=1 X Y Y Z Z
2 rc=3 X Y Z Y Z
3 rc=4 X Y Z Z Y
4 rc=2 X Z Y Y Z
5 rc=4 X Z Y Z Y
6 rc=3 X Z Z Y Y
```

```
7 rc=1 Y X Y Z Z
8 rc=3 Y X Z Y Z
9 rc=4 Y X Z Z Y
10 rc=2 Y Y X Z Z
11 rc=3 Y Y Z X Z
12 rc=4 Y Y Z Z X
13 rc=2 Y Z X Y Z
14 rc=4 Y Z X Z Y
15 rc=3 Y Z Y X Z
16 rc=4 Y Z Y Z X
17 rc=3 Y Z Z X Y
18 rc=4 Y Z Z Y X
19 rc=1 Z X Y Y Z
20 rc=4 Z X Y Z Y
21 rc=3 Z X Z Y Y
22 rc=2 Z Y X Y Z
23 rc=4 Z Y X Z Y
24 rc=3 Z Y Y X Z
25 rc=4 Z Y Y Z X
26 rc=3 Z Y Z X Y
27 rc=4 Z Y Z Y X
28 rc=2 Z Z X Y Y
29 rc=3 Z Z Y X Y
30 rc=4 Z Z Y Y X
31 rc=-1 Z Z Y Y X
```

関連項目:

関数:

- “ALLPERM 関数” (96 ページ)

CALL ルーチン:

- “CALL ALLPERM ルーチン” (153 ページ)
- “CALL RANPERK ルーチン” (220 ページ)
- “CALL RANPERM ルーチン” (222 ページ)

LFACT 関数

FACT(階乗)関数の対数を計算します。

カテゴリ: 組み合わせ関数

構文

LFACT(*n*)

必須引数

n
選択するサンプルの要素の合計数を表す整数です。

詳細

LFACT 関数は FACT 関数の対数を計算します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=lfact(5000); put x;	37591.143509
y=lfact(100); put y;	363.73937556

関連項目:

関数:

- “FACT 関数” (392 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: 数学関数

構文

LGAMMA(*argument*)

必須引数

引数

数値の定数、変数または式を指定します。

範囲: 正にする必要があります。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=lgamma(2);	0
x=lgamma(1.5);	−0.120782238

LIBNAME 関数

SAS ライブラリのライブラリ参照名の割り当てを設定または取り消します。

カテゴリ: SAS ファイル I/O 関数

参照項目: “LIBNAME Function: Windows” in *SAS Companion for Windows*

構文

LIBNAME(*libref*<,*SAS-library*<,*engine*<,*options*>>>)

必須引数

ライブラリ参照名

SAS ライブラリに割り当てるライブラリ参照名を指定する文字定数、変数または式です。

ヒント: *libref* の最大長は 8 文字です。

オプション引数

SAS-library

ライブラリ参照名に関連付ける SAS ライブラリの物理名を指定する文字定数、変数または式です。この名前はホスト動作環境の要件に従って指定します。この引数は NULL にできます。

engine

データライブラリ内で開いた SAS ファイルにアクセスするために使用されるエンジンを指定する文字定数、変数または式です。
SAS/SHARE サーバを指定する場合、engine の値はREMOTE にする必要があります。この引数はにできます。

オプション

指定したエンジンの 1 つ以上の有効なオプションを空白で区切って指定する文字定数、変数または式です。この引数は NULL にできます。

詳細

リターンコードの基本情報

LIBNAME 関数は SAS ライブラリからライブラリ参照名を割り当てたり、割り当てを取り消したりします。LIBNAME 関数で 2 つ以上の引数を使用すると、SAS はライブラリ参照名を割り当てようとします。1 つの引数を使用すると、SAS はライブラリ参照名の割り当てを取り消そうとします。リターンコードは、LIBNAME 関数で使用する引数の値、およびライブラリ参照名が割り当てられるかどうかによって生成されます。

ライブラリ参照名を割り当てるとき、ライブラリ参照名が正常に割り当てられると、リターンコードは 0 になります。リターンコードがゼロ以外で、SYSMSG 関数が警告メッセージまたはメモを返すと、割り当ては成功です。SYSMSG 関数がエラーを返すと、割り当ては失敗です。

ライブラリ参照名がすでに割り当てられていて、ライブラリに別の名前を割り当てようとし、ライブラリ参照名が割り当てられた場合、LIBNAME 関数はゼロ以外のリターンコードを返し、SYSMSG 関数はメモを返します。

LIBNAME に 1 つの引数がある場合

LIBNAME に 1 つの引数がある場合、次のルールが適用されます。

- ライブラリ参照名が割り当てられない場合、ゼロ以外のリターンコードが返され、SYSMSG 関数は警告メッセージを返します。
- ライブラリ参照名が割り当てられると、リターンコード 0 が返され、SYSMSG 関数は空白値を返します。

LIBNAME に 2 つの引数がある場合

LIBNAME に 2 つの引数がある場合、次のルールが適用されます。

- 第 2 引数が NULL、すべて空白または長さゼロの場合、SAS はライブラリ参照名の割り当てを取り消そうとします。
- 第 2 引数が NULL、すべて空白および長さゼロではない場合、SAS はライブラリ参照名に指定したパス(第 2 引数)を割り当てようとします。
- ライブラリ参照名が割り当てられない場合、ゼロ以外のリターンコードが返され、SYSMSG 関数はエラーメッセージを返します。
- ライブラリ参照名が割り当てられると、リターンコード 0 が返され、SYSMSG 関数は空白値を返します。

LIBNAME に 3 つまたは 4 つの引数がある場合

- 第 2 引数が NULL、すべて空白または長さゼロである場合、結果は動作環境によって異なります。
- 第 2 引数が NULL で、ライブラリ参照名がすでに割り当てられていない場合、ゼロ以外のリターンコードが返され、SYSMSG 関数はエラーメッセージを返します。
- 第 2 引数が NULL で、ライブラリ参照名がすでに割り当てられている場合、LIBNAME は値 0 を返し、SYSMSG 関数は空白値を返します。
- 前述の条件の少なくとも 1 つが満たされない場合、SAS はライブラリ参照名に指定したパス(第 2 引数)を割り当てようとします。

注: DATA ステップでは、間に空白を含まない 2 つの連続する引用符で構成される文字定数は、長さ 0 の文字列ではなく単一空白と解釈されます。長さ 0 の文字列を指定するには、TRIM 関数 (899 ページ)を使用します。

動作環境の情報

一部のシステムでは、*SAS-library* 値"(一重引用符で囲んだ空白)でライブラリ参照名を現在のディレクトリに割り当てることができます。その他のシステムでは、*SAS-library* 値に空白のみが含まれている場合、SAS ライブラリからライブラリ参照名の関連付けが取り消されます。*SAS-library* に単一空白が指定されている場合の LIBNAME の動作は、動作環境に依存します。一部の動作環境では、SAS セッション外でシステムコマンドを使用してライブラリ参照名を割り当てることができます。

サンプル

サンプル 1: ライブラリ参照名を割り当てる

この例では、SAS ライブラリ MYLIB に新しいライブラリ参照名 NEW を割り当てます。エラーまたは警告が発生すると、メッセージが SAS ログに書き込まれます。マクロステートメントでは、文字列を引用符で囲みません。

```
%if (%sysfunc(libname(new,MYLIB))) %then
%put %sysfunc(sysmsg());
```

サンプル 2: ライブラリ参照名の割り当てを取り消す

この例では、前述の例でデータライブラリ MYLIB に関連付けたライブラリ参照名 NEW の割り当てを取り消します。エラーまたは警告が発生すると、メッセージが SAS ログに書き込まれます。マクロステートメントでは、文字列を引用符で囲みません。

```
%if (%sysfunc(libname(new))) %then
%put %sysfunc(sysmsg());
```

サンプル 3: ライブラリを圧縮する

この例では、ライブラリ参照名 NEW を MYLIB データライブラリに割り当て、COMPRESS オプションを使用してライブラリを圧縮します。この例では、デフォルトの SAS エンジンを使用します。DATA ステップでは、文字列を引用符で囲みます。

```
data test;
rc=libname('new','MYLIB',,'compress=yes');
run;
```

関連項目:

関数:

- “LIBREF 関数” (620 ページ)
- “SYSMSG 関数” (880 ページ)

LIBREF 関数

ライブラリ参照名が割り当てられていることを確認します。

カテゴリ: SAS ファイル I/O 関数

構文

LIBREF(*libref*)

必須引数

ライブラリ参照名

確認するライブラリ参照名を指定します。DATA ステップの場合、*libref* には文字式、引用符で囲まれた文字列、または値にライブラリ参照名を含む DATA ステップ変数を指定できます。マクロの場合、*libref* には式を指定できます。

範囲: 1 ~ 8 文字

詳細

LIBREF 関数は、ライブラリ参照名が割り当てられている場合は 0、割り当てられていない場合は非ゼロ値を返します。

サンプル

この例では、ライブラリ参照名を確認します。エラーまたは警告が発生すると、メッセージがログに書き込まれます。一部の動作環境では、SAS セッション外でシステムコマンドを使用してライブラリ参照名を割り当てることができます。

```
%if (%sysfunc(libref(sashelp))) %then
%put %sysfunc(sysmsg());
```

関連項目:

関数:

- “LIBNAME 関数” (618 ページ)

LOG 関数

自然(底 e)対数を返します。

カテゴリ: 数学関数

構文

LOG(*argument*)

必須引数

引数

数値の定数、変数または式を指定します。

範囲: 正にする必要があります。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=log(1.0);	0
x=log(10.0);	2.302585093

LOG1PX 関数

1 に引数を加えた値の対数を返します。

カテゴリ: 数学関数

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。

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。

サンプル

サンプル 1: LOG1PX 関数を使用した対数の計算

次の例では、1 に値 0.5 を加えた値の対数を計算します。

```
data _null_;
x=log1px(0.5);
put x=;
run;
```

SAS は次の出力を口グに書き込みます。

```
x=0.4054651081
```

サンプル 2: LOG1PX 関数と LOG 関数の比較

次の例では、LOG1PC 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。

```
data _null_;
x=log1px(1.e-5);
put x= hex16.;
y=log(1+1.e-5);
put y= hex16.;
run;
```

SAS は次の出力を口グに書き込みます。

```
x=3EE4F8AEA9AE7317
y=3EE4F8AEA9AF0A25
```

関連項目:

関数:

- “LOG 関数” (621 ページ)

LOG10 関数

底 10 の対数を返します。

カテゴリ: 数学関数

構文

LOG10(*argument*)

必須引数

引数
数値の定数、変数または式を指定します。
範囲: 正にする必要があります。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=log10(1.0);	0
x=log10(10.0);	1
x=log10(100.0);	2

LOG2 関数

底 2 の対数を返します。

カテゴリ: 数学関数

構文

LOG2(*argument*)

必須引数

引数
数値の定数、変数または式を指定します。
範囲: 正にする必要があります。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=log2(2.0);	1
x=log2(0.5);	−1

LOGBETA 関数

ベータ関数の対数を返します。

カテゴリ: 数学関数

構文

LOGBETA(*a*,*b*)

必須引数

a
第 1 形状パラメータです(*a*>0)。

b
第 2 形状パラメータです(*b*>0)。

詳細

LOGBETA 関数は、式 $log(\beta(a, b)) = log(\Gamma(a)) + log(\Gamma(b)) - log(\Gamma(a + b))$ で数学的に求められます。ここで、$\Gamma(.)$ は gamma 関数です。

式が計算できない場合、LOGBETA は欠損値を返します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
LOGBETA(5,3);	−4.653960350

関連項目:

関数:

- “BETA 関数” (134 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” (273 ページ)

構文

LOGCDF(*'dist'*,*quantile*<,*parm-1*,...,*parm-k*>)

必須引数

'dist'
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
ベルヌーイ	'BERNOULLI'
ベータ	'BETA'
二項	'BINOMIAL'
コーシー	'CAUCHY'
χ2 乗	'CHISQUARE'
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'
ポアソン	'POISSON'

分布	引数
T	'T'
Tweedie	'TWEEDIE'
一様	'UNIFORM'
逆ガウス(Wald)	'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。

等量分類
ランダム変数の値を指定する、数値変数、定数または式です。

オプション引数

parm-1,...,parm-k
特定の分布に適した任意の *shape*、*location* または *scale* パラメータです。

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、“CDF 関数” (273 ページ)を参照してください。

関連項目:

関数:

- “CDF 関数” (273 ページ)
- “LOGPDF 関数” (626 ページ)
- “LOGSDF 関数” (628 ページ)
- “PDF 関数” (703 ページ)
- “SDF 関数” (832 ページ)
- “QUANTILE 関数” (778 ページ)
- “SQUANTILE 関数” (856 ページ)

LOGPDF 関数

確率密度(質量)関数の対数を返します。

カテゴリ: 確率

別名: LOGPMF

参照項目: “PDF 関数” (703 ページ)

構文

LOGPDF(*'dist'*,*quantile*,*parm-1*,...,*parm-k*)

必須引数

'dist'
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
ベルヌーイ	'BERNOULLI'
ベータ	'BETA'
二項	'BINOMIAL'
コーシー	'CAUCHY'
$\chi 2$ 乗	'CHISQUARE'
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'
ポアソン	'POISSON'
T	'T'
Tweedie	'TWEEDIE'

分布	引数
一様	'UNIFORM'
逆ガウス(Wald)	'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。

等量分類
ランダム変数の値を指定する、数値定数、変数または式です。

parm-1,...,parm-k
特定の分布に適した任意の *shape*、*location* または *scale* パラメータです。

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、“PDF 関数” (703 ページ)を参照してください。

関連項目:

関数:

- “CDF 関数” (273 ページ)
- “LOGCDF 関数” (624 ページ)
- “LOGSDF 関数” (628 ページ)
- “PDF 関数” (703 ページ)
- “SDF 関数” (832 ページ)
- “QUANTILE 関数” (778 ページ)
- “SQUANTILE 関数” (856 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” (832 ページ)

構文

LOGSDF(*'dist'*,*quantile*,*parm-1*,...,*parm-k*)

必須引数

'dist'
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
ベルヌーイ	'BERNOULLI'
ベータ	'BETA'
二項	'BINOMIAL'
コーシー	'CAUCHY'
$\chi 2$ 乗	'CHISQUARE'
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'
ポアソン	'POISSON'
T	'T'
Tweedie	'TWEEDIE'
一様	'UNIFORM'
逆ガウス(Wald)	'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。

等量分類
ランダム変数の値を指定する、数値定数、変数または式です。

parm-1,...,parm-k
特定の分布に適した任意の *shape*、*location* または *scale* パラメータです。

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” (832 ページ)を参照してください。

関連項目:

関数:

- “LOGCDF 関数” (624 ページ)
- “LOGPDF 関数” (626 ページ)
- “CDF 関数” (273 ページ)
- “PDF 関数” (703 ページ)
- “SDF 関数” (832 ページ)
- “QUANTILE 関数” (778 ページ)
- “SQUANTILE 関数” (856 ページ)

LOWCASE 関数

引数のすべての文字を小文字に変換します。

カテゴリ: 文字関数

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

LOWCASE(*argument*)

必須引数

引数
文字定数、変数または式を指定します。

詳細

DATA ステップでは、まだ長さが割り当てられていない変数に LOWCASE 関数から値が返される場合、その変数には引数の長さが設定されます。

LOWCASE 関数は、文字引数をコピーし、すべての大文字を小文字に変換して、変更した値を結果として返します。

LOWCASE 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。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x='INTRODUCTION'; y=lowcase(x); put y;	introduction

関連項目:

関数:

- “UPCASE 関数” (902 ページ)
- “PROPCASE 関数” (752 ページ)

LPERM 関数

PERM 関数の対数を計算します。これは、要素数 *r* を含むオプションを使用した *n* 個のオブジェクトの順列数の対数です。

カテゴリ: 組み合わせ関数

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す整数です。

オプション引数

r
選択した要素数を表す任意の整数値です。*r* を省略すると、関数は *n* の階乗を返します。

制限事項: $r \leq n$

詳細

LPERM 関数は、PERM 関数の対数を計算します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=lperm(5000,500); put x;	4232.7715946
y=lperm(100,10); put y;	45.586735935

関連項目:

関数:

- “PERM 関数” (723 ページ)

LPNORM 関数

第 2 引数および後続の非欠損引数の L*p* ノルムを返します。

カテゴリ: 記述統計

構文

LPNORM(*p*, *value-1* <,*value-2* …>)

必須引数

p
1 以上の数値定数、変数または式を指定します。これは、L_p ノルムを計算するための累乗として使用します。

value
数値の定数、変数または式を指定します。

詳細

すべての引数が欠損値の場合、結果は欠損値になります。それ以外の場合、結果は第 2 引数以降の非欠損値の L_p ノルムになります。

次の例では、*p* は第 1 引数の値です。*x*1, *x*2, ..., *xn* はその他の非欠損引数の値です。

$$LPNORM(p, x1, x2, ..., xn) = \left(abs(x1)^p + abs(x2)^p + ... + abs(xn)^p\right)^{1/p}$$

サンプル

サンプル 1: Lp ノルムの計算

次の例では、第 2 引数および後続の非欠損引数の L_p ノルムを返します。

```
data _null_;
x1 = lpnorm(1, ., 3, 0, .q, -4);
x2 = lpnorm(2, ., 3, 0, .q, -4);
x3 = lpnorm(3, ., 3, 0, .q, -4);
x999 = lpnorm(999, ., 3, 0, .q, -4);
put x1= / x2= / x3= / x999=;
run;
```

SAS は次の出力を口グに書き込みます。

```
x1=7
x2=5
x3=4.4979414453
x999=4
```

サンプル 2: 変数リスト使用時の Lp ノルムの計算

次の例では、変数リストを使用して L_p ノルムを返します。

```
data _null_;
x1 = 1;
x2 = 3;
x3 = 4;
x4 = 3;
x5 = 1;
x = lpnorm(of x1-x5);
put x=;
run;
```

SAS は次の出力を口グに書き込みます。

```
x=11
```

関連項目:

関数:

- “SUMABS 関数” (874 ページ) (L1 ノルム)
- “EUCLID 関数” (387 ページ) (L2 ノルム)
- “MAX 関数” (639 ページ) (L 無限ノルム)

MAD 関数

中央値からの平均絶対偏差を返します。

カテゴリ: 記述統計

構文

MAD(*value-1* <, *value-2*...>)

必須引数

value
中央値からの平均絶対偏差を計算する数値定数、変数または式を指定します。

詳細

すべての引数が欠損値の場合、結果は欠損値になります。それ以外の場合は、結果は非欠損値の中央値からの平均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*を参照してください。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
mad=mad(2,4,1,3,5,999999); put mad;	1.5

関連項目:

関数:

- “IQR 関数” (585 ページ)
- “MEDIAN 関数” (643 ページ)
- “PCTL 関数” (702 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: 財務関数

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。

t
有効期限までの時間を指定する非欠損値です。

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。

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。

要件 *sigma1* は、*t* の単位と同じ期間に対して指定する必要があります。

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。

要件 *sigma2* の値は、*t* の単位と同じ期間に対して指定する必要があります。

rho12
第 1 資産と第 2 資産 $\rho_{x_1 x_2}$ の間の相関を指定します。

範囲: -1 ~ 1

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。

$$CALL = X_1 N(d_1) - X_2 N(d_2)$$

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。

$$d_1 = \frac{\left(\ln\left(\frac{X_1}{X_2}\right) + \left(\frac{\sigma^2}{2}\right)t\right)}{\sigma\sqrt{t}}$$

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

$$\sigma^2 = \sigma_{x_1}^2 + \sigma_{x_2}^2 - 2\rho_{x_1, x_2}\sigma_{x_1}\sigma_{x_2}$$

前述の式には次の引数が適用されます。

t
失効日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。

$$CALL = \max\left((X_1 - X_2), 0\right)$$

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” (8 ページ)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値を返します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
	----+----1----+----2--
a=margrclprc(500, .5, 950, 4, 5, 1); put a;	46.441283642
b=margrclprc(850, 1.2, 125, 5, 3, 1); put b;	777.67008185
c=margrclprc(7500, .9, 950, 3, 2, 1); put c;	6562.0354886
d=margrclprc(5000, -.5, 237, 3, 3, 1); put d;	0

関連項目:

関数:

- “MARGRPTPRC 関数” (636 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: 財務関数

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。

t
有効期限までの時間を指定する非欠損値です。

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。

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。
要件 *sigma1* は、*t* の単位と同じ期間に対して指定する必要があります。

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。
要件 *sigma2* の値は、*t* の単位と同じ期間に対して指定する必要があります。

rho12
第 1 資産と第 2 資産 $\rho_{x_1 y_1}$ の間の相関を指定します。

範囲: -1 ~ 1

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。

$$PUT = X_2 N(pd_1) - X_1 N(pd_2)$$

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。

$$pd_1 = \frac{\left(\ln\left(\frac{N_1}{N_2}\right) + \left(\frac{\sigma^2}{2}\right)t\right)}{\sigma\sqrt{t}}$$

$$pd_2 = pd_1 - \sigma\sqrt{t}$$

$$\sigma^2 = \sigma_{x_1}^2 + \sigma_{x_2}^2 - 2\rho_{x_1, x_2}\sigma_{x_1}\sigma_{x_2}$$

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する非欠損値です。

$\sigma_{x_1}^2$
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。

対応する CALL 関係を表示するには、“MARGRCLPRC 関数” (634 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。

$$PUT = \max\left(\left(X_2 - X_1\right), 0\right)$$

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。

価格の基本情報については、“価格関数の使用” (8 ページ)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値を返します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
	----+----1----+---2--
a=margrptprc(500, .5, 950, 4, 5, 1); put a;	496.44128364
b=margrptprc(850, 1.2, 125, 5, 3, 1); put b;	52.670081846
c=margrptprc(7500, .9, 950, 3, 2, 1); put c;	12.035488581
d=margrptprc(5000, -.5, 237, 3, 3, 1); put d;	0

関連項目:

関数:

- “MARGRCLPRC 関数” (634 ページ)

MAX 関数

最大値を返します。

カテゴリ: 記述統計

構文

MAX(*argument-1*,*argument-2*<,...*argument-n*>)

必須引数

引数

数値の定数、変数または式を指定します。少なくとも 2 つの引数が必要です。引数リストには OF で始まる変数のリストを含められます。

比較

MAX 関数はすべての引数が欠損している場合にのみ欠損値(.)を返します。

MAX 演算子(<>)は両方のオペランドが欠損している場合にのみ欠損値を返します。この場合、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。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=max(8,3);	8
x1=max(2,6,.);	6
x2=max(2,-3,1,-1);	2
x3=max(3,.,-3);	3
x4=max(of x1-x3);	6

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返します。

カテゴリ: 文字関数

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。

ヒント: 文字のリテラル文字列を引用符で囲みます。

詳細

返される変数の長さ

DATA ステップでは、まだ長さが割り当てられていない変数に MD5 関数から値が返される場合、その変数には 200 バイトの長さが設定されます。

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。読み取り可能な結果を表示するには、有効な出力形式(hex32.または binary128.など)を指定する必要があります。

動作環境の情報
z/OS 動作環境では、MD5 は ASCII ではなく EBCDIC で出力を生成します。したがって、出力は異なります。

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、データセット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。このダイジェストは、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、ハッシュオブジェクト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。ハッシュオブジェクトの詳細については、“DATA ステップコンポーネントオブジェクト” (*SAS 言語リファレンス: 解説編* 22 章)を参照してください。

サンプル

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。

```
data _null_;
y = md5('abc');
z = md5('access method');
put y= / y = hex32.;
put z= / z = hex32.;
run;
```

このプログラムからの出力には、印刷不可文字が含まれます。

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

構文

MDY(*month*,*day*,*year*)

必須引数

月
1 ~ 12 の整数を表す数値定数、変数または式を指定します。

日
1 ~ 31 の整数を表す数値定数、変数または式を指定します。

年
年を表す 2 桁または 4 桁の整数値で数値定数、変数または式を指定します。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。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
birthday=mdy(8,27,90); put birthday; put birthday= worddate.;	11196 birthday=August 27, 1990
anniversary=mdy(7,11,2001); put anniversary; put anniversary=date9.;	15167 anniversary=11JUL2001

関連項目:

関数:

- “DAY 関数” (354 ページ)
- “MONTH 関数” (652 ページ)
- “YEAR 関数” (955 ページ)

MEAN 関数

算術平均(平均)を返します。

カテゴリ: 記述統計

構文

MEAN(*argument-1*<,...*argument-n*>)

必須引数

引数

数値の定数、変数または式を指定します。少なくとも 1 つの非欠損引数が必要です。非欠損引数がない場合は、関数から欠損値が返されます。

ヒント: 引数リストには OF で始まる変数のリストを含められます。

詳細

GEOMEAN 関数は幾何平均を返し、HARMEAN 関数は調和平均を返し、MEDIAN 関数は非欠損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。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x1=mean(2,.,.,6);	4
x2=mean(1,2,3,2);	2
x3=mean(of x1-x2);	3

関連項目:

関数:

- “GEOMEAN 関数” (501 ページ)
- “GEOMEANZ 関数” (502 ページ)
- “HARMEAN 関数” (515 ページ)
- “HARMEANZ 関数” (516 ページ)
- “MEDIAN 関数” (643 ページ)

MEDIAN 関数

中央値を返します。

カテゴリ: 記述統計

構文

MEDIAN(*value1* <, *value2*, ...>)

必須引数

value
数値定数、変数または式です。

詳細

MEDIAN 関数は、非欠損値の中央値を返します。すべての引数が欠損値の場合、結果は欠損値になります。

注: MEDIAN 関数で使用する式は、PROC UNIVARIATE で使用する計算式と同じです。詳細については、SAS Elementary Statistics Procedures を参照してください。

比較

MEDIAN 関数は非欠損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。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=median(2,4,1,3);	2.5
y=median(5,8,0,3,4);	4

関連項目:

関数:

- “MEAN 関数” (642 ページ)

MIN 関数

最小値を返します。

カテゴリ: 記述統計

構文

MIN(*argument-1*,*argument-2*<,...*argument-n*>)

必須引数

引数
数値の定数、変数または式を指定します。少なくとも 2 つの引数が必要です。引数リストには OF で始まる変数のリストを含められます。

詳細

MIN 関数はすべての引数が欠損している場合にのみ欠損値(.)を返します。

MIN 演算子(><)は一方のオペランドが欠損している場合に欠損値を返します。この場合、欠損値の並べ替え順がより下位のオペランドの値が返されます。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=min(7,4);	4
x1=min(2,.,6);	2
x2=min(2,-3,1,-1);	-3
x3=min(0,4);	0
x4=min(of x1-x3);	-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

構文

MINUTE(*time* | *datetime*)

必須引数

time
SAS 時間値を指定する数値定数、変数または式です。

datetime
SAS 日時値を指定する数値定数、変数または式です。

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 ~ 59 の範囲の正の値を返します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果

time='3:19:24't;
m=minute(time);
put m;
	19

関連項目:

関数:

- “HOUR 関数” (522 ページ)
- “SECOND 関数” (835 ページ)

MISSING 関数

引数が欠損値を含むかどうかの結果を表す数値を返します。

カテゴリ: 記述統計
文字関数

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

MISSING(*numeric-expression* | *character-expression*)

必須引数

numeric-expression
数値の定数、変数または式を指定します。

character-expression
文字定数、変数または式を指定します。

詳細

- MISSING 関数は、数値式または文字式に欠損値がないかを確認し、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。
- 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みなされます。

- 文字式の評価結果がすべて空白の文字列か長さがゼロの文字列の値を求める場合、その文字式は欠損とみなされます。

比較

MISSING 関数には引数を 1 つのみ指定できます。CMISS 関数には複数の引数を指定でき、欠損値の数が返されます。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。

サンプル

次の例では、MISSING 関数を使用して入力変数に欠損値が含まれているかどうかを確認します。

```
data values;
input @1 var1 3. @5 var2 3.;
if missing(var1) then
do;
put 'Variable 1 is Missing.';
end;
else if missing(var2) then
do;
put 'Variable 2 is Missing.';
end;
datalines;
127
988 195
;
run;
```

SAS は次の出力をログに書き込みます。

```
Variable 2 is Missing.
```

関連項目:

関数:

- “CMISS 関数” (297 ページ)
- “NMISS 関数” (666 ページ)

CALL ルーチン:

- “CALL MISSING ルーチン” (188 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: 数学関数

構文

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

必須引数

argument-1
除数を指定する数値定数、変数または式です。

argument-2
除数を指定する数値定数、変数または式です。

制限事項: 0 にはできません

詳細

MOD 関数は、*argument-1* を *argument-2* 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。

  正確な計算で最大整数を判定するには、次の DATA ステップを実行します。

```
data _null_;
exactint = constant('exactint');
put exactint=;
run;
```

  動作環境の情報
  最大整数の詳細については、現在の動作環境向けの SAS ドキュメントを参照してください。

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。この場合の例を次に示します。

- 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。

注: SAS 9 より前のバージョンでは、MOD 関数は前述した調整を余りに対して実行しませんでした。このため、SAS 9 での MOD 関数の結果は、前のバージョンとは異なる場合があります。

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 関数はどちらも欠損値を返します。

サンプル

次の SAS ステートメントは、MOD と MODZ の結果を生成します。

SAS ステートメント	結果
x1=mod(10,3); put x1 9.4;	1.0000
xa=modz(10,3); put xa 9.4;	1.0000
x2=mod(.3,-.1); put x2 9.4;	0.0000
xb=modz(.3,-.1); put xb 9.4;	0.1000
x3=mod(1.7,.1); put x3 9.4;	0.0000
xc=modz(1.7,.1); put xc 9.4;	0.0000
x4=mod(.9,.3); put x4 24.20;	0.00000000000000000000
xd=modz(.9,.3); put xd 24.20;	0.00000000000000005551

関連項目:

関数:

- “INT 関数” (543 ページ)
- “INTZ 関数” (582 ページ)
- “MODZ 関数” (650 ページ)

MODEXIST 関数

インストールされている SAS のバージョンにソフトウェアイメージが存在するかどうかを判断します。

カテゴリ: 数値

構文

MODEXIST('*product-name*')

必須引数

'product-name'
確認する製品イメージの名前である文字定数、変数または式を指定します。

詳細

MODEXIST 関数は、インストールされている SAS のバージョンにソフトウェアイメージが存在するかどうかを判断します。イメージが存在する場合、MODEXIST は値 1 を返します。イメージが存在しない場合、MODEXIST は値 0 を返します。

比較

MODEXIST 関数は、インストールされている SAS のバージョンにソフトウェアイメージが存在するかどうかを判断します。SYSPROD 関数は、製品のライセンスがあるかどうかを判断します。

サンプル

この例では、製品のライセンスがあるかどうかと、イメージがインストールされているかどうかを判断します。
SAS/GRAPH イメージが現在のSAS バージョンにインストールされている場合は値1、イメージが関数は、製品のライセンスがあるかどうかを判断します。

```
data _null_;
rc1 = sysprod('graph');
rc2 = modexist('sasgplot');
put rc1= rc2=;
run;
```

ログ 2.14 *MODEXIST からの出力*

```
rc1=1 rc2=1
```

MODULEC 関数

外部ルーチンを呼び出し、文字値を返します。

カテゴリ: 外部ルーチン

参照項目: “CALL MODULE ルーチン” (189 ページ)

構文

MODULEC(<*cntl-string*,> *module-name*<,*argument-1, ..., argument-n*>)

詳細

MODULEC 関数の詳細については、“CALL MODULE ルーチン” (189 ページ)を参照してください。

関連項目:

関数:

- “MODULEN 関数” (650 ページ)

CALL ルーチン:

- “CALL MODULE ルーチン” (189 ページ)

MODULEN 関数

外部ルーチンを呼び出し、数値を返します。

カテゴリ: 外部ルーチン

参照項目: “CALL MODULE ルーチン” (189 ページ)

構文

MODULEN(<*cntl-string*,> *module-name*<,*argument-1, ..., argument-n*>)

詳細

MODULEN 関数の詳細については、“CALL MODULE ルーチン” (189 ページ)を参照してください。

関連項目:

関数:

- “MODULEC 関数” (649 ページ)

CALL ルーチン:

- “CALL MODULE ルーチン” (189 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: 数学関数

構文

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

必須引数

argument-1
除数を指定する数値定数、変数または式です。

argument-2
除数を指定するゼロ以外の数値定数、変数または式です。

詳細

MODZ 関数は、*argument-1* を *argument-2* 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。

  正確な計算で最大整数を判定するには、次の DATA ステップを実行します。

```
data _null_;
exactint = constant('exactint');
put exactint=;
run;
```

  動作環境の情報
  最大整数の詳細については、現在の動作環境向けの SAS ドキュメントを参照してください。

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)。
- 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 関数はどちらも欠損値を返す。

サンプル

次の SAS ステートメントは、MOD と MODZ のこれらの結果を生成します。

SAS ステートメント	結果
x1=mod(10,3); put x1 9.4;	1.0000
xa=modz(10,3); put xa 9.4;	1.0000
x2=mod(.3,-.1); put x2 9.4;	0.0000
xb=modz(.3,-.1); put xb 9.4;	0.1000

SAS ステートメント	結果
x3=mod(1.7,.1); put x3 9.4;	0.0000
xc=modz(1.7,.1); put xc 9.4;	0.0000
x4=mod(.9,.3); put x4 24.20;	0.00000000000000000000
xd=modz(.9,.3); put xd 24.20;	0.00000000000000005551

関連項目:

関数:

- “INT 関数” (543 ページ)
- “INTZ 関数” (582 ページ)
- “MOD 関数” (646 ページ)

MONTH 関数

SAS 日付値から月を返します。

カテゴリ: 日付と時間

構文

MONTH(*date*)

必須引数

date
SAS 日付値を表す数値定数、変数または式を指定します。

詳細

MONTH 関数は、SAS 日付値から月を表す数値を返します。 1-12 の数値が表示されます。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果

date='25jan94'd;
m=month(date);
put m;
	1

関連項目:

関数:

- “DAY 関数” (354 ページ)
- “YEAR 関数” (955 ページ)

MOPEN 関数

ディレクトリ ID とメンバ名でファイルを開き、ファイル ID または 0 のいずれかを返します。

カテゴリ: 外部ファイル

参照項目: “MOPEN Function: UNIX” in *SAS Companion for UNIX Environments*
“MOPEN Function: z/OS” in *SAS Companion for z/OS*

構文

MOPEN(*directory-id*,*member-name*<,*open-mode*<,*record-length*<,*record-format*>>>)

必須引数

directory-id
ディレクトリが(通常は DOPEN 関数により)開かれたときに割り当てられた ID を指定する数値変数です。

member-name
ディレクトリ内のメンバ名を指定する文字定数、変数または式です。

オプション引数

open-mode
ファイルへのアクセスの種類を指定する文字定数、変数または式です。

A APPEND モード: 現在のファイルの最後の後に新しいレコードを書き込むことができます。

I INPUT モード: 読み取り専用です(デフォルト)。

O OUTPUT モード: FILENAME ステートメントまたは関数の動作環境オプションで指定したデフォルトの OPEN モードになります。動作環境オプションが指定されていない場合、ファイルの先頭に新しいレコードを書き込むことができます。

S Sequential Input モード: パイプや他のシーケンシャルデバイス(ハードウェアポートなど)に使用されます。

U UPDATE モード: 読み込みと書き込みの両方ができます。

W シーケンシャル更新モードは、パイプおよびその他のシーケンシャルデバイス(ポートなど)に使用します。

デフォルト: I

record-length
ファイルの新しい論理レコード長を指定する、数値変数、数値定数または数値式です。ファイルの既存のレコード長を使用するには、0 の長さを指定するか、ここでは値を指定しません。

record-format
ファイルの新しいレコード出力形式を指定する、文字定数、変数または式です。既存のレコード形式を使用する場合は、ここでは値を指定しません。有効な値は次のとおりです。

B データがバイナリデータとして解釈されることを指定します。

D デフォルトのレコード出力形式を指定します。

E 編集可能なレコード出力形式を指定します。

F ファイルに固定長のレコードが含まれることを指定します。

P ファイルに動作環境に依存するレコード出力形式でプリンタキャリッジ制御が含まれることを指定します。

V ファイルに可変長のレコードが含まれることを指定します。

注:
引数が無効の場合、MOPEN は0 を返します。SYSMSG 関数から対応するエラーメッセージ
ERROR_自動変数も設定されません。

詳細

MOPEN はファイルの ID を返します。ファイルを開けなかった場合は 0 を返します。MOPEN によって返される *file-id* は、FOPEN 関数によって返される *file-id* と同じように使用できます。

注意:
OUTPUT モードを使用する場合は注意が必要です。 出力する既存のファイルを開くと、ファイルの現在の内容が警告なしに上書きされる場合があります。

メンバは、ファイル参照名ではなく、*directory-id* と *member-name* で識別されます。また、FILENAME を使用してディレクトリメンバを開いてメンバにファイル参照名を割り当て、その後に FOPEN を呼び出すことができます。ただし、MOPEN を使用するときには、各メンバに別個のファイル参照名を使用する必要がありません。

ファイルがすでに存在する場合、出力モードと更新モードは、FILENAME ステートメントまたは関数で指定された動作環境オプション(追加または置換)にデフォルト設定されます。次に例を示します。

```
%let rc=%sysfunc(filename(file,physical-name,,mod));
%let did=%sysfunc(dopen(&file));
%let fid=%sysfunc(mopen(&did,member-name,o,0,d));
%let rc=%sysfunc(fput(&fid,This is a test.));
%let rc=%sysfunc(fwrite(&fid));
%let rc=%sysfunc(fclose(&fid));
```

'file'がすでに存在する場合、FWRITE はファイルの先頭に書き込むかわりに、新しいレコードを追加します。ただし、FILENAME 関数で動作環境オプションが指定されていない場合、出力モードはレコードが置換されることを示します。

開けなかった場合は、SYSMSG を使用してメッセージテキストを取得します。

動作環境の情報

この説明中の*ディレクトリ*という用語は、動作環境で管理されるファイルをグループ化した集合を指します。これらのグループは、ホストの動作環境によってはディレクトリ、サブディレクトリ、フォルダ、MACLIB、分割データセットなど、さまざまな名前で識別されます。詳細については、動作環境に関する SAS のドキュメントを参照してください。一部の動作環境では、出力または追加するディレクトリメンバを開くことができません。

サンプル

この例では、ファイル参照名 MYDIR をディレクトリに割り当てます。ディレクトリを開き、メンバ数を判断し、最初のメンバの名前を取得し、そのメンバを開きます。MOPEN の最後の 3 つの引数はデフォルトです。マクロステートメントでは、文字列を引用符で囲みません。

```
%let filrf=mydir;
%let rc=%sysfunc(filename(filrf,physical-name));
%let did=%sysfunc(dopen(&filrf));
%let frstname=' ';
%let memcount=%sysfunc(dnum(&did));
%if (&memcount > 0) %then
%do;
%let frstname =
%sysfunc(dread(&did,1));
%let fid =
%sysfunc(mopen(&did,&frstname,i,0,d));
macro statements to process the member
%let rc=%sysfunc(fclose(&fid));
%end;
%else
%put %sysfunc(sysmsg());
%let rc=%sysfunc(dclose(&did));
```

関連項目:

関数:

- “DCLOSE 関数” (354 ページ)
- “DNUM 関数” (374 ページ)
- “DOPEN 関数” (375 ページ)
- “DREAD 関数” (379 ページ)
- “FCLOSE 関数” (394 ページ)
- “FILENAME 関数” (404 ページ)
- “FOPEN 関数” (476 ページ)
- “FPUT 関数” (484 ページ)
- “FWRITE 関数” (491 ページ)
- “SYSMSG 関数” (880 ページ)

MORT 関数

割賦返済パラメータを返します。

カテゴリ: 財務関数

構文

MORT(*a*,*p*,*r*,*n*)

必須引数

a
数値で初期の金額を指定します。

p
定期的支払いを指定する数値です。

r
分数で表す定期的な利率を指定する数値です。

n
整数で複利計算を行う期間数を指定します。
範囲: $n \geq 0$

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
ERROR_を 1 に設定します。

- rate < –1 または 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

サンプル

次のステートメントでは、$50,000 を毎月複利計算される年利 10%の 30 年で借入します。毎月の支払いは次のように表すことができます。

```
payment=mort(50000, . , .10/12,30*12);
```

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 パーセントは、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。

MSPLINT 関数

単調性維持スプライン補間の縦座標を返します。

カテゴリ: 数学関数

構文

MSPLINT(*X*, *n*, X_1 <, X_2, …, X_n> , Y_1 <,Y_2, …, Y_n> <,D_1,D_n>)

必須引数

X
スプラインの縦座標を計算する横座標を指定する数値定数、変数または式を指定します。

n
ノット数を指定する数値定数、変数または式です。*N* は正の整数にする必要があります。

X_1, …, X_n
ノットの横座標を指定する数値定数、変数または式です。これらの値は欠損値で、非減少順に並べる必要があります。それ以外の場合、結果は未定義になります。MSPLINT では X_1 ~ X_n 引数の順序は確認されません。

Y_1, ..., Y_n
ノットの座標を指定する数値定数、変数または式です。Y_1 ~ Y_n 引数の数は、X_1 ~ X_n 引数の数と同じにする必要があります。

オプション引数

D_1, D_n
X_1 と X_n のスプラインの導関数を指定する任意の数値定数、変数または式です。これらの導関数は、X_2 より小さい、または X_{n-1} より大きい横座標にのみ影響します。

詳細

MSPLINT 関数は、単一横座標 *X* の単調性維持スプライン補間の縦座標を返します。

スプライン補間は、順序対(X_1, Y_1)、(X_2, Y_2)、…、(X_n, Y_n)で指定する各点を通る関数です。これらの点はノットと呼ばれます。

次の両方の条件を満たすと、スプラインは単調性を維持します。

- 非減少座標の 2 つ以上の連続するノットに対し、間隔内のすべての補間値も非減少になる。

- 非増加座標の 2 つ以上の連続するノットに対し、間隔内のすべての補間値も非増加になる。

ただし、D_1 または D_n の値を誤った符号で指定すると、X_2 より小さいまたは X_{n-1} より大きい値の単調性は維持されません。

引数 D_1 と D_n を省略または欠損すると、次のアクションが実行されます。

- n=1 に対し、MSPLINT は Y_1 を返す。
- n=2 に対し、MSPLINT は線形補間または補外を使用する。

引数 D_1 と D_n が非欠損値、または n>=3 のとき、次のアクションが実行されます。

- $X<X_1$ または $X>X_n$ のとき、MSPLINT は線形補外を使用する。
- $X_1<=X<=X_n$ のとき、MSPLINT は三次スプライン補間を使用する。

2 つのノットの横座標が等しいけれども縦座標が異なる場合、スプラインはその横座標で不連続になります。2 つのノットの横座標が等しく、縦座標が等しい場合、スプラインはその横座標で連続になりますが、通常第 1 導関数はその横座標で非連続になります。それ以外の場合、スプラインは連続で、第 1 導関数は連続になります。

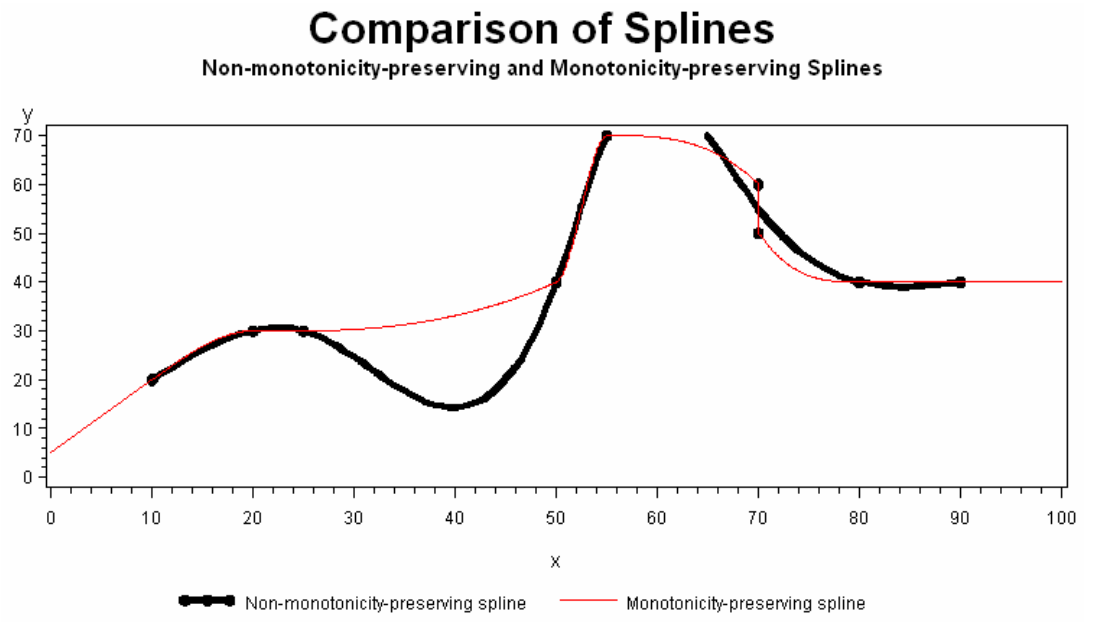
X が欠損している、または結果を計算するために必要なその他の引数が欠損していると、MSPLINT は欠損値を返します。MSPLINT では、欠損値の引数のすべては確認されません。引数 D_1 と D_n は任意であり、結果を計算するために必要ではないため、1 つまたは両方が欠損しており、エラーが発生しない場合には、MSPLINT は非欠損の結果を返します。

サンプル

MSPLINT 関数の例を次に示します。

```
data msplint;
do x=0 to 100 by .1;
msplint=msplint(x, 9,
10, 20, 25, 50, 55, 70, 70, 80, 90,
20, 30, 30, 40, 70, 60, 50, 40, 40);
output;
end;
run;
data knots;
input x y;
datalines;
10 20
20 30
25 30
50 40
55 70
70 60
70 50
80 40
90 40
;
data plot;
merge knots msplint;
by x;
run;
```

```
title "Comparison of Splines";
title2 "Non-monotonicity-preserving and Monotonicity-preserving
Splines";
legend1 value=('Non-monotonicity-preserving spline'
'Monotonicity-preserving spline') label=none;
symbol1 value=dot interpol=spline color=black width=5;
symbol2 value=none interpol=join color=red;
proc gplot data=plot;
plot y*x=1 msplint*x=2/overlay legend=legend1;
run;
quit;
```



リファレンス

Fritsch, F. N., and J. Butland. “A method for constructing local monotone piecewise cubic interpolants.” 1984. *Siam Journal of Scientific and Statistical Computing* 5:2: 300-304.

MVALID 関数

SAS メンバ名として使用する文字列の有効性を確認します。

カテゴリ: 文字関数

構文

MVALID(*libname*, *string*, *memtype*<,*validmemname*>)

必須引数

libname
SAS ライブラリとライブラリ参照名を関連付ける文字定数、変数または式を指定します。先頭と末尾の空白は無視されます。

string
値を SAS メンバ名として使用できるかどうかを判断するために確認する文字定数、変数または式を指定します。先頭と末尾の空白は無視されます。

memtype
使用するメンバ名のメンバの種類である文字定数、変数または式を指定します。先頭と末尾の空白は無視されます。*memtype* の値は検証されません。次のメンバの種類を使用できます。

ACCESS	SAS/ACCESS で作成するアクセスディスクリプタファイルを指定します。
CATALOG	SAS カタログを指定します。
DATA	SAS データファイルを指定します。
FDB	財務データベースを指定します。
ITEMSTOR	個別にアクセス可能な情報で構成される SAS データセットを指定します。SAS Registry は項目ストアの例です。
MDDB	多次元データベースを指定します。
PROGRAM	格納されているコンパイルした SAS プログラムを指定します。
VIEW	SAS ビューを指定します。

オプション引数

validmemname
文字定数、変数または式を指定します。*validmemname* の値には、大文字と小文字を使用できます。先頭と末尾の空白は無視されます。*validmemname* に使用できる値を次に示します。

COMPAT
COMPATIBLE
次の 3 つの条件すべてを満たすとき、*string* は有効な SAS メンバ名であると判断されます。

- *string* 引数が英文字またはアンダースコアで始まる。
- それ以降の文字はすべて英文字、アンダースコアまたは数字である。
- *string* に 32 文字以下の英数字が使用されている。

EXTEND
次の条件すべてを満たすとき、*string* は有効な SAS メンバ名であると判断されます。

- *string* の長さが 32 バイト以下である。
- *string* 引数に/ \ * ? " < > | : –が使用されていない。

  注: SPD Engine では、さらに'$'を最初の文字に使用できません。またメンバ名にはピリオド(.)を使用できません。
- *string* 引数に NULL のバイトが使用されていない。
- *string* 引数が空白またはピリオド(.)で始まらない。

- *string* 引数に少なくとも 1 文字が使用されている。すべて空白である名前は無効。

デフォルト: VALIDMEMNAME=は COMPAT に設定されます。

注: 値が指定されていない場合、MVALUE 関数は VALIDMEMNAME=システムオプションの値に基づいて *string* が有効な SAS メンバ名であることを判断します。

詳細

基本

MVALID 関数は、*string* の値を確認し、SAS メンバ名として使用できるかどうかを判断します。

MVALID 関数は、*string* を SAS メンバ名として使用できる場合は値 1、*string* を SAS メンバ名として使用できない場合は値 0 を返します。

次の条件のいずれかを満たすと、MVALID は欠損値を返します。

- *libname* 引数が、割り当てられているライブラリ参照名ではない。
- *memtype* 引数が 9 文字を超えている。
- *validmemname* 引数の値に COMPATIBLE、COMPAT または EXTEND のいずれかが使用されていない(小文字と大文字に関係なく)。

SAS メンバ名の検証の要件

string 引数は、有効な SAS メンバ名であるかどうかを判断するために評価されます。エンジン名、関連付けられているライブラリおよびメンバの種類が *string* の検証に影響します。メンバの種類のうち、DATA、ITEMSTOR および VIEW のみが名前に拡張文字を使用できます。*string* が評価されるとき、任意の *validmemname* 引数の EXTEND 値は考慮されません。すべてのエンジンで *validmemname* 処理がサポートされているわけではありません。サポートされていないエンジンの場合、*string* はそのエンジンのルールに基づいて検証されます。

次の例では、MVALID 関数を使用して、エンジン名、DATA メンバの種類、*validmemname* の EXTEND 値に基づいて *string* が有効な SAS メンバ名であるかどうかを判断する方法を示します。

```
libname V9eng V9 'mypath';
data _null_;
rc=MVALID('V9eng', 'my name', 'data', 'extend');
put rc=;
run;
```

次の項目が前の例に適用されます。

- メンバの種類が DATA に等しく、*validmemname* が EXTEND に等しいとき、'*my name*'が V9 エンジンの有効なメンバ名であることを示す値 1 が返される。
- この例で V6 エンジンを使用すると、メンバの種類が DATA と等しく、*validmemname* が EXTEND に等しいとき、'*my name*'が無効であることを示す値 0 が返される。V6 エンジンでは *validmemname* 処理がサポートされていない。

次の例では、メンバの種類に DATA のかわりに CATALOG を使用します。

```
libname V9eng V9 'mypath';
data _null_;
rc=MVALID('V9eng', 'my name', 'catalog', 'extend');
```

```
put rc=;
run;
```

次の項目が前の例に適用されます。

- この例で DATA のかわりに CATALOG を使用すると、メンバの種類が CATALOG に等しく、*validmemname* が EXTEND に等しいとき、'*my name*'が無効であることを示す値 0 が返される。メンバの種類 CATALOG では拡張名がサポートされていないため、*validmemname* の EXTEND 値は無効。
- この例で EXTEND のかわりに COMPAT を使用すると、メンバの種類が CATALOG に等しく、*validmemname* が COMPAT に等しいとき、'*my name*'が無効であることを示す値 0 が返される。*validmemname* の COMPAT 値では、メンバ名に空白を使用できない。

N 関数

数値の非欠損値の数を返します。

カテゴリ: 記述統計

構文

N(*argument-1*<,...*argument-n*>)

必須引数

引数

数値の定数、変数または式を指定します。少なくとも 1 つの引数が必要です。引数リストには OF で始まる変数のリストを含められます。

比較

N 関数は非欠損値を数えますが、NMISS および CMISS 関数は欠損値を数えます。N では数値の引数が必要ですが、CMISS は数値と文字値の両方に対応します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x1=n(1,0,.,2,5,.);	4
x2=n(1,2);	2
x3=n(of x1-x2);	2

NETPV 関数

現在正味価値をパーセントで返します。

カテゴリ: 財務関数

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

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

注: ケース *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。

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ように求められます。

$$NETPV(r,\ freq,\ c_0,\ c_1,\ ...,\ c_n) = \sum_{i=0}^{n} c_i x^i$$

前述の式には次の関係が適用されます。

$$x = \begin{cases} \dfrac{1}{(1+r)^{(1/freq)}} & freq > 0 \\ \varepsilon^{-r} & freq = 0 \end{cases}$$

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。

サンプル

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。

```
value=netpv(.10,.5,-500,200,300,400);
```

返される値は 95.98 です。

NLITERAL 関数

指定した文字列を SAS 名前リテラルに変換します。

カテゴリ: 文字関数

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

NLITERAL(*string*)

必須引数

string
SAS 名リテラルに変換する文字定数、変数または式を指定します。

制限事項: 文字列が有効な SAS 変数名であれば、変更されません。

ヒント: 文字のリテラル文字列を引用符で囲みます。

詳細

返される変数の長さ

DATA ステップでは、まだ長さが割り当てられていない変数に NLITERAL 関数から値が返される場合、その変数には 200 バイトの長さが設定されます。

基本

String は、デフォルトのルールで SAS 変数名とみなされない限り、名前リテラルに変換されます。これらのデフォルトのルールは、SAS システムオプションの VALIDVARNAME=V7 が次の場合に適用されます。

- 英文字またはアンダースコアで始まる。
- それ以降の文字はすべて英文字、アンダースコアまたは数字である。
- 32 文字以下の英数字が使用されている。

次のルールのすべてが満たされると、*String* は SAS 変数名とみなされます。

NLITERAL 関数は、*string* の内容に基づいて、*string* の値を一重または二重引用符で囲みます。

string の値	結果
アンパサンド(&)	一重引用符で囲む
パーセント記号(%)	一重引用符で囲む

string の値	結果
一重引用符より二重引用符が多い	一重引用符で囲む
前述のすべてに該当しない	二重引用符で囲む

結果のn-literal に十分なスペースがない場合、NLITERAL は空白の文字列を返し、エラーメッセージ
ERROR_を 1 に設定します。

サンプル

NLITERAL の複数の使用法を次の例に示します。

```
data test;
input string $32.;
length result $ 67;
result = nliteral(string);
datalines;
abc_123
This and That
cats & dogs
Company's profits (%)
"Double Quotes"
'Single Quotes'
;
proc print;
title 'Strings Converted to N-Literals or Returned Unchanged';
run;
```

画面 2.43 NLITERAL で文字列を名前リテラルに変換する

Strings Converted to N-Literals or Returned Unchanged

Obs	string	result
1	abc_123	abc_123
2	This and That	"This and That"N
3	cats & dogs	'cats & dogs'N
4	Company's profits (%)	'Company''s profits (%)'N
5	"Double Quotes"	'"Double Quotes"'N
6	'Single Quotes'	'''Single Quotes'''N

関連項目:

関数:

- “COMPARE 関数” (304 ページ)

- “DEQUOTE 関数” (361 ページ)
- “NVALID 関数” (692 ページ)

システムオプション:

- “VALIDVARNAME= System Option” in *SAS System Options: Reference*

その他のリファレンス:

- “SAS 言語におけるワード” (*SAS 言語リファレンス: 解説編* 3 章)

NMISS 関数

欠損数値の数を返します。

カテゴリ: 記述統計

構文

NMISS(*argument-1*<,...*argument-n*>)

必須引数

引数

数値の定数、変数または式を指定します。少なくとも 1 つの引数が必要です。引数リストには OF で始まる変数のリストを含められます。

詳細

NMISS 関数は欠損値の数を返すのに対し、N 関数は非欠損値の数を返します。NMISS には数値が必要なのに対し、CMISS は数値と文字値のどちらでも使用できます。NMISS には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。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x1=nmiss(1,0,.,2,5,.);	2
x2=nmiss(1,0);	0
x3=nmiss(of x1-x2);	0

NOMRATE 関数

名目年利を返します。

カテゴリ: 財務関数

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。

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。

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。

サンプル

- 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。

  ```
  effective_rate1 = NOMRATE('MONTH', 10);
  ```

- 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。

  ```
  effective_rate2 = NOMRATE('QUARTER', 10);
  ```

NORMAL 関数

正規(ガウス)分布からランダム変量を返します。

カテゴリ: 乱数

別名: RANNOR

参照項目: “RANNOR 関数” (798 ページ)

NOTALNUM 関数

文字列から英数字でない文字を検索し、最初に検索された文字の位置を返します。

カテゴリ: 文字関数

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

NOTALNUM(*string* <,*start*>)

必須引数

string
検索する文字の定数、変数または式を指定します。

オプション引数

start
検索の開始位置と検索方向を指定する整数値を使用する任意の数値定数、変数または式です。

詳細

NOTALNUM 関数の結果は、有効な変換テーブルに直接依存し(“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)、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。

サンプル

次の例では、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。

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

次の行が SAS ログに書き込まれます。

```
 j=5 c=
j=6 c==
j=7 c=
j=12 c=
j=13 c=+
j=14 c=
j=16 c=;
That's all
```

関連項目:

関数:

- “ANYALNUM 関数” (98 ページ)

NOTALPHA 関数

文字列から英字でない文字を検索し、最初に検索された文字の位置を返します。

カテゴリ: 文字関数

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

NOTALPHA(*string* <,*start*>)

必須引数

string
検索する文字の定数、変数または式です。

オプション引数

start
検索の開始位置と検索方向を指定する整数値を使用する任意の数値定数、変数または式です。

詳細

NOTALPHA 関数の結果は、有効な変換テーブルに直接依存し(“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)、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、文字列から英字を検索します。

サンプル

サンプル 1: 文字列からアルファベットでない文字を検索する

次の例では、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。

```
data _null_;
string='Next = _n_ + 12E3;';
j=0;
do until(j=0);
```

```
    j=notalpha(string,j+1);
    if j=0 then put +3 "That's all";
    else do;
    c=substr(string,j,1);
    put +3 j= c=;
    end;
    end;
    run;
```

次の行が SAS ログに書き込まれます。

```
 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=;
That's all
```

サンプル 2: NOTALPHA 関数を使用した制御文字を識別する

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。

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

関連項目:

関数:

- “ANYALPHA 関数” (100 ページ)

NOTCNTRL 関数

文字列から制御文字でない文字を検索し、最初に検索された文字の位置を返します。

カテゴリ: 文字関数

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

NOTCNTRL(*string* <,*start*>)

必須引数

string
検索する文字の定数、変数または式です。

オプション引数

start
検索の開始位置と検索方向を指定する整数値を使用する任意の数値定数、変数または式です。

詳細

NOTCNTRL 関数の結果は、有効な変換テーブルに直接依存し(“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)、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。

サンプル

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。

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

```
proc print data=test;
run;
```

関連項目:

関数:

- “ANYCNTRL 関数” (102 ページ)

NOTDIGIT 関数

文字列から数字でない文字を検索し、最初に検索された文字の位置を返します。

カテゴリ: 文字関数

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

NOTDIGIT(*string* <,*start*>)

必須引数

string
検索する文字の定数、変数または式です。

オプション引数

start
検索の開始位置と検索方向を指定する整数値を使用する任意の数値定数、変数または式です。

詳細

NOTDIGIT 関数の結果は、有効な変換テーブルに直接依存し(“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)、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。

サンプル

次の例では、NOTDIGIT 関数を使用して数字でない文字を検索します。

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

次の行が SAS ログに書き込まれます。

```
 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=;
That's all
```

関連項目:

関数:

- “ANYDIGIT 関数” (103 ページ)

NOTE 関数

SAS データセットの現在のオブザベーションのオブザベーション ID を返します。

カテゴリ: SAS ファイル I/O 関数

構文

NOTE(*data-set-id*)

必須引数

data-set-id
OPEN 関数が返すデータセット識別子を指定する数値変数です。

詳細

オブザベーション ID 値を使用して、POINT を使用して現在のオブザベーションに戻ることができます。オブザベーションは、NOTE を使用してマークを付けておき、後から POINT を使用して戻ることができます。各オブザベーション ID は重複しない数値です。

オブザベーション ID に関連付けられているメモリを解放するには、DROPNOTE を使用します。

サンプル

この例では、CUROBS を呼び出してオブザベーション番号を表示し、NOTE を呼び出してオブザベーションにマークを付け、POINT を呼び出して NOTEID に対応するオブザベーションに移動します。

```
%let dsid=%sysfunc(open(sasuser.fitness,i));
/* Go to observation 10 in data set */
%let rc=%sysfunc(fetchobs(&dsid,10));
%if %sysfunc(abs(&rc)) %then
%put FETCHOBS FAILED;
%else
%do;
/* Display observation number */
/* in the Log */
%let cur=%sysfunc(curobs(&dsid));
%put CUROBS=&cur;
/* Mark observation 10 using NOTE */
%let noteid=%sysfunc(note(&dsid));
/* Rewind pointer to beginning */
/* of data */
/* set using REWIND */
%let rc=%sysfunc(rewind(&dsid));
/* FETCH first observation into DDV */
%let rc=%sysfunc(fetch(&dsid));
/* Display first observation number */
%let cur=%sysfunc(curobs(&dsid));
%put CUROBS=&cur;
/* POINT to observation 10 marked */
/* earlier by NOTE */
%let rc=%sysfunc(point(&dsid,&noteid));
/* FETCH observation into DDV */
%let rc=%sysfunc(fetch(&dsid));
/* Display observation number 10 */
/* marked by NOTE */
%let cur=%sysfunc(curobs(&dsid));
```

```
%put CUROBS=&cur;
%end;
%if (&dsid > 0) %then
%let rc=%sysfunc(close(&dsid));
```

次の行が SAS ログに書き込まれます。

```
CUROBS=10
CUROBS=1
CUROBS=10
```

関連項目:

関数:

- “DROPNOTE 関数” (380 ページ)
- “OPEN 関数” (697 ページ)
- “POINT 関数” (725 ページ)
- “REWIND 関数” (807 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、最初に検索された文字の位置を返します。

カテゴリ: 文字関数

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

NOTFIRST(*string* <,*start*>)

必須引数

string
検索する文字の定数、変数または式です。

オプション引数

start
検索の開始位置と検索方向を指定する整数値を使用する任意の数値定数、変数または式です。

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
)、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

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。

サンプル

次の例では、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。

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

次の行が SAS ログに書き込まれます。

```
 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=;
That's all
```

関連項目:

関数:

- “ANYFIRST 関数” (105 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、最初に検索された文字の位置を返します。

カテゴリ: 文字関数

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

NOTGRAPH(*string* <,*start*>)

必須引数

string
検索する文字の定数、変数または式です。

オプション引数

start
検索の開始位置と検索方向を指定する整数値を使用する任意の数値定数、変数または式です。

詳細

NOTGRAPH 関数の結果は、有効な変換テーブルに直接依存し(“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)、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。

サンプル

サンプル 1: 文字列からの非グラフィック文字の検索

次の例では、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。

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

次の行が SAS ログに書き込まれます。

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

サンプル 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。

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

関連項目:

関数:

- “ANYGRAPH 関数” (106 ページ)

NOTLOWER 関数

文字列から小文字でない文字を検索し、最初に検索された文字の位置を返します。

カテゴリ: 文字関数

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

NOTLOWER(*string* <,*start*>)

必須引数

string
検索する文字の定数、変数または式です。

オプション引数

start
検索の開始位置と検索方向を指定する整数値を使用する任意の数値定数、変数または式です。

詳細

NOTLOWER 関数の結果は、有効な変換テーブルに直接依存し(“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)、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。

サンプル

次の例では、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。

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

次の行が SAS ログに書き込まれます。

```
 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=;
That's all
```

関連項目:

関数:

- “ANYLOWER 関数” (109 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、最初に検索された文字の位置を返します。

カテゴリ: 文字関数

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

NOTNAME*string* <,*start*>)

必須引数

string
検索する文字の定数、変数または式です。

オプション引数

start
検索の開始位置と検索方向を指定する整数値を使用する任意の数値定数、変数または式です。

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
)、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

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。

サンプル

次の例では、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。

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

```
    end;
    run;
```

次の行が SAS ログに書き込まれます。

```
 j=5 c=
j=6 c==
j=7 c=
j=11 c=
j=12 c=+
j=13 c=
j=18 c=;
That's all
```

関連項目:

関数:

- “ANYNAME 関数” (110 ページ)

NOTPRINT 関数

文字列から印刷不可文字を検索し、最初に検索された文字の位置を返します。

カテゴリ: 文字関数

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

NOTPRINT(*string* <,*start*>)

必須引数

string
検索する文字の定数、変数または式です。

オプション引数

start
検索の開始位置と検索方向を指定する整数値を使用する任意の数値定数、変数または式です。

詳細

NOTPRINT 関数の結果は、有効な変換テーブルに直接依存し(“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)、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、文字列から印刷不可能な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、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、文字列から印刷可能文字でない文字を検索します。
ANYPRINT 関数は、文字列から印刷可能文字を検索します。

サンプル

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。

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

関連項目:

関数:

- “ANYPRINT 関数” (112 ページ)

NOTPUNCT 関数

文字列から句読文字でない文字を検索し、最初に検索された文字の位置を返します。

カテゴリ: 文字関数

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

NOTPUNCT(*string* <,*start*>)

必須引数

string
検索する文字の定数、変数または式です。

オプション引数

start
検索の開始位置と検索方向を指定する整数値を使用する任意の数値定数、変数または式です。

詳細

NOTPUNCT 関数の結果は、有効な変換テーブルに直接依存し(“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)、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。

サンプル

サンプル 1: 文字列からの句読文字でない文字の検索

次の例では、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。

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

```
    end;
    end;
    run;
```

次の行が SAS ログに書き込まれます。

```
 j=1 c=N
j=2 c=e
j=3 c=x
j=4 c=t
j=5 c=
j=7 c=
j=9 c=n
j=11 c=
j=13 c=
j=14 c=1
j=15 c=2
j=16 c=E
j=17 c=3
That's all
```

サンプル 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。

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

関連項目:

関数:

- “ANYPUNCT 関数” (114 ページ)

NOTSPACE 関数

文字列から空白文字(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)でない文字を検索し、最初に検索された文字の位置を返します。

カテゴリ: 文字関数

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

NOTSPACE(*string* <,*start*>)

必須引数

string
検索する文字の定数、変数または式です。

オプション引数

start
検索の開始位置と検索方向を指定する整数値を使用する任意の数値定数、変数または式です。

詳細

NOTSPACE 関数の結果は、有効な変換テーブルに直接依存し(“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)、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。

サンプル

サンプル 1: 文字列からの空白文字でない文字の検索

次の例では、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。

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

```
    else do;
    c=substr(string,j,1);
    put +3 j= c=;
    end;
    end;
    run;
```

次の行が SAS ログに書き込まれます。

```
 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=;
That's all
```

サンプル 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。

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

関連項目:

関数:

- “ANYSPACE 関数” (116 ページ)

NOTUPPER 関数

文字列から大文字でない文字を検索し、最初に検索された文字の位置を返します。

カテゴリ: 文字関数

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

NOTUPPER(*string* <,*start*>)

必須引数

string
検索する文字の定数、変数または式です。

オプション引数

start
検索の開始位置と検索方向を指定する整数値を使用する任意の数値定数、変数または式です。

詳細

NOTUPPER 関数の結果は、有効な変換テーブルに直接依存し(“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)、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。

サンプル

次の例では、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。

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

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

次の行が SAS ログに書き込まれます。

```
 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=;
That's all
```

関連項目:

関数:

- “ANYUPPER 関数” (118 ページ)

NOTXDIGIT 関数

文字列から 16 進数字でない文字を検索し、最初に検索された文字の位置を返します。

カテゴリ: 文字関数

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

NOTXDIGIT(*string* <,*start*>)

必須引数

string
検索する文字の定数、変数または式です。

オプション引数

start
検索の開始位置と検索方向を指定する整数値を使用する任意の数値定数、変数または式です。

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。

サンプル

次の例では、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。

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

次の行が SAS ログに書き込まれます。

```
 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=;
That's all
```

関連項目:

関数:

- “ANYXDIGIT 関数” (119 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: 財務関数

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。

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

注: ケース *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。

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。

NVALID 関数

SAS 変数名として使用する文字列の有効性を確認します。

カテゴリ: 文字関数

制限事項: I18N レベル 0 の関数は 1 バイト文字セット(SBCS)でのみ使用するように設計されています。

構文

NVALID(*string* <,*validvarname*>)

必須引数

string
SAS 変数名として使用できる値かどうかを判断するために確認する文字定数、変数または式を指定します。

注: 末尾の空白は無視されます。

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

validvarname
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。

V7
次の 3 つをすべて満たす場合に *string* が有効な SAS 変数名であると判断します。

- *string* が英字またはアンダースコアで始まる
- 後続のすべての文字が英字、アンダースコアまたは数字である
- 長さが 32 文字以下の英数字である

ANY
あらゆる種類(空白を含む)の 32 文字以下の文字が含まれている場合に *string* が有効な SAS 変数名であると判断します。

NLITERAL
SAS 名リテラル('name'N)の形式である場合、または VALIDVARNAME=V7 の有効な SAS 変数名である場合に *string* が有効な SAS 変数名であると判断します。

参照項目: このリストの前述の V7。

デフォルト: 値を指定しないと、NVALID 関数は SAS システムオプション VALIDVARNAME=の値に基づいて *string* が有効な SAS 変数名であると判断します。

詳細

NVALID 関数は、*string* の値をチェックして、SAS 変数名として使用できるかどうかを判断します。

NVALID 関数は、値 1 または値 0 を返します。

条件	戻り値
string が SAS 変数名として使用できる	1
string が SAS 変数名として使用できない	0

サンプル

この例では、指定した文字列が SAS 変数名として有効かどうかを判断します。NVALID 関数の戻り値は、validvarname 引数によって異なります。指定した validvarname 引数のルールに基づいて文字列が有効な SAS 変数名であると判断された場合、値 1 が返されます。それ以外の場合、値 0 が返されます。

```
options validvarname=v7 ls=64;
data string;
input string $char40.;
v7=nvalid(string,'v7');
any=nvalid(string,'any');
nliteral=nvalid(string,'nliteral');
default=nvalid(string);
datalines;
Toooooooooooooooooooooooooo Long
OK
Very_Long_But_Still_OK_for_V7
1st_char_is_a_digit
Embedded blank
!@#$%^&*
"Very Loooong N-Literal with """N
'No closing quotation mark
;

proc print noobs;
title1 'NLITERAL and Validvarname Arguments Determine';
title2 'Invalid (0) and Valid (1) SAS Variable Names';
run;
```

画面 2.44 *NLITERAL を使用した SAS 変数名が有効であるかどうかの判断*

NLITERAL and Validvarname Arguments Determine
Invalid (0) and Valid (1) SAS Variable Names

string	v7	any	nliteral	default
Toooooooooooooooooooooooooo Long	0	0	0	0
OK	1	1	1	1
Very_Long_But_Still_OK_for_V7	1	1	1	1
1st_char_is_a_digit	0	1	1	0
Embedded blank	0	1	1	0
!@#$%^&*	0	1	1	0
"Very Loooong N-Literal with """N	0	0	1	0
'No closing quotation mark	0	1	0	0

関連項目:

関数:

- “COMPARE 関数” (304 ページ)
- “NLITERAL 関数” (664 ページ)

システムオプション:

- “VALIDVARNAME= System Option” in *SAS System Options: Reference*

その他のリファレンス:

- “SAS 言語のワードと命名規則について” (*SAS 言語リファレンス: 解説編* 3 章)

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

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: 1-255

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: 1-255

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。

月
年の月に対応する数値を指定します。

範囲: 1-255

年
4 桁のカレンダ年を指定します。

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、**UnivGrad = nwkdom(1, 7, 6, *year*);というステートメントを使用してその日付を計算できます。**

サンプル

サンプル 1: 日付値を返す

次の例では、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。

```
data _null_;
/* Return the date of the third Monday in May 2000. */
a=nwkdom(3, 2, 5, 2000);
/* Return the date of the fourth Wednesday in November 2007. */
b=nwkdom(4, 4, 11, 2007);
/* Return the date of the fourth Saturday in November 2007. */
c=nwkdom(4, 7, 11, 2007);
/* Return the date of the first Sunday in January 2007. */
d=nwkdom(1, 1, 1, 2007);
/* Return the date of the second Tuesday in September 2007. */
e=nwkdom(2, 3, 9, 2007);
/* Return the date of the fifth Thursday in December 2007. */
f=nwkdom(5, 5, 12, 2007);
put a= weekdatx.;
put b= weekdatx.;
put c= weekdatx.;
put d= weekdatx.;
put e= weekdatx.;
put f= weekdatx.;
run;
```

ログ 2.15 *日付値が返されたときの出力*

```
a=Monday, 15 May 2000
b=Wednesday, 28 November 2007
c=Saturday, 24 November 2007
d=Sunday, 7 January 2007
e=Tuesday, 11 September 2007
f=Thursday, 27 December 2007
```

サンプル 2: 5 月の最終月曜日の日付を返す

次の例では、2007 年 5 月の最終月曜日に対応する日付を返します。

```
data _null_;
/* The last Monday in May. */
x=nwkdom(5,2,5,2007);
put x date9.;
run;
```

次の出力が SAS ログに書き込まれます。

ログ 2.16 5 月の最終月曜日の日付を計算したときの出力

```
28MAY2007
```

関連項目:

関数:

- “HOLIDAY 関数” (519 ページ)
- “INTNX 関数” (567 ページ)
- “MONTH 関数” (652 ページ)
- “WEEKDAY 関数” (952 ページ)

OPEN 関数

SAS データセットを開きます。

カテゴリ: SAS ファイル I/O 関数

構文

OPEN(<*data-set-name*<,*mode*<,*generation-number*<,*type*>>>>)

オプション引数

data-set-name
開く SAS データセットまたは SAS SQL ビューの名前を指定する文字定数、変数または式です。この文字列の値は次の形式になります。

<*libref*.> *member-name*<(*data-set-options*)>

デフォルト: *data-set-name* のデフォルト値は _LAST_ です。

制限事項: FIRSTOBS=および OBS=データセットオプションを指定しても無視されます。その他のデータセットオプションはすべて有効です。

mode
データセットへのアクセスの種類を指定する文字定数、変数または式です。

I INPUT モードでデータセットを開きます(デフォルト)。値を読み込むことはできますが、変更することはできません。'I' では、エンジンで最も強力なアクセスモードが使用されます。つまり、エンジンでランダムアクセスがサポートされている場合、OPEN ではデフォルトでランダムアクセスが使用されます。それ以外の場合、ファイルは自動的に'IN'モードで開かれます。ファイルはシーケンシャルアクセスで開かれ、システムレベルの警告が設定されます。

IN INPUT モードでデータセットを開きます。オブザベーションを順次読み込みます(オブザベーションの再アクセスができます)。

IS INPUT モードでデータセットを開きます。オブザベーションを順次読み込みます(オブザベーションの再アクセスはできません)。

デフォルト: I

generation-number
生成グループの履歴バージョンの 1 つを識別するための恒常的に増加する番号を指定します。

ヒント: *type* = F の場合、*generation-number* 引数は無視されます。

type
次のいずれかの文字定数値になります。

D
第 1 引数 *data-set-name* が 1 レベルまたは 2 レベルのデータセット名であることを示します。次の例では、D *type* 値をどのように使用できるのかを示します。

```
rc = open('lib.mydata', , , 'D');
```

ヒント: 第 4 引数がない場合、D がデフォルトになります。

F
第 1 引数 *data-set-name* がファイル名(ファイルの物理パス)であることを示します。次の例では、F *type* 値をどのように使用できるのかを示します。

```
rc = open('c:¥data¥mydata.sas7bdat', , , 'F'); rc = open('c:¥data¥mydata', , , 'F');
```

ヒント: F 値を使用すると、第 3 引数 *generation-number* は無視されます。

注:
引数が無効な場合、OPEN は0 を返します。SYSMSG 関数から対応するエラーメッセージの ERROR_自動変数も設定されません。

詳細

OPEN 関数は、SAS データセット、DATA ステップまたは SAS SQL ビューを開き、一意の数値データセット識別子を返します。ほとんどの場合、この識別子は他のデータセットアクセス関数で使用されます。データセットを開けなかった場合、OPEN は 0 を返します。

OPEN 関数をマクロから呼び出す場合、マクロの関数に渡された呼び出しの結果のみが有効になります。OPEN 関数を DATA ステップから呼び出す場合、同じ DATA ステップの関数に渡された結果のみが有効になります。

デフォルトでは、SAS データセットは RECORD 制御レベルで開きます。詳細については、“CNTLLEV= Data Set Option” in *SAS Data Set Options: Reference* を参照してください。開いている SAS データセットが不要になったら閉じる必要があります。DATA ステップ内で開いたデータセットは、DATA ステップの終了時に自動的に閉じます。

OPEN はデフォルトでエンジンで最も強力なアクセスモードになります。つまり、エンジンでランダムアクセスがサポートされている場合、OPEN ではデフォルトでランダムアクセスが使用されます。それ以外の場合、データセットはシーケンシャルアクセスで開かれ、システムレベルの警告が設定されます。

サンプル

- この例では、INPUT モードを使用してライブラリ MASTER のデータセット PRICES を開きます。マクロステートメントでは、文字列を引用符で囲みません。

```
%let dsid=%sysfunc(open(master.prices,i));
%if (&dsid = 0) %then
```

```
%put %sysfunc(sysmsg());
%else
%put PRICES data set has been opened;
```

- この例では、データセットオプションで使用するマクロ変数または DATA ステップ変数の値を渡します。データセット SASUSER.HOUSES を開き、WHERE=データセットオプションを使用して永続的な WHERE 句を適用します。マクロステートメントでは、文字列を引用符で囲みません。

```
%let choice = style="RANCH";
%let dsid=%sysfunc(open(sasuser.houses
(where=(&choice)),i));
```

- この例では、エラーの戻り値の確認方法や SYSMSG 関数がエラーメッセージを書き込む方法を示します。

```
data _null_;
d=open('bad','?');
if not d then do;
m=sysmsg();
put m;
abort;
end;
... more SAS statements ...;
run;
```

関連項目:

関数:

- “CLOSE 関数” (297 ページ)
- “SYSMSG 関数” (880 ページ)

ORDINAL 関数

k 番目に小さい欠損値および非欠損値を返します。

カテゴリ: 記述統計

構文

ORDINAL(*k*,*argument-1*,*argument-2* <,…*argument-n*>)

必須引数

k
引数リストの後続の要素数以下の整数値となる数値定数、変数または式です。

引数
数値の定数、変数または式を指定します。少なくとも 2 つの引数が必要です。OF が前に付く変数リストで引数が構成される場合もあります。

詳細

ORDINAL 関数は、第 2 引数 ~ 最後の引数の中で k 番目に小さい欠損値または非欠損値を返します。

比較

ORDINAL 関数は欠損値と非欠損値の両方を数えますが、SMALLEST 関数は非欠損値のみを数えます。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x1=ordinal(4,1,2,3,-4,5,6,7);	3

PATHNAME 関数

外部ファイルや SAS ライブラリの物理名または空白を返します。

カテゴリ: SAS ファイル I/O 関数
外部ファイル

参照項目: “PATHNAME Function: UNIX” in *SAS Companion for UNIX Environments*
“PATHNAME Function: z/OS” in *SAS Companion for z/OS*

構文

PATHNAME((*fileref* | *libref*) <,*search-ref*>)

必須引数

fileref
外部ファイルに割り当てられたファイル参照名を指定する文字定数、変数または式です。

ライブラリ参照名
SAS ライブラリに割り当てるライブラリ参照名を指定する文字定数、変数または式です。

オプション引数

search-ref
ファイル参照名とライブラリ参照名のどちらを検索するのかを指定する文字定数、変数または式です。

F ファイル参照名の検索を示します。

L ライブラリ参照名の検索を示します。

詳細

PATHNAME は外部ファイルまたは SAS ライブラリの物理名を返します。*fileref* または *libref* が無効な場合は空白を返します。

ファイル参照名の名前とライブラリ参照名の名前が同じ場合、*search-ref* 引数を使用して、どちらの参照を検索するのかを選択できます。F 値を指定すると、SAS はファイル参照名を検索します。L 値を指定すると、SAS はライブラリ参照名を検索します。

search-ref 引数を指定しない場合、ファイル参照名の名前とライブラリ参照名の名前が同じだと、PATHNAME は最初にライブラリ参照名を検索します。ライブラリ参照名が存在しない場合、PATHNAME はファイル参照名を検索します。

DATA ステップの対象変数のデフォルトの長さは 200 文字です。

FILENAME ステートメントまたは FILENAME 関数を使用して、ファイル参照名を外部ファイルに割り当てることができます。

LIBNAME ステートメントまたは LIBNAME 関数を使用して、ライブラリ参照名を SAS ライブラリに割り当てることができます。動作環境によっては、システムコマンドを使用してライブラリ参照名を割り当てることもできます。

Windows 固有
: 一部の動作環境では、システムコマンドを使用してファイル参照名を割り当てることもできます。詳細については、動作環境に関する SAS のドキュメントを参照してください。

サンプル

この例では、FILEREF 関数を使用してファイル参照名 MYFILE を外部ファイルに割り当てます。次に、PATHNAME を使用して外部ファイルの実際の名前を取得します。

```
data _null_;
length fname $ 100;
rc=fileref('myfile');
if (rc=0) then
do;
fname=pathname('myfile');
put fname=;
end;
run;
```

関連項目:

関数:

- “FEXIST 関数” (400 ページ)
- “FILEEXIST 関数” (403 ページ)
- “FILENAME 関数” (404 ページ)
- “FILEREF 関数” (407 ページ)

ステートメント:

- “LIBNAME ステートメント” (*SAS ステートメント: リファレンス*)
- “FILENAME ステートメント” (*SAS ステートメント: リファレンス*)

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

構文

PCTL<*n*> (*percentage*, *value1* <,*value2*,…>)

必須引数

percentage
計算するパーセント点を指定する数値定数、変数または式です。
要件 0 ≤ パーセント ≤ 100 の数値。

value
数値変数、定数または式です。

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 ~ 5 の数字です。
デフォルト: 定義 5

詳細

PCTL 関数は、パーセントに対応する非欠損値のパーセント点を返します。*percentage* が欠損値、ゼロ未満または 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。

注: PCTL 関数で使用される式は、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” in Chapter 1 of *Base SAS Procedures Guide* を参照してください。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
lower_quartile=PCTL(25,2,4,1,3); put lower_quartile;	1.5
percentile_def2=PCTL2(25,2,4,1,3); put percentile_def2;	1
lower_tertile=PCTL(100/3,2,4,1,3); put lower_tertile;	2
percentile_def3=PCTL3(100/3,2,4,1,3); put percentile_def3;	2

SAS ステートメント	結果
median=PCTL(50,2,4,1,3); put median;	2.5
upper_tertile=PCTL(200/3,2,4,1,3); put upper_tertile;	3
upper_quartile=PCTL(75,2,4,1,3); put 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

構文

PDF (*dist*,*quantile*<,*parm-1*,...,*parm-k*>)

必須引数

dist

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
ベルヌーイ	**ベルヌーイ**
ベータ	**BETA**
二項	**BINOMIAL**
コーシー	**CAUCHY**
χ2 乗	**CHISQUARE**
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
ポアソン	POISSON
T	T
Tweedie	TWEEDIE
一様	UNIFORM
逆ガウス(Wald)	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。

等量分類
ランダム変数の値を指定する数値定数、変数または式です。

オプション引数

parm-1,...,parm-k
特定の分布に適した*形状*、*位置*または*尺度*パラメータ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: でこれらのパラメータの詳細については、“詳細” (704 ページ) を参照してください。

詳細

Bernoulli 分布

PDF('BERNOULLI',*x*,*p*)

引数

x
数値のランダム変数です。

p
数値の成功確率です。
範囲: $0 \le p \le 1$

詳細

Bernoulli 分布の PDF 関数は、Bernoulli 分布(成功確率は p)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。式は次のとおりです。

$$PDF('BERN', x, p) = \begin{cases} 0 & x < 0 \\ 1 - p & x = 0 \\ 0 & 0 < x < 1 \\ p & x = 1 \\ 0 & x > 1 \end{cases}$$

注: この分布には、*位置*または*尺度*パラメータはありません。

ベータ分布

PDF('BETA',*x*,*a*,*b*<,*l*,*r*>)

引数

x
数値のランダム変数です。

a
数値の形状パラメータです。
範囲: $a > 0$

b
数値の形状パラメータです。
範囲: $b > 0$

l
数値の左位置パラメータです。
デフォルト: 0

r
右位置パラメータです。
デフォルト: 1
範囲: $r > l$

詳細

ベータ分布の PDF 関数は、ベータ分布(形状パラメータは a および b)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。式は次のとおりです。

$$PDF('BETA', x, a, b, l, r) = \begin{cases} 0 & x < l \\ \frac{1}{\beta(a, b)} \frac{(x - l)^{a-1}(r - x)^{b-1}}{(r - l)^{a+b-1}} & l \le x \le r \\ 0 & x > r \end{cases}$$

注: 数量 $\frac{x - l}{r - l}$ は、強制的に $\varepsilon \le \frac{x - l}{r - l} \le 1 - 2\varepsilon$ になります。

二項分布

PDF('BINOMIAL',*m*,*p*,*n*)

引数

m
成功数を数える整数のランダム変数です。
範囲: $m = 0, 1, ...$

p
数値の成功確率です。
範囲: $0 \le p \le 1$

n
独立ベルヌーイ試行数を数える整数のパラメータです。
範囲: $n = 0, 1, ...$

詳細

二項分布の PDF 関数は、値 *m* で評価される二項分布(パラメータは *p* および *n*)の確率密度関数を返します。式は次のとおりです。

$$PDF('BINOM', m, p, n) = \begin{cases} 0 & m < 0 \\ \binom{n}{m} p^m (1-p)^{n-m} & 0 \le m \le n \\ 0 & m > n \end{cases}$$

注: 二項分布には、*位置*または*尺度*パラメータはありません。

Cauchy 分布

PDF('CAUCHY',*x*<,*θ*,*λ*>)

引数

x
数値のランダム変数です。

θ
数値の位置パラメータです。
デフォルト: 0

λ
数値の尺度パラメータです。
デフォルト: 1
範囲: $\lambda > 0$

詳細

Cauchy 分布の PDF 関数は、Cauchy 分布(位置パラメータは θ、尺度パラメータは 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。式は次のとおりです。

$$PDF('CAUCHY', x, \theta, \lambda) = \frac{1}{\pi}\left(\frac{\lambda}{\lambda^2 + (x-\theta)^2}\right)$$

カイ 2 乗分布

PDF('CHISQUARE',*x*,*df* <,*nc*>)

引数

x
数値のランダム変数です。

df

数値の自由度パラメータです。

範囲: $df > 0$

nc

任意の数値の非心度パラメータです。

範囲: $nc \geq 0$

詳細

χ2 乗分布の PDF 関数は、χ2 乗分布(自由度は *df*、非心度パラメータは *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* が除外されているかゼロの場合、心度 χ2 乗分布から値が返されます。次の式は、χ2 乗分布の PDF 関数を表しています。

$$PDF('CHISQ', x, v, \lambda) = \begin{cases} 0 & x < 0 \\ \sum_{j=0}^{\infty} e^{-\frac{\lambda}{2}} \frac{\left(\frac{\lambda}{2}\right)^j}{j!} p_c(x, v+2j) & x \geq 0 \end{cases}$$

この式では、$p_c(.,.)$は、心度 χ2 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、次の式によって得られるガンマ分布の密度です。

$$p_g(y, b) = \frac{1}{\Gamma(b)} e^{-y} y^{b-1}$$

指数分布

PDF('EXPONENTIAL',*x* <,*λ*>)

引数

x

数値のランダム変数です。

λ

尺度パラメータです。

デフォルト: 1

範囲: λ > 0

詳細

指数分布の PDF 関数は、指数分布(尺度パラメータは 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。式は次のとおりです。

$$PDF('EXPO', x, \lambda) = \begin{cases} 0 & x < 0 \\ \frac{1}{\lambda} exp\left(-\frac{x}{\lambda}\right) & x \geq 0 \end{cases}$$

F 分布

PDF('F',*x*,*ndf*,*ddf* <,*nc*>)

引数

x

数値のランダム変数です。

ndf

数値の分子の自由度パラメータです。

範囲: $ndf > 0$

ddf
数値の分母の自由度パラメータです。

範囲: $ddf > 0$

nc
数値の非心度パラメータです。

範囲: $nc \geq 0$

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。次の式では、$v_1 = ndf$、$v_2 = ddf$および $\lambda = nc$となります。次の式は、*F* 分布の PDF 関数を表しています。

$$PDF('F', x, v_1, v_2, \lambda) = \begin{cases} 0 & x < 0 \\ \sum_{j=0}^{\infty} e^{-\frac{\lambda}{2}} \frac{\left(\frac{\lambda}{2}\right)^j}{j!} p_f(f, v_1 + 2j, v_2) & x \geq 0 \end{cases}$$

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*パラメータはありません。

ガンマ分布

PDF('GAMMA',*x*,*a*<,*λ*>)

引数

x
数値のランダム変数です。

a
数値の形状パラメータです。

範囲: $a > 0$

λ
数値の尺度パラメータです。

デフォルト: 1

範囲: $\lambda > 0$

詳細

ガンマ分布の PDF 関数は、ガンマ分布(形状パラメータは *a*、尺度パラメータは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。式は次のとおりです。

$$PDF('GAMMA', x, a, \lambda) = \begin{cases} 0 & x < 0 \\ \frac{1}{\lambda^a \Gamma(a)} x^{a-1} exp\left(-\frac{x}{\lambda}\right) & x \geq 0 \end{cases}$$

一般化 Poisson 分布

PDF('GENPOISSON',*x*, *θ*, *η*)

引数

x
整数のランダム変数です。

θ
形状パラメータを指定します。
範囲: $<10^5$ かつ>0

η
形状パラメータを指定します。
範囲: ≥ 0 かつ<0.95
ヒント: $\eta = 0$ の場合、平均と分散が θ のポアソン分布になります。$\eta > 0$ の場合、平均は $\theta \div (1 - \eta)$ で、分散は $\theta \div (1 - \eta)^3$です。

詳細

一般化 Poisson 分布の確率質量関数の次のとおりです。

$$f(x; \theta, \eta) = \theta(\theta + \eta x)^{x-1} e^{-\theta - \eta x} / x!, \quad x = 0, 1, 2, \ldots, \qquad \theta > 0, \ 0 \leq \eta < 1$$

幾何分布

PDF('GEOMETRIC',*m*,*p*)

引数

m
初めて成功する前の失敗数を示す数値のランダム変数です。
範囲: $m \geq 0$

p
数値の成功確率です。
範囲: $0 \leq p \leq 1$

詳細

幾何分布の PDF 関数は、幾何分布(パラメータは p)の確率密度関数を返します。PDF 関数は、値 m で評価されます。式は次のとおりです。

$$PDF('GEOM', m, p) = \begin{cases} 0 & m < 0 \\ p(1 - p)^m & m \geq 0 \end{cases}$$

注: この分布には、*位置*または*尺度*パラメータはありません。

超幾何分布

PDF('HYPER',*x*,*N*,*R*,*n* <,*o*>)

引数

x
整数のランダム変数です。

N
整数の母集団サイズパラメータです。
範囲: $N = 1, 2, \ldots$

R
対象カテゴリの整数の項目数です。

範囲: $R = 0, 1, ..., N$

n

整数のサンプルサイズパラメータです。

範囲: $n = 1, 2, ..., N$

o

任意の数値のオッズ比パラメータです。

範囲: $o > 0$

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。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。式は次のとおりです。

$$PDF('HYPER', x, N, R, n, o) = \begin{cases} 0 & x < max(0, R + n - N) \\ \dfrac{\binom{R}{x}\binom{N-R}{n-x}o^x}{\sum_{j=max(0,R+n-N)}^{min(R,n)} \binom{R}{j}\binom{N-R}{n-j}o^j} & max(0, R + n - N) \le x \le min(R, n) \\ 0 & x > min(R, n) \end{cases}$$

Laplace 分布

PDF('LAPLACE',*x*<,*θ*,*λ*>)

引数

x

数値のランダム変数です。

θ

数値の位置パラメータです。

デフォルト: 0

λ

数値の尺度パラメータです。

デフォルト: 1

範囲: $\lambda > 0$

詳細

Laplace 分布の PDF 関数は、Laplace 分布(位置パラメータは θ、尺度パラメータは 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。式は次のとおりです。

$$PDF('LAPLACE', x, \theta, \lambda) = \frac{1}{2\lambda} exp\left(-\frac{|x - \theta|}{\lambda}\right)$$

ロジスティック分布

PDF('LOGISTIC',*x*<,*θ*,*λ*>)

引数

x

数値のランダム変数です。

θ

数値の位置パラメータです。

デフォルト: 0

λ

数値の尺度パラメータです。

デフォルト: 1

範囲: $\lambda > 0$

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(位置パラメータは θ、尺度パラメータは 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。式は次のとおりです。

$$PDF('LOGISTIC', x, \theta, \lambda) = \frac{exp\left(-\frac{x-\theta}{\lambda}\right)}{\lambda\left(1+exp\left(-\frac{x-\theta}{\lambda}\right)\right)^2}$$

対数正規分布

PDF('LOGNORMAL',*x*<,*θ*,*λ*>)

引数

x

数値のランダム変数です。

θ

数値の対数尺度パラメータを指定します (exp(θ)は尺度パラメータ)。

デフォルト: 0

λ

数値の形状パラメータを指定します。

デフォルト: 1

範囲: $\lambda > 0$

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は θ、形状パラメータは 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。式は次のとおりです。

$$PDF('LOGN', x, \theta, \lambda) = \begin{cases} 0 & x \le 0 \\ \frac{1}{x\lambda\sqrt{2\pi}}\ exp\left(-\frac{(log(x)-\theta)^2}{2\lambda^2}\right) & x > 0 \end{cases}$$

負数二項分布

PDF('NEGBINOMIAL',*m*,*p*,*n*)

引数

m

失敗数を数える正の整数のランダム変数です。

範囲: $m = 0, 1, \ldots$

p

数値の成功確率です。

範囲: $0 \le p \le 1$

n
成功数を数える数値です。
範囲: $n > 0$

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。PDF 関数は、値 m で評価されます。式は次のとおりです。

$$PDF('NEGB', m, p, n) = \begin{cases} 0 & m < 0 \\ \binom{n+m-1}{n-1} p^n (1-p)^m & m \geq 0 \end{cases}$$

注: 負の二項分布には、*位置*または*尺度*パラメータはありません。

正規分布

PDF('NORMAL',*x*<,*θ*,*λ*>)

引数

x
数値のランダム変数です。

θ
数値の位置パラメータです。
デフォルト: 0

λ
数値の尺度パラメータです。
デフォルト: 1
範囲: $\lambda > 0$

詳細

正規分布の PDF 関数は、正規分布(位置パラメータは θ、尺度パラメータは 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。式は次のとおりです。

$$PDF('NORMAL', x, \theta, \lambda) = \frac{1}{\lambda\sqrt{2\pi}} \exp\left(-\frac{(x-\theta)^2}{2\lambda^2}\right)$$

正規混合分布

PDF('NORMALMIX',*x*,*n*,*p*,*m*,*s*)

引数

x
数値のランダム変数です。

n
混合の整数です。
範囲: $n = 1, 2, \ldots$

p
n 個の母集団 $p_1, p_2, \ldots, p_n$ です。ここで、$\sum_{i=1}^{i=n} p_i = 1$ となります。
範囲: $p = 0, 1, \ldots$

m
は n 個の平均 $m_1, m_2, \ldots, m_n$ です。

s
は n 個の標準偏差 $s_1, s_2, \ldots, s_n$ です。

範囲: $s > 0$

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。式は次のとおりです。

$$PDF('NORMALMIX', x, n, p, m, s) = \sum_{i=1}^{i=n} p_i \, PDF('NORMAL', x, m_i, s_i)$$

注: 正規混合分布には、*位置*または*尺度*パラメータはありません。

パレート分布

PDF('PARETO',*x*,*a*<,*k*>)

引数

x
数値のランダム変数です。

a
数値の形状パラメータです。

範囲: $a > 0$

k
数値の尺度パラメータです。

デフォルト: 1

範囲: $k > 0$

詳細

パレート分布の PDF 関数は、パレート分布(形状パラメータは a、尺度パラメータは 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。式は次のとおりです。

$$PDF('PARETO', x, a, k) = \begin{cases} 0 & x < k \\ \frac{a}{k}\left(\frac{k}{x}\right)^{a+1} & x \geq k \end{cases}$$

Poisson 分布

PDF('POISSON',*n*,*m*)

引数

n
整数のランダム変数です。

範囲: $n = 0, 1, \ldots$

m
数値の平均パラメータです。

範囲: $m > 0$

詳細

Poisson 分布の PDF 関数は、Poisson 分布(平均は *m*)の確率密度関数を返します。PDF 関数は、値 *n* で評価されます。式は次のとおりです。

$$PDF('POISSON', n, m) = \begin{cases} 0 & n < 0 \\ e^{-m}\frac{m^n}{n!} & n \geq 0 \end{cases}$$

注: ポアソン分布には、*位置*または*尺度*パラメータはありません。

T 分布

PDF('T',*t*,*df* <,*nc*>)

引数

t

数値のランダム変数です。

df

数値の自由度パラメータです。

範囲: $df > 0$

nc

任意の数値の非心度パラメータです。

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。次の式では、$\nu = df$ および $\delta = nc$となります。

$$PDF('T', t, \nu, \delta) = \frac{1}{2^{\frac{\nu}{2}-1}\Gamma\left(\frac{\nu}{2}\right)}\int_0^{\infty} x^{\nu-1} e^{-\frac{1}{2}x^2} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{tx}{\sqrt{\nu}}-\delta\right)^2} \frac{x}{\sqrt{\nu}}\, dx$$

注: *T* 分布には、*位置*または*尺度*パラメータはありません。

Tweedie 分布

PDF('TWEEDIE', *y*, *p*<,*μ*, *φ*>)

引数

y

ランダム変数です。

範囲: $y \geq 0$

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

累乗パラメータです。

範囲: $p \geq 1$

注: この引数は必須です。

μ

平均です。

デフォルト: 1

範囲: $\mu > 0$

φ

分散パラメータです。

デフォルト: 1

範囲: $\varphi>0$

詳細

ツウィーディ分布の PDF 関数は、式 $variance = \phi * \mu^{p}$に関する分散と平均を使用する拡張分散モデルを返します。

式は次のとおりです。

$$\frac{1}{y}\sum_{j=1}^{\infty}\left(\frac{y^{-j\alpha}(p-1)^{j\alpha}}{\phi^{j(1-\alpha)}(2-p)^{j}\,j!\,\Gamma(-j\alpha)}\right)\exp\left(\frac{1}{\phi}\left(y\frac{\mu^{1-p}-1}{1-p}-\frac{\mu^{2-p}-1}{2-p}\right)\right)$$

前述の式には次の関係が適用されます。

$$\alpha = \frac{2-p}{1-p}$$

注: 計算されたツウィーディの確率の精度は、パラメータ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 ϕ 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。

一様分布

PDF('UNIFORM',*x*<,*l*,*r*>)

引数

x

数値のランダム変数です。

l

数値の左位置パラメータです。

デフォルト: 0

r

数値の右位置パラメータです。

デフォルト: 1

範囲: $r > l$

詳細

一様分布の PDF 関数は、一様分布(左位置パラメータは l、右位置パラメータは 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。式は次のとおりです。

$$PDF('UNIFORM', x, l, r) = \begin{cases} 0 & x < l \\ \frac{1}{r-l} & l \le x \le r \\ 0 & x > r \end{cases}$$

逆ガウス(Wald)分布

PDF('WALD',*x*,*λ*<,*μ*>)

PDF('IGAUSS',*x*,*λ*<,*μ*>)

引数

x

数値のランダム変数です。

λ
数値の形状パラメータです。
範囲: $\lambda > 0$

μ
平均です。
デフォルト: 1
範囲: $\mu > 0$

詳細

Wald 分布の PDF 関数は、値 *x* で評価される Wald 分布(形状パラメータは λ)の確率密度関数を返します。式は次のとおりです。

$$fx(x) = \left[\frac{\lambda}{2\pi x^3}\right]^{1/2} \exp\left\{-\frac{\lambda}{2\mu^2 x}(x-\mu)^2\right\}, \qquad x > 0$$

Weibull 分布

PDF('WEIBULL',*x*,*a*<,*λ*>)

引数

x
数値のランダム変数です。

a
数値の形状パラメータです。
範囲: $a > 0$

λ
数値の尺度パラメータです。
デフォルト: 1
範囲: $\lambda > 0$

詳細

Weibull 分布の PDF 関数は、Weibull 分布(形状パラメータは *a*、尺度パラメータは 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。式は次のとおりです。

$$PDF('WEIBULL', x, a, \lambda) = \begin{cases} 0 & x < 0 \\ exp\left(-\left(\frac{x}{\lambda}\right)^a\right)\frac{a}{\lambda}\left(\frac{x}{\lambda}\right)^{a-1} & x \geq 0 \end{cases}$$

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
y=pdf('BERN',0,.25);	0.75
y=pdf('BERN',1,.25);	0.25
y=pdf('BETA',0.2,3,4);	1.2288
y=pdf('BINOM',4,.5,10);	0.20508

SAS ステートメント	結果
y=pdf('CAUCHY',2);	0.063662
y=pdf('CHISQ',11.264,11);	0.081686
y=pdf('EXPO',1);	0.36788
y=pdf('F',3.32,2,3);	0.054027
y=pdf('GAMMA',1,3);	0.18394
y=pdf('GENPOISSON',9,1,.7);	0.0150130915
y=pdf('HYPER',2,200,50,10);	0.28685
y=pdf('LAPLACE',1);	0.18394
y=pdf('LOGISTIC',1);	0.19661
y=pdf('LOGNORMAL',1);	0.39894
y=pdf('NEGB',1,.5,2);	0.25
y=pdf('NORMAL',1.96);	0.058441
y=pdf('NORMALMIX',2.3,3,.33,.33,.34, .5,1.5,2.5,.79,1.6,4.3);	0.1166
y=pdf('PARETO',1,1);	1
y=pdf('POISSON',2,1);	0.18394
y=pdf('T',.9,5);	0.24194
y=pdf('TWEEDIE',.8,5);	0.7422908236
y=pdf('UNIFORM',0.25);	1
y=pdf('WALD',1,2);	0.56419
y=pdf('WEIBULL',1,2);	0.73576

関連項目:

関数:

- “LOGCDF 関数” (624 ページ)
- “LOGPDF 関数” (626 ページ)
- “LOGSDF 関数” (628 ページ)
- “CDF 関数” (273 ページ)

- “SDF 関数” (832 ページ)
- “QUANTILE 関数” (778 ページ)
- “SQUANTILE 関数” (856 ページ)

PEEK 関数

32 ビットプラットフォームのメモリアドレスの内容を数値変数に保存します。

カテゴリ: 特殊関数

制限事項: 32 ビットプラットフォームでのみ使用します。

構文

PEEK(*address*<,*length*>)

必須引数

address
メモリアドレスを指定する数値定数、変数または式です。

オプション引数

長さ
データ長を指定する数値定数、変数または式です。

デフォルト: 4 バイトアドレスポインタ

範囲: 2-255

詳細

要求しているメモリ保管場所へのアクセス権がない場合、PEEK 関数は"無効な引数"エラーを返します。

PEEK 関数は、64 ビットプラットフォームでは使用きません。使用しようとすると、この制限が適用されることを示すメッセージが SAS によって口グに書き込まれます。従来のアプリケーションで PEEK を使用している場合、アプリケーションを変更してかわりに PEEKLONG を使用します。PEEKLONG は、32 ビットと 64 ビットの両方のプラットフォームで使用できます。

比較

PEEK 関数は、メモリアドレスの内容を*数値*変数に保存します。PEEKC 関数は、メモリアドレスの内容を*文字*変数に保存します。

注: PEEKLONG は 32 ビットと 64 ビットの両方のプラットフォームで使用できるため、SAS では PEEK のかわりに PEEKLONG を使用することをお勧めします。

サンプル

z/OS 動作環境に固有の次の例では、Communication Vector Table (CVT)のアドレスを表す数値を返します。

```
data _null_;
/* 16 is the location of the CVT address */
y=16;
x=peek(y);
put 'x= ' x hex8.;
run;
```

関連項目:

関数:

- “ADDR 関数” (91 ページ)
- “PEEKC 関数” (719 ページ)

CALL ルーチン:

- “CALL POKE ルーチン” (192 ページ)

PEEKC 関数

32 ビットプラットフォームのメモリアドレスの内容を文字変数に保存します。

カテゴリ: 特殊関数

制限事項: 32 ビットプラットフォームでのみ使用します。

構文

PEEKC(*address*<,*length*>)

必須引数

address
メモリアドレスを指定する数値定数、変数または式です。

オプション引数

長さ
データ長を指定する数値定数、変数または式です。

デフォルト: 8 (LENGTH ステートメントなどで変数長がすでに設定されていない場合)

範囲: 1-255

詳細

要求しているメモリ保管場所へのアクセス権がない場合、PEEKC 関数は"無効な引数"エラーを返します。

PEEKC 関数は、64 ビットプラットフォームでは使用きません。使用しようとすると、この制限が適用されることを示すメッセージが SAS によって口グに書き込まれます。従来のアプリケーションで PEEKC を使用している場合、アプリケーションを変更してかわりに PEEKCLONG を使用します。PEEKCLONG は、32 ビットと 64 ビットの両方のプラットフォームで使用できます。

比較

PEEKC 関数は、メモリアドレスの内容を*文字*変数に保存します。PEEK 関数は、メモリアドレスの内容を*数値*変数に保存します。

注: PEEKCLONG は 32 ビットと 64 ビットの両方のプラットフォームで使用できるため、SAS では PEEKC のかわりに PEEKCLONG を使用することをお勧めします。

サンプル: ASCB のバイトのリスト

z/OS 動作環境に固有の次の例では、PEEK と PEEKC の両方を使用して、Address Space Control Block (ASCB)の最初の 4 バイトを出力します。

```
data _null_;
length y $4;
/* 220x is the location of the ASCB pointer */
x=220x;
y=peekc(peek(x));
put 'y= ' y;
run;
```

関連項目:

関数:

- “ADDR 関数” (91 ページ)
- “PEEK 関数” (718 ページ)

CALL ルーチン:

- “CALL POKE ルーチン” (192 ページ)

PEEKCLONG 関数

32 ビットおよび 64 ビットプラットフォームのメモリアドレスの内容を文字変数に保存します。

カテゴリ: 特殊関数

参照項目: “PEEKCLONG Function: z/OS” in *SAS Companion for z/OS*

構文

PEEKCLONG(*address*<,*length*>)

必須引数

address
バイナリポインタアドレスを含む文字定数、変数または式を指定します。

オプション引数

長さ

文字データの長さを指定する数値定数、変数または式です。

デフォルト: 8

範囲: 1 ~ 32,767

詳細

要求しているメモリ保管場所へのアクセス権がない場合、PEEKCLONG 関数は“無効な引数”エラーを返します。

比較

PEEKCLONG 関数は、メモリアドレスの内容を*文字*変数に保存します。

PEEKLONG 関数は、メモリアドレスの内容を*数値*変数に保存します。この関数は、入力アドレスがメモリ内の整数を参照していることを前提としています。

サンプル

サンプル 1: 32 ビットプラットフォームの例

次の例では、文字変数 Z のポインタアドレスを返します。

```
data _null_;
x='ABCDE';
y=addrlong(x);
z=peekclong(y,2);
put z=;
run;
```

SAS ログの出力は **z=AB** です。

サンプル 2: 64 ビットプラットフォームの例

z/OS 動作環境に固有の次の例では、文字変数 Y のポインタアドレスを返します。

```
data _null_;
length y $4;
x220addr=put(220x,pib4.);
ascb=peeklong(x220addr);
ascbaddr=put(ascb,pib4.);
y=peekclong(ascbaddr);
run;
```

SAS ログの出力は **y='ASCB'**です。

関連項目:

関数:

- “PEEKLONG 関数” (722 ページ)

PEEKLONG 関数

32 ビットおよび 64 ビットプラットフォームのメモリアドレスの内容を数値変数に保存します。

カテゴリ: 特殊関数

参照項目: “PEEKLONG Function: Windows” in *SAS Companion for Windows*
“PEEKLONG Function: UNIX” in *SAS Companion for UNIX Environments*
“PEEKLONG Function: z/OS” in *SAS Companion for z/OS*

構文

PEEKLONG(*address*<,*length*>)

必須引数

address
バイナリポインタアドレスを含む文字定数、変数または式を指定します。

オプション引数

長さ
文字データの長さを指定する数値定数、変数または式です。

デフォルト: 32 ビットコンピュータの場合は 4、64 ビットコンピュータの場合は 8。

範囲: 32 ビットコンピュータの場合は 1-4、64 ビットコンピュータの場合は 1-8。

詳細

要求しているメモリ保管場所へのアクセス権がない場合、PEEKLONG 関数は“無効な引数”エラーを返します。

比較

PEEKLONG 関数は、メモリアドレスの内容を*数値*変数に保存します。この関数は、入力アドレスがメモリ内の整数を参照していることを前提としています。

PEEKCLONG 関数は、メモリアドレスの内容を*文字*変数に保存します。この関数は、入力アドレスが文字データを参照していることを前提としています。

サンプル

サンプル 1: 32 ビットプラットフォームの例

次の例では、数値変数 Z のポインタアドレスを返します。

```
data _null_;
length y $4;
y=put(1,IB4.);
addry=addrlong(y);
z=peeklong(addry,4);
```

```
put z=;
run;
```

SAS は出力 **z=1** を口グに書き込みます。

サンプル 2: 64 ビットプラットフォームの例

z/OS 動作環境に固有の次の例では、数値変数 X のポインタアドレスを返します。

```
data _null_;
x=peeklong(put(16,pib4.));
put x=hex8.;
run;
```

SAS は出力 **x=00FCFCB0** を口グに書き込みます。

関連項目:

関数:

- “PEEKCLONG 関数” (720 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: 組み合わせ関数

構文

PERM(*n* <,*r*>)

必須引数

n

選択するサンプルの要素の合計数を表す整数です。

オプション引数

r

選択した要素数を表す整数値です。*r* を省略すると、関数は *n* の階乗を返します。

制限事項: $r \leq n$

詳細

PERM 関数の数学的表現は、次の式によって得られます。

$$PERM(n, r) = \left(\frac{n!}{(n - r)!}\right)$$

$n \geq 0$、$r \geq 0$ および $n \geq r$ 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。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=perm(5,1);	5
x=perm(5);	120
x=perm(5,2)	20

関連項目:

関数:

- “COMB 関数” (303 ページ)
- “FACT 関数” (392 ページ)
- “LPERM 関数” (631 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: 財務関数

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。

number-of-periods
支払期間の数を指定します。*number-of-periods* は正の整数値とする必要があります。

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。

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。

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。

サンプル

- 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。

```
Payment1 = PMT(0.08/12, 10, 10000, 0, 0);

Payment1 = PMT(0.08/12, 10, 10000);
```

これらの計算の戻り値は 1037.03 です。

- 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。

```
Payment2 = PMT(0.08/12, 10, 10000, 0, 1);
```

計算によって 1030.16 という値が返されます。

- 名目年利が 12%で払い戻し期間が 5 か月の$5,000 のローンの支払いは次のように計算されます。

```
Payment3 = PMT(.01, 5, -5000);
```

計算によって-1030.20 という値が返されます。

- 有利子期間が 18 年、名目年利が 6%、18 年後の累計が$50,000 となる月次貯蓄の支払いは次のように計算されます。

```
Payment4 = PMT(0.06/12, 216, 0, 50000, 0);
```

計算によって 129.081 という値が返されます。

POINT 関数

NOTE 関数で識別されたオブザベーションを検索します。

カテゴリ: SAS ファイル I/O 関数

構文

POINT(*data-set-id*,*note-id*)

必須引数

data-set-id
OPEN 関数が返すデータセット識別子を指定する数値変数です。

note-id
NOTE 関数がオブザベーションに割り当てる識別子を指定する数値変数です。

詳細

POINT は操作に成功すると 0 を返し、失敗すると≠0 を返します。POINT は SAS データセットから読み込むプログラムの準備を整えます。FETCH または

FETCHOBS を使用した読み込みが行われるまで、DDV(データセットデータデータベクトル)は更新されません。

サンプル

この例では、SAS データセット MYDATA のオブザベーションのうち、最後に読み込んだオブザベーションの ID を、NOTE の呼び出しによって取得します。ここでは POINT を呼び出して該当オブザベーションをポイントし、FETCH を呼び出してポインタが付けられたオブザベーションを返します。

```
%let dsid=%sysfunc(open(mydata,i));
%let rc=%sysfunc(fetch(&dsid));
%let noteid=%sysfunc(note(&dsid));
...more macro statements...
%let rc=%sysfunc(point(&dsid,&noteid));
%let rc=%sysfunc(fetch(&dsid));
...more macro statements...
%let rc=%sysfunc(close(&dsid));
```

関連項目:

関数:

- “DROPNOTE 関数” (380 ページ)
- “NOTE 関数” (674 ページ)
- “OPEN 関数” (697 ページ)

POISSON 関数

Poisson 分布の確率を返します。

カテゴリ: 確率

参照項目: “CDF 関数” (273 ページ), “PDF 関数” (703 ページ)

構文

POISSON(*m*,*n*)

必須引数

m
数値の平均パラメータです。
範囲: $m \geq 0$

n
整数のランダム変数です。
範囲: $n \geq 0$

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。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=poisson(1,2);	0.9196986029

関連項目:

関数:

- “CDF 関数” (273 ページ)
- “LOGCDF 関数” (624 ページ)
- “LOGPDF 関数” (626 ページ)
- “LOGSDF 関数” (628 ページ)
- “PDF 関数” (703 ページ)
- “SDF 関数” (832 ページ)

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: 財務関数

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。

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。*period* は、*number-of-periods* の値以下の正の整数値で指定する必要があります。

number-of-periods
支払期間の数を指定します。*number-of-periods* は正の整数値とする必要があります。

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。

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。

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。

サンプル

- $2,000 の 2 年ローンで名目年利が 10%の場合、最初の月次均等支払いの元本支払い額は次のように計算されます。

  ```
  PrincipalPayment = PPMT(0.1/12, 1, 24, 2000);
  ```

  計算によって 75.62 という値が返されます。

- $20,000 の 3 年ローンで期首支払いの場合、元本支払い額は次のように計算されます。

  ```
  PrincipalPayment2 = PPMT(0.1./12, 1, 36, 20000, 0, 1);
  ```

  この計算で、最初の支払いで支払った元本の値として 640.10 が返されます。

- 3 年経過後に未払い残高が$5,000 の月末支払いローンで支払う元本支払い額は次のように計算されます。

  ```
  PrincipalPayment3 = PPMT(0.1/12, 1, 36, 20000, 5000, 0);
  ```

  この計算で、最初の支払いで支払った元本の値として 389.914 が返されます。

PROBBETA 関数

ベータ分布の確率を返します。

カテゴリ: 確率

参照項目: “CDF 関数” (273 ページ), “PDF 関数” (703 ページ)

構文

PROBBETA(*x*,*a*,*b*)

必須引数

x
数値のランダム変数です。
範囲: $0 \le x \le 1$

a
数値の形状パラメータです。
範囲: $a > 0$

b
数値の形状パラメータです。

範囲: $b > 0$

詳細

PROBBETA 関数は、形状パラメータが 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。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=probbeta(.2,3,4);	0.09888

関連項目:

関数:

- “CDF 関数” (273 ページ)
- “LOGCDF 関数” (624 ページ)
- “LOGPDF 関数” (626 ページ)
- “LOGSDF 関数” (628 ページ)
- “PDF 関数” (703 ページ)
- “SDF 関数” (832 ページ)

PROBBNML 関数

二項分布の確率を返します。

カテゴリ: 確率

参照項目: “CDF 関数” (273 ページ), “PDF 関数” (703 ページ)

構文

PROBBNML(*p*,*n*,*m*)

必須引数

p

数値の成功率パラメータです。

範囲: $0 \leq p \leq 1$

n

整数値の独立した Bernoulli トライアル数パラメータです。

範囲: $n > 0$

m

整数値の成功のランダム変数です。

範囲: $0 \le m \le n$

詳細

PROBBNML 関数は、成功確率が *p*、トライアル数が *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。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=probbnml(0.5,10,4);	0.376953125

関連項目:

関数:

- “CDF 関数” (273 ページ)
- “LOGCDF 関数” (624 ページ)
- “LOGPDF 関数” (626 ページ)
- “LOGSDF 関数” (628 ページ)
- “PDF 関数” (703 ページ)
- “SDF 関数” (832 ページ)

PROBBNRM 関数

2 変量正規分布の確率を返します。

カテゴリ: 確率

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。

y
数値の定数、変数または式を指定します。

r
数値の相関係数です。
範囲: $-1 \le r \le 1$

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 がそれぞれランダム変数 x と y を表す PROBBNRM 関数を示しています。

$$PROBBNRM(x,\ y,\ r) = \frac{1}{2\ \pi\sqrt{1 - r^2}} \int_{-\infty}^{x} \int_{-\infty}^{y} exp\left[- \frac{u^2 - 2ruv + v^2}{2(1 - r^2)} \right] dv\ du$$

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
p=probbnrm(.4, −.3, .2); put p;	0.2783183345

関連項目:

関数:

- “CDF 関数” (273 ページ)
- “LOGCDF 関数” (624 ページ)
- “LOGPDF 関数” (626 ページ)
- “LOGSDF 関数” (628 ページ)
- “PDF 関数” (703 ページ)
- “SDF 関数” (832 ページ)

PROBCHI 関数

χ 2 乗分布の確率を返します。

カテゴリ: 確率

参照項目: “CDF 関数” (273 ページ), “PDF 関数” (703 ページ)

構文

PROBCHI(*x*,*df* <,*nc*>)

必須引数

x

数値のランダム変数です。

範囲: $x \geq 0$

df

数値の自由度パラメータです。

範囲: $df > 0$

オプション引数

nc
数値の非心度パラメータです(省略可能)。

範囲: $nc \geq 0$

詳細

PROBCHI 関数は、自由度が *df*、非心度パラメータが *nc* の $\chi 2$ 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 *df* を使用できます。パラメータ *nc* (省略可能)が指定されていないかゼロの場合、心度 $\chi 2$ 乗分布の値が返されます。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=probchi(11.264,11);	0.5785813293

関連項目:

関数:

- “CDF 関数” (273 ページ)
- “LOGCDF 関数” (624 ページ)
- “LOGPDF 関数” (626 ページ)
- “LOGSDF 関数” (628 ページ)
- “PDF 関数” (703 ページ)
- “SDF 関数” (832 ページ)

PROBF 関数

F 分布の確率を返します。

カテゴリ: 確率

参照項目: “CDF 関数” (273 ページ), “PDF 関数” (703 ページ)

構文

PROBF(*x*,*ndf*,*ddf* <,*nc*>)

必須引数

x
数値のランダム変数です。

範囲: $x \geq 0$

ndf
数値の分子の自由度パラメータです。

範囲: $ndf > 0$

ddf
数値の分母の自由度パラメータです。

範囲: $ddf > 0$

オプション引数

nc
数値の非心度パラメータです(省略可能)。

範囲: $nc \geq 0$

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 *ndf* と *ddf* に非整数を指定できます。パラメータ *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から求められます。

```
p=1-probf(x,ndf,ddf);
```

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=probf(3.32,2,3);	0.8263933602

関連項目:

関数:

- “CDF 関数” (273 ページ)
- “LOGCDF 関数” (624 ページ)
- “LOGPDF 関数” (626 ページ)
- “LOGSDF 関数” (628 ページ)
- “PDF 関数” (703 ページ)
- “SDF 関数” (832 ページ)

PROBGAM 関数

ガンマ分布の確率を返します。

カテゴリ: 確率

参照項目: “CDF 関数” (273 ページ), “PDF 関数” (703 ページ)

構文

PROBGAM(*x*,*a*)

必須引数

x

数値のランダム変数です。

範囲: $x \geq 0$

a

数値の形状パラメータです。

範囲: $a > 0$

詳細

PROBGAM 関数は、形状パラメータが 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。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=probgam(1,3);	0.0803013971

関連項目:

関数:

- “CDF 関数” (273 ページ)
- “LOGCDF 関数” (624 ページ)
- “LOGPDF 関数” (626 ページ)
- “LOGSDF 関数” (628 ページ)
- “PDF 関数” (703 ページ)
- “SDF 関数” (832 ページ)

PROBHYPR 関数

超幾何分布の確率を返します。

カテゴリ: 確率

参照項目: “CDF 関数” (273 ページ), “PDF 関数” (703 ページ)

構文

PROBHYPR(*N*,*K*,*n*,*x*<,*r*>)

必須引数

N

整数の母集団サイズパラメータです。

範囲: $N \geq 1$

K

整数値の目的カテゴリ内の項目数パラメータです。

範囲: $0 \leq K \leq N$

n

整数のサンプルサイズパラメータです。

範囲: $0 \leq n \leq N$

x

整数のランダム変数です。

範囲: $max(0, K + n-N) \leq x \leq min(K,n)$

オプション引数

r

任意の数値のオッズ比パラメータです。

範囲: $r \geq 0$

詳細

PROBHYPR 関数は、母集団のサイズが *N*、項目数が *K*、サンプルサイズが *n*、オッズ比が *r* からの拡張超幾何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。パラメータ *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、通常の超幾何分布の値が返されます。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=probhypr(200,50,10,2);	0.5236734081

関連項目:

関数:

- “CDF 関数” (273 ページ)
- “LOGCDF 関数” (624 ページ)
- “LOGPDF 関数” (626 ページ)
- “LOGSDF 関数” (628 ページ)
- “PDF 関数” (703 ページ)

- “SDF 関数” (832 ページ)

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

構文

PROBIT(*p*)

必須引数

p
数値の確率です。
範囲: 0<*p*<1

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-8.222～7.941 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。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=probit(.025);	-1.959963985
x=probit(1.e-7);	-5.199337582

関連項目:

関数:

- “CDF 関数” (273 ページ)
- “LOGCDF 関数” (624 ページ)
- “LOGPDF 関数” (626 ページ)
- “LOGSDF 関数” (628 ページ)
- “PDF 関数” (703 ページ)
- “SDF 関数” (832 ページ)

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

構文

PROBMC(*distribution*,*q*,*prob*,*df*,*nparms*<, *parameters*>)

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
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

q
分布の分位点です。
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)。

prob
分布の左側確率です。
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)。

df
自由度です。
注: 欠損値は無限値として解釈されます。

nparms
処理数です。
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

パラメータ

サンプルサイズが不揃いのケースを扱うのに指定する必要がある *nparms* パラメータのセットです(省略可能)。*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。

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

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。

$$d\mu_v(x) = \frac{v^{\frac{v}{2}}}{\Gamma\left(\frac{v}{2}\right)2^{\frac{v}{2}-1}} x^{v-1}\varepsilon^{-\frac{vx^2}{2}}\,dx$$

$$\phi(x) = \frac{1}{\sqrt{2\pi}}\varepsilon^{-\frac{x^2}{2}}$$

$$\Phi(x) = \int_{-\infty}^{x}\phi(u)\,du$$

平均分析の計算

平均分析(ANOM)は、*i*番目のサンプルサイズが n_i となる *k*(ガウス)サンプルのデータで使用されます。$I = \sqrt{-1}$ のようになります。分布関数[1, 2, 3, 4, 5]は *k* 次元の多変量Tベクトルの最大絶対値の CDF です。このとき、自由度は v 、関連相関行列は $\rho_{ij} = -\alpha_i\alpha_j$ です。この式は次のように表すことができます。

$$\begin{aligned} prob &= r\left(|t_1| < h,\ |t_2| < h,\ \ldots,\ |t_k| < h\right) \\ &= \int_0^{\infty}\left\{\int_0^{\infty}\prod_{j=0}^{j=k} g(sh,\ y,\ \alpha_j)\phi(y)\,dy\right\}d\mu_v(s) \end{aligned}$$

前述の式には次の関係が適用されます。

$$g(sh,\ y,\ \alpha_j) = \Phi\left(\frac{sh\ -\ y\alpha_j}{\sqrt{1+\alpha_j^2}}\right) - \Phi\left(\frac{-sh\ -\ y\alpha_j}{\sqrt{1+\alpha_j^2}}\right)$$

ここで、$\Gamma(.)$ 、$\phi(.)$ および $\Phi(.)$ はそれぞれ正規分布のガンマ関数、密度および CDF です。

$v = \infty$ のとき、分布は次に減少します。

$$r\left(|t_1| < h,\ |t_2| < h,\ ...,\ |t_k| < h\right) = \int_0^{\infty} \prod_{j=0}^{j=k} g(h,\ y,\ \alpha_j)\phi(y)dy$$

前述の式には次の関係が適用されます。

$$g(h,\ y,\ \alpha_j) = \Phi\left(\frac{h\ -\ y\alpha_j}{\sqrt{1+\alpha_j^2}}\right) - \Phi\left(\frac{-h\ -\ y\alpha_j}{\sqrt{1+\alpha_j^2}}\right)$$

均衡しているケースでは、分布は次に減少します。

$$r\left(|t_1| < h,\ |t_2| < h,\ ...,\ |t_n| < h\right) = \int_0^{\infty} f(h,\ y,\ \rho)^n \phi(y)dy$$

前述の式には次の関係が適用されます。

$$f(h,\ y,\ \rho) = \Phi\left(\frac{h\ -\ y\sqrt{\rho}}{\sqrt{1+\rho}}\right) - \Phi\left(\frac{-h\ -\ y\sqrt{\rho}}{\sqrt{1+\rho}}\right)$$

および $\rho = \frac{1}{n\ -\ 1}$

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 α 値を示す数値です(省略可能)。"平均分析の計算" (738 ページ)を参照。

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 $\lambda_1, ..., \lambda_k$、*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。

$$prob = \int_{0}^{\infty}\int_{-\infty}^{\infty} \phi(y)[\Phi(y + \sqrt{2qx})]^k dy\, du_v(x)$$

- 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 $\lambda_1, ..., \lambda_k$、*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。

$$prob = \int_{-\infty}^{\infty} \phi(y)[\Phi(y + \sqrt{2q})]^k dy$$

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 $\lambda_1, ..., \lambda_k$、*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。

$$prob = \int_{0}^{\infty}\int_{-\infty}^{\infty} \phi(y) \prod_{i=1}^{k} \left[\Phi\left(\frac{\lambda_i y + qx}{\sqrt{1 - \lambda_i^2}}\right) - \Phi\left(\frac{\lambda_i y - qx}{\sqrt{1 - \lambda_i^2}}\right)\right] dy\, du_v(x)$$

- 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、*nparms* の設定は *k*、*df* の設定値は *v* です。式は次のとおりです。

$$prob = \int_{0}^{\infty}\int_{-\infty}^{\infty} \phi(y)[\Phi(y + \sqrt{2qx}) - \Phi(y - \sqrt{2qx})]^k dy\, du_v(x)$$

- 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 $\lambda_1, ..., \lambda_k$、*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。

$$prob = \int_{-\infty}^{\infty} \phi(y)[\Phi(y + \sqrt{2q}) - \Phi(y - \sqrt{2q})]^k dy$$

分割された範囲の計算

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)。

$$prob = \int_0^{\infty} \prod_{i=1}^{i=k} \left(\int_{-\infty}^{\infty} k\phi(y)(\Phi(y) - \Phi(y - qx))^{k-1} dy \right)^{n_i} d\mu_v(x)$$

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* 値を示す数値です(省略可能)。"分割された範囲の計算" (741 ページ)を参照してください。

スチューデント化された範囲

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。

- 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。

$$prob = \int_0^{\infty} \int_{-\infty}^{\infty} k\phi(y)[\Phi(y) - \Phi(y - qx)]^{k-1} dy \, du_v(x)$$

- 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 $\sigma_1, \ldots, \sigma_k$、*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。

$$prob = \int_{-\infty}^{\infty} k\phi(y)[\Phi(y) - \Phi(y - q)]^{k-1} dy$$

スチューデント化された最大係数

- 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 $\sigma_1, ..., \sigma_k$、*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。

$$prob = \int_0^{\infty} \prod_{i=1}^{k} \left[2\Phi\left(\frac{qx}{\sigma_i}\right) - 1 \right] d\mu_v(x)$$

- 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、*nparms* の設定は *k*、*df* の設定値は *v* です。式は次のとおりです。

$$prob = \int_0^{\infty} [2\Phi(qx) - 1]^k d\mu_v(x)$$

- 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 $\sigma_1, ..., \sigma_k$、*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 検定

PROBMC は、Williams(1971、1972)によって定義されている分布(“リファレンス” (971 ページ) を参照)の確率または分位点を計算します。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ため使用します。

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)確率変数であるものとします。Y_k は次の式で表されるそれらの平均であるものとします。

$$Y_k = \frac{X_1 + X_2 + ... + X_k}{k}$$

次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
$\frac{1}{2}\nu S^2$ が χ^2 変数です(自由度 ν)。

Williams(1971)(“リファレンス” (971 ページ) を参照)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。

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 は次のように定義されるものとします。

$$U_i = \frac{X_1 + X_2 + \ldots + X_i}{i}, \quad i = 1, 2, \ldots, k$$

d に実数を使用し、$Y_k > d$ の確率の再帰式を記述できます。

$$\begin{aligned}\Pr(Y_k > d) &= \Pr(U_1 > d) \\ &\quad + \Pr(U_2 > d,\ U_1 < d) \\ &\quad + \Pr(U_3 > d,\ U_2 < d,\ U_1 < d) \\ &\quad + \ldots \\ &\quad + \Pr(U_k > d,\ U_{k-1} < d,\ \ldots,\ U_1 < d) \\ &= \Pr(Y_{k-1} > d) + \Pr(X_k + (k-1)U_{k-1} > kd)\end{aligned}$$

この確率を計算するには、次の N(0,1)密度関数から始めます。

$$D(U_1 = x) = \phi(x)$$

次に重畳を再帰的に計算します。

$$D(U_k = x,\ U_{k-1} < d,\ \ldots,\ U_1 < d) =$$

$$\int_{-\infty}^{d} D(U_{k-1} = y,\ U_{k-2} < d,\ \ldots,\ U_1 < d)(k-1)\phi(kx - (k-1)y)\,dy$$

この一連の重畳から、前掲の $\Pr(Y_k < d)$ 再帰式のすべての要素を計算できます。

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に別の重畳を必要とします。

$$\Pr((Y_k - Z) > d) = \int_{-\infty}^{\infty} \Pr(Y_k > \sqrt{2}d + y)\phi(y)\,dy$$

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に別の重畳を必要とします。

$$\Pr((Y_k - Z) > tS) = \int_{0}^{\infty} \Pr((Y_k - Z) > ty)\,d\mu_\nu(y)$$

$\nu = \infty$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

サンプル

サンプル 1: PROBMC を使用した確率の計算

次の例では、確率の計算方法を示します。

```
data probs;
array par{5};
par{1}=.5;
par{2}=.51;
par{3}=.55;
par{4}=.45;
par{5}=.2;
df=40;
q=1;
do test="dunnett1","dunnett2", "maxmod";
prob=probmc(test, q, ., df, 5, of par1-par5);
put test $10. df q e18.13 prob e18.13;
end;
run;
```

SAS は次の結果を口グに書き込みます。

ログ 2.17 PROBMC の確率

```
DUNNETT1 40 1.00000000000E+00 4.82992196083E-01
DUNNETT2 40 1.00000000000E+00 1.64023105316E-01
MAXMOD 40 1.00000000000E+00 8.02784203408E-01
```

サンプル 2: 平均分析の計算

```
data _null_;
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=;
run;
```

SAS は次の出力を口グに書き込みます。

ログ 2.18 平均分析の出力

```
q1=2.7895061016
q2=2.4549961967
q3=2.4549961967
q4=2.4532319994
```

サンプル 3: 平均の比較

この例では、グループ平均を比較して有意な差異がどこにあるかを調べる方法を示します。この例のデータは Duncan(1955)の論文から引用しています。また、Hochberg and Tamhane(1987)にも記載されています(この関数の末尾の参考文献セクションを参照してください)。

次の値はグループ平均です。

- 49.6
- 71.2

- 67.6
- 61.5
- 71.3
- 58.1
- 61.0

このデータでは、平均平方誤差は $\nu = 30$ のとき $s^2 = 79.64$ ($s = 8.924$)です。

```
data duncan;
array tr{7}$;
array mu{7};
n=7;
do i=1 to n;
input tr{i} $1. mu{i};
end;
input df s alpha;
prob= 1-alpha;
/* compute the interval */
x = probmc("RANGE", ., prob, df, 7);
w = x * s / sqrt(6);
/* compare the means */
do i = 1 to n;
do j = i + 1 to n;
dmean = abs(mu{i} - mu{j});
if dmean >= w then do;
put tr{i} tr{j} dmean;
end;
end;
end;
datalines;
A 49.6
B 71.2
C 67.6
D 61.5
E 71.3
F 58.1
G 61.0
30 8.924 .05
;
run;
```

SAS は次の出力を口グに書き込みます。

ログ 2.19 *グループ間差異*

```
A B 21.6
A C 18
A E 21.7
```

サンプル 4: 分割された範囲の計算

```
data _null_;
q1=probmc('partrange',.,0.9,.,4,3,4,5,6); put q1=;
q2=probmc('partrange',.,0.9,12,4,3,4,5,6); put q2=;
run;
```

SAS は次の出力を口グに書き込みます。

ログ 2.20 分割された範囲の出力

```
q1=4.1022397989
q2=4.7888626338
```

サンプル 5: 信頼区間の計算

この例では、Dunnett 検定の片側検定と両側検定で 95%信頼区間を計算する方法を示します。この例のデータは Dunnett(1955)を引用しています。また、Hochberg and Tamhane(1987)にも記載されています(この関数の末尾の参考文献セクションを参照してください)。データは、3 つの動物グループの血球数測定値です。次の表に示すとおり、最初の 2 つのグループには異なる薬物を投与し、第 3 グループをコントロールグループとします。3 つのグループの動物の数は不揃いです。

投与グループ:	薬物 A	薬物 B	コントロール
	9.76	12.80	7.40
	8.80	9.68	8.50
	7.68	12.16	7.20
	9.36	9.20	8.24
		10.55	9.84
			8.32
グループ平均	8.90	10.88	8.25
n	4	5	6

平均平方誤差は $\nu = 12$ のとき $s^2 = 1.3805$ ($s = 1.175$)です。

```
data a;
array drug{3}$;
array count{3};
array mu{3};
array lambda{2};
array delta{2};
array left{2};
array right{2};
/* input the table */
do i = 1 to 3;
input drug{i} count{i} mu{i};
end;
/* input the alpha level, */
/* the degrees of freedom, */
/* and the mean square error */
input alpha df s;

/* from the sample size, */
/* compute the lambdas */
```

```
do i = 1 to 2;
lambda[i] = sqrt(count[i]/
(count[i] + count[3]));
end;
/* run the one-sided Dunnett's test */
test="dunnett1";
x = probmc(test, ., 1 - alpha, df,
2, of lambda1-lambda2);
do i = 1 to 2;
delta[i] = x * s *
sqrt(1/count[i] + 1/count[3]);
left[i] = mu[i] - mu[3] - delta[i];
end;
put test $10. x left[1] left[2];
/* run the two-sided Dunnett's test */
test="dunnett2";
x = probmc(test, ., 1 - alpha, df,
2, of lambda1-lambda2);
do i=1 to 2;
delta[i] = x * s *
sqrt(1/count[i] + 1/count[3]);
left[i] = mu[i] - mu[3] - delta[i];
right[i] = mu[i] - mu[3] + delta[i];
end;
put test $10. left[1] right[1];
put test $10. left[2] right[2];
datalines;
A 4 8.90
B 5 10.88
C 6 8.25
0.05 12 1.175
;
run;
```

SAS は次の出力を口グに書き込みます。

ログ 2.21 *信頼区間*

```
DUNNETT1 2.1210448226 -0.958726041 1.1208812046
DUNNETT2 -1.256408109 2.5564081095
DUNNETT2 0.8416306717 4.4183693283
```

サンプル 6: Williams 検定の計算

次の例では、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	平均値
X_0	10.4
X_1	9.9
X_2	10.0

処理	平均値
X_3	10.6
X_4	11.4
X_5	11.9
X_6	11.7

自由度は$(7-1)(8-1)=42$ で、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。

- $X_0 > X_1$ であるから、$X_{0,1}=(X_0+X_1)/2=10.15$ である。
- $X_{0,1} > X_2$ であるから、$X_{0,1,2}=(X_0+X_1+X_2)/3=(2X_{0,1}+X_2)/3=10.1$ である。
- $X_{0,1,2} < X_3 < X_4 < X_5$
- $X_5 > X_6$ であるから、$X_{5,6}=(X_5+X_6)/2=11.8$ である。

これで順序制約が満たされました。

対立仮説では、最大尤度推定は次のとおりです。

- $M_0=M_1=M_2=X_{0,1,2}=10.1$
- $M_3=X_3=10.6$
- $M_4=X_4=11.4$
- $M_5=M_6=X_{5,6}=11.8$

次に $t=(11.8-10.4)/\sqrt{2s^2/8}=2.60$ を計算すると、$k=6$、$\nu=42$、$t=2.60$ に対応する確率は.9924467341 となります。すなわち、物質に対する強い反応があることが示されています。また、次の表に示すように、上の 5%および裾の 1%の分位点も計算できます。

SAS ステートメント	結果
prob=probmc("williams",2.6,.,42,6);	0.9924466872
quant5=probmc("williams",.,.95,42,6);	1.806562536
quant1=probmc("williams",.,.99,42,6);	2.490908273

関連項目:

関数:

- “CDF 関数” (273 ページ)
- “LOGCDF 関数” (624 ページ)
- “LOGPDF 関数” (626 ページ)
- “LOGSDF 関数” (628 ページ)

- “PDF 関数” (703 ページ)
- “SDF 関数” (832 ページ)

リファレンス

Guirguis, G. H., and R. D. Tobias. “On the computation of the distribution for the analysis of means.” 2004. *Communications in Statistics: Simulation and Computation* 33: 861-255.

Nelson, P. R. “Numerical evaluation of an equicorrelated multivariate non-central t distribution.” 1981. *Communications in Statistics: Part B - Simulation and Computation* 10: 41-255.

Nelson, P. R. “Exact critical points for the analysis of means.” 1982. *Communications in Statistics: Part A - Theory and Methods* 11: 699-255.

Nelson, P. R. “An Approximation for the Complex Normal Probability Integral.” 1982a. *BIT* 22(1): 94-255.

Nelson, P. R. “Application of the analysis of means.” 1988. *Proceedings of the SAS Users Group International Conference* 13: 225-255.

Nelson, P. R. “Numerical evaluation of multivariate normal integrals with correlations.” 1991. *The Frontiers of Statistical Scientific Theory and Industrial Applications* 2: 97-255.

Nelson, P. R. “Additional Uses for the Analysis of Means and Extended Tables of Critical Values.” 1993. *Technometrics* 35: 61-255.

PROBNEGB 関数

負数二項分布の確率を返します。

カテゴリ: 確率

参照項目: “CDF 関数” (273 ページ)

構文

PROBNEGB(*p*,*n*,*m*)

必須引数

p

数値の成功率パラメータです。

範囲: $0 \leq p \leq 1$

n

整数値の成功パラメータです。

範囲: $n \geq 1$

m

正の整数のランダム変数で、失敗の数です。

範囲: $m \geq 0$

詳細

PROBNEGB 関数は、成功確率が *p*、成功数が *n* の負数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、負数二項分布からの 2 つの確率(*m* の確率と *m*-1 の確率)の差異を計算します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=probnegb(0.5,2,1);	0.5

関連項目:

関数:

- “CDF 関数” (273 ページ)
- “LOGCDF 関数” (624 ページ)
- “LOGPDF 関数” (626 ページ)
- “LOGSDF 関数” (628 ページ)
- “PDF 関数” (703 ページ)
- “SDF 関数” (832 ページ)

PROBNORM 関数

標準正規分布の確率を返します。

カテゴリ: 確率

参照項目: “CDF 関数” (273 ページ)

構文

PROBNORM(*x*)

必須引数

x
数値のランダム変数です。

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。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=probnorm(1.96);	0.9750021049

関連項目:

関数:

- “CDF 関数” (273 ページ)
- “LOGCDF 関数” (624 ページ)
- “LOGPDF 関数” (626 ページ)
- “LOGSDF 関数” (628 ページ)
- “PDF 関数” (703 ページ)
- “SDF 関数” (832 ページ)

PROBT 関数

t 分布の確率を返します。

カテゴリ: 確率

参照項目: “CDF 関数” (273 ページ), “PDF 関数” (703 ページ)

構文

PROBT(*x*,*df* <,*nc*>)

必須引数

x
数値のランダム変数です。

df
数値の自由度パラメータです。
範囲: $df > 0$

オプション引数

nc
数値の非心度パラメータです(省略可能)。

詳細

PROBT 関数は、自由度が *df* で非心度パラメータが *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 *df* に非整数を指定できます。パラメータ *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から求められます。

```
p=(1-probt(abs(x),df))*2;
```

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=probt(0.9,5);	0.7953143998

関連項目:

関数:

- “CDF 関数” (273 ページ)
- “LOGCDF 関数” (624 ページ)
- “LOGPDF 関数” (626 ページ)
- “LOGSDF 関数” (628 ページ)
- “PDF 関数” (703 ページ)
- “SDF 関数” (832 ページ)

PROPCASE 関数

引数のすべての単語を適切に大文字と小文字に変換します。

カテゴリ: 文字関数

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

PROPCASE(*argument*<,*delimiters*>)

必須引数

引数
文字定数、変数または式を指定します。

オプション引数

delimiter
1 つ以上の区切り文字を引用符で囲んで指定します。デフォルトの区切り文字は空白、フォワードスラッシュ、ハイフン、開始かっこ、ピリオド、タブです。

ヒント: この引数を使用すると、空白を含め、デフォルトの区切り文字は無効になります。

詳細

返される変数の長さ

DATA ステップで、まだ長さが割り当てられていない変数に PROPCASE 関数から値が返される場合、その変数の長さは PROPCASE に渡される最初の引数の長さに設定されます。

基本

PROPCASE 関数は、文字引数をコピーし、すべての大文字を小文字に変換します。次に、空白、フォワードスラッシュ、ハイフン、開始かっこ、ピリオド、タブに続く最初の文字を大文字に変換します。PROPCASE は修正された値を返します。

第 2 引数を使用すると、デフォルトの区切り文字は無効になります。

PROPCASE 関数の結果は、有効な変換テーブルに直接依存し(“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)、ENCODING と LOCALE のシステムオプションに間接的に依存します。

サンプル

サンプル 1: ワードの大文字小文字の変更

次の例では、PROPCASE がワードの大文字小文字をどのように扱うかを示しています。

```
data _null_;
input place $ 1-40;
name=propcase(place);
put name;
datalines;
INTRODUCTION TO THE SCIENCE OF ASTRONOMY
VIRGIN ISLANDS (U.S.)
SAINT KITTS/NEVIS
WINSTON-SALEM, N.C.
;
run;
```

SAS は次の出力を口グに書き込みます。

```
Introduction To The Science Of Astronomy
Virgin Islands (U.S.)
Saint Kitts/Nevis
Winston-Salem, N.C.
```

サンプル 2: PROPCASE での第 2 引数の使用

次の例では、空白、ハイフンおよび一重引用符を第 2 引数に使用し、O'Keeffe や Burne-Jones などの名前が正しく記述されるようにします。

```
data names;
infile datalines dlm='#';
input CommonName : $20. CapsName : $20.;
PropcaseName=propcase(capsname, " -'");
datalines;
Delacroix, Eugene# EUGENE DELACROIX
O'Keeffe, Georgia# GEORGIA O'KEEFFE
Rockwell, Norman# NORMAN ROCKWELL
```

```
Burne-Jones, Edward# EDWARD BURNE-JONES
;
proc print data=names noobs;
title 'Names of Artists';
run;
```

画面 2.45 *PROPCASE で第 2 引数を使用したときの結果を示す出力*

Names of Artists

CommonName	CapsName	PropcaseName
Delacroix, Eugene	EUGENE DELACROIX	Eugene Delacroix
O'Keeffe, Georgia	GEORGIA O'KEEFFE	Georgia O'Keeffe
Rockwell, Norman	NORMAN ROCKWELL	Norman Rockwell
Burne-Jones, Edward	EDWARD BURNE-JONES	Edward Burne-Jones

関連項目:

関数:

- “UPCASE 関数” (902 ページ)
- “LOWCASE 関数” (630 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

構文

PRXCHANGE(*perl-regular-expressionregular-expression-id*, *times*, *source*)

必須引数

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。

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。

時間
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。

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。

ソース
検索する文字定数、変数または式を指定します。

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* で変数 *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* を使って変数 *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。

パターン照合の詳細については、“Perl 正規表現(PRX)を使用したパターン照合 ” (42 ページ)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、DATA ステップ、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。

一致の実行

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は、“Perl 正規表現(PRX)のメタ文字テーブル” (973 ページ)の表の簡易リストで確認できます。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

PRXCHANGE 関数は CALL PRXCHANGE ルーチンに似ていますが、関数の場合にはパターン照合の置換値をパラメータの 1 つとしてではなく、引数の戻り値として返す点が異なります。

Perl 正規表現(PRX)の関数と CALL ルーチンは、連携してパターンと一致する文字列を操作します。これらの関数および CALL ルーチンのリストと概要説明については、“カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の文字列マッチングカテゴリを参照してください。

サンプル

サンプル 1: DATA ステップを使用して姓名の順序を入れ替える

DATA ステップを使用して名と姓の順序を入れ替える例を次に示します。

```
 /* Create a data set that contains a list of names. */
data ReversedNames;
input name & $32.;
datalines;
Jones, Fred
Kavich, Kate
Turley, Ron
Dulix, Yolanda
;
/* Reverse last and first names with a DATA step. */
data names;
set ReversedNames;
name = prxchange('s/(¥w+), (¥w+)/$2 $1/', -1, name);
run;
proc print data=names;
run;
```

画面 2.46 *DATA ステップからの出力*

The SAS System

Obs	name
1	Fred Jones
2	Kate Kavich
3	Ron Turley
4	Yolanda Dulix

サンプル 2: PROC SQL を使用した名と姓の順序の入れ替え

次の例では、PROC SQL を使用して名と姓の順序を入れ替えます。

```
data ReversedNames;
input name & $32.;
datalines;
Jones, Fred
Kavich, Kate
Turley, Ron
Dulix, Yolanda
;
proc sql;
create table names as
select prxchange('s/(¥w+), (¥w+)/$2 $1/', -1, name) as name
from ReversedNames;
quit;
proc print data=names;
run;
```

画面 2.47 PROC SQL からの出力

The SAS System

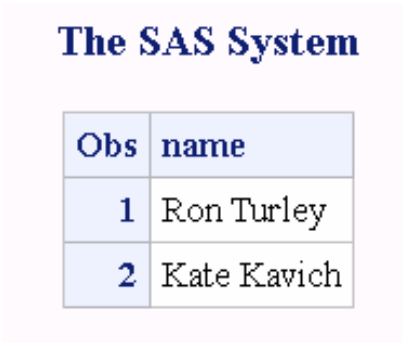
Obs	name
1	Fred Jones
2	Kate Kavich
3	Ron Turley
4	Yolanda Dulix

サンプル 3: 同一の名前持つ行の一致

次の例では、2 つのデータセットの名前を比較し、両方のデータセットに共通する名前を書き出します。

```
data names;
input name & $32.;
datalines;
Ron Turley
Judy Donnelly
Kate Kavich
Tully Sanchez
;
data ReversedNames;
input name & $32.;
datalines;
Jones, Fred
Kavich, Kate
Turley, Ron
Dulix, Yolanda
;
proc sql;
create table NewNames as
select a.name from names as a, ReversedNames as b
where a.name = prxchange('s/(¥w+), (¥w+)/$2 $1/', -1, b.name);
quit;
proc print data=NewNames;
run;
```

画面 2.48 同一の名前を持つ一致行からの出力



The SAS System

Obs	name
1	Ron Turley
2	Kate Kavich

サンプル 4: 小文字のテキストから大文字への変更

次の例では、\U、\L および\E のメタ文字を使用してテキストの文字列の大文字小文字を変更します。大文字小文字の変更はネストできません。この例では、\E メタ文字はすべての大文字小文字変更を終了するてめ、“bear”は大文字に変更されません。

```
data _null_;
length txt $32;
txt = prxchange ('s/(big)(black)(bear)/¥U$1¥L$2¥E$3/', 1, 'bigblackbear');
put txt=;
run;
```

SAS は次の出力を口グに返します。

```
txt=BIGblackbear
```

サンプル 5: 一致するパターンの固定値への変更

この例では、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。ここでは、DATA ステップを使用して電話番号を検索して情報メッセージに置換します。

```
 /* Create data set that contains confidential information. */
data a;
input text $80.;
datalines;
The phone number for Ed is (801)443-9876 but not until tonight.
He can be reached at (910)998-8762 tomorrow for testing purposes.
;
run;
/* Locate confidential phone numbers and replace them with message */
/* indicating that they have been removed. */
data b;
set a;
text = prxchange('s/¥([2-9]¥d¥d¥) ?[2-9]¥d¥d-¥d¥d¥d¥d/*PHONE NUMBER
REMOVED*/', -1, text);
put text=;
run;
proc print data = b;
run;
```

画面 2.49　一致するパターンから固定値への変更の出力

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” (759 ページ)
- “PRXPAREN 関数” (763 ページ)
- “PRXPARSE 関数” (765 ページ)
- “PRXPOSN 関数” (767 ページ)

CALL ルーチン:

- “CALL PRXCHANGE ルーチン” (194 ページ)
- “CALL PRXDEBUG ルーチン” (196 ページ)
- “CALL PRXFREE ルーチン” (198 ページ)
- “CALL PRXNEXT ルーチン” (199 ページ)
- “CALL PRXPOSN ルーチン” (201 ページ)
- “CALL PRXSUBSTR ルーチン” (204 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

構文

PRXMATCH (*regular-expression-id* | *perl-regular-expression*, *source*)

必須引数

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。

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。

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。

ソース
検索する文字定数、変数または式を指定します。

詳細

基本

regular-expression-id を使用する場合、PRXMATCH 関数は PRXPARSE によって返された *regular-expression-id* 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。

perl-regular-expression を使用する場合、PRXMATCH は *perl-regular-expression* で *source* を検索し、PRXPARSE を呼び出す必要はありません。

WHERE 句と PROC SQL で、PRXMATCH に Perl 正規表現を使用できます。パターン照合の詳細については、“Perl 正規表現(PRX)を使用したパターン照合 ” (42 ページ)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 オプションを使用する場合、Perl 正規表現は 1 回のみコンパイルされ、PRXMATCH を使用するたびにそのコンパイル済みの正規表現が再利用されます。*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。

注: コンパイル 1 回のみの動作は、DATA ステップの WHERE 句または PROC SQL で PRXMATCH を使用する場合に発生します。他のすべての使用では、PRXMATCH を呼び出すたびに *perl-regular-expression* が再コンパイルされます。

比較

Perl 正規表現(PRX)の関数と CALL ルーチンは、連携してパターンと一致する文字列を操作します。これらの関数および CALL ルーチンのリストと概要説明については、“カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の文字列マッチングカテゴリを参照してください。

サンプル

サンプル 1: PRXPARSE を使用した部分文字列の位置の検索

次の例では、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。

```
/* For 9.0: the following example makes a call to PRXPARSE. */
/* For 9.1, no call is required. */

    data _null_;
    /* Use PRXPARSE to compile the Perl regular expression. */
    patternID = prxparse('/world/');
    /* Use PRXMATCH to find the position of the pattern match. */
    position=prxmatch(patternID, 'Hello world!');
    put position=;
    run;
```

SAS は次の行を口グに書き込みます。

```
position=7
```

サンプル 2: Perl 正規表現を使用した部分文字列の位置の検索

次の例では、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。

```
data _null_;
/* Use PRXMATCH to find the position of the pattern match. */
position=prxmatch('/world/', 'Hello world!');
put position=;
run;
```

SAS は次の行を ログに書き込みます。

```
position=7
```

サンプル 3: 文字列内の部分文字列の位置の検索: 複雑な例

次の例では、複数の Perl 正規表現関数と CALL ルーチンを使用して、文字列内の部分文字列の位置を検索します。

```
data _null_;
if _N_ = 1 then
do;
retain PerlExpression;
pattern = "/(¥d+):(¥d¥d)(?:¥.(¥d+))?/";
PerlExpression = prxparse(pattern);
end;

array match[3] $ 8;
input minsec $80.;
position = prxmatch(PerlExpression, minsec);
if position ^= 0 then
do;
do i = 1 to prxparen(PerlExpression);
call prxposn(PerlExpression, i, start, length);
if start ^= 0 then
match[i] = substr(minsec, start, length);
end;
put match[1] "minutes, " match[2] "seconds" @;
if ^missing(match[3]) then
put ", " match[3] "milliseconds";
end;
datalines;
14:56.456
45:32
;
run;
```

次の行が SAS ログに書き込まれます。

```
14 minutes, 56 seconds, 456 milliseconds 45 minutes, 32 seconds
```

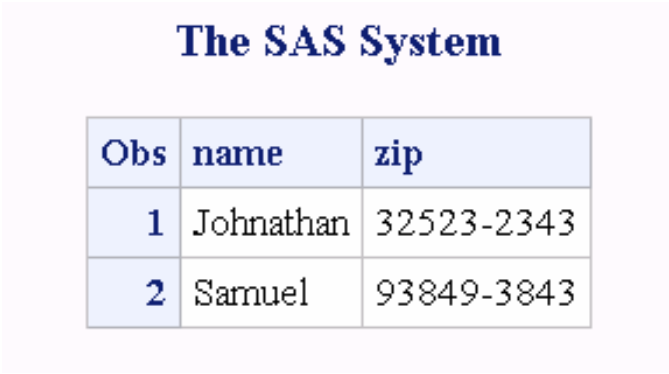
サンプル 4: DATA ステップを使用した郵便番号の抽出

次の例では、DATA ステップを使用して 9 桁の郵便番号のデータセットから各オブザベーションを検索し、それらのオブザベーションをデータセット ZipPlus4 に書き込みます。

```
data ZipCodes;
input name: $16. zip:$10.;
datalines;
```

```
Johnathan 32523-2343
Seth 85030
Kim 39204
Samuel 93849-3843
;
/* Extract ZIP+4 ZIP codes with the DATA step. */
data ZipPlus4;
set ZipCodes;
where prxmatch('/¥d{5}-¥d{4}/', zip);
run;
proc print data=ZipPlus4;
run;
```

画面 2.50 *DATA ステップからの郵便番号の出力*



The SAS System

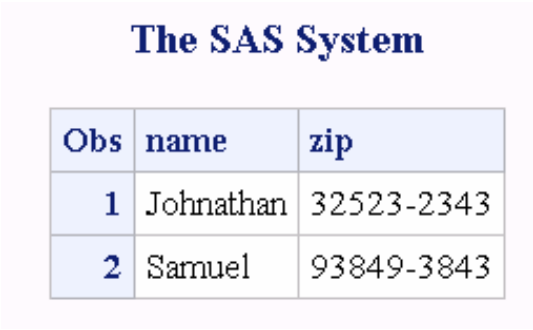
Obs	name	zip
1	Johnathan	32523-2343
2	Samuel	93849-3843

サンプル 5: PROC SQL を使用した郵便番号の抽出

次の例では、9 桁の郵便番号のデータセットから各オブザベーションを検索し、それらのオブザベーションをデータセット ZipPlus4 に書き込みます。

```
data ZipCodes;
input name: $16. zip:$10.;
datalines;
Johnathan 32523-2343
Seth 85030
Kim 39204
Samuel 93849-3843
;
/* Extract ZIP+4 ZIP codes with PROC SQL. */
proc sql;
create table ZipPlus4 as
select * from ZipCodes
where prxmatch('/¥d{5}-¥d{4}/', zip);
run;
proc print data=ZipPlus4;
run;
```

画面 2.51 PROC SQL からの郵便番号の出力



The SAS System

Obs	name	zip
1	Johnathan	32523-2343
2	Samuel	93849-3843

関連項目:

関数:

- “PRXCHANGE 関数” (754 ページ)
- “PRXPAREN 関数” (763 ページ)
- “PRXPARSE 関数” (765 ページ)
- “PRXPOSN 関数” (767 ページ)

CALL ルーチン:

- “CALL PRXCHANGE ルーチン” (194 ページ)
- “CALL PRXDEBUG ルーチン” (196 ページ)
- “CALL PRXFREE ルーチン” (198 ページ)
- “CALL PRXNEXT ルーチン” (199 ページ)
- “CALL PRXPOSN ルーチン” (201 ページ)
- “CALL PRXSUBSTR ルーチン” (204 ページ)
- “CALL PRXCHANGE ルーチン” (194 ページ)

PRXPAREN 関数

パターン内に一致が存在する場合の最後のかっこの一致を返します。

カテゴリ: 文字列マッチング

制限事項: PRXPARSE 関数とともに使用します。

構文

PRXPAREN (*regular-expression-id*)

必須引数

regular-expression-id
PRXPARSE 関数によって返される ID 番号の値が含まれる数値変数を指定します。

詳細

PRXPAREN 関数は、CALL PRXPOSN ルーチンに渡すことができる最大キャプチャバッファ数を確認する場合、または一致する部分パターンを特定する場合に役立ちます。

パターン照合の詳細については、“Perl 正規表現(PRX)を使用したパターン照合 ” (42 ページ)を参照してください。

比較

Perl 正規表現(PRX)の関数と CALL ルーチンは、連携してパターンと一致する文字列を操作します。これらの関数および CALL ルーチンのリストと概要説明については、“カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の文字列マッチングカテゴリを参照してください。

サンプル

次の例では、Perl 正規表現を使用して結果を SAS ログに書き込みます。

```
data _null_;
ExpressionID = prxparse('/(magazine)|(book)|(newspaper)/');
position = prxmatch(ExpressionID, 'find book here');
if position then paren = prxparen(ExpressionID);
put 'Matched paren ' paren;
position = prxmatch(ExpressionID, 'find magazine here');
if position then paren = prxparen(ExpressionID);
put 'Matched paren ' paren;
position = prxmatch(ExpressionID, 'find newspaper here');
if position then paren = prxparen(ExpressionID);
put 'Matched paren ' paren;
run;
```

次の行が SAS ログに書き込まれます。

```
 Matched paren 2
Matched paren 1
Matched paren 3
```

関連項目:

関数:

- “PRXCHANGE 関数” (754 ページ)
- “PRXMATCH 関数” (759 ページ)
- “PRXPARSE 関数” (765 ページ)
- “PRXPOSN 関数” (767 ページ)

CALL ルーチン:

- “CALL PRXCHANGE ルーチン” (194 ページ)
- “CALL PRXDEBUG ルーチン” (196 ページ)
- “CALL PRXFREE ルーチン” (198 ページ)
- “CALL PRXNEXT ルーチン” (199 ページ)

- “CALL PRXPOSN ルーチン” (201 ページ)
- “CALL PRXSUBSTR ルーチン” (204 ページ)
- “CALL PRXCHANGE ルーチン” (194 ページ)

PRXPARSE 関数

文字値のパターン照合に使用できる 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: 他の Perl 正規表現とともに使用します。

構文

regular-expression-id=**PRXPARSE** (*perl-regular-expression*)

必須引数

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。

perl-regular-expression
Perl 正規表現の文字値を指定します。

詳細

基本

PRXPARSE 関数は、他の Perl 関数と CALL ルーチンでパターンを照合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” (973 ページ)を参照してください。

パターン照合の詳細については、“Perl 正規表現(PRX)を使用したパターン照合 ” (42 ページ)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 オプションを使用する場合、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。

注: Perl 正規表現が定数の場合、または正規表現で/o オプションを使用する場合、PRXFREE を呼び出すとメモリ割り当てが解放され、次回 PRXPARSE によって呼び出されたときに正規表現を再コンパイルする必要があります。コンパイル 1 回のみの動作は、DATA ステップで PRXPARSE を使用する場合に発生します。他のすべての使用では、PRXPARSE を呼び出すたびに *perl-regular-expression* が再コンパイルされます。

比較

Perl 正規表現(PRX)の関数と CALL ルーチンは、連携してパターンと一致する文字列を操作します。これらの関数および CALL ルーチンのリストと概要説明については、“カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の文字列マッチングカテゴリを参照してください。

サンプル

次の例では、メタ文字と正規表現を使用して Perl 正規表現を構成します。この例では、アドレスを解析してフォーマットされた結果を SAS ログに書き込みます。

```
data _null_;
if _N_ = 1 then
do;
retain patternID;
/* The i option specifies a case insensitive search. */
pattern = "/ave|avenue|dr|drive|rd|road/i";
patternID = prxparse(pattern);
end;
input street $80.;
call prxsubstr(patternID, street, position, length);
if position ^= 0 then
do;
match = substr(street, position, length);
put match:$QUOTE. "found in " street:$QUOTE.;
end;
datalines;
153 First Street
6789 64th Ave
4 Moritz Road
7493 Wilkes Place
;
```

次の行が SAS ログに書き込まれます。

```
"Ave" found in "6789 64th Ave"
"Road" found in "4 Moritz Road"
```

関連項目:

関数:

- “PRXCHANGE 関数” (754 ページ)
- “PRXPAREN 関数” (763 ページ)
- “PRXMATCH 関数” (759 ページ)
- “PRXPOSN 関数” (767 ページ)

CALL ルーチン:

- “CALL PRXCHANGE ルーチン” (194 ページ)
- “CALL PRXDEBUG ルーチン” (196 ページ)
- “CALL PRXFREE ルーチン” (198 ページ)
- “CALL PRXNEXT ルーチン” (199 ページ)

- “CALL PRXPOSN ルーチン” (201 ページ)
- “CALL PRXSUBSTR ルーチン” (204 ページ)
- “CALL PRXCHANGE ルーチン” (194 ページ)

PRXPOSN 関数

キャプチャバッファ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: PRXPARSE 関数とともに使用します。

構文

PRXPOSN(*regular-expression-id*, *capture-buffer*, *source*)

必須引数

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。

capture-buffer
値を取得するキャプチャバッファ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 ~ 正規表現の開始かっこ数の間の場合、PRXPOSN はそのキャプチャバッファの値を返します。
- *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。

ソース
キャプチャバッファを抽出するテキストを指定します。

詳細

PRXPOSN 関数は、PRXMATCH、PRXSUBSTR、PRXCHANGE、PRXNEXT のいずれかの結果を使用してキャプチャバッファを返します。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。

キャプチャバッファは一致の一部としてかっこで囲まれ、正規表現で指定されます。この関数では CALL PRXPOSN のように SUBSTR を呼び出す必要がなく、キャプチャバッファのテキストが直接返されるため、キャプチャバッファの使用が簡素化されます。

パターン照合の詳細については、“Perl 正規表現(PRX)を使用したパターン照合 ” (42 ページ)を参照してください。

比較

PRXPOSN 関数は CALL PRXPOSN ルーチンと似ていますが、PRXPOSN 関数はキャプチャバッファの位置と長さではなくキャプチャバッファ自体を返す点が異なります。

Perl 正規表現(PRX)の関数と CALL ルーチンは、連携してパターンと一致する文字列を操作します。これらの関数および CALL ルーチンのリストと概要説明については、“カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の文字列マッチングカテゴリを参照してください。

サンプル

サンプル 1: 姓と名の抽出

次の例では、PRXPOSN を使用してデータセットから姓と名を抽出します。

```
data ReversedNames;
input name & $32.;
datalines;
Jones, Fred
Kavich, Kate
Turley, Ron
Dulix, Yolanda
;
data FirstLastNames;
length first last $ 16;
keep first last;
retain re;
if _N_ = 1 then
re = prxparse('/(¥w+), (¥w+)/');
set ReversedNames;
if prxmatch(re, name) then
do;
last = prxposn(re, 1, name);
first = prxposn(re, 2, name);
end;
run;
proc print data = FirstLastNames;
run;
```

画面 2.52 *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

サンプル 2: 一部の名前が無効な場合の名前の抽出

次の例では、名前のリストを含むデータセットを作成します。名のみまたは姓のみを含むオブザベーションは無効です。PRXPOSN はデータセットから有効な名前を抽出し、その名前をデータセット NEW に書き込みます。

```
data old;
input name $60.;
datalines;
Judith S Reaveley
Ralph F. Morgan
Jess Ennis
Carol Echols
Kelly Hansen Huff
Judith
Nick
Jones
;
data new;
length first middle last $ 40;
keep first middle last;
re = prxparse('/(¥S+)¥s+([^¥s]+¥s+)?(¥S+)/o');
set old;
if prxmatch(re, name) then
do;
first = prxposn(re, 1, name);
middle = prxposn(re, 2, name);
last = prxposn(re, 3, name);
output;
end;
run;
proc print data = new;
run;
```

画面 2.53 *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” (754 ページ)
- “PRXMATCH 関数” (759 ページ)
- “PRXPAREN 関数” (763 ページ)
- “PRXPARSE 関数” (765 ページ)

CALL ルーチン:

- “CALL PRXCHANGE ルーチン” (194 ページ)
- “CALL PRXDEBUG ルーチン” (196 ページ)
- “CALL PRXFREE ルーチン” (198 ページ)
- “CALL PRXNEXT ルーチン” (199 ページ)
- “CALL PRXPOSN ルーチン” (201 ページ)
- “CALL PRXSUBSTR ルーチン” (204 ページ)
- “CALL PRXCHANGE ルーチン” (194 ページ)

PTRLONGADD 関数

ポインタアドレスを 32 ビットおよび 64 ビットのプラットフォームの文字変数として返します。

カテゴリ: 特殊関数

構文

PTRLONGADD(*pointer*<,*amount*>)

必須引数

pointer
ポインタアドレスを指定する文字定数、変数または式です。

amount
アドレスに追加する量を指定する数値定数、変数または式です。

ヒント: *amount* には負の数を指定できます。

詳細

PTRLONGADD 関数は、ポインタ算術を実行してポインタアドレスを文字列として返します。

サンプル

次の例では、変数 Z のポインタアドレスを返します。

```
data _null_;
x='ABCDE';
y=ptrlongadd(addrlong(x),2);
z=peekclong(y,1);
put z=;
run;
```

SAS ログからの出力: **z=C**

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: 特殊関数

構文

PUT(*source*, *format.*)

必須引数

ソース
値の出力形式を変更する定数、変数または式を識別します。*source* 引数は文字または数値にできます。

format.
source で指定する値に適用する SAS 出力形式を含みます。この引数は、ピリオドを使用した出力形式の名前にする必要があります。必要に応じて幅と小数を指定します。文字定数、文字変数または文字式ではありません。デフォルトでは、source が数値の場合は結果の文字列が右揃えになり、source が文字の場合は結果が左揃えになります。デフォルトの配置をオーバーライドするには、配置の指定を出力形式に追加できます。

-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。

制限事項: *format*。source と同じ種類(文字か数値)にする必要があります。つまり、source が文字の場合は出力形式名の最初の文字をドル記号にする必要がありますが、source が数値の場合は出力形式名の最初の文字をドル記号にはできません。

詳細

まだ長さが割り当てられていない変数に PUT 関数から値が返される場合、デフォルトでその変数の長さは出力形式の幅によって決定されます。

数値を文字値に変換するには、PUT 関数を使用します。PUT 関数は、PUT ステートメントでどの出力形式を使用するのか、またはデータセット内の変数にどの出力形式を割り当てるのかには影響しません。PUT 関数を使用して、データセット内の変数の種類を数値から文字に直接変換することはできません。ただし、PUT 関数の結果として新しい文字変数を作成することができます。次に、必要に応じて、DROP ステートメントを使用して元の数値変数を削除した後、RENAME ステートメントを使用して新しい変数の名前を元の変数名に戻します。

比較

PUT ステートメントと PUT 関数は似ています。PUT 関数は、指定した出力形式を使用して値を返します。変数に値を格納するには、割り当てステートメントを使用する必要があります。PUT ステートメントは、外部の出力先(SAS ログか指定する出力先のいずれか)に値を書き込みます。

サンプル

サンプル 1: 数値から文字値への変換
この例では、第 1 ステートメントで cc(数値変数)の値を 4 文字の 16 進表記に変換し、第 2 ステートメントで PUT 関数が返す値と同じ値を書き込みます。

```
cchex=put(cc,hex4.);
put cc hex4.;
```

cc の元の変数名を文字変数として維持する必要がある場合は、PUT 関数の後に DROP ステートメントと RENAME ステートメントを使用します。

```
cchex=put(cc,hex4.);
drop cc;
rename cchex=cc;
```

cc 変数の数値から新しい cchex 変数が文字変数として作成されます。DROP ステートメントは、数値変数 cc がデータセットに書き込まれないようにし、RENAME ステートメントは新しい文字変数 cchex を cc という名前に戻します。

サンプル 2: PUT 関数と INPUT 関数を使用する

この例では、PUT 関数が数値を文字列として返します。値 122591 が CHARDATE 変数に割り当てられます。INPUT 関数は、SAS 日付入力形式を使用して文字列の値を SAS 日付値として返します。値 11681 が SASDATE 変数に格納されます。

```
numdate=122591;
chardate=put(numdate,z6.);
sasdate=input(chardate,mmddyy6.);
```

関連項目:

関数:

- “INPUT 関数” (538 ページ)
- “INPUTC 関数” (540 ページ)
- “INPUTN 関数” (541 ページ)
- “PUTC 関数” (772 ページ)
- “PUTN 関数” (774 ページ)

ステートメント:

- “PUT ステートメント” (*SAS ステートメント: リファレンス*)

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: 特殊関数

構文

PUTC(*source*, *format*.<,*w*>)

必須引数

ソース
出力形式を適用する文字定数、変数または式を指定します。

format.
source に適用する文字の出力形式の値が含まれる文字定数、変数または式です。

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、デフォルトで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。

サンプル

この例の 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 つの出力形式には、質問の種類に応じて、異なるワードとして"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、現在の質問の種類に適切な出力形式の値が含まれる変数 RESPFMT を作成します。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;
data answers;
length word $ 8;
input type response $;
respfmt = put(type, typefmt.);
word = putc(response, respfmt);
datalines;
1 positive
```

```
1 negative
1 neutral
2 positive
2 negative
2 neutral
3 positive
3 negative
3 neutral
;

proc print data=answers;
run;
SAS log:
Obs word type response respfmt 1 agree 1 positive $groupx
2 disagree 1 negative $groupx
3 notsure 1 neutral $groupx
4 accept 2 positive $groupy
5 reject 2 negative $groupy
6 possible 2 neutral $groupy
7 pass 3 positive $groupz
8 fail 3 negative $groupz
9 retest 3 neutral $groupz
```

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。

関連項目:

関数:

- “INPUT 関数” (538 ページ)
- “INPUTC 関数” (540 ページ)
- “INPUTN 関数” (541 ページ)
- “PUT 関数” (770 ページ)
- “PUTN 関数” (774 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: 特殊関数

構文

PUTN(*source*, *format*.<,*w*<,*d*>>)

必須引数

ソース
出力形式を適用する数値定数、変数または式を指定します。

format.
source に適用する数値の出力形式の値が含まれる文字定数、変数または式です。

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 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指定できるため、PUTN よりも高速です。

サンプル

この例の 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;
data dates;
input number key;
datefmt=put(key,writfmt.);
date=putn(number,datefmt);
datalines;
15756 1
14552 2
;
```

関連項目:

関数:

- “INPUT 関数” (538 ページ)

- “INPUTC 関数” (540 ページ)
- “INPUTN 関数” (541 ページ)
- “PUT 関数” (770 ページ)
- “PUTC 関数” (772 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: 財務関数

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: $A > 0$

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: $0 \le c < 1$

n

1 年当たりのクーポン数を指定します。

範囲: $n > 0$ 整数です。

K

クーポンの残数を指定します。

範囲: $K > 0$ 整数です。

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: $0 < k_0 \le \frac{1}{n}$

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: $y > 0$

詳細

PVP 関数は関係に基づきます。

$$P = \sum_{k=1}^{K} \frac{c(k)}{\left(1 + \frac{y}{n}\right)^{t_k}}$$

前述の式には次の関係が適用されます。

- $t_k = nk_0 + k - 1$
- $c(k) = \frac{c}{n}A \quad for\ k = 1, \ \ldots, K - 1$
- $c(K) = \left(1 + \frac{c}{n}\right)A$

サンプル

```
data _null_;
p=pvp(1000,.01,4,14,.33/2,.10);
put p;
run;
```

返される値は 743.168 です。

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

構文

QTR(*date*)

必須引数

date
SAS 日付値を表す数値定数、変数または式を指定します。

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。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果

x='20jan94'd;
y=qtr(x);
put y=;
	y=1

関連項目:

関数:

- “YYQ 関数” (959 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

参照項目: “CDF 関数” (273 ページ)

構文

QUANTILE(*dist*,*probability*,*parm-1, ... ,parm-k*)

必須引数

dist
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
ベルヌーイ	BERNOULLI
ベータ	BETA
二項	BINOMIAL
コーシー	CAUCHY
χ2 乗	CHISQUARE
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
ポアソン	POISSON
T	T
Tweedie	TWEEDIE
一様	UNIFORM
逆ガウス(Wald)	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。

確率
ランダム変数の値を指定する、数値定数、変数または式です。

parm-1, … ,parm-k
特定の分布に適した任意の *shape*、*location* または *scale* パラメータです。

詳細

QUANTIEL 関数は、さまざまな連続分布および離散分布の確率を計算します。詳細については、CDF 関数の"詳細" (274 ページ)のセクションを参照してください。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
y=quantile('BERN',.75,.25);	0
y=quantile('BETA',0.1,3,4);	0.2009088789
y=quantile('BINOM',.4,.5,10);	5
y=quantile('CAUCHY',.85);	1.9626105055
y=quantile('CHISQ',.6,11);	11.529833841
y=quantile('EXPO',.6);	0.9162907319
y=quantile('F',.8,2,3);	2.8860266073

SAS ステートメント	結果
y=quantile('GAMMA',.4,3);	2.285076904
y=quantile('GENPOISSON',.9,1,.7);	9
y=quantile('HYPER',.5,200,50,10);	2
y=quantile('LAPLACE',.8);	0.9162907319
y=quantile('LOGISTIC',.7);	0.8472978604
y=quantile('LOGNORMAL',.5);	1
y=quantile('NEGB',.5,.5,2);	1
y=quantile('NORMAL',.975);	1.9599639845
y=quantile('PARETO',.01,1);	1.0101010101
y=quantile('POISSON',.9,1);	2
y=quantile('T',.8,5);	0.9195437802
y=quantile('TWEEDIE',.8,5);	1.261087383
y=quantile('UNIFORM',0.25);	0.25
y=quantile('WALD',.6,2);	0.9526209927
y=quantile('WEIBULL',.6,2);	0.9572307621

関連項目:

関数:

- “CDF 関数” (273 ページ)
- “LOGCDF 関数” (624 ページ)
- “LOGPDF 関数” (626 ページ)
- “LOGSDF 関数” (628 ページ)
- “PDF 関数” (703 ページ)
- “SDF 関数” (832 ページ)
- “SQUANTILE 関数” (856 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: 文字関数

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

QUOTE(*argument*)

必須引数

引数
文字定数、変数または式を指定します。

詳細

返される変数の長さ

DATA ステップでは、まだ長さが割り当てられていない変数に QUOTE 関数から値が返される場合、その変数には 200 バイトの長さが設定されます。

基本

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口グに書き込みます。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x='A"B'; y=quote(x); put y;	"A""B"
x='A''B'; y=quote(x); put y;	"A'B"
x='Paul''s'; y=quote(x); put y;	"Paul's"
x='Catering Service Center '; y=quote(x); put y;	"Catering Service Center "
x='Paul''s Catering Service '; y=quote(trim(x)); put y;	"Paul's Catering Service"

RANBIN 関数

二項分布からランダム変量を返します。

カテゴリ: 乱数

ヒント: 実行時にシード値を変更するには、RANBIN 関数のかわりに CALL RANBIN ルーチンを使用する必要があります。

構文

RANBIN(*seed*,*n*,*p*)

必須引数

シード

整数を指定する数値定数、変数または式です。*seed* ≤ 0 の場合、シードストリームの初期化に時刻が使用されます。

範囲: $seed < 2^{31} - 1$

参照項目: シード値の詳細については、“シード値” (11 ページ) を参照してください。

n

独立 Bernoulli 試行のパラメータ数を指定する整数値を使用する数値定数、変数または式です。

範囲: $n > 0$

p

成功確率を指定する数値定数、変数または式です。

範囲: $0<p<1$

詳細

RANBIN 関数は、平均 *np* と分散 *np*(1−*p*)の二項分布から生成される変量を返します。*n*≤50、*np*≤5、または *n(1–p)*≤5 の場合、RANUNI 一様変量に適用される逆変換法が使用されます。*n*>50、*np*>5、*n(1–p)*>5 の場合、二項分布に対する正規近似が使用されます。この場合、RANUNI 一様変量の Box-Muller 変換が使用されます。

データのシードとストリームおよび乱数関数の使用例については、“乱数関数で単一のシードから複数の変数を生成する” (22 ページ)を参照してください。

比較

RANBIN 関数にかわる CALL RANBIN ルーチンは、シードおよび乱数ストリームをより高度に制御します。

関連項目:

関数:

- “RAND 関数” (784 ページ)

CALL ルーチン:

- “CALL RANBIN ルーチン” (206 ページ)

RANCAU 関数

Cauchy 分布からランダム変量を返します。

カテゴリ: 乱数

ヒント: 実行時にシード値を変更するには、RANCAU 関数のかわりに CALL RANCAU ルーチンを使用する必要があります。

構文

RANCAU(*seed*)

必須引数

シード

整数を指定する数値定数、変数または式です。*seed* ≤ 0 の場合、シードストリームの初期化に時刻が使用されます。

範囲: $seed < 2^{31} - 1$

参照項目: シード値の詳細については、“シード値” (11 ページ) を参照してください。

詳細

RANCAU 関数は、位置パラメータ 0 と尺度パラメータ 1 の Cauchy 分布から生成される変量を返します。RANUNI 一様変量に適用される受容-棄却手法が使用されます。*u* と *v* が独立一様(−1/2、1/2)変数で、$u^2 + v^2 \le 1/4$ の場合、*u*/*v* は Cauchy 変量です。位置パラメータ ALPHA と尺度パラメータ BETA で Cauchy 変量 X を次のように生成できます。

```
x=alpha+beta*rancau(seed);
```

データのシードとストリームおよび乱数関数の使用例については、“乱数関数で単一のシードから複数の変数を生成する” (22 ページ)を参照してください。

比較

RANCAU 関数にかわる CALL RANCAU ルーチンは、シードおよび乱数ストリームをより高度に制御します。

関連項目:

関数:

- “RAND 関数” (784 ページ)

CALL ルーチン:

- “CALL RANCAU ルーチン” (208 ページ)

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

構文

RAND (*dist*, *parm-1*,…,*parm-k*)

必須引数

dist

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
ベルヌーイ	BERNOULLI
ベータ	BETA
二項	BINOMIAL
コーシー	CAUCHY
χ2 乗	CHISQUARE
Erlang	ERLANG
指数	EXPONENTIAL
F	F
ガンマ	GAMMA
幾何	GEOMETRIC
超幾何	HYPERGEOMETRIC
対数正規	LOGNORMAL
負の二項	NEGBINOMIAL
正規	NORMAL|GAUSSIAN
ポアソン	POISSON
T	T
テーブル	TABLE

分布	引数
三角	TRIANGLE
一様	UNIFORM
Weibull	WEIBULL

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。

parm-1, … ,parm-k
特定の分布に適した *shape*、*location* または *scale* パラメータです。

参照項目: "詳細" (785 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、松本と西村(1998)によって開発された Mersenne-Twister 乱数ジェネレータ(RNG)を応用しています。乱数ジェネレータには、長い周期(2^{19937}-1)と高品質な統計的特性を有しています。この周期は Mersenne 素数で、RNG の名前はこれに由来します。アルゴリズムは、名前の後半部分を説明する、ひねり(twist)を入れた一般化フィードバックシフトレジスタ(TGFSR)です。TGFSR を使用した RNG では、高次元(32 ビット精度の 623 次元)の均等分布が生成されます。これは、623 次元疑似乱数の連続ベクトル間の相関性が非常に小さいことを意味します。

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。ジェネレータを停止し、停止時点から再開することはできません。

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には、CALL STREAMINIT ルーチンを使用して乱数生成のシード値を指定します。RAND 関数が起動される前に、DATA ステップごとに CALL STREAMINIT ルーチンを 1 回使用します。CALL STREAMINIT ルーチンの呼び出しを省略すると(または CALL STREAMINIT ルーチンで正以外のシード値を指定すると)、RAND はシステムクロックへの呼び出しを使用してシードを取得します。詳細については、CALL STREAMINIT"サンプル: 乱数の再現可能なストリームの作成" (251 ページ)を参照してください。

Mersenne-Twister RNG アルゴリズムの重複する値

Permutations RNG アルゴリズムには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に示すものではありません。RAND 関数は最大で 2^{32} 個の重複しない値を返します。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。

Bernoulli 分布

x = **RAND**('BERNOULLI',*p*)

引数

x

次の確率密度関数を使用した分布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

数値の成功確率です。

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分布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です。

範囲: $a > 0$

b

数値の形状パラメータです。

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分布からの整数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

数値の成功確率です。

範囲: $0 \le p \le 1$

n

独立ベルヌーイ試行数を数える整数のパラメータです。

範囲: $n = 1, 2, ...$

Cauchy 分布

x = **RAND**('CAUCHY')

引数

x

次の確率密度関数を使用した分布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分布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分布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

整数の数値の形状パラメータです。

範囲: $a = 1, 2, ...$

指数分布

x = **RAND**('EXPONENTIAL')

引数

x

次の確率密度関数を使用した分布のオブザベーションです。

$$f(x) = e^{-x}$$

範囲: $x > 0$

F 分布

x = **RAND**('F',*n*, *d*)

引数

x

次の確率密度関数を使用した分布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です。

範囲: $n > 0$

d

数値の分母の自由度パラメータです。

範囲: $d > 0$

ガンマ分布

x = **RAND**('GAMMA',*a*)

引数

x

次の確率密度関数を使用した分布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です。

範囲: $a > 0$

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整数の度数です。*X* は次の確率密度関数を使用した分布からの整数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: $x = 1, 2, \ldots$

p

数値の成功確率です。

範囲: $0 < p \leq 1$

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分布からの整数オブザベーションです。

$$f(x) = \frac{\binom{R}{x}\binom{N - R}{n - x}}{\binom{N}{n}}$$

範囲: $x = \max(0, (n - (N - R))), ..., \min(n, R)$

N

整数の母集団サイズパラメータです。

範囲: $N = 1, 2, ...$

R

対象カテゴリの整数の項目数です。

範囲: $R = 0, 1, ..., N$

n

整数のサンプルサイズパラメータです。

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。

対数正規分布

x = **RAND**('LOGNORMAL')

引数

x

次の確率密度関数を使用した分布のオブザベーションです。

$$f(x) = \frac{e^{-\ln^2(x)/2}}{x\sqrt{2\pi}}$$

範囲: $x > 0$

負数二項分布

x = **RAND**('NEGBINOMIAL',p,k)

引数

x

次の確率密度関数を使用した分布からの整数オブザベーションです。

$$f(x) = \begin{cases} \binom{x + k - 1}{k - 1}(1 - p)^x p^k & 0 < p < 1,\ x = 0, 1, ... \\ 1 & p = 1,\ x = 0 \end{cases}$$

範囲: $x = 0, 1, ...$

k

成功回数を示す整数パラメータです。ただし、整数ではない k 値も使用できます。

範囲: $k = 1, 2, ...$

p
数値の成功確率です。
範囲: $0 < p \leq 1$

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
平均パラメータです。
デフォルト: 0

λ
標準偏差パラメータです。
デフォルト: 1
範囲: $\lambda > 0$

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整数オブザベーションです。

$$f(x) = \frac{m^x e^{-m}}{x!}$$

範囲: $x = 0, 1, ...$

m
数値の平均パラメータです。
範囲: $m > 0$

T 分布
x = **RAND**('T',df)

引数

x
次の確率密度関数を使用した分布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整数のオブザベーションです。

次の場合、$\sum_{i=1}^{n} p_i < 1$ x はこの確率分布関数のオブザベーションです。

$f(i) = p_i, \quad i = 1, 2, \ldots, n$

and

$f(n+1) = 1 - \sum_{i=1}^{n} p_i$

一部のインデックスに対し、次の場合、$\sum_{i=1}^{n} p_i \geq 1$ x はこの確率分布関数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: 確率パラメータの最大数は動作環境によって異なりますが、パラメータ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分布のオブザベーションです。

$$f(x) = \begin{cases} \frac{2x}{h} & 0 \leq x \leq h \\ \frac{2(1-x)}{1-h} & h < x \leq 1 \end{cases}$$

この式では、$0 \leq h \leq 1$ になります。

範囲: $0 \leq x \leq 1$

注: 分布は簡単に移動および調整できます。

h
三角の頂点の水平位置です。
範囲: $0 \le h \le 1$

一様分布

x = **RAND**('UNIFORM')

引数

x
次の確率密度関数を使用した分布のオブザベーションです。

$$f(x) = 1$$

範囲: $0 < x < 1$

RAND 関数が使用する一様乱数ジェネレータは、Mersenne-Twister(松本と西村(1998))です。このジェネレータには次の周期と $2^{19937} - 1$ 最大で 32 ビットの精度で 623 次元の均等分布があります。このアルゴリズムは、RAND 関数で使用できるほかの分布のジェネレータの基礎となります。

Weibull 分布

x = **RAND**('WEIBULL',*a*,*b*)

引数

x
次の確率密度関数を使用した分布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: $x \ge 0$

a
数値の形状パラメータです。
範囲: $a > 0$

b
数値の尺度パラメータです。
範囲: $b > 0$

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=rand('BERN',.75);	0
x=rand('BETA',3,0.1);	.99920
x=rand('BINOM',0.75,10);	10
x=rand('CAUCHY');	−1.41525

SAS ステートメント	結果
x=rand('CHISQ',22);	25.8526
x=rand('ERLANG', 7);	7.67039
x=rand('EXPO');	1.48847
x=rand('F',12,322);	1.99647
x=rand('GAMMA',7.25);	6.59588
x=rand('GEOM',0.02);	43
x=rand('HYPER',10,3,5);	1
x=rand('LOGN');	0.66522
x=rand('NEGB',0.8,5);	33
x=rand('NORMAL');	1.03507
x=rand('POISSON',6.1);	6
x=rand('T',4);	2.44646
x=rand('TABLE',.2,.5);	2
x=rand('TRIANGLE',0.7);	.63811
x=rand('UNIFORM');	.96234
x=rand('WEIB',0.25,2.1);	6.55778

関連項目:

CALL ルーチン:

- “CALL STREAMINIT ルーチン” (250 ページ)

リファレンス

Fishman, G. S. 1996. *Monte Carlo: Concepts, Algorithms, and Applications*. New York, 米国: Springer-Verlag.

Fushimi, M., and S. Tezuka. “The k-Distribution of Generalized Feedback Shift Register Pseudorandom Numbers.” 1983. *Communications of the ACM* 26: 516-255.

Gentle, J. E. 1998. *Random Number Generation and Monte Carlo Methods*. New York, 米国: Springer-Verlag.

Lewis, T. G., and W. H. Payne. “Generalized Feedback Shift Register Pseudorandom Number Algorithm.” 1973. *Journal of the ACM* 20: 456-468.

Matsumoto, M., and Y. Kurita. “Twisted GFSR Generators.” 1992. *ACM Transactions on Modeling and Computer Simulation* 2: 179-255.

Matsumoto, M., and Y. Kurita. “Twisted GFSR Generators II.” 1994. *ACM Transactions on Modeling and Computer Simulation* 4: 254-255.

Matsumoto, M., and T. Nishimura. “Mersenne Twister: A 623–Dimensionally Equidistributed Uniform Pseudo-Random Number Generator.” 1998. *ACM Transactions on Modeling and Computer Simulation* 8: 3-255.

Ripley, B. D. 1987. *Stochastic Simulation*. New York, 米国: Wiley.

Robert, C. P., and G. Casella. 1999. *Monte Carlo Statistical Methods*. New York, 米国: Springer-Verlag.

Ross, S. M. 1997. *Simulation*. San Diego, 米国: Academic Press.

RANEXP 関数

指数分布からランダム変量を返します。

カテゴリ: 乱数

ヒント: 実行時にシード値を変更するには、RANEXP 関数のかわりに CALL RANEXP ルーチンを使用する必要があります。

構文

RANEXP(*seed*)

必須引数

シード

整数を指定する数値定数、変数または式です。*seed* ≤ 0 の場合、シードストリームの初期化に時刻が使用されます。

範囲: $seed < 2^{31} - 1$

参照項目: シード値の詳細については、“シード値” (11 ページ) を参照してください。

詳細

RANEXP 関数は、パラメータ 1 を使用した指数分布から生成される変量を返します。RANUNI 一様変量に適用される逆変換法が使用されます。

パラメータ LAMBDA を使用して指数変量 X を次のように生成できます。

```
x=ranexp(seed)/lambda;
```

位置パラメータ ALPHA と尺度パラメータ BETA を使用して極値変量 X を次のように生成できます。

```
x=alpha-beta*log(ranexp(seed));
```

パラメータ P を使用して幾何変量 X を次のように生成できます。

```
x=floor(-ranexp(seed)/log(1-p));
```

データのシードとストリームおよび乱数関数の使用例については、“乱数関数で単一のシードから複数の変数を生成する” (22 ページ)を参照してください。

比較

RANEXP 関数にかわる CALL RANEXP ルーチンは、シードおよび乱数ストリームをより高度に制御します。

関連項目:

関数:

- “RAND 関数” (784 ページ)

CALL ルーチン:

- “CALL RANEXP ルーチン” (213 ページ)

RANGAM 関数

ガンマ分布からランダム変量を返します。

カテゴリ: 乱数

ヒント: 実行時にシード値を変更するには、RANGAM 関数のかわりに CALL RANGAM ルーチンを使用する必要があります。

構文

RANGAM(*seed*,*a*)

必須引数

シード

整数を指定する数値定数、変数または式です。*seed* ≤ 0 の場合、シードストリームの初期化に時刻が使用されます。

範囲: $seed < 2^{31} - 1$

参照項目: シード値の詳細については、“シード値” (11 ページ) を参照してください。

a

形状パラメータを指定する数値定数、変数または式です。

範囲: $a > 0$

詳細

RANGAM 関数は、パラメータ *a* を使用したガンマ分布から生成される変量を返します。*a*>1 の場合、Cheng(1977)による受容-棄却法が使用されます (“リファレンス” (971 ページ)を参照)。 *a*≤1 の場合、Fishman による受容-棄却法(1978, Algorithm G2)が使用されます(“リファレンス” (971 ページ)を参照)。

形状パラメータ ALPHA と尺度 BETA を使用してガンマ変量 X を次のように生成できます。

```
x=beta*rangam(seed,alpha);
```

2*ALPHA が整数の場合、自由度 2*ALPHA を使用した χ2 乗変量 X を次のように生成できます。

```
x=2*rangam(seed,alpha);
```

N が正の整数の場合、Erlang 変量 X を次のように生成できます。

```
x=beta*rangam(seed,N);
```

平均値が BETA になる、N 個の独立指数変量の合計の分布になります。

最後に、パラメータ ALPHA と BETA を使用してベータ変量 X を次のように生成できます。

```
y1=rangam(seed,alpha);
y2=rangam(seed,beta);
x=y1/(y1+y2);
```

データのシードとストリームおよび乱数関数の使用例については、“乱数関数で単一のシードから複数の変数を生成する” (22 ページ)を参照してください。

比較

RANGAM 関数にかわる CALL RANGAM ルーチンは、シードおよび乱数ストリームを高度に制御します。

関連項目:

関数:

- “RAND 関数” (784 ページ)

CALL ルーチン:

- “CALL RANGAM ルーチン” (215 ページ)

RANGE 関数

非欠損値の範囲を返します。

カテゴリ: 記述統計

構文

RANGE(*argument-1*<,...*argument-n*>)

必須引数

引数
数値の定数、変数または式を指定します。少なくとも 1 つの非欠損引数が必要です。非欠損引数がない場合は、関数から欠損値が返されます。引数リストには OF で始まる変数のリストを含められます。

詳細

RANGE 関数は、最大と最小の非欠損引数の差異を返します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x0=range(.,.);	.
x1=range(-2,6,3);	8
x2=range(2,6,3,.);	4
x3=range(1,6,3,1);	5
x4=range(of x1-x3);	4

RANK 関数

ASCII 照合順序または EBCDIC 照合順序による文字の位置を返します。

カテゴリ: 文字関数

制限事項: I18N レベル 0 の関数は 1 バイト文字セット(SBCS)でのみ使用するように設計されています。

参照項目: “RANK Function: Windows” in *SAS Companion for Windows*
“RANK Function: UNIX” in *SAS Companion for UNIX Environments*

構文

RANK(*x*)

必須引数

x
文字定数、変数または式を指定します。

詳細

RANK 関数は、文字式内の最初の文字の位置を表す整数を返します。結果は動作環境によって異なります。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果	
	ASCII	EBCDIC
n=rank('A'); put n;	65	193

関連項目:

関数:

- “BYTE 関数” (147 ページ)
- “COLLATE 関数” (302 ページ)

RANNOR 関数

正規分布からランダム変量を返します。

カテゴリ: 乱数

ヒント: 実行時にシード値を変更するには、RANNOR 関数のかわりに CALL RANNOR ルーチンを使用する必要があります。

構文

RANNOR(*seed*)

必須引数

シード

整数を指定する数値定数、変数または式です。*seed* ≤ 0 の場合、シードストリームの初期化に時刻が使用されます。

範囲: $seed < 2^{31} - 1$

参照項目: シード値の詳細については、“シード値” (11 ページ) を参照してください。

詳細

RANNOR 関数は、平均 0 と分散 1 を使用した正規分布から生成される変量を返します。RANUNI 一様変量の Box-Muller 変換が使用されます。

平均 MU と分散 S2 を使用した正規変量 X は、次のコードで生成できます。

```
x=MU+sqrt(S2)*rannor(seed);
```

平均 *exp*(MU + S2/2)と分散 *exp*(2*MU + 2*S2) −*exp*(2*MU + S2)を使用した対数正規変量は、次のコードで生成できます。

```
x=exp(MU+sqrt(S2)*rannor(seed));
```

データのシードとストリームおよび乱数関数の使用例については、“乱数関数で単一のシードから複数の変数を生成する” (22 ページ)を参照してください。

比較

RANNOR 関数にかわる CALL RANNOR ルーチンは、シードおよび乱数ストリームをより高度に制御します。

関連項目:

関数:

- “RAND 関数” (784 ページ)

CALL ルーチン:

- “CALL RANNOR ルーチン” (218 ページ)

RANPOI 関数

ポアソン分布からランダム変量を返します。

カテゴリ: 乱数

ヒント: 実行時にシード値を変更するには、RANPOI 関数のかわりに CALL RANPOI ルーチンを使用する必要があります。

構文

RANPOI(*seed*,*m*)

必須引数

シード

整数を指定する数値定数、変数または式です。*seed* ≤ 0 の場合、シードストリームの初期化に時刻が使用されます。

範囲: $seed < 2^{31} - 1$

参照項目: シード値の詳細については、“シード値” (11 ページ)を参照してください。

m

分布の平均を指定する数値定数、変数または式です。

範囲: $m \geq 0$

詳細

RANPOI 関数は、平均 *m* を使用した Poisson 分布から生成される変量を返します。*m*<85 の場合、RANUNI 一様変量に適用される逆変換法が使用されます (Fishman 1976) (“リファレンス” (971 ページ)を参照)。*m*≥85 の場合、Poisson ランダム変数の正規近似が使用されます。迅速に実行するために、内部変数は初期呼び出し(新しい各 *m*)でのみ計算されます。

データのシードとストリームおよび乱数関数の使用例については、“乱数関数で単一のシードから複数の変数を生成する” (22 ページ)を参照してください。

比較

RANPOI 関数にかわる CALL RANPOI ルーチンは、シードおよび乱数ストリームをより高度に制御します。

関連項目:

関数:

- “RAND 関数” (784 ページ)

CALL ルーチン:

- “CALL RANPOI ルーチン” (224 ページ)

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。

カテゴリ: 乱数

ヒント: 実行時にシード値を変更するには、RANTBL 関数のかわりに CALL RANTBL ルーチンを使用する必要があります。

構文

RANTBL(*seed*,*p1*,... p_i... ,p_n)

必須引数

シード

整数を指定する数値定数、変数または式です。$seed \leq 0$ の場合、シードストリームの初期化に時刻が使用されます。

範囲: $seed < 2^{31} - 1$

参照項目: シード値の詳細については、"シード値" (11 ページ) を参照してください。

p_i

数値定数、変数または式です。

範囲: $0 < i \leq n$ に対し $0 \leq p_i \leq 1$

詳細

RANTBL 関数は $p_1 \sim p_n$ で定義される確率質量関数から生成される変量を返します。RANUNI 一様変量に適用される逆変換法が使用されます。RANTBL は次を返します。

1 *with probability* p_1

2 *with probability* p_2

.

.

.

n *with probability* p_n

$n+1$ *with probability* $1 - \sum_{i=1}^{n} p_i$ *if* $\sum_{i=1}^{n} p_i \leq 1$

インデックス $j < n$ の場合、$\sum_{i=1}^{j} p_i \geq 1$ RANTBL はインデックスは、インデックス $1 \sim j$ のみを返します。j の発生確率は $1 - \sum_{i=1}^{j-1} p_i$ と等しくなります。

n=3 および P1、P2、P3 を P1+P2+P3=1 で定義する 3 つの確率とし、M1、M2、M3 を 3 つの変数とします。これらのステートメントの変数 X は

```
array m{3} m1-m3;
x=m{rantbl(seed,of p1-p3)};
```

それぞれ P1、P2、P3 の発生確率を持つ値 M1、M2、M3 のいずれかに割り当てられます。

乱数 CALL ルーチンの効果的な使用方法と例については、“ストリームを開始、停止および再開する” (26 ページ)を参照してください。

比較

RANTBL 関数にかわる CALL RANTBL ルーチンは、シードおよび乱数ストリームをより高度に制御します。

関連項目:

関数:

- “RAND 関数” (784 ページ)

CALL ルーチン:

- “CALL RANTBL ルーチン” (226 ページ)

RANTRI 関数

三角分布からランダム変量を返します。

カテゴリ: 乱数

ヒント: 実行時にシード値を変更するには、RANTRI 関数のかわりに CALL RANTRI ルーチンを使用する必要があります。

構文

RANTRI(*seed*,*h*)

必須引数

シード

整数を指定する数値定数、変数または式です。*seed* ≤ 0 の場合、シードストリームの初期化に時刻が使用されます。

範囲: $seed < 2^{31} - 1$

参照項目: シード値の詳細については、“シード値” (11 ページ)を参照してください。

h

分布のモードを指定する数値定数、変数または式です。

範囲: $0<h<1$

詳細

CALL RANTRI 関数は分布のモーダル値である、パラメータ *h* を使用した間隔(0,1)の三角分布から生成される変量を返します。RANUNI 一様変量に適用される逆変換法が使用されます。

A≤C≤B の場合、モード C を使用する間隔(A,B)の三角分布 X は、次のように生成できます。

```
x=(b-a)*rantri(seed,(c-a)/(b-a))+a;
```

データのシードとストリームおよび乱数関数の使用例については、“乱数関数で単一のシードから複数の変数を生成する” (22 ページ)を参照してください。

比較

RANTRI 関数にかわる CALL RANTRI ルーチンは、シードおよび乱数ストリームをより高度に制御します。

関連項目:

関数:

- “RAND 関数” (784 ページ)

CALL ルーチン:

- “CALL RANTRI ルーチン” (229 ページ)

RANUNI 関数

一様分布からランダム変量を返します。

カテゴリ: 乱数

ヒント: 実行時にシード値を変更するには、RANUNI 関数のかわりに CALL RANUNI ルーチンを使用する必要があります。

構文

RANUNI(*seed*)

必須引数

シード

整数を指定する数値定数、変数または式です。$seed \leq 0$ の場合、シードストリームの初期化に時刻が使用されます。

範囲: $seed < 2^{31} - 1$

参照項目: シード値の詳細については、“シード値” (11 ページ) を参照してください。

詳細

RANUNI 関数は素数係数乗法ジェネレータで係数 2^{31}–および乗数 397204094 を使用して、間隔(0,1)の一様分布から生成される変量を返します(Fishman and Moore 1982) (“リファレンス” (971 ページ)を参照)。

乗数を使用して間隔の長さを変更し、付加定数を使用して間隔を移動できます。たとえば、

```
random_variate=a*ranuni(seed)+b;
```

は、間隔(b,a+b)の一様分布から生成される数値を返します。

比較

RANUNI 関数にかわる CALL RANUNI ルーチンは、シードおよび乱数ストリームをより高度に制御します。

関連項目:

関数:

- “RAND 関数” (784 ページ)

CALL ルーチン:

- “CALL RANUNI ルーチン” (231 ページ)

RENAME 関数

SAS ライブラリのメンバ、SAS カタログ内のエントリ、外部ファイル、ディレクトリのいずれかの名前を変更します。

カテゴリ: 外部ファイル
SAS ファイル I/O 関数

構文

RENAME(*old-name*, *new-name* <, *type*<, *description*<, *password*<, *generation*>>>>)

必須引数

old-name
SAS ライブラリのメンバ、SAS カタログ内のエントリ、外部ファイル、外部ディレクトリのいずれかの現在の名前である文字定数、変数または式を指定します。

データセットの場合、*old-name* は 1 レベルまたは 2 レベルの名前にできます。カタログエントリの場合、*old-name* は 1 レベル、2 レベルまたは 4 レベルの名前にできます。外部ファイルまたはディレクトリの場合、*old-name* はファイルまたはディレクトリの完全パス名にする必要があります。*old-name* の値を指定しない場合は、現在のディレクトリが使用されます。

new-name
ライブラリメンバ、カタログエントリ、外部ファイル、ディレクトリのいずれかの新しい 1 レベルの名前である文字定数、変数また式を指定します。

オプション引数

type
名前を変更する要素の種類を指定する文字定数、変数または式です。*Type* は NULL 文字列にするか、次のいずれかの値を使用できます。

ACCESS	SAS/ACCESS ソフトウェアで作成された SAS/ACCESS ディスクリプタを指定します。
CATALOG	SAS カタログまたはカタログエントリを指定します。
DATA	SAS データセットを指定します。

VIEW SAS データセットビューを指定します。

FILE 外部ファイルまたはディレクトリを指定します。

デフォルト: 'DATA'

description
カタログエントリの説明である文字定数、変数または式を指定します。*description* は、*type* の値が CATALOG である場合にのみ指定できます。*Description* は NULL 引数にできます。

password
名前を変更するデータセットのパスワードを指定する文字定数、変数または式です。*Password* は NULL 引数にできます。

generation
名前を変更するデータセットの世代番号を指定する数値定数、変数または式です。*Generation* は NULL 引数にできます。

詳細

RENAME 関数を使用して、SAS ライブラリのメンバまたは SAS カタログ内のエントリの名前を変更できます。SAS は操作が成功した場合は 0、失敗した場合は 0 以外の値を返します。

カタログ内のエントリの名前を変更するには、*old-name* の 4 レベルの名前を指定し、*new-name* に 1 レベルの名前を指定します。カタログ内のエントリの名前を変更するには、*type* に CATALOG を指定する必要があります。

動作環境の情報
RENAME はディレクトリベースの動作環境でのみ使用します。メインフレーム動作環境で RENAME を使用すると、エラーが生成されます。

サンプル

サンプル 1: データセットとカタログエントリの名前の変更

次の例では、SAS データセットの名前を DATA1 から DATA2 に変更し、カタログエントリの名前を A.SCL から B.SCL に変更します。

```
rc1=rename('mylib.data1', 'data2');
rc2=rename('mylib.mycat.a.scl', 'b', 'catalog');
```

サンプル 2: 外部ファイルの名前変更

次の例では、外部ファイルの名前を変更します。

```
 /* Rename a file that is located in another directory. */
rc=rename('/local/u/testdir/first',
'/local/u/second', 'file');
/* Rename a PC file. */
rc=rename('d:¥temp', 'd:¥testfile', 'file');
```

サンプル 3: ディレクトリの名前変更

次の例では、UNIX 動作環境のディレクトリの名前を変更します。

```
rc=rename('/local/u/testdir/', '/local/u/oldtestdir', 'file');
```

サンプル 4: 世代データセット名の変更

次の例では、生成データセットの名前を WORK.ONE から WORK.TWO に変更します。WORK.ONE#003 のパスワードは *my-password* です。

```
rc=rename('work.one','two',,,3,'my-password');
```

関連項目:

関数:

- “FDELETE 関数” (396 ページ)
- “FILEEXIST 関数” (403 ページ)
- “EXIST 関数” (389 ページ)

REPEAT 関数

最初の引数を n+1 回繰り返した文字値を返します。

カテゴリ: 文字関数

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

REPEAT(*argument*,*n*)

必須引数

引数

文字定数、変数または式を指定します。

n

argument 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。

詳細

DATA ステップで、まだ長さが割り当てられていない変数に REPEAT 関数から値が返される場合、その変数には 200 バイトの長さが割り当てられます。

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。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=repeat('ONE',2); put x;	ONEONEONE

RESOLVE 関数

マクロ機能によって処理された後の引数の解決値を返します。

カテゴリ: マクロ

構文

RESOLVE(*argument*)

必須引数

引数
マクロ式の値となる文字定数、変数または式です。

詳細

まだ長さが割り当てられていない変数に RESOLVE 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。

RESOLVE の詳細については、*SAS マクロ言語: リファレンス*を参照してください。

関連項目:

関数:

- “SYMGET 関数” (876 ページ)

REVERSE 関数

文字列を逆にします。

カテゴリ: 文字関数

制限事項: I18N レベル 0 の関数は 1 バイト文字セット(SBCS)でのみ使用するように設計されています。

ヒント: DBCS に相当する関数は、KREVERSE(*SAS 各国語サポート(NLS): リファレンスガイド*の中で)です。

構文

REVERSE(*argument*)

必須引数

引数
文字定数、変数または式を指定します。

詳細

DATA ステップで、まだ長さが割り当てられていない変数に REVERSE 関数から値が返される場合、その変数には第 1 引数の長さが割り当てられます。

引数の最後の文字が結果の最初の文字になり、引数の最後から 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: 引数の末尾の空白は結果の先頭の空白になります。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
	----+----1
backward=reverse('xyz '); put backward $5.;	zyx

REWIND 関数

データセットポインタを SAS データセットの先頭に配置します。

カテゴリ: SAS ファイル I/O 関数

構文

REWIND(*data-set-id*)

必須引数

data-set-id
OPEN 関数が返すデータセット識別子を指定する数値変数です。

制限事項: IS モードでデータセットを開くことはできません。

詳細

REWIND は、処理が成功した場合は 0、失敗した場合は≠0 を返します。REWIND の呼び出し後、FETCH の呼び出しでデータセットの開始オブザベーションを読み込みます。

アクティブな WHERE 句がある場合、REWIND は WHERE 条件を満たす開始オブザベーションにデータセットポインタを移動します。

サンプル

この例では、FETCHOBS を呼び出して、データセット MYDATA の 10 番目のオブザベーションをフェッチします。次に、REWIND を呼び出して開始オブザベーションに戻り、開始オブザベーションをフェッチします。

```
%let dsid=%sysfunc(open(mydata,i));
%let rc=%sysfunc(fetchobs(&dsid,10));
%let rc=%sysfunc(rewind(&dsid));
%let rc=%sysfunc(fetch(&dsid));
```

関連項目:

関数:

- “FETCH 関数” (398 ページ)
- “FETCHOBS 関数” (399 ページ)
- “FREWIND 関数” (487 ページ)
- “NOTE 関数” (674 ページ)
- “OPEN 関数” (697 ページ)
- “POINT 関数” (725 ページ)

RIGHT 関数

文字式を右揃えにします。

カテゴリ: 文字関数

制限事項: I18N レベル 0 の関数は 1 バイト文字セット(SBCS)でのみ使用するように設計されています。

ヒント: DBCS に相当する関数は、KRIGHT です。

構文

RIGHT(*argument*)

必須引数

引数
文字定数、変数または式を指定します。

詳細

DATA ステップで、まだ長さが割り当てられていない変数に RIGHT 関数から値が返される場合、その変数には第 1 引数の長さが割り当てられます。

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。結果の長さは、引数の長さと同じです。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
	----+----1----+

SAS ステートメント	結果
a='Due Date '; b=right(a); put a $10.; put b $10.;	 Due Date Due Date

関連項目:

関数:

- “COMPRESS 関数” (321 ページ)
- “LEFT 関数” (601 ページ)
- “TRIM 関数” (899 ページ)

RMS 関数

非欠損引数の 2 乗平均平方根を返します。

カテゴリ: 記述統計

構文

RMS(*argument*<,*argument*,…>)

必須引数

引数

数値定数、変数または式です。

ヒント: 引数リストには OF で始まる変数のリストを含められます。

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。すべての引数が欠損値の場合、結果は欠損値になります。それ以外の場合、結果は非欠損値の 2 乗平均平方根になります。

n は、非欠損値の引数の数で、$x_1, x_2, \ldots, x_n$ は、それらの引数の値です。2 乗平均平方根は次のようになります。

$$\sqrt{\frac{x_1^2 + x_2^2 + \ldots + x_n^2}{n}}$$

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x1=rms(1,7);	5

SAS ステートメント	結果
x2=rms(.,1,5,11);	7
x3=rms(of x1-x2);	6.0827625303

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: 切り捨て関数

構文

ROUND (*argument*<,*rounding-unit*>)

必須引数

引数
丸める数値定数、変数または式です。

オプション引数

rounding-unit
丸め単位を指定する正の数値定数、変数または式です。

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。次の例に示すように、これらの 2 つの値を SAS に書き込むと、同じ値のように見えます。ただし、差異を計算すると、値が異なることがわかります。

```
data _null_;
point_three=0.3;
three_times_point_one=3*0.1;
difference=point_three - three_times_point_one;
put point_three= ;
put three_times_point_one= ;
put difference= ;
run;
```

次の行が SAS ログに書き込まれます。

```
 point_three= 0.3
three_times_point_one= 0.3
difference= -5.55112E-17
```

動作環境の情報

前述の例は、z/OS 環境で実行されています。他の動作環境を使用する場合、結果がわずかに異なります。

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

SAS プログラムで文字"0.3"が定数として表示されている場合、値は標準の入力形式で 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 ... その他の SAS ステートメント...
```

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 SAS statements ...
```

SAS 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 が表示されている場合と同じ結果になります。SAS 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 進表現が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データセットを移植する場合、正確な結果を得られない可能性もあります。

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 ... その他の SAS ステートメント...
```

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 ... その他の SAS ステートメント...
```

期待される結果の生成

通常、ROUND(argument, rounding-unit)は、結果の有効桁数が 9 以下で、次のいずれかの条件が該当する場合、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 (丸め単位がこれよりも小さいと、最大 3 ~ 4 桁の最下位ビットで丸め単位の逆数が 10 のべき乗ではない場合に ROUND は丸め単位を 10 のべき乗として扱います)。
- 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。

```
options pageno=1 nodate;

data rounding;
d1 = round(1234.56789,100) - 1200;
d2 = round(1234.56789,10) - 1230;
d3 = round(1234.56789,1) - 1235;
d4 = round(1234.56789,.1) - 1234.6;
d5 = round(1234.56789,.01) - 1234.57;
d6 = round(1234.56789,.001) - 1234.568;
d7 = round(1234.56789,.0001) - 1234.5679;
d8 = round(1234.56789,.00001) - 1234.56789;
d9 = round(1234.56789,.1111) - 1234.5432;
/* d10 has too many decimal places in the value for */
/* rounding-unit. */
d10 = round(1234.56789,.11111) - 1234.54321;
run;
proc print data=rounding noobs;
run;
```

画面 2.54 丸め単位の値に基づいて丸めた場合の結果

```
                              The SAS System                              1

d1    d2    d3    d4    d5    d6    d7    d8    d9          d10

 0     0     0     0     0     0     0     0     0    -1.7053E-13
```

動作環境の情報

前述の例は、z/OS 環境で実行されています。他の動作環境を使用する場合、結果がわずかに異なります。

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、ROUND 関数は整数で除算して結果を計算します (最大 3 ~ 4 桁の最下位ビットで丸め単位の逆数が整数ではない場合に ROUND は丸め単位を整数の逆数として扱います)。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。

```
options pageno=1 nodate;

data rounding2;
drop pi unit;
```

```
pi = arcos(-1);
unit=1/7;
d1=round(pi,unit) - 22/7;
d2=round(pi, unit) - 22*unit;
run;
proc print data=rounding2 noobs;
run;
```

画面 2.55　整数の逆数で丸めた場合の結果

```
 The SAS System                                    1

  d1          d2

   0     2.2204E-16
```

動作環境の情報
前述の例は、z/OS 環境で実行されています。他の動作環境を使用する場合、結果がわずかに異なります。

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。
- 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。

```
data _null_;
difference=round(1234.56789,.11111) - 11111*.11111;
put difference=;
run;
```

次の行が SAS ログに書き込まれます。

```
 difference=0
```

動作環境の情報
前述の例は、z/OS 環境で実行されています。他の動作環境を使用する場合、結果がわずかに異なる場合があります。

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。

```
data test;
do i=8 to 17;
value=0.5 - 10**(-i);
round=round(value);
output;
end;
do i=8 to 17;
```

```
value=-0.5 + 10**(-i);
round=round(value);
output;
end;
run;
proc print data=test noobs;
format value 19.16;
run;
```

画面 2.56 *丸め単位の倍数の中間にある値を丸める場合の結果*

The SAS System

i	value	round
8	0.4999999900000000	0
9	0.4999999990000000	0
10	0.4999999999000000	0
11	0.4999999999900000	0
12	0.4999999999990000	0
13	0.4999999999999000	1
14	0.4999999999999900	1
15	0.4999999999999900	1
16	0.5000000000000000	1
17	0.5000000000000000	1
8	-0.4999999900000000	0
9	-0.4999999990000000	0
10	-0.4999999999000000	0
11	-0.4999999999900000	0
12	-0.4999999999990000	0
13	-0.4999999999999000	-1
14	-0.4999999999999900	-1
15	-0.4999999999999900	-1
16	-0.5000000000000000	-1
17	-0.5000000000000000	-1

動作環境の情報

前述の例は、z/OS 環境で実行されています。他の動作環境を使用する場合、結果がわずかに異なる場合があります。

近似は丸める値のサイズに関連し、次の DATA ステップのように計算されます。この DATA ステップコードの結果と 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。

```
data testfile;
do i = 1 to 17;
value = 0.5 - 10**(-i);
epsilon = min(1e-6, value * 1e-12);
temp = value + .5 + epsilon;
fraction = modz(temp, 1);
round = temp - fraction;
output;
end;
```

```
run;
proc print data=testfile noobs;
format value 19.16;
run;
```

比較

ROUND、ROUNDE および ROUNDZ 関数は類似していますが、次の 4 点が異なります。

- ROUND は、第 1 引数が第 2 引数の最も近い 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。
- ROUNDE は、第 1 引数が第 2 引数の最も近い 2 つの倍数のほぼ中間にある場合には偶数の倍数を返します。
- 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。
- 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。通常、ROUND と ROUNDE 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。ROUNDZ は、結果をファジー処理しません。

サンプル

次の例では、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、UNIX 動作環境で生成されています。

```
data results;
do x=0 to 4 by .25;
Rounde=rounde(x);
Round=round(x);
output;
end;
run;
proc print data=results noobs;
run;
```

画面 2.57 ROUND 関数と ROUNDE 関数によって返される結果

The SAS System

x	Rounde	Round
0.00	0	0
0.25	0	0
0.50	0	1
0.75	1	1
1.00	1	1
1.25	1	1
1.50	2	2
1.75	2	2
2.00	2	2
2.25	2	2
2.50	2	3
2.75	3	3
3.00	3	3
3.25	3	3
3.50	4	4
3.75	4	4
4.00	4	4

関連項目:

関数:

- “CEIL 関数” (288 ページ)
- “CEILZ 関数” (289 ページ)
- “FLOOR 関数” (470 ページ)
- “FLOORZ 関数” (471 ページ)
- “INT 関数” (543 ページ)
- “INTZ 関数” (582 ページ)
- “ROUNDE 関数” (816 ページ)
- “ROUNDZ 関数” (819 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: 切り捨て関数

構文

ROUNDE (*argument*<,*rounding-unit*>)

必須引数

引数
丸める数値定数、変数または式です。

オプション引数

rounding-unit
丸め単位を指定する正の数値定数、変数または式です。

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。第 2 引数を省略すると、ROUNDE は *rounding-unit* にデフォルト値の 1 を使用します。

比較

ROUND、ROUNDE および ROUNDZ 関数は類似していますが、次の 4 点が異なります。

- ROUND は、第 1 引数が第 2 引数の最も近い 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。
- ROUNDE は、第 1 引数が第 2 引数の最も近い 2 つの倍数のほぼ中間にある場合には偶数の倍数を返します。
- 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。
- 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。通常、ROUND と ROUNDE 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。ROUNDZ は、結果をファジー処理しません。

サンプル

次の例では、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。

```
data results;
do x=0 to 4 by .25;
Rounde=rounde(x);
Round=round(x);
output;
end;
run;
proc print data=results noobs;
run;
```

画面 2.58 ROUNDE 関数と ROUND 関数によって返される結果

The SAS System

x	Rounde	Round
0.00	0	0
0.25	0	0
0.50	0	1
0.75	1	1
1.00	1	1
1.25	1	1
1.50	2	2
1.75	2	2
2.00	2	2
2.25	2	2
2.50	2	3
2.75	3	3
3.00	3	3
3.25	3	3
3.50	4	4
3.75	4	4
4.00	4	4

関連項目:

関数:

- “CEIL 関数” (288 ページ)
- “CEILZ 関数” (289 ページ)
- “FLOOR 関数” (470 ページ)
- “FLOORZ 関数” (471 ページ)
- “INT 関数” (543 ページ)

- “INTZ 関数” (582 ページ)
- “ROUND 関数” (810 ページ)
- “ROUNDZ 関数” (819 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: 切り捨て関数

構文

ROUNDZ (*argument*<,*rounding-unit*>)

必須引数

引数
丸める数値定数、変数または式です。

オプション引数

rounding-unit
丸め単位を指定する正の数値定数、変数または式です。

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。第 2 引数を省略すると、ROUNDZ は *rounding-unit* にデフォルト値の 1 を使用します。

比較

ROUND、ROUNDE および ROUNDZ 関数は類似していますが、次の 4 点が異なります。

- ROUND は、第 1 引数が第 2 引数の最も近い 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。
- ROUNDE は、第 1 引数が第 2 引数の最も近い 2 つの倍数のほぼ中間にある場合には偶数の倍数を返します。
- 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。
- 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。通常、ROUND と ROUNDE 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。ROUNDZ は、結果をファジー処理しません。

サンプル

サンプル 1: ROUNDZ 関数と ROUND 関数の結果の比較

次の例では、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。

```
data test;
do i=10 to 17;
Value=3.5 - 10**(-i);
Roundz=roundz(value);
Round=round(value);
output;
end;
do i=16 to 12 by -1;
value=3.5 + 10**(-i);
roundz=roundz(value);
round=round(value);
output;
end;
run;
proc print data=test noobs;
format value 19.16;
run;
```

画面 2.59 ROUNDZ 関数と ROUND 関数によって返される結果

The SAS System

i	Value	Roundz	Round
10	3.4999999999000000	3	3
11	3.4999999999900000	3	3
12	3.4999999999990000	3	4
13	3.4999999999999000	3	4
14	3.4999999999999900	3	4
15	3.5000000000000000	3	4
16	3.5000000000000000	4	4
17	3.5000000000000000	4	4
16	3.5000000000000000	4	4
15	3.5000000000000000	4	4
14	3.5000000000000100	4	4
13	3.5000000000001000	4	4
12	3.5000000000010000	4	4

サンプル 2: ROUNDZ 関数からの出力例

これらの例では、ROUNDZ によって返される結果を示します。

表 2.2 ROUNDZ を使用した場合の結果

SAS ステートメント	結果

var1=223.456;
x=roundz(var1,1);
put x 9.5;
	223.00000

var2=223.456;
x=roundz(var2,.01);
put x 9.5;
	223.46000

x=roundz(223.456,100); put x 9.5;	200.00000
x=roundz(223.456); put x 9.5;	223.00000
x=roundz(223.456,.3); put x 9.5;	223.50000

関連項目:

関数:

- “ROUND 関数” (810 ページ)
- “ROUNDE 関数” (816 ページ)

SAVING 関数

定期預金の将来価値を返します。

カテゴリ: 財務関数

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: $f \geq 0$

p

数値の固定定期的支払いです。

範囲: $p \geq 0$

r
10 進で表される数値の定期利率です。

範囲: $r \geq 0$

n
整数の複利期間数です。

範囲: $n \geq 0$

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。引数は、次の式に関係があります。

$$f = \frac{p(1+r)\left((1+r)^n - 1\right)}{r}$$

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。

サンプル

ある定期預金口座は 5 パーセント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。

```
number=saving(12000,100,.05/12,.);
```

戻り値は 97.18 か月になります。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 パーセントの名目年利は、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。

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: 財務関数

構文

SAVINGS(*base-date*, *initial-deposit-date*, *deposit-amount*, *deposit-number*, *deposit-interval*, *compounding-interval*, *date-1*, *rate-2* <,*date-2*, *rate-2*,...>)

必須引数

base-date
SAS 日付です。戻り値は、*base-date* の預金残高になります。

initial-deposit-date
SAS 日付です。*initial-deposit-date* は、初回入金日です。後続の入金間隔の開始時に次の入金が行われます。

deposit-amount
数値です。すべての入金は定数を前提としています。*deposit-amount* は、各入金の値です。

deposit-number
正の整数です。*deposit-number* は、入金の回数です。

deposit-interval
SAS 間隔です。*deposit-interval* は、入金の頻度です。

compounding-interval
SAS 間隔です。*compounding-interval* は、複利間隔です。

date
SAS 日付です。各日付は利率とペアになります。*date* は、*rate* が有効になるタイミングです。

rate
数値のパーセントです。各利率は日付とペアになります。*rate* は、*date* に開始する利率です。

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 ~ 120 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。

サンプル

- 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。

```
amount_base1 = SAVINGS(″01jan2005″d, ″01jan2000″d, 300, 24,
″MONTH″, ″QUARTER″, ″01jan2000″d, 4.00);
```

- 利率が毎年 0.25%ずつ増加する場合、口座残高は次のように表すことができます。

```
amount_base2 = SAVINGS(″01jan2005″d, ″01jan2000″d, 300, 24,
″MONTH″, ″QUARTER″, ″01jan2000″d, 4.00,
″01jan2001″d, 4.25, ″01jan2002″d, 4.50,
″01jan2003″d, 4.75, ″01jan2004″d, 5.00);
```

- 次のステートメントでは、1 年間の入金後の残高を決定するために amount_base3 に目的の残高を設定します。

```
amount_base3 = SAVINGS(″01jan2001″d, ″01jan2000″d, 300, 24,
″MONTH″, ″QUARTER″, ″01jan2000″d, 4);
```

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。

SCAN 関数

文字列から *n* 番目の単語を返します。

カテゴリ: 文字関数

制限事項: I18N レベル 0 の関数は 1 バイト文字セット(SBCS)でのみ使用するように設計されています。

ヒント: DBCS に相当する関数は、DBCS に相当する関数は、KSCAN です。

構文

SCAN(*string*, *count*<,*charlist*<,*modifiers*>>)

必須引数

string
文字定数、変数または式を指定します。

count
SCAN で選択する文字列の単語数を指定する整数値となるゼロ以外の数値定数、変数または式です。たとえば、1 の値は 1 番目の単語、2 の値は 2 番目の単語を示します。次のルールが適用されます。

- *count* が正の場合、SCAN は文字列の単語を左から右へ数えます。
- *count* が負の場合、SCAN は文字列の単語を右から左へ数えます。

オプション引数

charlist
文字のリストを初期化する文字式を指定します(省略可能)。このリストは、単語を区切る区切り文字として使用する文字を決定します。次のルールが適用されます。

- デフォルトでは、*charlist* のすべての文字が区切り文字として使用されます。
- *modifier* 引数で K 修飾子を指定すると、*charlist* に *ない* すべての文字が区切り文字として使用されます。

ヒント: その他の修飾子を使用して *charlist* にさらに文字を追加できます。

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。空白は無視されます。次の文字を修飾子として使用できます。

a または A	文字のリストにアルファベット文字を追加します。
b または B	*count* 引数の符合に関係なく、逆方向(左から右ではなく右から左)にスキャンします。
c または C	文字のリストに制御文字を追加します。

d または D　文字のリストに数字を追加します。

f または F　文字のリストにアンダースコアと英文字(VALIDVARNAME=V7 を使用した SAS 変数名内の有効な最初の文字)を追加します。

g または 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 または H　文字のリストに水平タブを追加します。

i または I　大文字か小文字かは無視します。

k または 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 を指定すると、文字のリストに含まれている文字は区切り文字であるため、戻り値から除外されずにそのまま使用されます。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 または L　小文字を文字リストに追加します。

m または 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 または 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 または O　*charlist* 引数および *modifier* 引数を 1 回だけ処理します。SCAN 関数の呼び出し時に毎回処理しません。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、*charlist* および *modifier* 引数が変更されないループで SCAN を呼び出すときに、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 または P　文字のリストに句読点を追加します。

q または 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、一致しない引用符が含まれている場合、左から右へのスキャンでは、右から左へのスキャンとは異なる単語が生成されます。

r または 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 修飾子の両方を指定すると、SCAN 関数は単語からまず先頭および末尾の空白を削除します。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 または 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 または T	*string* 引数と *charlist* 引数から末尾の空白を取り除きます。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 または U	大文字を文字リストに追加します。
w または W	文字のリストに印刷可能(書き込み可能)な文字を追加します。
x または X	文字のリストに 16 進文字を追加します。

ヒント: *modifier* 引数が文字定数の場合は引用符で囲みます。一組の引用符で複数の修飾子を指定します。*modifier* 引数は、文字変数または文字式としても表すことができます。

詳細

"区切り文字"および"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。区切り文字は *charlist* 引数と *modifier* 引数で指定できます。

Q 修飾子を指定すると、引用符で囲まれた部分文字列内の区切り文字は無視されます。

SCAN 関数では、"単語"は、次のすべての特性がある部分文字列です。

- 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

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、長さがゼロの単語を無視されます。

注: 単語"の定義は SCAN 関数と COUNTW 関数の両方に共通します。

ASCII 環境と EBCDIC 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、コンピュータで使用している文字(ASCII または EBCDIC)によって異なります。

- コンピュータで ASCII 文字が使われている場合、デフォルトの区切り文字は次のとおりです。

  空白 ! $ % & ( ) * + , - . / ; < ^ |

  ^文字のない ASCII 環境の場合、SCAN 関数はかわりに~文字を使用します。

- コンピュータで EBCDIC 文字が使われている場合、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 環境と EBCDIC 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、*charlist* 引数で明示的に指定された区切り文字のリストに追加します。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

DATA ステップの多くの変数は固定長です。SCAN 関数によって返される単語がその単語の長さよりも長い固定長の変数に割り当てられる場合、その変数の値に空白が埋め込まれます。マクロ変数は可変長であるため、空白は埋め込まれません。

SCAN 関数によって返される単語の最大長は、呼び出し元の環境によって異なります。

- DATA ステップで、まだ長さが割り当てられていない変数に SCAN 関数から値が返される場合、その変数には 200 文字の長さが割り当てられます。SCAN 関数で 200 文字より長い単語を変数に割り当てる場合、その変数の長さを明示的に指定する必要があります。

  演算子または他の関数を含む式で SCAN 関数を使用する場合、SCAN 関数によって返される単語には最大で 32,767 文字を割り当てることができます。ただし、WHERE 句の場合は除きます。WHERE 句の場合、最大長は 200 文字になります。

- SQL プロシジャまたはプロシジャの WHERE 句では、SCAN 関数によって返される単語の最大長は 200 文字になります。

- マクロプロセッサでは、SCAN 関数によって返される単語の最大長は 65,534 文字になります。

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 修飾子を指定した SCAN 関数の使用” (827 ページ)を参照してください。“M 修飾子を指定しない SCAN 関数の使用” (827 ページ)も参照してください。

M 修飾子を指定した SCAN 関数の使用

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。

- 文字列が区切り文字で始まり、1 番目の単語を要求する。
- 文字列が区切り文字で終わり、最後の単語を要求する。
- 文字列に 2 つの連続する区切り文字が含まれており、2 つの区切り文字の間の単語を要求する。

M 修飾子を指定しない SCAN 関数の使用

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 関数は次のように処理します。

- 文字列の先頭または末尾の区切り文字を無視する

- 2 つ以上の連続する区切り文字を単一の区切り文字として扱う

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 引数を使用する

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。

サンプル

サンプル 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。注:

- 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。

```
data firstlast;
input String $60.;
First_Word = scan(string, 1);
Last_Word = scan(string, -1);
datalines4;
Jack and Jill
& Bob & Carol & Ted & Alice &
Leonardo
! $ % & ( ) * + , - . / ;
;;;;
proc print data=firstlast;
run;
```

画面 2.60 *文字列内の最初と最後の単語の検索結果*

The SAS System

Obs	String	First_Word	Last_Word
1	Jack and Jill	Jack	Jill
2	& Bob & Carol & Ted & Alice &	Bob	Alice
3	Leonardo	Leonardo	Leonardo
4	! $ % & ( ) * + , - . / ;		

サンプル 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

次の例では、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。

```
data all;
length word $20;
drop string;
string = ' The quick brown fox jumps over the lazy dog. ';
do until(word=' ');
count+1;
word = scan(string, count);
output;
end;
run;
proc print data=all noobs;
run;
```

画面 2.61 *M 修飾子を使用しないですべての単語を検索する場合の結果*

The SAS System

word	count
The	1
quick	2
brown	3
fox	4
jumps	5
over	6
the	7
lazy	8
dog	9
	10

サンプル 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。そのため、空白の単語をテストしてループを終了させないでください。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。

```
data comma;
keep count word;
length word $30;
string = ',leading, trailing,and multiple,,delimiters,,';
delim = ',';
modif = 'mo';
nwords = countw(string, delim, modif);
do count = 1 to nwords;
word = scan(string, count, delim, modif);
output;
end;
run;
proc print data=comma noobs;
run;
```

画面 2.62 *M 修飾子と O 修飾子を使用してすべての単語を検索した場合の結果*

The SAS System

word	count
	1
leading	2
trailing	3
and multiple	4
	5
delimiters	6
	7
	8

サンプル 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

次の例では、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。そのため、このインスタンスには O 修飾子を使用できます。
- 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。そのため、このインスタンスには O 修飾子を使用できます。

- 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。

```
data test;
keep count word word_r;
length word word_r $30;
string = 'He said, "She said, ""No!""", not "Yes!"';
delim = ',';
modif = 'oq';
nwords = countw(string, delim, modif);
do count = 1 to nwords;
word = scan(string, count, delim, modif);
word_r = scan(string, count, delim, modif||'r');
output;
end;
run;
proc print data=test noobs;
run;
```

画面 2.63 *カンマ区切り値と引用符内の部分文字列の結果*

The SAS System

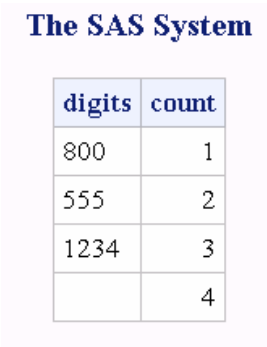
word	word_r	count
He said	He said	1
"She said, ""No!"""	She said, "No!"	2
not "Yes!"	not "Yes!"	3

サンプル 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

数字の部分文字列を検索する例を次に示します。*charlist* 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。

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

画面 2.64 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した結果



The SAS System

digits	count
800	1
555	2
1234	3
	4

関連項目:

関数:

- “COUNTW 関数” (337 ページ)
- “FINDW 関数” (457 ページ)

CALL ルーチン:

- “CALL SCAN ルーチン” (233 ページ)

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” (273 ページ)

構文

SDF(*dist*,*quantile*,*parm-1*,...,*parm-k*)

必須引数

dist

分布を識別する文字列です。有効な分布は、次のとおりです。

分布	引数
ベルヌーイ	BERNOULLI
ベータ	BETA
二項	BINOMIAL
コーシー	CAUCHY
$\chi 2$ 乗	CHISQUARE

分布	引数
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
ポアソン	POISSON
T	T
Tweedie	TWEEDIE
一様	UNIFORM
逆ガウス(Wald)	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。

等量分類
ランダム変数の値を指定する数値定数、変数または式です。

parm-1,...,parm-k
特定の分布に適した任意の *shape*、*location* または *scale* パラメータです。

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、“CDF 関数” (273 ページ)を参照してください。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
y=sdf('BERN',0,.25);	0.25
y=sdf('BETA',0.2,3,4);	0.09011
y=sdf('BINOM',4,.5,10);	0.62305
y=sdf('CAUCHY',2);	0.14758
y=sdf('CHISQ',11.264,11);	0.42142
y=sdf('EXPO',1);	0.36788
y=sdf('F',3.32,2,3);	0.17361
y=sdf('GAMMA',1,3);	0.91970
y=sdf('GENPOISSON',.9,1,.7);	0.6321205588
y=sdf('HYPER',2,200,50,10);	0.47633
y=sdf('LAPLACE',1);	0.18394
y=sdf('LOGISTIC',1);	0.26894
y=sdf('LOGNORMAL',1);	0.5
y=sdf('NEGB',1,.5,2);	0.5
y=sdf('NORMAL',1.96);	0.025
y=pdf('NORMALMIX',2.3,3,.33,.33,.34, .5,1.5,2.5,.79,1.6,4.3);	0.2819
y=sdf('PARETO',1,1);	1
y=sdf('POISSON',2,1);	0.08030
y=sdf('T',.9,5);	0.20469
y=sdf('TWEEDIE',.8,5);	0.4082370836
y=sdf('UNIFORM',0.25);	0.75
y=sdf('WALD',1,2);	0.37230
y=sdf('WEIBULL',1,2);	0.36788

関連項目:

関数:

- “CDF 関数” (273 ページ)
- “LOGCDF 関数” (624 ページ)
- “LOGPDF 関数” (626 ページ)
- “LOGSDF 関数” (628 ページ)
- “PDF 関数” (703 ページ)
- “QUANTILE 関数” (778 ページ)
- “SQUANTILE 関数” (856 ページ)

SECOND 関数

SAS 時間値または SAS 日時値の秒数を返します。

カテゴリ: 日付と時間

構文

SECOND(*time* | *datetime*)

必須引数

time
SAS 時間値を表す値となる数値定数、変数または式です。

datetime
SAS 日時値を表す値となる数値定数、変数または式です。

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。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果

time='3:19:24't;
s=second(time);
put s;
	24

time='6:25:65't;
s=second(time);
put s;
	5

SAS ステートメント	結果

time='3:19:60't;
s=second(time);
put s;
	0

関連項目:

関数:

- “HOUR 関数” (522 ページ)
- “MINUTE 関数” (644 ページ)

SIGN 関数

値の符合を返します。

カテゴリ: 数学関数

構文

SIGN(*argument*)

必須引数

引数
数値の定数、変数または式を指定します。

詳細

SIGN 関数は次の値を返します。

-1
argument < 0 の場合

0
argument = 0 の場合

1
argument > 0 の場合

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=sign(-5);	-1
x=sign(5);	1
x=sign(0);	0

SIN 関数

正弦を返します。

カテゴリ: 三角関数

構文

SIN(*argument*)

必須引数

引数
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。*argument* が非常に大きく、**mod(argument,pi)**の誤差がおよそ小数点以下 3 桁より小さい場合、SIN は欠損値を返します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=sin(0.5);	0.4794255386
x=sin(0);	0
x=sin(3.14159/4);	.7071063121

SINH 関数

双曲線正弦を返します。

カテゴリ: 双曲線関数

構文

SINH(*argument*)

必須引数

引数
数値の定数、変数または式を指定します。

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。

$$\left(\varepsilon^{argument} - \varepsilon^{-argument}\right) / 2$$

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=sinh(0);	0
x=sinh(1);	1.1752011936
x=sinh(-1.0);	-1.175201194

SKEWNESS 関数

非欠損引数の歪度を返します。

カテゴリ: 記述統計

構文

SKEWNESS(*argument-1*,*argument-2*,*argument-3* <,…*argument-n*>)

必須引数

引数
数値の定数、変数または式を指定します。

詳細

3 つ以上の非欠損引数が必要です。非欠損引数がない場合は、関数から欠損値が返されます。すべて
ERROR_ を 1 に設定します。

引数リストには OF で始まる変数のリストを含められます。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x1=skewness(0,1,1);	-1.732050808
x2=skewness(2,4,6,3,1);	0.5901286564
x3=skewness(2,0,0);	1.7320508076
x4=skewness(of x1-x3);	-0.953097714

SLEEP 関数

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。

カテゴリ: 特殊関数

参照項目: “SLEEP Function: Windows” in *SAS Companion for Windows*

構文

SLEEP(*n* <, *unit*>)

必須引数

n

プログラムの実行を中断する時間単位数を指定する数値定数、変数または式です。

範囲: $n \geq 0$

オプション引数

unit

n に適用される時間の単位(秒)を指定します。たとえば、1 は 1 秒、.001 は 1 ミリ秒、5 は 5 秒に相当します。

デフォルト: Windows PC 環境では 1、その他の環境では.001

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。プログラムには、DATA ステップ、マクロ、IML、SCL などの関数を呼び出せるものを指定できます。SLEEP 関数の最大スリープ期間は 46 日です。

サンプル

サンプル 1: 指定した期間の実行中断

次の例では、DATA ステップ PAYROLL の実行を 20 秒遅延させるように SAS に指示します。

```
data payroll;
time_slept=sleep(20,1);
...more SAS statements...
run;
```

サンプル 2: スリープ時間の計算に基づいた実行中断

次に、DATA ステップ BUDGET の実行を 2013 年 3 月 1 日午前 3 時まで中断するように SAS に指示する例を示します。SAS では、対象日と DATA ステップの実行開始日時に基づいて中断の長さが計算されます。

```
data budget;
sleeptime='01mar2013:03:00'dt-'01mar2013:2:59:30'dt;
time_calc=sleep(sleeptime,1);
put 'Calculation of sleep time:';
```

```
put sleeptime='seconds';
run;
```

SAS は次の出力を口グに書き込みます。

```
Calculation of sleep time:
sleeptime=30 seconds
```

関連項目:

CALL ルーチン:

- “CALL SLEEP ルーチン” (243 ページ)

SMALLEST 関数

k 番目に小さい非欠損値を返します。

カテゴリ: 記述統計

構文

SMALLEST (*k*, *value-1* <, *value-2*…>)

必須引数

k
返す値を指定する数値定数、変数または式です。

value
数値の定数、変数または式を指定します。

詳細

k
が欠損値、0 未満、または value の数よりも大きい値の場合、結果は欠損値になり、_ERROR_が 1 に設定されます。または、*k*
が非欠損値の value の数よりも大きい場合、結果は欠損値になりますが、_ERROR_は 1 に設定されません。

比較

SMALLEST 関数は ORDINAL 関数とは異なります。SMALLEST 関数は欠損値を無視しますが、ORDINAL 関数は欠損値を数えます。

サンプル

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。

```
data comparison;
label smallest_num='SMALLEST Function' ordinal_num='ORDINAL Function';
do k = 1 to 4;
smallest_num = smallest(k, 456, 789, .Q, 123);
ordinal_num = ordinal (k, 456, 789, .Q, 123);
```

```
output;
end;
run;
proc print data=comparison label noobs;
var k smallest_num ordinal_num;
title 'Results From the SMALLEST and the ORDINAL Functions';
run;
```

画面 2.65 値の比較: SMALLEST 関数と ORDINAL 関数

Results From the SMALLEST and the ORDINAL Functions

k	SMALLEST Function	ORDINAL Function
1	123	Q
2	456	123
3	789	456
4	.	789

関連項目:

関数:

- “LARGEST 関数” (597 ページ)
- “ORDINAL 関数” (699 ページ)
- “PCTL 関数” (702 ページ)

SOAPWEB 関数

基本 Web 認証を使用して Web サービスを呼び出します。認証情報は引数で指定します。

カテゴリ: Web サービス

構文

SOAPWEB (IN, URL <,*options*>)

必須引数

IN
ファイル参照名の文字値を指定します。IN は、SOAP 要求を含む XML データを入力するために使用されます。

URL
Web サービスエンドポイントの URL の文字値を指定します。

オプション引数

OUT
SOAP 応答の出力 XML が書き込まれるファイル参照名の文字値を指定します。

SOAPACTION
Web サービスで呼び出す SOAPAction 要素の文字値を指定します。

WEBUSERNAME
基本 Web 認証または NTLM Web 認証のユーザー名である文字値を指定します。

WEBPASSWORD
基本 Web 認証または NTLM Web 認証のパスワードである文字値を指定します。PROC PWENCODE によって生成されるエンコーディングがサポートされます。

WEBDOMAIN
NTLM Web 認証のユーザー名とパスワードを使うドメインまたは領域である文字値を指定します。

MUSTUNDERSTAND
SOAP ヘッダーの mustUnderstand 属性設定の数値を指定します。

PROXYPORT
HTTP プロキシサーバーポートの数値を指定します。

PROXYHOST
HTTP プロキシサーバーホストの文字値を指定します。

PROXYUSERNAME
HTTP プロキシサーバーユーザー名の文字値を指定します。

PROXYPASSWORD
HTTP プロキシサーバーパスワードの文字値を指定します。PROC PWENCODE によって生成されるエンコーディングがサポートされます。

CONFIGFILE
主としてタイムアウト値の設定に使用する Spring 構成ファイルである文字値を指定します。

DEBUG
ログ出力のデバッグに使用するファイルのフルパスの文字値を指定します。

サンプル

次の例では、DATA ステップで SOAPWEB 関数を使用する方法を示します。

```
FILENAME request 'c:¥temp¥Request.xml';
FILENAME response 'c:¥temp¥Response.xml';

data _null_;
url="http://www.weather.gov/forecasts/xml/SOAP_server/ndfdXMLserver.php";
soapaction=
"http://www.weather.gov/forecasts/xml/DWMLgen/wsdl/ndfdXML.wsdl#CornerPoints";
proxyhost="someproxy.abc.xyz.com";
proxyport=80;

rc = soapweb("request", url, "response", soapaction, , , , , proxyport,
proxyhost);
```

```
run;
```

このセクションでは SOAP 要求について説明します。

```
Request.xml:
<soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ndf="http://www.weather.gov/forecasts/xml/DWMLgen/wsdl/ndfdXML.wsdl">
<soapenv:Header/>
<soapenv:Body>
<ndf:CornerPoints soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/">
<sector xsi:type="dwml:sectorType"
xmlns:dwml="http://www.weather.gov/forecasts/xml/DWMLgen/schema/DWML.xsd">
alaska</sector>
</ndf:CornerPoints>
</soapenv:Body>
</soapenv:Envelope>
```

関連項目:

関数:

- “SOAPWS 関数” (849 ページ)
- “SOAPWSMETA 関数” (851 ページ)
- “SOAPWEBMETA 関数” (843 ページ)
- “SOAPWIPSERVICE 関数” (845 ページ)
- “SOAPWIPSRS 関数” (847 ページ)

SOAPWEBMETA 関数

基本 Web 認証を使用して Web サービスを呼び出します。認証ドメインの認証情報はメタデータから取得されます。

カテゴリ: Web サービス

構文

SOAPWEBMETA (IN, URL <,*options*>)

必須引数

IN
ファイル参照名の文字値を指定します。IN は、SOAP 要求を含む XML データを入力するために使用されます。

URL
Web サービスエンドポイントの URL の文字値を指定します。

オプション引数

OUT
SOAP 応答の出力 XML が書き込まれるファイル参照名の文字値を指定します。

SOAPACTION
Web サービスで呼び出す SOAPAction 要素の文字値を指定します。

WEBAUTHDOMAIN
基本 Web 認証用のユーザー名とパスワードをメタデータから取得する認証ドメインの文字値を指定します。

MUSTUNDERSTAND
SOAP ヘッダーの mustUnderstand 属性設定の数値を指定します。

PROXYPORT
HTTP プロキシサーバーポートの数値を指定します。

PROXYHOST
HTTP プロキシサーバーホストの文字値を指定します。

PROXYUSERNAME
HTTP プロキシサーバーユーザー名の文字値を指定します。

PROXYPASSWORD
HTTP プロキシサーバーパスワードの文字値を指定します。PROC PWENCODE によって生成されるエンコーディングがサポートされます。

CONFIGFILE
主にタイムアウト値を設定する Spring 構成ファイルの文字値を指定します。

DEBUG
ログ出力のデバッグに使用するファイルのフルパスの文字値を指定します。

サンプル

次の例では、DATA ステップで SOAPWEBMETA 関数を使用する方法を示します。

```
FILENAME request 'C:¥temp¥Request.xml';
FILENAME response 'C:¥temp¥Response.xml';

OPTIONS metauser="metadata-user"
metapass="password"
metaprotocol=bridge
metaport=8561
metaserver="somemachine.abc.xyz.com";

data _null_;
url="http://somemachine/basicauth/AddService.asmx";
soapaction="http://tempuri.org/Add";
webauthdomain="DefaultAuth";

rc = soapwebmeta("request", url, "response", soapaction, webauthdomain);
run;
```

関連項目:

関数:

SOAPWIPSERVICE 関数

WS セキュリティ認証を使用して SAS に登録されている Web サービスを呼び出します。認証情報は引数で指定します。

カテゴリ: Web サービス

注: この関数は SAS 環境ファイルを使用します。

構文

SOAPWIPSERVICE (IN, SERVICE <,*options*>)

必須引数

IN
ファイル参照名の文字値を指定します。IN は、SOAP 要求を含む XML データを入力するために使用されます。

SERVICE
サービスレジストリに格納されているとおりに、エンドポイントサービスのサービス名を指定します。

オプション引数

OUT
SOAP 応答の出力 XML が書き込まれるファイル参照名の文字値を指定します。

SOAPACTION
Web サービスで呼び出す SOAPAction 要素の文字値を指定します。

WSSUSERNAME
WS セキュリティのユーザー名である文字値を指定します。

WSSPASSWORD
WS セキュリティのパスワード(WSSUSERNAME のパスワード)である文字値を指定します。PROC PWENCODE によって生成されるエンコーディングがサポートされます。

ENVFILE
SAS 環境ファイルの場所である文字値を指定します。

ENVIRONMENT
使用する SAS 環境ファイルで定義されている環境の文字値を指定します。

MUSTUNDERSTAND
SOAP ヘッダーの mustUnderstand 属性設定の数値を指定します。

CONFIGFILE
主にタイムアウト値を設定する Spring 構成ファイルの文字値を指定します。

DEBUG
ログ出力のデバッグに使用するファイルのフルパスの文字値を指定します。

詳細

SAS 環境ファイル

サービス名はサービスレジストリに指定されます。SAS 環境ファイルはサービスレジストリと目的のサービスだけでなく、指定の認証情報を使ってセキュリティトークンを生成するセキュリティトークンサービスの場所を検索するのに使用されます。

サンプル

次の例では、DATA ステップで SOAPWIPSERVICE 関数を使用する方法を示します。

```
FILENAME request 'c:¥temp¥Request.xml';
FILENAME response 'c:¥temp¥Response.xml';

data _null;
service="ReportRepositoryService";
soapaction="http://www.test.com/xml/schema/test-svcs/reportrepository-9.3/
DirectoryServiceInterface/isDirectory";
envfile="http://somemachine.abc.xyz.com/schemas/test-environment.xml";
environment="test";
wssusername="user-name";
wsspassword="password";

rc=soapwipservice("REQUEST", service, "RESPONSE", soapaction, wssusername,
wsspassword, envfile, environment);
run;
```

このセクションでは SOAP 要求について説明します。

```
Request.xml:
<soapenv:Envelope xmlns:rep="http://www.test.com/xml/schema/test-svcs/
reportrepository-9.3"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
<soapenv:Header>
<Action xmlns="http://schemas.xmlsoap.org/ws/2004/08/addressing">http://
www.test.com/xml/schema/test-svcs/reportrepository-9.3/
DirectoryServiceInterface/isDirectory</Action>
</soapenv:Header>
<soapenv:Body>
<rep:isDirectoryDirectoryServiceInterfaceRequest>
<rep:dirPathUrl>SBIP://Foundation/Users/someuser/My Folder
</rep:dirPathUrl>
</rep:isDirectoryDirectoryServiceInterfaceRequest>
</soapenv:Body>
</soapenv:Envelope>

test-environments.xml:
```

```
<environments xmlns="http://www.test.com/xml/schema/test-environments-9.3
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.test.com/xml/schema/test-environments-9.3
http://www.test.com/xml/schema/test-environments-9.3/
test-environments-9.3.xsd">

<environment name="default" default="true">
<desc>Default Test Environment</desc>
<service-registry>http://machine1.abc.xyz.com:8080/TESTWIPServices/remote/
serviceRegistry
</service-registry>
</environment>

<environment name="test" default="false">
<desc>Environment for PROC SOAP testing</desc>
<service-registry>http://machine2.abc.xyz.com:8080/TESTWIPSoapServices/
Service Registry/serviceRegistry
</service-registry>
</environment>

</environments>
```

関連項目:

関数:

- “SOAPWS 関数” (849 ページ)
- “SOAPWSMETA 関数” (851 ページ)
- “SOAPWEB 関数” (841 ページ)
- “SOAPWEBMETA 関数” (843 ページ)
- “SOAPWIPSRS 関数” (847 ページ)

SOAPWIPSRS 関数

WS セキュリティ認証を使用して SAS に登録されている Web サービスを呼び出します。認証情報は引数で指定します。

カテゴリ: Web サービス

注: 指定される認証情報は、目的のサービスへの呼び出しに必要なセキュリティトークンを生成するのに使用されます。目的サービスの URL が指定されます。

セキュリティトークンサービスの検索方法を確認するためにレジストリサービスが直接呼び出されます。

構文

SOAPWIPSRS (IN, URL, SRSURL<,*options*>)

必須引数

IN
ファイル参照名の文字値を指定します。IN は、SOAP 要求を含む XML データを入力するために使用されます。

URL
Web サービスエンドポイントの URL の文字値を指定します。

SRSURL
システムレジストリサービスの URL である文字値を指定します。

オプション引数

OUT
SOAP 応答の出力 XML が書き込まれるファイル参照名の文字値を指定します。

SOAPACTION
Web サービスで呼び出す SOAPAction 要素の文字値を指定します。

WSSUSERNAME
WS セキュリティのユーザー名である文字値を指定します。

WSSPASSWORD
WS セキュリティのパスワード(WSSUSERNAME のパスワード)である文字値を指定します。PROC PWENCODE によって生成されるエンコーディングがサポートされます。

MUSTUNDERSTAND
SOAP ヘッダーの mustUnderstand 属性設定の数値を指定します。

CONFIGFILE
主にタイムアウト値を設定する Spring 構成ファイルの文字値を指定します。

DEBUG
ログ出力のデバッグに使用するファイルのフルパスの文字値を指定します。

サンプル

次の例では、DATA ステップで SOAPWIPSRS 関数を使用する方法を示します。

```
FILENAME request 'c:¥temp¥Request.xml';
FILENAME response 'c:¥temp¥Response.xml';

data _null_;
url="http://somemachine.abc.xyz.com:8080/TESTWIPSoapServices/services/
ReportRepositoryService";
soapaction="http://www.test.com/xml/schema/test-svcs/reportrepository-9.3/
DirectoryServiceInterface/isDirectory";
srsurl="http://somemachine.abc.xyz.com:8080/TESTWIPSoapServices/services/
ServiceRegistry";
WSSUSERNAME="user-name";
WSSPASSWORD="password";

rc = soapwipsrs("request", url, srsurl, "response", soapaction, wssusername,
wsspassword);
run;
```

このセクションでは SOAP 要求について説明します。

```
Request.xml:
<soapenv:Envelope xmlns:rep="http://www.test.com/xml/schema/test-svcs/
reportrepository-9.3"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
<soapenv:Header>
<Action
xmlns="http://schemas.xmlsoap.org/ws/2004/08/addressing">http://www.test.com/
xml/schema/test-svcs/reportrepository-9.3/DirectoryServiceInterface/
isDirectory</Action>
</soapenv:Header>
<soapenv:Body>
<rep:isDirectoryDirectoryServiceInterfaceRequest>
<rep:dirPathUrl>SBIP://Foundation/Users/someuser/My Folder
</rep:dirPathURL>
</rep:isDirectoryDirectoryServiceInterfaceRequest>
</soapenv:Body>
</soapenv:Envelope>
```

関連項目:

関数:

- “SOAPWS 関数” (849 ページ)
- “SOAPWSMETA 関数” (851 ページ)
- “SOAPWEB 関数” (841 ページ)
- “SOAPWEBMETA 関数” (843 ページ)
- “SOAPWIPSERVICE 関数” (845 ページ)

SOAPWS 関数

WS セキュリティ認証を使用して Web サービスを呼び出します。認証情報は引数で指定します。

カテゴリ: Web サービス

構文

SOAPWS (IN, URL <,*options*>)

必須引数

IN
ファイル参照名の文字値を指定します。IN は、SOAP 要求を含む XML データを入力するために使用されます。

URL
Web サービスエンドポイントの URL の文字値を指定します。

オプション引数

OUT
SOAP 応答の出力 XML が書き込まれるファイル参照名の文字値を指定します。

SOAPACTION
Web サービスで呼び出す SOAPAction 要素の文字値を指定します。

WSSUSERNAME
WS セキュリティのユーザー名である文字値を指定します。

WSSPASSWORD
WS セキュリティのパスワード(WSSUSERNAME のパスワード)である文字値を指定します。PROC PWENCODE によって生成されるエンコーディングがサポートされます。

MUSTUNDERSTAND
SOAP ヘッダーの mustUnderstand 属性設定の数値を指定します。

PROXYPORT
HTTP プロキシサーバーポートの数値を指定します。

PROXYHOST
HTTP プロキシサーバーホストの文字値を指定します。

PROXYUSERNAME
HTTP プロキシサーバーユーザー名の文字値を指定します。

PROXYPASSWORD
HTTP プロキシサーバーパスワードの文字値を指定します。PROC PWENCODE によって生成されるエンコーディングがサポートされます。

CONFIGFILE
主にタイムアウト値を設定する Spring 構成ファイルの文字値を指定します。

DEBUG
ログ出力のデバッグに使用するファイルのフルパスの文字値を指定します。

サンプル

次の例では、DATA ステップで SOAPWS 関数を使用する方法を示します。

```
FILENAME request 'C:¥temp¥Request.xml';
FILENAME response 'C:¥temp¥Response.xml';

data _null_;
url="http://somemachine.na.abc.com/SASBIWS/ProcSoapServices.asmx";
soapaction="http://tempuri.org/ProcSoapServices/copyintoout_xml_att";
WSSUSERNAME="sasuser";
WSSPASSWORD="password";

rc = soapws("request", url, "response", soapaction, wssusername,
wsspassword);
run;
```

関連項目:

関数:

- “SOAPWSMETA 関数” (851 ページ)
- “SOAPWEB 関数” (841 ページ)
- “SOAPWEBMETA 関数” (843 ページ)
- “SOAPWIPSERVICE 関数” (845 ページ)

- “SOAPWIPSRS 関数” (847 ページ)

SOAPWSMETA 関数

WS セキュリティ認証を使用して Web サービスを呼び出します。指定された認証ドメインの認証情報はメタデータから取得します。

カテゴリ: Web サービス

構文

SOAPWSMETA (IN, URL <,*options*>)

必須引数

IN
ファイル参照名の文字値を指定します。IN は、SOAP 要求を含む XML データを入力するために使用されます。

URL
Web サービスエンドポイントの URL の文字値を指定します。

オプション引数

OUT
SOAP 応答の出力 XML が書き込まれるファイル参照名の文字値を指定します。

SOAPACTION
Web サービスで呼び出す SOAPAction 要素の文字値を指定します。

WSSAUTHDOMAIN
WS セキュリティ認証で使用する認証ドメインである文字値を指定します。このドメインの認証情報を取得します。

MUSTUNDERSTAND
SOAP ヘッダーの mustUnderstand 属性設定の数値を指定します。

PROXYPORT
HTTP プロキシサーバーポートの数値を指定します。

PROXYHOST
HTTP プロキシサーバーホストの文字値を指定します。

PROXYUSERNAME
HTTP プロキシサーバーユーザー名の文字値を指定します。

PROXYPASSWORD
HTTP プロキシサーバーパスワードの文字値を指定します。PROC PWENCODE によって生成されるエンコーディングがサポートされます。

CONFIGFILE
主にタイムアウト値を設定する Spring 構成ファイルの文字値を指定します。

DEBUG
ログ出力のデバッグに使用するファイルのフルパスの文字値を指定します。

関連項目:

関数:

- “SOAPWS 関数” (849 ページ)
- “SOAPWEB 関数” (841 ページ)
- “SOAPWEBMETA 関数” (843 ページ)
- “SOAPWIPSERVICE 関数” (845 ページ)
- “SOAPWIPSRS 関数” (847 ページ)

SOUNDEX 関数

文字列をエンコードして検索しやすくします。

カテゴリ: 文字関数

制限事項: SOUNDEX アルゴリズムは英語中心に機能します。

I18N レベル 0 の関数は 1 バイト文字セット(SBCS)でのみ使用するように設計されています。

構文

SOUNDEX(*argument*)

必須引数

引数
文字定数、変数または式を指定します。

詳細

返される変数の長さ

DATA ステップで、まだ長さが割り当てられていない変数に SOUNDEX 関数から値が返される場合、その変数には 200 バイトの長さが割り当てられます。

基本

SOUNDEX 関数は、Margaret K. Odell と Robert C. Russel(米国特許 1261167(1918) および 1435663(1922))によって開発されたアルゴリズムに従って文字列をエンコードします。このアルゴリズムは Knuth の *The Art of Computer Programming, Volume 3* で説明されています (“リファレンス” (971 ページ)を参照)。
SOUNDEX アルゴリズムは英語中心に機能するため、英語以外の言語ではあまり有効ではありません。

SOUNDEX 関数は、次のステップに従ってエンコードした *argument* のコピーを返します。

1. *argument* の最初の文字を保持し、後続の文字を破棄します。

   A E H I O U W Y

2. 次の数値をこれらの文字クラスに割り当てます。

   - 1: B F P V

- 2: C G J K Q S X Z
- 3: D T
- 4: L
- 5: M N
- 6: R

3. 2 つ以上の隣接する文字がステップ 2 の同一クラスとなる場合、最初の文字を除いてすべてを破棄します (ここでの隣接は、文字を破棄する前の単語中の位置を意味します)。

Knuth によって説明されているアルゴリズムでは、末尾にゼロを付加して結果の長さが 4 となるよう切り捨てます。これらの処理は、他の SAS 関数で実行できます。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=soundex('Paul'); put x;	P4
word='amnesty'; x=soundex(word); put x;	A523

SPEDIS 関数

2 つの単語の一致尤度を調べて、2 単語間のスペルの違いをコストで表します。

カテゴリ: 文字関数

制限事項: I18N レベル 0 の関数は 1 バイト文字セット(SBCS)でのみ使用するように設計されています。

構文

SPEDIS(*query*,*keyword*)

必須引数

クエリ
一致尤度を調べるためのクエリ対象である単語を識別します。SPEDIS は値の比較前に末尾の空白を削除します。

キーワード
クエリの対象単語を指定します。SPEDIS は値の比較前に末尾の空白を削除します。

詳細

返される変数の長さ

DATA ステップで、まだ長さが割り当てられていない変数に SPEDIS 関数から値が返される場合、その変数には 200 バイトの長さが割り当てられます。

基本

SPEDIS はクエリとキーワードの間の距離を返します。通常、デフォルトコストでは負数でない 100 未満の値で、200 を超えることはありません。

SPEDIS は 2 単語間のスペルの違いを計算し、一連の演算によってキーワードをクエリ用の単語に変換するために必要な正規化コストとして表します。SPEDIS(*QUERY*, *KEYWORD*)は、SPEDIS(*KEYWORD*, *QUERY*)とは*異なります*。

キーワードをクエリに変換するための各演算のコストは、次の表のリストのとおりです。

操作	コスト	説明
match	0	変化なし
singlet	25	重複文字のいずれかを削除
doublet	50	文字を二重化
swap	50	連続する 2 文字の順序を逆転
truncate	50	末尾から文字を削除
append	35	末尾に文字を追加
delete	50	中央から文字を削除
insert	100	中央に文字を挿入
replace	100	中央の文字を置換
firstdel	100	最初の文字を削除
firstins	200	先頭に文字を挿入
firstrep	200	最初の文字を置換

距離はコストの合計をクエリの長さで除算して求めます。この比率が 1 より大きい場合、結果は切り捨てによって最も近い整数に丸められます。

比較

SPEDIS 関数は COMPLEV 関数と COMPGED 関数に似ていますが、特に長い文字列については COMPLEV と COMPGED の方がはるかに高速です。

サンプル

```
data words;
input Operation $ Query $ Keyword $;
Distance = spedis(query,keyword);
Cost = distance * length(query);
datalines;
match fuzzy fuzzy
singlet fuzy fuzzy
doublet fuuzzy fuzzy
swap fzuzy fuzzy
truncate fuzz fuzzy
append fuzzys fuzzy
delete fzzy fuzzy
insert fluzzy fuzzy
replace fizzy fuzzy
firstdel uzzy fuzzy
firstins pfuzzy fuzzy
firstrep wuzzy fuzzy
several floozy fuzzy
;
proc print data = words;
run;
```

画面 2.66 *SPEDIS 演算のコスト*

The SAS System

Obs	Operation	Query	Keyword	Distance	Cost
1	match	fuzzy	fuzzy	0	0
2	singlet	fuzy	fuzzy	6	24
3	doublet	fuuzzy	fuzzy	8	48
4	swap	fzuzy	fuzzy	10	50
5	truncate	fuzz	fuzzy	12	48
6	append	fuzzys	fuzzy	5	30
7	delete	fzzy	fuzzy	12	48
8	insert	fluzzy	fuzzy	16	96
9	replace	fizzy	fuzzy	20	100
10	firstdel	uzzy	fuzzy	25	100
11	firstins	pfuzzy	fuzzy	33	198
12	firstrep	wuzzy	fuzzy	40	200
13	several	floozy	fuzzy	50	300

関連項目:

関数:

- “COMPLEV 関数” (317 ページ)
- “COMPGED 関数” (311 ページ)

SQRT 関数

値の平方根を返します。

カテゴリ: 数学関数

構文

SQRT(*argument*)

必須引数

引数
数値の定数、変数または式を指定します。*argument* には負でない値にする必要があります。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=sqrt(36);	6
x=sqrt(25);	5
x=sqrt(4.4);	2.0976176963

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

参照項目: “SDF 関数” (832 ページ)

構文

SQUANTILE(*dist*, *probability*, *parm-1*,...,*parm-k*)

必須引数

dist
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
ベルヌーイ	BERNOULLI
ベータ	BETA
二項	BINOMIAL
コーシー	CAUCHY
χ2 乗	CHISQUARE
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
ポアソン	POISSON
T	T
Tweedie	TWEEDIE
一様	UNIFORM
逆ガウス(Wald)	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。

確率
ランダム変数の値を指定する、数値定数、変数または式です。

parm-1, … ,parm-k
特定の分布に対応する *shape*、*location* または *scale* パラメータです(省略可能)。

詳細

SQUANTILE 関数は各種の連続分布と離散分布の確率を計算します。詳細については、“詳細” (274 ページ)を参照してください。

サンプル

SQUANTILE 関数の例を次に示します。

```
data;
dist = 'logistic';
sdf = squantile(dist,1.e-20);
put sdf=;
p = sdf(dist,sdf);
put p= /* p will be 1.e-20 */;
run;
```

SAS は次の出力を口グに書き込みます。

```
sdf=46.05170186 p=1E-20
```

関連項目:

関数:

- “CDF 関数” (273 ページ)
- “LOGCDF 関数” (624 ページ)
- “LOGPDF 関数” (626 ページ)
- “LOGSDF 関数” (628 ページ)
- “PDF 関数” (703 ページ)
- “QUANTILE 関数” (778 ページ)
- “SDF 関数” (832 ページ)

STD 関数

非欠損引数の標準偏差を返します。

カテゴリ: 記述統計

構文

STD(*argument-1*,*argument-2*<,...*argument-n*>)

必須引数

引数

数値の定数、変数または式を指定します。少なくとも 2 つの非欠損引数が必要です。非欠損引数がない場合は、関数から欠損値が返されます。引数リストには OF で始まる変数のリストを含められます。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x1=std(2,6);	2.8284271247
x2=std(2,6,.);	2.8284271427
x3=std(2,4,6,3,1);	1.9235384062
x4=std(of x1-x3);	0.5224377453

STDERR 関数

非欠損引数の平均の標準誤差を返します。

カテゴリ: 記述統計

構文

STDERR(*argument-1*,*argument-2* <,…*argument-n*>)

必須引数

引数

数値の定数、変数または式を指定します。少なくとも 2 つの非欠損引数が必要です。非欠損引数がない場合は、関数から欠損値が返されます。引数リストには OF で始まる変数のリストを含められます。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x1=stderr(2,6);	2
x2=stderr(2,6,.);	2
x3=stderr(2,4,6,3,1);	0.8602325267
x4=stderr(of x1-x3);	0.3799224911

STFIPS 関数

州の郵便コードを FIPS コードに変換します。

カテゴリ: 州コード/郵便番号

構文

STFIPS(*postal-code*)

必須引数

postal-code
標準の州の郵便コードを表す 2 文字を含む文字式を指定します。大文字と小文字は混在できます。この関数は末尾の空白を無視しますが、式の先頭に空白があると エラーを返します。

詳細

STFIPS 関数は、2 文字の州の郵便コード(米国領の場合は世界中で利用されている GSA 地理コード)を対応する米国連邦情報処理標準(FIPS)の数字コードに変換します。

比較

STFIPS、STNAME および STNAMEL の各関数は同じ引数をとりますが、返す値は異なります。STFIPS は米国連邦情報処理標準(FIPS)の数字コードを返します。STNAME は大文字の州名を返します。STNAMEL は大文字と小文字混在の州名を返します。

サンプル

次の例では、STFIPS、STNAME および STNAMEL の各関数を使用した場合の違いを示します。

SAS ステートメント	結果
fips=stfips ('NC'); put fips;	37

SAS ステートメント	結果
state=stname('NC'); put state;	NORTH CAROLINA
state=stnamel('NC'); put state;	North Carolina

関連項目:

関数:

- “FIPNAME 関数” (466 ページ)
- “FIPNAMEL 関数” (467 ページ)
- “FIPSTATE 関数” (468 ページ)
- “STNAME 関数” (861 ページ)
- “STNAMEL 関数” (862 ページ)

STNAME 関数

州の郵便コードを大文字の州名に変換します。

カテゴリ: 州コード/郵便番号

構文

STNAME(*postal-code*)

必須引数

postal-code
標準の州の郵便コードを表す 2 文字を含む文字式を指定します。大文字と小文字は混在できます。この関数は末尾の空白を無視しますが、式の先頭に空白があるとエラーを返します。

詳細

STNAME 関数は、2 文字の州の郵便コード(米国領の場合は世界中で利用されている GSA 地理コード)を対応する大文字の州名に変換します。

注: バージョン 6 では、返される値の最大長は 200 文字です。バージョン 7 以降では、最大長は 20 文字です。

比較

STFIPS、STNAME および STNAMEL の各関数は同じ引数をとりますが、返す値は異なります。STFIPS は米国連邦情報処理標準(FIPS)の数字コードを返します。STNAME は大文字の州名を返します。STNAMEL は大文字と小文字混在の州名を返します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
fips=stfips ('NC'); put fips;	37
state=stname('NC'); put state;	NORTH CAROLINA
state=stnamel('NC'); put state;	North Carolina

関連項目:

関数:

- “FIPNAME 関数” (466 ページ)
- “FIPNAMEL 関数” (467 ページ)
- “FIPSTATE 関数” (468 ページ)
- “STFIPS 関数” (860 ページ)
- “STNAMEL 関数” (862 ページ)

STNAMEL 関数

州の郵便コードを大文字と小文字を使った州名に変換します。

カテゴリ: 州コード/郵便番号

構文

STNAMEL(*postal-code*)

必須引数

postal-code
標準の州の郵便コードを表す 2 文字を含む文字式を指定します。大文字と小文字は混在できます。この関数は末尾の空白を無視しますが、式の先頭に空白があるとエラーを返します。

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 文字の州の郵便コード(米国領の場合は世界中で利用されている GSA 地理コード)を対応する州名(大文字と小文字を使用)に変換します。

注: バージョン 6 では、返される値の最大長は 200 文字です。バージョン 7 以降では、最大長は 20 文字です。

比較

STFIPS、STNAME および STNAMEL の各関数は同じ引数をとりますが、返す値は異なります。STFIPS は米国連邦情報処理標準(FIPS)の数字コードを返します。STNAME は大文字の州名を返します。STNAMEL は大文字と小文字混在の州名を返します。

サンプル

次の例では、STFIPS、STNAME および STNAMEL の各関数を使用した場合の違いを示します。

SAS ステートメント	結果
fips=stfips ('NC'); put fips;	37
state=stname('NC'); put state;	NORTH CAROLINA
state=stnamel('NC'); put state;	North Carolina

関連項目:

関数:

- “FIPNAME 関数” (466 ページ)
- “FIPNAMEL 関数” (467 ページ)
- “FIPSTATE 関数” (468 ページ)
- “STFIPS 関数” (860 ページ)
- “STNAME 関数” (861 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: 文字関数

制限事項: I18N レベル 0 の関数は 1 バイト文字セット(SBCS)でのみ使用するように設計されています。

構文

STRIP(*string*)

必須引数

string
文字定数、変数または式です。

詳細

返される変数の長さ

DATA ステップで、まだ長さが割り当てられていない変数に STRIP 関数から値が返される場合、その変数には引数の長さに設定されます。

基本

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。

STRIP の結果を変数に割り当てる場合、受け取る変数の長さには影響しません。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: 連結演算子では先頭と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、TRIM 関数および TRIMN 関数の比較を次の表に示します。

- STRIP 関数と TRIMN 関数は空白の文字列に対して長さがゼロの文字列を返します。一方、TRIM 関数は空白 1 つを返します。
- 先頭に空白がない文字列では、STRIP 関数と TRIMN 関数は同一値を返します。
- 先頭に空白はないが、少なくとも 1 つの空白でない文字がある場合、STRIP 関数と TRIM 関数は同一値を返します。

注: **STRIP(*string*)は TRIMN(LEFT(*string*))**と同じ結果を返しますが、STRIP 関数の方が実行は高速です。

サンプル

次の例では、STRIP 関数を使用して先頭と末尾の空白を削除します。

```
data lengthn;
input string $char8.;
original = '*' || string || '*';
stripped = '*' || strip(string) || '*';
datalines;
abcd
abcd
abcd
abcdefgh
x y z
;
proc print data=lengthn;
run;
```

画面 2.67 STRIP 関数からの結果

The SAS System

Obs	string	original	stripped
1	abcd	*abcd *	*abcd*
2	abcd	* abcd *	*abcd*
3	abcd	* abcd*	*abcd*
4	abcdefgh	*abcdefgh*	*abcdefgh*
5	x y z	* x y z *	*x y z*

関連項目:

関数:

- “CAT 関数” (259 ページ)
- “CATS 関数” (265 ページ)
- “CATT 関数” (267 ページ)
- “CATX 関数” (270 ページ)
- “LEFT 関数” (601 ページ)
- “TRIM 関数” (899 ページ)
- “TRIMN 関数” (900 ページ)

SUBPAD 関数

必要に応じて空白埋め込みを使用し、指定した長さの部分文字列を返します。

カテゴリ: 文字関数

制限事項: 英語以外の言語を使用する場合、可能な限り I18N レベル 1 の関数の使用は避けてください。I18N レベル 1 の関数は、特定の環境下では 2 バイト文字セット(DBCS)または複数バイト文字セット(MBCS)エンコーディングを使用すると正常に動作しない場合があります。

構文

SUBPAD(*string*, *position* <, *length*>)

必須引数

string
文字定数、変数または式を指定します。

position
部分文字列の最初の文字の位置を指定する正の整数です。

オプション引数

長さ
部分文字列の長さを指定する負でない整数です。*length* を指定しない場合、SUBPAD 関数は文字列の指定した位置から最後までを含む部分文字列を返します。

詳細

DATA ステップで、まだ長さが割り当てられていない変数に SUBPAD 関数から値が返される場合、その変数には 200 バイトの長さが割り当てられます。

指定した部分文字列が文字列の長さを超える場合、結果には空白が埋め込まれます。

比較

SUBPAD 関数は SUBSTR 関数と似ていますが、次の点で異なります。

- SUBPAD 中の *length* の値がゼロの場合、SUBPAD は長さがゼロの文字列を返します。SUBSTR 中の *length* の値がゼロの場合、SUBSTR は次の処理を行います。
  - 第 3 引数が無効であることを示すメモがログに書き込まれます。
  - _ERROR_=1 を設定します。
  - 文字列の指定した位置から末尾までを含む部分文字列を返します。
- 指定した部分文字列が文字列の末尾を超える長さの場合、結果が要求した長さとなるよう、SUBPAD によって空白が埋め込まれます。指定した部分文字列が文字列の末尾を超える長さの場合、SUBSTR は次の処理を行います。
  - 第 3 引数が無効であることを示すメモがログに書き込まれます。
  - _ERROR_=1 を設定します。
  - 文字列の指定した位置から末尾までを含む部分文字列を返します。

関連項目:

関数:

- “SUBSTRN 関数” (869 ページ)

擬似 SUBSTR 関数(割り当ての左辺に用いた場合)

文字値の内容を置換します。

カテゴリ: 文字関数

制限事項: I18N レベル 0 の関数は 1 バイト文字セット(SBCS)でのみ使用するように設計されています。

ヒント: DBCS の相当関数は KSUBSTR と KSUBSTRB です。

構文

SUBSTR(*variable*, *position* <,*length*>)=*characters-to-replace*

必須引数

変数
文字変数を指定します。

position
開始文字の位置である数値定数、変数または式を指定します。

characters-to-replace
variable の内容を置換する文字定数、変数または式を指定します。

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

長さ
置き換え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。

制限事項: *length* は、*position* の後の *variable* に残る式の長さよりも大きくできません。

ヒント: *length* を省略すると、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 *variable* の値を置き換えます。

詳細

未宣言の変数を使用する場合、SUBSTR 関数のコンパイル時にデフォルトの長さとして 8 が割り当てられます。

割り当てステートメントの左側で SUBSTR 関数を使用すると、*variable* の値を右側の式で置き換えます。SUBSTR は、*position* で指定された開始文字から *length* 文字を置き換えます。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
a='KIDNAP'; substr(a,1,3)='CAT'; put a;	CATNAP
b=a; substr(b,4)='TY'; put b;	CATTY

関連項目:

関数:

- “SUBSTR 関数” (867 ページ)

SUBSTR 関数

引数から部分文字列を抽出します。

カテゴリ: 文字関数

制限事項: I18N レベル 0 の関数は 1 バイト文字セット(SBCS)でのみ使用するように設計されています。

ヒント: DBCS の相当関数は KSUBSTR と KSUBSTRB です。

構文

<*variable*=> **SUBSTR**(*string*,*position* <,*length*>)

必須引数

string
文字定数、変数または式を指定します。

position
開始文字の位置である数値定数、変数または式を指定します。

オプション引数

変数
有効な SAS 変数名を指定します。

長さ
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* 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、式の残りの部分が抽出されます。また、_ERROR_ が 1 に設定され、*length* 引数が無効であることを示すメモがログに出力されます。

ヒント: *length* を省略すると、式の残りの部分が抽出されます。

詳細

DATA ステップで、まだ長さが割り当てられていない変数に SUBSTR 関数から値が返される場合、その変数には第 1 引数の長さが割り当てられます。

SUBSTR 関数は、式内の *string* で指定された部分を返します。この部分は、*position* で指定した文字から始まり、*length* 文字の長さです。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
	----+----1----+----2

date='06MAY98';
month=substr(date,3,3);
year=substr(date,6,2);
put @1 month @5 year;
	MAY 98

関連項目:

関数:

- “SUBPAD 関数” (865 ページ)
- “擬似 SUBSTR 関数(割り当ての左辺に用いた場合)” (866 ページ)
- “SUBSTRN 関数” (869 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: 文字関数

制限事項: 英語以外の言語を使用する場合、可能な限り I18N レベル 1 の関数の使用は避けてください。I18N レベル 1 の関数は、特定の環境下では 2 バイト文字セット(DBCS)または複数バイト文字セット(MBCS)エンコーディングを使用すると正常に動作しない場合があります。

ヒント: KSUBSTR は同等の機能を備えています。

構文

SUBSTRN(*string*, *position* <, *length*>)

必須引数

string
文字または数値の定数、変数または式を指定します。

string が数値の場合、BEST32.形式の文字値に変換されます。先頭と末尾の空白は削除され、メッセージは SAS ログに送信されません。

position
部分文字列の最初の文字の位置を指定する整数です。

オプション引数

長さ
部分文字列の長さを指定する整数です。*length* を指定しない場合、SUBSTRN 関数は文字列の指定した位置から最後までを含む部分文字列を返します。

詳細

返される変数の長さ

DATA ステップで、まだ長さが割り当てられていない変数に SUBSTRN 関数から値が返される場合、その変数には第 1 引数の長さが割り当てられます。

基本

次の情報は SUBSTRN 関数についての説明です。

- SUBSTRN 関数は、*position* または *length* のいずれかが欠損値を持つ場合、長さがゼロの文字列を返します。

- 指定する位置が正でない場合、結果は最初に切り詰められ、結果の最初の文字は文字列の最初の文字となります。結果の長さはそれに応じて短縮されます。
- 指定した長さが文字列の末尾を超える長さの場合、結果は最後に切り詰められ、結果の最後の文字は文字列の最後の文字となります。

マクロでの SUBSTRN 関数の使用

%SYSFUNC マクロを使用して SUBSTRN を呼び出すと、マクロ処理で第 1 引数 (*string*)が解決されて、引数が文字か数値かの判定が行われます。第 1 引数をマクロ式として評価させないようにするには、第 1 引数でマクロ引用機能のいずれかを使用します。

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 の値が正でない	SUBSTRN	文字列の最初の文字で始まる結果を返します。
position の値が正でない	SUBSTR	• 第 2 引数が無効であることを示すメモがログに書き込まれます。 • _ERROR_ =1 を設定します。 • 文字列の指定した位置から末尾までを含む部分文字列を返します。
length の値が正でない	SUBSTRN	長さがゼロの結果を返します。
length の値が正でない	SUBSTR	• 第 3 引数が無効であることを示すメモがログに書き込まれます。 • _ERROR_ =1 を設定します。 • 文字列の指定した位置から末尾までを含む部分文字列を返します。
指定した部分文字列が文字列の末尾を超える	SUBSTRN	結果を切り捨てます。
指定した部分文字列が文字列の末尾を超える	SUBSTR	• 第 3 引数が無効であることを示すメモがログに書き込まれます。 • _ERROR_ =1 を設定します。 • 文字列の指定した位置から末尾までを含む部分文字列を返します。

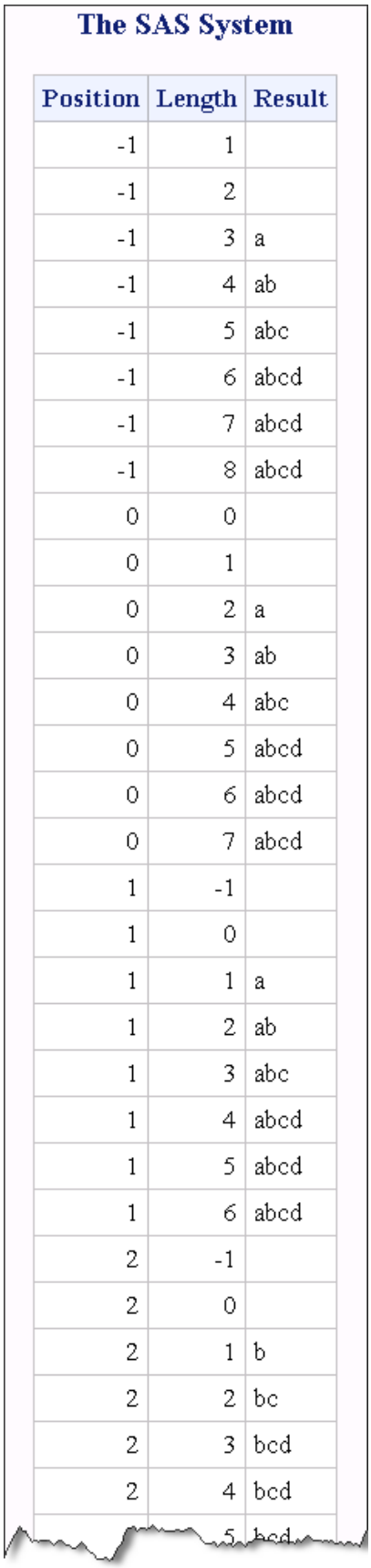
サンプル

サンプル 1: SUBSTRN 関数を使用した文字列の操作

次の例では、SUBSTRN 関数を使用して文字列を操作する方法を示します。

```
data test;
retain string "abcd";
drop string;
do Position = -1 to 6;
do Length = max(-1,-position) to 7-position;
Result = substrn(string, position, length);
output;
end;
end;
datalines;
abcd
;
proc print noobs data=test;
run;
```

画面 2.68 SUBSTRN 関数からの出力



The SAS System

Position	Length	Result
-1	1	
-1	2	
-1	3	a
-1	4	ab
-1	5	abc
-1	6	abcd
-1	7	abcd
-1	8	abcd
0	0	
0	1	
0	2	a
0	3	ab
0	4	abc
0	5	abcd
0	6	abcd
0	7	abcd
1	-1	
1	0	
1	1	a
1	2	ab
1	3	abc
1	4	abcd
1	5	abcd
1	6	abcd
2	-1	
2	0	
2	1	b
2	2	bc
2	3	bcd
2	4	bcd
2	5	bcd

サンプル 2: SUBSTR 関数と SUBSTRN 関数の比較

次の例では、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。

```
data _null_;
substr_result = "*" || substr(1234.5678,2,6) || "*";
put substr_result=;
substrn_result = "*" || substrn(1234.5678,2,6) || "*";
put substrn_result=;
run;
```

ログ 2.22　SUBSTR 関数と SUBSTRN 関数の結果

```
substr_result=* 1234*
substrn_result=*234.56*
```

関連項目:

関数:

- “SUBPAD 関数” (865 ページ)
- “擬似 SUBSTR 関数(割り当ての左辺に用いた場合)” (866 ページ)
- “SUBSTR 関数” (867 ページ)

SUM 関数

非欠損引数の合計を返します。

カテゴリ: 記述統計

構文

SUM(*argument*,*argument*,…)

必須引数

引数

数値の定数、変数または式を指定します。すべての引数が欠損値の場合、次のいずれかが発生します。

- 使用する引数が 1 つのみの場合、その引数の値が返されます。
- 使用する引数が 2 つ以上の場合、標準の欠損値(.)が返されます。

それ以外の場合、結果は欠損値の合計です。引数リストには OF で始まる変数のリストを含められます。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x1=sum(4,9,3,8);	24
x2=sum(4,9,3,8,.);	24

x1=9;
x2=39;
x3=sum(of x1-x2);
	48

x1=5; x2=6; x3=4; x4=9;
y1=34; y2=12; y3=74; y4=39;
result=sum(of x1-x4, of y1-y5);
	183

x1=55;
x2=35;
x3=6;
x4=sum(of x1-x3, 5);
	101

x1=7;
x2=7;
x5=sum(x1-x2);
	0

y1=20;
y2=30;
x6=sum(of y:);
	50

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

構文

SUMABS(*value-1* <,*value-2*…>)

必須引数

value
数値式を指定します。

詳細

すべての引数が欠損値の場合、結果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。

サンプル

サンプル 1: 絶対値の合計の計算

次の例では、非欠損引数の絶対値の合計を返します。

```
data _null_;
x=sumabs(1,.,-2,0,3,.q,-4);
put x=;
run;
```

SAS は次の出力をログに書き込みます。

```
x=10
```

サンプル 2: 変数リスト使用時の絶対値の合計の計算

次の例では、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。

```
data _null_;
x1 = 1;
x2 = 3;
x3 = 4;
x4 = 3;
x5 = 1;
x = sumabs(of x1-x5);
put x=;
run;
```

SAS は次の出力をログに書き込みます。

```
x=12
```

SYMEXIST 関数

マクロ変数が存在するかどうかを返します。

カテゴリ: マクロ

参照項目: “SYMEXIST Function” in *SAS Macro Language: Reference*

構文

SYMEXIST (*argument*)

必須引数

引数

次の項目のいずれかです。

- 二重引用符で囲まれたマクロ変数名(ただしアンパサンドを含まない)
- 引用符を使用せずに指定された、マクロ変数名を含む DATA ステップ文字変数の名前
- マクロ変数名を作成する文字式

詳細

SYMEXIST 関数は、囲んでいるローカルシンボルテーブルと、さらにグローバルシンボルテーブルを検索して、指定されたマクロ変数があるかどうかを確認し、マクロ変数が見つかった場合は **1**、見つからなかった場合は **0** を返します。

詳細については、“SYMEXIST Function” in *SAS Macro Language: Reference* を参照してください。

SYMGET 関数

DATA ステップの実行時にマクロ変数の値を返します。

カテゴリ: マクロ

構文

SYMGET(*argument*)

必須引数

引数
次の項目のいずれかです。

- 二重引用符で囲まれたマクロ変数名(ただしアンパサンドを含まない)
- 引用符を使用せずに指定された、マクロ変数名を含む DATA ステップ文字変数の名前
- マクロ変数名を作成する文字式

詳細

まだ長さが割り当てられていない変数に SYMGET 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。

SYMGET 関数は DATA ステップの実行時にマクロ変数の値を返します。 詳細については、“SYMGET Function” in *SAS Macro Language: Reference* を参照してください。

関連項目:

- *SAS マクロ言語: リファレンス*

CALL ルーチン:

- “CALL SYMPUT ルーチン” (252 ページ)

SYMGLOBL 関数

DATA ステップの実行時にマクロ変数が DATA ステップのグローバルスコープにあるかどうかを返します。

カテゴリ: マクロ

参照項目: “SYMGLOBL Function” in *SAS Macro Language: Reference*

構文

SYMGLOBL (*argument*)

必須引数

引数
次の項目のいずれかです。

- 二重引用符で囲まれたマクロ変数名(ただしアンパサンドを含まない)
- 引用符を使用せずに指定された、マクロ変数名を含む DATA ステップ文字変数の名前
- マクロ変数名を作成する文字式

詳細

SYMGLOBL 関数は、グローバルシンボルテーブルのみを検索して指定されたマクロ変数があるかどうかを確認し、マクロ変数が見つかった場合は 1、見つからなかった場合は 0 を返します。

詳細については、“SYMGLOBL Function” in *SAS Macro Language: Reference* を参照してください。

SYMLOCAL 関数

DATA ステップの実行時にマクロ変数が DATA ステップのローカルスコープにあるかどうかを返します。

カテゴリ: マクロ

参照項目: “SYMLOCAL Function” in *SAS Macro Language: Reference*

構文

SYMLOCAL (*argument*)

必須引数

引数
次の項目のいずれかです。

- 二重引用符で囲まれたマクロ変数名(ただしアンパサンドを含まない)
- 引用符を使用せずに指定された、マクロ変数名を含む DATA ステップ文字変数の名前
- マクロ変数名を作成する文字式

詳細

SYMLOCAL 関数は、囲んでいるローカルシンボルテーブルを検索して指定されたマクロ変数があるかどうかを確認し、マクロ変数が見つかった場合は 1、見つからなかった場合は 0 を返します。

詳細については、“SYMLOCAL Function” in *SAS Macro Language: Reference* を参照してください。

SYSEXIST 関数

環境内に動作環境変数が存在するかどうかを示す値を返します。

カテゴリ: SAS ファイル I/O 関数
特殊関数

構文

SYSEXIST (*argument*)

必須引数

引数
検証する動作環境変数名である文字変数を指定します。

詳細

SYSEXIST 関数は、動作環境変数があるかどうかを検索し、変数が見つかった場合は 1、見つからなかった場合は 0 を返します。

比較

SYSEXIST 関数は動作環境変数の存在を検証します。SYSGET 関数は動作環境変数の値を取得します。

サンプル

次の例では、環境内の動作環境変数として、HOME は有効な変数、TEST は無効な変数であるものとします。SYSEXIST は両方の値を検証します。

```
data _null_;
rc=sysexist("HOME");
put rc=;
rc=sysexist("TEST");
put rc=;
run;
```

SAS は次の出力を口グに書き込みます。

```
rc=1
rc=0
```

SYSEXIST が値 1 を返す場合、検証対象の変数は動作環境変数です。SYSEXIST が値 0 を返す場合、検証対象の変数は環境内の動作環境変数ではありません。

関連項目:

関数:

- “SYSGET 関数” (878 ページ)

SYSGET 関数

指定した動作環境変数の値を返します。

カテゴリ: 特殊関数

参照項目: “SYSGET Function: UNIX” in *SAS Companion for UNIX Environments*
“SYSGET Function: z/OS” in *SAS Companion for z/OS*

構文

SYSGET(*operating-environment-variable*)

必須引数

operating-environment-variable
動作環境変数名を値とする文字定数、変数または式を指定します。*operating-environment-variable* の大文字小文字は動作環境に格納されているのと同一である必要があります。SYSGET の引数の末尾の空白には意味があります。末尾の空白を削除するには TRIM 関数を使用します。

動作環境の情報
この関数の説明では、*operating-environment-variable* という用語は動作環境内の数値、文字または論理値を表す名前を指します。詳細については、動作環境に関する SAS のドキュメントを参照してください。

詳細

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。

サンプル

この例では、UNIX 環境の 2 つの環境変数の値を取得します。

```
data _null_;
length result $200;
input env_var $;
result=sysget(trim(env_var));
put env_var= result=;
datalines;
USER
PATH
;
```

ユーザー ABCDEF でこの DATA ステップを実行すると、次の行を表示します。

```
ENV_VAR=USER RESULT=abcdef
ENV_VAR=PATH RESULT=path-for-abcdef
```

関連項目:

関数:

- “ENVLEN 関数” (385 ページ)

SYSMSG 関数

最後のデータセットまたは外部ファイル関数の処理のエラーまたは警告メッセージを返します。

カテゴリ: SAS ファイル I/O 関数
外部ファイル

構文

SYSMSG()

詳細

SYSMSG は、データセットまたは外部ファイルアクセス関数でエラーが発生した場合に生成されるエラーメッセージまたは警告メッセージのテキストを返します。利用可能なエラーメッセージがない場合、返される値は空白です。内部的に保存されるエラーメッセージは SYSMSG への呼び出し後に空白にリセットされます。そのため、他のエラーが発生するまで、以降の SYSMSG への呼び出しは空白値を返します。

サンプル

この例では、SYSMSG を使用して、FETCH が次のオブザベーションを DDV(データセットデータベクトル)に書き込めない場合に生成されるエラーメッセージを SAS ログに書き込みます。レコードのフェッチに成功したときのみ、リターンコードは 0 です。

```
%let rc=%sysfunc(fetch(&dsid));
%if &rc ne 0 %then
%put %sysfunc(sysmsg());
```

関連項目:

関数:

- “FETCH 関数” (398 ページ)
- “SYSRC 関数” (884 ページ)

SYSPARM 関数

システムパラメータ文字列を返します。

カテゴリ: 特殊関数

構文

SYSPARM()

詳細

まだ長さが割り当てられていない変数に SYSPARM 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。

SYSPARM を使用すると、SAS の起動時に SYSPARM=システムオプションで指定されている文字列、または OPTIONS ステートメントで指定されている文字列にアクセスできます。

注: SYSPARM=システムオプションが指定されていない場合、SYSPARM 関数は長さがゼロの文字列を返します。

サンプル

この例では、SYSPARM=システムオプションと SYSPARM 関数を示します。

```
options sysparm='yes';
data a;
If sysparm()='yes' then
do;
...SAS Statements...
end;
run;
```

関連項目:

システムオプション:

- "SYSPARM= System Option" in *SAS Macro Language: Reference*

SYSPROCESSID 関数

現在のプロセスのプロセス ID を返します。

カテゴリ: 特殊関数

構文

SYSPROCESSID()

詳細

SYSPROCESSID 関数は、現在のプロセスの 32 文字からなる 16 進 ID を返します。この ID を SYSPROCESSNAME 関数に渡すと、現在のプロセスの名前を取得できます。

サンプル

サンプル 1: DATA ステップの使用

次の DATA ステップは、現在のプロセス ID を SAS ログに書き込みます。

```
data _null_;
id=sysprocessid();
```

```
put id;
run;
```

サンプル 2: SAS マクロ言語の使用

次の SAS マクロ言語コードは、現在のプロセス ID を SAS ログに書き込みます。

```
%let id=%sysfunc(sysprocessid());
%put &id;
```

関連項目:

関数:

- “SYSPROCESSNAME 関数” (882 ページ)

SYSPROCESSNAME 関数

指定プロセス ID に関連付けられているプロセス名、または現在のプロセスの名前を返します。

カテゴリ: 特殊関数

構文

SYSPROCESSNAME(<*process_id*>)

必須引数

process_id
32 文字からなる 16 進プロセス ID を指定します。

詳細

SYSPROCESSNAME 関数は、引数に指定するプロセス ID に関連付けられたプロセス名を返します。SYSPROCESSID 関数から返された値を SYSPROCESSNAME の引数として使用できます。引数を省略すると、SYSPROCESSNAME は現在のプロセスの名前を返します。

自動マクロ変数の SYSPROCESSID と SYSSTARTID に格納されている値も SYSPROCESSNAME の引数として使用できます。

サンプル

サンプル 1: DATA ステップでの引数を指定しない SYSPROCESSNAME の使用

次の DATA ステップは、現在のプロセス名を SAS ログに書き込みます。

```
data _null_;
name=sysprocessname();
put name;
run;
```

サンプル 2: SAS マクロ言語での引数を指定しない SYSPROCESSNAME の使用

次の SAS マクロ言語コードは、指定したプロセス ID に関連付けられているプロセス名を SAS ログに書き込みます。

```
%let id=&sysprocessid;
%let name=%sysfunc(sysprocessname(&id));
%put &name;
```

関連項目:

関数:

- “SYSPROCESSID 関数” (881 ページ)

SYSPROD 関数

製品のライセンスがあるかどうかを判断します。

カテゴリ: 特殊関数

構文

SYSPROD(*product-name*)

必須引数

product-name
SAS 製品名を値とする文字定数、変数または式を指定します。

要件 *Product-name* は製品またはソリューションの正確な正式名称にする必要があります。

詳細

SYSPROD 関数は、指定の SAS ソフトウェア製品がライセンスされている場合は 1 を返し、SAS 製品であるがシステムにライセンスされていない場合は 0 を返します。また、製品名が認識できない場合は-1 を返します。SYSPROD 関数が、製品がライセンスされていることを示している場合は、ライセンスの最終有効期限日を過ぎていないことを意味します。

SAS 製品は、製品のライセンスが有効期限切れとなってもシステム上に存在する場合があります。その場合、この製品にはアクセスできません。また、製品がライセンスされていても、インストールされていない場合があります。

SYSPROD は DATA ステップ、IML ステップまたは SCL プログラムで使用してください。

サンプル

これらの例では、特定の製品がライセンスされているかどうかを確認します。

- **x=sysprod('graph');**

  SAS/GRAPH ソフトウェアが現在ライセンスされている場合、SYSPROD は値1 を返します。SA
  を返します。

- **x=sysprod('abc');**

  ABC は有効な製品名ではないため、SYSPROD は値–1 を返します。

- **x=sysprod('base');**

  または

  x=sysprod('base sas');

  SYSPROD 関数を正常に実行するには Base 製品がライセンスされている必要があるため、SYSPROD は常に値 1 を返します。

SYSRC 関数

システムエラー番号を返します。

カテゴリ: SAS ファイル I/O 関数
外部ファイル

構文

SYSRC()

詳細

SYSRC は、データセット関数または外部ファイル関数のいずれかへの呼び出しで発生した最後のシステムエラーのエラー番号を返します。

サンプル

次の例では、FILEREF が存在しない場合にエラーメッセージを決定します。

```
%if %sysfunc(fileref(myfile)) ne 0 %then
%put %sysfunc(sysrc()) - %sysfunc(sysmsg());
```

関連項目:

関数:

- “FILEREF 関数” (407 ページ)
- “SYSMSG 関数” (880 ページ)

SYSTEM 関数

SAS セッションで動作環境コマンドを発行し、システムリターンコードを返します。

カテゴリ: 特殊関数

参照項目: “SYSTEM Function: z/OS” in *SAS Companion for z/OS*

構文

SYSTEM(*command*)

必須引数

command
引用符で囲まれたシステムコマンド(明示的な文字列)、値がシステムコマンドである式、または値が実行されるシステムコマンドである文字変数の名前のいずれかを指定します。

動作環境の情報
指定できる内容については、動作環境に関する SAS のドキュメントを参照してください。システムリターンコードは、ご利用の動作環境によって異なります。

制限事項: コマンドの長さは、末尾の空白を含めて 1024 文字以内にする必要があります。

比較

SYSTEM 関数は X ステートメント、X コマンドおよび CALL SYSTEM ルーチンに似ています。X ステートメント、X コマンドまたは%SYSEXEC マクロステートメントの方がオーバーヘッドが少ないため、これらの使用が望ましい場合がほとんどです。ただし、SYSTEM 関数は条件付きで実行でき、引数として式を使用できます。X ステートメントはグローバルステートメントであり、SAS での条件付きステートメントの有無に関係なく、DATA ステップがコンパイルされると実行されます。

サンプル

マクロ変数 SYSDAY が **Friday** の場合、ホストコマンド TIMEDATA を実行します。

```
data _null_;
if "&sysday"="Friday" then do;
rc=system("timedata");
end;
else rc=system("errorck");
run;
```

関連項目:

CALL ルーチン:

- “CALL SYSTEM ルーチン” (254 ページ)

ステートメント:

- “X ステートメント” (*SAS ステートメント: リファレンス*)

TAN 関数

正接を返します。

カテゴリ: 三角関数

構文

TAN(*argument*)

必須引数

引数
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。*argument* が非常に大きく、**mod(argument,pi)**の誤差がおよそ小数点以下 3 桁より小さい場合、TAN は欠損値を返します。

制限事項: π /2 の奇数倍とすることはできません。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=tan(0.5);	0.5463024898
x=tan(0);	0
x=tan(3.14159/3);	1.7320472695

TANH 関数

双曲線正接を返します。

カテゴリ: 双曲線関数

構文

TANH(*argument*)

必須引数

引数
数値の定数、変数または式を指定します。

詳細

TANH 関数は、次の式による引数の双曲線正接を返します。

$$\frac{\left(\varepsilon^{argument} - \varepsilon^{-argument}\right)}{\left(\varepsilon^{argument} + \varepsilon^{-argument}\right)}$$

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=tanh(0);	0
x=tanh(0.5);	0.4621171573
x=tanh(-0.5);	-0.462117157

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

構文

TIME()

サンプル

次のステートメントが午後 2:32 に実行された場合、14:32:00 に対応する SAS 時間値が CURRENT に割り当てられます。

```
current=time();
put current=time.;
```

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

構文

TIMEPART(*datetime*)

必須引数

datetime
SAS 日時値を表す数値定数、変数または式です。

サンプル

どの日でも午前 10:40:17 に次のステートメントが実行されると、10:40:17 に対応する SAS 値が TIME に割り当てられます。

```
datim=datetime();
time=timepart(datim);
```

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: 財務関数

構文

TIMEVALUE(*base-date*, *reference-date*, *reference-amount*, *compounding-interval*, *date-1*, *rate-1* <,*date-2*, *rate-2*,...>)

必須引数

base-date
SAS 日付です。返される値は、*base-date* での *reference-amount* の時間値です。

reference-date
SAS 日付です。*reference-date* は、*reference-amount* の日付です。

reference-amount
数値です。*reference-amount* は、*reference-date* の金額です。

compounding-interval
SAS 間隔です。*compounding-interval* は、複利間隔です。

date
SAS 日付です。各日付は利率とペアになります。*date* は、*rate* が有効になるタイミングです。

rate
数値のパーセントです。各利率は日付とペアになります。*rate* は、*date* に開始する利率です。

詳細

TIMEVALUE 関数には次の詳細が適用されます。

- 利率の値は–99 ~ 120 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。

サンプル

- 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。

```
amount_base1 = TIMEVALUE("01jan2001"d, "01jan2000"d, 1000,
"MONTH", "01jan2000"d, 10);
```

- 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。

```
amount_base2 = TIMEVALUE("01jan2001"d, "01jan2000"d, 1000,
"MONTH", 01jan2000"d, 10, "01jan2000"d, 20);
```

- 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。

```
amount_base3 = TIMEVALUE("01jan2001"d, "01jan2000"d, 1000,
"MONTH", "01jul2000"d, 20, "01jan2000"d, 10);
```

TINV 関数

t 分布から分位点を返します。

カテゴリ: 分位点

構文

TINV(*p*,*df* <,*nc*>)

必須引数

p
数値の確率です。
範囲: $0<p<1$

df
数値の自由度パラメータです。
範囲: $df > 0$

オプション引数

nc
数値の非心度パラメータです(省略可能)。

詳細

TINV 関数は、自由度が *df*、非心度パラメータが *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 *df* を指定できます。このパラメータ *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。

注: TINV は PROBT 関数の逆数です。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=tinv(.95,2);	2.9199855804
x=tinv(.95,2.5,3);	11.033833625

関連項目:

関数:

- “QUANTILE 関数” (778 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の値を返します。

カテゴリ: 数学関数

構文

TNONCT(*x*,*df*,*prob*)

必須引数

x
数値のランダム変数です。

df
数値の自由度パラメータです。
範囲: $df > 0$

prob
確率です。
範囲: $0 < prob < 1$

詳細

TNONCT 関数は、パラメータが *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を返します。式の平方根 *nc* を求めるために二ュートン型のアルゴリズムが使用されます。

$$P_t(x \mid df, nc) - prob = 0$$

前述の式には次の関係が適用されます。

$$P_t(x \mid df, nc) = \frac{1}{\Gamma\left(\frac{df}{2}\right)} \int_0^{\infty} v^{\frac{df}{2}-1} \varepsilon^{-v} \int_{-\infty}^{x\sqrt{\frac{2v}{df}}} \varepsilon^{-\frac{(u-nc)^2}{2}} dudv$$

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。

サンプル

次の例では、*t* 分布から非心度パラメータを計算します。

```
data work;
x=2;
df=4;
do nc=1 to 3 by .5;
prob=probt(x,df,nc);
ncc=tnonct(x,df,prob);
output;
end;
run;
proc print;
run;
```

画面 2.69 *t 分布からの非心度パラメータの計算*

The SAS System

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

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

別名: DATE

構文

TODAY()

詳細

TODAY 関数は、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。

サンプル

TODAY 関数の実用的なステートメントを次に示します。

```
data _null_;
tday=today();
if (tday-datedue)> 15 then
do;
put 'As of ' tday date9. ' Account #'
account 'is more than 15 days overdue.';
end;
run;
```

TRANSLATE 関数

文字列に含まれる特定の文字を置き換えます。

カテゴリ: 文字関数

制限事項: I18N レベル 0 の関数は 1 バイト文字セット(SBCS)でのみ使用するように設計されています。

ヒント: DBCS に相当する関数は、KTRANSLATE です。

参照項目: “TRANSLATE Function: Windows” in *SAS Companion for Windows*

“TRANSLATE Function: UNIX” in *SAS Companion for UNIX Environments*

“TRANSLATE Function: z/OS” in *SAS Companion for z/OS*

構文

TRANSLATE(*source*,*to-1*,*from-1*<,…*to-n*,*from-n*>)

必須引数

ソース
元の文字列が含まれる文字定数、変数または式を指定します。

to
TRANSLATE で代替文字として使用する文字を指定します。

from
TRANSLATE で置換する文字を指定します。

動作環境の情報
一部の動作環境では、*to* と *from* の引数ペアが必要です。他の動作環境では、照合順序のセグメントが NULL の *from* 引数に置き換わります。詳細については、動作環境に関する SAS のドキュメントを参照してください。

操作: *to* および *from* の値は、文字対文字の関係で対応します。TRANSLATE は、*from* の最初の文字を *to* の最初の文字に変換するという具合に処理を続けます。*to* の文字が *from* の文字より少ない場合、TRANSLATE は *from* の余った文字を空白に変更します。*to* の文字が *from* の文字より多い場合、TRANSLATE は *to* の余った文字を無視します。

詳細

DATA ステップで、まだ長さが割り当てられていない変数に TRANSLATE 関数から値が返される場合、その変数には第 1 引数の長さが割り当てられます。

TRANSLATE が受け入れる *to* および *from* の引数ペアの最大数は、SAS を実行するときの動作環境に応じて異なります。短い引数のペアを数個使用する場合と、長い引数のペアを少数使用する場合とでは、機能的な違いはありません。

比較

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。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=translate('XYZW','AB','VW'); put x;	XYZB

関連項目:

関数:

- “TRANWRD 関数” (895 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削除します。

カテゴリ: 文字関数

構文

TRANSTRN(*source*,*target*,*replacement*)

必須引数

ソース
変換する文字定数、変数または式を指定します。

target
source 内で検索する文字定数、変数または式を指定します。

要件 *target* 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。

置換
target を置き換える文字定数、変数または式を指定します。

詳細

返される変数の長さ

DATA ステップで、まだ長さが割り当てられていない変数に TRANSTRN 関数から値が返される場合、その変数には 200 バイトの長さが割り当てられます。TRANSTRN を呼び出す前に、LENGTH ステートメントを使用して値の長さを変更できます。

基本

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削除します。TRANSTRN 関数では、*target* 文字列と *replacement* 文字列の末尾の空白は削除されません。すべての *target* を削除するには、*replacement* 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、target 文字列内の文字が source 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。

サンプル

サンプル 1: 検出単語のすべての出現個所の置換

ステートメントとそれらの値による結果を次に示します。

```
 name=transtrn(name, "Mrs.", "Ms.");
name=transtrn(name, "Miss", "Ms.");
put name;
```

値	結果
Mrs. Joan Smith	Ms. Joan Smith
Miss Alice Cooper	Ms. Alice Cooper

サンプル 2: 検索文字列からのブランクの削除

この例では、target 文字列に空白が含まれるため、TRANSTRN 関数は source 文字列を置き換えません。

```
data list;
input salelist $;
length target $10 replacement $3;
target='FISH';
replacement='NIP';
```

```
salelist=transtrn(salelist,target,replacement);
put salelist;
datalines;
CATFISH
;
```

LENGTH ステートメントは *target* を 10 の長さまで空白で埋め込むため、TRANSTRN 関数は SALELIST 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。次の例では、*target* に TRIM 関数を使用します。

```
salelist=transtrn(salelist,trim(target),replacement);
put salelist;
```

これで次の行が SAS ログに書き込まれるようになります。

```
CATNIP
```

サンプル 3: TRANSTRN 関数の第 3 引数の長さがゼロの場合

次の例では、第 3 引数 *replacement* の長さがゼロの場合の TRANSTRN 関数の結果を示します。DATA ステップで 2 つの引用符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。次の例では、*string1* の結果は *string2* の結果とは異なります。

```
data _null_;
string1='*' || transtrn('abcxabc', 'abc', trimn('')) || '*';
put string1=;
string2='*' || transtrn('abcxabc', 'abc', '') || '*';
put string2=;
run;
```

SAS は次の出力を口グに書き込みます。

ログ 2.23 *TRANSTRN の第 3 引数の長さがゼロの場合の出力*

```
string1=*x*
string2=* x *
```

関連項目:

関数:

- “TRANSLATE 関数” (892 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: 文字関数

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

TRANWRD(*source*,*target*,*replacement*)

必須引数

ソース

変換する文字定数、変数または式を指定します。

target

source 内で検索する文字定数、変数または式を指定します。

要件 *target* 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。

置換

target を置き換える文字定数、変数または式を指定します。置換文字列の長さが 0 場合、TRANWRD ではかわりに 1 つの空白を使用します。

詳細

返される変数の長さ

DATA ステップで、まだ長さが割り当てられていない変数に TRANWRD 関数から値が返される場合、その変数には 200 バイトの長さが割り当てられます。TRANWRD を呼び出す前に、LENGTH ステートメントを使用して値の長さを変更できます。

基本

TRANWRD 関数は、*source* の値を結果文字列にコピーする一方で、*target* の値と等しい *source* 内のすべての重複しない部分文字列を検索します。これらの部分文字列は結果から削除され、*replacement* の値がその場所にコピーされます。TRANWRD 関数では、*target* 文字列または *replacement* 文字列の末尾の空白は削除されません。

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、target 文字列内の文字が source 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。

サンプル

サンプル 1: 検出単語のすべての出現個所の置換

ステートメントとそれらの値による結果を次に示します。

```
 name=tranwrd(name, "Mrs.", "Ms.");
name=tranwrd(name, "Miss", "Ms.");
put name;
```

値	結果
Mrs. Joan Smith	Ms. Joan Smith
Miss Alice Cooper	Ms. Alice Cooper

サンプル 2: 検索文字列からの空白の削除

この例では、target 文字列に空白が含まれるため、TRANWRD 関数は source 文字列を置き換えません。

```
data list;
input salelist $;
length target $10 replacement $3;
target='FISH';
replacement='NIP';
salelist=tranwrd(salelist,target,replacement);
put salelist;
datalines;
CATFISH
;
```

LENGTH ステートメントは *target* を 10 の長さまで空白で埋め込むため、TRANWRD 関数は SALELIST 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。次の例では、*target* に TRIM 関数を使用します。

```
salelist=tranwrd(salelist,trim(target),replacement);
put salelist;
```

これで次の行が SAS ログに書き込まれるようになります。

```
CATNIP
```

サンプル 3: TRANWRD 関数の第 3 引数の長さがゼロの場合

次の例では、第 3 引数 *replacement* の長さがゼロの場合の TRANWRD 関数の結果を示します。この場合、TRANWRD は 1 つの空白を使用します。DATA ステップで 2 つの連続する引用符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。この例では、*string1* の結果と *string2* の結果は同じです。

```
data _null_;
string1='*' || tranwrd('abcxabc', 'abc', trimn('')) || '*';
put string1=;
string2='*' || tranwrd('abcxabc', 'abc', '') || '*';
put string2=;
run;
```

ログ 2.24 *TRANWRD の第 3 引数の長さがゼロの場合の出力*

```
string1=* x *
string2=* x *
```

サンプル 4: カンマの繰り返しの削除

TRANWRD 関数を使用して、テキスト内のカンマの繰り返しを削除して 1 つのカンマに置き換えることができます。次の例では、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。

```
data _null_;
mytxt='If you exercise your power to vote,,,then your opinion will be heard,,';
newtext=tranwrd(mytxt, ',,,', ',');
newtext2=tranwrd(newtext, ',,' , '.');
put // mytxt= / newtext= / newtext2=;
run;
```

ログ 2.25 *カンマの繰り返しの削除からの出力*

```
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” (892 ページ)

TRIGAMMA 関数

トリガンマ関数の値を返します。

カテゴリ: 数学関数

構文

TRIGAMMA(*argument*)

必須引数

引数

数値の定数、変数または式を指定します。

制限事項: 正でない整数は無効です。

詳細

TRIGAMMA 関数は、DIGAMMA 関数の導関数を返します。*argument*>0 の場合、TRIGAMMA 関数は LGAMMA 関数の第 2 導関数です。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=trigamma(3);	0.3949340668

TRIM 関数

文字列から末尾の空白を取り除きます。文字列が欠損値の場合は、1 つの空白を返します。

カテゴリ: 文字関数

制限事項: I18N レベル 0 の関数は 1 バイト文字セット(SBCS)でのみ使用するように設計されています。

ヒント: DBCS に相当する関数は、"KTRIM 関数" (*SAS 各国語サポート(NLS): リファレンスガイド*) です。

構文

TRIM(*argument*)

必須引数

引数
文字定数、変数または式を指定します。

詳細

返される変数の長さ

DATA ステップで、まだ長さが割り当てられていない変数に TRIM 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。

基本

TRIM では、文字の引数をコピーし、末尾の空白を削除して、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 1 つの空白を返します。連結では末尾の空白は削除されないため、TRIM は連結する場合に便利です。

TRIM の結果を変数に割り当てる場合、受け取る変数の長さには影響しません。調整された値が受け取る変数の長さより短い場合、その変数への割り当て時に空白で値が埋め込まれます。

比較

TRIM 関数と TRIMN 関数は似ています。TRIM は、空白の文字列の場合は 1 つの空白を返します。TRIMN は、空白の文字列の場合は長さがゼロの文字列を返します。

サンプル

サンプル 1: 末尾の空白の削除

ステートメントとデータ行による結果を次に示します。

```
data test;
input part1 $ 1-10 part2 $ 11-20;
hasblank=part1||part2;
noblank=trim(part1)||part2;
put hasblank;
```

```
put noblank;
datalines;
```

データ行	結果
	----+----1----+----2
apple sauce	apple sauce
	applesauce

サンプル 2: 空白の文字式の連結

SAS ステートメント	結果
x="A"||trim(" ")||"B"; put x;	A B
x=" "; y=">"||trim(x)||"<"; put y;	><

関連項目:

関数:

- “COMPRESS 関数” (321 ページ)
- “LEFT 関数” (601 ページ)
- “RIGHT 関数” (808 ページ)
- “TRIMN 関数” (900 ページ)

TRIMN 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: 文字関数

制限事項: I18N レベル 0 の関数は 1 バイト文字セット(SBCS)でのみ使用するように設計されています。

構文

TRIMN(*argument*)

必須引数

引数
文字定数、変数または式を指定します。

詳細

返される変数の長さ

DATA ステップで、まだ長さが割り当てられていない変数に TRIMN 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。

TRIMN の結果を変数に割り当てる場合、受け取る変数の長さには影響しません。調整された値が受け取る変数の長さより短い場合、その変数への割り当て時に空白で値が埋め込まれます。

基本

TRIMN では、文字の引数をコピーし、すべての末尾の空白を削除して、その結果調整された引数を返します。引数が空白の場合、TRIMN 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削除されないため、TRIMN は連結する場合に便利です。

比較

TRIMN 関数と TRIM 関数は似ています。TRIMN は、空白の文字列の場合は長さがゼロの文字列を返します。TRIM は、空白の文字列の場合は 1 つの空白を返します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x="A"||trimn(""")||"B"; put x;	AB
x=" "; z=">"||trimn(x)||"<"; put z;	><

関連項目:

関数:

- “COMPRESS 関数” (321 ページ)
- “LEFT 関数” (601 ページ)
- “RIGHT 関数” (808 ページ)
- “TRIM 関数” (899 ページ)

TRUNC 関数

数値を指定したバイト数で切り捨てます。

カテゴリ: 切り捨て関数

構文

TRUNC(*number*,*length*)

必須引数

number
数値の定数、変数または式を指定します。

長さ
整数を指定します。

詳細

TRUNC 関数では、完全な長さの *number*(倍精度浮動小数点型として保存)を *length* 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。

サンプル

```
data test;
length x 3;
x=1/5;
run;
data test2;
set test;
if x ne 1/5 then
put 'x ne 1/5';
if x eq trunc(1/5,3) then
put 'x eq trunc(1/5,3)';
run;
```

変数 X は 3 の長さで保存されるため、前述の各条件式は真になります。

UNIFORM 関数

一様分布からランダム変量を返します。

カテゴリ: 乱数

別名: RANUNI

参照項目: “RANUNI 関数”(802 ページ)

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: 文字関数

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

UPCASE(*argument*)

必須引数

引数
文字定数、変数または式を指定します。

詳細

DATA ステップで、まだ長さが割り当てられていない変数に UPCASE 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。

UPCASE 関数は、文字引数をコピーし、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。

UPCASE 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を参照)に直接依存し、 ENCODING システムオプションおよび LOCALE システムオプションに間接的に依存します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
name=upcase('John B. Smith'); put name;	JOHN B. SMITH

関連項目:

関数:

- “LOWCASE 関数” (630 ページ)
- “PROPCASE 関数” (752 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

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

URLDECODE(*argument*)

必須引数

引数
文字定数、変数または式を指定します。

詳細

DATA ステップで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。

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る 3 つの文字の後に 2 つの 16 進文字が続く文字列。指定した 16 進値を含む 1 文字で置き換えられます。

argument 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。SAS セッションエンコーディングを使用して *argument* をデコードするには、システムオプション URLENCODING=SESSION を設定します。UTF-8 エンコーディングを使用して *argument* をデコードするには、システムオプション URLENCODING=UTF8 を設定します。

動作環境の情報
EBCDIC を使用する動作環境では、SAS がエスケープシーケンスを認識すると追加の変換ステップが実行されます。指定した文字は ASCII エンコーディングとみなされます。SAS はトランスポートからローカルへの変換テーブルを使用して、EBCDIC を使用する動作環境でこの文字を EBCDIC 文字に変換します。 詳細については、 TRANTAB オプション を参照してください。

サンプル

次の SAS ステートメントでは、SAS セッションデコーディングを使用して結果を生成します。

SAS ステートメント	結果
x1=urldecode ('abc+def'); put x1;	abc def
x2=urldecode ('why%3F'); put x2;	why?
x3=urldecode ('%41%42%43%23%31'); put x3;	ABC#1

関連項目:

関数:

- “URLENCODE 関数” (905 ページ)

システムオプション:

- “URLENCODING= System Option” in *SAS System Options: Reference*

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

制限事項: I18N レベル 2 の関数は SBCS、DBCS および MBCS(UTF8)で使用するために設計されています。

構文

URLENCODE(*argument*)

必須引数

引数
文字定数、変数または式を指定します。

詳細

DATA ステップで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 URLENCODE 関数から値が返される場合、その変数には 200 バイトの長さが割り当てられます。

基本

argument 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。SAS セッションエンコーディングを使用して *argument* をエンコードするには、システムオプション URLENCODING=SESSION を設定します。UTF-8 エンコーディングを使用して *argument* をエンコードするには、システムオプション URLENCODING=UTF8 を設定します。

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。

サンプル

次の SAS ステートメントでは、SAS セッションエンコーディングを使用して結果を生成します。

SAS ステートメント	結果
x1=urlencode ('abc def'); put x1;	abc%20def
x2=urlencode ('why?'); put x2;	why%3F
x3=urlencode ('ABC#1'); put x3;	ABC%231

関連項目:

関数:

- “URLDECODE 関数” (903 ページ)

システムオプション:

- “URLENCODING= System Option” in *SAS System Options: Reference*

USS 関数

非欠損引数の無修正平方和を返します。

カテゴリ: 記述統計

構文

USS(*argument-1*<,...*argument-n*>)

必須引数

引数

数値の定数、変数または式を指定します。少なくとも 1 つの非欠損引数が必要です。非欠損引数がない場合は、関数から欠損値が返されます。引数が 1 つ以上ある場合、引数リストには OF で始まる変数のリストを含められます。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x1=uss(4,2,3.5,6);	68.25
x2=uss(4,2,3.5,6,.);	68.25
x3=uss(of x1-x2);	9316.125

UUIDGEN 関数

UUID (Universal Unique Identifier)の短い形式またはバイナリ形式を返します。

カテゴリ: 特殊関数

構文

UUIDGEN(<*max-warnings*<,*binary-result*>>)

必須引数

max-warnings
この関数で口グに書き込まれる警告の最大数を表す整数値を指定します。

デフォルト: 1

オプション引数

binary-result
この関数でバイナリの結果を返すかどうかを示す整数値を指定します。非ゼ口値はバイナリの結果が返されることを示します。ゼ口値は文字の結果が返されることを示します。

デフォルト: 0

詳細

DATA ステップで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 UUIDGEN 関数から値が返される場合、その変数には 200 バイトの長さが割り当てられます。

基本

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
```

バイナリの結果は 16 バイトです。

関連項目:

Universal Unique Identifier:

- “汎用一意識別子(UUID)” (*SAS 言語リファレンス: 解説編* 39 章)

VAR 関数

非欠損引数の分散を返します。

カテゴリ: 記述統計

構文

VAR(*argument-1*,*argument-2*<,...*argument-n*>)

必須引数

引数
数値の定数、変数または式を指定します。少なくとも 2 つの非欠損引数が必要です。非欠損引数がない場合は、関数から欠損値が返されます。引数リストには OF で始まる変数のリストを含められます。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x1=var(4,2,3.5,6);	2.7291666667
x2=var(4,6,.);	2
x3=var(of x1-x2);	0.2658420139

VARFMT 関数

SAS データセット変数に割り当てられた出力形式を返します。

カテゴリ: SAS ファイル I/O 関数

構文

VARFMT(*data-set-id*,*var-num*)

必須引数

data-set-id
OPEN 関数が返すデータセット識別子を指定します。

var-num
SAS データセット内の変数の位置番号を指定します。

ヒント:
この番号は、CONTENTS プロシジャによって生成されるリスト内で変数の横に示されます。

VARNUM 関数はこの番号を返します。

詳細

変数に出力形式が割り当てられていない場合、空白の文字列が返されます。

サンプル

サンプル 1: VARFMT を使用した変数 NAME の出力形式の取得

この例では、SAS データセット MYDATA 内の変数 NAME の出力形式を取得します。

```
%let dsid=%sysfunc(open(mydata,i));
%if &dsid %then
%do;
%let fmt=%sysfunc(varfmt(&dsid,
%sysfunc(varnum
(&dsid,NAME))));
%let rc=%sysfunc(close(&dsid));
%end;
```

サンプル 2: VARFMT を使用したデータセット内のすべての数値変数の出力形式の取得

この例は、SAS データセット MYDATA 内に各数値変数の名前と書式化された内容を含むデータセットを作成します。

```
data vars;
length name $ 8 content $ 12;
drop dsid i num rc fmt;
dsid=open("mydata","i");
num=attrn(dsid,"nvars");
do while (fetch(dsid)=0);
do i=1 to num;
name=varname(dsid,i);
if (vartype(dsid,i)='N') then do;
fmt=varfmt(dsid,i);
if fmt='' then fmt="BEST12.";
content=putc(putn(getvarn
(dsid,i),fmt),"$char12.");
output;
end;
end;
end;
rc=close(dsid);
run;
```

関連項目:

関数:

- “VARINFMT 関数” (910 ページ)
- “VARNUM 関数” (914 ページ)

VARINFMT 関数

SAS データセット変数に割り当てられた入力形式を返します。

カテゴリ: SAS ファイル I/O 関数

構文

VARINFMT(*data-set-id*,*var-num*)

必須引数

data-set-id
OPEN 関数が返すデータセット識別子を指定します。

var-num
SAS データセット内の変数の位置番号を指定します。

ヒント:
この番号は、CONTENTS プロシジャによって生成されるリスト内で変数の横に示されます。

VARNUM 関数はこの番号を返します。

詳細

変数に入力形式が割り当てられていない場合、空白の文字列が返されます。

サンプル

サンプル 1: VARINFMT を使用した変数 NAME の入力形式の取得

この例では、SAS データセット MYDATA 内の変数 NAME の入力形式を取得します。

```
%let dsid=%sysfunc(open(mydata,i));
%if &dsid %then
%do;
%let fmt=%sysfunc(varinfmt(&dsid,
%sysfunc(varnum
(&dsid,NAME))));
%let rc=%sysfunc(close(&dsid));
%end;
```

サンプル 2: VARINFMT を使用したデータセット内のすべての変数の入力形式の取得

この例では、MYDATA 内の変数の名前と入力形式を含むデータセットを作成します。

```
data vars;
length name $ 8 informat $ 10 ;
drop dsid i num rc;
dsid=open("mydata","i");
num=attrn(dsid,"nvars");
do i=1 to num;
```

```
name=varname(dsid,i);
informat=varinfmt(dsid,i);
output;
end;
rc=close(dsid);
run;
```

関連項目:

関数:

- “OPEN 関数” (697 ページ)
- “VARFMT 関数” (908 ページ)
- “VARNUM 関数” (914 ページ)

VARLABEL 関数

SAS データセット変数に割り当てられたラベルを返します。

カテゴリ: SAS ファイル I/O 関数

構文

VARLABEL(*data-set-id*,*var-num*)

必須引数

data-set-id
OPEN 関数が返すデータセット識別子を指定します。

var-num
SAS データセット内の変数の位置番号を指定します。

ヒント:
この番号は、CONTENTS プロシジャによって生成されるリスト内で変数の横に示されます。
VARNUM 関数はこの番号を返します。

詳細

変数にラベルが割り当てられていない場合、空白の文字列が返されます。

比較

VLABEL は、指定した変数に関連付けられたラベルを返します。

サンプル

この例では、SAS データセット MYDATA 内の変数 NAME のラベルを取得します。

サンプルコード 2.1 *変数 NAME のラベルを取得する*

```
%let dsid=%sysfunc(open(mydata,i)); %if &dsid %then %do; %let fmt=%sysfunc(varlabel(&dsid, %sysfunc(varnum (&dsid,NAME)))); %le
```

関連項目:

関数:

- “OPEN 関数” (697 ページ)
- “VARNUM 関数” (914 ページ)

VARLEN 関数

SAS データセット変数の長さを返します。

カテゴリ: SAS ファイル I/O 関数

構文

VARLEN(*data-set-id*,*var-num*)

必須引数

data-set-id
OPEN 関数が返すデータセット識別子を指定します。

var-num
SAS データセット内の変数の位置番号を指定します。

ヒント:
この番号は、CONTENTS プロシジャによって生成されるリスト内で変数の横に示されます。

VARNUM 関数はこの番号を返します。

詳細

VLENGTH は、指定した変数のコンパイル時の(割り当てられた)サイズを返します。

サンプル

- この例では、SAS データセット MYDATA 内の変数 ADDRESS の長さを取得します。

```
%let dsid=%sysfunc(open(mydata,i));
%if &dsid %then
%do;
%let len=%sysfunc(varlen(&dsid,
%sysfunc(varnum
(&dsid,ADDRESS))));
%let rc=%sysfunc(close(&dsid));
%end;
```

- この例では、MYDATA 内の変数の名前、種類および長さを含むデータセットを作成します。

```
data vars;
length name $ 8 type $ 1;
drop dsid i num rc;
```

```
dsid=open("mydata","i");
num=attrn(dsid,"nvars");
do i=1 to num;
name=varname(dsid,i);
type=vartype(dsid,i);
length=varlen(dsid,i);
output;
end;
rc=close(dsid);
run;
```

関連項目:

関数:

- "OPEN 関数" (697 ページ)
- "VARNUM 関数" (914 ページ)

VARNAME 関数

SAS データセット変数の名前を返します。

カテゴリ: SAS ファイル I/O 関数

構文

VARNAME(*data-set-id*,*var-num*)

必須引数

data-set-id
OPEN 関数が返すデータセット識別子を指定します。

var-num
SAS データセット内の変数の位置番号を指定します。

ヒント:
この番号は、CONTENTS プロシジャによって生成されるリスト内で変数の横に示されます。

VARNUM 関数はこの番号を返します。

サンプル

この例では、SAS データセット CITY 内の最初の 5 つの変数(または変数の数が 5 以下の場合はすべての変数)の名前をマクロ変数にコピーします。

```
%macro names;
%let dsid=%sysfunc(open(city,i));
%let varlist=;
%do i=1 %to
%sysfunc(min(5,%sysfunc(attrn
(&dsid,nvars))));
%let varlist=&varlist %sysfunc(varname
(&dsid,&i));
```

```
%end;
%put varlist=&varlist;
%mend names;
%names
```

関連項目:

関数:

- “OPEN 関数” (697 ページ)
- “VARNUM 関数” (914 ページ)

VARNUM 関数

SAS データセット内の変数の位置番号を返します。

カテゴリ: SAS ファイル I/O 関数

構文

VARNUM(*data-set-id*,*var-name*)

必須引数

data-set-id
OPEN 関数が返すデータセット識別子を指定します。

var-name
変数の名前を指定します。

詳細

VARNUM は、SAS データセット内の変数の位置番号を返すか、SAS データセット内に変数が存在しない場合は 0 を返します。この変数番号は、PROC CONTENTS からの出力で変数の横に示される番号と同じです。

サンプル

- この例では、変数の名前に基づいて、SAS データセット CITY 内の変数の位置番号を取得します。

```
%let dsid=%sysfunc(open(city,i));
%let citynum=%sysfunc(varnum(&dsid,CITYNAME));
%let rc=%sysfunc(fetch(&dsid));
%let cityname=%sysfunc(getvarc
(&dsid,&citynum));
```

- この例では、SASUSER.HOUSES 内の変数の名前、種類、出力形式、入力形式、ラベル、長さ、位置を含むデータセットを作成します。

```
/* Tested 2/27/98 - OK */
%INCLUDE '/local/u/lirezn/sasuser/assist.src';

data vars;
length name $ 8 type $ 1
```

```
format informat $ 10 label $ 40;
drop dsid i num rc;
dsid=open("sasuser.houses","i");
num=attrn(dsid,"nvars");
do i=1 to num;
name=varname(dsid,i);
type=vartype(dsid,i);
format=varfmt(dsid,i);
informat=varinfmt(dsid,i);
label=varlabel(dsid,i);
length=varlen(dsid,i);
position=varnum(dsid,name);
output;
end;
rc=close(dsid);
run;
```

関連項目:

関数:

- “OPEN 関数” (697 ページ)
- “VARNAME 関数” (913 ページ)

VARRAY 関数

指定した名前が配列かどうかを示す値を返します。

カテゴリ: 変数情報

制限事項: DATA ステップのみで使用します。

構文

VARRAY (*name*)

必須引数

name
スカラまたは配列参照として表される名前を指定します。
制限事項: 式は引数として使用できません。

詳細

VARRAY は、指定した名前が配列の場合は 1、配列でない場合は 0 を返します。

比較

- VARRAY は、指定した名前が配列かどうかを示す値を返します。VARRAYX は、指定した式の値が配列かどうかを示す値を返します。
- VARRAY は、引数として式を受け入れません。VARRAYX は式を受け入れますが、指定された変数の値で配列参照を示すことはできません。

- 関連する関数が、変数名、入力形式、出力形式など他の変数属性の値を返します。このリストについては、“カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数を参照してください。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果

array x(3) x1-x3;
a=varray(x);
B=varray(x1);
put a=;
put B=;
	a=1
	B=0

関連項目:

関数:

- “カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数

VARRAYX 関数

指定した引数の値が配列かどうかを示す値を返します。

カテゴリ: 変数情報

構文

VARRAYX (*expression*)

必須引数

式

文字定数、変数または式を指定します。

制限事項: 指定された式の値で配列参照を示すことはできません。

詳細

VARRAYX は、指定した引数の値が配列名の場合は 1、配列名でない場合は 0 を返します。

比較

- VARRAY は、指定した名前が配列名かどうかを示す値を返します。
  VARRAYX は、指定した式の値が配列名かどうかを示す値を返します。
- VARRAY は、引数として式を受け入れません。VARRAYX は式を受け入れますが、指定された変数の値で配列参照を示すことはできません。

- 関連する関数が、変数名、入力形式、出力形式など他の変数属性の値を返します。このリストについては、“カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数を参照してください。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果

array x(3) x1-x3;
array vx(4) $6 vx1 vx2 vx3 vx4
('x' 'x1' 'x2' 'x3');
y=varrayx(vx(1));
z=varrayx(vx(2));
put y=;
put z=;
	y=1
	z=0

関連項目:

関数:

- “カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数

VARTYPE 関数

SAS データセット変数のデータの種類を返します。

カテゴリ: SAS ファイル I/O 関数

構文

VARTYPE(*data-set-id*,*var-num*)

必須引数

data-set-id
OPEN 関数が返すデータセット識別子を指定します。

var-num
SAS データセット内の変数の位置番号を指定します。

ヒント:
この番号は、CONTENTS プロシジャによって生成されるリスト内で変数の横に示されます。

VARNUM 関数はこの番号を返します。

詳細

VARTYPE は、文字変数には C、数値変数には N を返します。

サンプル

サンプル 1: VARTYPE を使用した数値変数の判別

この例では、SAS データセット MYDATA 内のすべての数値変数の名前をマクロ変数に挿入します。

```
%let dsid=%sysfunc(open(mydata,i));
%let varlist=;
%do i=1 %to %sysfunc(attrn(&dsid,nvars));
%if (%sysfunc(vartype(&dsid,&i)) = N) %then
%let varlist=&varlist %sysfunc(varname
(&dsid,&i));
%end;
%let rc=%sysfunc(close(&dsid));
```

サンプル 2: VARTYPE を使用した文字変数の判別

この例は、SAS データセット MYDATA 内に各文字変数の名前と書式化された内容を含むデータセットを作成します。

```
data vars;
length name $ 8 content $ 20;
drop dsid i num fmt rc;
dsid=open("mydata","i");
num=attrn(dsid,"nvars");
do while (fetch(dsid)=0);
do i=1 to num;
name=varname(dsid,i);
fmt=varfmt(dsid,i);
if (vartype(dsid,i)='C') then do;
content=getvarc(dsid,i);
if (fmt ne '' ) then
content=left(putc(content,fmt));
output;
end;
end;
end;
rc=close(dsid);
run;
```

関連項目:

関数:

- “VARNUM 関数” (914 ページ)

VERIFY 関数

他の文字列に存在しない文字の最初の出現位置を返します。

カテゴリ: 文字関数

制限事項: I18N レベル 0 の関数は 1 バイト文字セット(SBCS)でのみ使用するように設計されています。

ヒント: DBCS に相当する関数は、KVERIFY です。

構文

VERIFY(*source*,*excerpt-1*<, …, *excerpt-n*>)

必須引数

ソース
文字定数、変数または式を指定します。

excerpt
文字定数、変数または式を指定します。複数の excerpt を指定する場合は、カンマで区切ります。

詳細

VERIFY 関数は、いずれの *excerpt* 内にも存在しない文字のうち、最初の文字の *source* 内での位置を返します。VERIFY は、*source* 内のすべての文字が、少なくとも 1 つの *excerpt* 内にあることを確認すると、0 を返します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
data scores; input Grade : $1. @@; check='abcdf'; if verify(grade,check)>0 then put @1 'INVALID ' grade=; datalines; a b c b c d f a a q a b d d b ;	INVALID Grade=q

関連項目:

関数:

- “FINDC 関数” (450 ページ)

VFORMAT 関数

指定した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

制限事項: DATA ステップのみで使用します。

構文

VFORMAT (*var*)

必須引数

var
スカラまたは配列参照として表される変数を指定します。

制限事項: 式は引数として使用できません。

詳細

まだ長さが割り当てられていない変数に VFORMAT 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。

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)。

比較

- VFORMAT は、指定した変数に関連付けられた出力形式を返します。一方、VFORMATX は、変数名を判別するために引数を評価します。その後、VFORMATX 関数はその変数名に関連付けられた出力形式を返します。
- VFORMAT は、引数として式を受け入れません。VFORMATX は式を受け入れますが、指定された式の値で配列参照を示すことはできません。
- 関連する関数が、変数名、種類、長さなど他の変数属性の値を返します。このリストについては、“カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数を参照してください。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果

array x(3) x1-x3;
format x1 best6.;
y=vformat(x(1));
put y=;
	y=BEST6.

関連項目:

関数:

- “カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数

VFORMATD 関数

指定した変数に関連付けられた出力形式の 10 進値を返します。

カテゴリ: 変数情報

構文

VFORMATD (*var*)

必須引数

var
スカラまたは配列参照として表される変数を指定します。
制限事項: 式は引数として使用できません。

比較

- VFORMATD は、指定した変数に関連付けられた出力形式の 10 進値を返します。一方、VFORMATDX は、変数名を判別するために引数を評価します。その後、VFORMATDX 関数はその変数名に関連付けられた出力形式の 10 進値を返します。
- VFORMATD は、引数として式を受け入れません。VFORMATDX は式を受け入れますが、指定された式の値で配列参照を示すことはできません。
- 関連する関数が、変数名、種類、長さなど他の変数属性の値を返します。このリストについては、“カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数を参照してください。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果

array x(3) x1-x3;
format x1 comma8.2;
y=vformatd(x(1));
put y=;
	y=2

関連項目:

関数:

- “カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数

VFORMATDX 関数

指定した引数の値に関連付けられた出力形式の 10 進値を返します。

カテゴリ: 変数情報

構文

VFORMATDX (*expression*)

必須引数

式
変数名に対して評価する SAS 文字定数、変数または式を指定します。
制限事項: 指定された式の値で配列参照を示すことはできません。

詳細

- VFORMATD は、指定した変数に関連付けられた出力形式の 10 進値を返します。一方、VFORMATDX は、変数名を判別するために引数を評価します。その後、VFORMATDX 関数はその変数名に関連付けられた出力形式の 10 進値を返します。
- VFORMATD は、引数として式を受け入れません。VFORMATDX は式を受け入れますが、指定された式の値で配列参照を示すことはできません。
- 関連する関数が、変数名、長さ、種類など他の変数属性の値を返します。このリストについては、“カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数を参照してください。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果

array x(3) x1-x3;
format x1 comma8.2;
array vx(3) $6 vx1 vx2 vx3
('x1' 'x2' 'x3');
y=vformatdx(vx(1));
z=vformatdx('x'||'1');
put y=;
put z=;
	y=2
	z=2

関連項目:

関数:

- “カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数

VFORMATN 関数

指定した変数に関連付けられた出力形式名を返します。

カテゴリ: 変数情報

構文

VFORMATN (*var*)

必須引数

var

スカラまたは配列参照として表される変数を指定します。

制限事項: 式は引数として使用できません。

詳細

まだ長さが割り当てられていない変数に VFORMATN 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。

VFORMATN は出力形式名のみを返します。この名前には、幅およびピリオドは含まれません($CHAR など)。

比較

- VFORMATN は、指定した変数に関連付けられた出力形式名を返します。一方、VFORMATNX は、変数名を判別するために引数を評価します。その後、VFORMATNX 関数はその変数名に関連付けられた出力形式名を返します。
- VFORMATN は、引数として式を受け入れません。VFORMATNX は式を受け入れますが、指定された式の値で配列参照を示すことはできません。
- 関連する関数が、変数名、種類、長さなど他の変数属性の値を返します。このリストについては、“カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数を参照してください。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果

array x(3) x1-x3;
format x1 best6.;
y=vformatn(x(1));
put y=;
	y=BEST

関連項目:

関数:

- “カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数

VFORMATNX 関数

指定した引数の値に関連付けられた出力形式名を返します。

カテゴリ: 変数情報

構文

VFORMATNX (*expression*)

必須引数

式

変数名に対して評価する文字定数、変数または式を指定します。

制限事項: 指定された式の値で配列参照を示すことはできません。

詳細

まだ長さが割り当てられていない変数に VFORMATNX 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。

VFORMATNX は出力形式名のみを返します。この名前には、長さおよびピリオドは含まれません($CHAR など)。

比較

- VFORMATN は、指定した変数に関連付けられた出力形式名を返します。一方、VFORMATNX は、変数名を判別するために引数を評価します。その後、VFORMATNX 関数はその変数名に関連付けられた出力形式名を返します。
- VFORMATN は、引数として式を受け入れません。VFORMATNX は式を受け入れますが、指定された式の値で配列参照を示すことはできません。
- 関連する関数が、変数名、長さ、種類など他の変数属性の値を返します。このリストについては、“カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数を参照してください。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果

array x(3) x1-x3;
format x1 best6.;
array vx(3) $6 vx1 vx2 vx3
('x1' 'x2' 'x3');
y=vformatnx(vx(1));
put y=;
	y=BEST

関連項目:

関数:

- “カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数

VFORMATW 関数

指定した変数に関連付けられた出力形式の幅を返します。

カテゴリ: 変数情報

構文

VFORMATW (*var*)

必須引数

var
スカラまたは配列参照として表される変数を指定します。

制限事項: 式は引数として使用できません。

比較

- VFORMATW は、指定した変数に関連付けられた出力形式の幅を返します。一方、VFORMATWX は、変数名を判別するために引数を評価します。その後、VFORMATWX 関数はその変数名に関連付けられた出力形式の幅を返します。
- VFORMATW は、引数として式を受け入れません。VFORMATWX は式を受け入れますが、指定された式の値で配列参照を示すことはできません。
- 関連する関数が、変数名、種類、長さなど他の変数属性の値を返します。このリストについては、“カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数を参照してください。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果

array x(3) x1-x3;
format x1 best6.;
y=vformatw(x(1));
put y=;
	y=6

関連項目:

関数:

- “カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数

VFORMATWX 関数

指定した引数の値に関連付けられた出力形式の幅を返します。

カテゴリ: 変数情報

構文

VFORMATWX (*expression*)

必須引数

式

変数名に対して評価する文字定数、変数または式を指定します。

制限事項: 指定された式の値で配列参照を示すことはできません。

比較

- VFORMATW は、指定した変数に関連付けられた出力形式の幅を返します。一方、VFORMATWX は、変数名を判別するために引数を評価します。その

後、VFORMATWX 関数はその変数名に関連付けられた出力形式の幅を返します。

- VFORMATW は、引数として式を受け入れません。VFORMATWX は式を受け入れますが、指定された式の値で配列参照を示すことはできません。
- 関連する関数が、変数名、長さ、種類など他の変数属性の値を返します。このリストについては、“カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数を参照してください。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果

array x(3) x1-x3;
format x1 best6.;
array vx(3) $6 vx1 vx2 vx3
('x1' 'x2' 'x3');
y=vformatwx(vx(1));
put y=;
	y=6

関連項目:

関数:

- “カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数

VFORMATX 関数

指定した引数の値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

構文

VFORMATX (*expression*)

必須引数

式

変数名に対して評価する文字定数、変数または式を指定します。

制限事項: 指定された式の値で配列参照を示すことはできません。

詳細

まだ長さが割り当てられていない変数に VFORMATX 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。

VFORMATX 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)。

比較

- VFORMAT は、指定した変数に関連付けられた出力形式を返します。一方、VFORMATX は、変数名を判別するために引数を評価します。その後、VFORMATX 関数はその変数名に関連付けられた出力形式を返します。
- VFORMAT は、引数として式を受け入れません。VFORMATX は式を受け入れますが、指定された式の値で配列参照を示すことはできません。
- 関連する関数が、変数名、長さ、種類など他の変数属性の値を返します。このリストについては、“カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数を参照してください。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果

array x(3) x1-x3;
format x1 best6.;
format x2 20.10;
array vx(3) $6 vx1 vx2 vx3
('x1' 'x2' 'x3');
y=vformatx(vx(1));
z=vformatx(vx(2));
put y=;
put z=;
	y=BEST6.
	z=F20.10

関連項目:

関数:

- “カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数

VINARRAY 関数

指定した変数が配列のメンバかどうかを示す値を返します。

カテゴリ: 変数情報

制限事項: DATA ステップのみで使用します。

構文

VINARRAY (*var*)

必須引数

var

スカラまたは配列参照として表される変数を指定します。

制限事項: 式は引数として使用できません。

詳細

VINARRAY は、指定した変数が配列のメンバの場合は 1、配列のメンバでない場合は 0 を返します。

比較

- VINARRAY は、指定した変数が配列のメンバかどうかを示す値を返します。一方、VINARRAYX は、変数名を判別するために引数を評価します。その後、VINARRAYX はその変数名が配列のメンバかどうかを示す値を返します。
- VINARRAY は、引数として式を受け入れません。VINARRAYX は式を受け入れますが、指定された式の値で配列参照を示すことはできません。
- 関連する関数が、変数名、入力形式、出力形式など他の変数属性の値を返します。このリストについては、“カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)を参照してください。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果

array x(3) x1-x3;
y=vinarray(x);
Z=vinarray(x1);
put y=;
put Z=;
	y=0
	z=1

関連項目:

関数:

- “カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数

VINARRAYX 関数

指定した引数の値が配列のメンバかどうかを示す値を返します。

カテゴリ: 変数情報

構文

VINARRAYX (*expression*)

必須引数

式

変数名に対して評価する文字定数、変数または式を指定します。

制限事項: 指定された式の値で配列参照を示すことはできません。

詳細

VINARRAYX は、指定した引数の値が配列のメンバの場合は 1、配列のメンバでない場合は 0 を返します。

比較

- VINARRAY は、指定した変数が配列のメンバかどうかを示す値を返します。一方、VINARRAYX は、変数名を判別するために引数を評価します。その後、VINARRAYX はその変数名が配列のメンバかどうかを示す値を返します。
- VINARRAY は、引数として式を受け入れません。VINARRAYX は式を受け入れますが、指定された式の値で配列参照を示すことはできません。
- 関連する関数が、変数名、入力形式、出力形式など他の変数属性の値を返します。このリストについては、“カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数を参照してください。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果

array x(3) x1-x3;
array vx(4) $6 vx1 vx2 vx3 vx4
('x' 'x1' 'x2' 'x3');
y=vinarrayx(vx(1));
z=vinarrayx(vx(2));
put y=;
put z=;
	y=0
	z=1

関連項目:

関数:

- “カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数

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

制限事項: DATA ステップのみで使用します。

構文

VINFORMAT (*var*)

必須引数

var
スカラまたは配列参照として表される変数を指定します。

制限事項: 式は引数として使用できません。

詳細

まだ長さが割り当てられていない変数に VINFORMAT 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。

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)。

比較

- VINFORMAT は、指定した変数に関連付けられた入力形式を返します。一方、VINFORMATX は、変数名を判別するために引数を評価します。その後、VINFORMATX 関数はその変数名に関連付けられた入力形式を返します。
- VINFORMAT は、引数として式を受け入れません。VINFORMATX は式を受け入れますが、指定された式の値で配列参照を示すことはできません。
- 関連する関数が、変数名、種類、長さなど他の変数属性の値を返します。このリストについては、“カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数を参照してください。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
informat x $char6.; input x; y=vinformat(x); put y=;	y=$CHAR6.

関連項目:

関数:

- “カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数

VINFORMATD 関数

指定した変数に関連付けられた入力形式の 10 進値を返します。

カテゴリ: 変数情報

構文

VINFORMATD (*var*)

必須引数

var
スカラまたは配列参照として表される変数を指定します。
制限事項: 式は引数として使用できません。

比較

- VINFORMATD は、指定した変数に関連付けられた入力形式の 10 進値を返します。一方、VINFORMATDX は、変数名を判別するために引数を評価します。その後、VINFORMATDX 関数はその変数名に関連付けられた入力形式の 10 進値を返します。
- VINFORMATD は、引数として式を受け入れません。VINFORMATDX は式を受け入れますが、指定された式の値で配列参照を示すことはできません。
- 関連する関数が、変数名、種類、長さなど他の変数属性の値を返します。このリストについては、“カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数を参照してください。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
informat x comma8.2; input x; y=vinformatd(x); put y=;	y=2

関連項目:

関数:

- “カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数

VINFORMATDX 関数

指定した変数の値に関連付けられた入力形式の 10 進値を返します。

カテゴリ: 変数情報

構文

VINFORMATDX (*expression*)

必須引数

式
変数名に対して評価する文字定数、変数または式を指定します。
制限事項: 指定された変数の値で配列参照を示すことはできません。

比較

- VINFORMATD は、指定した変数に関連付けられた入力形式の 10 進値を返します。一方、VINFORMATDX は、変数名を判別するために引数を評価します。その後、VINFORMATDX 関数はその変数名に関連付けられた入力形式の 10 進値を返します。
- VINFORMATD は、引数として式を受け入れません。VINFORMATDX は式を受け入れますが、指定された式の値で配列参照を示すことはできません。
- 関連する関数が、変数名、長さ、種類など他の変数属性の値を返します。このリストについては、“カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数を参照してください。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
informat x1 x2 x3 comma9.3; input x1 x2 x3; array vx(3) $6 vx1 vx2 vx3 ('x1' 'x2' 'x3'); y=vinformatdx(vx(1)); put y=;	y=3

関連項目:

関数:

- “カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数

VINFORMATN 関数

指定した変数に関連付けられた入力形式名を返します。

カテゴリ: 変数情報

構文

VINFORMATN (*var*)

必須引数

var

スカラまたは配列参照として表される変数を指定します。

制限事項: 式は引数として使用できません。

詳細

まだ長さが割り当てられていない変数に VINFORMATN 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。

VINFORMATN は入力形式名のみを返します。この名前には、幅およびピリオドは含まれません($CHAR など)。

比較

- VINFORMATN は、指定した変数に関連付けられた入力形式名を返します。一方、VINFORMATNX は、変数名を判別するために引数を評価します。その後、VINFORMATNX 関数はその変数名に関連付けられた入力形式名を返します。
- VINFORMATN は、引数として式を受け入れません。VINFORMATNX は式を受け入れますが、指定された式の値で配列参照を示すことはできません。
- 関連する関数が、変数名、種類、長さなど他の変数属性の値を返します。このリストについては、“カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数を参照してください。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
informat x $char6.; input x; y=vinformatn(x); put y=;	y=$CHAR

関連項目:

関数:

- “カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数

VINFORMATNX 関数

指定した引数の値に関連付けられた入力形式名を返します。

カテゴリ: 変数情報

構文

VINFORMATNX (*expression*)

必須引数

式

変数名に対して評価する文字定数、変数または式を指定します。

制限事項: 指定された式の値で配列参照を示すことはできません。

詳細

まだ長さが割り当てられていない変数に VINFORMATNX 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。

VINFORMATNX は入力形式名のみを返します。この名前には、幅およびピリオドは含まれません($CHAR など)。

比較

- VINFORMATN は、指定した変数に関連付けられた入力形式名を返します。一方、VINFORMATNX は、変数名を判別するために引数を評価します。その後、VINFORMATNX 関数はその変数名に関連付けられた入力形式名を返します。
- VINFORMATN は、引数として式を受け入れません。VINFORMATNX は式を受け入れますが、指定された式の値で配列参照を示すことはできません。
- 関連する関数が、変数名、長さ、種類など他の変数属性の値を返します。このリストについては、“カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数を参照してください。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
informat x1 x2 x3 $char6.; input x1 x2 x3; array vx(3) $6 vx1 vx2 vx3 ('x1' 'x2' 'x3'); y=vinformatnx(vx(1)); put y=;	y=$CHAR

関連項目:

関数:

- “カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数

VINFORMATW 関数

指定した変数に関連付けられた入力形式の幅を返します。

カテゴリ: 変数情報

構文

VINFORMATW (*var*)

必須引数

var
スカラまたは配列参照として表される変数を指定します。
制限事項: 式は引数として使用できません。

比較

- VINFORMATW は、指定した変数に関連付けられた入力形式の幅を返します。一方、VINFORMATWX は、変数名を判別するために引数を評価します。その後、VINFORMATWX 関数はその変数名に関連付けられた入力形式の幅を返します。
- VINFORMATW は、引数として式を受け入れません。VINFORMATWX は式を受け入れますが、指定された式の値で配列参照を示すことはできません。
- 関連する関数が、変数名、種類、長さなど他の変数属性の値を返します。このリストについては、“カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数を参照してください。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
informat x $char6.; input x; y=vinformatw(x); put y=;	y=6

関連項目:

関数:

- “カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数

VINFORMATWX 関数

指定した引数の値に関連付けられた入力形式の幅を返します。

カテゴリ: 変数情報

構文

VINFORMATWX (*expression*)

必須引数

式
変数名に対して評価する文字定数、変数または式を指定します。
制限事項: 指定された式の値で配列参照を示すことはできません。

比較

- VINFORMATW は、指定した変数に関連付けられた入力形式の幅を返します。一方、VINFORMATWX は、変数名を判別するために引数を評価します。その後、VINFORMATWX 関数はその変数名に関連付けられた入力形式の幅を返します。
- VINFORMATW は、引数として式を受け入れません。VINFORMATWX は式を受け入れますが、指定された式の値で配列参照を示すことはできません。
- 関連する関数が、変数名、長さ、種類など他の変数属性の値を返します。このリストについては、“カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数を参照してください。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
informat x1 x2 x3 $char6.; input x1 x2 x3; array vx(3) $6 vx1 vx2 vx3 ('x1' 'x2' 'x3'); y=vinformatwx(vx(1)); put y=;	y=6

関連項目:

関数:

- “カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数

VINFORMATX 関数

指定した引数の値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

構文

VINFORMATX (*expression*)

必須引数

式

変数名に対して評価する文字定数、変数または式を指定します。

制限事項: 指定された式の値で配列参照を示すことはできません。

詳細

まだ長さが割り当てられていない変数に VINFORMATX 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。

VINFORMATX 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)。

比較

- VINFORMAT は、指定した変数に関連付けられた入力形式を返します。一方、VINFORMATX は、変数名を判別するために引数を評価します。その後、VINFORMATX 関数はその変数名に関連付けられた入力形式を返します。
- VINFORMAT は、引数として式を受け入れません。VINFORMATX は式を受け入れますが、指定された式の値で配列参照を示すことはできません。
- 関連する関数が、変数名、長さ、種類など他の変数属性の値を返します。このリストについては、“カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数を参照してください。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
informat x1 x2 x3 $char6.; input x1 x2 x3; array vx(3) $6 vx1 vx2 vx3 ('x1' 'x2' 'x3'); y=vinformatx(vx(1)); put y=;	y=$CHAR6.

関連項目:

関数:

- “カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数

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

制限事項: DATA ステップのみで使用します。

構文

VLABEL (*var*)

必須引数

var

スカラまたは配列参照として表される変数を指定します。

制限事項: 式は引数として使用できません。

詳細

まだ長さが割り当てられていない変数に VLABEL 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。

ラベルが存在しない場合、VLABEL は変数名を返します。

比較

- VLABEL は指定した変数のラベルを返します。ラベルが存在しない場合には、指定した変数の名前を返します。一方、VLABELX は、変数名を判別するために引数を評価します。その後、VLABELX 関数はその変数名に関連付けられたラベルを返すか、ラベルが存在しない場合は変数名を返します。
- VLABEL は、引数として式を受け入れません。VLABELX は式を受け入れますが、指定された式の値で配列参照を示すことはできません。
- VLABEL には CALL LABEL と同じ機能があります。
- 関連する関数が、変数名、入力形式、出力形式など他の変数属性の値を返します。このリストについては、“カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)を参照してください。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果

array x(3) x1-x3;
label x1='Test1';
y=vlabel(x(1));
put y=;
	y=Test1

関連項目:

関数:

- “カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報関数

VLABELX 関数

指定した引数の値に関連付けられたラベルを返します。

カテゴリ: 変数情報

構文

VLABELX (*expression*)

必須引数

式

変数名に対して評価する文字定数、変数または式を指定します。

制限事項: 指定された式の値で配列参照を示すことはできません。

詳細

まだ長さが割り当てられていない変数に VLABELX 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。

ラベルが存在しない場合、VLABELX は変数名を返します。

比較

- VLABEL は、指定した変数のラベルを返すか、ラベルが存在しない場合は指定した変数の名前を返します。一方、VLABELX は、変数名を判別するために引数を評価します。その後、VLABELX 関数はその変数名に関連付けられたラベルを返すか、ラベルが存在しない場合は変数名を返します。
- VLABEL は、引数として式を受け入れません。VLABELX は式を受け入れますが、指定された式の値で配列参照を示すことはできません。
- 関連する関数が、変数名、入力形式、出力形式など他の変数属性の値を返します。このリストについては、“カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)を参照してください。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果

array x(3) x1-x3;
array vx(3) $6 vx1 vx2 vx3
('x1' 'x2' 'x3');
label x1='Test1';
y=vlabelx(vx(1));
put y=;
	y=Test1

関連項目:

関数:

- “カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数

VLENGTH 関数

指定した変数のコンパイル時(配分された)サイズを返します。

カテゴリ: 変数情報

制限事項: DATA ステップのみで使用します。

構文

VLENGTH (*var*)

必須引数

var
スカラまたは配列参照として表される変数を指定します。

制限事項: 式は引数として使用できません。

比較

- LENGTH は、実行時に変数を調べ、末尾の空白を削除して長さを判断します。VLENGTH はコンパイル時の定数値を返します。この値は最大長を反映します。
- LENGTHC は VLENGTH と同じ値を返しますが、LENGTHC はどの呼び出し環境でも使用でき、引数にどのような式でも使用できます。
- VLENGTH は指定した変数の長さを返します。一方、VLENGTHX は、変数名を判別するために引数を評価します。その後、その変数名に関連付けられたコンパイル時のサイズを返します。
- VLENGTH は引数として式を受け入れません。VLENGTHX は式を受け入れますが、指定した式の値で配列参照を示すことはできません。
- 関連する関数が、変数名、入力形式、出力形式など他の変数属性の値を返します。このリストについては、“カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)を参照してください。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果

length x $8;
x='abc';
y=vlength(x);
z=length(x);
put y=;
put z=;
	y=8
	z=3

関連項目:

関数:

- “カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数

VLENGTHX 関数

引数の値と同じ名前の変数のコンパイル時(配分された)サイズを返します。

カテゴリ: 変数情報

構文

VLENGTHX (*expression*)

必須引数

式
変数名に対して評価する文字定数、変数または式を指定します。
制限事項: 指定された式の値で配列参照を示すことはできません。

比較

- LENGTH は、実行時に変数を調べ、末尾の空白を削除して長さを判断します。一方、VLENGTHX は、変数名を判別するために引数を評価します。その後、その変数名に関連付けられたコンパイル時のサイズを返します。
- LENGTHC は引数として式を受け入れますが、式の値と等しい名前が付けられた変数の長さではなく、式の値の長さを返します。
- VLENGTH は指定した変数の長さを返します。VLENGTHX は指定した式の値の長さを返します。
- VLENGTH は引数として式を受け入れません。VLENGTHX は式を受け入れますが、指定した式の値で配列参照を示すことはできません。
- 関連する関数が、変数名、入力形式、出力形式など他の変数属性の値を返します。このリストについては、“カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数を参照してください。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果

length x1 $8;
x1='abc';
array vx(3) $6 vx1 vx2 vx3
('x1' 'x2' 'x3');
y=vlengthx(vx(1));
z=length(x1);
put y=;
put z=;
	y=8
	z=3

関連項目:

関数:

- “カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数

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

制限事項: DATA ステップのみで使用します。

構文

VNAME (*var*)

必須引数

var
スカラまたは配列参照として表される変数を指定します。

制限事項: 式は引数として使用できません。

詳細

まだ長さが割り当てられていない変数に VNAME 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。

比較

- VNAME は指定した変数の名前を返します。一方、VNAMEX は、変数名を判別するために引数を評価します。名前が既知の変数名の場合は、その名前が返されます。既知の名前でない場合は、空白が返されます。
- VNAME は引数として式を受け入れません。VNAMEX は式を受け入れますが、指定した式の値で配列参照を示すことはできません。
- VNAME には CALL VNAME と同じ機能があります。
- 関連する関数が、変数ラベル、入力形式、出力形式など他の変数属性の値を返します。リストについては、“カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報関数を参照してください。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果

array x(3) x1-x3;
y=vname(x(1));
put y=;
	y=x1

関連項目:

関数:

- “カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数

VNAMEX 関数

変数名として指定した引数の値を検証します。

カテゴリ: 変数情報

構文

VNAMEX (*expression*)

必須引数

式

文字定数、変数または式を指定します。

制限事項: 指定された式の値で配列参照を示すことはできません。

詳細

まだ長さが割り当てられていない変数に VNAMEX 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。

比較

- VNAME は指定した変数の名前を返します。一方、VNAMEX は、変数名を判別するために引数を評価します。名前が既知の変数名の場合は、その名前が返されます。既知の名前でない場合は、空白が返されます。
- VNAME は引数として式を受け入れません。VNAMEX は式を受け入れますが、指定した変数の値で配列参照を示すことはできません。
- 関連する関数が、変数ラベル、入力形式、出力形式など他の変数属性の値を返します。リストについては、“カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報関数を参照してください。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果

array x(3) x1-x3;
array vx(3) $6 vx1 vx2 vx3
('x1' 'x2' 'x3');
y=vnamex(vx(1));
z=vnamex('x'||'1');
put y=;
put z=;
	y=x1
	z=x1

関連項目:

関数:

- “カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数

VTYPE 関数

指定した変数の種類(文字または数値)を返します。

カテゴリ: 変数情報

制限事項: DATA ステップのみで使用します。

構文

VTYPE (*var*)

必須引数

var
スカラまたは配列参照として表される変数を指定します。

制限事項: 式は引数として使用できません。

詳細

まだ長さが割り当てられていない変数に VTYPE 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 1 が割り当てられます。

VTYPE は数値変数には N、文字変数には C を返します。

比較

- VTYPEX は指定した変数の種類を返します。一方、VTYPEX は、変数名を判別するために引数を評価します。その後、その変数名に関連付けられた種類(文字または数値)を返します。
- VTYPE は引数として式を受け入れません。VTYPEX は式を受け入れますが、指定した式の値で配列参照を示すことはできません。
- 関連する関数が、変数名、入力形式、出力形式など他の変数属性の値を返します。リストについては、“カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報関数を参照してください。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果

array x(3) x1-x3;
y=vtype(x(1));
put y=;
	y=N

関連項目:

関数:

- “カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数

VTYPEX 関数

指定した引数の値の種類(文字または数値)を返します。

カテゴリ: 変数情報

構文

VTYPEX (*expression*)

必須引数

式

変数名に対して評価する文字定数、変数または式を指定します。

制限事項: 指定された式の値で配列参照を示すことはできません。

詳細

まだ長さが割り当てられていない変数に VTYPEX 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 1 が割り当てられます。

VTYPEX は数値変数には N、文字変数には C を返します。

比較

- VTYPEX は指定した変数の種類を返します。一方、VTYPEX は、変数名を判別するために引数を評価します。その後、その変数名に関連付けられた種類(文字または数値)を返します。
- VTYPE は引数として式を受け入れません。VTYPEX は式を受け入れますが、指定した式の値で配列参照を示すことはできません。
- 関連する関数が、変数名、入力形式、出力形式など他の変数属性の値を返します。リストについては、“カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報関数を参照してください。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果

array x(3) x1-x3;
array vx(3) $6 vx1 vx2 vx3
('x1' 'x2' 'x3');
y=vtypex(vx(1));
put y=;
	y=N

関連項目:

関数:

- “カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数

VVALUE 関数

指定する変数に関連付けられている出力形式を適用した値を返します。

カテゴリ: 変数情報

制限事項: DATA ステップのみで使用します。

構文

VVALUE(*var*)

必須引数

var
スカラまたは配列参照として表される変数を指定します。

制限事項: 式は引数として使用できません。

詳細

まだ長さが割り当てられていない変数に VVALUE 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。

VVALUE は指定する変数の現在の値を含む文字列を返します。この値には、変数に現在関連付けられている出力形式が適用されます。

比較

- VVALUE は指定する変数に関連付けられている値を返します。一方、VVALUEX は、変数名を判別するために引数を評価します。その後、その変数名に関連付けられた値を返します。
- VVALUE は引数として式を受け入れません。VVALUEX は式を受け入れますが、式の値で配列参照を示すことはできません。
- VVALUE と割り当てステートメントはどちらも、指定する変数の現在の値を含む文字列を返します。VVALUE を使用すると、変数に現在関連付けられている出力形式が値に適用されます。一方、割り当てステートメントを使用すると、値に出力形式は適用されません。
- PUT 関数では、指定した変数または定数の出力形式を変更できます。VVALUE は変数に関連付けられている現在の出力形式を使用します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果

y=9999;
format y comma10.2;
v=vvalue(y);
put v;
	9,999.00

関連項目:

関数:

- “VVALUEX 関数” (947 ページ)
- “カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数

VVALUEX 関数

指定する引数に関連付けられている出力形式を適用した値を返します。

カテゴリ: 変数情報

構文

VVALUEX(*expression*)

必須引数

式

変数名に対して評価する文字定数、変数または式を指定します。

制限事項: 指定された式の値で配列参照を示すことはできません。

詳細

まだ長さが割り当てられていない変数に VVALUEX 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。

VVALUEX は指定する引数の現在の値を含む文字列を返します。この値には、引数に現在関連付けられている出力形式が適用されます。

比較

- VVALUE は引数として変数を受け入れ、その変数の値を返します。一方、VVALUEX は引数として文字式を受け入れます。その後、変数名を判別する式を評価し、その変数名に関連付けられた値を返します。
- VVALUE は引数として式を受け入れませんが、配列参照は受け入れます。VVALUEX は式を受け入れますが、式の値で配列参照を示すことはできません。
- VVALUEX と割り当てステートメントはどちらも、指定する変数の現在の値を含む文字列を返します。VVALUEX を使用すると、変数に現在関連付けられている出力形式が値に適用されます。一方、割り当てステートメントを使用すると、値に出力形式は適用されません。
- PUT 関数では、指定した変数または定数の出力形式を変更できます。VVALUEX は変数に関連付けられている現在の出力形式を使用します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果

date1='31mar02'd;
date2='date1';
format date1 date7.;
datevalue=vvaluex(date2);
put datevalue;
	31MAR02

関連項目:

関数:

- “VVALUE 関数” (945 ページ)
- “カテゴリ別の SAS 関数と CALL ルーチン” (65 ページ)の変数情報の関数

WEEK 関数

週番号の値を返します。

カテゴリ: 日付と時間

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。

descriptor
ディスクリプタの値を指定します。次のディスクリプタを大文字または小文字で指定できます。

U(デフォルト)
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 ~ 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('31dec2006'd, 'u')**の値は 53 です。

ヒント: U および W ディスクリプタは似ていますが、U ディスクリプタが日曜日を週の第 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” (949 ページ)

V
1 ~ 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” (949 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 ~ 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('31dec2006'd, 'w')**の値は 53 です。

ヒント: U および W ディスクリプタは似ていますが、U ディスクリプタが日曜日を週の第 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” (949 ページ)

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 関数は口ケールに依存せず、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

U ディスクリプタを使用する 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 ~ 53 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('01jan2007'd, 'u')**の値は 0 です。

V ディスクリプタ

V ディスクリプタを使用する 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 ~ 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、01jan2006 と 30dec2005 は同じ週になります。この週の第 1 日 (月曜日)は 26dec2005 です。したがって、**week('01jan2006'd, 'v')**と **week('30dec2005'd, 'v')**は両方とも値 52 を返します。つまり、両方の日付が 2005 年の第 52 週に含まれるという意味です。

W ディスクリプタ

W ディスクリプタを使用する 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 ~ 53 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('01jan2007'd, 'w')**の値は 1 です。

ディスクリプタの比較

U はデフォルトのディスクリプタです。範囲は 0 ~ 53 で、週の第 1 日は日曜日です。V ディスクリプタの範囲は 1 ~ 53 で、週の第 1 日は月曜日です。W ディスクリプタの範囲は 0 ~ 53 で、週の第 1 日は月曜日です。

説明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 ~ 当年 1 月 4 日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 ~ 土曜日の期間は、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 ~ 日曜日の期間は、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。

サンプル

特定の年の年末および翌年の年始付近の日付に対する U、V および W ディスクリプタの値を次の例に示します。すべてのデータセットを確認し、1 月 1 日の曜日によってさまざまなディスクリプタの動作にどのような違いがあるかを示します。出力には最初の 20 件のオブザベーションが表示されます。

```
title 'Values of the U, V, and W Descriptors';
data a(drop=i date0 date1 y);
date0 = '20dec2005'd;
do y = 0 to 5;
date1 = intnx("YEAR",date0,y,'s');
do i = 0 to 20;
date = intnx("DAY",date1,i);
year = YEAR(date);
week = week(date);
week_u = week(date, 'u');
week_v = week(date, 'v');
week_w = week(date, 'w');
output;
end;
end;
format date WEEKDATX17.;
run;
proc print;
run;
```

画面 2.70 U、V および W ディスクリプタの値の識別結果

Values of the U, V, and W Descriptors

Obs	date	year	week	week_u	week_v	week_w
1	Tue, 20 Dec 2005	2005	51	51	51	51
2	Wed, 21 Dec 2005	2005	51	51	51	51
3	Thu, 22 Dec 2005	2005	51	51	51	51
4	Fri, 23 Dec 2005	2005	51	51	51	51
5	Sat, 24 Dec 2005	2005	51	51	51	51
6	Sun, 25 Dec 2005	2005	52	52	51	51
7	Mon, 26 Dec 2005	2005	52	52	52	52
8	Tue, 27 Dec 2005	2005	52	52	52	52
9	Wed, 28 Dec 2005	2005	52	52	52	52
10	Thu, 29 Dec 2005	2005	52	52	52	52
11	Fri, 30 Dec 2005	2005	52	52	52	52
12	Sat, 31 Dec 2005	2005	52	52	52	52
13	Sun, 1 Jan 2006	2006	1	1	52	0
14	Mon, 2 Jan 2006	2006	1	1	1	1
15	Tue, 3 Jan 2006	2006	1	1	1	1
16	Wed, 4 Jan 2006	2006	1	1	1	1
17	Thu, 5 Jan 2006	2006	1	1	1	1
18	Fri, 6 Jan 2006	2006	1	1	1	1
19	Sat, 7 Jan 2006	2006	1	1	1	1
20	Sun, 8 Jan 2006	2006	2	2	1	1
21	Mon, 9 Jan 2006	2006	2	2	2	2
22	Wed, 20 Dec 2006	2006	51	51	51	51
23	Thu, 21 Dec 2006	2006	51	51	51	51
24	Fri, 22 Dec 2006	2006	51	51	51	51
25	Sat, 23 Dec 2006	2006	51	51	51	51
26	Sun, 24 Dec 2006	2006	52	52	51	51
27	Mon, 25 Dec 2006	2006	52	52	52	52

関連項目:

関数:

- “INTNX 関数” (567 ページ)

出力形式:

- “WEEKUw. 出力形式” (*SAS 出力形式と入力形式: リファレンス*)
- “WEEKVw. 出力形式” (*SAS 出力形式と入力形式: リファレンス*)
- “WEEKWw. 出力形式” (*SAS 出力形式と入力形式: リファレンス*)

入力形式:

- “WEEKUw. 入力形式” (*SAS 出力形式と入力形式: リファレンス*)
- “WEEKVw. 入力形式” (*SAS 出力形式と入力形式: リファレンス*)
- “WEEKWw. 入力形式” (*SAS 出力形式と入力形式: リファレンス*)

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

構文

WEEKDAY(*date*)

必須引数

date
SAS 日付値を表す SAS 式を指定します。

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)。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
x=weekday('16mar97'd); put x;	1

WHICHC 関数

第 1 引数に等しい文字値を検索し、最初に一致した値のインデックスを返します。

カテゴリ: 検索

構文

WHICHC(*string*, *value-1* <, *value-2*, …>)

必須引数

string
検索対象の値を指定する文字定数、変数または式です。

value
検索条件の値を指定する文字定数、変数または式です。

詳細

WHICHC 関数は、第 2 以降の引数から第 1 引数に等しい値を検索し、最初に一致した値のインデックスを返します。

string が欠損している場合、WHICHC は欠損値を返します。欠損していない場合、WHICHC は *string* の値と *value-1*、*value-2* などを順番に比較します。引数 *value-i* が *string* と等しい場合、WHICHC は正の整数 *i* を返します。*string* が後続のどの引数とも等しくない場合、WHICHC は 0 を返します。

検索条件の値が頻繁に変更される場合に WHICHC を使用すると便利です。検索条件の値を変更せずに多数の検索を実行する必要がある場合、HASH オブジェクトを使用すると効率が上がります。

サンプル

次の例では、第 1 引数の配列を検索し、最初に一致した値のインデックスを返します。

```
data _null_;
array fruit (*) $12 fruit1-fruit3 ('watermelon' 'apple' 'banana');
x1=whichc('watermelon', of fruit[*]);
x2=whichc('banana', of fruit[*]);
x3=whichc('orange', of fruit[*]);
put x1= / x2= / x3=;
run;
```

SAS は次の出力を口グに書き込みます。

```
x1=1
x2=3
x3=0
```

関連項目:

関数:

- “WHICHN 関数” (954 ページ)

その他のリファレンス:

- “ハッシュオブジェクトの使用 ” (*SAS 言語リファレンス: 解説編* 22 章)
- “文字の比較の IN 演算子” (*SAS 言語リファレンス: 解説編* 6 章)

WHICHN 関数

第 1 引数に等しい数値を検索し、最初に一致した値のインデックスを返します。

カテゴリ: 検索

構文

WHICHN(*argument*, *value-1* <, *value-2*, …>)

必須引数

引数
検索対象の値を指定する数値定数、変数または式です。

value
検索条件の値を指定する数値定数、変数または式です。

詳細

WHICHN 関数は、第 2 以降の引数から第 1 引数に等しい値を検索し、最初に一致した値のインデックスを返します。

string が欠損している場合、WHICHN は欠損値を返します。欠損していない場合、WHICHN は *string* の値と *value-1*、*value-2* などを順番に比較します。引数 *value-i* が *string* と等しい場合、WHICHN は正の整数 *i* を返します。*string* が後続のどの引数とも等しくない場合、WHICHN は 0 を返します。

検索条件の値が頻繁に変更される場合に WHICHN を使用すると便利です。検索条件の値を変更せずに多数の検索を実行する必要がある場合、HASH オブジェクトを使用すると効率が上がります。

サンプル

次の例では、第 1 引数の配列を検索し、最初に一致した値のインデックスを返します。

```
data _null_;
array dates[*] Columbus Hastings Nicea US_Independence missing
Magna_Carta Gutenberg
(1492 1066 325 1776 . 1215 1450);
x0=whichn(., of dates[*]);
x1=whichn(1492, of dates[*]);
x2=whichn(1066, of dates[*]);
x3=whichn(1450, of dates[*]);
x4=whichn(1000, of dates[*]);
put x0= / x1= / x2= / x3= / x4=;
run;
```

SAS は次の出力を口グに書き込みます。

```
x0=.
x1=1
x2=2
x3=7
x4=0
```

関連項目:

関数:

- “WHICHC 関数” (953 ページ)

その他のリファレンス:

- “数値の比較の IN 演算子” (*SAS 言語リファレンス: 解説編* 6 章)
- “ハッシュオブジェクトの使用 ” (*SAS 言語リファレンス: 解説編* 22 章)

YEAR 関数

SAS 日付値から年を返します。

カテゴリ: 日付と時間

構文

YEAR(*date*)

必須引数

date
SAS 日付値を表す SAS 式を指定します。

詳細

YEAR 関数は、年を表す 4 桁の数値を生成します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果

date='25dec97'd;
y=year(date);
put y;
	1997

関連項目:

関数:

- “DAY 関数” (354 ページ)

- “MONTH 関数” (652 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: 財務関数

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: $A > 0$

c

分数で表す、年間の名目クーポン率を指定します。

範囲: $0 \le c < 1$

n

1 年当たりのクーポン数を指定します。

範囲: $n > 0$ 整数です。

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: $K > 0$ 整数です。

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: $0 < k_0 \le \frac{1}{n}$

p

未払い利息込みの価格を指定します。

範囲: $p > 0$

詳細

YIELDP 関数は関係に基づきます。

$$P = \sum_{k=1}^{K} c(k) \frac{1}{\left(1 + \frac{y}{n}\right)^{t_k}}$$

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。

サンプル

次の例では、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。

```
data _null_;
y=yieldp(1000,.01,4,14,.165,800);
put y;
run;
```

返される値は 0.0775031248 です。

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

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。

end-date
終了日を識別する SAS 日付値を指定します。

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。

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。

サンプル

サンプル 1: Basis に基づいた年単位の差の計算

次の例では、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。

```
data _null_;
sdate='16oct1998'd;
edate='16feb2010'd;
y30360=yrdif(sdate, edate, '30/360');
yactact=yrdif(sdate, edate, 'ACT/ACT');
yact360=yrdif(sdate, edate, 'ACT/360');
yact365=yrdif(sdate, edate, 'ACT/365');
put y30360= / yactact= / yact360= / yact365= ;
run;
```

SAS は次の出力を口グに書き込みます。

```
y30360=11.333333333
yactact=11.336986301
yact360=11.502777778
yact365=11.345205479
```

サンプル 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。

```
data _null_;
sdate='16oct1998'd;
edate='16feb2010'd;
age=yrdif(sdate, edate, 'AGE');
put age= 'years';
run;
```

SAS は次の出力をログに書き込みます。

```
age=11.336986301 years
```

関連項目:

関数:

- “DATDIF 関数” (349 ページ)

リファレンス

“Day count convention.” *Wikipedia*, 2010. URI: Day count convention.

ISDA International Swaps and Derivatives Association, Inc “EMU and Market Conventions: Recent Developments.” 1998. Wikipedia. URI: EMU and Market Conventions: Recent Developments.

Mayle, Jan. 1994. *Standard Securities Calculation Methods – Fixed Income Securities Formulas for Analytic Measures*. 2 巻 NY, NY: 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

構文

YYQ(*year*,*quarter*)

必須引数

年

年を表す 2 桁または 4 桁の整数を示します。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。

quarter
年の四半期(1、2、3、4)を指定します。

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* のいずれかが欠損している場合、または四半期値が無効の場合、結果は欠損します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
DateValue=yyq(2001,3); put DateValue; put DateValue date7.; put DateValue date9.;	 15157 01JUL01 01JUL2001
StartOfQtr=yyq(99,4); put StartOfQtr; put StartOfQtr=worddate.;	14518 StartOfQtr=October 1, 1999

関連項目:

関数:

- “QTR 関数” (777 ページ)
- “YEAR 関数” (955 ページ)

システムオプション:

- “YEARCUTOFF= System Option” in *SAS System Options: Reference*

ZIPCITY 関数

都市名と、郵便番号に対応する 2 文字の郵便コードを返します。

カテゴリ: 州コード/郵便番号

構文

ZIPCITY(*ZIP-code*)

必須引数

ZIP-code
5 桁の郵便番号を含む数値式または文字式を指定します。

ヒント: *ZIP-code* の値がゼロで始まる場合は、先頭のゼロを省略した値を入力できます。たとえば、1040 と入力すると、ZIPCITY は値を 01040 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 ZIPCITY 関数から値が返される場合、変数にはデフォルトで長さ 20 が割り当てられます。

ZIPCITY は、都市名と、5 桁の郵便番号引数に対応する 2 文字の郵便コードを返します。ZIPCITY は大文字小文字混在の文字値を返します。郵便番号が不明の場合、ZIPCITY は空白値を返します。

注: この関数を使用するときには SASHELP.ZIPCODE データセットが必要です。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。

郵便番号から州の郵便コードへの変換方法

特定の郵便番号に対応する州を判断するために、この関数は各州の郵便番号の開始値と終了値で構成されるゾーンテーブルを使用します。次に、その郵便番号範囲に対応する州を検索します。ゾーンテーブルは、例外を考慮して各州の郵便番号の開始値と終了値で構成され、郵便番号値は検証されません。

若干の例外はありますが、1 つのゾーンが複数の州におよぶことはありません。この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新しいゾーンまたは例外が追加される可能性もあります。ただし、SAS ソフトウェアは製品の新しいリリースが発表されるときに更新されます。

州の郵便コードテーブルの最終更新日の判断

SASHELP.ZIPCODE データセットには、ZIPCITY およびその他の郵便番号関数で使用される郵便番号情報が含まれています。このデータセットの最新更新日を調べるには、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には、最新更新日と SASHELP.ZIPCODE データセットの内容が表示されます。

注: SAS 社外向け Web サイトから SASHELP.ZIPCODE の最新バージョンを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。ダウンロードを開始するには、Name 列から **Zipcode Dataset** を選択します。データセットをダウンロードし、展開した後、CIMPORT プロシジャを実行する必要があります。

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、5 桁の郵便番号引数に対応する大文字小文字混在の都市名と 2 文字の郵便コードを返します。
- ZIPNAME は、5 桁の郵便番号引数に対応する米国州または米国領の名前を大文字で返します。
- ZIPNAMEL は、5 桁の郵便番号引数に対応する米国州または米国領の大文字小文字混在の名前を返します。
- ZIPSTATE は、5 桁の郵便番号引数に対応する 2 文字の州の郵便コード(または米国領の世界 GSA 地理コード)を大文字で返します。

サンプル: 例

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
city1=zipcity(27511); put city1;	Cary, NC

length zip $10.;
zip='90049-1392';
zip=substr(zip,1,5);
city2=zipcity(zip);
put city2;
	Los Angeles, CA

city3=zipcity(4338); put city3;	Augusta, ME
city4=zipcity(01040); put city4;	Holyoke, MA

関連項目:

関数:

- “ZIPFIPS 関数” (963 ページ)
- “ZIPNAME 関数” (965 ページ)
- “ZIPNAMEL 関数” (966 ページ)
- “ZIPSTATE 関数” (968 ページ)

ZIPCITYDISTANCE 関数

2 つの郵便番号が示す場所間の測地距離を返します。

カテゴリ: 距離
州コード/郵便番号

構文

ZIPCITYDISTANCE(*zip-code-1*, *zip-code-2*)

必須引数

zip-code
米国内の場所の郵便番号を含む数値式または文字式を指定します。

詳細

ZIPCITYDISTANCE 関数は、2 つの郵便番号が示す場所間の測地距離をマイル単位で返します。各郵便番号の重心が計算に使用されます。

この関数を使用するときには SASHELP.ZIPCODE データセットが必要です。この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 およびその他の郵便番号関数で使用される郵便番号情報が含まれています。このデータセットの最新更新日を調べるには、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には、最新更新日と SASHELP.ZIPCODE データセットの内容が表示されます。

注: SAS 社外向け Web サイトから SASHELP.ZIPCODE の最新バージョンを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。ダウンロードを開始するには、**Name** 列から **Zipcode Dataset** を選択します。データセットをダウンロードし、展開した後、CIMPORT プロシジャを実行する必要があります。

サンプル

次の例では、最初の郵便番号はカリフォルニア州サンフランシスコ市内の場所を示し、2 番目の郵便番号はメイン州バンガー市内の場所を示します。ZIPCITYDISTANCE は、これらの 2 つの場所間の距離をマイル単位で返します。

```
data _null_;
distance=zipcitydistance('94103', '04401');
put 'Distance from San Francisco, CA, to Bangor, ME: ' distance 4. ' miles';
run;
```

SAS は次の出力をログに書き込みます。

```
Distance from San Francisco, CA, to Bangor, ME: 2782 miles
```

関連項目:

関数:

- “ZIPCITY 関数” (960 ページ)

ZIPFIPS 関数

郵便番号を 2 桁の FIPS コードに変換します。

カテゴリ: 州コード/郵便番号

構文

ZIPFIPS(*zip-code*)

必須引数

zip-code
5 桁の郵便番号を含む数値式または文字式を指定します。

ヒント: *ZIP-code* の値がゼロで始まる場合は、先頭のゼロを省略した値を入力できます。たとえば、1040 と入力すると、ZIPFIPS は値を 01040 と想定します。

詳細

基本

ZIPFIPS 関数は、5 桁の郵便番号引数に対応する 2 桁の数値の米連邦情報処理標準(FIPS)コードを返します。

郵便番号から州の郵便コードへの変換方法

特定の郵便番号に対応する州を判断するために、この関数は各州の郵便番号の開始値と終了値で構成されるゾーンテーブルを使用します。次に、その郵便番号範囲に対応する州を検索します。ゾーンテーブルは、例外を考慮して各州の郵便番号の開始値と終了値で構成され、郵便番号値は検証されません。

若干の例外はありますが、1 つのゾーンが複数の州におよぶことはありません。この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新しいゾーンまたは例外が追加される可能性もあります。ただし、SAS ソフトウェアは製品の新しいリリースが発表されるときに更新されます。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
fips1=zipfips('27511'); put fips1;	37
fips2=zipfips('01040'); put fips2;	25
fips3=zipfips(1040); put fips3;	25
fips4=zipfips(59017); put fips4;	30
fips5=zipfips(24862); put fips5;	54

関連項目:

関数:

- “ZIPCITY 関数” (960 ページ)
- “ZIPNAME 関数” (965 ページ)
- “ZIPNAMEL 関数” (966 ページ)
- “ZIPSTATE 関数” (968 ページ)

ZIPNAME 関数

郵便番号を大文字の州名に変換します。

カテゴリ: 州コード/郵便番号

構文

ZIPNAME(*zip-code*)

必須引数

zip-code
5 桁の郵便番号を含む数値式または文字式を指定します。

ヒント: *ZIP-code* の値がゼロで始まる場合は、先頭のゼロを省略した値を入力できます。たとえば、1040 と入力すると、ZIPNAME は値を 01040 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 ZIPNAME 関数から値が返される場合、変数にはデフォルトで長さ 20 が割り当てられます。

ZIPNAME は、5 桁の郵便番号引数に対応する米国州または米国領の名前を返します。ZIPNAME は最大で 20 文字の長さの文字値をすべて大文字で返します。

郵便番号から州の郵便コードへの変換方法

特定の郵便番号に対応する州を判断するために、この関数は各州の郵便番号の開始値と終了値で構成されるゾーンテーブルを使用します。次に、その郵便番号範囲に対応する州を検索します。ゾーンテーブルは、例外を考慮して各州の郵便番号の開始値と終了値で構成され、郵便番号値は検証されません。

若干の例外はありますが、1 つのゾーンが複数の州におよぶことはありません。この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新しいゾーンまたは例外が追加される可能性もあります。ただし、SAS ソフトウェアは製品の新しいリリースが発表されるときに更新されます。

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、5 桁の郵便番号引数に対応する大文字小文字混在の都市名と 2 文字の郵便コードを返します。
- ZIPNAME は、5 桁の郵便番号引数に対応する米国州または米国領の名前を大文字で返します。
- ZIPNAMEL は、5 桁の郵便番号引数に対応する米国州または米国領の大文字小文字混在の名前を返します。
- ZIPSTATE は、5 桁の郵便番号引数に対応する 2 文字の州の郵便コード(または米国領の世界 GSA 地理コード)を大文字で返します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
state1=zipname('27511'); put state1;	NORTH CAROLINA
state2=zipname('01040'); put state2;	MASSACHUSETTS
state3=zipname(1040); put state3;	MASSACHUSETTS
state4=zipname('59017'); put state4;	MONTANA

length zip $10.;
zip='90049-1392';
zip=substr(zip,1,5);
state5=zipname(zip);
put state5;
	CALIFORNIA

関連項目:

関数:

- “ZIPCITY 関数” (960 ページ)
- “ZIPFIPS 関数” (963 ページ)
- “ZIPNAMEL 関数” (966 ページ)
- “ZIPSTATE 関数” (968 ページ)

ZIPNAMEL 関数

郵便番号を大文字小文字混在の州名に変換します。

カテゴリ: 州コード/郵便番号

構文

ZIPNAMEL(*zip-code*)

必須引数

zip-code
5 桁の郵便番号を含む数値式または文字式を指定します。

ヒント: *ZIP-code* の値がゼロで始まる場合は、先頭のゼロを省略した値を入力できます。たとえば、1040 と入力すると、ZIPNAMEL は値を 01040 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、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 桁の郵便番号引数に対応する米国州または米国領の名前を返します。ZIPNAMEL は最大で 20 文字の長さの大文字小文字混在の文字値を返します。

郵便番号から州の郵便コードへの変換方法

特定の郵便番号に対応する州を判断するために、この関数は各州の郵便番号の開始値と終了値で構成されるゾーンテーブルを使用します。次に、その郵便番号範囲に対応する州を検索します。ゾーンテーブルは、例外を考慮して各州の郵便番号の開始値と終了値で構成され、郵便番号値は検証されません。

若干の例外はありますが、1 つのゾーンが複数の州におよぶことはありません。この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新しいゾーンまたは例外が追加される可能性もあります。ただし、SAS ソフトウェアは製品の新しいリリースが発表されるときに更新されます。

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、5 桁の郵便番号引数に対応する大文字小文字混在の都市名と 2 文字の郵便コードを返します。
- ZIPNAME は、5 桁の郵便番号引数に対応する米国州または米国領の名前を大文字で返します。
- ZIPNAMEL は、5 桁の郵便番号引数に対応する米国州または米国領の大文字小文字混在の名前を返します。
- ZIPSTATE は、5 桁の郵便番号引数に対応する 2 文字の州の郵便コード(または米国領の世界 GSA 地理コード)を大文字で返します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
state1=zipnamel('27511'); put state1;	North Carolina
state2=zipnamel('01040'); put state2;	Massachusetts
state3=zipnamel(1040); put state3;	Massachusetts
state4=zipnamel(59017); put state4;	Montana

SAS ステートメント	結果

length zip $10.;
zip='90049-1392';
zip=substr(zip,1,5);
state5=zipnamel(zip);
put state5;
	California

関連項目:

関数:

- “ZIPCITY 関数” (960 ページ)
- “ZIPFIPS 関数” (963 ページ)
- “ZIPNAME 関数” (965 ページ)
- “ZIPSTATE 関数” (968 ページ)

ZIPSTATE 関数

郵便番号を 2 桁の州の郵便コードに変換します。

カテゴリ: 州コード/郵便番号

構文

ZIPSTATE(*ZIP-code*)

必須引数

ZIP-code
有効な 5 桁の郵便番号を含む数値式または文字式を指定します。

ヒント: *ZIP-code* の値がゼロで始まる場合は、先頭のゼロを省略した値を入力できます。たとえば、1040 と入力すると、ZIPSTATE は値を 01040 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、デフォルトで変数には長さ 20 が割り当てられます。

ZIPSTATE は、5 桁の郵便番号引数に対応する 2 文字の州の郵便コード(または米国領の世界 GSA 地理コード)を返します。ZIPSTATE は大文字で文字値を返します。

注: ZIPSTATE では郵便番号は検証されません。

郵便番号から州の郵便コードへの変換方法

特定の郵便番号に対応する州を判断するために、この関数は各州の郵便番号の開始値と終了値で構成されるゾーンテーブルを使用します。次に、その郵便番号範

囲に対応する州を検索します。ゾーンテーブルは、例外を考慮して各州の郵便番号の開始値と終了値で構成され、郵便番号値は検証されません。

若干の例外はありますが、1 つのゾーンが複数の州におよぶことはありません。この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新しいゾーンまたは例外が追加される可能性もあります。ただし、SAS ソフトウェアは製品の新しいリリースが発表されるときに更新されます。

米国陸軍郵便局(APO)および米国海軍郵便局(FPO)の郵便番号

ZIPSTATE 関数では、APO および FPO の郵便番号が認識されます。これらの軍の郵便番号は米国内の出口基地に対応します。郵便番号は、SASHELP.ZIPMIL データセットに含まれています。このデータセットの最新更新日を調べるには、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には、最新更新日と SASHELP.ZIPMIL データセットの内容が表示されます。

注: SAS 社外向け Web サイトから SASHELP.ZIPMIL の最新バージョンをダウンロードできます。このデータセットは **テクニカルサポート Web サイト**にあります。ダウンロードを開始するには、Name 列から **Zipcode Dataset** を選択します。データセットをダウンロードし、展開した後、CIMPORT プロシジャを実行する必要があります。

州の郵便コードテーブルの最終更新日の判断

SASHELP.ZIPCODE データセットには、APO および FPO の住所を除き、ZIPCITY およびその他の郵便番号関数で使用される郵便番号情報が含まれています。このデータセットの最新更新日を調べるには、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には、最新更新日と SASHELP.ZIPCODE データセットの内容が表示されます。

注: SAS 社外向け Web サイトから SASHELP.ZIPCODE の最新バージョンをダウンロードできます。このデータセットは **テクニカルサポート Web サイト**にあります。ダウンロードを開始するには、Name 列から **Zipcode Dataset** を選択します。データセットをダウンロードし、展開した後、CIMPORT プロシジャを実行する必要があります。

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、5 桁の郵便番号引数に対応する大文字小文字混在の都市名と 2 文字の郵便コードを返します。
- ZIPNAME は、5 桁の郵便番号引数に対応する米国州または米国領の名前を大文字で返します。
- ZIPNAMEL は、5 桁の郵便番号引数に対応する米国州または米国領の大文字小文字混在の名前を返します。
- ZIPSTATE は、5 桁の郵便番号引数に対応する 2 文字の州の郵便コード(または米国領の世界 GSA 地理コード)を大文字で返します。

サンプル

SAS ステートメントとその結果を次に示します。

SAS ステートメント	結果
state1=zipstate('27511'); put state1;	NC
state2=zipstate('01040'); put state2;	MA
state3=zipstate(1040); put state3;	MA
state4=zipstate(59017); put state4;	MT

length zip $10.;
zip='90049-1392';
zip=substr(zip,1,5);
state5=zipstate(zip);
put state5;
	CA

関連項目:

関数:

- “ZIPCITY 関数” (960 ページ)
- “ZIPFIPS 関数” (963 ページ)
- “ZIPNAME 関数” (965 ページ)
- “ZIPNAMEL 関数” (966 ページ)

3 章

リファレンス

リファレンス

Abramowitz, Milton, and Irene Stegun. 1964. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables – National Bureau of Standards Applied Mathematics Series #55*. Washington, 米国: U.S. Government Printing Office.

Amos, D. E., S. L. Daniel, and K. Weston. “CDC 6600 Subroutines IBESS and JBESS for Bessel Functions I(v,x) and J(v,x), x ≥ 0, v ≥ 0.” 1977. *ACM Transactions on Mathematical Software* 3: 76-255.

Aho, A. V., J. E. Hopcroft, and J. D. Ullman. 1974. *The Design and Analysis of Computer Algorithms*. Reading, 米国: Addison-Wesley Publishing Co..

Cheng, R. C. H. “The Generation of Gamma Variables.” 1977. *Applied Statistics* 26: 71-75.

Duncan, D. B. “Multiple Range and Multiple F Tests.” 1955. *Biometrics* 11: 1-42.

Dunnett, C. W. “A Multiple Comparisons Procedure for Comparing Several Treatments with a Control.” . *Journal of the American Statistical Association* 50: 1096-1121.

Fishman, G. S. “Sampling from the Poisson Distribution on a Computer.” 1976. *Computing* 17: 145-156.

Fishman, G. S. 1978. *Principles of Discrete Event Simulation*. New York, 米国: John Wiley & Sons, Inc.

Fishman, G. S., and L. R. Moore. “A Statistical Evaluation of Multiplicative Congruential Generators with Modulus ($2^{31} – 1$).” 1982. *Journal of the American Statistical Association* 77: 1, 29–136.

Knuth, D. E. 1973. *The Art of Computer Programming, Volume 3. Sorting and Searching*. Reading, 米国: Addison-Wesley.

Hochberg, Y., and A. C. Tamhane. 1987. *Multiple Comparison Procedures*. New York, 米国: John Wiley & Sons, Inc.

Williams, D. A. “A Test for Differences Between Treatment Means when Several Dose Levels are Compared with a Zero Dose Control.” 1971. *Biometrics* 27: 103-117.

Williams, D. A. “The Comparison of Several Dose Levels with a Zero Dose Control.” 1972. *Biometrics* 28: 519-531.

Vincenty, T. “Direct and Inverse Solutions of Geodesics on the Ellipsoid with Application of Nested Equations.” 1975. 22: 88-93.

付録 1

Perl 正規表現(PRX)のメタ文字テーブル

一般的な構成

表 A1.1 *一般的な構成*

メタ文字	説明
( )	グループ化を示します。
非メタ文字	その文字と一致します。
{ } [ ] ( ) ^ $. | * + ? \	これらの文字と一致させるには、\でオーバライド(エスケープ)します。
\	次に続くメタ文字をオーバーライドします。
n	キャプチャバッファ *n* と一致します。
(?:...)	非キャプチャグループを指定します。

基本 Perl メタ文字

次の表に、Perl 正規表現でパターンと照合するために使用できるメタ文字を示します。

表 A1.2 *基本 Perl メタ文字とその説明*

メタ文字	説明
\a	アラーム(ベル)文字と一致します。
\A	文字列の最初の文字のみと一致します。
\b	ワードの境界(ワードと空白の間の位置)と一致します。 • "er\b"は"never"の"er"と一致します。 • "er\b"は"verb"の"er"とは一致しません。

メタ文字	説明
\B	ワードの境界以外と一致します。 • "er\B"は"verb"の"er"と一致します。 • "er\B"は"never"の"er"とは一致しません。
\cA-\cZ	制御文字と一致します。たとえば、\cX は制御文字 control-X と一致します。
\C	1 バイトと一致します。
\d	数字と一致します。[0−9]と同じ意味です。
\D	数字以外と一致します。[^0−9]と同じ意味です。
\e	エスケープ文字と一致します。
\E	大文字と小文字の変更の終了を指定します。
\f	フォームフィード文字と一致します。
\l	次に続く文字を小文字に指定します。
\L	次に続く\E メタ文字までの文字列を小文字に指定します。
\n	改行文字と一致します。
\num $num	キャプチャバッファ *num* と一致します。*num* は正の整数です。Perl の変数構文($num)は、キャプチャバッファを参照する場合は有効ですが、それ以外では無効です。
\Q	すべての非ワード構成文字をエスケープ(前にバックスラッシュを挿入)します。
r	復帰文字と一致します。
\s	空白、タブ、フォームフィードなどのあらゆる空白文字と一致します。[\f\n\r\t\v]と同じ意味です。
\S	空白文字以外のあらゆる文字と一致します。[^\f\n\r\t\v]と同じ意味です。
\t	タブ文字と一致します。
\u	次に続く文字を大文字に指定します。
\U	次に続く\E メタ文字までの文字列を大文字に指定します。
\w	ワード構成文字、英数字またはアンダースコアと一致します。
\W	ワード構成文字以外、英数字以外およびアンダースコア以外と一致します。
ddd	8 進文字 *ddd* と一致します。

メタ文字	説明
\x*dd*	16 進文字 *dd* と一致します。
\z	文字列の末尾の文字のみと一致します。
\Z	文字列の末尾の文字のみと一致します。文字列の末尾が改行の場合はその直前の文字と一致します。

メタ文字と置換文字列

置換正規表現を使用した場合、正規表現と置換テキストの両方で次のメタ文字を使用できます。

- \l
- \u
- \L
- \E
- \U
- \Q

これらのメタ文字は、置換テキスト内で使用されるキャプチャバッファを制御する場合に役立ちます。これらのメタ文字の使用例については、"テキストの置換" (45 ページ) を参照してください。

これらのメタ文字の説明については、表 A1.2 (973 ページ) を参照してください。

その他の量指定子

次の表に、Perl 正規表現で使用できるその他の量指定子を示します。表内のメタ文字の説明には、そのメタ文字の使用例が含まれています。

表 A1.3 *その他の量指定子*

メタ文字	説明
\	次に続く文字を特殊文字、リテラル、前方参照、8 進エスケープのいずれかにマークします。 • "\n"は改行文字と一致します。 • "\\"は"\"と一致します。 • "\("は"("と一致します。
|	英数字の文字列を比較する場合の *or* 条件を指定します。たとえば、構成 x|y は x または y のいずれかと一致します。 • "z|food"は"z"または"food"のいずれかと一致します。 • "(z|f)ood"は"zood"または"food"のいずれかと一致します。
^	入力文字列の先頭の位置と一致します。

メタ文字	説明
$	入力文字列の末尾の位置と一致します。
ピリオド(.)	改行文字以外のあらゆる 1 文字と一致します。改行文字も含むあらゆる文字と一致させるには、"[.\n]"などのパターンを使用します。
(pattern)	グループ化を指定します。pattern と一致した文字列のキャプチャバッファを作成します。キャプチャされた一致の位置と長さを取得するには、CALL PRXPOSN を使用します。キャプチャバッファの値を取得するには、PRXPOSN 関数を使用します。かっこと一致させるには、"\("または"\)"を使用します。

最長一致と最短一致の繰り返し要素

Perl 正規表現は、"最長一致"の繰り返し要素と"最短一致"の繰り返し要素をサポートしています。繰り返し要素は、その繰り返し要素が特定の開始位置から文字列と一致する回数が可能な限り多い場合に最長一致と見なされます。また、繰り返し要素が文字列と一致する必要のある回数が最小の場合は最短一致と見なされます。繰り返し要素を最短一致に指定するには、?を 繰り返し要素の末尾に追加します。デフォルトでは、繰り返し要素は最長一致と見なされます。

次の表に、最長一致の繰り返し要素を示します。表内の繰り返し要素の説明には、その要素の使用例が含まれています。

表 A1.4 最長一致の繰り返し要素

メタ文字	説明
*	直前の副次式と 0 回以上一致します。 • zo*は"z"および"zoo"と一致します。 • *は{0,}と同じ意味です。
+	直前の副次式と 1 回以上一致します。 • "zo+"は"zo"および"zoo"と一致します。 • "zo+"は"z"とは一致しません。 • +は{1,}と同じ意味です。
?	直前の副次式と 0 回または 1 回一致します。 • "do(es)?"は"do"または"does"の"do"と一致します。 • ?は{0,1}と同じ意味です。
{n}	*n* 回以上一致します。
{n,}	*n* 回以上パターンと一致します。

メタ文字	説明
{n,m}	m および n は負でない整数です。n<=m です。*n* 回～*m* 回一致します。 • "o{1,3}"は"fooooood"の最初の 3 つの o と一致します。 • "o{0,1}"は"o?"と同じ意味です。 カンマと数の間には空白を入れません。

次の表に、最短一致の繰り返しメタ文字を示します。

表 A1.5 *最短一致の繰り返し要素*

メタ文字	説明
*?	0 回以上パターンと一致します。
+?	1 回以上パターンと一致します。
??	0 回または 1 回パターンと一致します。
{n}?	*n* 回完全に一致します。
{n,}?	*n* 回以上パターンと一致します。
{n,m}?	*n* 回～*m* 回パターンと一致します。

クラスのグループ化

次の表に、文字クラスのグループ化を示します。文字をかっこで囲んでこれらのクラスを指定します。これらのメタ文字は、共通のプロパティセットを共有します。処理が正常に行われるには、文字クラスが常に文字と一致する必要があります。否定文字クラスは、かっこ内で指定された文字のリストに含まれない文字と常に一致する必要があります。表内のメタ文字の説明には、そのメタ文字の使用例が含まれています。

表 A1.6 *文字クラスのグループ化*

メタ文字	説明
[...]	かっこで囲まれた文字のいずれかと一致する文字セットを指定します。 • "[abc]"は"plain"の"a"と一致します。
[^...]	かっこで囲まれていない文字と一致する負の文字セットを指定します。 • "[^abc]"は"plain"の"p"と一致します。
[a-z]	文字の範囲を指定します。その範囲内の文字と一致します。 • "[a-z]"は"a"～"z"の範囲の小文字の英字と一致します。

メタ文字	説明
[^a-z]	文字の範囲を指定します。その範囲外の文字と一致します。 • "[^a-z]"は"a" ~ "z"の範囲外の文字と一致します。
[[:alpha:]]	英字と一致します。
[[:^alpha:]]	英字以外の文字と一致します。
[[:alnum:]]	英数字と一致します。
[[:^alnum:]]	英数字以外の文字と一致します。
[[:ascii:]]	ASCII 文字と一致します。[\0–\177]と同じ意味です。
[[:^ascii:]]	非 ASCII 文字と一致します。[^\0–\177]と同じ意味です。
[[:blank:]]	空白文字と一致します。
[[:^blank:]]	空白以外の文字と一致します。
[[:cntrl:]]	制御文字と一致します。
[[:^cntrl:]]	制御文字以外の文字と一致します。
[[:digit:]]	数字と一致します。\d と同じ意味です。
[[:^digit:]]	数字以外の文字と一致します。\D と同じ意味です。
[[:graph:]]	空白文字以外の表示可能な文字です。[[:alnum:][:punct:]]と同じ意味です。
[[:^graph:]]	表示可能な文字ではありません。[^[:alnum:][:punct:]]と同じ意味です。
[[:lower:]]	小文字と一致します。
[[:^lower:]]	小文字と一致しません。
[[:print:]]	文字列を印刷します。
[[:^print:]]	文字列を印刷しません。
[[:punct:]]	句読点または表示可能な文字(空白および英数字以外)と一致します。
[[:^punct:]]	句読点または表示可能な文字(空白および英数字以外)と一致しません。
[[:space:]]	空白と一致します。\s と同じ意味です。
[[:^space:]]	空白と一致しません。\S と同じ意味です。
[[:upper:]]	大文字と一致します。
[[:^upper:]]	大文字と一致しません。
[[:word:]]	ワードと一致します。\w と同じ意味です。
[[:^word:]]	ワードと一致しません。\W と同じ意味です。
[[:xdigit:]]	16 進文字と一致します。
[[:^xdigit:]]	16 進文字と一致しません。

先読みと後読みの動作

先読みと後読みは、特定のテキストが存在するかどうかを確認するために一致の前または後ろを検索する方法です。先読みまたは後読みで検索されたテキストは、最終的な一致には含まれません。たとえば、末尾が"Jr."の名前を検索し、一致には"Jr."が含まれないようにするには、正規表現/.*(?=Jr\.)/を使用します。値"John Wainright Jr."の場合、この正規表現は"Jr."の前にある"John Wainright"を一致と見なします。

表 A1.7 先読みと後読みの動作

メタ文字	説明
(?=...)	ゼロ幅の正の先読みアサーションを指定します。たとえば、正規表現 *regex1 (?=regex2)*では、*regex1* と *regex2* の両方が一致する場合に一致と見なされます。*regex2* は最終的な一致には含まれません。
(?!...)	ゼロ幅の負の先読みアサーションを指定します。たとえば、正規表現 *regex1 (?!regex2)*では、*regex1* が一致して *regex2* は一致しない場合に一致と見なされます。*regex2* は最終的な一致には含まれません。
(?<=...)	ゼロ幅の正の後読みアサーションを指定します。たとえば、正規表現*(?<=regex1) regex2* では、*regex1* と *regex2* の両方が一致する場合に一致と見なされます。*regex1* は最終的な一致には含まれません。固定幅の後読みのみで動作します。
(?<!...)	ゼロ幅の負の後読みアサーションを指定します。固定幅の後読みのみで動作します。

コメントとインライン修飾子

この表のメタ文字には、かっこ内の最初の要素として疑問符が含まれています。疑問符に続く文字は拡張子です。

表 A1.8 コメントとインライン修飾子

メタ文字	説明
(?#text)	テキストが無視されるコメントを指定します。
(?imsx)	1 つ以上の埋め込まれたパターン照合修飾子を指定します。パターンが大文字と小文字を区別しない場合、パターンの先頭に(?i)を使用できます。たとえば、**$pattern=″(?i)foobar″;**と指定します。ハイフン(-)の後の文字では修飾子がオフになります。

結合演算子を使用した最適な条件の選択

前述の表に示した基本的な正規表現(\a や\w など)は、入力文字列内の指定した位置で多くても 1 つの部分文字列のみと一致します。ただし、一般的な正規表現内で結合を行う演算子では、基本的なメタ文字を組み合わせてより複雑なパターンを作成できます。あいまいな状況では、これらの演算子で最適な一致または不適切な一致を判断できます。最適な一致が常に選択されます。

表 A1.9 結合演算子を使用した最適な一致

メタ文字	説明
ST	次の例では、AB と A'B'を指定します。A と A'は S によって一致する部分文字列で、B と B'は T によって一致する部分文字列です。 • S にとって A'よりも A の方が適切な一致の場合、AB が A'B'よりも適切な一致になります。 • A と A'が同じで T にとって B'よりも B の方が適切な一致の場合、AB が AB'よりも適切な一致になります。
S|T	S が一致する場合、T のみが一致する場合よりも適切な一致であることを指定します。S に対する 2 つの一致の順序は、S の順序と同じです。同様に、T に対する 2 つの一致の順序は、T の順序と同じです。
S{repeat-count}	SSS . . . S (必要な回数分繰り返す)と一致します。
S{min,max}	S{max}|S{max-1}| . . . |S{min+1}|S{min}と一致します。
S{min,max}?	S{min}|S{min+1}| . . . |S{max-1}|S{max}と一致します。
S?, S*, S+	それぞれ S{0,1}、S{0, big-number}、S{1,big-number}と同じ意味です。
S??, S*?, S+	それぞれ S{0,1}?、S{0, big-number}?、S{1,big-number}?と同じ意味です。
(?=S)、(?<=S)	S に最適な一致を考慮します (これは、S にキャプチャのかっこが含まれ、正規表現全体の他の場所で前方参照が使用されている場合にのみ重要です)。
(?!S)、(?<!S)	このグループ化演算子では、S が一致するかどうかのみが重要なため、順序を記述する必要はありません。

キーワード

C

さ

な

は