SAS® Cloud Analytic Services: CASL リファレンス

2024.01*

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

SAS® ドキュメント
2024 年 3 月 15 日

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2021. *SAS® Cloud Analytic Services: CASL リファレンス*. Cary, NC: SAS Institute Inc.

SAS® Cloud Analytic Services: CASL リファレンス

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

January 2024

v_002-P1:proccas

目次

1

CASL プログラミングについて

CAS 言語について

CASL は、SAS クライアントおよびその他のクライアントが SAS Cloud Analytic Services (CAS)と対話するために使用する言語仕様です。

CAS 言語(CASL)の特徴は次のとおりです。

- CASL はステートメントベースの言語です。
- CASL は動的型付け言語です。[1]
- この言語では大文字と小文字は区別されません。
- CASL は、次の長所を備えたスクリプト言語です。
  - アクションの実行
  - 結果の処理
  - 分析パイプラインの開発
  - ユーザー定義アクションを使用して CAS でコードを実行
- ステートメントに、アクションや関数の名前などのキーワードを含めることができます。
- ステートメントに式を含めることができます。
- ステートメントはセミコロン(;)で終了します。
- 1 つの PROC CAS ステップに、複数の CASL プログラムを含めることができます。

CASL の使用法をいくつか次に示します。

- 分析パイプラインを開発する

1. 変数のデータ型は、実行時に SAS によって割り当てられます。変数を使用する前に、特定の型の変数を宣言する必要はありません。

- アクションを使用して CAS サーバーに要求をサブミットし、処理を行ってから結果を返す
- アクションによって返された結果を評価および操作する
- アクションの引数を作成する
- ユーザー定義のアクションと関数を作成する

CASL プログラムの実行

SAS で CASL 言語を使用するには、次のものが必要です。

- CAS セッション。CAS ステートメントは、サーバー上の既存のセッションに接続するか、サーバー上で新しいセッションを開始します。セッションの詳細については、*"セッション" (SAS Cloud Analytic Servcies: 基本)*を参照してください。

  > ヒント　CAS アクションをサブミットしない場合は、CAS セッションは必要ありません。

- CAS プロシジャ。PROC CAS を使用すると、SAS でサーバーと対話する CASL プログラミングステートメントを解釈できます。

CASL プログラムは、次の方法でサブミットできます。

- SAS ウィンドウ環境または SAS Studio で CAS プロシジャを使用します。
- CAS サーバー上で、ユーザー定義アクションによるサーバー側処理を使用します。これにより、SAS、Python、Lua、R のいずれかから CASL プログラムを実行できます。サーバー側処理の詳細については、*"ユーザー定義アクションの作成" (SAS Cloud Analytic Services: CASL プログラマガイド)*を参照してください。

2

CAS プロシジャ

概要: CAS プロシジャ

CAS プロシジャについて

CAS プロシジャを使用すると、CASL 言語仕様に基づいたプログラミング環境を提供することにより、SAS クライアントから SAS Cloud Analytic Services (CAS)と対話できます。このプログラミング環境では、CAS アクションを実行し、その結果を使用して別のアクションのパラメーターを準備できます。

CASL 言語は動的に型付けされます。変数の型は、使用前に宣言する必要はありません。変数は、任意の型の値を保持できます。値は必要に応じて型間で自動的に変換されますが、ある型から別の型に手動でキャストすることもできます。

CAS プロシジャには、次の操作を実行できるいくつかの機能があります。

- サーバーでサポートされている CAS アクションを実行します。
- 新しいアクションセットをサーバーにロードします。
- 複数のセッションを使用して、非同期実行を実行します。
- 関数式パーサーを使用して、パラメーターおよび結果を変数として操作します。

用語

アクション
: ユーザーの要求に応じてサーバーによって実行されるタスク。ユーザーは、アクションを実行するための要求をサーバーに送信します。アクションには、入力引数用のパブリッシュ済み API があります。引数は辞書としてサーバーに送信されます。サーバーはアクションを実行し、アクションが成功したかどうかを示すステータスとともに結果の辞書を返送します。

アクションセット
: セッション管理やテーブル管理などの機能をグループ化するアクション(タスク)のコレクション。

引数
: パラメーターの実際の値が関数に渡されます。

条件
: 評価結果がブール値になる 1 つ以上の数値式または文字式。

式
: ルールと優先順位を使用して評価され、結果を計算する 1 つ以上の値、定数、変数、演算子、関数の組み合わせ。

関数
: 特定のタスクを実行して結果を返すステートメントのグループ。関数にアクセスするには、関数のエントリポイントを呼び出し、関数に指定された API の引数を指定します。

パラメーター
: サブルーチンへの入力として提供されるデータの一部を参照するためにサブルーチンで使用される変数の一種。サブルーチンに送られる実際の情報は、*引数*と呼ばれます。

結果テーブル
: CAS テーブルの転送可能なメモリ表現。結果テーブルまたは結果テーブルグループをアクションへの応答として送信することも、CASL 内で結果テーブルを最初から作成することもできます。結果テーブルには、行、列、ラベル、データ型、属性が含まれます。

CAS セッション
: セッションは、サーバーにログオンしたユーザーを表します。その後、セッションを使用してアクションを実行し、結果を生成できます。

変数
: 式で使用できる値のシンボリック名。値は、リスト、辞書、または単純なデータ型(文字列、整数、または浮動小数点数)にすることができます。プログラム全体で値を割り当てることができます。CASL には、グローバル変数、ローカル変数、およびパラメーター変数があります。

CASL ステートメントの概要

CASL ステートメントは、キーワード、識別子、および演算子を含む、アイテムが連続したものです。CASL ステートメントは、次のアクションを実行できます。

- 変数の割り当ての実行
- プログラムコントロールへの影響
- データの入出力の実行
- 関数の作成と呼び出し
- アクションの呼び出しと結果テーブルの処理
- CASL プログラムの動作の指定または変更
- 式を使用した、CASL ステートメントのパラメーターの作成

構文: CAS プロシジャ

注: 浮動小数点例外(FPE)が発生した場合、CASL は **CASL reset after signal** というメッセージをログに出力し、実行を続行します。

ヒント: CAS プロシジャで実行できるアクションについては、*名前と製品順の SAS Viya プラットフォームアクションとアクションセット*を参照してください。

```
PROC CAS;
   ACTION< action-set-name.action-name>< RESULT=<variable>><
     STATUS=<rc>>< ASYNC=name>
     /<parameters>;
   <ASSIGNMENT> target = expression;
   CALLfunction (argument1, argument2...);
   CONTINUE;
   DEFAULTRESULT=variable-expressionSTATUS=variable-expression |
     CLEAR<clear-option-1<clear-option-2<...clear-option-N>>>;
   DELETEvalue;
   DESCRIBE variable;
   DO;
      ... more CASL statements ...;
      END;
   DO UNTIL
      condition;
      ... more CASL statements ...;
```

```
    END;
DO WHILEcondition;
    ... more CASL statements ...;
    END;
DO [<key,>] <var>OVER<value>;
    ... more CASL statements ...;
    END;
DO<var>=<start>[TO<stop>] [BY<increment>]
    [WHILE<condition> | UNTIL<condition>];
    ... more CASL statements ...;
    END;
ENDEXTERNALSOURCE;
ENDSOURCE;
EXITexpression;
EXTERNALSOURCEvariable-expression;
  <embedded-text>
  ENDEXTERNALSOURCE;
FILEfileref;
FNC<category-name>;
FORMATvariable-1...variable-Nformat;
FUNCTION function name ( [argument 1 [, argument 2...] ] );
    ... more CASL statements ...;
    END;
FUNCTIONLIST<name>;
    FLIST<name>;
GLOBALvariable;
GOTO label;
IF condition
    THEN statement;
    ELSE <statement>;
Labelstatement
LOCALvariable;
LEAVE;
LOADACTIONSETactionsetname;
NULL;
OUTPUT <ODS | LOG>;
PRINTvalue-1...value-N;
PRINTLISTexpression;
RETURN<expression | value>;
SAVERESULTvariable-name<NOREPLACE>
  <DATAOUT=<libref.> data-set-name> |<LIB=libref>|
```

```
    <FILE=<pathname> | <filename>>
    <<CASLIB=casref> | <CASOUT=name>>;
  SELECT< (select-expression)>;
    WHEN-1 < (when-expression-1 ..., when-expression-n> ) statement;
      <...WHEN-n (when-expression-1 ..., when-expression-n)>statement;
    <OTHERWISE statement>;END;
  SESSION session-reference;
  SET (DISP<ECHOEXTSRC><LOGS><LOGS=DEBUG | NOTE | TRACE |
    WARN><STDJSON><WARNINGDUP><USEWIDTH>) ;
  SOURCE variable; text; endsource;
  UNSETDISP<ECHOEXTSRC><LOGS><STDJSON><USEWIDTH
    ><WARNINGDUP>
  UPLOAD
    PATH=path-to-file
    <CASOUT={output-table-options>}
    <IMPORTOPTIONS={FILETYPE= "BASESAS" | "CSV" | "DTA" | "ESP" |
      "EXCEL" | "FMT" | "EXCEL" | "HDAT" | "JMP" | "LASR" | "SPSS" |
      "XLSX",fileType-specific-parameters}>;
  RUN;
QUIT;
```

ステートメント	タスク
"ACTION ステートメント"	SAS Cloud Analytic Services アクションを実行します。
"Assignment ステートメント"	式を評価し、結果を変数に保存します。
"CALL ステートメント"	指定された引数で関数を呼び出します。関数が値を返す場合、無視されます。
"CONTINUE ステートメント"	現在の DO ループの反復の処理を停止し、次の反復から再開します。
"DEFAULT ステートメント"	CASL が特定のステートメントを処理する方法を構成できます。
"DELETE ステートメント"	指定した値をメモリから削除します。
"DESCRIBE ステートメント"	CASL 変数と式の構造とデータ型を表示できます。
"DO ステートメント"	1 つのステートメントとして実行されるコードのブロックを作成します。
"DO ステートメント: 反復"	インデックス変数の値に基づいて、DO ステートメントと END ステートメントの間のステートメントを繰り返し実行します。
"DO OVER ステートメント"	配列、辞書、またはテーブルを反復処理します。

ステートメント	タスク
“DO UNTIL ステートメント”	条件が真になるまで、DO ループ内のステートメントを繰り返し実行します。
“DO WHILE ステートメント”	条件が真の間、DO ループ内のステートメントを繰り返し実行します。
“END ステートメント”	DO グループ、FUNCTION、または SELECT グループの処理を終了します。
“EXTERNALSOURCE ステートメント”	EXTERNALSOURCE ステートメントの終わりを示します。
“ENDSOURCE ステートメント”	SOURCE ステートメントの終わりを示します。
“EXIT ステートメント”	CASL コードの現在のブロックを終了します。
“EXTERNALSOURCE ステートメント”	プログラムにテキストを埋め込み、特定の変数に割り当てることができます。
“FILE ステートメント”	別の出力先を指定できます。
“FNC ステートメント”	すべての 共通関数を、その簡単な説明を付けて、名前とカテゴリに基づいてリストします。
“FORMAT ステートメント”	出力形式を単一の変数または変数のリストに関連付けます。
“FUNCTION ステートメント”	式で呼び出すことができる新しい関数を作成します。
“FUNCTIONLIST ステートメント”	すべての CASL ビルドイン関数とユーザー定義関数を、簡単な説明を付けて、名前に基づいてリストします。
“GLOBAL ステートメント”	グローバルスコープを持つ変数を宣言します。
“GOTO ステートメント”	実行をラベル付きステートメントにすぐに送ります。
“IF ステートメント”	式の値に基づいて、ステートメントを条件付きで実行します。
“Label ステートメント”	GOTO ステートメントのラベルターゲットを指定します。セミコロンは必要ありません。
“LEAVE ステートメント”	現在のループの処理を停止し、シーケンス内の次のステートメントから再開します。
“LOADACTIONSET ステートメント”	アクションセットをロードします。
“LOCAL ステートメント”	ローカルスコープを持つ変数を宣言します。
“OUTPUT ステートメント”	アクティブな出力場所を設定します。
“PROC CAS ステートメント”	SAS クライアントから SAS Cloud Analytic Services アクションをプログラムおよびスケジュールできます。

ステートメント	タスク
“PRINT ステートメント”	定数、変数、および式の値を現在の出力場所に書き込みます。
“PRINTLST ステートメント”	アプリケーションで定義されたリストファイルに出力します。値がリストの場合、リストは横向きで、各アイテムが表示されます。アイテムがテーブルの 場合は、現在の出力場所に出力されます。
“RETURN ステートメント”	現在の関数から値を返します。
“RUN ステートメント”	入力済みの SAS ステートメントを実行します。
“SAVERESULT ステートメント”	指定された形式で結果テーブルを変数に保存します。
“SELECT ステートメント”	複数のステートメントのいずれか、またはステートメントグループを実行します。
“SESSION ステートメント”	プログラムによって起動されるアクションに使用するセッションを指定します。
“SET ステートメント”	アクションによって SAS ログに書き込まれるメッセージを制御します。
“SOURCE ステートメント”	プログラムにテキストを埋め込み、特定の変数に割り当てることができます。
“UNSET ステートメント”	CAS アクションの一部の動作を変更するオプションをオフにし、SAS ログ内の対応するメッセージを抑制します。
“UPLOAD ステートメント”	SAS クライアントからサーバーにファイルを転送します。データ転送後、サーバーはデータをインメモリテーブルにロードします。

PROC CAS ステートメント

SAS クライアントから SAS Cloud Analytic Services アクションをプログラムおよびスケジュールできます。

ヒント: CAS プロシジャで実行できるアクションについては、*名前と製品順の SAS Viya プラットフォームアクションとアクションセット*を参照してください。

構文

PROC CAS;

引数なし

PROC CAS には、必須やオプションの引数はありません。PROC CAS を実行すると、次のいずれかに到達するまで実行が継続されます。

- DATA ステップ
- 別の PROC ステップ
- QUIT ステートメント
- プロシジャの進行を妨げるエラー状態。

その時点で、前述の構文を使用して PROC CAS を再度実行する必要があります。

詳細

注: グローバルステートメント、SAS マクロコード、および RUN ステートメントによって、プロシジャは終了されません。

ACTION ステートメント

SAS Cloud Analytic Services アクションを実行します。

注: 式を使用して、ACTION ステートメントのパラメーターを作成できます。

参照項目: ACTION ステートメントで実行できるアクションのドキュメントについては、*SAS Viya プラットフォーム: システムプログラミングガイド*および *SAS Visual Analytics: プログラミングガイド*を参照してください。

例: “例 2: アクションの実行” (81 ページ)

構文

<ACTION><*action-set-name.*>*action-name*< RESULT= <*variable*> < STATUS = <*rc*>
< ASYNC = *name*>
<*parameters*>;

必須引数

<*action-set-name.*>*action-name*
実行するアクションを指定します。

action-set-name
実行するアクションを含むアクションセットの名前を指定します。たとえば、**table**、**simple**、**network** などです。

action-name
実行するアクションの名前を指定します。たとえば、**loadTable**、**listActions**、**serverStatus** などです。

例 たとえば、次のステートメントでは、テーブルアクションセットの loadTable アクションが実行されます。
```
action table.loadTable / path="cholesterol.csv";
run;
```
この例では、PATH=は loadTable アクションのパラメーターです。

オプション引数

parameter
指定されたアクションのパラメーター。help アクションを使用して、アクションのパラメーターを確認できます。

例
```
action help / action="action-name";
run;
```

RESULT=*variable-name*
アクションの結果を変数に保存します。その後、他の CASL ステートメントとアクションで変数を使用できます。この変数は、DESCRIBE などのステートメントや、CAS プロシジャ呼び出しの範囲内で選択された SAS ステートメントセットで使用できます。

別名 R=

操作 RESULT=引数を指定しない場合、結果テーブルを返すアクションはデフォルトの出力先に出力されます。他のデータ型を返すアクションの場合、結果は SAS ログに出力されます。

参照項目: プログラムで結果を使用する例については、“結果の操作” (*SAS Viya プラットフォーム: システムプログラミングガイド*)を参照してください。

STATUS=<*variable-name*>
アクションのステータスコードを変数に保存します。ステータスコードを調べつことで、アクション完了時のエラーの有無を判断できます。*variable-name* を指定しなかった場合は、ステータスコードを含む**_status** という名前の変数が作成されます。

参照項目: ステータスコードの詳細については、“重要度コード、ステータスコード、理由コード” (*SAS Viya プラットフォーム: システムプログラミングガイド*)を参照してください。

ASYNC=*result name*
サーバーに複数の要求をサブミットできます。要求が同じセッションにある場合、要求は受信した順にキューに入れられます。要求が別々のセッションにある場合、要求は並列実行されます。

例: テーブルアクションセットを使用したテーブルのロードとテーブル情報の表示

この例では、サーバー側ロードを実行する table.loadTable アクションを使用します。

```
/* change the host and port to match your site */
options cashost="cloud.example.com" casport=5570;

/* start a session, if you do not already have one */
*cas casauto;

proc cas;
  session casauto;

  /* Load source data (IRIS) into a table. */
  table.loadTable /              /* #1 */
    path="iris.sashdat"
    casout={name="iris"};
run;

  table.tableInfo   / table="iris";  /* #2 */
  table.tableDetails / table="iris"; /* #3 */
quit;
```

1 table.loadTable アクションを使用して、caslib のデータソースからテーブルをロードします。詳細については、loadTable アクションを参照してください。

2 table.tableInfo アクションを使用して、テーブルに関する情報を表示します。詳細については、tableInfo アクションを参照してください。

3 table.tableDetails アクションを使用して、テーブルに関する詳細情報を取得します。詳細については、tableDetails アクションを参照してください。

Assignment ステートメント

式を評価し、結果を変数に保存します。

カテゴリ: ローカル

ヒント: 変数がすでに存在する場合、新しい割り当てによって変数が置換され、古い値が上書きされます。

構文

variable = *expression*;

必須引数

variable
新規または既存の変数の名前を指定します。

expression
ルールと優先順位を使用して評価され、結果を計算する 1 つ以上の値、定数、変数、演算子、関数の組み合わせ。CASL 式の詳細については、"CASL 式" (*SAS Cloud Analytic Services: CASL プログラマガイド*)を参照してください。

ヒント 式には、等号の左側で使用される変数を含めることができます。ステートメントの両側に変数が現れると、右側の元の値が式の評価に使用され、結果は等号の左側の変数に保存されます。

詳細

割り当てステートメントでは、等号の右側にある式を評価し、等号の左側に指定されている変数に結果を保存します。

変数は、任意のデータ型の値を参照できます。ある割り当てステートメントでは値を変数に割り当て、別の割り当てステートメントでは同じ変数に異なるデータ型の値を割り当てることができます。割り当てステートメントが **a = b = 5** の場合、そのステートメントは **a = (b = 5)**です。ここで、**b = 5** は比較式です。

例

例 1: 有効な割り当てステートメント

この割り当てステートメントは、ターゲット変数が文字列値"Jane Smith"を参照するものとして定義しています。

```
Name = 'Jane Smith';
```

この割り当てステートメントは、値が"88"のターゲット変数を定義しています。

```
xx.y = 88;
```

このターゲットステートメントは、MIN 関数の結果を変数 *z* に割り当てます。

```
z = min(y, 70);
```

例 2: 割り当てステートメントの使用

次の例では、割り当てステートメントを使用して、値が"1"のターゲット変数を定義します。

```
proc cas;
 x.y[2].z.a["abc"].t[10]=1;
run;
```

DESCRIBE ステートメントを使用して、データの構造を表示できます。

```
describe x
```

DESCRIBE ステートメントでは、配列と辞書の完全な構造が示されます。*x* は、2 つのエントリを持つ配列 *y* を含む辞書です。次に例を示します。

a[“abc”]

これは次の例と同じように機能します。

a.abc

どちらの方法でも辞書が作成されます。

```
dictionary ( 1 entries, 1 used);
 [y] array ( 2 entries, 1 used);
  [ 2] dictionary ( 1 entries, 1 used);
   [z] dictionary ( 1 entries, 1 used);
    [a] dictionary ( 1 entries, 1 used);
     [abc] dictionary ( 1 entries, 1 used);
      [t] array ( 10 entries, 1 used);
       [10] int64_t;
```

CALL ステートメント

指定された引数で関数を呼び出します。関数が値を返す場合、無視されます。

構文

<CALL> *function* (<*argument-1*, *argument-2*...>);

オプション引数

function
ビルトイン関数またはユーザー定義関数の名前。

argument
呼び出し時に関数に渡される値を指定します。

関連項目

“FUNCTION ステートメント”

CONTINUE ステートメント

現在の DO ループの反復の処理を停止し、次の反復から再開します。

要件 CONTINUE ステートメントは、反復 DO ループのステートメントリストにのみ記述できます(例: DO *i*=、DO WHILE、または DO UNTIL)。

構文

CONTINUE;

引数なし

CONTINUE ステートメントに引数はありません。DO ループの次の反復でステートメントの処理を再開します。

比較

- CONTINUE ステートメントは、DO ステートメントの現在の反復の処理を停止し、現在の DO ステートメントの次の反復でプログラムの実行を再開します。
- LEAVE ステートメントは、現在の DO ステートメントの処理を停止し、現在の DO ステートメントの外部でプログラムの実行を再開します。

例

この例は、CONTINUE ステートメントの使用法を示しています。*x* が *y* と等しい場合は、ログに **GOOD** が出力され、それ以外の場合は、**ERROR** が出力され、**t** の値が返されて、1 が加算されます。DO ループが反復され、**test** の増分値が出力されます。**test** が 10 に等しくなると、CONTINUE ステートメントは、実行を DO ループの次の反復にジャンプさせ、**test** が出力されないようにします。

```
proc cas;
  function evaltest( x, y, t);
   if (x = y) then put "GOOD " t;
    else           put "ERROR " t;

   return( t+1);
  end;

test = 1;

  do i = 1 to 10 by 1;

   continue;
   put i;

  end;

test = evaltest( i, 11,test);

   put i;
```

```
run;
```

次の行が SAS ログに書き込まれます。

```
GOOD 1
11
```

関連項目

- DO
- “DO ステートメント: 反復”
- “DO OVER ステートメント”
- “DO UNTIL ステートメント”
- “DO WHILE ステートメント”
- “LEAVE ステートメント”

DEFAULT ステートメント

CASL がアクションの結果とアクションの処理ステータスを返す方法を制御できます。

構文

形式 1:

DEFAULTRESULT=*variable-expression*;

形式 2:

DEFAULTSTATUS=*variable-expression*;

形式 3:

DEFAULTCLEAR<*clear-option-1*<*clear-option-2*<...*clear-option-N*>>>

引数

注:

- 形式 1 および 2 の DEFAULT ステートメントには、RESULT=または STATUS=のいずれかのリテラル引数が少なくとも 1 つ必要です。DEFAULT ステートメントだけを指定することはできません。ただし、RESULT=引数と STATUS=引数の両方を 1 つの DEFAULT ステートメントに含めることができます。

```
default result=r1;
default status=s1;
default result=r1 status=s2;
```

default;を指定すると、ログに WARNING が記録され、DEFAULT ステートメントは無視されます。

- 形式 3 の DEFAULT ステートメントには、少なくとも 1 つの引数(RESULT 引数または STATUS 引数、あるいはその両方)も必要です。ただし、形式 1 および 2 の引数(RESULT=および STATUS=)を含めることはできません。

```
default clear result;
default clear status;
default clear result status;
```

たとえば、

```
default clear;
```

または

```
default clear result status=s1;
```

を指定すると、ログに WARNING が記録され、DEFAULT ステートメントは無視されます。

CLEAR
アクションの結果とステータスが保存されるデフォルトの変数割り当てを削除します。

(clear-options)
アクションの結果、ステータス、または結果とステータスの両方に対するデフォルトの変数割り当てを削除します。

RESULT
アクションの RESULT が保存されるデフォルトの変数割り当てを削除します。

例 default clear result;

default clear status;

default clear result status;

STATUS
アクションのステータスが保存されるデフォルトの変数割り当てを削除します。

例 default clear results status;

例 “例: デフォルトの結果変数とステータス変数の設定とクリア”

RESULT=*variable-expression*
アクションの結果が保存されるデフォルト変数を設定します。*variable-expression* は、値を保存できる変数またはデータ構造要素のいずれかになる任意の式です。

別名 R

例 default result=myRes;

STATUS=*variable-expression*
アクションのステータスが保存されるデフォルト変数を設定します。*variable-expression* は、値を保存できる変数またはデータ構造要素のいずれかになる任意の式です。

例 default status=myStat;

例 “例: デフォルトの結果変数とステータス変数の設定とクリア”

例: デフォルトの結果変数とステータス変数の設定とクリア

次の例では、DEFAULT ステートメントを使用して、アクションのステータス変数と結果変数のデフォルト値を作成します。次に、CLEAR ステートメントを使用してデフォルト値をクリアします。

```
proc cas;
  builtins.help status = s1 result = r1 / action = "serverStatus"; /* 1 */
run;
  default status = d["s2"] result = d["r2"];        /* 2 */
  builtins.help / action = "serverStatus";
  print d["s2"] d["r2"];                          /* 3 */
  describe d["s2"] d["r2"];                         /* 4 */
run;
  default clear status result;                       /* 5 */
  builtins.help / action = "serverStatus";
run;
quit;
```

1 builtins.help アクションを実行し、status=および result=の変数割り当てを手動で指定します。status=と result=のデフォルトは、単純な変数名にすることも、配列要素や辞書エントリなどのより複雑なものにすることもできます。この例では、辞書キー s2 と r2 は、辞書 d のエントリを指定します。

2 DEFAULT ステートメントに STATUS=と RESULT=を指定して、デフォルト値をそれぞれ s2 と r2 に設定します。builtins.help アクションを指定すると、STATUS=および RESULT=オプションは、DEFAULT ステートメントで指定された変数に自動的に設定されます。また、builtins.help アクションは、結果をログに出力します。

3 DESCRIBE ステートメントを指定して、辞書キー s2 および r2 のデータ型を表示します。

4 PRINT ステートメントを指定して、結果とステータスを出力します。

5 DEFAULT CLEAR ステートメントに RESULT オプションおよび STATUS オプションを指定して、デフォルト値をクリアします。

アウトプット 2.1　デフォルトの結果変数とステータス変数の設定とクリアの出力

```
{severity=0,reason=0,,statusCode=0} {serverStatus=TRUE}
dictionary ( 4 entries, 4 used);
 [severity] int64_t;
 [reason] int64_t;
 [status] nil;
 [statusCode] int64_t;
dictionary ( 1 entries, 1 used);
 [serverStatus] boolean;
{serverStatus=TRUE}
```

DELETE ステートメント

指定した値をメモリから削除します。

構文

DELETE *value*

必須引数

value
削除する値の場所の定義。値は、変数、リスト、配列、辞書のいずれかにすることができます。

詳細

指定した値が変数の場合、その変数はメモリから削除されます。指定した値が配列または辞書内に含まれている場合、その値はメモリから削除されます。指定した値が配列、辞書、またはリストの場合、配列、辞書、またはリスト全体とその中の値が削除されます。

注: 配列から値を削除する場合、CAS はその値の位置を削除しません。n 個の値を持つ配列は、1 つまたは複数の値を削除すると、n 個の位置を持つ疎な配列になります。

例

次の例では、DELETE ステートメントを使用してさまざまなデータ値を削除します。データが削除されると、PRINT ステートメントは、変数が初期化されておらず、欠損値に設定されているという警告を返します。

```
*cas casauto;

proc cas;

/* Create your sample data */

  x=25;
  arr={1,5,25};
  dict = {John=90, Mary=92, Sally=89};

  print "x=" x "arr=" arr "dict=" dict;

/* Use the DELETE statement to delete your data */

  delete x;
  print "x=" x;

  delete arr[2];
  print "arr=" arr;
  print "dimension of arr="dim(arr);

  delete arr;
  print "arr=" arr;

  delete dict.John;
  print "dict=" dict;
  print "dimension of dict=" dim(dict);

  delete dict;
  print "dict=" dict;

run;
quit;
```

```
x=25 arr={1,5,25} dict={John=90,Mary=92,Sally=89}
WARNING: Variable 'x' is uninitialized. It has been set to missing.
x=.
arr={1,,25}
3
WARNING: Variable 'arr' is uninitialized. It has been set to missing.
arr=.
dimension of arr=1
dict={Mary=92,Sally=89}
dimension of dict=2
WARNING: Variable 'dict' is uninitialized. It has been set to missing.
dict=.
```

DESCRIBE ステートメント

CASL 変数と式の構造とデータ型を表示できます。

操作: 結果が変数に保存されると、DESCRIBE ステートメントは変数の説明を SAS ログに書き込みます。説明には、変数の深さを表示するために、展開された配列と辞書が表示されます。

構文

DESCRIBE *variable-name* | *expression*;

必須引数

variable-name
構造とデータ型を表示する変数の名前。

expression
ルールと優先順位を使用して評価され、結果を計算する 1 つ以上の値、定数、変数、演算子、関数の組み合わせ。

例

例 1: DESCRIBE ステートメントの使用

次の例では、serverStatus アクションの結果を *r* という名前の変数に保存します。

```
proc cas;
  action serverStatus result=r;
  describe r;
run;
```

DESCRIBE ステートメントは、前記のアクションの結果をリストとして表示します。リストには、**About** (リスト)、**Server** (テーブル)、および **Nodestatus** (テーブル)という 3 つのエントリが含まれています。リストの最初のエントリである **About** 内に、**System** という名前の別のリストがあります。

```
List ( 3 entries, 3 used);
[About] List( 6 entries, 6 used);
[CAS] String;
[Version] String;
[Built] String;
[Copyright] String;
[System] List ( 6 entries, 6 used);
[Hostname] String;
[OS Name] String;
[OS Family] String;
[OS Version] String;
[OS Release] String;
[Model Number] String;
[Documentation] String;
[server] Table( [1] Rows [2] columns
Column Names:
[1] Node Count (Double)
[2] Total Actions (Double)
[nodestatus] Table( [143] Rows [5] columns
Column Names:
[1] Node Name (String)
[2] Role (String)
[3] Uptime (Sec) (Double)
[4] Running (Double)
[5] Stalled (Double)
```

DESCRIBE ステートメントを使用して serverStatus 結果の構造について学習すると、任意の CASL プログラムからその情報にアクセスできます。

例 2: 前の割り当て変数を使用した結果の生成

前の例では、変数 *r* に 3 つのエントリを持つリストを保存しました。エントリの 1 つを使用して、データを出力できます。

```
print r.server[1];
run;
```

出力では、CAS サーバーに関する情報が提供されます。

```
{nodes=143 , actions=17}
```

例 3: 式に対する DESCRIBE ステートメントの使用

式に対して DESCRIBE ステートメントを使用することができます。次の例では、DESCRIBE ステートメントは、評価された後の式を記述します。

```
proc cas;
 describe ((1500*0.09)+1500)*1;
run;
```

SAS ログには、評価された式のデータ型が表示されます。

例のコード 2.1　SAS ログ

```
double;
```

関連項目

- “CASL データ型” (*SAS Cloud Analytic Services: CASL プログラマガイド*)
- “CASL 式” (*SAS Cloud Analytic Services: CASL プログラマガイド*)
- “CASL 結果テーブル” (*SAS Cloud Analytic Services: CASL プログラマガイド*)
- “CASL 変数” (*SAS Cloud Analytic Services: CASL プログラマガイド*)
- “DESCRIBE ステートメントの使用” (*SAS Cloud Analytic Services: CASL プログラマガイド*)

DO ステートメント

1 つのステートメントとして実行されるコードのブロックを作成します。

要件　DO ステートメントとすべての DO グループ処理ステートメントの後に END ステートメントを指定する必要があります。

例:　“例 3: ユーザー定義関数の作成” (82 ページ)

構文

DO;

... more CASL statements ...;

END;

詳細

DO ステートメントと END ステートメントの間のステートメントは、DO グループと呼ばれます。DO ステートメントは、DO グループ処理の最も単純な形式です。DO ステートメントと END ステートメントの間のステートメントは、DO グループと呼ばれます。DO グループ内に DO ステートメントをネストできます。単純な DO ステートメントは、IF-THEN/ELSE ステートメント内でよく使用されます。“IF ステートメント”を参照してください。DO ステートメントは、IF 条件が真か偽かに応じて実行されます。

例: DO ステートメントの使用

この単純な DO グループでは、DO と END の間のステートメントは、YEARS が 7 より大きい場合にのみ実行されます。YEARS が 7 以下の場合は、ELSE ステートメントの後の DO グループ内のステートメントが実行されます。

```
if year > 7 then
  do;
   months = years*12;
   put years= months=;
  end;
else;
  do;
   yrsleft=7-years;
  end;
```

関連項目

- “CONTINUE ステートメント”
- “END ステートメント”
- “LEAVE ステートメント”

DO ステートメント: 反復

インデックス変数の値に基づいて、DO ステートメントと END ステートメントの間のステートメントを繰り返し実行します。

要件
: DO ステートメントとすべての DO グループ処理ステートメントの後に END ステートメントを指定する必要があります。

ヒント:
: オプションの TO 句と BY 句の順序は逆にすることができます。
複数の指定を使用する場合、各指定は実行前に評価されます。

構文

DO <*variable*> = <*start*> TO<*stop*><BY *increment*> <WHILE (*condition*) | UNTIL (*condition*)>;

... *more CASL statements* ...;

END;

オプション引数

variable
: DO グループの実行を制御する値を持つ変数を指定します。

start
: 変数の初期値を指定します。

  start と stop の両方が存在する場合、インデックス変数の値が stop の値を超えるまで、実行は(increment の値に基づいて)継続します。start と increment のみが存在する場合、ステートメントが実行をループ外に向けるまで、または DO ステートメントで指定された WHILE または UNTIL 式が満たされるまで、実行は(increment の値に基づいて)継続します。stop も increment も指定されていない場合、グループは start の値に従って実行されます。ループの最初の実行の前に、stop の値が評価されます。

stop
: インデックス変数の終了値を指定します。

BY *increment*
: 正数または負数(または数を生成する式)を指定して、インデックス変数の増分を制御します。

  increment の値は、ループの実行前に評価されます。DO グループ内で行われる increment の変更は、反復回数に影響しません。increment が指定されていない場合、インデックス変数は 1 ずつ増加します。increment が正数の場合、start はループの下限、stop は(存在する場合)ループの上限にする必要があります。increment が負数の場合、start はループの上限、stop は(存在する場合)ループの下限にする必要があります。

WHILE (*condition*) | UNTIL(*condition*)
DO グループの実行前または実行後に、指定した条件を評価します。条件はかっこで囲みます。

WHILE 式はループの実行前に毎回評価されるため、グループ内のステートメントは、式が真である間は繰り返し実行されます。UNTIL 式はループの実行後に毎回評価されるため、グループ内のステートメントは、式が真になるまで繰り返し実行されます。

制限事項 WHILE または UNTIL の指定は、配置されている句の最後のアイテムのみに影響します。

詳細

CASL は、DO グループ処理用のステートメントをいくつかサポートしています。

次の表に、CASL で使用できるさまざまな種類の DO ステートメントを示します。

ステートメント	説明	構文
“DO ステートメント: 反復” (p. 25)	DO グループ処理の最も単純な形式である DO ステートメントは、通常は IF-THEN/ELSE ステートメントの一部として実行されるステートメントのグループを 1 単位として指定します。IF-THEN/ELSE ステートメントの詳細については、IF-THEN/ELSE ステートメントを参照してください。	**DO** <*variable*> = <*start*> TO <*stop*><BY *increment*> <WHILE (*condition*) | UNTIL (*condition*)>; *... more CASL statements ...*; **END**;
“DO OVER ステートメント” (p. 27)	DO OVER ステートメントは、配列、辞書、またはテーブルを反復処理します。	**DO** <*key*>, <*var*> **OVER** <*value*>; *... more CASL statements ...;* **END**;
“DO UNTIL ステートメント” (p. 30)	DO UNTIL ステートメントは、条件が真になるまで DO ループ内のステートメントを繰り返し実行し、DO ループの反復後に毎回条件をチェックします。	**DO UNTIL** (*condition*); *... more CASL statements ...;* **END**;
“DO WHILE ステートメント” (p. 32)	DO WHILE ステートメントは、条件が真の間、DO	**DO WHILE** (*condition*);

ステートメント	説明	構文
	ループ内のステートメントを繰り返し実行し、DO ループの反復前に毎回条件をチェックします。	*... more CASL statements ...;* **END**;

LEAVE ステートメントを使用すると、現在の反復の処理を停止し、シーケンス内の次のステートメントから再開できます。“CONTINUE ステートメント”を使用すると、現在の反復の処理を停止し、次の反復から再開できます。

例: Do Loop ステートメントの使用

このステートメントは、ループ変数を **i** として識別します。

```
do i=1 to 10;
```

このステートメントは、増分 3 でループ変数を実行します。

```
do i=1 to 10 by 3;
do i=1 to 10 while( i < 5);
do i=1 to 10 by 1 until ( i>=5);
```

関連項目

- “CONTINUE ステートメント”
- “END ステートメント”
- “LEAVE ステートメント”

DO OVER ステートメント

配列、辞書、またはテーブルを反復処理します。

要件 DO ステートメントとすべての DO グループ処理ステートメントの後に END ステートメントを指定する必要があります。

例: “例 4: ランタイム数学関数とインライン演算子の使用” (84 ページ)

構文

DO <*key*>, <*var*> **OVER** <*value*>;

... more CASL statements ...;

END;

オプション引数

key
辞書要素のキー値が割り当てられる変数を指定します。

var
配列または辞書要素のキーに関連付けられた値が割り当てられる変数を指定します。

value
配列、辞書またはテーブルを指定します。

詳細

CASL は、DO グループ処理用のステートメントをいくつかサポートしています。

次の表に、CASL で使用できるさまざまな種類の DO ステートメントを示します。

ステートメント	説明	構文
“DO ステートメント: 反復” (p. 25)	DO グループ処理の最も単純な形式である DO ステートメントは、通常は IF-THEN/ELSE ステートメントの一部として実行されるステートメントのグループを 1 単位として指定します。IF-THEN/ELSE ステートメントの詳細については、IF-THEN/ELSE ステートメントを参照してください。	**DO** <*variable*> = <*start*> TO <*stop*><BY *increment*> <WHILE (*condition*) | UNTIL (*condition*)>; *... more CASL statements ...*; **END**;
“DO OVER ステートメント” (p. 27)	DO OVER ステートメントは、配列、辞書、またはテーブルを反復処理します。	**DO** <*key*>, <*var*> **OVER** <*value*>; *... more CASL statements ...;* **END**;
“DO UNTIL ステートメント” (p. 30)	DO UNTIL ステートメントは、条件が真になるまで DO ループ内のステートメントを繰り返し実行し、DO ループの反復後に毎回条件をチェックします。	**DO UNTIL** (*condition*); *... more CASL statements ...;* **END**;
“DO WHILE ステートメント” (p. 32)	DO WHILE ステートメントは、条件が真の間、DO ループ内のステートメン	**DO WHILE** (*condition*); *... more CASL statements ...;*

ステートメント	説明	構文
	トを繰り返し実行し、DO ループの反復前に毎回条件をチェックします。	**END**;

LEAVE ステートメントを使用すると、現在の反復の処理を停止し、シーケンス内の次のステートメントから再開できます。"CONTINUE ステートメント"を使用すると、現在の反復の処理を停止し、次の反復から再開できます。

例: DO OVER ステートメントの使用

次の例では、アクションの結果を取得し、その結果を反復処理して、欠損値が 20 以下の変数の名前を配列に挿入します。その配列は、別のアクションへの入力として使用できます。完全な例については、"例" (*SAS Viya プラットフォーム: システムプログラミングガイド*)を参照してください。

```
action simple.summary result=CPSSum                    /* 1 */
  table={caslib="casdata", name="phoneSubs"};
run;
  cinfo = findTable(CPSSum);                            /* 2 */
   nmissVar = {"Country Name", "Indicator Name"};       /* 3 */
 do col over cInfo;                                     /* 4 */
 if (col.NMiss <20) then do;                            /* 5 */
   nmissVar= nmissVar + col.Column;                     /* 6 */
 end;
 end;
print nmissVar;                                         /* 7 */
run;
```

1 Summary アクションでは、合計、平均、欠損値の数などの記述統計量が作成されます。RESULT=オプションでは、結果が CPSSum という名前の変数に保存されます。詳細については、summary アクションを参照してください。

2 cinfo は、FINDTABLE 関数の結果を含む変数の名前です。FINDTABLE は、テーブル内の値を検索するビルトイン関数です。CPSSum は、Summary アクションの結果を保持する変数です。詳細については、FINDTABLE 関数を参照してください。

3 nmissVar は、プログラムの結果を保持する変数の名前です。中かっこ{ }は、リストが作成されることを示します。"Country Name"と"Indicator Name"は、配列に含まれる列です。DO OVER ループの本文には、割り当てステートメントと +演算子を使用して nmissVar 配列に値を追加するコードが含まれています。詳細については、割り当てステートメントを参照してください。

4 DO OVER ステートメントは、変数 cInfo に保存されている結果を反復処理します。col は、配列のインデックスの名前です。詳細については、DO OVER ステートメントを参照してください。

5 IF-THEN DO ステートメントでは、NMiss の値が 20 より大きい列が検索されます。詳細については、IF-THEN/ELSE ステートメントを参照してください。

6 次に、変数 nmissVar の配列に列が追加されます。

7 PRINT ステートメントでは、変数 nmissVar が SAS ログに出力されます。詳細については、PRINT ステートメントを参照してください。

例のコード 2.2 SAS ログ

```
{Country Name,Indicator Name,1990,1995,1996,1997,1998,1999,2000,
2001,2002,2003,2004,2005,2006,2007,2008,2009,2010,2011,2012,2013,2014}
```

関連項目

- “CONTINUE ステートメント”
- “END ステートメント”
- “LEAVE ステートメント”

DO UNTIL ステートメント

条件が真になるまで、DO ループ内のステートメントを繰り返し実行します。

要件 DO ステートメントとすべての DO グループ処理ステートメントの後に END ステートメントを指定する必要があります。

構文

DO UNTIL (*condition*);

... more CASL statements ...;

END;

必須引数

condition
ブール値の結果となる 1 つの数値または文字式の評価を指定します。その結果をもとに、次にどのコードパスを実行するかを決定します。条件は、ループの末尾で実行されます。これは、ループの本体が常に最低 1 回は実行されることを意味しています。式評価が真であれば、ループの反復を停止します。

詳細

CASL は、DO グループ処理用のステートメントをいくつかサポートしています。

次の表に、CASL で使用できるさまざまな種類の DO ステートメントを示します。

ステートメント	説明	構文
“DO ステートメント: 反復” (p. 25)	DO グループ処理の最も単純な形式である DO ステートメントは、通常は IF-THEN/ELSE ステートメントの一部として実行されるステートメントのグループを 1 単位として指定します。IF-THEN/ELSE ステートメントの詳細については、IF-THEN/ELSE ステートメントを参照してください。	**DO** <*variable*> = <*start*> TO <*stop*><BY *increment*> <WHILE (*condition*) | UNTIL (*condition*)>; *... more CASL statements ...;* **END**;
“DO OVER ステートメント” (p. 27)	DO OVER ステートメントは、配列、辞書、またはテーブルを反復処理します。	**DO** <*key*>, <*var*> **OVER** <*value*>; *... more CASL statements ...;* **END**;
“DO UNTIL ステートメント” (p. 30)	DO UNTIL ステートメントは、条件が真になるまで DO ループ内のステートメントを繰り返し実行し、DO ループの反復後に毎回条件をチェックします。	**DO UNTIL** (*condition*); *... more CASL statements ...;* **END**;
“DO WHILE ステートメント” (p. 32)	DO WHILE ステートメントは、条件が真の間、DO ループ内のステートメントを繰り返し実行し、DO ループの反復前に毎回条件をチェックします。	**DO WHILE** (*condition*); *... more CASL statements ...;* **END**;

LEAVE ステートメントを使用すると、現在の反復の処理を停止し、シーケンス内の次のステートメントから再開できます。“CONTINUE ステートメント”を使用すると、現在の反復の処理を停止し、次の反復から再開できます。

例: DO UNTIL ステートメントを使用してループを繰り返す

このステートメントでは、**x** が 10 を超えるまでループが繰り返されます。式 *x>10* は、ループの末尾で評価されます。全部で 10 回の反復があります(0、1、2、3、4、5、6、7、8、9)。

```
x=0;
 do until (x>10);
print x;
x = x+1;
 end;
```

関連項目

- “CONTINUE ステートメント”
- “END ステートメント”
- “LEAVE ステートメント”

DO WHILE ステートメント

条件が真の間、DO ループ内のステートメントを繰り返し実行します。

制限事項: DO ステートメントとすべての DO グループ処理ステートメントの後に END ステートメントを指定する必要があります。

参照項目: “DO ステートメント: 反復” (25 ページ)

構文

DO WHILE (*condition*);

... more CASL statements ...;

END;

必須引数

condition

ブール値の結果となる 1 つの数値または文字式の評価を指定します。その結果をもとに、次にどのコードパスを実行するかを決定します。条件は、ループの本体を実行する前に評価されます。これは、ループの本体が実行されない可能性があることを意味します。

詳細

CASL は、DO グループ処理用のステートメントをいくつかサポートしています。

次の表に、CASL で使用できるさまざまな種類の DO ステートメントを示します。

ステートメント	説明	構文
“DO ステートメント: 反復” (p. 25)	DO グループ処理の最も単純な形式である DO ステートメントは、通常は IF-THEN/ELSE ステートメントの一部として実行されるステートメントのグループを 1 単位として指定します。IF-THEN/ELSE ステートメントの詳細については、IF-THEN/ELSE ステートメントを参照してください。	**DO** <*variable*> = <*start*> TO <*stop*><BY *increment*> <WHILE (*condition*) | UNTIL (*condition*)>; *... more CASL statements ...;* **END**;
“DO OVER ステートメント” (p. 27)	DO OVER ステートメントは、配列、辞書、またはテーブルを反復処理します。	**DO** <*key*>, <*var*> **OVER** <*value*>; *... more CASL statements ...;* **END**;
“DO UNTIL ステートメント” (p. 30)	DO UNTIL ステートメントは、条件が真になるまで DO ループ内のステートメントを繰り返し実行し、DO ループの反復後に毎回条件をチェックします。	**DO UNTIL** (*condition*); *... more CASL statements ...;* **END**;
“DO WHILE ステートメント” (p. 32)	DO WHILE ステートメントは、条件が真の間、DO ループ内のステートメントを繰り返し実行し、DO ループの反復前に毎回条件をチェックします。	**DO WHILE** (*condition*); *... more CASL statements ...;* **END**;

LEAVE ステートメントを使用すると、現在の反復の処理を停止し、シーケンス内の次のステートメントから再開できます。“CONTINUE ステートメント”を使用すると、現在の反復の処理を停止し、次の反復から再開できます。

例: DO WHILE ステートメントの使用

これらのステートメントでは、**x** が 10 より小さい間、ループが繰り返されます。式 *x < 10* は、ループの先頭で評価されます。全部で 10 回の反復があります(0、1、2、3、4、5、6、7、8、9)。

```
x=0;
```

```
 do while (x < 10);
print x;
x = x + 1;
 end;
```

関連項目

- “CONTINUE ステートメント”
- “END ステートメント”
- “LEAVE ステートメント”

ELSE ステートメント

IF ステートメントで指定した式の結果が真の値の場合は、ELSE ステートメントの直後のステートメントが実行されます。

参照項目: “IF ステートメント”

構文

ELSE *statement*;

END ステートメント

DO グループ、FUNCTION、または SELECT グループの処理を終了します。

要件 END ステートメントは、DO グループ、FUNCTION、または SELECT グループの最後のステートメントにする必要があります。

構文

END;

引数なし

END ステートメントを使用して、DO グループ、FUNCTION、または SELECT グループの処理を終了します。

関連項目

ステートメント

- “DO ステートメント”
- “DO ステートメント: 反復”
- “DO OVER ステートメント”
- “DO UNTIL ステートメント”
- “DO WHILE ステートメント”
- “FUNCTION ステートメント”
- “SELECT ステートメント”

ENDEXTERNALSOURCE ステートメント

EXTERNALSOURCE ステートメントの終わりを示します。

構文

ENDEXTERNALSOURCE;

引数なし

ENDEXTERNALSOURCE ステートメントには、引数やパラメーターはありません。

詳細

- EXTERNALSOURCE テキストブロックを終了するために PROC CAS 内で ENDEXTERNALSOURCE ステートメントを使用する場合、単独で 1 行に表示し、その後にセミコロンステートメントターミネータを付ける必要があります。“例 1: EXTERNALSOURCE ステートメントの使用”を参照してください。
- ENDEXTERNALSOURCE ステートメントと同じ行のセミコロンに続くテキストはすべて無視されます。“例 1: EXTERNALSOURCE ステートメントの使用”を参照してください。
- EXTERNALSOURCE ステートメントが EXECUTE 関数または runCasl アクションによって処理される場合、EXTERNALSOURCE ステートメントおよび ENDEXTERNALSOURCE ステートメントは、埋め込みテキストと別の行にする必要はありません。どちらのステートメントもセミコロンで終了する必要があります。“例 2: EXECUTE 関数での EXTERNALSOURCE ステートメントの使用”を参照してください。

関連項目

- “SOURCE ステートメント”
- “EXTERNALSOURCE ステートメント”

ENDSOURCE ステートメント

SOURCE ステートメントの終わりを示します。

制限事項: SOURCE ステートメントと ENDSOURCE ステートメントはネストできません。

構文

ENDSOURCE;

引数なし

SOURCE ステートメントには、引数やパラメーターはありません。

詳細

EXTERNALSOURCE テキストブロックを終了するために ENDEXTERNALSOURCE が PROC CAS 内のトップレベルステートメントとして使用される場合、単独で 1 行に表示し、その後にセミコロンステートメントターミネータを付ける必要があります。同じ行のセミコロンに続くテキストはすべて 無視されます。

ENDEXTERNALSOURCE ステートメントが EXECUTE 関数または CAS サーバーのいずれかによって処理される場合、sccasl.runCasl アクションを実行するときに、ステートメントを別の行にする必要はありませんが、セミコロンが後に続く必要があります。セミコロンに続くテキストは、CAS 言語コードとして解釈されます。

関連項目

“SOURCE ステートメント”

EXIT ステートメント

CASL コードの現在のブロックを終了します。

構文

EXIT *expression*;

必須引数

expression
ルールと優先順位を使用して評価され、結果を計算する 1 つ以上の値、定数、変数、演算子、関数の組み合わせ。

EXTERNALSOURCE ステートメント

テキストをコードのブロックとしてプログラムに埋め込み、そのコードを特定の変数に割り当てることができます。

制限事項: EXTERNALSOURCE ステートメントと ENDSOURCE ステートメントはネストできません。

要件 コードブロックを終了するには、"ENDEXTERNALSOURCE ステートメント"が必要です。

参照項目: "ENDEXTERNALSOURCE ステートメント"

構文

EXTERNALSOURCE*variable-expression*;
<*embedded-text*>
ENDEXTERNALSOURCE;

必須引数

variable-expression
埋め込みテキストを割り当てる変数、またはテキストが割り当てられる変数を生成する式を指定します。

ENDEXTERNALSOURCE;
EXTERNALSOURCE ステートメントの終わりを指定します。

オプション引数

embedded-text
値の文字列表現を指定します。

詳細

EXTERNALSOURCE ステートメントは、埋め込みテキストの内容とフォーマットの両方を保持します。埋め込みテキストが、Python などの空白に敏感なプログラミング言語で記述されたコードである場合は、EXTERNALSOURCE ステートメントを使用する必要があります。

このステートメントは、改行が保持されず、追加の空白の挿入が必要になる可能性がある、引用符で囲まれたテキスト文字列のコード埋め込みよりも望ましい方法です。

EXTERNALSOURCE ステートメントの使用は、Python などの言語に限定されず、IML、CMP、DS2、DATA ステップ、SQL、CASL など、SAS Viya 内の他の言語にも使用できます。

EXTERNALSOURCE ステートメントに関するその他の考慮事項を次に示します。

- EXTERNALSOURCE ステートメントは、場合によっては追加の空白の挿入が必要になる可能性がある SOURCE ステートメント, よりも望ましい場合があります。
- EXECUTE 関数と一緒に使用しない限り、EXTERNALSOURCE ステートメントは開始ステートメントとして単独で 1 行に表示し、その後にセミコロンを付ける必要があります。“例 1: EXTERNALSOURCE ステートメントの使用”を参照してください。
- 埋め込みテキストは、EXTERNALSOURCE ステートメントとセミコロンに続く新しい行から開始する必要があります。埋め込みテキストを引用符で囲んだり、セミコロンで終わらせたりする必要はありません。“例 1: EXTERNALSOURCE ステートメントの使用”を参照してください。
- 埋め込みテキストを指定した後、ENDEXTERNALSOURCE ステートメントを指定してコードブロックを終了する必要があります。ENDEXTERNALSOURCE ステートメントは、新しい行に単独で指定する必要があります。ENDEXTERNALSOURCE;の後に、ENDEXTERNALSOURCE ステートメントと同じ行に指定されたテキストはすべて 無視されます。“例 1: EXTERNALSOURCE ステートメントの使用”を参照してください。
- EXTERNALSOURCE ステートメントが EXECUTE 関数または runCasl アクションによって処理される場合、EXTERNALSOURCE ステートメントおよび ENDEXTERNALSOURCE ステートメントは、埋め込みテキストと別の行にする必要はありません。ただし、ステートメントはセミコロンで終了する必要があります。“例 1: EXTERNALSOURCE ステートメントの使用”を参照してください。

比較

EXTERNALSOURCE ステートメントは、次の場合を除き、SOURCE ステートメントと似ています。

- EXTERNALSOURCE ステートメントでは、空白と改行が保持されます。
- 埋め込みテキストは、指定された文字列に変更されずに保存されます。テキストは、フォーマットを変更または追加する必要なく、*そのまま*保存されます。SOURCE ステートメントでは、埋め込まれたテキストは改行を保持しないため、追加の空白またはその他のフォーマットの挿入が必要になる場合があります。

- マクロ処理は EXTERNALSOURCE ステートメントではサポートされておらず、マクロ、マクロ変数、およびマクロ関数への参照もサポートされていません。マクロ処理は、SOURCE ステートメントでサポートされています。
- デフォルトでは、EXTERNALSOURCE ステートメントコードブロックに埋め込まれたテキストは、SAS ログにエコーされません。SOURCE ステートメントコードブロックに埋め込まれたテキストは、デフォルトで SAS ログにエコーされます。“例 1: EXTERNALSOURCE ステートメントの使用”を参照してください。

注: ECHOEXTSRC オプションを SET ステートメントで指定すると、SAS が EXTERNALSOURCE ステートメントコードブロックに埋め込まれたテキストを SAS ログにエコーできるようになります。例については、“例 1: EXTERNALSOURCE ステートメントの使用”を参照してください。

例

例 1: EXTERNALSOURCE ステートメントの使用

次の例では、EXTERNALSOURCE ステートメントを使用して IML コードを変数に保存し、iml アクションを使用して IML コードを実行します。

```
proc cas;
  action loadactionset / actionset='iml';      /* 1 */
  set echoextsrc;                              /* 2 */
  externalsource pgm;                          /* 3 */
    a = 123;
    x =/* embedded comment */ 3;               /* 4 */
    y=a-x;
  endexternalsource; print "This text is ignored";     /* 5 */
  print "This statement is part of regular CASL code";  /* 6 */
  run;
  iml / code=pgm;                              /* 7 */
  run;
quit;
```

1 iml アクションセットをロードします。

2 ECHOEXTSRC オプションを SET ステートメントで指定して、SAS が埋め込みテキストを SAS ログにエコーできるようにします。

3 EXTERNALSOURCE ステートメントを指定し、その後に変数名 pgm とセミコロンを続けます。このステートメントは、単独で 1 行に指定します。

4 変数 pgm に保存する埋め込みテキストを指定します。埋め込みテキストを引用符で囲んだり、埋め込みテキスト内のステートメントをセミコロンで終了したりする必要はありません。

5 新しい行に ENDEXTERNALSOURCE ステートメントを(単独で)指定して、コードブロックを終了します。ENDEXTERNALSOURCE ステートメントの直後にセミコロンを付ける必要があります。ENDEXTERNALSOURCE ステートメントと同じ行でセミコロンの後に指定されたテキストは無視されます。

6 ENDEXTERNALSOURCE ステートメントの後の新しい行に追加の CASL コードを指定します。

7 変数 pgm を iml アクションの Code=パラメーターに指定して、変数 pgm に含まれるコードを実行します。

アウトプット 2.2　EXTERNALSOURCE ステートメントを使用した場合のログ出力

```
82   proc cas;
83      action loadactionset / actionset='iml';
84      set echoextsrc;
85      externalsource pgm;
86         a = 123;
87         x =/* embedded comment */ 3;
88         y=a-x;
89      endexternalsource; print "This text is ignored";      /*  */
90      print "This statement is part of regular CASL code";  /*  */
91      run;
NOTE: Active Session now CASAUTO.
NOTE: Added action set 'iml'.
{actionset=iml}
This statement is part of regular CASL code
92      iml / code=pgm;                  /*  */
93      run;
94   quit;
NOTE: PROCEDURE CAS used (Total process time):
      real time           0.22 seconds
      cpu time            0.00 seconds
```

例 2: EXECUTE 関数での EXTERNALSOURCE ステートメントの使用

次の例は、EXECUTE 関数で EXTERNALSOURCE ステートメントを使用する方法を示しています。

```
proc cas;
  execute( "print 'this is my program'; externalsource mystring; 10*5 = 50; i*2=x
endexternalsource;" );
run;
  print mystring;
run;
quit;
```

アウトプット 2.3　EXECUTE 関数で使用される EXTERNALSOURCE ステートメントのログ出力

```
12
13    proc cas;
14       execute( "print 'this is my program'; externalsource mystring; 10*5 =
50; i*2=x endexternalsource;" );
15    run;
this is my program
16       print mystring;
17    run;
10*5 = 50; i*2=x
18    quit;
```

関連項目

- “ENDEXTERNALSOURCE ステートメント”
- “SOURCE ステートメント”

FILE ステートメント

別の出力先を指定できます。

デフォルト: アプリケーションによって定義されます。

例: “例 5: ディシジョンツリーモデルの学習とデータのスコアリング” (85 ページ)

構文

FILE “*fileref*” ;

必須引数

fileref
fileref は次のいずれかになります。

- **ods** では、必要に応じて、ODS 出力でデフォルトの出力場所を使用します。
- **log** では、デフォルトの出力場所を使用しますが、ODS は使用しません。内部出力形式ルーチンを使用し、必要に応じて SAS ログに出力します。
- <*name*>では、名前の付いたファイル参照名の場所を出力に使用します。
- <*path*>では、与えられたパスを出力の場所として使用します。

FILE ステートメントを使用して、アクティブな出力場所を指定ファイルに変更することができます。各 *fileref* には名前があります。この名前を使用すれば、アクティブなファイルをいつでも別の *fileref* に変更することができます。

例: FILE ステートメントで出力場所を変更する

```
file tablefile "/u/user/table.lst"
print table;
file ods;                    /* #1 */
file nodes "/u/user/nodes";      /* #2 */
action listnodes result=r;
print r;                     /* #3 */
file log;                    /* #4 */
print r;                     /* #5 */
file ods;
```

1 アクティブな出力場所を ODS に設定します。

2 アクティブな出力場所を**/u/user/nodes** に設定します。
3 listnodes アクションの結果は、**/u/user/nodes** に出力されます。
4 アクティブな出力場所を SAS ログに設定します。
5 同じ結果が SAS ログに出力されます。

関連項目

“OUTPUT ステートメント”

FNC ステートメント

すべての共通関数を、その簡単な説明を付けて、名前とカテゴリに基づいてリストします。

構文

FNC *category-name*;

必須引数

category-name
関数を表示するカテゴリの名前を指定します。カテゴリ名のリストについては、“例: すべての共通関数とそのカテゴリを表示”を参照してください。

例: すべての共通関数とそのカテゴリを表示

```
proc cas;
fnc cate;
run;
```

例のコード 2.3 SAS ログ

```
bitwise
char
combinatorial
datetime
distance
financial
math
probability
quantile
random
special
statistics
trig
truncation
```

関連項目

- “CASL ビルトイン関数”
- “共通関数”
- “FUNCTION ステートメント”
- “FUNCTIONLIST ステートメント”

FORMAT ステートメント

出力形式を単一の変数または変数のリストに関連付けます。

ヒント: 変数や出力形式のリストを複数指定できます。

構文

FORMAT *variable-1...variable-N format*;

詳細

出力形式を削除する

出力形式を削除するには、最後に出力形式を指定せずに変数のリストを指定します。次の例では、*Todaysdate*、*Tdate*、*Tod*、*DateT* は、現在、それぞれに出力形式が関連付けられている変数です。次の FORMAT ステートメントには出力形式の指定がないため、変数から関連する出力形式が削除されます。

```
format todaysdate tdate tod datet;
```

結果テーブル内の列に出力形式を適用する

結果テーブル内の列に出力形式を適用することができます。次の例では、結果テーブルには、*Cylinders*、*Wheels*、*MSRP* の 3 つの列が含まれており、これをフォーマットする必要があります。*Cylinders* と *Wheel* は Best5.で、*MSRP* 列は Best8.でフォーマットされています。

> ヒント　各出力形式の末尾には、出力形式名の後にピリオド"."があることを確認してください。

```
format table "cylinders" "wheels" best5. "msrp" best8.;
```

例: 出力形式の適用

次の例では、変数 *Todaysdate*、*Tdate* および *DateT* に出力形式 Date12.を、変数 *Tod* に出力形式 Time12.を適用しています。

```
proc cas;
format todaysdate tdate datet date12. tod TIME12.;
run;
```

FUNCTION ステートメント

式で呼び出すことができる新しい関数を作成します。

デフォルト: return ステートメントが指定されていない場合、デフォルトの戻り値はゼロになります。

要件 END ステートメントを FUNCTION ステートメントの後に配置する必要があります。

例: “例 3: ユーザー定義関数の作成” (82 ページ)

構文

FUNCTION *function- name* (<*parameter–1*><, *parameter–2*><, ..., *parameter–n*>);
    statements;
END;

必須引数

function- name
    ユーザー定義関数名を指定します。

statement
    一連の CASL ステートメントです。

オプション引数

parameter
    呼び出し時に関数に渡される引数値を保持する変数名を指定します。

関連項目

- “CALL ステートメント”
- “END ステートメント”
- “FUNCTIONLIST ステートメント”

FUNCTIONLIST ステートメント

すべての CASL ビルトイン関数とユーザー定義関数を、簡単な説明を付けて、名前に基づいてリストします。

構文

FUNCTIONLIST <*name*>;

FLIST <*name*>;

オプション引数

名前が指定されていない場合は、使用可能な関数が SAS ログに出力されます。

name
情報をリストする関数の名前を指定します。

次の名前を使用して、使用可能な関数を確認することができます。

- **user** は、ユーザー定義関数を示します。
- <*name*>は、インポートされた関数を拡張子名で示します。

GLOBAL ステートメント

CASL プログラム内のどこからでもアクセスして使用できる変数を作成します。

構文

GLOBAL *variable*;

必須引数

variable
変数名を指定します。

詳細

変数のスコープには、ローカルまたはグローバルがあります。ローカル変数は、その変数が作成された関数内にのみ存在します。グローバル変数は、CASL プログラム

内のどこからでもアクセスして 使用できます。LOCAL ステートメントを使用すると、関数のローカル変数を明示的に作成できます。また、GLOBAL ステートメントを使用すると、グローバル変数を明示的に作成できます。

注意
LOCAL ステートメントを使用して関数内にローカル変数を作成しない場合、その変数は暗黙的にグローバルになります。同じ名前のグローバル変数が存在する場合は、グローバル変数が参照されます。 関数内に変数を作成するときは、LOCAL ステートメントを使用することをお勧めします。

- グローバル変数は、CASL プログラム内のどこからでもアクセスして 使用できます。
- グローバル変数を明示的に作成するには、GLOBAL ステートメントを使用します。
- 変数が関数の外部で作成された場合、変数は暗黙的にグローバルになります。
- 変数が関数内で作成された際、その変数がコントロールループ変数でない場合や LOCAL ステートメントによって作成されていない場合、その変数は暗黙的にグローバルになります。
- グローバル変数は、PROC CAS の QUIT ステートメントがサブミットされるまで保持されます。
- LOCAL ステートメントは、変数が作成される関数のローカル変数を明示的に作成しました。LOCAL ステートメントを使用して作成された変数は、その関数内にのみ存在します。
- 関数内で使用されるループ反復変数は、暗黙的にローカルスコープを持ち、その関数内にのみ存在します。
- ローカル変数は、関数が返されるまで保持されます。

例: グローバル変数の指定

次の例では、x という名前の 2 つのグローバル変数を作成します。この関数内で作成された変数はグローバルスコープで作成されるため、同じ名前の変数が上書きされます。

```
proc cas;
  function myFunction( y );
    global x;          1
    print "NOTE: When myFunction begins execution, x=" x " and y=" y;
    x=y;
  end func;
run;
  x=5;               2
  print "NOTE: Before myFunction execution, x=" x;
  myFunction(2);
  print "NOTE: After myFunction execution, x=" x;
run;
quit;
```

1 GLOBAL ステートメントは、グローバルスコープを持つ x という名前の変数を作成します。
2 この関数の外で作成された x 変数もグローバルです。

例のコード 2.4 SAS ログ

```
NOTE: Before myFunction execution, x=5
NOTE: When myFunction begins execution, x=5  and y=2
NOTE: After myFunction execution, x=2
```

関連項目

- *"変数スコープ" (SAS Cloud Analytic Services: CASL プログラマガイド)*
- "LOCAL ステートメント"

GOTO ステートメント

実行をラベル付きステートメントにすぐに送ります。

構文

GOTO *label*;

必須引数

label
GOTO 移動先を識別するステートメントラベルを指定します。ステートメントラベルは、Label ステートメントを使用して定義します。

詳細

GOTO ステートメントの移動先ラベルは、同じ CASL メソッド内にある必要があります。label 引数の指定が必要で、指定しないとエラーが発生します。ステートメントラベルは、Label ステートメントを使用して定義します。

比較

GOTO ステートメントは、多くの場合、DO-END や IF-THEN/ELSE のプログラミングロジックで置き換えることができます。

例: GOTO ステートメントの使用

次の例では、x はゼロに初期化されます。DO WHILE ループが実行され、ループするたびに x が 1 ずつ増分されます。x=5 の場合、プログラムは GOTO ステートメントを実行し、DO WHILE ループを終了します。GOTO ステートメントを使用すると、DO WHILE ループを早期に(x<10 条件が満たされる前に)終了できます。GOTO ステートメントは、null の Label ステートメント done を参照しています。

```
proc cas;
  x=0;
  do while (x < 10);
    print x;
    x = x + 1;
    if (x=5) then do;
      put "x now equals 5 x= " x;
      goto done;
    end;              /* end the IF statement */
  end;                /* end the DO WHILE loop */
  done:               /* specify a null label that GOTO can refer to */
  print "finished, x should equal 5 x= " x;
run;
quit;
```

関連項目

- “DO ステートメント”
- “DO ステートメント: 反復”
- “DO OVER ステートメント”
- “DO UNTIL ステートメント”
- “DO WHILE ステートメント”
- “IF ステートメント”
- “Label ステートメント”
- “LEAVE ステートメント”

IF ステートメント

式の値に基づいて、ステートメントを条件付きで実行します。

注: 式の結果が偽の値で、ELSE ステートメントがある場合は、ELSE ステートメントの直後のステートメントが実行されます。

ヒント: 効率を高めるには、確率を下げるように IF-THEN/ELSE ステートメントを作成します。

参照項目: “ELSE ステートメント” (34 ページ)

例: “例 3: ユーザー定義関数の作成” (82 ページ)

構文

IF *condition*THEN *statement*;

ELSE *statement*;

必須引数

THEN
式の結果が真の値の場合は、THEN 引数の直後のステートメントが実行されます。

condition
次に実行するステートメントを決定します。

statement
任意の CASL ステートメントを指定します。

詳細

CAS では、IF-THEN ステートメントの式が評価されて、ゼロ以外、ゼロ、または欠損のいずれかの結果が生成されます。ゼロ以外の値であれば、その式は真になります。結果がゼロまたは欠損であれば、その式は偽になります。

式の結果が真の値の場合は、THEN 引数の直後のステートメントが実行されます。式の結果が偽の値で、ELSE ステートメントがある場合は、ELSE の直後のステートメントが実行されます。ELSE 引数がない場合は、IF ステートメントの後のステートメントが実行されます。

注: IF 式を使用すると、条件式の評価が真か偽かに基づいて 2 つの値から選択することができます。さらに、IF 式をネストして、多方向の決定のために多くの値から選択することができます。

比較

相互に排他的な条件が長く連続する場合は、IF-THEN ステートメントを連続で使用するかわりに SELECT グループを使用します。詳細については、SELECT ステートメントを参照してください。

例: IF-THEN/ELSE ステートメントの指定

この例では、IF-THEN/ELSE ステートメントで、Grade1、Grade2、Grade3 がクラス平均の 80 以上であるかどうかを示しています。

```
proc cas;
  grade1=90;
  grade2=89;
  grade3=74;
  if (grade1>=80) then put 'Grade 1 is equal to or above class average.';
   else put 'Grade 1 is less than 80;
  if (grade2=80) then put 'Grade 2 is equal to or above class average.';
   else put 'Grade 2 is less than 80;
  if (grade3>=80) then put 'Grade 3 is equal to or above class average.';
   else put 'Grade 3 is less than 80;
run;
quit;
```

例のコード 2.5　SAS ログ

```
Grade 1 is equal to or above class average.
Grade 2 is equal to or above class average.
Grade 3 is less than 80
```

関連項目

- “DO ステートメント”
- “DO ステートメント: 反復”
- “DO OVER ステートメント”
- “DO UNTIL ステートメント”
- “DO WHILE ステートメント”
- SELECT ステートメント

Label ステートメント

GOTO ステートメントのラベルターゲットを指定します。セミコロンは必要ありません。

サポート:　GOTO ステートメント

構文

label:*statement*;

必須引数

label
任意の識別子を指定し、その後にコロン(:)が続きます。label 引数を指定する必要があります。

注: ラベルに非ラテン文字が含まれている場合は、二重引用符で囲む必要があります。

statement
null ステートメント(;)を含む任意の実行可能なステートメントを指定します。statement 引数を指定する必要があります。

詳細

ラベルは、特定のステートメントに識別子を関連付け、GOTO や LEAVE などの他のステートメントからそのステートメントを参照できるようにします。ステートメントには複数のラベルを設定できます。

例

```
cas mysess;
proc cas;
flag=false;
if flag then goto test;
print (Note) "Your test ran successfully.";
goto finish;
test:print (Warn)"Your test did not run successfully.";
finish:;
end;
run;
```

```
WARNING: Your test did not run sucessully.
```

関連項目

- “GOTO ステートメント”
- “LEAVE ステートメント”

LEAVE ステートメント

現在のループの処理を停止し、シーケンス内の次のステートメントから再開します。

ヒント: LEAVE ステートメントを使用すると、条件に基づいて DO ループまたは SELECT グループを途中で終了できます。

構文

LEAVE;

詳細

LEAVE ステートメントを使用すると、反復する DO ループを途中で終了させることができます。LEAVE ステートメントは、単独で使用することも、条件に基づいて(たとえば、IF ステートメントと組み合わせて)使用することもできます。DO ループ内から LEAVE ステートメントが検出されると、ループの反復はそこで停止し、ループからループ後の最初のステートメントに制御が戻ります。

注: PROC CAS の非反復 DO ステートメントでは、1 つのステートメントとして実行されるコードのブロックが作成されます。LEAVE ステートメントは、DO ステートメントで作成されたプログラムには影響しません。LEAVE ステートメントは“DO ステートメント: 反復”に影響します。

例: LEAVE ステートメント

この例は、LEAVE ステートメントを示しています。

```
proc cas;
  function evaltest(x, y);
  if (x == y) then put "GOOD ";
    else           put "ERROR ";
  end;

  do i = 1 to 10 by 1;

  if (i == 5) then leave;
  put i;
  end;

  test = evaltest( i, 5,test);

  put i;
```

```
run;
```

次の行が SAS ログに書き込まれます。

例のコード 2.6　SAS ログ

```
1
2
3
4
GOOD
5
```

LOADACTIONSET ステートメント

SAS Cloud Analytic Services アクションセットをロードします。

要件　アクションセットをセッションにロードする機能は、アクセスコントロールの対象です。

例:　“例 5: ディシジョンツリーモデルの学習とデータのスコアリング” (85 ページ)

構文

LOADACTIONSET “*action-set-name*”;

必須引数

action-set-name
SAS Cloud Analytic Services アクションセットの名前を指定します。

LOCAL ステートメント

CASL 関数のローカル変数を作成します。

構文

LOCAL *variable*;

必須引数

variable
変数名を指定します。

詳細

変数のスコープには、ローカルまたはグローバルがあります。ローカル変数は、その変数が作成された関数内にのみ存在します。グローバル変数は、CASL プログラム内のどこからでもアクセスして 使用できます。LOCAL ステートメントを使用すると、関数のローカル変数を明示的に作成できます。また、GLOBAL ステートメントを使用すると、グローバル変数を明示的に作成できます。

注意
LOCAL ステートメントを使用して関数内にローカル変数を作成しない場合、その変数は暗黙的にグローバルになります。同じ名前のグローバル変数が存在する場合は、グローバル変数が参照されます。 関数内に変数を作成するときは、LOCAL ステートメントを使用することをお勧めします。

- グローバル変数は、CASL プログラム内のどこからでもアクセスして 使用できます。
- グローバル変数を明示的に作成するには、GLOBAL ステートメントを使用します。
- 変数が関数の外部で作成された場合、変数は暗黙的にグローバルになります。
- 変数が関数内で作成された際、その変数がコントロールループ変数でない場合や LOCAL ステートメントによって 作成されていない場合、その変数は暗黙的にグローバルになります。
- グローバル変数は、PROC CAS の QUIT ステートメントがサブミットされるまで保持されます。
- LOCAL ステートメントは、変数が作成される関数のローカル変数を明示的に作成しました。LOCAL ステートメントを使用して作成された変数は、その関数内にのみ存在します。
- 関数内で使用されるループ反復変数は、暗黙的にローカルスコープを持ち、その関数内にのみ存在します。
- ローカル変数は、関数が返されるまで保持されます。

例: ローカル変数の指定

次の例では、x という名前のローカル変数とグローバル変数の両方を作成します。この関数内で作成された変数はローカルスコープで作成されるため、同じ名前の他の変数を上書きすることはありません。

```
proc cas;
  function myFunction( y );
    local x;            1
    print "NOTE: When myFunction begins execution, x=" x " and y=" y;
    x=y;
    print "NOTE: After the myFunction assignment statement, x=" x " and y=" y;
  end func;
run;
  x=5;                  2
  print "NOTE: Before myFunction execution, x=" x;
```

```
  myFunction(2);
  print "NOTE: After myFunction execution, x=" x;
run;
quit;
```

1 LOCAL ステートメントは、ローカルスコープを持つ x という名前の変数を作成します。

2 この関数の外で作成された x 変数は、グローバルです。

例のコード 2.7 SAS ログ

```
NOTE: Before myFunction execution, x=5
NOTE: When myFunction begins execution, x=  and y=2
NOTE: After the myFunction assignment statement, x=2  and y=2
NOTE: After myFunction execution, x=5
```

関連項目

- *"変数スコープ" (SAS Cloud Analytic Services: CASL プログラマガイド)*
- "GLOBAL ステートメント" (45 ページ)

Null ステートメント

何も操作を実行しないステートメント。

構文

;

詳細

Null ステートメントは、セミコロンのみで構成されています。何も操作を実行しないステートメントです。

OUTPUT ステートメント

アクティブな出力場所を設定します。

デフォルト: ODS

構文

OUTPUT <ODS | LOG>;

オプション引数

<**ODS**>
出力が ODS 出力先に送信されることを指定します。

<**LOG**>
出力が SAS ログに送信されることを指定します。

例

例 1: アクティブな出力を SAS ログに設定

OUTPUT ステートメントで LOG オプションを指定すると、アクティブな出力が SAS ログに設定されます。

```
proc cas;
  output log;
  table.columnInfo / table="iris";
run;
```

例のコード 2.8　SAS ログ

```
Column          Label                      ID  Type
Species         Iris Species                1  char
SepalLength     Sepal Length (mm)           2  double
SepalWidth      Sepal Width (mm)            3  double
PetalLength     Petal Length (mm)           4  double
PetalWidth      Petal Width (mm)            5  double

     RawLength    FormattedLength          NFL          NFD
            15             15              0            0
             8             12              0            0
             8             12              0            0
             8             12              0            0
             8             12              0            0
```

例 2: アクティブな出力を ODS に設定

OUTPUT ステートメントで ODS オプションを指定すると、アクティブな出力の結果が HTML テーブルに生成されるように設定されます。

```
proc cas;
  output ods;
  table.columnInfo / table="iris";
run;
```

アウトプット 2.4　ODS HTML 出力

Column Information for IRIS in Caslib casdata							
Column	Label	Id	Type	Length	Formatted Length	Format Width	Format Decimal
Species	Iris Species	1	char	15	15	0	0
SepalLength	Sepal Length (mm)	2	double	8	12	0	0
SepalWidth	Sepal Width (mm)	3	double	8	12	0	0
PetalLength	Petal Length (mm)	4	double	8	12	0	0
PetalWidth	Petal Width (mm)	5	double	8	12	0	0

関連項目

“PRINT ステートメント”

PRINT ステートメント

定数、変数、および式の値を現在の出力場所に書き込みます。

注:　出力時に、PRINT ステートメントでは、文字列を含む 2 つの変数の間にスペースは追加されません。必要に応じてスペースを追加してください。

例:　“例 4: ランタイム数学関数とインライン演算子の使用” (84 ページ)

構文

PRINT <(*level*)>*value-1*<*value-2*>...<*value-N*>;

オプション引数

(*level*)
アクションからのログイベントに関連付けられている診断レベルを指定します。レベル付けされたメッセージがログに書き込まれると、その表示はレベルに基づいて適切にフォーマットされます。

level は、低い方から順に、次のいずれかになります。

TRACE
トレースレベルのメッセージを指定します。

DEBUG
デバッグレベルのメッセージを指定します。

INFO
情報レベルのメッセージを指定します。

NOTE
注レベルのメッセージを指定します。

WARN
警告レベルのメッセージを指定します。

ERROR
エラーレベルのメッセージを指定します。

デフォルト　TRACE

例　“例 4: メッセージレベルの追加 ” (59 ページ)

value
定数、変数または式を指定できます。

ヒント　PRINT ステートメントには複数の値を含めることができます。

例

例 1: 1 つの値を出力

次の例では、PRINT ステートメントを使用して、1 つの変数 *Grades* を出力しています。

```
proc cas;
  x={96,98,82,83,87,82,99,99,100};
  y={85,90,87,85,92};
  z=x+y;
  Grades=sort(z);
  print Grades;
run;
```

例のコード 2.9　SAS ログ

```
{82,82,83,85,85,87,87,90,92,96,98,99,99,100}
```

例 2: 複数の値を出力

次の例では、PRINT ステートメントを使用して、複数の変数 *x*、*y*、*z*、*Grades* を出力しています。

```
proc cas;
  x={96,98,82,83,87,82,99,99,100};
  y={85,90,87,85,92};
  z=x+y;
  Grades=sort(z);
 print x y z Grades;
run;
```

次の SAS ログでは、出力されるログの順序は、PRINT ステートメントの変数の順序と同じです。

例のコード 2.10　SAS ログ

```
{96,98,82,83,87,82,99,99,100} {85,90,87,85,92} {96,98,82,83,87,82,99,99,100,85,90,87,85,92}
{82,82,83,85,85,87,87,90,92,96,98,99,99,100}
```

例 3: 文字列変数の間隔の管理

PRINT ステートメントでは、値の間に自動的にスペースが追加されることはありません。次の例では、PRINT ステートメントを使用して、いくつかのテキストと変数 *a*、*b*、*c*、*d* の値を出力します。例の 2 番目の PRINT ステートメントでは、結果をフォーマットするために必要に応じてスペースを追加します。

```
proc cas;
  a="is";
  b="desired";
  c="text";
  d=".";
  print "This" a "my" b c d;
  print "This " a " my " b " " c d;
run;
```

次の SAS ログでは、1 行目が最初の PRINT ステートメントに対応し、2 行目が 2 番目の PRINT ステートメントに対応しています。

例のコード 2.11　SAS ログ

```
Thisismydesiredtext.
This is my desired text.
```

例 4: メッセージレベルの追加

次の例では、PRINT ステートメントを使用して、メッセージレベルを SAS ログに追加します。

```
proc cas;
  print;
  print "Plain text";
  print (trace) "TRACE level text";
  print (debug) "DEBUG level text";
  print (note) "NOTE level text";
  print (warn) "WARNING level text";
  print (error) "ERROR level text";
quit;
```

アウトプット 2.5　SAS ログ

```
Plain text
TRACE level text
      DEBUG level text
NOTE: NOTE level text
WARNING: WARNING level text
ERROR: ERROR level text
```

PRINTLST ステートメント

アプリケーションで定義されたリストファイルに出力します。値がリストの場合、リストは横向きになり、各アイテムが表示されます。アイテムがテーブルの 場合は、現在の出力場所に出力されます。

別名: plist

構文

PRINTLST *expression*;

必須引数

expression
ルールと優先順位を使用して評価され、結果を計算する 1 つ以上の値、定数、変数、演算子、関数の組み合わせ。

RETURN ステートメント

現在の関数から値を返します。

例: “例 5: ディシジョンツリーモデルの学習とデータのスコアリング” (85 ページ)

構文

RETURN*expression | value*;

必須引数

expression
式の結果は、関数が呼び出し環境(または呼び出し元)に返す値です。

value
関数から返される値を指定します。

例: RETURN ステートメントの使用

次の例では、2 つの関数を作成して、クラスと学生がコースの修了に向けて順調に進んでいるかどうかを判断する方法について詳しく説明します。

```
proc cas;
```

```
  grade={80,85,90,86,89,90,75,76,88,70,67};
  classavg=81;
  function evaltest (grade, classavg, student);        /* 1 */
  if (classavg<=67) then put "Class is ON TRACK";
   else put "Class is NOT ON TRACK";
  return (grade+1);                                    /* 2 */
end;
  function grade(classavg);
  return(67);                                          /* 3 */
end;
  grade=grade(1)+1;                                    /* 4 */
  test=evaltest(grade, 5, test);
   if (grade>=classavg) then put "Student is ON TRACK";
    else put "Student is NOT ON TRACK";
  put grade;
run;
```

1 ユーザー定義関数 Evaltest で、クラスが順調に進んでいるかどうかを判断します。この例では、クラス平均が 81 で、67 よりも大きいので、クラスは順調に進んでいます。

2 RETURN ステートメントは、学生の成績の値を決定します。現在の学生の成績は 67 で、すべての学生の成績に 1 点が追加されます。したがって、返される値は 68 です。

3 ユーザー定義関数 Grade では、学生の最終成績と、その学生がコースの修了に向けて順調に進んでいるかどうかを判断します。RETURN ステートメントは、67 の値を使用して、学生が順調に進んでいるかどうかを判断します。値 67 は 81 以上ではないため、この学生は順調に進んでいません。

4 別のクラスまたは学生が順調に進んでいるかどうかを判断するには、最初の IF ステートメントと 2 番目の RETURN ステートメントの値 67 を置き換えるだけです。

例のコード 2.12　SAS ログ

```
Class is ON TRACK
Student is NOT ON TRACK
68
```

RUN ステートメント

入力済みの SAS ステートメントを実行します。

構文

RUN;

引数なし

RUN ステートメントは、入力済みの SAS ステートメントを実行します。

詳細

RUN ステートメントは SAS プログラムのステップ間では必須ではありませんが、使用することでステップ境界を作成し、SAS ログを読みやすくすることができます。CASL コードは、RUN ステートメントが読み取られるか、他の PROC、DATA または QUIT ステートメントによって PROC が終了するまで実行されません。RUN ステートメントが検出されると、コードが実行され、RUN ステートメントの後にさらにコードが受け入れられます。先行する RUN ブロックで作成された変数は、後続の RUN ブロックのスコープに残ります。PROC CAS が PROC、DATA または QUIT ステートメントで終了した後は、変数はスコープから外れます。

注: DO ループ内で RUN ステートメントを使用しないでください。使用すると、DO ループが壊れて、エラーが発生します。

SAVERESULT ステートメント

指定された形式で結果テーブルを変数に保存します。

制限事項: サーバー側のみで実行する場合、SAVERESULT ステートメントはインメモリ CAS テーブルのみを保存します。それ以外の出力形式は無効です。

注: 32,767 文字を超える列は、大きすぎるためデータセットに保存できません。

例: “例 5: ディシジョンツリーモデルの学習とデータのスコアリング” (85 ページ)

構文

形式 1: **SAVERESULT** *variable-name* <DATAOUT=<*libref*.>*data-set-name* | LIB=*libref*> <REPLACE | NOREPLACE>;

形式 2: **SAVERESULT** *variable-name* <CSV="*file-name*" | FILE=*output-file*> <REPLACE | NOREPLACE>;

形式 3: **SAVERESULT** *variable-name* <CASLIB=*caslib-reference-name*> <CASOUT="*table-name*"> <REPLACE | NOREPLACE>;

必須引数

variable-name
1 つ以上の結果テーブルを含む変数を指定します。

オプション引数

CASLIB=*caslib-reference-name*
使用する caslib を指定します。

CASOUT="*table-name*"
結果テーブルをサーバーに転送し、インメモリテーブルとして保存することを指定します。デフォルトでは、アクティブな caslib が使用されますが、CASLIB=引数でオーバーライドできます。

CSV="*file-name*"
CSV ファイル名を指定します。ベストプラクティスは、完全修飾パスを指定することです。完全修飾パスを指定しない場合は、SAS プロセスの現在の作業ディレクトリにファイルを作成しようとします。

CSV ファイルはセッションエンコーディングで保存されます。ただし、UPLOAD ステートメントでファイルをアップロードすると、エンコーディングはデフォルトで UTF-8 になります。保存したファイルをアップロードするには、そのファイルの作成に使用したエンコーディングを指定する必要があります。

ENCODING=IMPORTOPTION パラメーターを使用して、エンコーディングを指定します。IMPORTOPTIONS オプションの詳細については、IMPORTOPTIONS= を参照してください。

ENCODING 関数と ENCODING=パラメーターを併用することで、CSV ファイルのエンコーディングを動的に指定できます。

DATAOUT=<*libref.*.>*data-set-name*
SAS ライブラリとデータセット名を指定します。デフォルトのライブラリ参照名は Work です。

FILE="*external-file*" | *fileref*
使用する外部ファイルまたはファイル参照名を指定します。

"external-file"
ファイルパスを指定します。ベストプラクティスは、完全修飾パスを指定することです。完全修飾パスを指定しない場合は、SAS プロセスの現在の作業ディレクトリにファイルを作成しようとします。

fileref
SAS ファイル参照名を指定します。

LIB=*libref*
使用する SAS ライブラリを指定します。指定するライブラリ参照名は、V9 エンジンライブラリ参照名にする必要があります。

REPLACE | NOREPLACE
既存のファイルを上書きするか、上書きしないかを指定します。デフォルトでは、SAS データセット、外部ファイル、インメモリテーブルは上書きされません。既存のファイルを上書きしないようにするには REPLACE を指定します。

デフォルト NOREPLACE

詳細

CAS にロードされたテーブルを置き換えるには、REPLACE オプションを指定するか、SAVERESULT ステートメントを実行する前に、ロードされたテーブルを CAS から削除します。次の表では、出力ファイルの種類と、各ファイルの種類に対するデフォルトの REPLACE オプションを示しています。

注: DATAOUT=が指定されていない場合、SAS データセットは Work.*result-table-name* として保存されます。

表 2.1 SAVERESULT 出力ファイルの種類とデフォルトの REPLACE 値

オプション	出力ファイルの種類	デフォルトの REPLACE 値
casout=*table-name*	CAS インメモリーテーブル	NOREPLACE
CSV=*path-name*	CSV	REPLACE
dataout=*libname.name*	SAS7bdat	REPLACE
lib=*libname*	SAS7bdat	REPLACE
file=*path-name*	テキスト	REPLACE

例

例 1: SAVERESULT ステートメントの使用: SAS データセット

このサンプルプログラムでは、SAVERESULT ステートメントを使用して、アクションから生成された結果テーブルを保存する方法を示しています。

```
proc cas;
  session casauto;
  output log;
  table.loadTable / path="iris.sashdat";                    /* #1 */
  simple.summary result=iris / table={name="iris"};          /* #2 */
  table01=findtable(iris);                                  /* #3 */
  saveresult table01 dataout=work.resultIris;               /* #4 */
run;
```

1 table.loadTable アクションを使用して、caslib のデータソースから *Iris* をロードします。

2 simple.Summary アクションを使用して、*Iris* テーブルの記述統計量を生成します。

3 *Table01* は、FINDTABLE という関数の結果を保存する新しいテーブルです。

4 SAVERESULT ステートメントは、*Table01* を SAS データセット ResultIris として Work ライブラリに保存します。

例 2: SAVERESULT ステートメントの使用: CSV ファイル

SAVERESULT ステートメントを使用して、アクションの結果を CSV ファイルに保存します。

```
proc cas;
 session casauto;
 output log;
 table.loadTable / path="iris.sashdat";
 simple.summary result=iris / table={name="iris"};
 tableIris=findtable(iris);
 saveresult tableIris csv="sum.csv";
run;
```

次の注が SAS ログに出力されます。

例のコード 2.13 SAS ログ

```
NOTE: The CSV file sum.csv has been created.
```

関連項目

"ACTION ステートメント"

SELECT ステートメント

複数のステートメントのいずれか、またはステートメントグループを実行します。

制限事項: END ステートメントを SELECT ステートメントの後に配置する必要があります。

注: SELECT ステートメントはネストできます。

構文

SELECT <(*select-expression*)>;
  WHEN-1 <(*when-expression-1* ..., *when-expression-n*>) *statement*;
   <...WHEN-n (*when-expression-1* ..., *when-expression-n*)> *statement*;
  <OTHERWISE *statement*>;
END;

必須引数

(*select-expression*)
単一の値に評価される式を指定します。

(*when-expression*)
複合式を含む SAS 式を指定します。SELECT では、*when-expression* を少なくとも 1 つ指定する必要があります。

注: 複数の *when-expressions* をカンマで区切ることができます。これは、論理演算子 OR で区切るのと同じです。

statement
DO、SELECT、および null ステートメントを含む、実行可能な SAS ステートメントを指定します。*statement* 引数を指定する必要があります。

詳細

SELECT グループでの WHEN ステートメントの使用

この SELECT ステートメントは、SELECT グループを開始します。SELECT グループには、特定の条件が真のときに実行される CAS ステートメントを識別する WHEN ステートメントが含まれます。SELECT グループで少なくとも 1 つの WHEN ステートメントを使用します。オプションの OTHERWISE ステートメントでは、WHEN 条件が満たされない場合に実行されるステートメントを指定します。END ステートメントで、SELECT グループを終了します。

注: SELECT ステートメントはネストできます。

例: SELECT ステートメントの使用

複数のステートメントを実行するには、SELECT ステートメントを使用します。次の例では、SELECT ステートメントは 2 つの when 式と 1 つの OTHERWISE ステートメントを実行します。

```
proc cas;
  a=10;
  select (a);
    when (10) put a;
    when (20) put a=a*2;
      otherwise;
  end;
run;
```

関連項目

- “DO ステートメント”
- “END ステートメント”
- “IF ステートメント”

SESSION ステートメント

プログラムによって起動されるアクションに使用するセッションを指定します。

例: “例 5: ディシジョンツリーモデルの学習とデータのスコアリング” (85 ページ)

構文

形式 1: **SESSION** *session-reference*;

必須引数

session-reference
既存セッションの名前を指定します。

例

例 1: セッションとセッション UUID の出力

この例では、*casauto* セッションのセッション UUID を出力します。

```
options cashost="cloud.example.com" casport=5570 casuser="sasdemo";
cas casauto;
proc cas;
   session casauto;
   action sessionid result=uuid;
   print uuid;
run;
```

```
{CASAUTO:Tue Sep 13 11:16:18 2016=4ebd26ad-9dc4-cf4f-a108-4f497d06a23f}
```

例 2: 既存のセッションに接続

この例では、UUID を使用して既存のセッションに接続します。UUID はセッション固有です。

```
options cashost="cloud.example.com" casport=5570 casuser="sasdemo";
cas casauto uuid="4ebd26ad-9dc4-cf4f-a108-4f497d06a23f";
proc cas;
   session casauto;
   session.sessionstatus result=s;
   put s;
run;
```

SET ステートメント

CAS アクションの一部の動作を変更するオプションを制御し、SAS ログからの対応するメッセージを制御します。

構文

SETDISP<ECHOEXTSRC><LOGS><LOGS=DEBUG | NOTE | TRACE | WARN><STDJSON><WARNINGDUP><USEWIDTH>

引数

SET ステートメントには、次の引数の少なくとも 1 つが必要です。

DISP
CAS アクションから返された処理を SAS ログに出力することを指定します。

別名 DISPOSITION

ECHOEXTSRC
EXTERNALSOURCE ステートメントコードブロックに埋め込まれたテキストを SAS ログにエコーできます。

例 “例 1: EXTERNALSOURCE ステートメントの使用”

LOGS
CAS からのメッセージを SAS ログに出力することを指定します。デフォルト値は TRACE です。

デフォルト TRACE

ヒント 引数のない LOGS オプションは、LOGS=TRACE オプションと同等です。追加のタイプのログメッセージの指定については、 LOGS= (68 ページ)オプションを参照してください。

UNSET ステートメントで LOGS オプションを指定すると、CAS からログへのメッセージの出力を抑制できます。

参照項目: UNSET ステートメントの詳細については、“UNSET ステートメント”を参照してください。

LOGS=
次のいずれかのオプションを指定して、ログへのメッセージを制御できます。

DEBUG
TRACE オプションで表示されたものを除くすべてのログメッセージを 出力します。

NOTE
DEBUG および TRACE オプションで表示されたものを除くすべてのログメッセージを出力します。

例 “例 3: 診断メッセージを SAS ログに出力する”

TRACE
すべてのログメッセージをログに 出力します。

例 “例 3: 診断メッセージを SAS ログに出力する”

WARN
DEBUG、NOTE、および TRACE オプションで表示されたものを除くすべてのログメッセージを出力します。

STDJSON
CASL2JSON 関数が JSON 標準に準拠したテキストのみを生成し、JSON2CASL 関数が JSON 標準に準拠したテキストのみを受け入れることを指定します。

STDJSON の設定は、CASL コードが SAS Compute Server (SAS クライアント)で実行されているか CAS サーバーで実行されているかに関係なく、CASL コードが実行されている環境にのみ影響します。

- 既存のコードとの下位互換性のために、標準からの差異がこれらの関数で許可されるように、STDJSON 引数はデフォルトで設定解除されています。SAS クライアント上で実行されているプログラムで STDJSON 引数を設定すると、SAS クライアント上の関数の動作にのみ影響します。

  SAS クライアントで STDJSON オプションを設定するには、PROC CAS コードブロックで set stdjson;を指定します。

```
proc cas;
  set stdjson;
run; quit;
```

- CAS サーバー上で実行されているプログラムで STDJSON 引数を設定すると、CAS サーバー上の関数の動作にのみ影響します。

  CAS サーバーで STDJSON オプションを設定するには、sccasl.runCasl アクションにサブミットされるコードに set stdjson;が含まれていることを確認してください。

```
proc cas;
  source pgm;
    set stdjson;
    < more-statements >;
  endsource;
  sccasl.runCasl / code=pgm;
 run;
```

STDJSON 設定は、CASL_STD_JSON 環境変数の値をローカルでオーバーライドできます。構成インスタンスでの環境変数の設定の詳細については、“Edit Server Configuration Instances” (*SAS Environment Manager: User's Guide*)を参照してください。

デフォルト STDJSON は設定解除されています。

参照項目: “Edit Configuration Instances” (*SAS Viya Platform: Programming Run-Time Servers*)

“CASL2JSON 関数” (107 ページ)

“JSON2CASL 関数” (146 ページ)

USEWIDTH
テーブル内の文字列値の定義された列幅の使用を指定します。列の定義された幅よりも幅が狭い値は、定義された列の幅全体まで空白が埋め込まれます。USEWIDTH は、結果テーブル内の固定長の文字変数にのみ影響します。可変長の文字変数は影響を受けません。

デフォルト: USEWIDTH は設定解除されています。

ヒント: UNSET ステートメントを使用して、列内のすべての値の最大測定幅まで、すべての値に空白を埋め込むように指定できます。

WARNINGDUP
CAS アクションから返された 2 つ以上の応答辞書エントリが同じキーを持つ場合に、SAS ログに警告メッセージを出力することを指定します。runCasl アクションの場合、SEND_RESPONSE 関数を CASL コードで使用して、アクションからクライアントに 1 つ以上の応答辞書を送信できます。応答に重複した名前の値が含まれている場合、応答で CAS アクションから最後に受け取った値がそのキーの値になります。

デフォルト: WARNINGDUP は設定解除されています。

ヒント: UNSET ステートメントを指定することにより、重複した応答辞書エントリの警告が出力されないように指定できます。

例: “例 2: 重複キーの警告メッセージを SAS ログに出力する ”

例

例 1: SAS ログでの処理の表示

SET ステートメントを使用して、オプションをオンにすることができます。SET DISP は、SAS ログに処理を表示するために使用されます。

```
proc cas;
  set disposition;                /* 1 */
  builtins.help / action="serverStatus";
run;
  builtins.help / action="nonExistent";  /* 2 */
run;
  unset disposition;              /* 3 */
  builtins.help / action="nonExistent";
run;
quit;
```

1 SET DISPOSITION ステートメントを指定して、処理をログに出力します。アクションは正常に実行されるため、次の処理がログに出力されます: NOTE: Disposition: Severity *Normal*。

2 SET DISPOSITION ステートメントがまだアクティブであり、nonExistent アクションがエラーになっているため、次の処理がログに出力されます: ERROR: Disposition: Severity *Error*。

3 DISPOSITION オプションを UNSET ステートメントで指定して、処理メッセージのログへの出力を停止します。

アウトプット 2.6　SAS ログでの処理の表示のログ出力

```
{serverStatus=TRUE}
ERROR: Action 'nonExistent' was not found.
ERROR: The action stopped due to errors.
ERROR: Disposition: Severity Error
ERROR: Action 'nonExistent' was not found.
ERROR: The action stopped due to errors.
```

例 2: 重複キーの警告メッセージを SAS ログに出力する

次の例では、WARNINGDUP オプションを SET ステートメントで使用して、2 つの応答辞書エントリが同じキーを返すため、警告メッセージを SAS ログに出力します。

```
proc cas;
  source serversource;
    resp1={"t"=99};
    send_response(resp1);
    resp2={"t"=v};
    send_response(resp2);
    exit( {severity=0,reason=0,statusCode=0} );
  endsource;
  set warningdup;                    /* 1 */
  sccasl.runCasl result = r status = sc  /
    code=serversource, vars={v=1};
  print r;
run;
quit;
```

1 プログラムの割り当てで respl 変数が重複しているため、SET WARNINGDUP ステートメントは警告メッセージをログに出力します。

注: EXIT 関数は、指定されたステータスで CASL コードのブロックを終了します。

アウトプット 2.7　ログへの重複キーの警告メッセージの出力に関するログ出力

```
This is WARNING: The key 't' has a duplicate in the results.
{t=1}
```

例 3: 診断メッセージを SAS ログに出力する

次の例では、SET LOGS ステートメント、SET LOGS=ステートメント、および UNSET LOGS ステートメントを使用して、SAS ログへのメッセージを制御します。UNSET ステートメントは、CAS から SAS ログへのメッセージの出力を抑制します。

```
proc cas;
  source serversource;
     print (trace) "This is a trace-level message.";
     print (debug) "This is a debug-level message";
     print  (note) "This is a note-level message";
     print  (warn) "This is a warning-level message";
     print (error) "This is an error-level message";
     exit( {severity=0,reason=0,statusCode=0} );
  endsource;
  print "Specify LOGS (with no arguments) to print all messages to the log";
  set logs;      /* 1 */
  sccasl.runCasl result = r
              status = sc / code=serversource;
run;
  print "Specify LOGS=NOTE to print all messages except DEBUG and TRACE to the log";
  set logs=note;  /* 2 */
  sccasl.runCasl result = r
              status = sc / code=serversource;
run;
  print "Specify UNSET LOGS to suppress all messages to the log";
  unset logs;     /* 3 */
  sccasl.runCasl result = r
              status = sc / code=serversource;
run;
quit;
```

1 CAS からのすべてのメッセージを SAS ログに出力します。デフォルト値は TRACE です。

2 DEBUG オプションと TRACE オプションで表示されたものを除くすべてのログメッセージを出力します。

3 UNSET ステートメントを指定して、SAS ログで CAS からのメッセージの出力を抑制します。

アウトプット 2.8　診断メッセージをログに出力するためのログ出力

```
Specify LOGS (with no arguments) to print all messages to the log
This is a trace-level message.
This is a debug-level message
NOTE: This is a note-level message
WARNING: This is a warning-level message
ERROR: This is an error-level message
Specify LOGS=NOTE to print all messages except DEBUG and TRACE to the log
NOTE: This is a note-level message
WARNING: This is a warning-level message
ERROR: This is an error-level message
Specify UNSET LOGS to suppress all messages to the log
```

関連項目

“UNSET ステートメント”

SOURCE ステートメント

プログラムにテキストを埋め込み、特定の変数に割り当てることができます。

制限事項: SOURCE ステートメントと ENDSOURCE ステートメントはネストできません。

構文

SOURCE *variable*;

<*text*>

ENDSOURCE;

必須引数

variable
テキストを割り当てる変数。

オプション引数

text
変数の値の文字列表現を指定します。

例: SOURCE ステートメントの使用

次の例では、SOURCE ステートメントを使用してプログラムにテキストを埋め込みます。PRINT ステートメントを使用すると、埋め込みテキストが SAS ログに表示されます。埋め込みテキストは *pgm* 変数に割り当てられます。

```
proc cas;
  source pgm;
    v1='"weight"'n/"hei'ght"n;
    if missing('ag"e'n) then v2=0;
      else v2=v1/'ag"e'n;
  endsource;
run;
  print pgm;
run;
quit;
```

例のコード 2.14　SAS ログ

```
v1='"weight"'n / "hei'ght"n; if missing('ag"e'n) then v2=0;
else v2=v1/'ag"e'n;
```

関連項目

“EXTERNALSOURCE ステートメント”

UNSET ステートメント

CAS アクションの一部の動作を変更するオプションをオフにし、SAS ログ内の対応するメッセージを抑制します。

ヒント:　SET ステートメントを使用して、SAS ログに書き込むメッセージを設定できます。

構文

UNSETDISP<ECHOEXTSRC><LOGS><STDJSON><USEWIDTH ><WARNINGDUP>;

引数

DISPOSITION
CAS アクションから返された処理を SAS ログに出力することを指定します。

別名　DISP

例　“例: UNSET ステートメントの使用”

ECHOEXTSRC
SAS が EXTERNALSOURCE ステートメントによって収集された行を SAS ログにエコーしないようにします。

LOGS
CAS からのメッセージを SAS ログに出力することを指定します。

STDJSON
CASL2JSON 関数が JSON 標準に厳密に準拠していない出力テキストを生成できること、および JSON2CASL 関数が JSON 標準に厳密に準拠していないテキストを受け入れることができることを指定します。

既存のコードとの下位互換性のために、標準からの差異がこれらの関数で許可されるように、STDJSON 引数はデフォルトで設定解除されています。STDJSON の設定解除は、CASL コードが SAS Compute Server (SAS クライアント)で実行されているか CAS サーバーで実行されているかに関係なく、CASL コードが実行されている環境にのみ影響します。

- SAS クライアント上で実行されているプログラムでの STDJSON 引数の設定解除は、SAS クライアント上の関数の動作にのみ影響します。

  SAS クライアントで STDJSON オプションを設定解除するには、PROC CAS コードブロックで unset stdjson;を指定します。

```
proc cas;
  unset stdjson;
run; quit;
```

- CAS サーバー上で実行されているプログラムで STDJSON 引数を設定解除すると、CAS サーバー上の関数の動作にのみ影響します。

  CAS サーバーで STDJSON オプションを設定解除するには、sccasl.runCasl アクションにサブミットされるコードに unset stdjson が含まれていることを確認してください。

```
proc cas;
  source pgm;
    unset stdjson;
    < more-statements >;
  endsource;
  sccasl.runCasl / code=pgm;
 run;
```

STDJSON 設定は、CASL_STD_JSON 環境変数の値をローカルでオーバーライドできます。構成インスタンスでの環境変数の設定の詳細については、“Edit Server Configuration Instances” (*SAS Environment Manager: User's Guide*)を参照してください。

デフォルト　STDJSON は設定解除されています。

参照項目:　“Edit Configuration Instances” (*SAS Viya Platform: Programming Run-Time Servers*)

“CASL2JSON 関数” (107 ページ)

“JSON2CASL 関数” (146 ページ)

USEWIDTH

すべての値に対してグローバルに定義された幅に基づいて幅を定義するのではなく、各文字値に使用される最大幅に基づいてテーブル内の文字列の列の幅を定義することを指定します。USEWIDTH は、結果テーブル内の固定長の文字変数にのみ影響します。可変長の文字変数は影響を受けません。

デフォルト　USEWIDTH は設定解除されています。

ヒント　SET ステートメントを使用して、定義された列幅まですべての値に空白を埋め込むように指定できます。

WARNINGDUP

重複する応答辞書エントリの警告が出力されないように指定します。

例: UNSET ステートメントの使用

UNSET ステートメントはオプションをオフにします。unset disposition ステートメントは、SAS ログでの処理の出力を抑制します。

```
proc cas;
  session casauto;
  unset disposition;
  setsessopt / caslib="casuser";
run;
```

関連項目

“SET ステートメント” (68 ページ)

UPLOAD ステートメント

SAS クライアントからサーバーにファイルを転送します。データ転送後、サーバーはデータをインメモリテーブルにロードします。

制限事項: CSV ファイルをアップロードする際には、CSV ファイルのエンコーディングを指定する必要があります。詳細については IMPORTOPTIONS=を参照してください。

注: サーバーが受信したファイルは、ファイルがインメモリテーブルにロードされるまで一時的に保存されます。ロードされると、ファイルは削除されます。

アップロードされたファイル

構文

UPLOAD

PATH=*path-to-file*

<CASOUT={*output-table-options*}>

<IMPORTOPTIONS={FILETYPE=“BASESAS” | “CSV” | “DTA” | “ESP” | “EXCEL” | “FMT” | “EXCEL” | “HDAT” | “JMP” | “LASR” | “SPSS” | “XLSX”<, *fileType-specific-parameters*>}>;

必須引数

PATH=*path-to-file*
アップロードするファイルのパスを指定します。パスは、SAS クライアントがアクセスできるディレクトリから完全修飾されている必要があります。

オプション引数

CASOUT= *output-table-options*
基本出力テーブルの設定を指定します。次のパラメーターを指定できます。

- CASLIB= “*string*”
- INDEXVARS={ “*variable-name-1*” <, “*variable-name-2, ” ...*>}
- LABEL= “*string*”
- NAME= “*table-name*”
- PROMOTE=TRUE | FALSE
- REPLACE=TRUE | FALSE
- REPLICATION=*integer*

IMPORTOPTIONS={"BASESAS" | "CSV" | "DTA" | "ESP" | "EXCEL" | "FMT" | "HDAT" | "JMP" | "LASR" | "SPSS" | "XLSX"<, *fileType-specific-parameters*>}
ファイル形式とオプションを指定します。fileType パラメーターに指定する値によって、適用される他のパラメーターが決まります。fileType パラメーターの詳細については、Upload アクションの"importOptions"を参照してください。

CSV ファイルはセッションエンコーディングで保存されます。ただし、UPLOAD ステートメントでファイルをアップロードすると、エンコーディングはデフォルトで UTF-8 になります。保存したファイルをアップロードするには、そのファイルの作成に使用したエンコーディングを指定する必要があります。

ENCODING 関数を使用して、セッションエンコーディングを ENCODING=パラメーターに動的に渡すことができます。次の IMPORTOPTION での CSV ファイルの指定では、ENCODING 関数を使用して、セッションエンコーディング文字列が返され、それが ENCODING=パラメーターに渡されます。
importoptions={filetype="CSV", encoding=encoding()};

例: UPLOAD ステートメントの使用

次の例は、UPLOAD ステートメントを使用して CSV ファイルを SAS クライアントからサーバーに転送し、サーバー側のインメモリテーブルで基本的なプログラミングタスクを実行する方法を示しています。

```
proc cas;
  session casauto;
  upload path="your-file-path\titanic3.csv"
    importoptions={filetype="CSV", encoding=encoding()}
    casout={name="Titanic01"};                    /* #1 */
run;
  table.tableInfo;                                /* #2 */
run;
proc cas;
  simple.Summary result=Titanic02 / table={name="Titanic01"};   /* #3 */
  print Titanic02;
  Titanic03=findTable(Titanic02);                 /* #4 */
  saveresult Titanic03 dataout=work.Titanic03;          /* #5 */
run;
```

1. UPLOAD ステートメントを使用して、Titanic3.csv を SAS クライアントからサーバーに転送します。IMPORTOPTION での CSV ファイルの指定では、ENCODING 関数を使用して、セッションエンコーディング文字列が返され、それが ENCODING=パラメーターに渡されます。CASOUT オプションでは、name パラメーターを使用して新しいテーブル名が作成されます。
2. table.Tableinfo アクションを使用して、サーバーにアップロードしたテーブルを表示します。詳細については、tableInfo アクションを参照してください。
3. simple.Summary アクションを使用して、*Titanic01* テーブルの数値変数の記述統計量を生成し、その結果を *Titanic02* に保存します。詳細については、summary アクションを参照してください。
4. *Titanic03* は、関数 FINDTABLE の結果を保存する新しいテーブルです。詳細については、FINDTABLE 関数を参照してください。
5. SAVERESULT ステートメントを使用して、*Titanic03* を SAS データセットとして work ディレクトリに保存します。詳細については、SAVERESULT ステートメントを参照してください。

使用法: CAS プロシジャ

使用法: CAS プロシジャ

PROC CAS ステップは対話型プロシジャを開始し、次のいずれかが発生するまで終了しません。

- DATA ステップ
- 別の PROC ステップ
- QUIT ステートメント
- プロシジャの進行を妨げるエラー状態。

グローバルステートメント、SAS マクロコード、および RUN ステートメントでは、SAS プロシジャは終了しません。

結果: CAS プロシジャ

結果テーブル

CAS アクションが結果を返す場合、結果のデータ型は通常、1 つ以上の結果テーブルを含む辞書です。結果テーブルは、CAS アクションが CASL に情報を返すために用いる主要な手段です。行と列に加えて、結果テーブルにはラベルとデータ型も含まれます。結果テーブルに対する操作ではいずれも新しい結果テーブルが作成される可能性があり、CASL には独自の結果テーブルを作成するための構文が含まれています。

CASL は、CAS 結果テーブルに対してさまざまな操作を提供します。

- 結果テーブル内の特定の行と列の値を参照できます。
- 行、列名、および種類を辞書に抽出できます。
- 新しいテーブルに保持する行と列をリストすることにより、テーブルをサブセット化できます。
- WHERE 式処理を使用して、WHERE 式に一致する行を持つ新しいテーブルを作成できます。
- COMPUTE 句を使用して、計算列を作成したり、各行の計算値で配列を作成したりできます。
- アクションのサブミットから受け取った結果を反復処理できます。

CASL を使用すると、既存の結果テーブルから独自の結果テーブルを作成できます。また、他の結果テーブルのサブセットまたは組み合わせを作成することもできます。

関連項目

“CASL 結果テーブル” (*SAS Cloud Analytic Services: CASL プログラマガイド*)

例: CAS プロシジャ

例 1: PROC CAS のセットアッププログラム

本書のいくつかの例では、データをダウンロードして CAS にロードする必要があります。次の例では、HTTP プロシジャと PROC CAS UPLOAD ステートメントを使用してデータにアクセスし、CAS にロードします。

プログラム

```
%let data='http://support.sas.com/documentation/onlinedoc/viya/exampledatasets/
iris.csv';
filename t temp;
proc http method="get" url=&data. out=t;
run;
%let temppath = %sysfunc(quote(%sysfunc(pathname(t))));

proc cas;
  upload path=&temppath.
      casOut={
      name="iris"
      replace=True
    }
   importOptions={fileType="csv"}
;
run;
```

例のコード 2.15　SAS ログ

```
NOTE: Cloud Analytic Services made the uploaded file available as table IRIS in caslib CASUSER(sasdemo).
NOTE: The table IRIS has been created in caslib CASUSER(sasdemo) from binary data uploaded to Cloud
Analytic Services.
{caslib=CASUSER(sasdemo),tableName=IRIS}
```

例 2: アクションの実行

この例では、指定されたアクション(この場合は listNodes アクション)を実行し、テーブルのコンテンツを Output Delivery System (ODS)に表示します。

プログラム

```
proc cas;                          /* #1 */
 action listnodes result=res;      /* #2 */
 print res;                        /* #3 */
run;
```

1 PROC CAS ステートメントは、CAS プロシジャに固有のステートメントと CASL ステートメントの選択サブセットを実行するように SAS クライアントの準備をします。

2 ACTION ステートメントでは、指定されたアクション(この場合は listNodes アクション)が実行されます。listNodes アクションの結果は、*Res* という名前の変数に保存されます。

3 PRINT ステートメントでは、変数 *Res* のコンテンツが表示されます。listNodes アクションではテーブルが返され、PRINT ステートメントでは表示のためにテーブルが Output Delivery System (ODS)に提供されます。

次の出力は、8 台のマシンのサンプル結果を示しています。

アウトプット 2.9　出力: listNodes アクションの部分出力

res: Results from builtins.listNodes

Name	Role	Connected	IP Address
cloud.example.com	controller	Yes	192.0.2.0
cloud.example.com	controller	Yes	192.0.2.0
cloud.example.com	worker	Yes	192.0.2.0
cloud.example.com	worker	Yes	192.0.2.0
cloud.example.com	worker	Yes	192.0.2.0
cloud.example.com	worker	Yes	192.0.2.0
cloud.example.com	worker	Yes	192.0.2.0
cloud.example.com	worker	Yes	192.0.2.0
cloud.example.com	worker	Yes	192.0.2.0
cloud.example.com	worker	Yes	192.0.2.0
cloud.example.com	worker	Yes	192.0.2.0

例 3: ユーザー定義関数の作成

この例では、新しい関数を定義し、IF-ELSE/THEN、DO、および PRINT ステートメントを使用して関数を実行します。また、この例では、内部関数 *factorial* を使用して、新しく作成された関数 *x* を定義および実行します。

プログラム

```
proc cas;                                  /* #1 */
  function factorial(x);                       /* #2 */
    if (x < 1.0) then return(x);                 /* #3 */
    else do;
      return exp(lgamma (x+1));
    end;                                   /* #4 */
  end;
run;
```

```
  do i = 1 to 9;                          /* #5 */
    x = factorial(i);
    print "   Factorial (" i ") = " put(x,best9.); /* #6 */
  end;
  x = factorial(75);
  print "Factorial of 75 = " x;
run;
  x = factorial(75.0);
  print "Factorial of 75.0 = " x;
run;
```

1 PROC CAS ステートメントは、CAS プロシジャに固有のステートメントと CASL ステートメントの選択サブセットを実行するように SAS クライアントの準備をします。

2 FUNCTION ステートメントでは、1 つの引数を取る新しい関数が作成されます。

3 この関数では、**lgamma** および **exp** 内部関数を使用して、数値の階乗が計算されます。IF ステートメントでは、*x* の値が 1.0 より小さいかどうかが問われます。ELSE ステートメントは、ランタイム数学式関数 lgamma に *x* と 1 を足した値を掛けて返すよう要求します。

4 END ステートメントで、関数の処理が終了します。

5 DO ステートメントで、引数 1-9 を使用して関数を呼び出します。

6 PRINT ステートメントで結果が出力されます。

例のコード 2.16 出力ログ: 新しい関数の定義

```
Factorial (1 ) =       1
Factorial (2 ) =       2
Factorial (3 ) =       6
Factorial (4 ) =      24
Factorial (5 ) =     120
Factorial (6 ) =     720
Factorial (7 ) =    5040
Factorial (8 ) =   40320
Factorial (9 ) =  362880
Factorial of 75= 2.480914E109
Factoiral of 75.0= 2.480914E109
```

例 4: ランタイム数学関数とインライン演算子の使用

プログラム

この例では、新しい関数を作成し、ビルトインランタイム数学関数を使用して、同じ部屋で同じ誕生日を共有する人数の確率を計算します。この例では、共通関数 *lgamma* および *exp* を使用して、数値の階乗を計算します。この例では、インライン演算子も使用しています。これにより、指定された関数がスタックフレームなしで呼び出され、現在の変数のローカル変数にアクセスできるようになります。構文 inline.function-name は、コード内の正確なコードを呼び出して、コンパイル中に実行されるようにします。

```
proc cas;                                    /* #1 */
  function SharedFeature (feature,number);        /* #2 */
  p = exp(lgamma (feature+1)- lgamma              /* #3 */
        (feature-number+1)-number*log(feature));
  return (1-p);
  end;
run;
  do n over {3 10 22 23 50 75};                 /* #4 */
  p =inline.SharedFeature(365,n);                 /* #5 */
  print "Chance at least 1 out of " put(n,best3.)
 " share same birthday = " put(p,best8.4);        /* #6 */
  p =inline.SharedFeature(365,3);
  print "Chance at least 2 out of " put(n,best3.)
 " share same birthday = " put(p,best8.4);
end;
run;
quit;
```

1 PROC CAS ステートメントは、CAS プロシジャに固有のステートメントと CASL ステートメントの選択サブセットを実行するように SAS クライアントの準備をします。

2 FUNCTION ステートメントでは、新しい関数が作成されます。この関数には 2 つの引数が含まれ、アイテムの *number* と *features* を指定して、少なくとも 2 つのアイテムが同じ特徴を共有する確率を計算します。

3 この関数では、**lgamma**、**log** および **exp** 内部関数を使用して確率を計算します。

4 DO ステートメントでは、リストが反復処理されます。

5 変数 *p* はここで定義されます。インライン演算子を使用して **SharedFeature** 関数を呼び出し、部屋の人数を指定して、少なくとも 2 人の誕生日が同じになる確率を計算します。

6 PRINT ステートメントで結果が出力されます。PUT 関数では、BEST.出力形式が適用されます。

注: 関数は再帰的であり、CASL ステートメントを実行できます。関数を再定義することもできます。

例のコード 2.17 ログ: 1 人または 2 人が同じ部屋で同じ誕生日を共有する確率

```
Chance at least 1 out of 3  share same birthday = 0.0082041659
Chance of at least 2 out of 3  share same birthday=0.0082041659
Chance at least 1 out of 10  share same birthday = 0.1169481777
Chance of at least 2 out of 10  share same birthday=0.0082041659
Chance at least 1 out of 22  share same birthday = 0.4756953077
Chance of at least 2 out of 22  share same birthday=0.0082041659
Chance at least 1 out of 23  share same birthday = 0.5072972343
Chance of at least 2 out of 23  share same birthday=0.0082041659
Chance at least 1 out of 50  share same birthday = 0.9703735796
Chance of at least 2 out of 50  share same birthday=0.0082041659
Chance at least 1 out of 75  share same birthday = 0.9997198782
Chance of at least 2 out of 75  share same birthday=0.0082041659
```

例 5: ディシジョンツリーモデルの学習とデータのスコアリング

この例は、ディシジョンツリーモデルに学習させ、CASL アクションによって 生成されたデータをスコアリングする方法を示しています。

プログラム

```
cas casauto;
libname mycas cas sessref=casauto;
data mycas.golf;                          /* #1 */
  format outlook     $8.;
  format temperature best10.;
  format humidity    best10.;
  format windy       $5.;
  format golf        $10.;
  input outlook $ 1-8 temperature humidity windy $ 16 - 21 golf $22 - 32;
datalines;
sunny    85 85 false Don't Play
sunny    80 90 true  Don't Play
overcast 83 78 false Play
rain     70 96 false Play
rain     68 80 false Play
rain     65 70 true  Don't Play
overcast 64 65 true  Play
sunny    72 95 false Don't Play
```

```
sunny    69 70 false Play
rain     75 80 false Play
sunny    75 70 true  Play
overcast 72 90 true  Play
overcast 81 75 false Play
rain     71 80 true  Don't Play
;
run;
filename score temp;                        /* #2 */
proc cas;
 decisionTree.dtreeTrain result=r /          /* #3 */
  table={name = "golf"}
   inputs={"outlook", "windy", "humidity", "temperature"}
   target="golf"
   maxlevel =4
   maxbranch=2
   nbins    =5
   binorder =1
   varImp   =true
   code={labelid=999, comment=true, tabForm=true};
run;
   print r['ModelInfo'];
   print r['DTreeVarImpInfo'];
run;
   saveresult r['CodeGen'] file=score;       /* #4 */
run;
quit;
data mycas.more_golf;                       /* #5 */
   format outlook     $8.;
   format temperature best10.;
   format humidity    best10.;
   format windy       $5.;
   *format golf       $10.;
   input outlook $ 1-8 temperature humidity windy $ 16 - 21 /* golf $22 - 32 */;
datalines;
sunny    75 85 true
overcast 83 78 false
sunny    68 80 false
;
run;
proc cas;
  code = readfile("score");                 /* #6 */
 dscode =
   "data more_golf_scored;
     set more_golf;"
     || code ||
"run;";
run;
 dataStep.runCode / code = dscode;          /* #7 */
run;
 table.fetch / table = "more_golf_scored";    /* #8 */
run;
quit;

libname casuser v9 '/u/casuser';
data casuser.golf;
```

```
 set mycas.golf;
run;

data casuser.more_golf;
 set mycas.more_golf;
run;
```

1 CASL を使用して、DATA ステップスコアコードを生成、保存、および実行します。CAS の DATA ステップは CAS テーブルを操作対象とします。入力および出力データセットは CAS LIBNAME エンジンを使用する必要があります。このエンジンを使用すると、プログラムをコンパイルするときに、DATA ステップで CAS テーブルの列メタデータをフェッチできます。他のエンジンを使用するデータセットは、CAS にロードするか、そのデータソースに対して caslib を定義する必要があります。

2 FILENAME ステートメントでは、SAS ファイル参照名が外部ファイルに関連付けられます。この例では、FILENAME ステートメントで、*score_golf.sas* が外部の *score* ファイルに関連付けられます。

3 ディシジョンツリーアクションセットでは、ディシジョンツリーモデルの DATA ステップスコアリングコードを生成できるアクションが提供されます。dtreeTrain アクションは、ディシジョンツリーに学習させます。dtreeTrain の構文の詳細については、*"ディシジョンツリーの学習" (SAS Visual Analytics: プログラミングガイド)*を参照してください。

4 SAVERESULT ステートメントでは、DATA ステップスコアコードが SAS データセットに保存されます。

5 DATA ステップでは、スコアリングする新しいデータが作成されます。DATA ステップでコメントアウトされたコードに注意してください。*golf* は DATA ステップから除外されています。

6 Readfile を使用して、ファイルのコンテンツを読み取ります。Readfile 関数では、指定されたファイルのコンテンツを文字列として変数に読み込みます。

7 dataStep.runCode アクションでは、新しいオブザベーションをスコアリングしています。スコアリングされたデータには、ゴルフをするための予測と予測確率が含まれます。

8 テーブルから行をフェッチします。format パラメーターを使用して、変数に出力形式を適用します。sortBy を使用して、結果を並べ替えるための変数と変数設定を指定します。パラメーターを使用して、最後に返す行の順序位置を指定します。

アウトプット 2.10　HTML 出力: 学習したディシジョンツリー

rModelInfo: Results from decisionTree.dtreeTrain

Decision Tree for GOLF	
Number of Tree Nodes	3.000000
Max Number of Branches	2.000000
Number of Levels	2.000000
Number of Leaves	2.000000
Number of Bins	5.000000
Minimum Size of Leaves	5.000000
Maximum Size of Leaves	9.000000
Number of Variables	4.000000
Confidence Level for Pruning	0.250000
Number of Observations Used	14.000000
Misclassification Error (%)	28.571429

rDTreeVarImpInfo: Results from decisionTree.dtreeTrain

Variable Importance in Decision Tree Related Analytics			
Variable Name	Importance	Std Dev.	Count
outlook	0.9175	0	1.0000

アウトプット 2.11　HTML 出力: 新しいデータセット

Results from dataStep.runCode

CAS Library	Name	Number of Rows	Number of Columns
CASUSER(casuser)	more_golf	3	4

Output CAS Tables			
CAS Library	Name	Number of Rows	Number of Columns
CASUSER(casuser)	more_golf_scored	3	7

アウトプット 2.12　HTML 出力: テーブルフェッチアクション

Results from table.fetch

Selected Rows from Table MORE_GOLF_SCORED							
Index	outlook	temperature	humidity	windy	_leaf_id_	DT_golf	DT_golf_PredP
1	sunny	75	85	true	2	Don't Play	0.6
2	overcast	83	78	false	1	Play	0.7777777778
3	sunny	68	80	false	2	Don't Play	0.6

例 6: 計算列のサブセット化

この例では、CAS アクションを使用して SAS データセットから行を取得し、すべての行の比率を計算し、結果をサブセット化して、テーブルをディスクに保存します。この例では、CAS ステートメント、アクション、および CASUTIL プロシジャを使用しています。

プログラム

```
%let data='http://support.sas.com/documentation/onlinedoc/viya/exampledatasets/
cars.csv';
filename t temp;
proc http method="get" url=&data. out=t;                    /* #1 */
run;
%let temppath = %sysfunc(quote(%sysfunc(pathname(t))));

proc cas;
  upload path=&temppath.
      casOut={
      name="cars"
      replace=True
    }
   importOptions={fileType="csv"}
;
run;
proc cas;
 table.recordCount result=count /                           /* #2 */
 table="cars";
run;
table.fetch result=fetchr/                                  /* #3 */
 table="cars"
  to=findtable(count)[1,1];
run;
lowMSRP=sort(findtable(fetchr), "MSRP")[1:5,{"MODEL","MAKE","MSRP"}];    /* #4 */
print lowMSRP;
run;
computedcolumn=findtable(fetchr).                           /* #5 */
   compute({"ratio","Invoice/MSRP",best5.3},Invoice/MSRP)
```

```
   [,{"MODEL", "MAKE","MSRP","ratio"}];
subset= sort(computedcolumn,"ratio")[1:5];                    /* #6 */
print subset;
run;
nrows=findtable(count)[1,1];
print"total rows:" nrows;
run;
saveresult lowMSRP dataout=sasuser.lowMSRP;                   /* #7 */
saveresult subset dataout=sasuser.subset;
run;
quit;
```

1 HTTP プロシジャと PROC CAS UPLOAD ステートメントを使用して、CARS データセットにアクセスし、CAS にロードします。

2 recordCount アクションでは、CARS テーブルの行数が表示され、レコードカウントが変数 *count* に保存されます。

3 fetch アクションでは、テーブル CARS から行がフェッチされます。

4 割り当てステートメントでは、変数 *lowMSRP* を定義します。変数 *lowMSRP* には、MSRP が最も低い 5 台の車両が含まれます。

5 割り当てステートメントでは、変数 *computedcolumn* を定義します。**compute** 関数では、すべての行の請求書と MSRP の比率が計算されます。

6 割り当てステートメントでは、変数 *subset* を定義します。**sort** 関数では、最も低い 5 つの比率を並べ替えてサブセット化します。行の総数は SAS ログに表示されます。

7 SAVERESULT ステートメントでは、CAS テーブルがディスクに保存されます。出力テーブルはファイルとして保存されます。

アウトプット 2.13　lowMSRP 結果テーブル

lowMSRP: Results

Model	Make	MSRP
Rio 4dr manual	Kia	$10,280
Accent 2dr hatch	Hyundai	$10,539
Echo 2dr manual	Toyota	$10,760
Ion1 4dr	Saturn	$10,995
Rio 4dr auto	Kia	$11,155

アウトプット 2.14　subset 結果テーブル

subset: Results

Model	Make	MSRP	Invoice/MSRP
911 Carrera 4S coupe 2dr (convert)	Porsche	$84,165	0.858
LX 470	Lexus	$64,800	0.871
SC 430 convertible 2dr	Lexus	$63,200	0.871
LS 430 4dr	Lexus	$55,750	0.871
GS 430 4dr	Lexus	$48,450	0.872

3

CASL ビルトイン関数

CASL 関数の概要

CASL インターフェイスは、ルーチンが返す可能性のある値を作成および管理するためのランタイムサポートを提供します。特に指定がない限り、すべての CASL 関数はクライアント側で機能します。すべてのサーバー側関数は、カテゴリに記載されています。関数はさまざまな方法で定義できます。

- CASL は、CASL プログラムのランタイムサポートを提供するビルトイン関数をサポートしています。
- FUNCTION ステートメントを使用すると、CASL 構文を使用して、式で呼び出される新しい関数を作成できます。詳細については、FUNCTION ステートメントを参照してください。
- CASL は共通 SAS 関数もサポートしています。

サポートされている共通 SAS 関数のリストについては、共通関数を参照してください。

一般的な CASL ビルトイン関数の構文

一般的な CASL ビルトイン関数の構文は、次の形式になります。

function-name (*argument*, ...<*argument*>);

function-name
CASL ビルトイン関数の名前。

argument
変数名、定数、式、または別の関数。CASL ビルトイン関数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。すべての関数に引数が含まれているわけではありません。引数が複数のときはカンマで区切ります。

注: 引数の値が無効な場合は、エラーが発生し、関数の戻り式が欠損値または null 値に設定されます。

関数の例外処理

例外ハンドラー関数を定義できます。例外ハンドラー関数はすべて、情報の辞書であるパラメーターを 1 つだけ受け取ります。辞書内で、例外のコンテキストとパラメーターが指定されます。

これは、浮動小数点エラーなどのシステム例外のハンドラーの例です。

```
function myfphandler(env) do;
put "error detected, using_MISSING_instead";
setexceptvalue(_MISSING_);
resume;
end do;
on_fpexception call fphandler;
```

前記の例では、エラーが見つかったときに"_MISSING_"を使用するようハンドラーに指示しています。SAS がデータでエラーまたは浮動小数点エラーを検出した場合、エラーが発生するたびに MISSING が表示されます。それ以外の場合は、表示されるはずの値で続行します。

関数カテゴリ

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

表 3.1　関数カテゴリの説明

カテゴリ	説明
テストカテゴリ	テスト
配列	同種データの名前付き集計コレクションを操作する関数 関数のリストについては、配列を参照してください。
コントロール	プログラムがプログラムフローを維持する方法を指定する関数 関数のリストについては、コントロールを参照してください。
辞書	辞書を操作する関数 関数のリストについては、辞書を参照してください。
ファイル IO	クライアント側またはサーバー側からのデータまたは結果の入出力に使用される関数 関数のリストについては、ファイル IO を参照してください。
関数 IO	名前、型、長さ、入力形式、出力形式、ラベル、その他の値情報など、入力値と出力値に関する情報を返す関数。 関数のリストについては、関数 IO を参照してください。
フォーマット	変数のデータ型を変更する関数 関数のリストについては、フォーマットを参照してください。
マクロ	CASL 内でマクロ変数を使用できるようにする関数 関数のリストについては、マクロを参照してください。
結果テーブル	結果テーブルの操作を可能にする関数 関数のリストについては、結果テーブルを参照してください。
サーバー側	サーバー側または非同期セッションで使用できる関数 関数のリストについては、サーバー側を参照してください。

カテゴリ	説明
ステータス	プログラムの実行状態またはプロセスフローを任意のタイミングで操作する関数 関数のリストについては、ステータスを参照してください。
変数情報	変数を操作し、名前、型、長さ、入力形式、ラベルなどの変数情報を返す関数 関数のリストについては、変数情報を参照してください。

次の表に、CASL 関数の簡単な説明を示します。詳細については、各関数を参照してください。

カテゴリ	言語要素	説明
関数 IO	CLEAR 関数 (p. 113)	指定されたすべての変数をクリアします。変数が指定されていない場合、すべての変数がクリアされます。
	EXISTS 関数 (p. 132)	変数が存在するかどうか、または辞書に指定されたキーが含まれているかどうかを確認します。
結果テーブル	ADD_TABLE_ATTR 関数 (p. 99)	結果テーブルに属性を追加します。
	ADDBYGROUP 関数 (p. 100)	BY グループテーブルから新しいテーブルを作成します。BY 変数は、各行の最初の変数として追加されます。BY グループ処理の属性は削除されます。
	ADDROW 関数 (p. 102)	テーブルに 1 つ以上の行を追加します。
	COMBINE_TABLES 関数 (p. 115)	最初のテーブルの名前を持ち、すべてのテーブルのすべての行を含む新しいテーブルを作成します。これが BY グループテーブルの場合は、テーブルの最初の変数として BY グループ変数を追加します。
	FINDTABLE 関数 (p. 134)	指定された変数で、最初に表示されたテーブルを検索します。これは、アクションの結果が結果テーブルのコピーを返す場合に役立ちます。
	GETCOLUMN 関数 (p. 137)	列を配列に抽出します。
	GETRESULT 関数 (p. 140)	ID またはタグに関連付けられた結果を取得します。
	LOC 関数 (p. 149)	指定された列で指定された値が見つかった行番号を返します。
	NEWTABLE 関数 (p. 151)	新しいテーブルを作成します。
	RESULT_BY_COL 関数 (p. 156)	指定された列で新しいテーブルを作成します。
	RESULT_BY_TYPE 関数 (p. 156)	型の指定に一致する列を持つ新しいテーブルを作成します。

カテゴリ	言語要素	説明
	TABCOLUMNS 関数 (p. 167)	結果テーブルから列を取得します。
	TABTYPES 関数 (p. 168)	結果テーブルから辞書にデータ型を抽出します。
コントロール	CANCELACTION 関数 (p. 106)	指定されたセッションで実行中のアクションをキャンセルします。
	EXECUTE 関数 (p. 131)	指定されたコードをコンパイルして実行します。
	EXIT 関数 (p. 133)	指定されたステータスで CASL コードのブロックを終了します。
	GETENV 関数 (p. 139)	指定された文字列の環境変数を返します
	SLEEP 関数 (p. 157)	指定した秒数の間、オペレーティングシステム(OS)をスリープさせます。
	TIMESTAMP 関数 (p. 170)	現在の日付と時間を返します。
	UUID 関数 (p. 176)	指定されたセッションのユニバーサル一意識別子(UUID)を返します。
サーバー側	CANCELACTIONS 関数 (p. 106)	指定されたセッションのリストで実行中のアクションをキャンセルします。
	CREATE_PARALLEL_SESSION 関数 (p. 125)	アクションの呼び出しと同じ ID でセッションを開始します。この関数を呼び出すと、複数のセッションを作成できます。
	LIST_PARALLEL_SESSIONS 関数 (p. 149)	並列セッションをリストします。
	SEND_RESPONSE 関数 (p. 157)	指定された結果をクライアントに送り返します。
	TERM PARALLEL SESSION 関数 (p. 169)	並列セッションを終了します。
	WAIT_FOR_NEXT_ACTION 関数 (p. 177)	アクションが完了するのを待ちます。
辞書	CASL2JSON 関数 (p. 107)	CASL 辞書のコンテンツを JSON 形式のテキストを含む文字列に変換します。
	DICTIONARY 関数 (p. 128)	指定された辞書でキーを検索し、見つかった場合は、関連付けられた値を返します。
	DIM 関数 (p. 129)	配列または辞書の次元を取得します。
	EXISTS 関数 (p. 132)	変数が存在するかどうか、または辞書に指定されたキーが含まれているかどうかを確認します。
	GETKEYS 関数 (p. 139)	辞書からキーを配列に抽出します。

カテゴリ	言語要素	説明
	GETVALUES 関数 (p. 142)	辞書から値を配列に抽出します。
	JSON2CASL 関数 (p. 146)	JSON 形式のテキストを含む文字列を CASL 辞書に変換します。
ステータス	CODETOSTATUS 関数 (p. 114)	アクションステータスを辞書に変換します。
	GETENV 関数 (p. 139)	指定された文字列の環境変数を返します
	MEMORYSTATS 関数 (p. 150)	CPU とメモリの統計情報を表示します。
	SLEEP 関数 (p. 157)	指定した秒数の間、オペレーティングシステム(OS)をスリープさせます。
	TIMESTAMP 関数 (p. 170)	現在の日付と時間を返します。
	UUID 関数 (p. 176)	指定されたセッションのユニバーサル一意識別子(UUID)を返します。
配列	ADDUNIQUE 関数 (p. 104)	値が配列内にまだ存在しない場合、配列に値を追加します。
	DIM 関数 (p. 129)	配列または辞書の次元を取得します。
	SORT 関数 (p. 158)	昇順で並べ替えたリストを返します。
	SORT REV 関数 (p. 159)	降順で並べ替えたリストを返します。
	UNIQUE 関数 (p. 173)	配列を検索して、値が配列に存在するかどうかを判別します。
ファイル IO	READFILE 関数 (p. 154)	指定されたファイル参照名のコンテンツを読み取って返します。
	READPATH 関数 (p. 155)	指定されたファイル名のコンテンツを読み取って返します。
フォーマット	FMTVAR 関数 (p. 136)	SAS 出力形式の代替として使用できる CASL 出力形式変数を作成します。
	INPUT_FORMAT 関数 (p. 142)	文字列を最適な数値に変換します。
	PUT 関数 (p. 153)	指定された出力形式で値をフォーマットし、それを文字列として返します。
変数情報	isType 関数 (p. 144)	expression が指定された型の値になるかに基づいて、TRUE または FALSE を返します。
マクロ	SYMDEL 関数 (p. 161)	指定されたマクロ変数を削除します。
	SYMGET 関数 (p. 161)	マクロ変数の値を返します。
	SYMPUT 関数 (p. 162)	SAS マクロ変数に値を保存します。

カテゴリ	言語要素	説明
	SYMPUTX 関数 (p. 163)	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。

ディクショナリ

ACTIONQ 関数

アクションキューを表示します。

構文

形式 1: **ACTIONQ**("*session-reference* ");
ACTIONQ("*session-reference* ", <"yes">);

必須引数

session-reference
アクションがキューに入れられるセッション名を指定します。

注: セッション名が指定されていない場合は、現在のアクティブなセッションが使用されます。

オプション引数

yes
指定すると、接続 ID とシーケンス番号が出力に表示されます。

デフォルト no

詳細

CASL を使用すると、サーバーに非同期でアクションをサブミットできます。アクションがサブミットされたセッションに対して実行されている場合、そのアクションはアクションキューに入れられます。必要な数のアクションをサブミットできますが、一度に実行できるアクションは 1 つだけです。残りのアクションは、キューに入れられた順序で実行されます。

例

例 1: アクションキューの表示

```
cas mySession1;
proc cas;
  session mySession1;
    session.listsessions async="session1";run;
    session.listsessions async="session2"; run;
    listsessions async="session3"; run;
  print actionq("mySession1");
  symputx("uuid",uuid("mySession1"));
  print symget("uuid");
run;
quit;
```

アウトプット 3.1　アクションキューの結果

Timestamp	Action	Action Parameter Count	Action Tag	Action is Active
10/29/19 17	session.listsessions	0	session1	Running
10/29/19 17	session.listsessions	0	session2	Active
10/29/19 17	session.listsessions	0	session3	Active

例 2: Yes 引数を使用したアクションキューの表示

```
cas mySession1;
proc cas;
  session mySession1;
    listsession async="session1"; run;
    listsession async="session2"; run;
    listsession async="session3"; run;
  print actionq("mySession1", "yes");
  symputx("uuid",uuid("mySession1"));
  print symget("uuid");
run;
quit;
```

アウトプット 3.2　アクションキューの結果

Connection ID	Sequence Number	Timestamp	Action	Action Parameter Count	Action Tag	Action is Active
12568	25	10/29/19 17	session.listsessions	0	session1	Running
12568	26	10/29/19 17	session.listsessions	0	session2	Active
12568	27	10/29/19 17	session.listsessions	0	session3	Active

ADD_TABLE_ATTR 関数

結果テーブルに属性を追加します。

カテゴリ:　結果テーブル

構文

ADD_TABLE_ATTR(*table-name*, *dictionary-1*<, *dictionary-2*, ..., *dictionary-n*>)

必須引数

table-name
属性が追加されるテーブルの名前を指定します。

dictionary
属性のキーと値のペアを含む辞書を指定します。

参照項目: “CASL 辞書” (*SAS Cloud Analytic Services: CASL プログラマガイド*)

例 次の例では、辞書リテラルを指定しています。

```
add_table_attr(table,{attrA="valueA"})
```

詳細

結果テーブル

結果テーブル属性とは、CAS 結果テーブルに設定できる値が関連付けられた名前付きキーです。CASL PRINT ステートメントを使用して、テーブル属性を表示できます。結果テーブル属性の出力を示す例については、“結果テーブルのプロパティ” (*SAS Cloud Analytic Services: CASL プログラマガイド*)を参照してください。

戻り値

ADD_TABLE_ATTR 関数で返される可能性のあるデータ値は次のとおりです。

0 ADD_TABLE_ATTR 関数は成功しました。

>0 ADD_TABLE_ATTR 関数はテーブル属性の追加に失敗しました。

ADDBYGROUP 関数

BY グループテーブルから新しいテーブルを作成します。BY 変数は、各行の最初の変数として追加されます。BY グループ処理の属性は削除されます。

カテゴリ: 結果テーブル

返されるデータの種類: 新しいテーブルの作成が成功しなかった場合、戻り値は 0 です。

構文

ADDBYGROUP (*result-table-name*);

必須引数

result-table-name
BY グループテーブルの名前を指定します。

例: ADDBYGROUP 関数の使用

simple.Summary アクションを使用して *Iris* データセットの単純統計量を生成し、groupBy オプションを使用して BY グループテーブルを 作成します。
ADDBYGROUP 関数を使用して、BY グループテーブルから 新しいテーブルを作成します。

```
proc cas;
  session casauto;
    table.loadTable result=r/
      caslib="sasdemo"
      path="iris.sashdat"
      casOut={name="iris" replace=true};
run;
proc cas;
  simple.Summary result=summTab / table={name="iris" groupBy="Species"};
run;
  print summTab;
run;
  NewTab=addbygroup(summTab."ByGroup1.Summary"n);
run;
  print NewTab;
run;
```

アウトプット 3.3　simple.Summary の結果: summTab の結果

summTab: Results from simple.summary

Species=Setosa

Descriptive Statistics for IRIS																
Column	Minimum	Maximum	N	Sum	Mean	Std Dev	Std Error	Variance	Coeff of Variation	Corrected SS	USS	t Value	Pr > |t|	N Miss	Skewness	Kurtosis
SepalLength	43.0000	58.0000	50	2503	50.0600	3.5249	0.4985	12.4249	7.0413	608.82	125909	100.42	<.0001	0	0.1201	-0.2527
SepalWidth	23.0000	44.0000	50	1714	34.2800	3.7906	0.5361	14.3690	11.0579	704.08	59460	63.95	<.0001	0	0.04117	0.9547
PetalLength	10.0000	19.0000	50	731	14.6200	1.7366	0.2456	3.0159	11.8785	147.78	10835	59.53	<.0001	0	0.1064	1.0216
PetalWidth	1.0000	6.0000	50	123	2.4600	1.0539	0.1490	1.1106	42.8397	54.4200	357.00	16.51	<.0001	0	1.2539	1.7191

summTab: Results from simple.summary

Species=Versicolor

Descriptive Statistics for IRIS																
Column	Minimum	Maximum	N	Sum	Mean	Std Dev	Std Error	Variance	Coeff of Variation	Corrected SS	USS	t Value	Pr > |t|	N Miss	Skewness	Kurtosis
SepalLength	49.0000	70.0000	50	2968	59.3600	5.1617	0.7300	26.6433	8.6956	1305.52	177486	81.32	<.0001	0	0.1054	-0.5330
SepalWidth	20.0000	34.0000	50	1385	27.7000	3.1380	0.4438	9.8469	11.3285	482.50	38847	62.42	<.0001	0	-0.3628	-0.3662
PetalLength	30.0000	51.0000	50	2130	42.6000	4.6991	0.6646	22.0816	11.0308	1082.00	91820	64.10	<.0001	0	-0.6065	0.04790
PetalWidth	10.0000	18.0000	50	663	13.2600	1.9775	0.2797	3.9106	14.9135	191.62	8983.00	47.41	<.0001	0	-0.03118	-0.4101

summTab: Results from simple.summary

Species=Virginica

Descriptive Statistics for IRIS

Column	Minimum	Maximum	N	Sum	Mean	Std Dev	Std Error	Variance	Coeff of Variation	Corrected SS	USS	t Value	Pr > |t|	N Miss	Skewness	Kurtosis
SepalLength	49.0000	79.0000	50	3294	65.8800	6.3588	0.8993	40.4343	9.6521	1981.28	218990	73.26	<.0001	0	0.1180	0.03290
SepalWidth	22.0000	38.0000	50	1487	29.7400	3.2250	0.4561	10.4004	10.8439	509.62	44733	65.21	<.0001	0	0.3659	0.7061
PetalLength	45.0000	69.0000	50	2776	55.5200	5.5189	0.7805	30.4588	9.9405	1492.48	155616	71.13	<.0001	0	0.5494	-0.1538
PetalWidth	14.0000	25.0000	50	1013	20.2600	2.7465	0.3884	7.5433	13.5563	369.62	20893	52.16	<.0001	0	-0.1295	-0.6023

アウトプット 3.4　ADDBYGROUP 関数の結果: NewTab の結果

NewTab: Results from simple.summary

Species=Setosa

Descriptive Statistics for IRIS

Column	Minimum	Maximum	N	Sum	Mean	Std Dev	Std Error	Variance	Coeff of Variation	Corrected SS	USS	t Value	Pr > |t|	N Miss	Skewness	Kurtosis
SepalLength	43.0000	58.0000	50	2503	50.0600	3.5249	0.4985	12.4249	7.0413	608.82	125909	100.42	<.0001	0	0.1201	-0.2527
SepalWidth	23.0000	44.0000	50	1714	34.2800	3.7906	0.5361	14.3690	11.0579	704.08	59460	63.95	<.0001	0	0.04117	0.9547
PetalLength	10.0000	19.0000	50	731	14.6200	1.7366	0.2456	3.0159	11.8785	147.78	10835	59.53	<.0001	0	0.1064	1.0216
PetalWidth	1.0000	6.0000	50	123	2.4600	1.0539	0.1490	1.1106	42.8397	54.4200	357.00	16.51	<.0001	0	1.2539	1.7191

ADDROW 関数

テーブルに 1 つ以上の行を追加します。

カテゴリ:　結果テーブル

返されるデータの種類:　CAS テーブル

構文

ADDROW (*table*, *row-1*<, *row-2*, ... *row-N*>);

必須引数

table
行を追加するテーブルを指定します。この引数は、結果がテーブルになる任意の式にすることができます。

オプション引数

row-1, row-2, ..., row-N
テーブルに行として追加する列値のリストを 1 つ以上指定します。各行引数は、結果が値のリストになる任意の式にすることができます。リストは配列として扱われ、リスト内の値は配列インデックスの昇順でテーブル行の列に割り当てられます。

注 1 回の関数呼び出しで複数の行を追加する方が、複数の関数呼び出し(行ごとに 1 回の呼び出し)を指定するよりも効率的です。

詳細

戻り値

ADDROW 関数で返される可能性のあるデータ値は次のとおりです。

表 3.2 ADDROW 関数の戻り値

リターン	説明
1	ADDROW 関数は成功しました。
CAS テーブル	ADDROW 関数は、1 つまたは複数の行をテーブルに追加できませんでした。

例

例 1: 結果テーブルに 3 行を追加する

job.result.tableName、*job.result.caslib*、および *job.status.severity* の行がテーブル *results* に追加されます。

```
addrow(results,{job.result.tableName, job.result.caslib, job.status.severity});
```

例 2: 新しいテーブルを作成して複数の行を追加する

この例では、NEWTABLE 関数を使用してテーブルを作成し、ADDROW 関数を使用してテーブルに複数の行を追加する方法を示します。プログラムは、NEWTABLE 関数と ADDROW 関数の両方で、行の値の引数として式を使用します。この例では、1 つの ADDROW 関数で複数の行を効率的に追加する方法も示しています。

```
proc cas;

table_array[1] = newtable("Table1",{"x","y","z"},{"double","double","double"});
table_array[2] = newtable("Table2",{"x","y","z"},{"double","double","double"});

row_array = {
  { 1, 2, 3 },
  { 4, 5, 6 },
  { "x"=7, "z"=8, "y"=9 },
  { "z"=10, "y"=11, "x"=12 },
  { "y"=13, "z"=14, "x"=15 },
  { "x"=16, "z"=17, "y"=18 }
};
```

```
addrow( table_array[1], row_array[1] );
addrow( table_array[1], row_array[2] );
addrow( table_array[1], row_array[3] );

addrow( table_array[2], row_array[4] );
addrow( table_array[2], row_array[5] );
addrow( table_array[2], row_array[6] );

addrow( table_array[2], row_array[1], row_array[2], row_array[3] );

describe table_array[1];
print table_array[1];
run;

describe table_array[2];
print table_array[2];
run;

quit;
```

ADDUNIQUE 関数

値が配列内にまだ存在しない場合、配列に値を追加します。

カテゴリ: 配列

返されるデータの種類: 関数が値を配列に正常に追加した場合は、ブール値 TRUE が返されます。それ以外の場合は、FALSE が返されます。

構文

ADDUNIQUE (*array*, *value*);

必須引数

array
カウント可能な数の値を指定します。

value
文字または数値の定数、変数または式を指定します。値が数値の場合、SAS は BEST.出力形式を使用してその値を文字列に変換します。

base64Decode 関数

文字列、整数、double 配列、バイナリオブジェクトを Base64 にデコードします。

構文

base64Decode("*name*")

必須引数

name
文字列、整数、double 配列、バイナリオブジェクトのデコードが必要な変数または式の名前を指定します。

詳細

BASE64DECODE 関数は、指定された文字列内のすべての空白を無視します。

関連項目

- “base64Encode 関数”
- “unpackData 関数”

base64Encode 関数

文字列、整数、二重配列、バイナリオブジェクトを Base64 にエンコードします。

構文

base64Encode("*name*")

必須引数

name
文字列、整数、double 配列、バイナリオブジェクトのエンコードが必要な変数または式の名前を指定します。

関連項目

- “base64Decode 関数”
- “unpackData 関数”

CANCELACTION 関数

指定されたセッションで実行中のアクションをキャンセルします。

カテゴリ: コントロール

操作: この関数はクライアント側でも機能します。

返されるデータの種類: 関数がアクションを正常にキャンセルした場合は、ブール値 TRUE が返されます。それ以外の場合は、FALSE が返されます。

構文

CANCELACTION (“*session-reference*”);

必須引数

session-reference
アクションが実行されているセッション名を指定します。

CANCELACTIONS 関数

指定されたセッションのリストで実行中のアクションをキャンセルします。

カテゴリ: サーバー側

操作: この関数はクライアント側でも機能します。

構文

CANCELACTIONS ({*list-of-session-references*});

必須引数

session-reference
アクションが実行されているセッションのリストを指定します。複数のセッションを指定できます。

CASL2JSON 関数

CASL 辞書のコンテンツを JSON 形式のテキストを含む文字列に変換します。

カテゴリ: 辞書

操作: “相互作用” (107 ページ)を参照してください。

構文

CASL2JSON(*dictionary*)

必須引数

dictionary
変換する辞書を指定します。

詳細

戻り値

CASL2JSON 関数で返される可能性のあるデータ値は次のとおりです。

string JSON 形式のテキストを含む文字列

FALSE 辞書の JSON 文字列への変換に失敗しました。

相互作用

CASL_STD_JSON 環境変数と STDJSON オプションの設定は、CASL2JSON 関数によって生成されるテキストが JSON 標準に準拠するかどうかに影響します。

デフォルトでは、既存のコードの下位互換性のために、CASL2JSON 関数によって生成されたテキストは JSON 標準に準拠していない可能性があります。JSON 標準に準拠していないテキストは、サードパーティのアプリケーションで処理するときに問題が発生する可能性があります。この非準拠の動作は、次の方法で管理できます。

- STDJSON オプションを CASL セッションで設定します。これは、SAS Compute Server (SAS クライアント)または CAS サーバーのどちらで実行されるかに関係なく、CASL プログラムが実行されている環境にのみ影響するローカル設定です。
  - SAS クライアントで STDJSON オプションを設定するには、PROC CAS コードブロックで set stdjson または unset stdjson を指定します。

```
proc cas;
  set stdjson;
run; quit;
```

SAS クライアント上で実行されているプログラムで STDJSON 引数を設定すると、SAS クライアント上の関数の動作にのみ影響します。

- CAS サーバーで STDJSON オプションを設定するには、sccasl.runCasl アクションにサブミットされるコードに set stdjson または unset stdjson が含まれていることを確認してください。

  CAS サーバー上で実行されているプログラムで STDJSON 引数を設定すると、CAS サーバー上の関数の動作にのみ影響します。

- CASL_STD_JSON 環境変数を設定します。環境変数は、SAS Compute Server (SAS クライアント)または CAS サーバーで設定できます。これらの環境のいずれかで環境変数を設定すると、グローバルな効果があり、各サーバーで実行するためにサブミットされたすべての CASL プログラムに適用されます。

  - SAS Compute Server で CASL_STD_JSON 環境変数を設定するには、SAS Environment Manager の計算サービスの sas.compute.server: startup_commands 構成インスタンスに環境変数を追加します。

    ```
    # to enable the JSON standard
        export CASL_STD_JSON = true
    # to disable the JSON standard
        export CASL_STD_JSON = false
    ```

    SAS Compute Server 構成インスタンス(sas.compute.server: startup_commands)での環境変数の設定は、SAS クライアントで実行されている CASL コードにのみ影響します。

  - CAS サーバーで CASL_STD_JSON 環境変数を設定するには、SAS Environment Manager の cas-shared-default サービスの sas.cas.instance.config: settings 構成インスタンスに環境変数を追加します。

    ```
    # to enable the JSON standard
        export CASL_STD_JSON = true
    # to disable the JSON standard
        export CASL_STD_JSON = false
    ```

    CAS サーバー構成インスタンス(sas.cas.instance.config: settings)での環境変数の設定は、CAS サーバーで実行されている CASL コードにのみ影響します。

  構成インスタンスでの環境変数の設定の詳細については、"Edit Server Configuration Instances" (*SAS Environment Manager: User's Guide*)を参照してください。

JSON の詳細については、RFC 8259 を参照してください。

CASL2JSON 関数と JSON2CASL 関数を同じプログラムで実行し、一方の関数の出力をもう一方の関数への入力として使用する場合は、STDJSON オプションを一貫して設定する必要があります。

- CASL2JSON 関数の実行時に STDJSON オプションが*設定されていない*場合は、JSON2CASL 関数の実行時にも設定しないでください。
- JSON2CASL 関数の実行時に STDJSON オプションが設定されて*いる*場合は、CASL2JSON 関数の実行時にも設定する必要があります。

例

例 1: Fetch アクションの結果を JSON に変換する

次の例では、Fetch アクションの結果を JSON に変換します。Fetch アクションの結果は、変数 FetchRes に辞書として保存されます。JSON は、変数 **json1** に文字列として保存されます。

```
cas casauto;                          /* 1 */
proc casutil;                         /* 2 */
  load data=sashelp.cars(obs=3
      keep=Make Model DriveTrain)
      casout="cars" outcaslib="casuser"
  replace;
run;

proc cas;
  table.fetch result=FetchRes /         /* 3 */
    table={name="cars",
          caslib="casuser"};
;
  describe FetchRes;                  /* 4 */
  json1 = casl2json(FetchRes);          /* 5 */
  describe json1;                     /* 6 */
  print json1;                        /* 7 */
run;
```

1 必要に応じて、セッション CASAUTO を作成します。

2 PROC CASUTIL を使用して、SAS データセットを CAS に転送します。

3 Fetch アクションの結果は、辞書として変数 FetchRes に保存されます。

4 DESCRIBE ステートメントで、変数 FetchRes の構造が SAS ログに出力されます。

5 CASL2JSON 関数で、変数 FetchRes の要約結果が JSON に変換されます。JSON は変数 Json1 に保存されます。

6 DESCRIBE ステートメントで、変数 Json1 の構造が SAS ログに出力されます。

7 PRINT ステートメントで、変数 Json1 のコンテンツが SAS ログに書き込まれます。変数 Json1 には、JSON に変換された Fetch アクションの結果が含まれています。

例のコード 3.1　JSON に変換された Fetch アクションの結果

```
{ "_cas_table_.Fetch" : { "attributes" : { "name" : "Fetch","label" : "Selected Rows from Table CARS" } ,
"metadata" : [ {
"name" : "_Index_","type" : "int64" } , { "name" : "Make","type" : "string" } , { "name" : "Model","type" :
"string" } , {
"name" : "DriveTrain","type" : "string" } ] , "rows" : [ [ "1" , "Acura" , " MDX          " , "All  " ] , [ "2" , "Acura" ,
" RSX Type S 2dr" , "Front" ] , [ "3" , "Acura" , " TSX 4dr      " , "Front" ] ] } }
```

例 2: Summary アクションの結果を JSON に変換する

次の例では、Summary アクションの結果を JSON に変換します。Summary アクションの結果は、変数 SummRes に辞書として保存されます。JSON は、変数 **json2** に文字列として保存されます。

```
cas casauto;                          /* 1 */
proc casutil;                         /* 2 */
  load data=sashelp.cars
    casout="cars" outcaslib="casuser"
  replace;
run;

proc cas;
  simple.summary result=SummRes /          /* 3 */
    table={name="cars",caslib="casuser"}
    ,inputs={"MPG_City","MPG_Highway"}
    ,subSet={"NMISS","MIN", "MEAN", "MAX"}
   ;
  describe SummRes;                     /* 4 */
  json2 = casl2json(SummRes);              /* 5 */
  describe json2;                       /* 6 */
  print json2;                          /* 7 */
run;
```

1 必要に応じて、セッション CASAUTO を作成します。

2 PROC CASUTIL を使用して、SAS データセットを CAS に転送します。

CAS エンジンを使用して、データを CAS サーバーに転送します。

3 Summary アクションの結果は、辞書として変数 SummRes に保存されます。

4 DESCRIBE ステートメントで、変数 SummRes の構造が SAS ログに出力されます。

5 CASL2JSON 関数で、変数 SummRes の要約結果が JSON に変換されます。JSON は変数 Json2 に保存されます。

6 DESCRIBE ステートメントで、変数 Json2 の構造が SAS ログに出力されます。

7 PRINT ステートメントで、変数 Json2 のコンテンツが SAS ログに書き込まれます。変数 Json2 には、JSON に変換された Summary アクションの結果が含まれています。

例のコード 3.2　JSON に変換された Summary アクションの結果


```
{ "_cas_table_.Summary" : { "attributes" : { "name" : "Summary","label" : "Descriptive Statistics for CARS" } ,
"metadata" : [
{ "name" : "Column","label" : "Analysis Variable","type" : "string" } , { "name" : "Min","label" :
"Minimum","format" :
"D8.4","type" : "double" } , { "name" : "Max","label" : "Maximum","format" : "D8.4","type" : "double" } ,
{ "name" :
"NMiss","label" : "N Miss","format" : "BEST10.","type" : "double" } , { "name" : "Mean","label" :
"Mean","format" : "D8.4","type"
: "double" } ] , "rows" : [ [ "MPG_City   " , "10" , "60" , "0" , "20.060747664" ] , [ "MPG_Highway" , "12" , "66" ,
"0" ,
"26.843457944" ] ] } }
```

例 3: Freq アクションの結果を JSON に変換する

次の例では、Freq アクションの結果を JSON に変換します。Freq アクションの結果は、変数 FreqRes に辞書として保存されます。変換された JSON は、変数 **json3** に文字列として保存されます。

```
cas casauto;                          /* 1 */
proc casutil;                         /* 2 */
  load data=sashelp.iris
    casout="iris" outcaslib="casuser"
  replace;
run;


proc cas;
  simple.freq result=FreqRes /
    inputs={"Species"},
    table={name="iris", caslib="casuser"};  /* 3 */
  describe FreqRes;                   /* 4 */
  json3 = casl2json(FreqRes);             /* 5 */
  describe json3;                     /* 6 */
  print json3;                        /* 7 */
run;
```

1 必要に応じて、セッション CASAUTO を作成します。

2 PROC CASUTIL を使用して、SAS データセットを CAS に転送します。

3 Freq アクションの結果は、変数 FreqRes に保存された辞書です。

4 DESCRIBE ステートメントで、変数 FreqRes の構造が SAS ログに出力されます。

5 CASL2JSON 関数で、変数 FreqRes の辞書が JSON に変換されます。JSON 文字列は変数 Json2 に保存されます。

6 DESCRIBE ステートメントで、変数 Json3 の構造が SAS ログに出力されます。

7 PRINT ステートメントで、変数 Json3 のコンテンツが SAS ログに書き込まれます。変数 Json3 には、JSON に変換された Freq アクションの結果が含まれています。

例のコード 3.3 JSON に変換された Freq アクションの結果


```
{ "_cas_table_.Frequency" : { "attributes" : { "name" : "Frequency","label" : "Frequency for IRIS" } ,
"metadata" : [ {
"name" : "Column","label" : "Analysis Variable","type" : "varchar" } , { "name" : "CharVar","label" : "Character
Value","type" :
"varchar" } , { "name" : "FmtVar","label" : "Formatted Value","type" : "varchar" } , { "name" : "Level","label" :
"Level","format" : "BEST12.","type" : "int64" } , { "name" : "Frequency","label" : "Frequency","format" :
"BEST12.","type" :
"double" } ] , "rows" : [ [ "Species" , "Setosa    " , "Setosa    " , "1" , "50" ] , [ "Species" , "Versicolor" ,
"Versicolor" , "2" , "50" ] , [ "Species" , "Virginica " , "Virginica " , "3" , "50" ] ] } }
```


関連項目

- “SET ステートメント” (68 ページ)
- “UNSET ステートメント” (74 ページ)

- “CAS LIBNAME エンジン” (*SAS Cloud Analytic Services: ユーザーガイド*)
- “DESCRIBE ステートメント” (21 ページ)
- “行の取得” (*SAS Viya プラットフォーム: システムプログラミングガイド*)
- “度数” (*SAS Visual Analytics: プログラミングガイド*)
- “JSON2CASL 関数” (146 ページ)
- “PRINT ステートメント” (57 ページ)
- “要約” (*SAS Visual Analytics: プログラミングガイド*)

CASLSTORE 関数

caslstore 値を作成します。

構文

形式 1: **CASLSTORE**({ CASLIB= *caslib-reference-name*, NAME= *caslstore-table-name*}<, { CASLIB=*caslib-reference-name-2*, NAME= *caslstore-table-name-2*}... , {CASLIB= *caslib-reference-name-n*, NAME=*caslstore-table-name-n*}>);

形式 2: **CASLSTORE**({PATH=*pathname*} <, {PATH=*pathname-2*}, ..., {PATH=*pathname-n*}>);

必須引数

caslib-reference-name
caslstore が保存される caslib を指定します。

caslstore-table-name
caslstore テーブルの名前を指定します。

pathname
caslstore が保存されるパスを指定します。

詳細

caslstore は、CAS サーバーにユーザー定義関数を保存するための CASL コードリポジトリです。CASLSTORE 関数を使用すると、UPLOAD_CASLSTORE 関数を使用して CAS にアップロードされたユーザー定義関数をプログラムで使用できるようになります。caslstore は、プログラムの実行後は保持されないため、プログラムが実行されるたびに、ユーザー定義関数を使用するすべてのプログラム内で定義する必要があります。

例

```
proc cas;
  source funcs;
    function c2f(c);
      /*(°C×9/5)+32*/
      return ((c*9/5)+32);
    end;
    function f2c(f);
      /* (°F−32)×5/9 */
      return ((f-32)*5/9);
    end;
  endsource;
  upload_caslstore({caslib="casuser",
name="store1", replace=true}, funcs);
run;

proc cas;
  cs = caslstore({caslib="casuser", name="store1"});
  DegF=37.0;
  DegC=37.0;
  f=round(c2f(DegC),.1);
  c=round(f2c(DegF),.1);
  print "NOTE: " degF "°F is " c "°C.";
  print "NOTE: " degC "°C is " f "°F.";
run;
```

例のコード 3.4　ログに返される結果:

```
NOTE: 37 °F is 2.8 °C.
NOTE: 37 °C is 98.6 °F.
```

関連項目

- “DEFAULT_CASLSTORE 関数”
- “UPLOAD_CASLSTORE 関数”

CLEAR 関数

指定されたすべての変数をクリアします。変数が指定されていない場合、すべての変数がクリアされます。

カテゴリ:　関数 IO

構文

CLEAR (<*variable-1*<, *variable-2 ...*, *variable-n*>>);

オプション引数

variable
変数名を指定する文字列。

詳細

戻り値

CLEAR 関数で返される可能性のあるデータ値は次のとおりです。

0 変数がクリアされました。

CODETOSTATUS 関数

アクションステータスを辞書に変換します。

カテゴリ: ステータス

返されるデータの種類: 失敗した場合は値 0 を返します。

構文

CODETOSTATUS (*status_code*);

必須引数

status_code
サーバーからのステータスを表す番号。

詳細

辞書に返される詳細は、ステータスコードから派生した値です。次の情報が辞書に含まれています。

Status
CAS はステータスを番号で返します。

0
失敗

Group
ステータスが定義されているメッセージファイル。番号は 0-1000 です。

Number
メッセージファイルの番号。

Msg
ステータスコードは文字列に変換され、メッセージファイルで使用されます。

例: CODETOSTATUS 関数の使用

次の例は、CODETOSTATUS 関数を使用してアクションステータスを取得し、そのステータスを辞書に配置する方法を示しています。CODETOSTATUS 関数には、情報を取得するためのステータスコードが必要です。ステータスコードが取得する情報は、ステータス、グループ、メッセージ番号、メッセージです。

```
proc cas;
  x=codetostatus(5280316);
  describe x;
  print put(x, hex8.);
run;
```

次のように SAS ログに出力されます。

例のコード 3.5　SAS ログ

```
dictionary ( 4 entries, 4 used);
[status] int64_t;
[group] int64_t;
[number] int64_t;
[msg] string;
{status=90BFC13C,group=00000210,number=0000013C,msg=The action has been cancelled.}
```

関連項目

“EXIT 関数”

COMBINE_TABLES 関数

最初のテーブルの名前を持ち、すべてのテーブルのすべての 行を含む新しいテーブルを作成します。これが BY グループテーブルの 場合は、テーブルの最初の変数として BY グループ変数を追加します。

カテゴリ:　結果テーブル

構文

COMBINE_TABLES (*list-of-tables*, <*"key-pattern"*>);

必須引数

list-of-tables
結合するテーブルのリスト。

オプション引数

key-pattern
list-of-tables 引数に送られるテーブルの名前とマッチングします。キーパターンを指定した場合、キーパターンでは、そのキーがテーブルのキー内のパターンと一致するテーブルのみが保持されます。

注: ByGroupInfo テーブルは無視されます。

詳細

戻り値

COMBINE_TABLES 関数で返される可能性のあるデータ値は次のとおりです。

CAS 結果テーブル	関数が成功した場合、返されるデータの種類は CAS 結果テーブルです。
0	関数が成功せず、データの種類 0 が返された場合は、引数が多すぎます。
3	関数が成功せず、データの種類 3 が返された場合は、テーブルの結合ができません。

例

例 1: COMBINE_TABLES 関数の使用

simple.mdSummary アクションを使用して、cars データセット内の列の記述要約統計量を生成し、groupBy パラメーターを使用して、結果を変数 Origin と Type の一意の組み合わせによって別のテーブルに保存します。COMBINE_TABLES 関数を使用して、すべてのテーブルのすべての 行を含む新しいテーブルを作成します。

```
data casuser.cars;                    /* 1 */
   set sashelp.cars;
run;

proc cas;
```

```
  simple.mdSummary result=summTab /              /* 2 */
    table={caslib="casuser",
           name="cars",
           groupBy={"origin", "type"}
    };
  describe summTab;                              /* 3 */
  title "Results of simple.mdSummary action";
  print summTab;                                 /* 4 */
run;
  combTable = combine_tables(summTab);           /* 5 */
  title "Results of combine_tables function";
  print combTable;                               /* 6 */
  saveresult combTable dataout=work.combTable;   /* 7 */
run;
quit;

title "Combined Table";
proc print data=work.combTable;                  /* 8 */
run;
```

1 cars テーブルを SASHELP から Casuser ライブラリにロードします。

2 simple.mdSummary を使用して、グループ化変数の Origin と Type の一意の組み合わせごとに要約統計量を計算し、その結果を summTab という名前の変数に保存します。

3 DESCRIBE ステートメントを使用して、summTab のデータ型と値を表示します。ログの DESCRIBE ステートメントの出力は、summTab が 16 のエントリを持つ辞書であり、各エントリがテーブルであることを示しています。最初のテーブルは ByGroupInfo テーブルです。

4 PRINT ステートメントを使用して、計算されたグループ化テーブルを表示します。結果には、Origin と Type の一意の組み合わせによる要約統計量を含む 15 のテーブルが表示されます。

5 COMBINE_TABLES 関数を summTab 変数を引数として使用して、テーブルを結合します。結果を combTable として保存します。

6 PRINT ステートメントを使用して、combTable を表示します。結果は、キーに ByGroupInfo が含まれている summTab 辞書の最初のテーブルを除いて、15 のテーブルすべてが 結合されたことを示しています。

7 saveresult を使用して、combTable を work.combTable として保存します。

8 work.combTable で PROC PRINT を使用して、summTab 内のすべてのテーブル (ByGroupInfo テーブルを除く)のすべての行を含む結合テーブルを出力します。これには、Origin と Type の列が含まれるようになりました。

アウトプット 3.5　DESCRIBE ステートメントからの部分的なログ出力

```
dictionary ( 16 entries, 16 used);
 [ByGroupInfo] Table ( [15] Rows [5] columns
Column Names:
 [1] Origin           [                ] (char)
 [2] Origin_f         [                ] (char)
 [3] Type             [                ] (char)
 [4] Type_f           [                ] (char)
 [5] _key_            [                ] (varchar)
 [ByGroup1.MDSummary] Table[MDSummary] ( [10] Rows [15] columns
Column Names:
 [1] Column           [Analysis Variable] (char)
 [2] Min              [Minimum         ] (double) [D8.4]
 [3] Max              [Maximum         ] (double) [D8.4]
 [4] N                [N               ] (double) [BEST10.]
 [5] NMiss            [N Miss          ] (double) [BEST10.]
 [6] Mean             [Mean            ] (double) [D8.4]
 [7] Sum              [Sum             ] (double) [BEST10.]
 [8] Std              [Std Dev         ] (double) [D8.4]
 [9] StdErr           [Std Error       ] (double) [D8.4]
 [10] Var              [Variance        ] (double) [D8.4]
 [11] USS              [USS             ] (double) [D8.4]
 [12] CSS              [Corrected SS    ] (double) [D8.4]
 [13] CV               [Coeff of Variation] (double) [D8.4]
 [14] TValue           [t Value         ] (double) [7.2]
 [15] ProbT            [Pr > |t|        ] (double) [PVALUE6.4]
 [ByGroup2.MDSummary] Table[MDSummary] ( [10] Rows [15] columns
Column Names:
 [1] Column           [Analysis Variable] (char)
 [2] Min              [Minimum         ] (double) [D8.4]
 [3] Max              [Maximum         ] (double) [D8.4]
 [4] N                [N               ] (double) [BEST10.]
 [5] NMiss            [N Miss          ] (double) [BEST10.]
 [6] Mean             [Mean            ] (double) [D8.4]
 [7] Sum              [Sum             ] (double) [BEST10.]
 [8] Std              [Std Dev         ] (double) [D8.4]
 [9] StdErr           [Std Error       ] (double) [D8.4]
 [10] Var              [Variance        ] (double) [D8.4]
 [11] USS              [USS             ] (double) [D8.4]
 [12] CSS              [Corrected SS    ] (double) [D8.4]
 [13] CV               [Coeff of Variation] (double) [D8.4]
 [14] TValue           [t Value         ] (double) [7.2]
 [15] ProbT            [Pr > |t|        ] (double) [PVALUE6.4]
.
.
.
.
.
```

アウトプット 3.6　最初の PRINT ステートメントの部分的な結果

Results of simple.mdSummary action

summTab: Results from simple.mdSummary

Origin=Asia Type=Hybrid

Descriptive Statistics for CARS

Column	Minimum	Maximum	N	Sum	Mean	Std Dev	Std Error	Variance	Coeff of Variation	Corrected SS	USS	t Value	Pr > |t|	N Miss
MSRP	19110	20510	3	59760	19920	725.47	418.85	526300	3.6419	1052600	1.1915E9	47.56	0.0004	0
Invoice	17911	18926	3	55288	18429	507.85	293.21	257908	2.7556	515817	1.0194E9	62.85	0.0003	0
EngineSize	1.4000	2.0000	3	4.9	1.6333	0.3215	0.1856	0.1033	19.6809	0.2067	8.2100	8.80	0.0127	0
Cylinders	3.0000	4.0000	3	11	3.6667	0.5774	0.3333	0.3333	15.7459	0.6667	41.0000	11.00	0.0082	0
Horsepower	73.0000	110.00	3	276	92.0000	18.5203	10.6927	343.00	20.1307	686.00	26078	8.60	0.0132	0
MPG_City	46.0000	60.0000	3	165	55.0000	7.8102	4.5092	61.0000	14.2005	122.00	9197.00	12.20	0.0067	0
MPG_Highway	51.0000	66.0000	3	168	56.0000	8.6603	5.0000	75.0000	15.4647	150.00	9558.00	11.20	0.0079	0
Weight	1850.00	2890.00	3	7472	2490.67	560.43	323.56	314081	22.5012	628163	19238424	7.70	0.0165	0
Wheelbase	95.0000	106.00	3	304	101.33	5.6862	3.2830	32.3333	5.6114	64.6667	30870	30.87	0.0010	0
Length	155.00	175.00	3	505	168.33	11.5470	6.6667	133.33	6.8596	266.67	85275	25.25	0.0016	0

Results of simple.mdSummary action

summTab: Results from simple.mdSummary

Origin=Asia Type=Sedan

Descriptive Statistics for CARS

Std	Coeff of	Corrected

アウトプット 3.7　2 番目の PRINT ステートメントの部分的な結果

Results of combine_tables function

combTable: Results from simple.mdSummary

Origin=Asia Type=Hybrid

MDSummary

Column	Minimum	Maximum	N	Sum	Mean	Std Dev	Std Error	Variance	Coeff of Variation	Corrected SS	USS	t Value	Pr > |t|	N Miss
MSRP	19110	20510	3	59760	19920	725.47	418.85	526300	3.6419	1052600	1.1915E9	47.56	0.0004	0
Invoice	17911	18926	3	55288	18429	507.85	293.21	257908	2.7556	515817	1.0194E9	62.85	0.0003	0
EngineSize	1.4000	2.0000	3	4.9	1.6333	0.3215	0.1856	0.1033	19.6809	0.2067	8.2100	8.80	0.0127	0
Cylinders	3.0000	4.0000	3	11	3.6667	0.5774	0.3333	0.3333	15.7459	0.6667	41.0000	11.00	0.0082	0
Horsepower	73.0000	110.00	3	276	92.0000	18.5203	10.6927	343.00	20.1307	686.00	26078	8.60	0.0132	0
MPG_City	46.0000	60.0000	3	165	55.0000	7.8102	4.5092	61.0000	14.2005	122.00	9197.00	12.20	0.0067	0
MPG_Highway	51.0000	66.0000	3	168	56.0000	8.6603	5.0000	75.0000	15.4647	150.00	9558.00	11.20	0.0079	0
Weight	1850.00	2890.00	3	7472	2490.67	560.43	323.56	314081	22.5012	628163	19238424	7.70	0.0165	0
Wheelbase	95.0000	106.00	3	304	101.33	5.6862	3.2830	32.3333	5.6114	64.6667	30870	30.87	0.0010	0
Length	155.00	175.00	3	505	168.33	11.5470	6.6667	133.33	6.8596	266.67	85275	25.25	0.0016	0
MSRP	1028[illegible]	[illegible]750	94	2139813	22764	9613.14	991.52	92412548	42.2297	8.5944E9	5.731E10	22.96	< .0001	0

アウトプット 3.8　PRINT プロシジャの部分的な結果

Combined Table

Obs	Origin	Type	Column	Min	Max	N	NMiss	Mean	Sum	Std	StdErr	Var	USS	CSS	CV	TValue	ProbT
1	Asia	Hybrid	MSRP	19110	20510	3	0	19920	59760	725.47	418.85	526300	1.1915E9	1052600	3.6419	47.56	0.0004
2	Asia	Hybrid	Invoice	17911	18926	3	0	18429	55288	507.85	293.21	257908	1.0194E9	515817	2.7556	62.85	0.0003
3	Asia	Hybrid	EngineSize	1.4000	2.0000	3	0	1.6333	4.9	0.3215	0.1856	0.1033	8.2100	0.2067	19.6809	8.80	0.0127
4	Asia	Hybrid	Cylinders	3.0000	4.0000	3	0	3.6667	11	0.5774	0.3333	0.3333	41.0000	0.6667	15.7459	11.00	0.0082
5	Asia	Hybrid	Horsepower	73.0000	110.00	3	0	92.0000	276	18.5203	10.6927	343.00	26078	686.00	20.1307	8.60	0.0132
6	Asia	Hybrid	MPG_City	46.0000	60.0000	3	0	55.0000	165	7.8102	4.5092	61.0000	9197.00	122.00	14.2005	12.20	0.0067
7	Asia	Hybrid	MPG_Highway	51.0000	66.0000	3	0	56.0000	168	8.6603	5.0000	75.0000	9558.00	150.00	15.4647	11.20	0.0079
8	Asia	Hybrid	Weight	1850.00	2890.00	3	0	2490.67	7472	560.43	323.56	314081	19238424	628163	22.5012	7.70	0.0165
9	Asia	Hybrid	Wheelbase	95.0000	106.00	3	0	101.33	304	5.6862	3.2830	32.3333	30870	64.6667	5.6114	30.87	0.0010
10	Asia	Hybrid	Length	155.00	175.00	3	0	168.33	505	11.5470	6.6667	133.33	85275	266.67	6.8596	25.25	0.0016
11	Asia	Sedan	MSRP	10280	55750	94	0	22764	2139813	9613.14	991.52	92412548	5.731E10	8.5944E9	42.2297	22.96	<.0001
12	Asia	Sedan	Invoice	9875.00	48583	94	0	20788	1954101	8363.51	862.63	69948245	4.713E10	6.5052E9	40.2318	24.10	<.0001

例 2: COMBINE_TABLES 関数の使用とキーパターンの指定

simple.mdSummary アクションを使用して、cars データセット内の列の記述要約統計量を生成し、groupBy パラメーターを使用して、結果を変数 Origin と Type の一意の組み合わせによって別のテーブルに保存します。COMBINE_TABLES 関数を使用して、パターン"2"に一致するキーを持つテーブルの行のみを含む新しいテーブルを作成します。

```
data casuser.cars;                          /* 1 */
   set sashelp.cars;
run;

proc cas;
  simple.mdSummary result=summTab /         /* 2 */
    table={caslib="casuser",
          name="cars",
          groupBy={"origin", "type"}
    };
  describe summTab;                         /* 3 */
  title "Results of the simple.mdSummary action";
  print summTab;                            /* 4 */
run;
  combTable = combine_tables(summTab, "2");     /* 5 */
  title "Results of the combine_tables function";
  print combTable;                          /* 6 */
  saveresult combTable dataout=work.combTable;    /* 7 */
run;
quit;

title "Combined Table";
proc print data=work.combTable;                 /* 8 */
run;
```

1 cars テーブルを SASHELP から Casuser ライブラリにロードします。

2 simple.mdSummary を使用して、グループ化変数の Origin と Type の一意の組み合わせごとに要約統計量を計算し、その結果を summTab という名前の変数に保存します。

3 DESCRIBE ステートメントを使用して、summTab のデータ型と値を表示します。ログの DESCRIBE ステートメントの出力は、summTab が 16 のエントリを持つ

辞書であり、各エントリがテーブルであることを示しています。最初のテーブルは ByGroupInfo テーブルです。

4 PRINT ステートメントを使用して、計算されたグループ化テーブルを表示します。結果には、Origin と Type の一意の組み合わせによる要約統計量を含む 15 のテーブルが表示されます。

5 COMBINE_TABLES 関数を summTab 変数を引数として使用して、テーブルを結合します。第 2 引数はキーパターン"2"として指定されているため、テーブルのキー内でキーがパターン"2"に一致するテーブルのみが保持されます。これには、キー 2 と 12 を持つテーブルが含まれます(ByGroup2.MDSummary と ByGroup12.MDSummary)。キーパターンが指定されている場合、ByGroupInfo テーブルは無視されます。

6 PRINT ステートメントを使用して、combTable を表示します。結果は、キーに 2 と 12 が含まれる 2 つのテーブルが結合されたことを示しています。

7 saveresult を使用して、combTable を work.combTable として保存します。

8 work.combTable で PROC PRINT を使用して、Origin と Type の列と、キー 2 と 12 を持つテーブル(ByGroup2.MDSummary と ByGroup12.MDSummary)の行を含む結合テーブルを出力します。

アウトプット 3.9 最初の PRINT ステートメントの部分的な結果

Results of the simple.mdSummary action

summTab: Results from simple.mdSummary

Origin=Asia Type=Hybrid

Descriptive Statistics for CARS

Column	Minimum	Maximum	N	Sum	Mean	Std Dev	Std Error	Variance	Coeff of Variation	Corrected SS	USS	t Value	Pr > |t|	N Miss
MSRP	19110	20510	3	59760	19920	725.47	418.85	526300	3.6419	1052600	1.1915E9	47.56	0.0004	0
Invoice	17911	18926	3	55288	18429	507.85	293.21	257908	2.7556	515817	1.0194E9	62.85	0.0003	0
EngineSize	1.4000	2.0000	3	4.9	1.6333	0.3215	0.1856	0.1033	19.6809	0.2067	8.2100	8.80	0.0127	0
Cylinders	3.0000	4.0000	3	11	3.6667	0.5774	0.3333	0.3333	15.7459	0.6667	41.0000	11.00	0.0082	0
Horsepower	73.0000	110.00	3	276	92.0000	18.5203	10.6927	343.00	20.1307	686.00	26078	8.60	0.0132	0
MPG_City	46.0000	60.0000	3	165	55.0000	7.8102	4.5092	61.0000	14.2005	122.00	9197.00	12.20	0.0067	0
MPG_Highway	51.0000	66.0000	3	168	56.0000	8.6603	5.0000	75.0000	15.4647	150.00	9558.00	11.20	0.0079	0
Weight	1850.00	2890.00	3	7472	2490.67	560.43	323.56	314081	22.5012	628163	19238424	7.70	0.0165	0
Wheelbase	95.0000	106.00	3	304	101.33	5.6862	3.2830	32.3333	5.6114	64.6667	30870	30.87	0.0010	0
Length	155.00	175.00	3	505	168.33	11.5470	6.6667	133.33	6.8596	266.67	85275	25.25	0.0016	0

Results of the simple.mdSummary action

summTab: Results from simple.mdSummary

Origin=Asia Type=Sedan

アウトプット 3.10　2 番目の PRINT ステートメントの部分的な結果

Results of the combine_tables function

combTable: Results from simple.mdSummary

Origin=Asia Type=Sedan

MDSummary

Column	Minimum	Maximum	N	Sum	Mean	Std Dev	Std Error	Variance	Coeff of Variation	Corrected SS	USS	t Value	Pr > |t|	N Miss
MSRP	10280	55750	94	2139813	22764	9613.14	991.52	92412548	42.2297	8.5944E9	5.731E10	22.96	<.0001	0
Invoice	9875.00	48583	94	1954101	20788	8363.51	862.63	69948245	40.2318	6.5052E9	4.713E10	24.10	<.0001	0
EngineSize	1.5000	4.5000	94	248.9	2.6479	0.7790	0.08035	0.6068	29.4194	56.4346	715.49	32.96	<.0001	0
Cylinders	4.0000	8.0000	94	474	5.0426	1.1632	0.1200	1.3530	23.0675	125.83	2516.00	42.03	<.0001	0
Horsepower	103.00	340.00	94	17106	181.98	57.2929	5.9093	3282.47	31.4833	305270	3418198	30.80	<.0001	0
MPG_City	16.0000	36.0000	94	2147	22.8404	4.9390	0.5094	24.3936	21.6239	2268.61	51307	44.84	<.0001	0
MPG_Highway	22.0000	44.0000	94	2817	29.9681	4.8846	0.5038	23.8592	16.2993	2218.90	86639	59.48	<.0001	0
Weight	2035.00	4802.00	94	297169	3161.37	584.29	60.2654	341400	18.4823	31750244	9.7121E8	52.46	<.0001	0
Wheelbase	93.0000	124.00	94	9931	105.65	6.4068	0.6608	41.0475	6.0643	3817.41	1053017	159.88	<.0001	0
Length	154.00	204.00	94	17297	184.01	10.4506	1.0779	109.21	5.6793	10157	3192989	170.71	<.0001	0
MSRP	18345	81795	9	407315	45257	21279	7093.15	4.5281E8	47.0189	3.6225E9	2.206E10	6.38	0.0002	0
Invoice	16943	74451	9	370302	41145	19430	6476.54	3.7751E8	47.2227					

アウトプット 3.11　PRINT プロシジャの部分的な結果

Combined Table

Obs	Origin	Type	Column	Min	Max	N	NMiss	Mean	Sum	Std	StdErr	Var	USS	CSS	CV	TValue	ProbT
1	Asia	Sedan	MSRP	10280	55750	94	0	22764	2139813	9613.14	991.52	92412548	5.731E10	8.5944E9	42.2297	22.96	<.0001
2	Asia	Sedan	Invoice	9875.00	48583	94	0	20788	1954101	8363.51	862.63	69948245	4.713E10	6.5052E9	40.2318	24.10	<.0001
3	Asia	Sedan	EngineSize	1.5000	4.5000	94	0	2.6479	248.9	0.7790	0.08035	0.6068	715.49	56.4346	29.4194	32.96	<.0001
4	Asia	Sedan	Cylinders	4.0000	8.0000	94	0	5.0426	474	1.1632	0.1200	1.3530	2516.00	125.83	23.0675	42.03	<.0001
5	Asia	Sedan	Horsepower	103.00	340.00	94	0	181.98	17106	57.2929	5.9093	3282.47	3418198	305270	31.4833	30.80	<.0001
6	Asia	Sedan	MPG_City	16.0000	36.0000	94	0	22.8404	2147	4.9390	0.5094	24.3936	51307	2268.61	21.6239	44.84	<.0001
7	Asia	Sedan	MPG_Highway	22.0000	44.0000	94	0	29.9681	2817	4.8846	0.5038	23.8592	86639	2218.90	16.2993	59.48	<.0001
8	Asia	Sedan	Weight	2035.00	4802.00	94	0	3161.37	297169	584.29	60.2654	341400	9.7121E8	31750244	18.4823	52.46	<.0001
9	Asia	Sedan	Wheelbase	93.0000	124.00	94	0	105.65	9931	6.4068	0.6608	41.0475	1053017	3817.41	6.0643	159.88	<.0001
10	Asia	Sedan	Length	154.00	204.00	94	0	184.01	17297	10.4506	1.0779	109.21	3192989	10157	5.6793	170.71	<.0001
11	USA	Sports	MSRP	18345	81795	9	0	45257	407315	21279	7093.15	4.5281E8	2.206E10	3.6225E9	47.0189	6.38	0.0002
12	USA	Sports	Invoice	16943	74451	9	0	41145	370302	19430	6476.54	3.7751E8	1.826E10	3.0201E9	47.2227	6.35	0.0002
13	USA	Sports	Engine...	...	8.3000	...	...	...	...	...	...	...	...	...	...	...	0.5075

例 3: COMBINE_TABLES 関数を使用した複数の ByGroupSet テーブルの結合

simple.mdSummary アクションを使用して、cars データセットの Horsepower 列の記述要約統計量を生成します。groupBy パラメーターを使用して、結果を Origin と、変数 Type と Drivetrain の重複しない組み合わせごとに別のテーブルに保存します。COMBINE_TABLES 関数を使用して、キー名に"ByGroupSetn"を含むすべてのテーブルを結合します。

```
data casuser.cars;                                  /* 1 */
 set sashelp.cars;
run;

proc cas;
  simple.mdSummary result=res status=s /                  /* 2 */
    table={name="cars"}
    inputs={"Horsepower"},
    subSet={"mean", "N"},
    sets={
        {groupBy={"origin"}}
```

```
        {groupBy={"type", "drivetrain"}}
    };
run;
describe res;                                    /* 3 */
run;
do i=1 to 2;                                     /* 4 */
 s=combine_tables(res, cats("ByGroupSet", i));
 table = cats("summary_set", i);
 saveresult s casout=table caslib="casuser" replace;
end;
run;
quit;
```

1 cars テーブルを SASHELP から Casuser ライブラリにロードします。

2 simple.mdSummary アクションを使用して、cars データセットの Horsepower 列の記述要約統計量(平均とカウント)を生成します。groupBy パラメーターを使用して、結果を Origin と、変数 Type と Drivetrain の重複しない組み合わせごとに別のテーブルに保存します。groupBy パラメーターが 2 つあるため、このアクションでは複数の ByGroupSet が作成されます。

3 DESCRIBE ステートメントを使用して、res のデータ型と値を表示します。ログの DESCRIBE ステートメントの出力は、res が 19 のエントリを持つ辞書であり、各エントリがテーブルであることを示しています。各 ByGroupSet の最初のテーブルは、ByGroupInfo テーブルです。

4 combo_tables 関数は、キー名に"ByGroupSet1"または"ByGroupSet2"を含むすべてのテーブルを結合します(ByGroupInfo は無視されます)。

アウトプット 3.12　DESCRIBE ステートメントからのログ出力

```
dictionary ( 19 entries, 19 used);
 [ByGroupSet1.ByGroupInfo] Table ( [3] Rows [3] columns
Column Names:
 [1] Origin           [               ] (char)
 [2] Origin_f         [               ] (char)
 [3] _key_            [               ] (varchar)
 [ByGroupSet1.ByGroup1.MDSummary] Table[MDSummary] ( [1] Rows [3] columns
Column Names:
 [1] Column           [Analysis Variable] (char)
 [2] N                [N              ] (double) [BEST10.]
 [3] Mean             [Mean           ] (double) [NLD8.4]
 [ByGroupSet1.ByGroup2.MDSummary] Table[MDSummary] ( [1] Rows [3] columns
Column Names:
 [1] Column           [Analysis Variable] (char)
 [2] N                [N              ] (double) [BEST10.]
 [3] Mean             [Mean           ] (double) [NLD8.4]
 [ByGroupSet1.ByGroup3.MDSummary] Table[MDSummary] ( [1] Rows [3] columns
Column Names:
 [1] Column           [Analysis Variable] (char)
 [2] N                [N              ] (double) [BEST10.]
 [3] Mean             [Mean           ] (double) [NLD8.4]
 [ByGroupSet2.ByGroupInfo] Table ( [14] Rows [5] columns
Column Names:
 [1] Type             [               ] (char)
 [2] Type_f           [               ] (char)
 [3] DriveTrain       [               ] (char)
 [4] DriveTrain_f     [               ] (char)
 [5] _key_            [               ] (varchar)
 [ByGroupSet2.ByGroup1.MDSummary] Table[MDSummary] ( [1] Rows [3] columns
Column Names:
 [1] Column           [Analysis Variable] (char)
 [2] N                [N              ] (double) [BEST10.]
 [3] Mean             [Mean           ] (double) [NLD8.4]
 [ByGroupSet2.ByGroup2.MDSummary] Table[MDSummary] ( [1] Rows [3] columns
Column Names:
 [1] Column           [Analysis Variable] (char)
 [2] N                [N              ] (double) [BEST10.]
 [3] Mean             [Mean           ] (double) [NLD8.4]
 [ByGroupSet2.ByGroup3.MDSummary] Table[MDSummary] ( [1] Rows [3] columns
Column Names:
 [1] Column           [Analysis Variable] (char)
 [2] N                [N              ] (double) [BEST10.]
 [3] Mean             [Mean           ] (double) [NLD8.4]
 [ByGroupSet2.ByGroup4.MDSummary] Table[MDSummary] ( [1] Rows [3] columns
Column Names:
 [1] Column           [Analysis Variable] (char)
 [2] N                [N              ] (double) [BEST10.]
 [3] Mean             [Mean           ] (double) [NLD8.4]
 [ByGroupSet2.ByGroup5.MDSummary] Table[MDSummary] ( [1] Rows [3] columns
Column Names:
 [1] Column           [Analysis Variable] (char)
 [2] N                [N              ] (double) [BEST10.]
 [3] Mean             [Mean           ] (double) [NLD8.4]
 [ByGroupSet2.ByGroup6.MDSummary] Table[MDSummary] ( [1] Rows [3] columns
Column Names:
 [1] Column           [Analysis Variable] (char)
 [2] N                [N              ] (double) [BEST10.]
 [3] Mean             [Mean           ] (double) [NLD8.4]
 [ByGroupSet2.ByGroup7.MDSummary] Table[MDSummary] ( [1] Rows [3] columns
Column Names:
 [1] Column           [Analysis Variable] (char)
 [2] N                [N              ] (double) [BEST10.]
 [3] Mean             [Mean           ] (double) [NLD8.4]
 [ByGroupSet2.ByGroup8.MDSummary] Table[MDSummary] ( [1] Rows [3] columns
Column Names:
 [1] Column           [Analysis Variable] (char)
 [2] N                [N              ] (double) [BEST10.]
 [3] Mean             [Mean           ] (double) [NLD8.4]
 [ByGroupSet2.ByGroup9.MDSummary] Table[MDSummary] ( [1] Rows [3] columns
Column Names:
```

アウトプット 3.13　ログ出力

```
NOTE: The table summary_set1 loaded into casuser with 3 observations and 4
variables.
NOTE: The table summary_set2 loaded into casuser with 14 observations and 5
variables.
```

CREATE_PARALLEL_SESSION 関数

アクションの呼び出しと同じ ID でセッションを開始します。この関数を呼び出すと、複数のセッションを作成できます。

カテゴリ: サーバー側

返されるデータの種類: 関数が成功しなかった場合、ブール値 FALSE が返されます。それ以外の場合、返されるデータの種類はセッション名の文字列です。

ヒント: アクション呼び出しでセッションを指定することもできます。

構文

CREATE_PARALLEL_SESSION (<*number-of-workers*>);

必須引数

number-of-workers
セッションを実行するワーカーノードの数を指定します。数を指定しない場合は、すべてのワーカーが使用されます。

デフォルト　すべてのワーカー

例

例 1: すべてのワーカーで 並列セッションを作成する

```
session[i] = create_parallel_session();
```

例 2: n ワーカーで並列セッションを作成する

```
session[i] = create_parallel_session(37);
```

DEFAULT_CASLSTORE 関数

デフォルトの caslstore を設定します。

参照項目: CASL プログラマガイドのユーザー定義関数

構文

DEFAULT_CASLSTORE(*caslstore-value*);

必須引数

caslstore-value
デフォルトの caslstore 値の名前を指定します。

詳細

デフォルトの caslstore は、修飾されていないユーザー定義関数を探すために検索されるコードリポジトリです。2 つの関数が同じ名前で、別々の caslstore に保存されている場合、DEFAULT_CASLSTORE 関数は、関数を最初に検索する caslstore を指定します。DEFAULT_CASLSTORE 関数を使用してデフォルトの caslstore が設定されていない場合は、最後に指定された caslstore で最初に関数が検索されます。デフォルトの caslstore は、新しいデフォルトの caslstore が設定されるまで保持されます。

例: DEFAULT_CASLTORE 関数を使用した caslstore 優先度の指定

この例では、"myfunction"という名前の 2 つの関数が別々の caslstore に保存されています。最初の"myfunction"は、入力値の階乗を計算します。2 番目の"myfunction"は、入力華氏温度を摂氏温度に変換します。プログラムで両方の caslstore が呼び出されると、最新の caslstore で最初に関数"myfunction"が検索されます。このプログラムでは、デフォルトの caslstore が設定されていない場合、"myfunction"が呼び出されたときに、最後に定義された caslstore である温度変換関数が使用されます。デフォルトの caslstore が設定されると、デフォルトの caslstore に保存されている"myfunction"が、"myfunction"が呼び出されたときに使用されます。この場合は階乗関数です。

```
proc cas;
source funcs;                                    /* 1 */
  function myfunction(x);
    if (x < 1.0) then return(x);
```

```
    else do;
      return exp(lgamma (x+1));
    end;
  end func;
endsource;

upload_caslstore({caslib="casuser",
   name="store1", replace=true}, funcs);

source funcs2;                                          /* 2 */
  function myfunction(temp);
    Celsius = (5/9*(temp-32));
    return(Celsius);
  end func;
endsource;

upload_caslstore({caslib="casuser",
   name="store2", replace=true}, funcs2);
run;

proc cas;
  cs1 = caslstore({caslib="casuser", name="store1"});
  cs2 = caslstore({caslib="casuser", name="store2"});
  dcs = default_caslstore(cs1);                         /* 3 */

  myresult=myfunction(10);
  print "10! = 3628800";
  print "10'F = -12.2'C";
  print "My function result= " myresult;
run;
```

1 最初の"myfunction"が定義され、"store1"という名前の caslstore に保存されます。

2 2 番目の"myfunction"が定義され、"store2"という名前の 2 番目の calstore に保存されます。

3 デフォルトの caslstore は、階乗関数"store1"に設定されています。

例のコード 3.6 ログに返される結果:

```
10! = 3628800
10'F = -12.2'C
My function result= 3628800
```

関連項目

- “CASLSTORE 関数”
- “UPLOAD_CASLSTORE 関数”
- “CASLstore” (*SAS Cloud Analytic Services: CASL プログラマガイド*)

DICTIONARY 関数

指定された辞書でキーを検索し、見つかった場合は、関連付けられた値を返します。

カテゴリ: 辞書

構文

DICTIONARY (*dictionary*, *string*);

必須引数

dictionary
検索する辞書を指定します。

string
辞書キーを指定します。

詳細

戻り値

DICTIONARY 関数で返される可能性のあるデータ値は次のとおりです。

FALSE キーが辞書内に見つからない、またはパラメーターエラーがあります。

string キーに関連付けらた値を返します。

例: DICTIONARY 関数の使用

次の例は、DICTIONARY 関数を使用して、辞書内のキーに関連付けられた値を取得する方法を示しています。この例では、*studentId* 辞書でキー *Keefer* を検索します。キー Keefer が辞書内に存在するため、DICTIONARY 関数はキーに関連付けられた 68101 の値を返します。

```
proc cas;
  studentID={Woods=13445, Shafer=36772, McKenna=37854, Keefer=68101,
Johnson=34690,          Shena=20047, Kingstem=60439, Downdry=37710};
  print 'StudentID= ' dictionary(studentID, 'Keefer');
run;
```

例のコード 3.7 SAS ログ

```
StudentId=68101
```

関連項目

"CASL 辞書" (*SAS Cloud Analytic Services: CASL プログラマガイド*)

DIM 関数

配列または辞書の次元を取得します。

カテゴリ: 配列
辞書

返されるデータの種類: INTEGER

構文

DIM (*array-name*);

必須引数

array-name
配列の名前を指定します。

詳細

DIM 関数は、1 次元配列にある要素数、または多次元配列内の指定した次元にある要素数を返します。

例: DIM 関数の使用

この例では、DIM 関数を使用して、*studentId* という名前の辞書の次元を取得しています。

```
proc cas;
  studentId={Woods=13445, Shafer=36772, McKenna=37854, Keefer=68101,
Johnson=34690,          Shena=20047, Kingstem=60439, Downdry=37710};
  print 'Dimensions are=' dim(studentId);
run;
```

例のコード 3.8 SAS ログ

```
Dimensions are=8
```

関連項目

- “CASL 配列” (*SAS Cloud Analytic Services: CASL プログラマガイド*)
- CASL 辞書

ENCODING 関数

現在のセッションエンコーディングのエンコーディング 文字列を返します。

データの種類: CHAR

構文

ENCODING( );

引数なし

ENCODING 関数には引数がありません。

詳細

UPLOAD ステートメントで CSV ファイルをアップロードする場合は、ENCODING= パラメーターを使用してファイルエンコーディングを指定する必要があります。ENCODING=パラメーターについては、“リソースのアップロード” (*SAS Viya プラットフォーム: システムプログラミングガイド*)の"Parameters for fileType="CSV""を参照してください。ENCODING 関数を使用して、セッションエンコーディングを ENCODING=パラメーターに動的に渡すことができます。次の IMPORTOPTION での CSV ファイルの指定では、ENCODING 関数を使用して、セッションエンコーディング文字列が返され、それが ENCODING=パラメーターに渡されます。
importoptions={filetype="CSV", encoding=encoding()};

例

オペレーティングシステムのエンコーディングが 日本語の場合、UPLOAD ステートメントの ENCODING 関数では、shift-jis エンコーディングでファイル Mycsv.csv がアップロードされます。

```
proc cas;
upload path="mycsv.csv" casout={name="casout"} importoptions={filetype="CSV",
encoding=encoding()};
run;
```

EXECUTE 関数

指定されたコードをコンパイルして実行します。

カテゴリ: コントロール

構文

EXECUTE("*code* ");

必須引数

code
コンパイルして実行する CASL コードを指定します。作成された関数はすべてアクティブなままです。

詳細

戻り値

EXECUTE 関数で返される可能性のあるデータ値は次のとおりです。

表 3.3 リターンコード

リターン	説明
1	コンパイルは成功しませんでした
0	コンパイルは成功しました

例: EXECUTE 関数の使用

```
proc cas;
  values=execute("x=10; y=50; r=x*y; print r;");
  print values;
run;
```

アウトプット 3.14 ログ

```
500
```

EXISTS 関数

変数が存在するかどうか、または辞書に指定されたキーが含まれているかどうかを確認します。

カテゴリ: 辞書
関数 IO

構文

形式 1: **EXISTS** (*dictionary*, *key*);

形式 2: **EXISTS** (*variable-name*);

必須引数

dictionary
検索する辞書を指定します。

key
辞書キーを指定する文字列。

variable-name
変数の名前を指定する文字列。

注 値が割り当てられていなくても、変数は存在できます。

詳細

戻り値

EXISTS 関数で返される可能性のあるデータ値は次のとおりです。

0 変数またはキーが存在しないか、パラメーターエラーがあります。

1 変数またはキーが存在します。

例

例 1: 既存の変数の確認

次の例では、EXISTS()関数は、変数 Name が存在するかを確認します。

```
proc cas;
Name = 'Jane Smith';
```

```
run;

print "Name variable exists: " exists("Name");
run;
quit;
```

戻り値 1 は、変数 Name が存在することを示します。

```
Name variable exists: 1
```

例 2: 既存のキーの確認

次の例では、EXISTS()関数は、hmeqTable という辞書にキー Copies が存在するかを確認します。

```
proc cas;
  hmeqTable["name"] = "hmeq";
  hmeqTable["copies"] = 2;
run;

  print "Key exists: " exists(hmeqTable, "copies");
run;
quit;
```

戻り値 1 は、キー Copies が hmeqTable 辞書に存在することを示します。

```
Key exists: 1
```

EXIT 関数

指定されたステータスで CASL コードのブロックを終了します。

カテゴリ: コントロール

構文

EXIT;

EXIT ({**severity** = *severity-code*, **reason** = *reason-code*, **statuscode** = *status-number*, **formatted** = “*message*” });

引数なし

引数が指定されていない場合、すべての引数はゼロまたは NULL であると見なされます。引数なしで EXIT 関数を使用する場合、ゼロまたは NULL 値はリスト内の位置の値に基づきます。したがって、ステータスは第 1 引数、重要度は第 2 引数、メッセージは第 3 引数です。

注: 重要度 0 とともに、ゼロ以外の正のステータスコードを指定できます。

必須引数

message
文字列に変換され、メッセージファイルで使用されるステータスコード。

reason-code
ステータスコードに関する一般的な情報を提供します。受け入れ可能な理由コードのリストについては、"理由コード" (*SAS Cloud Analytic Services: CASL プログラマガイド*)を参照してください。

status-number
サーバーからのステータスを表す番号。CODETOSTATUS 関数を使用してこの番号を検索し、詳細情報を取得できます。

severity-code
コードの実行が成功したか失敗したかを示します。受け入れ可能な重要度コードのリストについては、"重要度コード" (*SAS Cloud Analytic Services: CASL プログラマガイド*)を参照してください。

関連項目

"CODETOSTATUS 関数"

FINDTABLE 関数

指定された変数で、最初に表示されたテーブルを検索します。これは、アクションの結果が結果テーブルのコピーを返す場合に役立ちます。

カテゴリ:	結果テーブル
返されるデータの種類:	テーブルが見つからない場合は、ブール値 FALSE を返します。
例:	計算列のサブセット化 (89 ページ)

構文

FINDTABLE (*variable*);

必須引数

variable
テーブルを検索する変数の名前を指定します。多くのアクションは、結果テーブルを含む結果を返します。この関数を使用して、変数の最初の結果テーブルにアクセスします。

例: FINDTABLE 関数の使用

次の例では、FINDTABLE 関数を使って結果 *r* を見つける方法を示しています。*r* は Fetch アクションの結果変数として定義されています。*Findcar* は、FINDTABLE および SORT 関数の結果を入れる新しい変数です。

```
proc cas;
 table.fetch result=r / table="cars" to 428;
 findcar=sort_rev(findtable(r), "Invoice)[1:25, {"Make", "Model",          "Type",
"DriveTrain" "MSRP", "Invoice"}];
 print findcar;
run;
```

アウトプット 3.15　Findcar の結果

findcar: Results

Make	Model	Type	DriveTrain	MSRP	Invoice
Porsche	911 GT2 2dr	Sports	Rear	192465	173560
Mercedes-Benz	CL600 2dr	Sedan	Rear	128420	119600
Mercedes-Benz	SL600 convertible 2dr	Sports	Rear	126670	117854
Mercedes-Benz	SL55 AMG 2dr	Sports	Rear	121770	113388
Mercedes-Benz	CL500 2dr	Sedan	Rear	94820	88324
Mercedes-Benz	SL500 convertible 2dr	Sports	Rear	90520	84325
Mercedes-Benz	S500 4dr	Sedan	All	86970	80939
Acura	NSX coupe 2dr manual S	Sports	Rear	89765	79978
Jaguar	XKR convertible 2dr	Sports	Rear	86995	79226
Audi	RS 6 4dr	Sports	Front	84600	76417
Jaguar	XKR coupe 2dr	Sports	Rear	81995	74676
Dodge	Viper SRT-10 convertible 2dr	Sports	Rear	81795	74451
Porsche	911 Carrera 4S coupe 2dr (convert)	Sports	All	84165	72206
Mercedes-Benz	G500	SUV	All	76870	71540
Cadillac	XLR convertible 2dr	Sports	Rear	76200	70546
Porsche	911 Carrera convertible 2dr (coupe)	Sports	Rear	79165	69229
Mercedes-Benz	S430 4dr	Sedan	Rear	74320	69168
Volkswagen	Phaeton W12 4dr	Sedan	Front	75000	69130
Jaguar	XJR 4dr	Sedan	Rear	74995	68306
Jaguar	XK8 convertible 2dr	Sports	Rear	74995	68306
Porsche	911 Targa coupe 2dr	Sports	Rear	76765	67128
BMW	745Li 4dr	Sedan	Rear	73195	66830
Land Rover	Range Rover HSE	SUV	All	72250	65807
Audi	A8 L Quattro 4dr	Sedan	All	69190	64740
Jaguar	XK8 coupe 2dr	Sports	Rear	69995	63756

FMTVAR 関数

SAS 出力形式の代替として使用できる CASL 出力形式変数を作成します。

カテゴリ: フォーマット

構文

FMTVAR(*format-name*, <*format-width*>, <*d*>);

必須引数

format-name
出力形式の名前を含む文字列を指定します。幅、小数指定、小数点は含まれません。この出力形式は、SAS 出力形式または事前に定義されたユーザー定義出力形式です。有効な出力形式名の例は、BEST、COMMA、DAY、$HEX、DATETIME などです。

オプション引数

format-width
出力形式幅を指定します。ほとんどの出力形式では、出力データのカラム数です。

d
オプションの 10 進スケール因子を数値形式で指定します。出力形式で値を指定すると、小数点以下の桁数がその数だけ表示されます。

詳細

戻り値

FMTVAR 関数で返される可能性のあるデータ値は次のとおりです。

出力形式	FMTVAR 関数が成功すると、関数は変数に割り当て可能な出力形式値を返します。この変数は、定数出力形式指定のかわりに使用できます。
0	FMTVAR 関数が出力形式の検索に失敗しました。

例: 出力形式変数の作成

```
proc cas;
  x=62427229.72748;
```

```
  fv=fmtvar("comma",14,2);
run;
print put(x,fv);
run;
```

例のコード 3.9　ログ

```
62,427,229.73
```

GETCOLUMN 関数

列を配列に抽出します。

カテゴリ: 結果テーブル

返されるデータの種類: 配列

構文

GETCOLUMN(*table-name*, *column-name* | *column-number*);

必須引数

table-name
　列を抽出するテーブルの名前を指定します。

column-name
　配列に抽出する列の名前を指定します。

column-number
　配列に抽出する列の番号を指定します。

詳細

GETCOLUMN 関数は、列からすべての行を配列に抽出します。列は、列名または列番号で指定できます。

注: 無効な列名または無効な列番号を指定すると、GETCOLUMN 関数は空の配列を返します。

注: STRING 列で GETCOLUMN 関数を使用すると、配列のすべての値を同じ長さにするために末尾の空白が追加されます。VARCHAR 列で GETCOLUMN 関数を使用すると、配列の値は null で終了し、末尾の空白は追加されません。

例: テーブルから列を抽出

この例は、テーブルから配列に列を抽出する方法を示しています。

```
proc casutil;
  load data=sashelp.cars;
quit;

proc cas;
  table.fetch result=r / table="cars" to=428;
  findcar=findtable(r);

describe findcar;

  colarray=getcolumn(findcar, "Make");
  print colarray;
run;
quit;
```

例のコード 3.10 (DESCRIBE ステートメント)

```
[Fetch] Table ( [50] Rows [16] columns
Column Names:
[1] _Index_         [              ] (int64)
[2] Make            [              ] (char)
[3] Model           [              ] (char)
[4] Type            [              ] (char)
[5] Origin          [              ] (char)
[6] DriveTrain      [              ] (char)
[7] MSRP            [              ] (double) [DOLLAR8.]
[8] Invoice         [              ] (double) [DOLLAR8.]
[9] EngineSize      [Engine Size (L) ] (double)
[10] Cylinders      [              ] (double)
[11] Horsepower     [              ] (double)
[12] MPG_City       [MPG (City)     ] (double)
[13] MPG_Highway    [MPG (Highway)   ] (double)
[14] Weight         [Weight (LBS)    ] (double)
[15] Wheelbase      [Wheelbase (IN)  ] (double)
[16] Length         [Length (IN)     ] (double)
```

例のコード 3.11 (PRINT ステートメント)

```
{Acura   ,Acura   ,Acura   ,Audi    ,Audi    ,Audi    ,Audi    ,Audi    ,Audi    ,BMW     ,BMW     ,BMW     ,BMW
 ,
BMW     ,BMW     ,BMW     ,Buick   ,Buick   ,Buick   ,Cadillac ,Cadillac ,Chevrolet,Chevrolet,Chevrolet,Chevrole
t,Chevrolet,
Chevrolet,Chevrolet,Chevrolet,Chevrolet,Chrysler ,Chrysler ,Chrysler ,Chrysler ,Chrysler ,Dodge   ,Dodge   ,D
odge   ,Dodge   ,
Dodge   ,Ford    ,Ford    ,Ford    ,Ford    ,Ford    ,Ford    ,Ford    ,GMC     ,GMC     ,GMC     }
```

GETENV 関数

指定された文字列の環境変数を返します

カテゴリ: コントロール
ステータス

返されるデータの種類: 関数が成功した場合、返されるデータの種類は環境変数の値を持つ文字列です。それ以外の場合、戻り値は失敗を示す 0 です。

構文

GETENV(<*string*>);

オプション引数

string
1 つ以上の連続する英数字またはその他のキーボード文字、あるいはその両方。

GETKEYS 関数

辞書からキーを配列に抽出します。

カテゴリ: 辞書

返されるデータの種類: 配列

構文

GETKEYS(*dictionary*);

必須引数

dictionary
検索する辞書を指定します。

例: 辞書キーを配列に抽出

この例は、辞書から配列にキーを抽出する方法を示しています。

```
proc cas;
  studentID={Woods=13445, Shafer=36772, McKenna=37854};
  keyArray=getkeys(studentID);
  print keyArray;
run;
```

例のコード 3.12　ログ

```
{Woods, Shafer, McKenna}
```

GETRESULT 関数

ID またはタグに関連付けられた結果を取得します。

カテゴリ:　結果テーブル

構文

GETRESULT("*session-reference*", <*action-tag*><*id*>);

必須引数

session-reference
結果を取得するセッション名を指定します。

オプション引数

action-tag
アクションのジョブ名を指します。アクションタグは、非同期セッションで使用されます。

id
ジョブのアクションキュー内の番号を指します。LISTRESULTS アクションを使用すると、接続識別子を取得できます。詳細については、“保存結果の表示” (*SAS Viya プラットフォーム: システムプログラミングガイド*)を参照してください。

例: GETRESULT 関数の使用

```
proc cas;
  session mySession1;
  sessionid async='session1';
run;
session.listresults;
run;
result=getresult("mySession1","session1");
describe job;
```

```
print job;
run;
```

アウトプット 3.16　Session.ListResults 出力

Results from session.listresults

Id	Timestamp	Action	Responses	Size	Action Tag	Description
5	10/28/09 17	listsessions	2	1180	session1	Final
6	10/28/09 17	listsessions	2	1180	session2	Final
7	10/28/09 17	listsessions	2	1171	session3	Final
8	10/28/09 17	sessionid	2	149	session1	Final

アウトプット 3.17　ジョブ: session.listSessions の結果出力

job: Results from session.listSessions

Name	UUID	Session Properties	Authentication Type	User ID
MYSESSION1:Tue Oct 29 13:10:44 2019	010be3c4-a5af-8944-b560-28f3778a0241	Connected	OAuth/External PAM	sasdemo
MYSESSION1:Tue Oct 29 13:20:42 2019	0165286d-af96-cc4a-bdb4-63dd9ee66115	Connected	OAuth/External PAM	sasdemo

---

例のコード 3.13　ログ(DESCRIBE ステートメント)

```
dictionary ( 8 entries, 8 used);
 [session] string;
 [job] string;
 [status] dictionary ( 4 entries, 4 used);
  [severity] int64_t;
  [reason] int64_t;
  [status] nil;
  [statusCode] int64_t;
 [result] dictionary ( 1 entries, 1 used);
  [Session] Table[Listsessions] ( [2] Rows [5] columns
Column Names:
  [1] SessionName     [Name          ] (varchar)
  [2] UUID            [UUID          ] (char)
  [3] State           [Session Properties] (varchar)
  [4] Authentication  [Authentication Type] (varchar)
  [5] Userid          [User ID       ] (varchar)
 [logs] array ( 5 entries, 5 used);
  [ 1] nil;
  [ 2] nil;
  [ 3] nil;
  [ 4] nil;
  [ 5] nil;
 [loglevels] array ( 5 entries, 5 used);
  [ 1] nil;
  [ 2] nil;
  [ 3] nil;
  [ 4] nil;
  [ 5] nil;
 [timedout] nil;
 [canceled] nil;
```

例のコード 3.14　ログ(PRINT ステートメント)

```
{session=mySession1,job=session1,status={severity=0,reason=0,,statusCode=0},,logs={,,,,},loglevels={,,,,},,}
```

GETVALUES 関数

辞書から値を配列に抽出します。

カテゴリ:　辞書

返されるデータの種類:　配列

構文

GETVALUES(*dictionary*);

必須引数

dictionary
　検索する辞書を指定します。

例: 辞書の値を配列に抽出

この例は、辞書から配列に値を抽出する方法を示しています。

```
proc cas;
  studentID={Woods=13445, Shafer=36772, McKenna=37854};
  valArray=getvalues(studentID);
  print valArray;
run;
```

例のコード 3.15　ログ

```
{13445,36772,37854}
```

INPUT_FORMAT 関数

文字列を最適な数値に変換します。

カテゴリ:　フォーマット

返されるデータの種類: 返される可能性のあるデータの種類のリストについては、後述の詳細セクションを参照してください。

構文

INPUT_FORMAT (*string*);

必須引数

string
1 つ以上の連続する英数字またはその他のキーボード文字、あるいはその両方。

詳細

戻り値

INPUT_FORMAT 関数で返される可能性のあるデータ値は次のとおりです。

0 間違った引数が指定されました。

-1 結果が無効です。

-2 オーバーフロー

-3 値が指定されていません。

-4 変換にエラーがあります。

-5 指定された値は文字列ではありません。

例: 文字列の変換

```
proc cas;
  x=input_format("12345678901234567890123456789012345678901234567890");
  print x;
run;
```

例のコード 3.16 ログ

```
1.2345679E49
```

isType 関数

expression が指定された型の値になるかに基づいて、TRUE または FALSE を返します。

カテゴリ: 変数情報

返されるデータの種類: ブール(TRUE または FALSE)

参照項目: *“ブール” (SAS Cloud Analytic Services: CASL プログラマガイド)*

構文

is*Type* (*value-1*, <*value-1*><,...><, *value-n*>);

必須引数

Type
関数型を指定します。

Supported function types:

- Array
- Blob
- Double
- Dictionary
- Integer
- List
- String
- Table
- Varbinary

value
指定されたサポートされている関数型と一致する、または一致しない値になる*式*を指定します。式には、リテラル値、関数、またはこれらを組み合わせて結果の値を生成する一連のオペランドと演算子を指定できます。

注: *value* には、1 つ以上の式を定義できます。複数の引数がある場合、関数が TRUE を返すためには、すべての引数は指定された関数型に評価される必要があります。

参照項目: *“CASL 式の定義” (SAS Cloud Analytic Services: CASL プログラマガイド)*

“式の例” (SAS Cloud Analytic Services: CASL プログラマガイド)

“式の演算子” (*SAS Cloud Analytic Services: CASL プログラマガイド*)

“CASL データ型” (*SAS Cloud Analytic Services: CASL プログラマガイド*)

例

例 1: isDouble 関数: Double の場合

次の例では、isType 関数を使用して、(x に割り当てられた)リテラルが double かを判断します。

```
proc cas;
 x=21/3;
   if isDouble(x) then print x "is a double.";
     else print x "is not a double.";
run;
```

例のコード 3.17　SAS ログ

```
7 is a double.
```

例 2: isType 関数でさまざまな種類の式を指定する

次の例では、値が、どのように、リテラル、関数、または演算子を含む式として表現されるのかが示されています。これらの例では、次の点に注意してください。

- CAT 関数は文字列を返します。
- DIM 関数は整数を返します。
- EXISTS 関数は整数を返します。

```
proc cas;
  num=isInteger( max(10, 4) );
  print 'The result of the isInteger function is ' num;
run;

proc cas;
  x="Hello"; y="World!";
  x=isString(cat(x||y));
  print 'The result of the isString function is ' x;
run;

proc cas;
  my_array = 1:10;
  x=isInteger( DIM(my_array), exists("my_array"), 5+5 );
  print 'The result of the isInteger function is ' x;
run;

proc cas;
  x=5;
  y=( x IN (1:10) );
  string=isString( (x IN (1:10)) ) ;
  print 'The result of the isString function is ' string;
```

```
run;
quit;
```

```
82   proc cas;
83      num=isInteger( max(10, 4) );
84      print 'The result of the isInteger function is ' num;
85   run;
The result of the isInteger function is TRUE
86
NOTE: PROCEDURE CAS used (Total process time):
      real time           0.01 seconds
      cpu time            0.02 seconds

87   proc cas;
89      x="Hello"; y="World!";
90      x=isString(cat(x||y));
91      print 'The result of the isString function is ' x;
92   run;
The result of the isString function is TRUE
93
NOTE: PROCEDURE CAS used (Total process time):
      real time           0.10 seconds
      cpu time            0.19 seconds

94   proc cas;
95      my_array = 1:10;
96      x=isInteger( DIM(my_array), exists("my_array"), 5+5 );
97      print 'The result of the isInteger function is ' x;
98   run;
The result of the isInteger function is TRUE
99
NOTE: PROCEDURE CAS used (Total process time):
      real time           0.01 seconds
      cpu time            0.03 seconds

100  proc cas;
101     x=5;
102     y=( x IN (1:10) );
103     string=isString( (x IN (1:10)) ) ;
104     print 'The result of the isString function is ' string;
105  run;
The result of the isString function is FALSE
106  quit;
```

JSON2CASL 関数

JSON 形式のテキストを含む文字列を CASL 辞書に変換します。

カテゴリ: 辞書

操作: “相互作用” (147 ページ)を参照してください。

構文

JSON2CASL(*string*)

必須引数

string
JSON 形式のテキストの文字列。

詳細

戻り値

JSON2CASL 関数で返される可能性のあるデータ値は次のとおりです。

dictionary 変換された JSON を含む辞書。

FALSE JSON 文字列の辞書への変換に失敗しました。

相互作用

CASL_STD_JSON 環境変数と STDJSON オプションの設定は、JSON2CASL 関数に渡される文字列引数が JSON 標準に関して厳密に解釈されるかどうかに影響します。

デフォルトでは、既存のコードの下位互換性のために、JSON2CASL 関数への入力に対する JSON 標準への厳密な準拠は期待されておらず、強制もされていません。JSON2CASL 関数への入力として使用するために、CASL2JSON 関数によって生成された非準拠テキストに対しては、特別に許容されます。この非準拠の動作は、次の方法で管理できます。

- STDJSON オプションを CASL セッションで設定します。これは、SAS Compute Server (SAS クライアント)または CAS サーバーのどちらで実行されるかに関係なく、CASL プログラムが実行されている環境にのみ影響するローカル設定です。
  - SAS クライアントで STDJSON オプションを設定するには、PROC CAS コードブロックで set stdjson または unset stdjson を指定します。

    ```
    proc cas;
      set stdjson;
    run; quit;
    ```

    SAS クライアント上で実行されているプログラムで STDJSON 引数を設定すると、SAS クライアント上の関数の動作にのみ影響します。
  - CAS サーバーで STDJSON オプションを設定するには、sccasl.runCasl アクションにサブミットされるコードに set stdjson または unset stdjson が含まれていることを確認してください。

    CAS サーバー上で実行されているプログラムで STDJSON 引数を設定すると、CAS サーバー上の関数の動作にのみ影響します。
- CASL_STD_JSON 環境変数を設定します。環境変数は、SAS Compute Server (SAS クライアント)または CAS サーバーで設定できます。これらの環境のいずれかで環境変数を設定すると、グローバルな効果があり、各サーバーで実行するためにサブミットされたすべての CASL プログラムに適用されます。
  - SAS Compute Server で CASL_STD_JSON 環境変数を設定するには、SAS Environment Manager の計算サービスの sas.compute.server: startup_commands 構成インスタンスに環境変数を追加します。

```
# to enable the JSON standard
    export CASL_STD_JSON = true
# to disable the JSON standard
    export CASL_STD_JSON = false
```

SAS Compute Server 構成インスタンス(sas.compute.server: startup_commands)での環境変数の設定は、SAS クライアントで実行されている CASL コードにのみ影響します。

- CAS サーバーで CASL_STD_JSON 環境変数を設定するには、SAS Environment Manager の cas-shared-default サービスの sas.cas.instance.config: settings 構成インスタンスに環境変数を追加します。

```
# to enable the JSON standard
    export CASL_STD_JSON = true
# to disable the JSON standard
    export CASL_STD_JSON = false
```

CAS サーバー構成インスタンス(sas.cas.instance.config: settings)での環境変数の設定は、CAS サーバーで実行されている CASL コードにのみ影響します。

構成インスタンスでの環境変数の設定の詳細については、"Edit Server Configuration Instances" (*SAS Environment Manager: User's Guide*)を参照してください。

JSON の詳細については、RFC 8259 を参照してください。

CASL2JSON 関数と JSON2CASL 関数を同じプログラムで実行し、一方の関数の出力をもう一方の関数への入力として使用する場合は、STDJSON オプションを一貫して設定する必要があります。

- CASL2JSON 関数の実行時に STDJSON オプションが*設定されていない*場合は、JSON2CASL 関数の実行時にも設定しないでください。
- JSON2CASL 関数の実行時に STDJSON オプションが設定されて*いる*場合は、CASL2JSON 関数の実行時にも設定する必要があります。

例: JSON ファイルを CASL 辞書に変換する

次の例では、JSON を CASL 辞書に変換します。JSON ファイルは PROC JSON によって作成され、ファイル myjson.json に保存されます。READPATH 関数は、ファイルのコンテンツを文字列として変数 **Json1** に保存します。JSON2CASL 関数は、**Json1** 変数の JSON 文字列を CASL 辞書に変換します。

```
proc json out="/tmp/myjson.json";          /* 1 */
  export sashelp.class (where=(age=12));
run;

proc cas;
  json1=readpath("/tmp/myjson.json");      /* 2 */
  mydict=json2casl(json1);                 /* 3 */
  print mydict;                            /* 4 */
  describe mydict;                         /* 5 */
run;
```

1 PROC JSON を使用して、JSON ファイル Myjson.json を作成します。

2 READPATH 関数は、Myjson.json のコンテンツを文字列として変数 json1 に保存します。

3 JSON2CASL 関数は、Json1 変数の JSON 文字列を CASL 辞書に変換します。辞書は変数 MyDict に保存されます。

4 PRINT ステートメントで、CASL 辞書が SAS ログに出力されます。

5 DESCRIBE ステートメントで、変数 MyDict の構造が SAS ログに出力されます。

例のコード 3.18　CASL 辞書に変換された JSON

```
{SASJSONExport=1.0,SASTableData+CLASS={{Name=James,Sex=M,Age=12,Height=57.3,Weight=83},
{Name=Jane,Sex=F,Age=12,Height=59.8,
Weight=84.5},{Name=John,Sex=M,Age=12,Height=59,Weight=99.5},
{Name=Louise,Sex=F,Age=12,Height=56.3,Weight=77},{Name=Robert,Sex=M,
Age=12,Height=64.8,Weight=128}}}
```

関連項目

- “CASL2JSON 関数” (107 ページ)
- “READPATH 関数” (155 ページ)
- “PRINT ステートメント” (57 ページ)
- “JSON プロシジャ” (*Base SAS プロシジャガイド*)

LIST_PARALLEL_SESSIONS 関数

並列セッションをリストします。

カテゴリ: サーバー側

構文

LIST_PARALLEL_SESSIONS< ("*session-reference*")>;

オプション引数

session-reference
リストするセッション名を指定します。

LOC 関数

指定された列で指定された値が見つかった行番号を返します。

カテゴリ: 結果テーブル

返されるデータの種類: 値が見つからない場合は-1 の値を返し、それ以外の場合は行番号を返します。

構文

LOC (*table-name*, *column*, *value*);

必須引数

table-name
行値の取得元のテーブルの名前。

column
列は数値または文字値にすることができます。

value
テーブルで検索する値。値は文字または数値にすることができます。

例: LOC 関数の使用

次の例は、LOC 関数を使用して、指定した値 *75000* の行番号を取得する方法を示しています。

```
proc cas;
  table.fetch result=r/ table="cars" to=428
   findcar=sort_rev(findtable(r), "Invoice")[1:25,    {"Make", "Model", "Type", "DriveTrain"
"MSRP", "Invoice"}];
  y=loc(findcar, "MSRP", 75000);
   print "The row in which the value appears is= " y;
run;
```

例のコード 3.19　SAS ログ

```
The row in which the value appears is= 18
```

MEMORYSTATS 関数

CPU とメモリの統計情報を表示します。

カテゴリ: ステータス

構文

MEMORYSTATS ();

引数なし

この関数に引数はありません。

例: MEMORYSTATS 関数の使用

```
proc cas;
  memorystats();
print memorystats();
run;
```

例のコード 3.20　SAS ログ

```
{user_cpu_time=9.68,system_cpu_time=2.64,timestamp=1840474683,current_memory=52015104,high_water
_mark_memory=52015104,current_thread_count=12}
```

NEWTABLE 関数

新しいテーブルを作成します。

カテゴリ:　結果テーブル

構文

NEWTABLE(“*table-name*”, *column-name*, *data-type*, *row-1*<, *row-2*, ... *row-N*>);

必須引数

“*table-name*”
テーブルの名前を指定します。

column-name
新しいテーブルの列名のリストを指定します。*column-name* は、結果が値のリストになる任意の式にすることができます。

注: リストは配列として扱われ、リスト内の値は配列インデックスの昇順でテーブルに割り当てられます。

data-type
列値のデータ型を定義する値のリストを指定します。*data-type* は、結果が値のリストになる任意の式にすることができます。

注: リストは配列として扱われ、リスト内の値は配列インデックスの昇順でテーブル行の列に割り当てられます。

参照項目: サポートされているデータ型については、CASL でサポートされるデータ型を参照してください。

オプション引数

row
テーブルに行として追加する列値のリストを 1 つ以上指定します。各行引数は、結果が値のリストになる任意の式にすることができます。

注: リストは配列として扱われ、リスト内の値は配列インデックスの昇順でテーブル行の列に割り当てられます。

例

この例では、3 つの行と 3 つの列を含む新しいテーブル Average Class Grades を作成します。列には int64 型のデータが含まれます。

```
proc cas;
title "Average Class Grades";
 colNames={"Class A","Class B","Class C"}; /* 1 */
 colTypes={"int64","int64","int64"};      /* 2 */
 row1={71 74 84};                         /* 3 */
 row2={65 74 65};
 row3={80 80 90};
 myTable=newtable("myTable",colNames,colTypes,row1,row2,row3); /* 4 */
 print myTable;
 describe myTable;                        /* 5 */
run;
quit;
```

1 列名のリストを指定します。この例では、名前は配列として指定されています。

2 各列のデータ型のリストを指定します。型は整数の配列として指定されます。

3 テーブルに 3 つの行を指定します。行の値は配列として指定されます。

4 NEWTABLE 関数を指定してテーブルを作成します。

5 describe ステートメントを指定して、設定どおりの各列とそのデータ型を含む新しいテーブルをリストします。

アウトプット 3.18　NEWTABLE 関数の出力: Average Class Grades の出力

Average Class Grades

myTable: Results

Class A	Class B	Class C
71	74	84
65	74	65
80	80	90

例のコード 3.21　SAS ログ: DESCRIBE ステートメント

```
[myTable] Table ( [3] Rows [3] columns
Column Names:
[1] Class A       [Class A      ] (int64)
[2] Class B       [Class B      ] (int64)
[3] Class C       [Class C      ] (int64)
```

関連項目

- “CASL データ型” (*SAS Cloud Analytic Services: CASL プログラマガイド*)
- “DESCRIBE ステートメント”

PUT 関数

指定された出力形式で値をフォーマットし、それを文字列として返します。

カテゴリ:　フォーマット

構文

PUT (*value*, *format*);

必須引数

value
　フォーマットされる数値を指定します。

format
出力形式を指定します。

例: PUT 関数の使用

PRINT ステートメントで PUT 関数を使用して、値をフォーマットできます。

```
proc cas;
  print put(10, hex8.);
run;
```

例のコード 3.22　SAS ログ

```
0000000A
```

関連項目

“PRINT ステートメント”

READFILE 関数

指定されたファイル参照名のコンテンツを読み取って返します。

該当要素:	クライアント側のみ
カテゴリ:	ファイル IO
返されるデータの種類:	String | Varbinary

構文

READFILE("*file-reference*");

必須引数

"*file-reference*"
コンテンツを入力しようとしているファイルのファイル参照名。

詳細

デフォルトでは、READFILE 関数はファイル内にテキストベースのデータを必要とします。SAS セッションエンコーディングでエンコードされた入力テキストファイル

は、UTF-8 エンコーディングにトランスコードされてから、文字列として出力されます。

例: READFILE 関数の使用

次の例は、ファイル *Findtable1.sas* のコンテンツを読み取る方法を示しています。

```
proc cas;
  filename findtest '/u/sasdemo/public/findtable1.sas';
  test=readfile("findtest");
  print test;
run;
```

この例の出力は *Findtable1.sas* のコンテンツで、アクションと CASL ステートメントを含むプログラムが含まれています。

READPATH 関数

指定されたファイル名のコンテンツを読み取って返します。

該当要素:	クライアント側のみ
カテゴリ:	ファイル IO
返されるデータの種類:	String | Varbinary

構文

READPATH ("*file-name*");

必須引数

"*file-name*"
　読み取るファイルの名前。

詳細

デフォルトでは、READPATH 関数はファイル内にテキストベースのデータを必要とします。SAS セッションエンコーディングでエンコードされた入力テキストファイルは、UTF-8 エンコーディングにトランスコードされてから、文字列として出力されます。

例: ファイルのコンテンツを変数に読み込む

```
store = readpath ("/u/sasdemo/ds/samplecode.sas");
```

RESULT_BY_COL 関数

指定された列で新しいテーブルを作成します。

カテゴリ: 結果テーブル

返されるデータの種類: CAS テーブル

構文

RESULT_BY_COL (*table-name*, "*column-name*", *column_number*);

必須引数

table-name
新しいテーブルの名前を指定します。

"*column-name*," or *column_number*
列は名前または番号のいずれかで参照されます。新しいテーブルの列の順序は、呼び出しで指定された順序と同じです。

RESULT_BY_TYPE 関数

型の指定に一致する列を持つ新しいテーブルを作成します。

カテゴリ: 結果テーブル

構文

RESULT_BY_TYPE (*table-name*, "<*data types*>");

必須引数

table-name
新しいテーブルの名前を指定します。

オプション引数

data types
データ型とは、列に保存されるデータの型を指定する属性です。サポートされているデータ型については、CASL でサポートされるデータ型を参照してください。

関連項目

"データ型とは" (SAS Cloud Analytic Services: CASL プログラマガイド)

SEND_RESPONSE 関数

指定された結果をクライアントに送り返します。

カテゴリ: サーバー側

構文

SEND_RESPONSE (*result*);

必須引数

result
クライアントに送り返す結果を指定します。

例: 結果をクライアントに送り返す

```
send_response(r, {test="test"});
```

SLEEP 関数

指定した秒数の間、オペレーティングシステム(OS)をスリープさせます。

カテゴリ: コントロール
ステータス

構文

SLEEP (<*value*>);

オプション引数

value
OS がスリープする秒数を指定します。

SORT 関数

昇順で並べ替えたリストを返します。

カテゴリ: 配列

デフォルト: 昇順

構文

SORT (*list*);

必須引数

list
カウント可能な数の値を指定します。

例

例 1: リストを昇順で並べ替える

次の例では、数値でリストを作成し、SORT 関数を使用してリストを昇順で返します。

```
proc cas;
  list={98, 74, 54, 18, 101, 67, 80, 90, 62};   /* #1 */
  alist= sort(list);                             /* #2 */
print alist;                                     /* #3 */
run;
```

1 数値でリストを作成します。

2 SORT 関数は、昇順で並べ替えたリストを返し、そのリストを変数 *alist* に保存します。

3 *alist* 変数を出力します。詳細については、PRINT ステートメントを参照してください。

並べ替えたリストは SAS ログに出力されます。

例のコード 3.23　SAS ログ

```
{18,54,62,67,74,80,90,98,101}
```

例 2: キー付きリストを昇順で並べ替える

次の例では、数値でキー付きリストを作成し、SORT 関数を使用してキー付きリストを昇順で返します。

```
proc cas;
 session casauto;
 Mary_Jones_Average={Math_Avg=88.6            /* #1 */
  Science_Avg=90.3
  English_Avg=89.5
  Economics_Avg=82.5};
 MJ_Average=sort(Mary_Jones_Average);        /* #2 */
print MJ_Average;                            /* #3 */
run;
```

1 *Mary_Jones_Average* という名前のキー付きリストを作成します。この例では、すべてのクラスで Mary Jones の平均を求めています。

2 SORT 関数は、昇順で並べ替えたキー付きリストを返し、そのリストを変数 *MJ_Average* に保存します。

3 *MJ_Average* 変数を出力します。詳細については、PRINT ステートメントを参照してください。

並べ替えたキー付きリストは SAS ログに出力されます。

例のコード 3.24　SAS ログ

```
{Economics_Avg=82.5,Math_Avg=88.6,English_Avg=89.5,Science_Avg=90.3}
```

関連項目

“SORT REV 関数”

SORT REV 関数

降順で並べ替えたリストを返します。

カテゴリ:　配列

構文

SORT_REV (*list*);

必須引数

list
カウント可能な数の値を指定します。

例

例 1: リストを降順で並べ替える

次の例では、数値でリストを作成し、SORT 関数を使用してリストを降順で返します。

```
proc cas;
 list={98, 74, 54, 18, 101, 67, 80, 90, 62};     /* #1 */
 alist= sort_rev(list);                          /* #2 */
print alist;                                     /* #3 */
run;
```

1 数値でリストを作成します。

2 SORT_REV 関数は、降順で並べ替えたリストを返し、そのリストを変数 *alist* に保存します。

3 *alist* 変数を出力します。詳細については、PRINT ステートメントを参照してください。

並べ替えたリストは SAS ログに出力されます。

例のコード 3.25　SAS ログ

```
{101,98,90,80,74,67,62,54,18}
```

例 2: キー付きリストを降順で並べ替える

次の例では、数値でキー付きリストを作成し、SORT 関数を使用してキー付きリストを降順で返します。

```
proc cas;
 session casauto;
 Mary_Jones_Average={Math_Avg=88.6                /* #1 */
  Science_Avg=90.3
  English_Avg=89.5
  Economics_Avg=82.5};
 MJ_Average=sort_rev(Mary_Jones_Average);         /* #2 */
print MJ_Average;                                /* #3 */
run;
```

1 *Mary_Jones_Average* という名前のキー付きリストを作成します。この例では、すべてのクラスで Mary Jones の平均を求めています。

2 SORT_REV 関数は、降順で並べ替えたキー付きリストを返し、そのリストを変数 *MJ_Average* に保存します。

3 *MJ_Average* 変数を出力します。詳細については、PRINT ステートメントを参照してください。

並べ替えたキー付きリストは SAS ログに出力されます。

例のコード 3.26　SAS ログ

```
{Science_Avg=90.3,English_Avg=89.5,Math_Avg=88.6,Economics_Avg=82.5}
```

関連項目

“SORT 関数”

SYMDEL 関数

指定されたマクロ変数を削除します。

カテゴリ:　マクロ

構文

SYMDEL (“*macro-variable*”);

必須引数

macro-variable
削除するマクロ変数の名前。

関連項目

- “SYMGET 関数”
- “SYMPUT 関数”
- “SYMPUTX 関数”

SYMGET 関数

マクロ変数の値を返します。

カテゴリ:　マクロ

構文

SYMGET(“*macro-variable*”);

必須引数

macro-variable
値が返されるマクロ変数の名前。

関連項目

- “SYMDEL 関数”
- “SYMPUT 関数”
- “SYMPUTX 関数”

SYMPUT 関数

SAS マクロ変数に値を保存します。

カテゴリ:	マクロ
返されるデータの種類:	サーバー側: 返される可能性のあるデータの種類のリストについては、後述の詳細セクションを参照してください。クライアント側: 1 - 成功。これ以外のデータの種類が返された場合は失敗です。
注:	SYMPUT 関数を使用しても、SAS クライアントのマクロ変数の値は変更されません。

構文

SYMPUT (*name*, *value*);

必須引数

name
値を保存するマクロ変数の名前。

value
マクロ変数に保存する数値または文字値。

関連項目

- “SYMGET 関数”

SYMPUTX 関数

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

SYMPUTX (*macro-variable-name*, *macro-variable-value*, <*symbol-table*>);

必須引数

macro-variable-name
次のいずれかの値になります。

- SAS 変数名を表す、引用符で囲んだ文字列。
- SAS 名の値を持つ文字変数の名前。
- 有効な SAS 変数名を生成する式。この形式は、一連のマクロ変数を作成する場合に役立ちます。
- *macro-variable-name* から先頭と末尾の空白が削除されます。

macro-variable-value
文字または数値の定数、変数または式を指定します。*macro-variable-value* が数値の場合、値は BESTw.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。SAS Compute Server では、*symbol-table* の最初の空白以外の文字で、マクロ変数の保存先のシンボルテーブルを指定します。CAS サーバーでは、このパラメーターは無視されます。*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。

デフォルト F

制限事項 CAS サーバーで SYMPUTX 関数を実行する場合、SYMPUTX 関数の *symbol-table* 引数はサポートされません。パラメーターは無視されます。

F　マクロ変数がシンボルテーブルに存在する場合、SYMPUTX はマクロ変数が存在する最もローカルなシンボル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、SYMPUTX は最もローカルなシンボルテーブルに変数を保存します。これは、*symbol-table* 引数が指定されていない場合のデフォルトの動作です。

G　マクロ変数は、ローカルシンボルテーブルが存在する場合でも、グローバルシンボル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シンボルテーブルに保存されます。SYMPUTX がマクロ定義の外側で呼び出される場合、グローバルシンボルテーブルが常に使用されます。

詳細

SYMPUTX 関数について

Compute Server には SAS マクロ機能が含まれていますが、CAS サーバーには含まれていません。SYMPUTX は、Compute Server で実行されると、値から先頭と末尾の両方の空白を削除し、マクロシンボルテーブルに保存されているマクロ変数に値を割り当てます。CAS サーバーで実行された場合、SYMPUTX は先頭と末尾の両方の空白を削除し、CAS の名前付きメモリ位置に値を割り当てます。ドキュメントでは、両方の保存場所はマクロ変数と呼ばれます。

比較

SYMPUTX 関数は SYMPUT 関数と似ていますが、次の点で異なります。

表 3.4　SYMPUT と SYMPUTX の比較

SYMPUT 関数	SYMPUTX 関数
macro-variable-value が数値定数の場合、マクロ変数を文字列に変換し、SAS ログにメモを書き込む	*macro-variable-value* が数値定数の場合、BESTw.出力形式を使用してマクロ変数を文字列に変換し、SAS ログにメモは書き込まない
最大で 12 文字のフィールド幅を使用する	マクロ変数を文字データ型に変換するときに、最大 32 文字のフィールド幅を使用する
macro-variable-name の先頭に空白を含めることはできない	*macro-variable-name* の先頭に空白を含めることができる
ラテンアルファベットの文字 A...Z、a...z、またはアンダースコア(_)で始まる有効な SAS 名にする必要がある	*macro-variable-name* から先頭と末尾の空白が削除される

SYMPUT 関数	SYMPUTX 関数
macro-variable-name と *macro-variable-value* のいずれからも先頭と末尾の空白を削除しない	

SYMPUT マクロ関数と SYMPUTX マクロ関数の違いについては、“例 2: SYMPUT 関数と SYMPUTX 関数を使用して SAS Compute Server でマクロ変数を作成する”を参照してください。

例

例 1: マクロ変数に値を割り当て、先頭と末尾の空白を削除する

次の例は、SYMPUTX 関数を CAS サーバーで実行し、変数に値を割り当てる方法を示しています。この関数は、*macro-variable-name* と *macro-variable-value* の両方から先頭と末尾の空白を削除します。

注: CAS サーバーは SAS マクロ処理をサポートしていません。SYMPUTX 関数が CAS サーバーで実行されると、指定された値から先頭と末尾の空白が削除され、CAS 内の名前付きメモリ位置に値が割り当てられます。CAS にはマクロ機能がないため、マクロシンボルテーブルはありません。ドキュメントでは、両方の保存場所は *macro-variable* と呼ばれます。

```
cas casauto;                        /* 1 */

proc cas;                          /* 2 */
  source myProgram;                    /* 3 */
    x = symputx("    mvar1   ","    removes leading/trailing blanks   "); /* 4 */
    y = symputx("   mvar2    ",    123.456    ); /* 5 */
    A = symget("mvar1");                    /* 6 */
    B = symget("   mvar2   ");
    print "SYMPUTX " A;                     /* 7 */
    print "Numeric constant " B " is converted to string and blanks removed.";
  endsource;
run;

  runcasl code=myProgram;              /* 8 */
run;
```

1 CAS セッションをまだ開始していない場合は、CAS ステートメントを指定して CAS セッションを開始します。

2 PROC CAS ステートメントを指定して、CASL プログラムを定義します。PROC CAS では、CAS サーバーで実行される CAS アクションを指定することもできます。

3 SOURCE ステートメントを指定して CASL プログラムを作成し、そのプログラムを変数 *myProgram* に保存します。

4 SYMPUTX 関数を指定して、*macro-variable-name* の *mvar1* (名前付きメモリ位置)の値を変数 x に保存します。

5 SYMPUTX 関数を指定して、*macro-variable-name* の *mvar2* (名前付きメモリ位置)の値を変数 y に保存します。*mvar2* 変数値は、CAS が文字列に変換する数値定数です。

6 SYMGET 関数を指定して、*A* と *B* の値を保存します。

7 PRINT ステートメントを指定して、変数 *A* および *B* の値をログに書き込みます。

8 runCasl アクションを指定してプログラムを実行し、結果をログに出力します。

結果は、先頭と末尾の空白が両方のマクロ値から削除されていることを示しています。

```
NOTE: Active Session now CASAUTO.
SYMPUTX removes leading/trailing blanks
Numeric constant 123.456 is converted to string and blanks removed.
```

例 2: SYMPUT 関数と SYMPUTX 関数を使用して SAS Compute Server でマクロ変数を作成する

次の例は、SAS Compute Server で SYMPUTX 関数を実行したときに先頭と末尾の両方の空白を削除する方法を示し、この動作を SYMPUT 関数と比較しています。SYMPUT 関数は、*macro-variable-name* と *macro-variable-value* のいずれからも先頭と末尾の空白を削除しません。

注: SAS クライアント(SAS Compute Server)は、完全なマクロ処理をサポートしています。SAS Compute Server で実行されると、SYMPUTX 関数はマクロ変数を作成し、関数で指定した値を割り当て、値からすべての 先頭と末尾の空白を削除します。次に、値が *マクロシンボルテーブル*のマクロ変数に割り当てられます。

```
proc cas;                          /* 1 */
  symputx('   items1   ', '    removes leading/trailing blanks   ');   /* 2 */
  symput('items2',   '  does NOT remove leading/trailing blanks    '); /* 3 */
run;

  print "SYMPUTX " symget("items1");   /* 4 */
  print "SYMPUT "  symget("items2");
run;
```

1 PROC CAS ステートメントを指定して、CASL プログラムを定義します。PROC CAS では、CAS サーバーで実行される CAS アクションを指定することもできます。

2 SYMPUTX 関数を指定して、マクロ変数 items1 を作成し、その値を保存します。

3 SYMPUT 関数を指定して、マクロ変数 items2 を作成し、その値を保存します。

4 PRINT ステートメントを指定して、変数の値をログに書き込みます。結果は、SYMPUTX 関数が先頭と末尾の空白を削除するが、SYMPUT 関数は削除しないことを示しています。

```
SYMPUTX removes leading/trailing blanks
SYMPUT  does NOT remove leading/trailing blanks
```

関連項目

- “SYMGET 関数”
- “SYMDEL 関数”
- “SYMPUT 関数”

TABCOLUMNS 関数

結果テーブルから列を取得します。

カテゴリ: 結果テーブル

注: TABCOLUMNS 関数は、結果テーブルに対してのみ機能します。結果テーブル名が使用されなかった場合、この関数は SAS ログにエラーを生成します。

構文

TABCOLUMNS (*table-name*);

必須引数

table-name
テーブルの名前を指定します。

例: TABCOLUMNS 関数の使用

次の例は、TABCOLUMN 関数を使用して結果テーブルから列を取得する方法を示しています。結果テーブルが必要なため、table.Fetch アクションが実行され、結果が変数 *r* に格納されます。SORT_REV 関数と FINDTABLE 関数の結果を入れるために、新しい変数 *findcar* が作成されます。TABCOLUMNS 関数は、*findcar* 結果テーブルから列を取得し、*carscol* という名前の新しい変数に列を入れます。TABCOLUMN 関数の結果をリストするには、PRINT ステートメントを使用します。

```
proc cas;
  table.fetch result=r / table="cars" to=428;
   findcar=sort_rev(findtable(r), "Invoice");
  carscol=tabcolumns(findcar);
  print carscol;
run;
```

findcar 結果テーブルには 16 の列がありました。16 列すべてが、ラベルとともに SAS ログにリストされます。

例のコード 3.27　SAS ログ

```
{_Index_=_Index_,Make=Make,Model=Model,Type=Type,Origin=Origin,DriveTrain=DriveTrain,MSRP=MSRP,Inv
oice=Invoice,EngineSize=

EngineSize,Cylinders=Cylinders,Horsepower=Horsepower,MPG_City=MPG_City,MPG_Highway=MPG_Highway
,Weight=Weight,Wheelbase=Wheelbase,Len
 gth=
 Length}
```

関連項目

- “FINDTABLE 関数”
- “SORT REV 関数”
- “PRINT ステートメント”

TABTYPES 関数

結果テーブルから辞書にデータ型を抽出します。

カテゴリ:　結果テーブル

注:　TABTYPES 関数は、結果テーブルに対してのみ機能します。結果テーブル名が使用されなかった場合、この関数は SAS ログにエラーを生成します。

構文

TABTYPES (*table-name*);

必須引数

table-name
　テーブルの名前を指定します。

例: TABTYPES 関数の使用

次の例は、TABTYPES 関数を使用して結果テーブルからデータ型を取得する方法を示しています。結果テーブルが必要なため、table.Fetch アクションが実行され、結果が変数 *r* に格納されます。SORT_REV 関数と FINDTABLE 関数の結果を入れるために、新しい変数 *findcar* が作成されます。TABTYPES 関数は、*findcar* 結果テーブ

ルから各列のデータ型を取得し、*carscol* という名前の新しい変数に列とそのデータ型を入れます。TABTYPES 関数の結果をリストするには、DESCRIBE ステートメントを使用します。

```
proc cas;
  table.fetch result=r / table="cars" to=428;
   findcar=sort_rev(findtable(r), "Invoice");
  carscol=tabtypes(findcar);
  describe carscol;
run;
```

findcar 結果テーブルには 16 の列がありました。16 列すべてが、データ型とともに SAS ログにリストされます。

例のコード 3.28　SAS ログ

```
dictionary ( 16 entries, 16 used);
 [_Index_] string;
 [Make] string;
 [Model] string;
 [Type] string;
 [Origin] string;
 [DriveTrain] string;
 [MSRP] string;
 [Invoice] string;
 [EngineSize] string;
 [Cylinders] string;
 [Horsepower] string;
 [MPG_City] string;
 [MPG_Highway] string;
 [Weight] string;
 [Wheelbase] string;
 [Length] string;
```

関連項目

- “CASL データ型” (*SAS Cloud Analytic Services: CASL プログラマガイド*)
- “DESCRIBE ステートメント”
- “FINDTABLE 関数”
- “SORT REV 関数”

TERM PARALLEL SESSION 関数

並列セッションを終了します。

カテゴリ:	サーバー側
返されるデータの種類:	関数の終了に失敗したセッションの数

構文

TERM_PARALLEL_SESSION (<*session-reference-1*>, <*session-reference-N*>);

オプション引数

session-reference
キャンセルするセッション名を指定します。

> ヒント　複数のセッション名をリストできます。

例: 並列セッションの終了

```
term_parallel_session(casauto);
```

TIMESTAMP 関数

現在の日付と時間を返します。

カテゴリ:	コントロール ステータス
デフォルト:	**string**
返されるデータの種類:	日付と時間

構文

TIMESTAMP <("*string* ")> <("*int64* ")> <("DATETIME*w.d*")> <("NLDATMTM*w.*")> <("NLDATM*w.*")> <("MDYAMPM*w.d*")>;

オプション引数

string
1 つ以上の連続する英数字またはその他のキーボード文字、あるいはその両方。

int64
19 桁の精度で 64 ビット符号付きの正確な整数。

DATETIME*w.d*
ddmmmyy:*hh*:*mm*:*ss.ss* の形式で日時値を書き込みます。

w
出力フィールドの幅を指定します。

デフォルト 16

範囲 7-40

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

d
秒値の小数点以下の桁数を指定します。この引数はオプションです。

範囲 0-39

要件 *d* は *w* より小さい必要があります。

配置 右

制限事項 *w-d*< 17 の場合、SAS は小数値を切り捨てます。

NLDATMTM*w*.
SAS 日時値の時間部分を指定ロケールの時刻値に変換してから、その値を時刻として書き込みます。

w
出力フィールドの幅を指定します。

デフォルト 16

範囲 16-200

配置 左

NLDATM*w*.
SAS 日時値を指定ロケールの日時値に変換してから、その値を日時として書き込みます。

w
出力フィールドの幅を指定します。必要に応じて、SAS は出力形式幅に合うように日時値を短縮します。

デフォルト 30

範囲 10-200

配置 左

MDYAMPM*w.d*
mm/*dd*/*yy*<*yy*>*hh*:*mm* AM|PM の形式で日時値を書き込みます。年は 2 桁または 4 桁のいずれかです。

w
出力フィールドの幅を指定します。

デフォルト 19

範囲 8-40

d

分値の小数点以下の桁数を指定します。この引数はオプションです。

デフォルト 0

範囲 0-39

配置 右

注 デフォルトの期間は AM です。

例

例 1: TIMESTAMP 関数の使用

引数なしで TIMESTAMP 関数を使用すると、現在の日付と時間を取得できます。

```
print timestamp();
```

次のように SAS ログに出力されます。

```
Wed May 23 23:58:54 2018
```

TIMESTAMP 関数を **int64** 引数とともに使用すると、現在の日付と時間を **int64** データ型として取得できます。

```
print timestamp("int64");
```

次のように SAS ログに出力されます。

```
1842739241
```

TIMESTAMP 関数を **string** 引数とともに使用すると、現在の日付と時間を **string** データ型として取得できます。

```
print timestamp("string");
```

次のように SAS ログに出力されます。

```
Thu May 24 00:01:06 2018
```

例 2: 現在の日付と時間のフォーマット

TIMESTAMP 関数の結果をフォーマットできます。現在の日付と時間は 2019 年 9 月 19 日午後 4:13 です。

```
proc cas;
  print timestamp(datetime14.2);
  print timestamp(MDYAMPM20.);
```

```
  print timestamp(NLDATMTM16.);
  print timestamp(NLDATM30.);
run;
```

次のように SAS ログに出力されます。

```
20SEP19:16:13
9/20/2019  4:13 PM
16:13:18
20Sep2019:16:13:18
```

関連項目

リファレンス

- “使用頻度の高いデータ型” (*SAS Cloud Analytic Services: CASL プログラマガイド*)

ステートメント

- “PRINT ステートメント”

UNIQUE 関数

配列を検索して、値が配列に存在するかどうかを判別します。

カテゴリ: 配列

返されるデータの種類: 値が配列に見つからない場合は、ブール値の TRUE が返されます。値 TRUE は、値が一意の値であることを意味します。それ以外の場合は、FALSE が返されます。

構文

UNIQUE (*array*, <*value*>);

必須引数

array
カウント可能な数の値を指定します。

オプション引数

value
文字または数値の定数、変数または式を指定します。値が数値の場合、SAS は BEST.出力形式を使用してその値を文字列に変換します。

unpackData 関数

指定された形式に従って、base64Encode によってパックされた文字列をアンパックします。

構文

unpackData("*name*")

必須引数

name
文字列のアンパックが必要な変数または式の名前を指定します。

関連項目

- “base64Decode 関数”
- “base64Encode 関数”

UPLOAD_CASLSTORE 関数

指定された CASL ソースコード文字列から指定されたコードストレージリポジトリに CASL 関数をアップロードします。

構文

形式 1: **UPLOAD_CASLSTORE**({NAME=*store-name* <, CASLIB=*caslib-name*> <, REPLACE=TRUE | FALSE> <, ENCRYPT=TRUE | FALSE>}, *casl-code–1*, <, ...*casl-code-n*>);

形式 2: **UPLOAD_CASLSTORE**({PATH=*path-name* <, REPLACE=TRUE | FALSE> <, ENCRYPT=TRUE | FALSE>}, *casl-code–1*, <, ...*casl-code-n*>);

必須引数

location
CASL コードが保存される場所を指定します。

casl-code
アップロードする CASL 関数コードを指定します。コードをカンマで区切ると、一度に複数の関数を指定できます。

オプション引数

ENCRYPT=TRUE | FALSE
アップロードされた関数を暗号化するかどうかを指定します。

例

```
proc cas;
  source funcs;
    function c2f(c);
      /*(°C×9/5)+32*/
      return ((c*9/5)+32);
    end;
    function f2c(f);
      /* (°F−32)×5/9 */
      return ((f-32)*5/9);
    end;
  endsource;
  upload_caslstore({caslib="casuser",
name="store1", replace=true}, funcs);
run;

proc cas;
  cs = caslstore({caslib="casuser", name="store1"});
  DegF=37.0;
  DegC=37.0;
  f=round(c2f(DegC),.1);
  c=round(f2c(DegF),.1);
  print "NOTE: " degF "°F is " c "°C.";
  print "NOTE: " degC "°C is " f "°F.";
run;
```

例のコード 3.29　ログに返される結果:

```
NOTE: 37 °F is 2.8 °C.
NOTE: 37 °C is 98.6 °F.
```

関連項目

- “CASLSTORE 関数”
- “DEFAULT_CASLSTORE 関数”
- “SOURCE ステートメント”

UUID 関数

指定されたセッションのユニバーサル一意識別子(UUID)を返します。

カテゴリ: コントロール
ステータス

返されるデータの種類: 36 文字の UUID

構文

UUID ("*session-reference*");

必須引数

session-reference
UUID を取得するセッション名を指定します。

注: 有効なセッション名が使用されていない場合は、NULL 文字列が返されます。

例: セッションのアクションキューにあるアクションのリスト

この例では、listactionq アクションを実行して、UUID で識別されるセッション Mysess のキューに現在入っているアクションをリストします。この例では、セッション Mysess が SAS クライアントで実行されており、action ステートメントの ASYNC=パラメーターを使用して実行のためにキューに入れられたアクションが 1 つ以上あることを想定しています。

```
*options cashost="cloud.example.com" casport=5570;
cas casauto;                                /* 1 */
proc cas;
  session casauto;
  uuid = uuid("mysess");                    /* 2 */
  if uuid != "" then
    do;
      session.listactionq result=res / uuid=uuid;  /* 3 */
      print res;
    end;
run;
quit;
```

1 必要に応じて、セッション Casauto を作成します。システムオプション CASHOST=および CASPORT=を、サイトで有効なホストとポートに設定します。

2 関数 UUID を使用してセッション Mysess の UUID を取得し、その結果を変数 Uuid に割り当てます。

3 セッション Mysess の UUID のが正常に取得された場合は、listactionq UUID=パラメーターの UUID を使用して、セッション Mysess の現在のアクションキューを取得します。結果を出力します。

結果

次の結果はその一例です。

Connection ID	Sequence Number	Timestamp	Action	Action Parameter Count	Action Tag	Action is Active
4	12	08/15/19 16	myActionSet.prepareData	1	data_prep	Running
4	13	08/15/19 16	sessionProp.addFmtLib	3	load_formats	Active
4	14	08/15/19 16	simple.summary	3	summarize	Active

WAIT_FOR_NEXT_ACTION 関数

アクションが完了するのを待ちます。

カテゴリ: サーバー側

操作: この関数は、クライアント側の非同期アクションで機能します。

構文

WAIT_FOR_NEXT_ACTION (*job-name*);

必須引数

job-name
関数が完了するのを待つジョブの名前を指定します。

例

例 1: wait_for_next_action: ジョブ名なし

```
 job = wait_for_next_action(0);
do while(job); do;
        print ;
        print "Job [" i "]" ;
        do  k,j over job;
```

```
        print k " "  j;
      end;
      job = wait_for_next_action(0);
      i = i + 1;
      end;
end;
```

例 2: wait_for_next_action: ジョブ名あり

```
job = wait_for_next_action({'a','b'});
do while(job); do;
      print ;
      print "Job [" i "]" ;
      do  k,j over job;
        print k " "  j;
      end;
      job = wait_for_next_action(0);
      i = i + 1;
      end;
end;
```

4

共通関数

概要

関数は CASL 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。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。CASL 言語では、CASL に固有の機能を提供するビルトイン関数と、SAS ソフトウェアに共通する機能を提供する関数が提供されます。また、独自のユーザー定義関数を作成することもできます。CASL 関数の例を次に示します。

関数の種類	例	参照
ビルトイン	readpath ("/u/sasdemo/ds/samplecode.sas");	“CASL ビルトイン関数” (p. 91)
共通	datetime();	“共通関数” (p. 179)
ユーザー定義	SharedBday(365,n);	“FUNCTION ステートメント” (p. 44)

日付定数、時間定数、日時定数

日付定数、時間定数、日時定数は、単一引用符または二重引用符で囲まれた日付、時間、日時であり、その後に値の種類を示す D(日付)、T(時間)、または DT(日時)が続きます。D、T、DT を使用して日付定数、時間定数、日時定数を作成する方法の

例については、“Example: Define Date, Time, and Datetime Values in Date Constants” (*SAS Programmer's Guide: Essentials*)を参照してください。

定数	形式	例
日付	'*ddmmm<yy>yy*'D “*ddmmm<yy>yy*”D	'*1jan2018*'D "*01jan18*"D
時間	'*hh:mm<:ss.s>*'T “*hh:mm<:ss.s>*”T	'*9:25*'T "*9:25:19pm*"T
日時	'*ddmmm<yy>yy:hh:mm<:ss.s>*'DT “*ddmmm<yy>yy:hh:mm<:ss.s>*”DT	'*01may18:9:30:00*'DT "*18jan2018:9:27:05am*"DT
タイムゾーン指定子を含む日時 (ISO 8601 規格)	'*yyyy-mm-dd*T*hh:mm:ss*Z'DT '*yyyy-mm-dd*T*hh:mm:ss*+|-*hh:ss*'DT	'*2018-07-20T12:00:00Z*'DT '*2018-05-17T09:15:30-05:00*'DT

> 重要　UTC または ISO 8601 の日時定数は、Zulu タイムゾーン表示または協定世界時からの数値オフセットを持ち、システムのタイムゾーンオフセットに従って内部値を調整することによってローカルタイムに変換されます。この調整は、システムの TIMEZONE オプションが明示的に設定されているかどうかに関係なく行われます。TIMEZONE=システムオプションが指定された場合、SAS では、TIMEZONE=システムオプションで指定した値に基づいて UTC と ISO 8601 の DATETIME 定数を変換します。TIMEZONE=システムオプションが指定されない場合、SAS では、システムの UTC タイムゾーンオフセットに基づいて UTC と ISO 8601 の DATETIME 定数を変換します。

引用符の中にある末尾の空白または先頭の空白は、日付定数、時間定数、日時定数の処理には影響しません。

関連項目

“Example: Define Date, Time, and Datetime Values in Date Constants” (*SAS Programmer's Guide: Essentials*)

関数カテゴリ

表 4.1　カテゴリと定義のテーブル

カテゴリ	定義
CAS	CAS サーバーで実行される CALL ルーチンと関数です。 関数のリストについては、CAS を参照してください。
文字	文字データに基づいて情報を返します。 関数のリストについては、文字を参照してください。
記述統計	平均、中央値、標準偏差などの統計値を返します。 関数のリストについては、記述統計を参照してください。
財務	株式のヨーロピアンオプションの利息、定期的支払い、償却、価格などの金融価値を計算します。 関数のリストについては、財務を参照してください。
乱数	特定の分布からランダム変量を返します。 関数のリストについては、乱数を参照してください。
切り捨て	多くの場合、ファジーまたはゼロファジーを使用して、数値を切り捨てて数値を返します。 関数のリストについては、切り捨てを参照してください。

カテゴリ	言語要素	説明
CAS	BLACKPTPRC 関数 (p. 226)	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	言語要素	説明
	BLKSHCLPRC 関数 (p. 228)	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。
	COMPOUND 関数 (p. 265)	複利パラメーターを返します。
	FUZZ 関数 (p. 335)	引数が整数の 1E-12 以内の場合、最も近い整数を返します。
	GARKHCLPRC 関数 (p. 338)	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。
	MEDIAN 関数 (p. 435)	中央値を返します。
	SMALLEST 関数 (p. 547)	k 番目に小さい非 null 値または非欠損値を返します。
	STDERR 関数 (p. 549)	平均の標準誤差を返します。
	UNIFORM 関数 (p. 576)	一様分布からランダム変量を返します。
記述統計	SMALLEST 関数 (p. 547)	k 番目に小さい非 null 値または非欠損値を返します。
	STDERR 関数 (p. 549)	平均の標準誤差を返します。
切り捨て	FUZZ 関数 (p. 335)	引数が整数の 1E-12 以内の場合、最も近い整数を返します。
財務	BLACKPTPRC 関数 (p. 226)	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. 228)	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。
	COMPOUND 関数 (p. 265)	複利パラメーターを返します。
	GARKHCLPRC 関数 (p. 338)	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。
文字	CAT 関数 (p. 237)	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。
乱数	UNIFORM 関数 (p. 576)	一様分布からランダム変量を返します。

ディクショナリ

ABS 関数

絶対値を返します。

返されるデータの種類: BIGINT、DECIMAL、DOUBLE、NUMERIC

構文

ABS(*expression*)

必須引数

expression
数値の定数、変数または式を指定します。

データの種類 BIGINT、DECIMAL、DOUBLE、NUMERIC

詳細

結果が引数のデータ型の範囲に収まらない数値の場合、ABS 関数は失敗します。

この関数の引数が数値以外の場合、その引数は DOUBLE に変換されます。いずれかの引数が DOUBLE または REAL の場合、すべての引数が(まだ変換されていなければ) DOUBLE に変換され、結果は DOUBLE になります。それ以外の場合で、いずれかの引数が DECIMAL のときは、すべての引数が(まだ変換されていなければ) DECIMAL に変換され、結果は DECIMAL になります。それ以外の場合は、すべての引数が BIGINT に変換され、結果は BIGINT になります。

例

```
proc cas;
  x=abs(2.4);
  y=abs(-3);
  print "x=" x;
  print "y=" y;
run;
```

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);	x=2.4
y=abs(-3);	y=3

AIRY 関数

Airy 関数の値を返します。

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。

例

```
proc cas;
  x=airy(2.0);
  y=airy(-2.0);
 print "x=" x;
 print "y=" y;
run;
```

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
y=airy(-2.0);	0.2274074282

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。

返されるデータの種類: DOUBLE

注: この関数は VARCHAR 型をサポートしています。

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。

データの種類 CHAR、NCHAR

オプション引数

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

ANYALNUM 関数の結果は、有効になっている変換テーブル(“TRANTAB= システムオプション” (*SAS 各国語サポート(NLS): リファレンスガイド*)を参照)に直接依存し、ENCODING および LOCALE システムオプションに間接的に依存します。

ANYALNUM 関数は、文字列で最初に出現する数字、大文字または小文字を検索します。このような文字が検出されると、ANYALNUM はその文字の文字列の位置を返します。このような文字が検出されないと、ANYALNUM は 0 の値を返します。

1 つの引数のみを使用する場合、ANYALNUM は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。

- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

ANYALNUM は、次のいずれかに該当する場合にゼロ値を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

ANYALNUM 関数は、文字列から英数字を検索します。NOTALNUM 関数は、文字列から英数字でない文字を検索します。

例: 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。

```
proc cas;
  string='Next = Last + 1;';
  j=999999;
  do until(j=0);
    j=anyalnum(string, 1-j);
    if j=0 then print 'The end';
    else do;
      c=substr(string, j, 1);
    print "j=" j "c=" c;
       end;
     end;
  end;
run;
```

SAS は次の出力をログに書き込みます。

```
j=15 c=1
j=11 c=t
j=10 c=s
j=9 c=a
j=8 c=L
j=4 c=t
j=3 c=x
j=2 c=e
j=1 c=N
The end
```

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。

返されるデータの種類: DOUBLE

注: この関数は VARCHAR 型をサポートしています。

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。

データの種類 CHAR、NCHAR

オプション引数

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

ANYALPHA 関数の結果は、有効になっている変換テーブル(“TRANTAB= システムオプション” (*SAS 各国語サポート(NLS): リファレンスガイド*)を参照)に直接依存し、ENCODING および LOCALE システムオプションに間接的に依存します。

ANYALPHA 関数は、文字列で最初に出現する大文字または小文字を検索します。このような文字が検出されると、ANYALPHA はその文字の文字列の位置を返します。このような文字が検出されないと、ANYALPHA は 0 の値を返します。

1 つの引数のみを使用する場合、ANYALPHA は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。

- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

ANYALPHA は、次のいずれかに該当する場合にゼロ値を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

ANYALPHA 関数は、文字列からアルファベットを検索します。NOTALPHA 関数は、文字列からアルファベットでない文字を検索します。

例: 文字列からのアルファベットの検索

次の例では、ANYALPHA 関数を使用して文字列の左から右へアルファベットを検索します。

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

SAS は次の出力をログに書き込みます。

```
j=2 c=e
j=3 c=e
j=4 c=t
j=9 c=n
j=16 c=E
That's all
```

関連項目

“NOTALPHA 関数”

ANYCNTRL 関数

文字列から制御文字を検索し、最初に検索された文字の位置を返します。

データの種類: DOUBLE

注: この関数は VARCHAR 型をサポートしています。

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。

データの種類 CHAR、NCHAR

オプション引数

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

ANYCNTRL 関数の結果は、有効になっている変換テーブル(“TRANTAB= システムオプション” (*SAS 各国語サポート(NLS): リファレンスガイド*)を参照)に直接依存し、ENCODING および LOCALE システムオプションに間接的に依存します。

ANYCNTRL 関数は、文字列で最初に出現する制御文字を検索します。このような文字が検出されると、ANYCNTRL はその文字の文字列の位置を返します。このような文字が検出されないと、ANYCNTRL は 0 の値を返します。

1 つの引数のみを使用する場合、ANYCNTRL は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

次のいずれかの条件が満たされると、ANYCNTRL は値に 0 を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

ANYCNTRL 関数は、文字列から制御文字を検索します。NOTCNTRL 関数は、文字列から制御文字でない文字を検索します。

関連項目

“NOTCNTRL 関数”

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。

返されるデータの種類: DOUBLE

注: この関数は VARCHAR 型をサポートしています。

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。

データの種類 CHAR、NCHAR

オプション引数

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

ANYDIGIT 関数は、TRANTAB、ENCODING または LOCALE システムオプションに依存しません。

ANYDIGIT 関数は、文字列で最初に出現する数字を検索します。このような文字が検出されると、ANYDIGIT はその文字の文字列の位置を返します。このような文字が検出されないと、ANYDIGIT は 0 の値を返します。

1 つの引数のみを使用する場合、ANYDIGIT は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

ANYDIGIT は、次のいずれかに該当する場合にゼロ値を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

ANYDIGIT 関数は、文字列から数字を検索します。NOTDIGIT 関数は、文字列から数字でない文字を検索します。

例

次の例では、ANYDIGIT 関数を使用して数字を検索します。

```
proc cas;
  string='Next = _n_ + 12E3;';
  j=1;
  do until(j=0);
    j=anydigit(string, j+1);
    if j=0 then print +3 "The end";
    else do;
      c=substr(string, j, 1);
      print "j=" j "c=" c;
    end;
  end;
run;
```

SAS は次の出力をログに書き込みます。

```
j=14 c=1
j=15 c=2
j=17 c=3
The end
```

関連項目

“NOTDIGIT 関数”

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。

データの種類: DOUBLE

注: この関数は VARCHAR 型をサポートしています。

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。

データの種類 CHAR、NCHAR

オプション引数

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

ANYFIRST 関数は、TRANTAB、ENCODING または LOCALE システムオプションに依存しません。

ANYFIRST 関数は、VALIDVARNAME=V7 の SAS 変数名の開始文字として有効な文字のうち最初に出現する文字を文字列から検索します。アンダースコア(_)および英語

の大文字または小文字が有効な文字になります。このような文字が検出されると、ANYFIRST はその文字の文字列の位置を返します。このような文字が検出されないと、ANYFIRST は 0 の値を返します。

1 つの引数のみを使用する場合、ANYFIRST は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

ANYFIRST は、次のいずれかに該当する場合にゼロ値を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

ANYFIRST 関数は、VALIDVARNAME=V7 の SAS 変数名の開始文字として有効な文字のうち最初に出現する文字を文字列から検索します。NOTFIRST 関数は、VALIDVARNAME=V7 の SAS 変数名の開始文字として有効でない文字のうち最初に出現する文字を文字列から検索します。

例

次の例では、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。

```
proc cas;
  string='Next = _n_ + 12E3;';
  j=1;
  do until(j=0);
    j=anyfirst(string, j+1);
    if j=0 then print 'The end';
    else do;
      c=substr(string, j, 1);
      print "j=" j;
      print "c=" c;
    end;
  end;
run;
```

SAS は次の出力をログに書き込みます。

```
j=2 c=e
j=3 c=x
j=4 c=t
j=8 c=_
j=9 c=n
j=10 c=_
j=16 c=E
The end
```

関連項目

“NOTFIRST 関数”

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。

返されるデータの種類: DOUBLE

注: この関数は VARCHAR 型をサポートしています。

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。

データの種類 CHAR、NCHAR

オプション引数

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

ANYGRAPH 関数の結果は、有効になっている変換テーブル(“TRANTAB= システムオプション” (*SAS 各国語サポート(NLS): リファレンスガイド*)を参照)に直接依存し、ENCODING および LOCALE システムオプションに間接的に依存します。

ANYGRAPH 関数は、文字列で最初に出現するグラフィカル文字を検索します。グラフィカル文字は、空白以外の印刷可能文字として定義されます。このような文字が検出されると、ANYGRAPH はその文字の文字列の位置を返します。このような文字が検出されないと、ANYGRAPH は 0 の値を返します。

1 つの引数のみを使用する場合、ANYGRAPH は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

ANYGRAPH は、次のいずれかに該当する場合にゼロ値を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

ANYGRAPH 関数は、文字列からグラフィカル文字を検索します。NOTGRAPH 関数は、文字列からグラフィカル文字でない文字を検索します。

例: 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。

```
proc cas;
  string='Next = _n_ + 12E3;';
  j=1;
  do until(j=0);
    j=anygraph(string, j+1);
    if j=0 then print 'The end';
    else do;
      c=substr(string, j, 1);
      print "j=" j;
      print "c=" c;
    end;
  end;
run;
```

SAS は次の出力をログに書き込みます。

```
j=2 c=e
j=3 c=x
j=4 c=t
j=6 c==
j=8 c=_
j=9 c=n
j=10 c=_
j=12 c=+
j=14 c=1
j=15 c=2
j=16 c=E
j=17 c=3
j=18 c=;
The end
```

関連項目

“NOTGRAPH 関数”

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。

返されるデータの種類: DOUBLE

注: この関数は VARCHAR 型をサポートしています。

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。

データの種類 CHAR、NCHAR

オプション引数

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

ANYLOWER 関数の結果は、有効になっている変換テーブル(“TRANTAB= システムオプション” (*SAS 各国語サポート(NLS): リファレンスガイド*)を参照)に直接依存し、ENCODING および LOCALE システムオプションに間接的に依存します。

ANYLOWER 関数は、文字列で最初に出現する小文字を検索します。このような文字が検出されると、ANYLOWER はその文字の文字列の位置を返します。このような文字が検出されないと、ANYLOWER は 0 の値を返します。

1 つの引数のみを使用する場合、ANYLOWER は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

ANYLOWER は、次のいずれかに該当する場合にゼロ値を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

ANYLOWER 関数は、文字列から小文字を検索します。NOTLOWER 関数は、文字列から小文字でない文字を検索します。

例

次の例では、ANYLOWER 関数を使用して文字列から小文字を検索します。

```
proc cas;
  string='Next = _n_ + 12E3;';
  j=1;
  do until(j=0);
    j=anylower(string, j+1);
    if j=0 then print "The end";
    else do;
      c=substr(string, j, 1);
      print "j=" j "c=" c;
    end;
  end;
run;
```

SAS は次の出力をログに書き込みます。

```
j=2 c=e
j=3 c=x
j=4 c=t
j=9 c=n
The end
```

関連項目

“NOTLOWER 関数”

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。

返されるデータの種類: DOUBLE

注: この関数は VARCHAR 型をサポートしています。

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。

データの種類 CHAR、NCHAR

オプション引数

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

ANYNAME 関数は、TRANTAB、ENCODING または LOCALE システムオプションに依存しません。

ANYNAME 関数は、VALIDVARNAME=V7 の SAS 変数名で有効な文字のうち最初に出現する文字を文字列から検索します。アンダースコア(_)、数字、英語の大文字または小文字が有効な文字になります。このような文字が検出されると、ANYNAME はその文字の文字列の位置を返します。このような文字が検出されないと、ANYNAME は 0 の値を返します。

1 つの引数のみを使用する場合、ANYNAME は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

ANYNAME は、次のいずれかに該当する場合にゼロ値を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

ANYNAME 関数は、VALIDVARNAME=V7 の SAS 変数名で有効な文字のうち最初に出現する文字を文字列から検索します。NOTNAME 関数は、VALIDVARNAME=V7 の SAS 変数名で有効でない文字のうち最初に出現する文字を文字列から検索します。

例

次の例では、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。

```
proc cas;
  string='Next = _n_ + 12E3;';
  j=1;
  do until(j=0);
    j=anyname(string, j+1);
    if j=0 then print 'The end';
    else do;
      c=substr(string, j, 1);
      print "j=" j "c=" c;
    end;
  end;
run;
```

SAS は次の出力をログに書き込みます。

```
j=2 c=e
j=3 c=x
j=4 c=t
j=8 c=_
j=9 c=n
j=10 c=_
j=14 c=1
j=15 c=2
j=16 c=E
j=17 c=3
The end
```

関連項目

“NOTNAME 関数”

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。

返されるデータの種類: DOUBLE

注: この関数は VARCHAR 型をサポートしています。

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。

データの種類 CHAR、NCHAR

オプション引数

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

ANYPRINT 関数の結果は、有効になっている変換テーブル(“TRANTAB= システムオプション” (*SAS 各国語サポート(NLS): リファレンスガイド*)を参照)に直接依存し、ENCODING および LOCALE システムオプションに間接的に依存します。

ANYPRINT 関数は、文字列で最初に出現する印刷可能文字を検索します。このような文字が検出されると、ANYPRINT はその文字の文字列の位置を返します。このような文字が検出されないと、ANYPRINT は 0 の値を返します。

1 つの引数のみを使用する場合、ANYPRINT は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

ANYPRINT は、次のいずれかに該当する場合にゼロ値を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

ANYPRINT 関数は、文字列から印刷可能文字を検索します。NOTPRINT 関数は、文字列から印刷不可文字を検索します。

例: 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。

```
proc cas;
  string='Next = _n_ + 12E3;';
  j=1;
  do until(j=0);
    j=anyprint(string, j+1);
    if j=0 then print 'The end';
    else do;
      c=substr(string, j, 1);
      print "j=" j;
      print "c=" c;
    end;
  end;
run;
```

SAS は次の出力をログに書き込みます。

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

関連項目

“NOTPRINT 関数”

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。

返されるデータの種類: DOUBLE

注: この関数は VARCHAR 型をサポートしています。

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。

返されるデータの種類 DOUBLE

オプション引数

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類　DOUBLE

詳細

ANYPUNCT 関数の結果は、有効になっている変換テーブル(“TRANTAB= システムオプション” (*SAS 各国語サポート(NLS): リファレンスガイド*)を参照)に直接依存し、ENCODING および LOCALE システムオプションに間接的に依存します。

ANYPUNCT 関数は、文字列で最初に出現する句読文字を検索します。このような文字が検出されると、ANYPUNCT はその文字の文字列の位置を返します。このような文字が検出されないと、ANYPUNCT は 0 の値を返します。

1 つの引数のみを使用する場合、ANYPUNCT は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

ANYPUNCT は、次のいずれかに該当する場合にゼロ値を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

ANYPUNCT 関数は、文字列から句読文字を検索します。NOTPUNCT 関数は、文字列から句読文字でない文字を検索します。

例: 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。

```
proc cas;
  string='Next = _n_ + 12E3;';
  j=1;
  do until(j=0);
    j=anypunct(string, j+1);
    if j=0 then print "The end";
    else do;
      c=substr(string, j, 1);
      print "j=" j;
      print "c=" c;
    end;
```

```
    end;
run;
```

SAS は次の出力をログに書き込みます。

```
j=8 c=_
j=10 c=_
j=18 c=;
The end
```

関連項目

“NOTPUNCT 関数”

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。

返されるデータの種類: DOUBLE

注: この関数は VARCHAR 型をサポートしています。

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。

返されるデータの種類 CHAR、NCHAR

オプション引数

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

ANYSPACE 関数の結果は、有効になっている変換テーブル(“TRANTAB= システムオプション” (*SAS 各国語サポート(NLS): リファレンスガイド*)を参照)に直接依存し、ENCODING および LOCALE システムオプションに間接的に依存します。

ANYSPACE 関数は、文字列で最初に出現する空白、水平タブ、垂直タブ、キャリッジリターン、ラインフィード、フォームフィードの文字を検索します。このような文字が検出されると、ANYSPACE はその文字の文字列の位置を返します。このような文字が検出されないと、ANYSPACE は 0 の値を返します。

1 つの引数のみを使用する場合、ANYSPACE は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

ANYSPACE は、次のいずれかに該当する場合にゼロ値を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

ANYSPACE 関数は、文字列で最初に出現する空白、水平タブ、垂直タブ、キャリッジリターン、ラインフィード、フォームフィードの文字を検索します。NOTSPACE 関数は、文字列で最初に出現する空白、水平タブ、垂直タブ、キャリッジリターン、ラインフィード、フォームフィードでない文字を検索します。

例: 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。

```
proc cas;
  string='Next = _n_ + 12E3;';
  j=1;
  do until(j=0);
    j=anyspace(string, j+1);
    if j=0 then print "The end";
    else do;
      c=substr(string, j, 1);
      print "j=" j;
      print "c=" c;
    end;
  end;
```

```
run;
```

SAS は次の出力をログに書き込みます。

```
j=5 c=
j=7 c=
j=11 c=
j=13 c=
The end
```

関連項目

“NOTSPACE 関数”

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。

注: この関数は VARCHAR 型をサポートしています。

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
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

ANYUPPER 関数の結果は、有効になっている変換テーブル(“TRANTAB= システムオプション” (*SAS 各国語サポート(NLS): リファレンスガイド*)を参照)に直接依存し、ENCODING および LOCALE システムオプションに間接的に依存します。

ANYUPPER 関数は、文字列で最初に出現する大文字を検索します。このような文字が検出されると、ANYUPPER はその文字の文字列の位置を返します。このような文字が検出されないと、ANYUPPER は 0 の値を返します。

1 つの引数のみを使用する場合、ANYUPPER は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

ANYUPPER は、次のいずれかに該当する場合にゼロ値を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

ANYUPPER 関数は、文字列から大文字を検索します。NOTUPPER 関数は、文字列から大文字でない文字を検索します。

例

次の例では、ANYUPPER 関数を使用して文字列から大文字を検索します。

```
proc cas;
  string='Next = _n_ + 12E3;';
  j=1;
  do until(j=0);
    j=anyupper(string, j+1);
    if j=0 then print "The end";
    else do;
      c=substr(string, j, 1);
      print "j=" j  "c=" c;
    end;
  end;
run;
```

SAS は次の出力をログに書き込みます。

```
j=16 c=E
The end
```

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。

注: この関数は VARCHAR 型をサポートしています。

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
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

ANYXDIGIT 関数は、TRANTAB、ENCODING または LOCALE システムオプションに依存しません。

ANYXDIGIT 関数は、文字列で最初に出現する数字または A、B、C、D、E、F の大文字や小文字を検索します。このような文字が検出されると、ANYXDIGIT はその文字の文字列の位置を返します。このような文字が検出されないと、ANYXDIGIT は 0 の値を返します。

1 つの引数のみを使用する場合、ANYXDIGIT は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

ANYXDIGIT は、次のいずれかに該当する場合にゼロ値を返します。

- 検索文字が見つからない。

- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

ANYXDIGIT 関数は、文字列から 16 進文字を検索します。NOTXDIGIT 関数は、文字列から 16 進文字でない文字を検索します。

例

次の例では、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。

```
proc cas;
  string='Next = _n_ + 12E3;';
  j=1;
  do until(j=0);
    j=anyxdigit(string, j+1);
    if j=0 then print "The end";
    else do;
      c=substr(string, j, 1);
      print "j=" j   "c=" c;
    end;
  end;
run;
```

SAS は次の出力をログに書き込みます。

```
j=2 c=e
j=14 c=1
j=15 c=2
j=16 c=E
j=17 c=3
The end
```

ARCOS 関数

逆余弦を返します。

構文

ARCOS (*argument*)

必須引数

argument
数値の定数、変数または式を指定します。

範囲 -1 から 1 まで

詳細

ARCOS 関数は、引数の逆余弦(逆コサイン)を返します。戻り値はラジアンで示されます。

例

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。

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。

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

例

次の例では、逆双曲線余弦を計算します。

```
proc cas;
  x=arcosh(5);
  x1=arcosh(13);
  print "x=" x;
  print "y=" y;
run;
```

SAS は次の出力をログに書き込みます。

```
x=2.2924316696
y=3.2566139548
```

ARSIN 関数

逆正弦を返します。

構文

ARSIN(*argument*)

必須引数

argument
数値の定数、変数または式を指定します。

範囲 -1 から 1 まで

詳細

ARSIN 関数は、引数の逆正弦(逆サイン)を返します。戻り値はラジアンで示されます。

例

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。

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。$-\infty < x < \infty$

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

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。

例

次の例では、逆双曲線正弦を計算します。

```
proc cas;
  x=arsinh(5);
```

```
  y=arsinh(-5);
  print "x=" x;
  print "y=" y;
run;
```

SAS は次の出力をログに書き込みます。

```
x=2.3124383413
y=-2.312438341
```

ARTANH 関数

逆双曲線正接を返します。

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。$ARTANH(x) = \frac{1}{2} log\left(\frac{1+x}{1-x}\right)$

例

次の例では、逆双曲線正接を計算します。

```
proc cas;
  x=artanh(0.5);
  print "x=" x;
run;
```

SAS は次の出力をログに書き込みます。

```
x=0.5493061443
```

ATAN 関数

逆正接を返します。

構文

ATAN(*argument*)

必須引数

argument
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- π /2から π /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。

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

proc cas;
  x=atan(0);
  print "x=" x;
run;
	x=0

proc cas;
  x=atan(1);
  print "x=" x;
run;
	x=0.7853981634

SAS ステートメント	結果

proc cas;
  x=atan(-9.0);
  print "x=" x;
run;
	x=-1.460139106

ATAN2 関数

2 つの数値変数の比率の逆正接を返します。

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-πから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。

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
a=atan2(-1, 0.5);	-1.107148718
b=atan2(6, 8);	0.6435011088
c=atan2(5, -3);	2.1112158271

BAND 関数

2 つの引数のビットごとの論理積を返します。

返されるデータの種類: DOUBLE

構文

BAND(*expression-1, expression-2*)

引数

expression-1, expression-2
数値に評価される有効な式を指定します。

範囲 0 から(2^{32})-1 まで(両端を含む)

データの種類 DOUBLE

例

次のステートメントは、BAND 関数を示しています。

ステートメント	結果
x=band(9,11);	9
x=band(15,5);	5

BETA 関数

beta 関数の値を返します。

返されるデータの種類: DOUBLE

構文

BETA(*a*, *b*)

引数

a
第 1 形状パラメーターです。

範囲 *a* >0

データの種類 DOUBLE

b
第 2 形状パラメーターです。

範囲 *b* >0

データの種類 DOUBLE

詳細

BETA 関数は、次の式で数学的に求められます。

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

注:

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

前述の式で、$\Gamma(.)$はガンマ関数です。

式を計算できない場合、BETA は欠損値を返します。

例

次のステートメントは、BETA 関数を示しています。

ステートメント	結果
x=beta(5,3);	0.0095238095238
x=beta(15,45);	1.6710294365008E-15

BETAINV 関数

ベータ分布から分位点を返します。

返されるデータの種類: DOUBLE

構文

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

引数

p

数値の確率です。

範囲 $0 \le p \le 1$

データの種類 DOUBLE

a

数値の形状パラメーターです。

範囲 $a > 0$

データの種類 DOUBLE

b

数値の形状パラメーターです。

範囲 $b > 0$

データの種類 DOUBLE

詳細

BETAINV 関数は、ベータ分布(形状パラメーターは *a* および *b*)の *p* 分位点を返します。ベータ分布のオブザベーションが返される分位点以下になる確率は *p* です。

注: BETAINV は、PROBBETA 関数の逆数です。

例

次の例は、BETAINV 関数を示しています。

```
proc cas;
 y=betainv(0.001, 2, 4);
 print "y=" y;
run;
```

次の行が SAS ログに書き込まれます。

```
y= 0.0101017879
```

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。

返されるデータの種類: DOUBLE

構文

BLACKCLPRC(*E*, *t*, *F*, *r*, *sigma*)

引数

E

権利行使価格を指定する正の非欠損値。

要件 *E* および *F* は同じ単位で指定します。

データの種類 DOUBLE

t

満期までの時間を年単位で指定する非欠損値。

データの種類 DOUBLE

F
先物価格を指定する正の非欠損値。

要件 *F* および *E* は同じ単位で指定します。

データの種類 DOUBLE

r
連続複利の無リスク年金利を指定する正の非欠損値。

データの種類 DOUBLE

sigma
ボラティリティ (*r* の分散の平方根)を指定する正の非欠損分数。

データの種類 DOUBLE

詳細

BLACKCLPRC 関数は、Black モデルに基づき、先物のヨーロピアンオプションのコール価格を計算します。この関数は次の関係に基づきます。

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

引数

F
先物価格を指定します。

N
累積正規密度関数を指定します。

E
オプションの権利行使価格を指定します。

r
無リスク金利を指定します。これは連続複利を使用して表される年利です。

t
有効期限までの時間を年単位で指定します。

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

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

前述の式には次の引数が適用されます。

σ
原資産のボラティリティを 指定します。

σ^2
利益率の分散を指定します。

t=0 となる特別な場合には、次の式が真です。

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

比較

BLACKCLPRC 関数は、Black モデルに基づき、先物のヨーロピアンオプションのコール価格を計算します。BLACKPTPRC 関数は、Black モデルに基づき、先物のヨーロピアンオプションのプット価格を計算します。これらの関数はスカラー値を返します。

例

次のステートメントは、BLACKCLPRC 関数を示しています。

ステートメント	結果
	----+----1----+----2--
a=blackclprc(50, .25, 48, .05, .25);	1.55130142723117
b=blackclprc(9, 1/12, 10, .05, .2);	1

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: CAS
財務

返されるデータの種類: DOUBLE

構文

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

引数

E
権利行使価格を指定する正の非欠損値。

要件 *E* および *F* は同じ単位で指定します。

データの種類 DOUBLE

t
満期までの時間を年単位で指定する非欠損値。

データの種類 DOUBLE

F
先物価格を指定する正の非欠損値。

要件 *F* および *E* は同じ単位で指定します。

データの種類 DOUBLE

r
連続複利の無リスク年金利を指定する正の非欠損値。

データの種類 DOUBLE

sigma
ボラティリティ (*r* の分散の平方根)を指定する正の非欠損分数。

データの種類 DOUBLE

詳細

BLACKPTPRC 関数は、Black モデルに基づき、先物のヨーロピアンオプションのプット価格を計算します。この関数は次の関係に基づきます。

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

引数

E
オプションの権利行使価格を指定します。

r
無リスク金利を指定します。これは連続複利を使用して表される年利です。

t
有効期限までの時間を年単位で指定します。

F
先物価格を指定します。

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

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

前述の式には次の引数が適用されます。

σ
原資産のボラティリティを 指定します。

σ^2
利益率の分散を指定します。

t=0 となる特別な場合には、次の式が真です。

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

比較

BLACKPTPRC 関数は、Black モデルに基づき、先物のヨーロピアンオプションのプット価格を計算します。BLACKCLPRC 関数は、Black モデルに基づき、先物のヨーロピアンオプションのコール価格を計算します。これらの関数はスカラー値を返します。

例

次のステートメントは、BLACKPTPRC 関数を示しています。

ステートメント	結果
	----+----1----+----2--
a=blackptprc(298, .25, 350, .06, .25);	1.85980563934969
b=blackptprc(145, .5, 170, .05, .2);	1.41234979911583

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: CAS
財務

返されるデータの種類: DOUBLE

構文

BLKSHCLPRC(*E*, *t*, *S*, *r*, *sigma*)

引数

E
権利行使価格を指定する正の非欠損値。

要件 *E* および *S* は同じ単位で指定します。

データの種類 DOUBLE

t
満期までの時間を年単位で指定する非欠損値。

データの種類 DOUBLE

S
株価を指定する正の非欠損値。

要件 *S* および *E* は同じ単位で指定します。

データの種類 DOUBLE

r
連続複利の無リスク年金利を指定する正の非欠損値。

データの種類 DOUBLE

sigma
原資産のボラティリティを 指定する正の非欠損分数。

データの種類 DOUBLE

詳細

BLKSHCLPRC 関数は、Black-Scholes モデルに基づき、株式のヨーロピアンオプションのコール価格を計算します。この関数は次の関係に基づきます。

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

引数

S
株価を指定する正の非欠損値。

N
累積正規密度関数を指定します。

E
オプションの権利行使価格を指定する正の非欠損値。

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

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

前述の式には次の引数が適用されます。

t
有効期限までの時間を年単位で指定します。

r
無リスク金利を指定します。これは連続複利を使用して表される年利です。

σ
ボラティリティ (分散の平方根)を指定します。

σ^2
利益率の分散を指定します。

t=0 となる特別な場合には、次の式が真です。

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

比較

BLKSHCLPRC 関数は、Black-Scholes モデルに基づき、株式のヨーロピアンオプションのコール価格を計算します。BLKSHPTPRC 関数は、Black-Scholes モデルに基づき、株式のヨーロピアンオプションのプットを計算します。これらの関数はスカラー値を返します。

例

次のステートメントは、BLKSHCLPRC 関数を示しています。

ステートメント	結果
	----+----1----+----2--
a=blkshclprc(50, .25, 48, .05, .25);	1.79894201954462
b=blkshclprc(9, 1/12, 10, .05, .2);	1

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。

返されるデータの種類: DOUBLE

構文

BLKSHPTPRC(*E*, *t*, *S*, *r*, *sigma*)

引数

E
権利行使価格を指定する正の非欠損値。

要件 *E* および *S* は同じ単位で指定します。

データの種類 DOUBLE

t
満期までの時間を年単位で指定する非欠損値。

データの種類 DOUBLE

S
株価を指定する正の非欠損値。

要件 *S* および *E* は同じ単位で指定します。

データの種類 DOUBLE

r
連続複利の無リスク年金利を指定する正の非欠損値。

データの種類 DOUBLE

sigma
原資産のボラティリティを 指定する正の非欠損分数。

データの種類 DOUBLE

詳細

BLKSHPTPRC 関数は、Black-Scholes モデルに基づき、株式のヨーロピアンオプションのプットを計算します。この関数は次の関係に基づきます。

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

引数

S
株価を指定する正の非欠損値。

E
オプションの権利行使価格を指定する正の非欠損値。

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

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

前述の式には次の引数が適用されます。

t
有効期限までの時間を年単位で指定します。

r
無リスク金利を指定します。これは連続複利を使用して表される年利です。

σ
ボラティリティ (分散の平方根)を指定します。

σ^2
利益率の分散を指定します。

t=0 となる特別な場合には、次の式が真です。

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

比較

BLKSHPTPRC 関数は、Black-Scholes モデルに基づき、株式のヨーロピアンオプションのプットを計算します。BLKSHCLPRC 関数は、Black-Scholes モデルに基づき、株式のヨーロピアンオプションのコール価格を計算します。これらの関数はスカラー値を返します。

例

次のステートメントは、BLKSHPTPRC 関数を示しています。

ステートメント	結果
	----+----1----+----2--
a=blkshptprc(230,.5,290,.04,.25);	1.56597442946066
b=blkshptprc(350,.3,400,.05,.2);	1.64091943067592

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。

返されるデータの種類: DOUBLE

構文

BLSHIFT(*expression-1*, *expression-2*)

引数

expression-1
数値に評価される有効な式を指定します。

範囲 0 から(2^{32})-1 まで(両端を含む)

データの種類 DOUBLE

expression-2
数値に評価される有効な式を指定します。

範囲 0 から 31(0 と 31 を含む)

データの種類 DOUBLE

例

次のステートメントは、BLSHIFT 関数を示しています。

ステートメント	結果
x=blshift(7,2);	28

BNOT 関数

引数のビットごとの論理否定を返します。

返されるデータの種類: DOUBLE

構文

BNOT(*expression*)

引数

expression
数値に評価される有効な式を指定します。

範囲 0 から(2^{32})-1 まで(両端を含む)

データの種類 DOUBLE

例

次のステートメントは、BNOT 関数を示しています。

ステートメント	結果
a=bnot(16);	4294967279

BOR 関数

2 つの引数のビットごとの論理和を返します。

返されるデータの種類: DOUBLE

構文

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

引数

expression-1, expression-2
数値に評価される有効な式を指定します。

範囲 0 から(2^{32})-1 まで(両端を含む)

データの種類 DOUBLE

例

次のステートメントは、BOR 関数を示しています。

ステートメント	結果
a=bor(4,8);	12

BRSHIFT 関数

2 つの引数を使ってビットを論理右シフトした値を返します。

返されるデータの種類: DOUBLE

構文

BRSHIFT(*expression-1*, *expression-2*)

引数

expression-1
数値に評価される有効な式を指定します。

範囲 0 から(2^{32})-1 まで(両端を含む)

データの種類 DOUBLE

expression-2
数値に評価される有効な式を指定します。

範囲 0 から 31(0 と 31 を含む)

データの種類 DOUBLE

例

次のステートメントは、BRSHIFT 関数を示しています。

ステートメント	結果
x=brshift(64,2);	16

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。

返されるデータの種類: DOUBLE

構文

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

引数

expression-1, expression-2
数値に評価される有効な式を指定します。

範囲 0 から(2^{32})-1 まで(両端を含む)

データの種類 DOUBLE

例

次のステートメントは、BXOR 関数を示しています。

ステートメント	結果
x=bxor(128,64);	192

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。

返されるデータの種類: CHAR、NCHAR、NVARCHAR、VARCHAR

構文

BYTE(*n*)

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
データの種類	DOUBLE

詳細

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。

例

次のステートメントは、BYTE 関数を示しています。

ステートメント	結果	
	ASCII	EBCDIC
	----+----1----+----2	----+----1----+----2
x=byte(80);	P	&

CAT 関数

先頭または末尾の空白を削除せずに、連結文字列を返します。

カテゴリ:	文字
返されるデータの種類:	CHAR、NCHAR、NVARCHAR、VARCHAR

構文

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

引数

item
文字または数値の、定数、変数または式を指定します。*item* が数値の場合、その値は BEST*w*.出力形式を使用して文字列に変換されます。この場合、先頭の空白は削除され、ログにメモは記録されません。

詳細

返される変数の長さ

CAT 関数が、あらかじめ長さが割り当てられていない変数に値を返す場合、その変数には 200 バイトの長さが設定されます。| |や...連結演算子が、あらかじめ長さが割り当てられていない変数に値を返す場合、その変数には、連結される値の長さの合計となる長さが設定されます。

返される変数の長さ: 特殊な場合

CAT 関数は、変数または一時バッファーに値を返します。CAT 関数の戻り値の長さは次のとおりです。

- 最大 200 文字(WHERE 句および PROC SQL)
- PROC DS2 では、WHERE 句を除き、32767 文字まで
- 最大 65534 文字(CAT がマクロプロセッサから呼び出される場合)

CAT が値を一時バッファーに返す場合、バッファーの長さは呼び出し環境によって異なります。バッファーの値は CAT の処理後に切り捨てられる可能性があります。この場合、切り捨てに関するメッセージはログに出力されません。

変数またはバッファーの長さが不十分で連結結果を格納できない場合、SAS は次の手順を実行します。

- PROC DS2 と PROC SQL では、結果を空白行に変更します。
- 呼び出し環境に応じて、結果が切り捨てられたことを示す警告メッセージ、または空白値に設定されたことを示す警告メッセージがログに出力されます。
- 関数呼び出しの場所と、切り捨ての原因となった引数のリストを示すメモがログに出力されます。
- _ERROR_を 1 に設定します。

CAT 関数は、BEST*w*.出力形式で数値をフォーマットした後、数値引数から先頭と末尾の空白を削除します。

比較

CAT 関数、CATS 関数、CATT 関数、CATX 関数の結果は、*通常*、連結演算子| |と..、TRIM 関数、LEFT 関数の特定の組み合わせからの結果と同等になります。ただし、CAT 関数、CATS 関数、CATT 関数、CATX 関数のデフォルトの長さは、連結演算子を使用したときに得る長さとは異なります。

CAT 関数、CATS 関数、CATT 関数、CATX 関数を使用すると、TRIM および LEFT を使用するよりも高速になります。

例

次の例では、CAT 関数がどのように文字列を連結するのかを示します。

```
proc cas;
    x='  The 2012 Olym';
    y='pic Arts Festi';
    z='  val included works by D  ';
    a='ale Chihuly.';
    result=cat(x,y,z,a);
    print "result=" result;
  end;
run;
```

SAS は次の出力をログに書き込みます。

```
result=  The 2012 Olympic Arts Festi  val included works by D  ale Chihuly.
```

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。

返されるデータの種類: CHAR、NCHAR、NVARCHAR、VARCHAR

構文

CATQ(*modifiers* <, *delimiter*>, *item-1* <, ..., *item-n*>)

必須引数

modifier
文字定数、変数または式を指定します。これらの空白以外の各文字で CATQ 関数のアクションを変更します。空白は無視されます。次の文字を修飾子として使用できます。

1 or '
CATQ は、文字列に引用符を追加するときに単一引用符を使用します。

2 or "
CATQ は、文字列に引用符を追加するときに二重引用符を使用します。

a or A
すべての項目引数に引用符を追加します。

b or B
S または T 修飾子で削除されていない先頭や末尾の空白がある項目引数に引用符を追加します。

c or C
区切り文字としてカンマを使用します。

d or D
指定の区切り文字引数があることを示します。

h or H
区切り文字として水平タブを使用します。

m or M
最初の項目引数以降の各項目引数に区切り文字を挿入します。M 修飾子を使用しない場合、CATQ は他の修飾子で指定された処理の後に長さがゼロの項目引数の区切り文字を挿入しません。M 修飾子を使用すると、区切り文字が結果の最初または最後に表示される可能性があります。また、複数の連続する区切り文字が結果に表示される場合もあります。

n or N
値が SAS 名の通常の構文規則に準拠していない場合に項目引数を名前リテラルに変換します。名前リテラルは、引用符に囲まれた空白のない文字列で、後に文字"N"が続きます。SAS ステートメントで名前リテラルを使用するには、SAS オプション VALIDVARNAME=ANY を指定する必要があります。

q or Q
すでに引用符が含まれている項目引数に引用符を追加します。

s or S
後で処理される引数の先頭および末尾の空白を削除します。

- 区切り文字引数の先頭および末尾の空白を削除する場合、D 修飾子の*前に* S 修飾子を指定します。
- 項目引数の先頭および末尾の空白は削除するが、区切り文字引数の先頭および末尾の空白は削除*しない*場合、D 修飾子の*後に* S 修飾子を指定します。

t or T
後で処理される引数の末尾の空白を取り除きます。

- 区切り文字引数の末尾の空白を削除する場合、D 修飾子の前に T 修飾子を指定します。
- 項目引数の末尾の空白は削除するが、区切り文字引数の末尾の空白は削除しない場合、D 修飾子の後に T 修飾子を指定します。

x or X
値に印刷不可文字が含まれている場合に項目引数を 16 進リテラルに変換します。

ヒント　*modifier* が定数の場合、引用符で囲みます。*modifier* を変数名または式として表すこともできます。

A、B、N、Q、S、T、X 修飾子は、CATQ 関数内部で動作します。項目引数が変数の場合、結果がその変数に割り当てられていない限り、その変数の値は CATQ によって変更されません。

item
文字または数値の、定数、変数または式を指定します。*item* が数値の場合、その値は BEST*w*.出力形式を使用して文字列に変換されます。この場合、先頭の空白は削除され、ログにメモは記録されません。

オプション引数

delimiter
連結文字列間の区切り文字として使用する文字定数、変数または式を指定します。この引数を指定する場合、D 修飾子も指定する必要があります。

詳細

返される変数の長さ

CATQ 関数は変数に値を返します。CATQ が式内で呼び出される場合は一時バッファーに値を返します。戻り値の長さは次のとおりです。

- 最大 200 文字(WHERE 句および PROC SQL)
- 最大 32767 文字(WHERE 句以外の DATA ステップ)
- 最大 65534 文字(CATQ がマクロプロセッサから呼び出される場合)

変数またはバッファーの長さが不十分で連結結果を格納できない場合、SAS は次の手順を実行します。DATA ステップおよび PROC SQL の結果を空白値に変更します。

- DATA ステップと PROC SQL では、結果を空白値に変更します。
- 呼び出し環境に応じて、結果が切り捨てられたことを示す警告メッセージ、または空白値に設定されたことを示す警告メッセージがログに出力されます。
- 関数呼び出しの場所と、切り捨ての原因となった引数のリストを示すメモがログに出力されます。
- DATA ステップで_ERROR_を 1 に設定します。

バッファーの値は CATQ の処理後に切り捨てられる可能性があります。この場合、切り捨てに関するメッセージはログに出力されません。

基本

C、D または H 修飾子を使用しない場合、CATQ は区切り文字として空白を使用します。

modifier に引用符、あるいは 1 または 2 修飾子のいずれも指定しないと、CATQ は引用符が必要な場合に各項目引数に使用する引用符の種類を独自に決定します。次の規則が適用されます。

- CATQ は、アンパサンド(&)やパーセント(%)記号を含む文字列、または単一引用符より二重引用符の方が多い文字列には、単一引用符を使用します。
- その他のすべての文字列には二重引用符を使用します。

CATQ 関数は、結果を初期化して長さをゼロにし、各項目引数で次のアクションを実行します。

1 *item* が文字列でない場合、CATQ は BEST*w* 形式を使用して *item* を文字列に変換し、 先頭の空白を削除します。

2 S 修飾子を使用している場合、CATQ は文字列から先頭の空白を削除します。

3 S または T 修飾子を使用すると、CATQ は文字列から末尾の空白を削除します。

4 CATQ は、次の条件に基づいて引用符を追加するかどうかを決定します。
   - X 修飾子を使用する場合、文字列に制御文字が含まれていると、文字列は 16 進リテラルに変換されます。
   - N 修飾子を使用する場合、次のいずれかの条件に該当すると、文字列は名前リテラルに変換されます。
     - 文字列の最初の文字がアンダースコアまたは英字ではない。
     - 文字列に数字、アンダースコアまたは英字以外の文字が含まれている。
   - X または N 修飾子を使用しない場合、次のいずれかの条件に該当すると、CATQ は引用符を文字列に追加します。
     - A 修飾子を使用している。
     - B 修飾子を使用していて、S または T 修飾子で削除されなかった先頭や末尾の空白が文字列に含まれている。
     - Q 修飾子を使用していて、文字列に引用符が含まれている。
     - 先頭および末尾の空白を削除した状態で区切り文字と同じになる部分文字列が文字列に含まれている。

5 2 番目以降の項目引数では、CATQ は次のいずれかの条件に該当する場合に区切り文字を結果に追加します。
   - M 修飾子を使用している。
   - 前述のステップで処理された後の文字列の長さがゼロよりも大きい。

6 CATQ は結果に文字列を追加します。

比較

CATX 関数は CATQ 関数と類似していますが、CATX は引用符を追加しない点が異なります。

例: CATQ 関数による文字列の連結

次の例では、CATQ 関数がどのように文字列を連結するのかを示します。

```
proc cas;
      result1=CATQ(' ',
            'noblanks',
            'one blank',
            12345,
            '  lots of blanks    ');
      print "result1=" result1;
      result2=CATQ('CS',
            'Period (.)              ',
            'Ampersand (&)           ',
            'Comma (,)               ',
            'Double quotation marks (") ',
            '   Leading Blanks');
      print "result2=" result2;
      result3=CATQ('BCQT',
            'Period (.)              ',
            'Ampersand (&)           ',
            'Comma (,)               ',
            'Double quotation marks (") ',
            '   Leading Blanks');
      print "result3=" result3;
      result4=CATQ('ADT',
            '#=#',
            'Period (.)              ',
            'Ampersand (&)           ',
            'Comma (,)               ',
            'Double quotation marks (") ',
            '   Leading Blanks');
      print "result4=" result4;
      result5=CATQ('N',
            'ABC_123   ',
            '123    ',
            'ABC 123');
      print "result5=" result5;
    end;
run;
```

SAS は次の出力をログに書き込みます。

```
result1=noblanks "one blank" 12345 " lots of blanks "

result2=Period (.),Ampersand (&),"Comma (,)",Double quotation marks ("),Leading Blanks

result3=Period (.),Ampersand (&),"Comma (,)",'Double quotation marks (")'," Leading Blanks"

result4="Period (.)"#=#'Ampersand (&)'#=#"Comma (,)"#=#'Double quotation marks (")'#=#" Leading Blanks"

result5=ABC_123 "123"n "ABC 123"n
```

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。

返されるデータの種類: CHAR、NCHAR、NVARCHAR、VARCHAR

構文

CATS(*item*<, ...*item*>)

引数

item
文字または数値の、定数、変数または式を指定します。*item* が数値の場合、その値は BEST*w*.出力形式を使用して文字列に変換されます。この場合、先頭の空白は削除され、ログにメモは記録されません。

詳細

返される変数の長さ

CATS 関数が、あらかじめ長さが割り当てられていない変数に値を返す場合、その変数には 200 バイトの長さが設定されます。| |や...連結演算子が、あらかじめ長さが割り当てられていない変数に値を返す場合、その変数には、連結される値の長さの合計となる長さが設定されます。

返される変数の長さ: 特殊な場合

CATS 関数は、変数または一時バッファーに値を返します。CATS 関数の戻り値の長さは次のとおりです。

- 最大 200 文字(WHERE 句および PROC SQL)
- PROC DS2 では、WHERE 句を除き、32767 文字まで
- 最大 65534 文字(CATS がマクロプロセッサから呼び出される場合)

CATS が値を一時バッファーに返す場合、バッファーの長さは呼び出し環境によって異なります。バッファーの値は CATS の処理後に切り捨てられる可能性があります。この場合、切り捨てに関するメッセージはログに出力されません。

変数またはバッファーの長さが不十分で連結結果を格納できない場合、SAS は次の手順を実行します。

- PROC DS2 と PROC SQL では、結果を空白値に変更します。

- 呼び出し環境に応じて、結果が切り捨てられたことを示す警告メッセージ、または空白値に設定されたことを示す警告メッセージがログに出力されます。
- 関数呼び出しの場所と、切り捨ての原因となった引数のリストを示すメモがログに出力されます。
- _ERROR_を 1 に設定します。

CATS 関数は、BEST*w*.出力形式で数値をフォーマットした後、数値引数から先頭と末尾の空白を削除します。

比較

CAT 関数、CATS 関数、CATT 関数、CATX 関数の結果は、*通常*、連結演算子| |と..、TRIM 関数、LEFT 関数の特定の組み合わせからの結果と同等になります。ただし、CAT 関数、CATS 関数、CATT 関数、CATX 関数のデフォルトの長さは、連結演算子を使用したときに得る長さとは異なります。

CAT 関数、CATS 関数、CATT 関数、CATX 関数を使用すると、TRIM および LEFT を使用するよりも高速になります。

例

次の例では、CATS 関数がどのように文字列を連結するのかを示します。

```
proc cas;
    x='  The   Olym';
    y='pic Arts Festi';
    z='  val includes works by D  ';
    a='ale Chihuly.';
  result=cats(x,y,z,a);
  print "result=" result;
 end;
run;
```

SAS は次の出力をログに書き込みます。

```
result=The   Olympic Arts Festival includes works by Dale Chihuly.
```

CATT 関数

末尾の空白を削除して、連結文字列を返します。

返されるデータの種類: CHAR、NCHAR、NVARCHAR、VARCHAR

構文

CATT(*item*<, ...*item*>)

引数

item
文字または数値の、定数、変数または式を指定します。*item* が数値の場合、その値は BEST*w*.出力形式を使用して文字列に変換されます。この場合、先頭の空白は削除され、ログにメモは記録されません。

詳細

返される変数の長さ

CATT 関数が、あらかじめ長さが割り当てられていない変数に値を返す場合、その変数には 200 バイトの長さが設定されます。| |や...連結演算子が、あらかじめ長さが割り当てられていない変数に値を返す場合、その変数には、連結される値の長さの合計となる長さが設定されます。

返される変数の長さ: 特殊な場合

CATT 関数は、変数または一時バッファーに値を返します。CATT 関数の戻り値の長さは次のとおりです。

- 最大 200 文字(WHERE 句および PROC SQL)
- PROC DS2 では、WHERE 句を除き、32767 文字まで
- 最大 65534 文字(CATT がマクロプロセッサから呼び出される場合)

CATT が値を一時バッファーに返す場合、バッファーの長さは呼び出し環境によって異なります。バッファーの値は CATT の処理後に切り捨てられる可能性があります。この場合、切り捨てに関するメッセージはログに出力されません。

変数またはバッファーの長さが不十分で連結結果を格納できない場合、SAS は次の手順を実行します。

- PROC DS2 と PROC SQL では、結果を空白値に変更します。
- 呼び出し環境に応じて、結果が切り捨てられたことを示す警告メッセージ、または空白値に設定されたことを示す警告メッセージがログに出力されます。
- 関数呼び出しの場所と、切り捨ての原因となった引数のリストを示すメモがログに出力されます。
- _ERROR_を 1 に設定します。

CATT 関数は、BEST*w*.出力形式で数値をフォーマットした後、数値引数から先頭と末尾の空白を削除します。

比較

CAT 関数、CATS 関数、CATT 関数、CATX 関数の結果は、*通常*、連結演算子| |と..、TRIM 関数、LEFT 関数の特定の組み合わせからの結果と同等になります。ただし、CAT 関数、CATS 関数、CATT 関数、CATX 関数のデフォルトの長さは、連結演算子を使用したときに得る長さとは異なります。

CAT 関数、CATS 関数、CATT 関数、CATX 関数を使用すると、TRIM および LEFT を使用するよりも高速になります。

例

次の例では、CATT 関数がどのように文字列を連結するのかを示します。

```
proc cas;
    x='  The 2012 Olym';
    y='pic Arts Festi';
    z='  val included works by D  ';
    a='ale Chihuly.';
    result=catt(x,y,z,a);
    print "result=" result;
  end;
run;
```

SAS は次の出力をログに書き込みます。

```
result=  The 2012 Olympic Arts Festi  val included works by Dale Chihuly.
```

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。

返されるデータの種類: CHAR、NCHAR、NVARCHAR、VARCHAR

構文

CATX(*delimiter*, *item-1*<, ...*item-n*>)

引数

delimiter
連結項目間の区切り文字として使用する文字列を指定します。

item
文字または数値の、定数、変数または式を指定します。*item* が数値の場合、その値は BEST*w*.出力形式を使用して文字列に変換されます。この場合、ログにメモは記録されません。

詳細

基本

CATX 関数は、まず *item-1* を先頭および末尾の空白を削除して結果にコピーします。次に、CATX は後続の各引数 *item-i* (i=2, ..., n)で、*item-i* に 1 つ以上の空白以外の文字が含まれている場合に *delimiter* および *item-i* を結果に追加します。このとき、*item-i* から先頭および末尾の空白を削除します。CATX は、結果の先頭または末尾に区切り文字を挿入しません。空白の項目では、結果の先頭または末尾に区切り文字は生成されません。また、複数の連続する区切り文字も生成されません。

返される変数の長さ

CATX 関数は、変数または一時バッファーに値を返します。CATX 関数から返される値は、WHERE 句を除いて、最大 32767 文字まで可能です。

変数またはバッファーの長さが不十分で連結結果を格納できない場合、SAS は結果を切り捨てます。

比較

CAT 関数、CATS 関数、CATT 関数、CATX 関数の結果は、*通常*、連結演算子| |と..、TRIM 関数、LEFT 関数の特定の組み合わせからの結果と同等になります。ただし、CAT、CATS、CATT、CATX 関数のデフォルトの長さは、連結演算子の使用時に取得する長さとは異なります。

CAT 関数、CATS 関数、CATT 関数、CATX 関数を使用すると、TRIM および LEFT を使用するよりも高速になります。

注: 欠損値のある変数の場合、連結で異なる結果が生成されます。

例

次の例では、CATX 関数がどのように文字列を連結するのかを示します。最初のデータプログラムでは、値テーブルを作成します。2 番目と 3 番目のデータプログラムでは、値テーブルを入力として使用します。

```
proc cas;
  separator='%%$%%';
  x='The Olympic  ';
  y='   Arts Festival ';
```

```
  z='   includes works by ';
  a='Dale Chihuly.';
  result=catx(separator, x, y, z, a);
  print "result=" result;
run;
```

SAS は次の出力をログに書き込みます。

```
result= The Olympic%%$%%Arts Festival%%$%%includes works by%%$%%Dale Chihuly.
```

CEIL 関数

数値式以上の最小の整数を返します。

返されるデータの種類: DECIMAL、DOUBLE、NUMERIC

構文

CEIL(*expression*)

引数

expression
数値に評価される有効な式を指定します。

データの種類 DECIMAL、DOUBLE、NUMERIC

詳細

expression が null の場合、CEILING 関数は null を返します。結果が引数のデータ型の範囲に収まらない数値の場合、CEIL 関数は失敗します。

引数が DECIMAL の場合、結果は DECIMAL になります。それ以外の場合、引数は (まだ変換されていなければ) DOUBLE に変換され、結果は DOUBLE になります。

比較

CEILZ 関数とは異なり、CEIL 関数は結果をファジー処理します。引数が整数の 1E-12 内にある場合、CEIL 関数はその整数に等しくなるように結果をファジー処理します。CEILZ 関数は結果をファジー処理しません。そのため、CEILZ 関数では、予期しない結果になる可能性があります。

例

次のステートメントは、CEIL 関数を示しています。

ステートメント	結果
a=ceil(-2.4);	-2
b=ceil(1+1.e-11);	2
c=ceil(-1+1.e-11);	0
d=ceil(1+1.e-13);	1

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。

返されるデータの種類: DOUBLE

構文

CEILZ(*expression*)

引数

expression
数値に評価される有効な式を指定します。

データの種類 DOUBLE

比較

CEIL 関数とは異なり、CEILZ 関数はゼロファジーを使用します。引数が整数の 1E-12 内にある場合、CEIL 関数はその整数に等しくなるように結果をファジー処理します。CEILZ 関数は結果をファジー処理しません。そのため、CEILZ 関数では、予期しない結果になる可能性があります。

例

次のステートメントは、CEILZ 関数を示しています。

ステートメント	結果
a=ceilz(2.1);	3
b=ceilz(3);	3
c=ceilz(1+1.e-11);	2
d=ceilz(223.456);	224
e=ceilz(-223.456);	-223

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。

返されるデータの種類: VARCHAR、NVARCHAR

構文

CHOOSEC(*index-expression*, *selection-1*<, ...*selection-n*>)

引数

index-expression
数値に評価される有効な式を指定します。

データの種類 DOUBLE

selection
文字定数、変数または式を指定します。この引数の値は、CHOOSEC 関数によって返されます。

データの種類 DOUBLE

詳細

CHOOSEC 関数は、*index-expression* の値を使用して、後続の引数から選択します。たとえば、*index-expression* が 3 の場合、CHOOSEC は *selection-3* の値を返します。第 1 引数が負の場合、関数は引数のリストを逆方向に数えてその値を返します。

比較

CHOOSEC 関数は CHOOSEN 関数と類似していますが、CHOOSEN が数値を返すのに対して CHOOSEC は文字値を返す点が異なります。

例

次の例では、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。

```
proc cas;
  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');
  items={Fruit, Color, Planet, Sport};
  print "Choices Are=" items;
run;
```

SAS は次の出力をログに書き込みます。

```
Choices Are={apple,green,Mercury,baseball}
```

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。

返されるデータの種類: DOUBLE

構文

CHOOSEN(*index-expression*, *selection-1*<, ...*selection-n*>)

引数

index-expression
数値に評価される有効な式を指定します。

データの種類 DOUBLE

selection
数値の定数、変数または式を指定します。この引数の値は、CHOOSEN 関数によって返されます。

データの種類 DOUBLE

詳細

CHOOSEN 関数は、*index-expression* の値を使用して、後続の引数から選択します。たとえば、*index-expression* が 3 の場合、CHOOSEN は *selection-3* の値を返します。第 1 引数が負の場合、関数は引数のリストを逆方向に数えてその値を返します。

比較

CHOOSEN 関数は CHOOSEC 関数と類似していますが、CHOOSEN が数値を返すのに対して CHOOSEC は文字値を返す点が異なります。

例

次の例では、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。

```
proc cas;
  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);
  print "Item Number=" ItemNumber;
  print "Rank=" Rank;
  print "Score=" Score;
  print "Value=" Value;
run;
```

SAS は次の出力をログに書き込みます。

```
Item Number=679
Rank=4
Score=33
Value=1014
```

CMISS 関数

欠損引数の数を数えます。

構文

CMISS(*argument* <, ...*argument,*>)

引数

argument
定数、変数または式を指定します。*argument* は、文字値または数値のいずれかになります。

詳細

文字式の評価結果がすべて空白の文字列または長さがゼロの文字列の場合、マクロ処理で CMISS 関数を使用していない限り、その文字式は欠損とみなされます。数値式の評価結果が数値欠損値(.、._、.A、...、.Z)の場合、その数値式は欠損として数えられます。

マクロ処理で CMISS 関数を使用する場合は、ピリオド(.)を使用して、文字欠損値と数値欠損値の両方を表します。文字引数に空白または NULL 値を使用した場合、SAS はエラーを返します。

例

次のステートメントは、CMISS 関数を示しています。

ステートメント	結果
x = CMISS(1,0, ' ' , 2,5,' ');	1
x = CMISS(1, ' ');	0

CNONCT 関数

カイ 2 乗分布の非心度パラメーターを返します。

返されるデータの種類: DOUBLE

構文

CNONCT(*x*, *df*, *probability*)

必須引数

x
数値の確率変数です。

範囲 $x \geq 0$

データの種類 DOUBLE

df
数値の自由度パラメーターです。

範囲 $df > 0$

データの種類 DOUBLE

probability
確率です。

範囲 $0 < probability < 1$

データの種類 DOUBLE

詳細

CNONCT 関数は、非心度カイ 2 乗分布(パラメーターは *x*、*df*、*nc*)から負でない非心度パラメーターを返します。*probability* が心度カイ 2 乗分布(パラメーターは *x* および *df*)からの確率よりも大きい場合、この問題の平方根は存在しません。この場合、欠損値が返されます。式の負でない平方根 *nc* を検索するには、ニュートン型のアルゴリズムを使用します。

$$P_c(x|\mathrm{df}, nc) - prob = 0$$

前述の式には次の関係が適用されます。

$$P_c(x|\mathrm{df}, \mathrm{nc}) = \varepsilon^{\frac{-nc}{2}} \sum_{j=0}^{\infty} \frac{\left(\frac{nc}{2}\right)^j}{j!} P_g\left(\frac{x}{2}\middle|\frac{df}{2} + j\right)$$

前述の式には次の関係が適用されます。

$$P_g(x|a)$$

次の式によって得られるガンマ分布の確率です。

$$P_g(x|a) = \frac{1}{\Gamma(a)} \int_0^x t^{a-1} \varepsilon^{-t} dt$$

アルゴリズムで固定小数点を収束できない場合、欠損値が返されます。

例

```
proc cas;
  x=2;
  df=4;
  do nc=1 to 3 by .5;
    probability=probchi(x, df, nc);
    ncc=cnonct(x, df, probability);
  print "probability=" probability;
  print "ncc=" ncc;
  end;
run;
```

SAS は次の出力をログに書き込みます。

```
probability=0.1861097793
ncc=1
probability=0.1559154065
ncc=1.5
probability=0.1304765495
ncc=2
probability=0.1090741908
ncc=2.5
probability=0.0910916212
ncc=3
```

COALESCE 関数

数値の引数のリストから最初の非 null または非欠損値を返します。

返されるデータの種類: DOUBLE

構文

COALESCE(*expression*<, ...*expression*>)

引数

expression
数値に評価される有効な式を指定します。

データの種類 DOUBLE

詳細

COALESCE では、1 つ以上の数値式を使用できます。COALESCE 関数は、リストされている順に各式の値を確認し、最初の非 null または非欠損値を返します。リストされている値が 1 つだけの場合、COALESCE 関数はその引数の値を返します。すべての式のすべての値が null または欠損値の場合、COALESCE 関数は、ANSI モードまたは SAS モードのどちらであるかに応じて、null または欠損値を返します。

比較

COALESCE 関数は数値式を検索するのに対し、COALESCEC 関数は文字式を検索します。

例

次のステートメントは、COALESCE 関数を示しています。

ステートメント	結果
x = COALESCE(., .A, 33, 22, 44, .);	33

COALESCEC 関数

文字引数のリストから最初の非 null または非欠損値を返します。

返されるデータの種類: CHAR、NCHAR、NVARCHAR、VARCHAR

構文

COALESCEC(*expression*<, ...*expression*>)

引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

詳細

COALESCEC では、1 つ以上の文字式を使用できます。COALESCEC 関数は、リストされている順に各式の値を確認し、最初の非 null または非欠損値を返します。リストされている値が 1 つだけの場合、COALESCEC 関数はその式の値を返します。すべての式のすべての値が null または欠損値の場合、COALESCEC 関数は、ANSI モードまたは SAS モードのどちらであるかに応じて、null または欠損値を返します。

比較

COALESCEC 関数は文字式を検索するのに対し、COALESCE 関数は数値式を検索します。

例

次のステートメントは、COALESCEC 関数を示しています。

ステートメント	結果
a=coalescec('', 'Hello');	Hello
a=coalescec('', 'Goodbye', 'Hello');	Goodbye

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。

返されるデータの種類: DOUBLE

構文

COMB(*n*, *r*)

引数

n

負でない整数で要素の合計数を表します。サンプルはこの中から選ばれます。

データの種類 DOUBLE

r
負でない整数で選ばれた要素の数を表します。

制限事項 $r \leq n$

データの種類 DOUBLE

詳細

COMB 関数の数学的表現は、次の式によって得られます。

$$COMB\,(n, r) = \binom{n}{r} = \frac{n!}{r!\,(n-r)!}$$

前述の式で、$n \geq 0$、$r \geq 0$ 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。やや大きな値の場合、COMB 関数を計算できないことがあります。

例

次のステートメントは、COMB 関数を示しています。

ステートメント	結果
x=comb(27, 2);	351

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。

返されるデータの種類: CHAR、NCHAR、NVARCHAR、VARCHAR

構文

COMPARE(*string-1*, *string-2*<, *modifiers*>)

引数

string-1
文字列を評価するか、文字列に強制変換できる文字定数、変数または式を指定します。

データの種類　CHAR、NCHAR、NVARCHAR、VARCHAR

string-2
文字列を評価するか、文字列に強制変換できる文字定数、変数または式を指定します。

データの種類　CHAR、NCHAR、NVARCHAR、VARCHAR

modifiers
COMPARE 関数のアクションを変更できる文字列を指定します。次の文字の 1 つ以上を有効な修飾子として使用できます。I (大文字と小文字の区別を無視する)、L (先頭の空白を削除する)、N (引用符を削除して、大文字と小文字の区別を無視する)、: (コロン。長い方の文字列が短い方の長さか 1 文字になるように切り捨てる)。

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、テーブルまたは変数名に使用できなかった特殊文字(空白文字を含む)を使用できるようになります。COMPARE で文字列が名前リテラルとして認識されるように、最初の文字を引用符にする必要があります。
:	*string-1* と *string-2* の短い方の文字列の長さまたは 1 文字(どちらか大きい方)になるように長い方の文字列を切り捨てます。この修飾子を指定しない場合、長い方の文字列の長さと同じになるように短い方の文字列に空白が埋め込まれます。

データの種類　CHAR、NCHAR、NVARCHAR、VARCHAR

ヒント　COMPARE は、修飾子として使用される空白を無視します。

詳細

基本

COMPARE 関数では、修飾子の出現順序に意味があります。

- "LN"は各文字列から先頭の空白を削除した後、名前リテラルから引用符を削除します。
- "NL"は名前リテラルから引用符を削除した後、各文字列から先頭の空白を削除します。

COMPARE 関数では、*string–1* と *string–2* に違いがない場合はゼロ値を返します。引数が異なる場合、次のようになります。

- 並べ替え順序で *string–1* が *string–2* よりも前にある場合は結果の符合が負になり、*string–1* が *string–2* よりも後にある場合は正になります。
- 結果の大きさは、文字列が異なる最も左側の文字の位置と等しくなります。

DBCS の互換性

この関数に相当する DBCS 関数は、KCOMPARE です。COMPARE 関数と KCOMPARE 関数には若干の違いがあります。どちらの関数もさまざまな数の引数を受け入れますが、第 3 引数の使用法には互換性がありません。次の例は、構文の違いを示しています。

COMPARE(*string-1*, *string-2*<, *modifiers*>)

KCOMPARE(*string-1*<, *position*<, *count*>>, *string-2*)

例: 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。

```
proc cas;
  pad1=compare('abc', 'abc          ');
  pad2=compare('abc', 'abcdef       ');
  truncate1=compare('abc', 'abcdef',':');
  truncate2=compare('abcdef', 'abc',':');
  blank=compare('', 'abc',         ':');
  columns={'pad1', 'pad2', 'truncate1', 'truncate2', 'blank'};
  coltypes={'int64', 'int64', 'int64', 'int64', 'int64'};
  row1={pad1, pad2, truncate1, truncate2, blank};
  mytable=newtable('mytable', columns, coltypes, row1);
  print mytable;
  describe mytable;
run;
```

SAS は次の出力をログに書き込みます。

```
[mytable] Table ( [1] Rows [5] columns
 Column Names:
 [1] pad1          [pad1          ] (int64)
 [2] pad2          [pad2          ] (int64)
 [3] truncate1     [truncate1     ] (int64)
 [4] truncate2     [truncate2     ] (int64)
 [5] blank         [blank         ] (int64)
```

アウトプット 4.1　COMPARE 関数の使用

mytable: Results

pad1	pad2	truncate1	truncate2	blank
0	-4	0	0	-1

COMPBL 関数

文字列内の単語間にある複数の空白を取り除きます。

返されるデータの種類: CHAR、NCHAR、NVARCHAR、VARCHAR

構文

COMPBL(*character-expression*)

引数

character-expression
文字列を評価するか、文字列に強制変換できる有効な式であり、取り除く文字列を指定します。

データの種類　CHAR、NCHAR、NVARCHAR、VARCHAR

詳細

文字列中に 2 つ以上の空白が連続して出現するたびに、COMPBL 関数はそれらを 1 つの空白に変換して、複数の空白を削除します。

比較

COMPRESS 関数は、特定の文字が文字列に出現するたびにそれを削除します。ソース文字列から削除する文字として空白を指定した場合、COMPRESS 関数は COMPBL 関数と同様の動作をします。ただし、COMPRESS 関数は、ソース文字列からすべての空白を削除します。COMPBL 関数は、複数の空白を 1 つの空白に圧縮するため、1 つの空白には影響しません。

例

次のステートメントは、COMPBL 関数を示しています。

ステートメント	結果
	----+----1----+---–2--
a=compbl('January Status');	January Status
string='125 E. Main St.'; street=compbl(string);	125 E. Main St.

COMPFUZZ 関数

2 つの数値をファジー比較します。

返されるデータの種類: DOUBLE

構文

COMPFUZZ(*expression-1*, *expression-2*<, *fuzz*<, *scale*>>)

引数

expression-1
数値に評価される有効な式を指定します。

データの種類 DOUBLE

expression-2
数値に評価される有効な式を指定します。

データの種類 DOUBLE

fuzz
比較の相対しきい値を指定する負でない数値です。1 以上の値はマシン精度の倍数単位で扱われます。

デフォルト 1024

データの種類 DOUBLE

scale
スケール係数を指定します。

デフォルト MAX (ABS (*expression-1*), ABS (*expression-2*))

データの種類 DOUBLE

詳細

COMPFUZZ 関数は、4 つの引数をすべて指定した場合、次の値を返します。

- -1 (*expression-1* < *expression-2* – threshold の場合)
- 0 (ABS(*expression-1* - *expression-2*) ≤ threshold の場合)
- 1 (*expression-1* > *expression-2* + threshold の場合)

次の関係が存在します。

- 0 ≤ *fuzz* < 1 のとき、threshold = *fuzz* * ABS(*scale*)
- 1 ≤ *fuzz* < 1 / CONSTANT('MACEPS')のとき、threshold = *fuzz* * ABS(*scale*) * CONSTANT('MACEPS')

COMPFUZZ は浮動小数点のアンダーフローまたはオーバーフローを 回避します。

比較

COMPFUZZ 関数は 2 つの浮動小数点の数値を比較し、比較に基づく値を返します。ROUND 関数は、第 2 引数の倍数にきわめて近い値に引数を丸めます。結果は第 2 引数の正確な倍数でない場合があります。

例

浮動小数点計算では、数値を加算する順番によって合計値が異なる場合があります。x1 から x*n* までの *n* 個の数値の和を計算するときの、ある浮動小数点エラーの近似境界は次の式で表されます。

n * machine_precision * sum (abs(x1) + ... + abs(x*n*))

したがって、*n* 個の浮動小数点の数値の和を COMPFUZZ 関数で比較するのに、次の DATA ステップに示すように、*n* をファジー値、絶対値の和をスケール係数として使用できます。

```
proc cas;
  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);
  print "sum1=" sum1;
  print "sum2=" sum2;
  print "diff=" 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);
  print "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);
  print "fuzz and scale (correct):  " compfuzz;
run;
```

次の行が SAS ログに書き込まれます。

```
sum1=1.8095238209
 sum2=1.8095238095
 diff=1.1353266E-8
 fuzz only (wrong):        1
 fuzz and scale (correct):  0
```

COMPOUND 関数

複利パラメーターを返します。

カテゴリ: CAS
財務

返されるデータの種類: DOUBLE

構文

COMPOUND(*a*, *f*, *r*, *n*)

引数

a
初期の金額を指定します。

範囲　$a \geq 0$

データの種類　DOUBLE

f
将来の金額を指定します(*n* 期間の終了時)。

範囲　$f \geq 0$

データの種類　DOUBLE

r
期間単位の金利を分数で指定します。

範囲　$r \geq 0$

データの種類　DOUBLE

n
複利期間数を指定します。

範囲　$n \geq 0$

データの種類　DOUBLE

詳細

COMPOUND 関数は、複利計算の 4 つの引数のリストの欠損引数を返します。これらの引数には次の式の関係があります。

$$f = a(1 + r)^n$$

引数は 1 つを欠損値とする必要があります。他の 3 つの値から複利パラメーターが計算されます。結果を変換して丸めた数字にする調整は行われません。

n=0 のとき、

$$f = a$$

および

$$(1 + r)^n$$

は 1 です。

注: *r* を欠損値に選ぶと COMPOUND からエラーが返されます。

例

$2000 の投資で名目年利 9%、30 か月複利運用したときの累計額は次の式で表します。

```
future=compound(2000, ., 0.09/12, 30);
```

返される値は 2502.544 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%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。

返されるデータの種類: CHAR、NCHAR、NVARCHAR、VARCHAR

構文

COMPRESS(*character-expression*<, *character-list-expression*>)

引数

character-expression
文字式に評価され、指定文字の削除対象になる有効な式を指定します。

要件 文字のリテラル文字列は単一引用符で囲みます。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

character-list-expression
文字リストを初期化する変数または任意の有効な式を指定します。

デフォルトでは、このリスト内の文字が *character-expression* から削除されます。

要件 文字のリテラル文字列は単一引用符で囲みます。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

詳細

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*	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。
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。

数字、プラス記号またはマイナス記号を削除するには、次の関数呼び出しを使用します。

```
COMPRESS(source, "1234567890+-");
```

例

例 1: 空白の取り除き

これらの例は、文字列から空白を削除する方法を示しています。

ステートメント	結果
	----+----1
a='AB C D '; b=compress(a);	ABCD
a='AB C D'; b=compress(a,'A ');	BCD

例 2: 母音の取り除き

これらの例は、文字列から文字を削除する方法を示しています。

ステートメント	結果
	----+----1----+----2----+----3
x='123-4567-8901 e 234-5678-9012 i'; y=compress(x,'aeiou');	123-4567-8901 234-5678-9012

CONSTANT 関数

マシン定数および数学定数を計算します。

返されるデータの種類: DOUBLE

構文

CONSTANT(*constant*<, *parameter*>)

引数

constant
返す定数を特定する文字定数、変数または式を指定します。有効な定数は次のとおりです。'E'、'EULER'、'PI'、'EXACTINT'、'BIG'、'LOGBIG'、'SQRTBIG'、'SMALL'、'LOGSMALL'、'SQRTSMALL'、'MACEPS'、'LOGMACEPS'、'SQRTMACEPS'。

返す定数を特定する文字定数、変数または式を指定します。有効な定数は次のとおりです。

定数	説明
'E'	自然数の底
'EULER'	Euler 定数
'PI'	PI
'EXACTINT' [,nbytes]	正確な整数
'BIG'	最大倍精度浮動小数点数
'LOGBIG' [,base]	*base* を底とする BIG の対数
'SQRTBIG'	BIG の平方根
'SMALL'	最小倍精度浮動小数点数
'LOGSMALL' [,base]	*base* を底とする SMALL の対数
'SQRTSMALL'	SMALL の平方根
'MACEPS'	マシン精度の定数

定数	説明
'LOGMACEPS' [,base]	*base* を底とする MACEPS の対数
'SQRTMACEPS'	MACEPS の平方根

parameter
数値パラメーターで、*constant* で指定された一部の定数と一緒にオプション引数として使用できます。この *parameter* を使用すると、CONSTANT 関数の機能が変更されます。

詳細

概要

注意
一部の動作環境では、ランタイムライブラリに制限があるために、ハードウェアが持つ浮動小数点数を最大限に活用できません。 そのような場合、CONSTANT 関数はラインタイムライブラリの制限内で使用できる値を返そうとします。たとえば、ランタイムライブラリで EXP(LOG(CONSTANT('BIG')))を計算できない場合は、CONSTANT('LOGBIG')で LOG(CONSTANT('BIG'))と同じ値が返されず、EXP(CONSTANT('LOGBIG'))を計算できる値が返されます。

自然数の底

CONSTANT('E')

自然数の底は次の式で表されます。

$$\lim_{x \to 0} (1+x)^{\frac{1}{x}} \approx 2.718281828459045$$

Euler 定数

CONSTANT('EULER')

Euler の定数は次の式で表されます。

$$\lim_{n \to \infty} \left\{ \sum_{j=1}^{j=n} \frac{1}{j} - \log(n) \right\} \approx 0.577215664901532860$$

PI

CONSTANT('PI')

PI(円周率)は、円の周の長さと直径の比です。この定数を計算する式は多数あります。この数列を表す式の 1 例は次のとおりです。

$$4 \sum_{j=0}^{j=\infty} \frac{(-1)^j}{2j+1} \approx 3.14159265358979323846$$

正確な整数

CONSTANT('EXACTINT'<, *nbytes*>)

引数

nbytes
バイト数を表す数値です。

デフォルト 8

範囲 2 ≤nbytes ≤ 8

正確な整数は、絶対値で *k* 以下のすべての整数を、長さが *nbytes* の SAS 数値変数で正確に表現できる最大整数 *k* です。この情報は、容量を節約するために SAS 数値変数をデフォルトの 8 バイトから小さいバイト数に調整するときにあらかじめ知っておくと役に立つ情報です。

最大倍精度浮動小数点数

CONSTANT('BIG')

使用しているコンピューターで表現できる最大倍精度浮動小数点数(8 バイト)を返します。

BIG の対数

CONSTANT('LOGBIG'<, *base*>)

引数

base
対数の底となる数値です。

デフォルト 自然数の底、E。

制限事項 指定する *base* は 1+SQRTMACEPS の値より大きくする必要があります。

base を底として、使用しているコンピューターで表現できる最大倍精度浮動小数点数(8 バイト)の対数を返します。

CONSTANT('LOGBIG', *base*)以下に累乗された指定の *base* は、累乗演算(**)を使っても、オーバーフローを発生させず安全に累乗できます。

浮動小数点数は、CONSTANT('LOGBIG')以下の任意の浮動小数点数で累乗関数 EXP を使っても、オーバーフローを発生させず安全に累乗できます。

BIG の平方根

CONSTANT('SQRTBIG')

使用しているコンピューターで表現できる最大倍精度浮動小数点数(8 バイト)の平方根を返します。

浮動小数点数は、CONSTANT('SQRTBIG')以下の任意の浮動小数点数で、オーバーフローを発生させず安全に平方根を求めることができます。

最小倍精度浮動小数点数

CONSTANT('SMALL')

使用しているコンピューターで表現できる最小倍精度浮動小数点数(8 バイト)を返します。

SMALL の対数

CONSTANT('LOGSMALL'<, *base*>)

引数

base
対数の底となる数値です。

デフォルト　自然数の底、E。

制限事項　指定する *base* は 1+SQRTMACEPS の値より大きくする必要があります。

base を底として、使用しているコンピューターで表現できる最小倍精度浮動小数点数(8 バイト)の対数を返します。

CONSTANT('LOGSMALL', *base*)以上に累乗された指定の *base* は、累乗演算(**)を使って安全に累乗できます。アンダーフローまたは 0 にはなりません。

浮動小数点数は、CONSTANT('LOGSMALL')以上の任意の浮動小数点数で累乗関数 EXP を使って安全に累乗できます。アンダーフローまたは 0 にはなりません。

SMALL の平方根

CONSTANT('SQRTSMALL')

コンピューターで表現できる最小倍精度浮動小数点数(8 バイト)の平方根を返します。

浮動小数点数は、CONSTANT('SQRTBIG')以上の任意の浮動小数点数で、安全に平方根を求めることができます。アンダーフローまたはゼロにはなりません。

マシンの精度

CONSTANT('MACEPS')

この場合、$1+\varepsilon > 1$のように、ある整数 *j* に対して最小倍精度浮動小数点数(8 バイト) $\varepsilon = 2^{-j}$が返されます。

この定数は有限精度の計算を行うときに重要です。

MACEPS の対数

CONSTANT('LOGMACEPS'<, *base*>)

引数

base
対数の底となる数値です。

デフォルト　自然数の底、E。

制限事項　指定する *base* は 1+SQRTMACEPS の値より大きくする必要があります。

base を底とする CONSTANT('MACEPS')の対数を返します。

MACEPS の平方根

CONSTANT('SQRTMACEPS')

CONSTANT('MACEPS')の平方根を返します。

例

次の例では、CONSTANT 関数を使用して、さまざまな定数の値を返します。

```
proc cas;
   a=constant('E');
   b=constant('EULER');
   c=constant('PI');
   d=constant('EXACTINT');
   e=constant('BIG');
   f=constant('LOGBIG');
   g=constant('SQRTBIG');
   h=constant('SMALL');
   i=constant('LOGSMALL');
   j=constant('SQRTSMALL');
   k=constant('MACEPS');
   l=constant('LOGMACEPS');
   m=constant('SQRTMACEPS');
   print 'a= ' a;
   print 'b= ' b;
   print 'c= ' c;
   print 'd= ' d;
   print 'e= ' e;
   print 'f= ' f;
   print 'g= ' g;
   print 'h= ' h;
   print 'i= ' i;
   print 'j= ' j;
   print 'k= ' k;
   print 'l= ' l;
   print 'm= ' m;
 end;
run;
```

SAS は次の出力をログに書き込みます。

```
a= 2.7182818285
b= 0.5772156649
c= 3.1415926536
d= 9.0071993E15
e= 1.797693E308
f= 709.78271289
g= 1.340781E154
h= 2.22507E-308
i= -708.4018337
j= 1.49167E-154
k= 2.220446E-16
l= -36.04365339
m= 1.4901161E-8
```

CONVX 関数

列挙キャッシュフローのコンベクシティを返します。

返されるデータの種類: DOUBLE

構文

CONVX(*y*, *f*, *c(1)*, ..., *c(k)*)

引数

y

期間当たりの有効満期利回りを指定します。

範囲 $0 < y < 1$

データの種類 DOUBLE

ヒント *y* を分数で表した場合、配当は小数値として記述する必要があります。DS2 では、整数除算の結果はゼロになります。ゼロは DOUBLE に変換され、CONVX 関数の最初の引数として渡されます。CONVX 関数は、最初のパラメーターとしてゼロを渡すと欠損値を返します。

f

期間当たりのキャッシュフローの頻度を指定します。

範囲 $f > 0$

データの種類 DOUBLE

c(1), ... , c(k)
キャッシュフローのリストを指定します。

データの種類 DOUBLE

詳細

CONVX 関数は、次の式から値を返します。

$$C = \sum_{k=1}^{K} \frac{k(k+f)\frac{c(k)}{(1+y)^{\frac{k}{f}}}}{P\left((1+y)^2\right)f^2}$$

前述の式には次の関係が適用されます。

$$P = \sum_{k=1}^{K} \frac{c(k)}{(1+y)^{\frac{k}{f}}}$$

例: CONVX 関数の使用

```
proc cas;
  c=convx(.1,.33,.44,.55,.49,.50,.22,.4,.8,.01);
  print "c=" c;
run;
```

SAS は次の出力をログに書き込みます。

```
c=100.04943781
```

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。

返されるデータの種類: DOUBLE

構文

CONVXP(*A*, *c*, *n*, *K*, k_0, *y*)

引数

A
額面価格を指定します。

範囲 $A > 0$

A > 0

データの種類 DOUBLE

c
期間当たりの名目クーポンレートを小数で指定します。

範囲 0 ≤ *c* < 1

データの種類 DOUBLE

n
期間当たりのクーポン数を指定します。

範囲 *n* > 0

データの種類 DOUBLE

K
クーポンの残数を指定します。

範囲 *K* > 0

データの種類 DOUBLE

k_0
現在の日付から最初のクーポン日までの時間を指定します。期間数で表します。

範囲 $0 < k_0 \leq \frac{1}{n}$

$0 < k_0 \leq 1/n$

データの種類 DOUBLE

y
期間当たりの名目満期利回りを指定します。

範囲 *y* > 0

データの種類 DOUBLE

詳細

CONVXP 関数は、次の式から値を返します。

$$C = \frac{1}{n^2}\left(\frac{\sum_{k=1}^{K} t_k(t_k+1)\frac{c(k)}{\left(1+\frac{y}{n}\right)^{t_k}}}{P\left(1+\frac{y}{n}\right)^2}\right)$$

前述の式には次の関係が適用されます。

$$t_k = nk_0 + k - 1$$

$$c(k) = \frac{c}{n}A \quad for\ k = 1, \ldots, K-1$$

$$c(K) = \left(1+\frac{c}{n}\right)A$$

前述の式には次の関係が適用されます。

$$P = \sum_{k=1}^{K} \frac{c(k)}{\left(1+\frac{y}{n}\right)^{t_k}}$$

例: 債権のコンベクシティの計算

次の例では、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。

```
proc cas;
  c=convxp(1000,.01,4,14,.33/2,.08);
  print "c=" c;
run;
```

SAS は次の出力をログに書き込みます。

```
c=11.729001987
```

COS 関数

余弦をラジアンで返します。

返されるデータの種類: DOUBLE

構文

COS(*expression*)

引数

expression
数値に評価される有効な式です。

データの種類 DOUBLE

例

次のステートメントは、COS 関数を示しています。

ステートメント	結果
x=cos(0.5);	0.87758256189037
x=cos(0);	1
x=cos(3.14159/3);	0.50000076602519

COSH 関数

双曲線余弦をラジアンで返します。

返されるデータの種類: DOUBLE

構文

COSH(*expression*)

引数

expression
数値に評価される有効な式です。

データの種類 DOUBLE

詳細

COSH 関数は、次の式によって得られる引数の双曲線余弦を返します。

$$(e^{argument} + e^{-argument})/2$$

例

次のステートメントは、COS 関数を示しています。

ステートメント	結果
x=cosh(0);	1
x=cosh(-5.0);	74.2099485247878
x=cosh(4.37);	39.5281414700662
x=cosh(0.5);	1.12762596520638

COT 関数

余接を返します。

返されるデータの種類: DOUBLE

構文

COT(*argument*)

必須引数

argument
ラジアンで表される数値定数、変数または式を指定します。

制限事項 *argument* は、0 または PI の倍数にはできません。

比較

COT 関数と TAN 関数は次のような関係です。

cot(x)=1/tan(x)

例

次のステートメントは、COS 関数を示しています。

ステートメント	結果
x=cot(0.5);	1.83048772171245
x=cot(1);	0.64209261593433
x=cot(3.14159/3);	0.57735144856346

注: x=cot(0);を使用して COT 関数が欠損値を返した場合は、関数に無効な引数が入力されたことを示す NOTE がログに書き込まれます。これは正しい動作です。

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。

返されるデータの種類: DOUBLE

構文

COUNT(*string*, *substring*<, *modifiers*>)

引数

string
文字定数、変数または式を指定します。この中にある部分文字列を数えます。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

ヒント 文字のリテラル文字列を引用符で囲みます。

substring
string でカウントされる文字定数、変数または式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

ヒント 文字のリテラル文字列を引用符で囲みます。

modifiers
1 つ以上の修飾子を文字定数、変数または式で指定します。修飾子として使用できる文字(大文字または小文字)は次のとおりです。

i または I	数えるときに大文字と小文字を区別しません。この修飾子を指定しないと、COUNT は *substring* の文字の大文字と小文字に一致する部分文字列のみを数えます。
t または T	*string* および *substring* から末尾の空白を取り除きます。
データの種類	CHAR、NCHAR、NVARCHAR、VARCHAR
ヒント	*modifiers* が定数の場合、引用符で囲みます。一組の引用符で複数の定数を指定します。*modifiers* を変数または式として表すこともできます。

詳細

基本

COUNT 関数は左から右方向に *string* を検索し、指定した *substring* が何回出現するかを検出して出現数を返します。*string* 内に部分文字列が見つからない場合、COUNT は値 0 を返します。

注意
指定した部分文字列が文字列中に重複して 2 回出現する場合、結果は未定義になります。 たとえば、COUNT('boobooboo', 'booboo')は 1 または 2 のいずれかを返す可能性があります。

比較

COUNT 関数は文字列中の部分文字列を数えるのに対し、COUNTC 関数は文字列中の個別の文字を数えます。

例

次のステートメントは、COUNT 関数を示しています。

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

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

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。

返されるデータの種類: DOUBLE

構文

COUNTC(*string*, *charlist*<, *modifiers*>)

引数

string
文字定数、変数または式を指定します。この中にある文字を数えます。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

ヒント 文字のリテラル文字列を引用符で囲みます。

charlist
文字リストを初期化する文字定数、変数または式を指定します。COUNTC はこのリストの文字を数えます(ただし、V 修飾子を *modifiers* 引数で指定しない場合)。V 修飾子を指定すると、このリストにないすべての文字を数えます。他の修飾子を使うことでリストに文字をさらに追加できます。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

ヒント　文字のリテラル文字列を引用符で囲みます。

修飾子を処理した後でリストに文字がない場合、COUNTC はゼロを返します。

modifiers
文字定数、変数または式を指定します。これらの空白以外の各文字で COUNTC 関数のアクションを変更します。空白は無視されます。A (アルファベット文字をリストに追加)、C (制御文字を追加)、D (数字を追加)の大文字と小文字を修飾子として使用できます。他の修飾子も使用することができます。

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　アンダースコアと 英字(つまり、VALIDVARNAME=V7 を使用して SAS 変数名を開始できる文字)を文字のリストに追加します。

g 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。以降の呼び出しにおける *charlist* または *modifier* の値の変更は、COUNTC で無視されることがあります。

p 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。

データの種類　CHAR、VARCHAR

ヒント　*modifier* が定数の場合、引用符で囲みます。一組の引用符で複数の定数を指定します。

詳細

COUNTC 関数では、文字引数を NULL に指定できます。ヌル引数は長さがゼロの文字列として扱われます。文字リストに数える文字がない場合、COUNTC はゼロを返します。

注: CHAR データ型の文字列には、常に宣言された長さまで空白が埋め込まれることに注意してください。VARCHAR データ型の文字列では、宣言された長さではなく、実際の文字列の長さが返されます。

比較

COUNTC 関数は文字列中の個別の文字を数えるのに対し、COUNT 関数は文字列中の部分文字列の文字を数えます。

例

COUNTC 関数で文字列中の文字数を数えるときに、修飾子を使用する場合と使用しない場合の例を次に示します。

```
proc cas;
 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');
columns={'string', 'a', 'b', 'b_i', 'abc_i', 'abc_iv', 'abc_ivt'};
coltypes={'string','int64','int64','int64','int64','int64','int64'};
row1={string, a, b, b_i, abc_i, abc_iv, abc_ivt};
countctab=newtable('countctab', columns, coltypes, row1);
print countctab;
run;
```

アウトプット 4.2　COUNTC 関数で修飾子を使用する場合としない場合の結果

countctab: Results

string	a	b	b_i	abc_i	abc_iv	abc_ivt
Baboons Eat Bananas	5	1	3	8	16	11

COUNTW 関数

文字列中の単語数を数えます。

返されるデータの種類:　DOUBLE

構文

COUNTW(*string*<, *chars*<, *modifiers*>>)

引数

string
文字定数、変数または式を指定します。この中にある単語を数えます。

データの種類　CHAR、NCHAR、NVARCHAR、VARCHAR

chars
文字のリストを初期化する任意の文字定数、変数または式を指定します。リスト中の文字は単語を区切るための区切り文字です(ただし *modifier* 引数で K 修飾子を使用しない場合)。K 修飾子を指定すると、このリストにないすべての文字が区切り文字となります。他の修飾子を使うことでリストに文字をさらに追加できます。

データの種類　CHAR、NCHAR、NVARCHAR、VARCHAR

modifiers
文字定数、変数または式を指定します。これらの空白以外の各文字で COUNTW 関数のアクションを変更します。A (アルファベット文字をリストに追加)、C (制

御文字を追加)、D (数字を追加)の大文字と小文字を修飾子として使用できます。他の修飾子も使用することができます。

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 関数の呼び出し時毎には処理されません。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内で COUNTW を呼び出すとき、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、対になっていない引用符が含まれている場合、左から右へのスキャンと、右から左へのスキャンとでは異なる単語が生成されます。

s 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* 引数と *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。

データの種類　CHAR、VARCHAR

詳細

"単語"の定義

COUNTW 関数では、"単語"とは、次の特性のいずれかを持つ部分文字列を指します。

- 左側が区切り文字または文字列の先頭で境界設定されている
- 右側が区切り文字または文字列の末尾で境界設定されている
- 区切り文字を含まない。ただし、Q 修飾子を使用している場合で、引用符を含む部分文字列内に区切り文字が囲まれている場合を除きます。

注: "単語"の定義は SCAN 関数と COUNTW 関数の両方で共通しています。

区切り文字は、単語を区切るために指定できるいくつかの文字を示します。

ASCII および EBCDIC 環境での COUNTW 関数の使用

2 つの引数のみを指定して COUNTW 関数を使用する場合、デフォルトの区切り文字は、コンピューターで使用している文字(ASCII または EBCDIC)によって異なります。

- お使いのコンピューターが ASCII 文字を使用している場合、デフォルトの区切り文字は次のようになります。

  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。
- お使いのコンピューターが EBCDIC 文字を使用している場合、デフォルトの区切り文字は次のようになります。

  空白 ! $ % & ( ) * + , - . / ; < ¬ | ¢

null 引数の使用

COUNTW 関数では、文字引数を NULL に指定できます。ヌル引数は長さがゼロの文字列として扱われます。数値引数はヌルにできません。

M 修飾子の使用

M 修飾子を使用しない場合、単語には少なくとも 1 つの文字が含まれている必要があります。M 修飾子を使用する場合、単語の長さをゼロとすることができます。この場合、単語数は文字列中の区切り文字数に 1 を加算した数です。Q 修飾子を使用するときは引用符に囲まれている文字列中の区切り文字は数えません。

例

次の例は、COUNTW 関数に M 修飾子および P 修飾子を使う方法を示しています。

各文字列の **mp** の値の説明は次のとおりです。

- ピリオドは区切り文字であり、m 修飾子を使用すると、末尾のピリオドが参照する後続の単語は長さゼロですが、それでも 1 単語になります。つまり、ピリオドの前に 1 単語、ピリオドの後に 1 単語で、合計 2 つの単語があるということです。
- 区切り文字がないため、1 単語だけです。
- p 修飾子が区切り文字として句読点を追加するため、3 単語になります。
- p 修飾子が句読点を追加するため、/は区切り文字です。m 修飾子を使用すると、先頭の/が長さゼロの先頭の単語を参照して、合計 6 単語になります。
- 最初の\はエスケープ文字です 2 番目の\は区切り文字なので、6 単語があります。

```
proc cas;
    string1='The quick brown fox jumps over the lazy dog.';
    string2='       Leading blanks';
    string3='2+2=4';
    string4='/unix/path/names/use/slashes';
    string5='\Windows\Path\Names\Use\Backslashes';
  a      = countw(string1, 'a');
  b      = countw(string2, 'b');
  b_i    = countw(string3, 'b', 'i');
  abc_i  = countw(string4, 'abc', 'i');
  abc_j  = countw(string5, 'abc', 'j');
 columns={'a', 'b', 'b_i', 'abc_i', 'abc_j'};
 coltypes={'string','int64','int64','int64','int64','int64','int64'};
 row1={string1, a, b, b_i, abc_i, abc_j};
 row2={string2, a, b, b_i, abc_i, abc_j};
 row3={string3, a, b, b_i, abc_i, abc_j};
 row4={string4, a, b, b_i, abc_i, abc_j};
 row5={string5, a, b, b_i, abc_i, abc_j};
 countwtab=newtable('countwtab', columns, coltypes, row1, row2, row3, row4, row5);
 print countwtab;
run;
```

アウトプット 4.3　COUNTW 関数使用時の結果

countwtab: Results

a	b	b_i	abc_i	abc_j
The quick brown fox jumps over the lazy dog.	2	2	1	4
Leading blanks	2	2	1	4
2+2=4	2	2	1	4
/unix/path/names/use/slashes	2	2	1	4
\Windows\Path\Names\Use\Backslashes	2	2	1	4

CSC 関数

余割を返します。

返されるデータの種類: DOUBLE

構文

CSC(*argument*)

必須引数

argument
ラジアンで表される数値定数、変数または式を指定します。

制限事項　*argument* は、0 または PI の倍数にはできません。

データの種類　DOUBLE

比較

CSC 関数と SIN 関数は次のような関係です。

csc(x)=1/sin(x)

例

次のステートメントは、CSC 関数を示しています。

ステートメント	結果
x=csc(0.5);	2.08582964293348
x=csc(1);	1.18839510577812
x=csc(3.14159/3);	1.1547011280666

注: x=csc(0);を使用して CSC 関数が欠損値を返した場合は、関数に無効な引数が入力されたことを示す NOTE がログに書き込まれます。これは正しい動作です。

CSS 関数

修正済み平方和を返します。

返されるデータの種類: DOUBLE

構文

CSS(*expression*<, ...*expression*>)

引数

expression
数値に評価される有効な式を指定します。

要件 少なくとも 1 つの非 null または非欠損の式が必要です。

データの種類 DOUBLE

例

次のステートメントは、CSS 関数を示しています。

ステートメント	結果
a=css(5,9,3,6);	18.75
b=css(5,8,9,6,.);	10
c=css(8,9,6,.);	4.66666666666666

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。

返されるデータの種類: DOUBLE

構文

CUMIPMT(*rate*, *number-of-periods*, *principal-amount*<, *start-period*><, *end-period*><, *type*>)

引数

rate
支払期間ごとの利率を指定します。

データの種類 DOUBLE

number-of-periods
支払期間の数を指定します。*number-of-periods* は正の整数にする必要があります。

データの種類 DOUBLE

principal-amount
ローンの元金を指定します。欠損値が指定されている場合、ゼロとみなされます。

データの種類 DOUBLE

start-period
計算の開始期間を指定します。

データの種類 DOUBLE

end-period
計算の終了期間を指定します。

データの種類 DOUBLE

type
期首と期末のどちらで支払がなされるのかを指定します。0 は期末の支払を表し、1 は期首の支払を表します。*type* が省略された場合、または欠損値が指定されている場合は、ゼロとみなされます。

データの種類 DOUBLE

例

- 名目年利 9%で$125,000 の 30 年ローンを組み、期末の月次払いとした場合に 2 年目に支払う累積利息は次のように計算します。

```
proc cas;
    TotalInterest= CUMIPMT(0.09/12, 360, 125000, 13, 24, 0);
    print 'Total Interest=' TotalInterest;
run;
```

  計算によって$11,135.23 という値が返されます。

- 同じローンの最初の期間に支払われる利息は、次の方法で計算されます。

```
proc cas;
   first_period_interest= CUMIPMT(0.09/12, 360, 125000, 1, 1, 0);
   print 'Total Interest=' first_period_interest;
run;
```

  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。

返されるデータの種類: DOUBLE

構文

CUMPRINC(*rate*, *number-of-periods*, *principal-amount*<, *start-period*><, *end-period*><, *type*>)

引数

rate
支払期間ごとの利率を指定します。

データの種類 DOUBLE

number-of-periods
支払期間の数を指定します。

要件 *number-of-periods* は正の整数にする必要があります。

データの種類 DOUBLE

principal-amount
ローンの元金を指定します。

データの種類 DOUBLE

注 欠損値または null 値が指定されている場合、ゼロとみなされます。

start-period
計算の開始期間を指定します。

データの種類 DOUBLE

end-period
計算の終了期間を指定します。

データの種類 DOUBLE

type
期首と期末のどちらで支払がなされるのかを指定します。0 は期末の支払を表し、1 は期首の支払を表します。*type* が省略された場合、または欠損値が指定されている場合は、ゼロとみなされます。

データの種類 DOUBLE

例

- 名目年利 9%で$125,000 の 30 年ローンを組み、期末の月次払いとした場合に 2 年目に支払う累積元本は次のように計算します。

```
proc cas;
   PrincipalYear2=CUMPRINC(0.09/12, 360, 125000, 12, 24, 0);
   print 'Principal Year 2 EOP=' PrincipalYear2;
run;
```

計算によって$1008.23 という値が返されます。

- 同じローンで、期首支払の場合に 2 年目に支払う元本は次のように計算します。

```
proc cas;
   PrincipalYear2b = CUMPRINC(0.09/12, 360, 125000, 12, 24, 1);
   print 'Principal Year 2 BOP=' PrincipalYear2b;
run;
```

計算によって$1000.73 という値が返されます。

CV 関数

変動係数を返します。

返されるデータの種類: DOUBLE

構文

CV(*expression-1*, *expression-2* <, ...*expression-n*>)

引数

expression
数値に評価される有効な式を指定します。

要件 少なくとも 2 つの引数が必要です。

データの種類 DOUBLE

例

次のステートメントは、CV 関数を示しています。

ステートメント	結果
x1=cv(5,9,3,6);	43.4782608695652
x2=cv(5,8,9,6,.);	26.082026547865
x3=cv(8,9,6,.);	19.9242421519819

DAIRY 関数

AIRY 関数の導関数を返します。

返されるデータの種類: DOUBLE

構文

DAIRY(*x*)

必須引数

x
数値の定数、変数または式を指定します。

詳細

DAIRY 関数は AIRY 関数の導関数の値を返します。

例

次のステートメントは、DAIRY 関数を示しています。

ステートメント	結果
x=dairy(2.0);	-0.05309038443365
x=dairy(-2.0);	0.61825902074169

DATDIF 関数

指定された日数計算規則に従って 2 つの日付間の差を計算した後に、その日付間の日数を返します。

返されるデータの種類: DOUBLE

構文

DATDIF(*sdate*, *edate*, *basis*)

引数

sdate
開始日を識別する SAS 日付値を指定します。

データの種類 DATE

ヒント *sdate* が月末の場合、SAS はその日付を 1 か月が 30 日の月の最終日として処理します。

edate
終了日を識別する SAS 日付値を指定します。

データの種類 DATE

ヒント *edate* が月末の場合、SAS はその日付を 1 か月が 30 日の月の最終日として処理します。

basis
日数計算基準を表す文字列を指定します。*basis* の有効な値は、'30/360'、'ACT/ACT'、'ACT/360'、'ACT/365'です。たとえば、'ACT/365'は、各月に実際のカレンダー日数を使用し、すべての年の日数に実際の日数に関わらず 365 日を使用します。

日数計算基準を表す文字列を指定します。*basis* の有効な値は次のとおりです。

'30/360'
その月または年の実際のカレンダー日数に関わらず、1 か月を 30 日、1 年を 360 日として指定します。

月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。

別名　'360'

'ACT/ACT'
日付間の実際の日数を使用します。各月にその月の実際のカレンダー日数を使用し、各年にその年の実際のカレンダー日数を使用します。

別名　*'Actual'*

'ACT/360'
各月に実際のカレンダー日数を使用し、すべての年の日数に実際の日数に関わらず 360 日を使用します。

ヒント　*ACT/360* は短期間の証券で使用されます。

'ACT/365'
各月に実際のカレンダー日数を使用し、すべての年の日数に実際の日数に関わらず 365 日を使用します。

ヒント　*ACT/365* は短期間の証券で使用されます。

データの種類　CHAR、NCHAR、NVARCHAR、VARCHAR

詳細

基本

DATDIF 関数は証券業界にとって特定の意味を持ち、その計算方法は実際の日数の数え方とは異なります。実際の日数を含む月と年を計算に使用できます。1 か月を 30 日または 1 年を 360 日として計算することもできます。標準の証券計算方法の詳細については、この関数の最後にあるリファレンスセクションを参照してください。

注: 月の日数を数える場合、DATDIF では *常に* 開始日が含まれ、終了日は除外されます。

日数計算基準(30/360)の計算方法

2 つの日付間の日数を計算するには、次の式を使用します。

$$Number of days = [(Y2 - Y1) * 360] + [(M2 - M1) * 30] + (D2 - D1)$$

引数

Y2
後の日付の年を指定します。

Y1
前の日付の年を指定します。

M2
後の日付の月を指定します。

M1
前の日付の月を指定します。

D2
後の日付の日を指定します。

D1
前の日付の日を指定します。

すべての月は 30 日のみになるため、30 日以外の日数が含まれる月は調整する必要があります。この調整は、2 つの日付間の日数を計算する前に行います。

次の規則が適用されます。

- 証券が月末ルールを使用していて、D2 と D1 の両方が 2 月(うるう年以外は 28 日、うるう年は 29 日)の最終日の場合、D2 を 30 に変更します。
- 証券が月末ルールを使用していて、D1 が 2 月の最終日の場合、D1 を 30 に変更します。
- D2 の値が 31 で D1 の値が 30 または 31 の場合、D2 を 30 に変更します。
- D1 の値が 31 の場合、D1 を 30 に変更します。

例

次の例では、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。

```
proc cas;
    sdate=19185;
    edate=21185;
    actual=datdif(sdate, edate, 'act/act');
    days360=datdif(sdate, edate, '30/360');
    print  'Actual= ' actual;
    print 'Days 360 = ' days360;
run;
```

次の行が SAS ログに書き込まれます。

```
Actual= 2000
Days 360= 1970
```

参考文献

Securities Industry Association 1994. *Standard Securities Calculation Methods - Fixed Income Securities Formulas for Analytic Measures*. 2 巻 New York, NY: Securities Industry Association.

DATE 関数

SAS 日付値として現在の日付を返します。

別名: TODAY

返されるデータの種類: DOUBLE

構文

DATE( )

引数なし

DATE 関数には引数がありません。

比較

DATE 関数は引数を取りません。返される SAS 日付値は、1960 年 1 月 1 日から現在の日付までの日数です。

例

次のステートメントは、DATE 関数を示しています。

ステートメント	結果
x=date();	18773

DATEJUL 関数

ユリウス暦の日付を SAS 日付値に変換します。

返されるデータの種類: DOUBLE

構文

DATEJUL(*julian-date*)

引数

julian-date
数値に評価され、ユリウス暦の日付を表す有効な式を指定します。ユリウス暦の日付は *yyddd* または *yyyyddd* 形式の日付です。このとき、*yy* と *yyyy* はそれぞれ年を表す 2 桁または 4 桁の整数、*ddd* は該当日がその年の何日目かを表す数です。*ddd* の値は 1 から 365(閏年の場合は 366)の間である必要があります。

データの種類 DOUBLE

詳細

SAS 日付値は、1960 年 1 月 1 日から指定日付までの日数です。DATEJUL 関数は、1960 年 1 月 1 日から *julian-date* で指定されたユリウス暦の日付までの日数を返します。

例

次のステートメントは、DATEJUL 関数を示しています。

ステートメント	結果
x=datejul(11365);	18992

DATEPART 関数

SAS 日時値から日付を抽出します。

返されるデータの種類: DOUBLE

構文

DATEPART(*datetime*)

引数

datetime
SAS 日時値を表す有効な式を指定します。

データの種類 DOUBLE

詳細

SAS 日時値は、1960 年 1 月 1 日から特定の日付内の時、分、秒までの秒数です。DATEPART 関数は、SAS 日時値の日付部分を決定し、日付を SAS 日付値として返します。これは 1960 年 1 月 1 日からの日数です。

例

次のステートメントは、SAS 日時である変数 dtvalue の値が 1652165417 である場合の DATEPART 関数を示しています。

ステートメント	結果
dp=put(datepart(dtvalue),date9.);	09MAY2012

DATETIME 関数

SAS 日時値として現在の日時を返します。

返されるデータの種類: DOUBLE

構文

DATETIME( )

比較

DATETIME 関数は引数を取りません。返される SAS 日時値は、1960 年 1 月 1 日から現在の日時までの秒数です。

例

次のステートメントは、DATETIME 関数を示しています。

ステートメント	結果
dt=datetime();	1622021468

DAY 関数

SAS 日付値として月の日を返します。

返されるデータの種類: DOUBLE

構文

DAY(*date*)

引数

date
SAS 日付値を表す有効な式を指定します。

データの種類 DOUBLE

詳細

DAY 関数は、月の日を表す 1 から 31 までの整数を生成します。

SAS 日付値は、1960 年 1 月 1 日から特定の日付までの日数です。

例

次のステートメントは、SAS 日付値である dayvalue の値が 17531 (2007 年 12 月 31 日)である場合の DAY 関数を示しています。

ステートメント	結果
dt=day(dayvalue);	dt=31

DEQUOTE 関数

単一引用符で始まる文字列から一致する単一引用符を削除し、閉じ引用符の右側にあるすべての文字を削除します。

返されるデータの種類: CHAR、NCHAR、NVARCHAR、VARCHAR

構文

DEQUOTE(*expression*)

引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

詳細

DEQUOTE 関数で返される値は、*expression* の最初の文字または最初の 2 文字に依存します。

- *expression* の最初の文字が引用符でない場合、DEQUOTE は構文エラーを返します。

- *expression* の最初の文字が単一引用符の場合、DEQUOTE 関数は結果からその単一引用符を削除します。DEQUOTE は、次に *expression* を左から右にスキャンし、他の単一引用符または二重引用符を探します。

  ペアになっている単一引用符はすべて、1 つの単一引用符に置き換えられます。

  ペアになっている二重引用符はすべて保持されます。

  二重引用符が 2 文字目の場合、DEQUOTE は結果から二重引用符を削除します。DEQUOTE は、次に、*expression* を左から右にスキャンしていきます。一致する二重引用符が見つかった場合は、二重引用符の間のテキストが返されます。閉じ二重引用符の右側、つまり *expression* の末尾までのテキストは、結果から削除されます。

  expression でペアになっていない最初の単一引用符は閉じ単一引用符であり、削除されます。

  閉じかっこが単一引用符の後に続く場合、関数は引用符で囲まれていない文字列を返します。閉じ単一引用符の右側に文字が存在する場合、関数は構文エラーになり、エラーが SAS ログに出力されます。

- *expression* が二重引用符で囲まれている場合、DEQUOTE 関数は null または欠損値を返します。

注: *expression* が引用符に囲まれた定数の場合、それらの引用符は *expression* の値には含まれません。そのため、定数を示す引用符は DEQUOTE を使用して削除する必要はありません。

例

次のステートメントは、DEQUOTE 関数を示しています。

ステートメント	結果
string=dequote(No quotation marks);	ERROR: [HY000]Parse error. Expecting ')' in statement x: char(x) string; string=dequote(no ==>quotation. (0x817ff05c)
string=dequote(No 'leading' quotation marks);	ERROR: [HY000]Parse error. Expecting ')' in statement x: char(yyy) string; string=dequote(No ==>'leading'. (0x817ff05c)
string=dequote('Single matched quotation marks are removed');	Single matched quotation marks are removed
string=dequote("Matched double quotation marks result in a null or missing value");	.
string=dequote('Paired ''single'' quotation marks are reduced to a	Paired 'single' quotation marks are reduced to a single quotation mark

ステートメント	結果
single quotation mark');	
string=dequote(' "Double quotation marks" within “single quotation marks“, with space before open quotation mark');	"Double quotation marks" within “single quotation marks“, with space before open quotation mark'
string=dequote('"Double quotation marks within single quotation marks, without space before open quotation mark');	Double quotation marks within single quotation marks, without space before open quotation mark'
string=dequote('"Text after closing double quotation mark" is removed')	Text after closing double quotation mark
string=dequote('No matching quotation mark);	ステートメントの実行は完了しません。 次の文字をサブミットして実行を完了します。 ');
string=dequote('Identifiers after close quotation mark' results in a syntax error);	ERROR: [HY000]Parse error. Expecting ')' in statement x: string=dequote('Identifiers after close quotation mark' ==>results. (0x817ff05c)

DEVIANCE 関数

確率分布に基づくデビアンスを返します。

返されるデータの種類: DOUBLE

構文

DEVIANCE(*distribution*, *variable*, *shape-parameter(s)*<, *ε*>)

引数

distribution
分布を特定する文字定数、変数または式です。有効な分布は、'BERNOULLI', 'BINOMIAL'、'GAMMA'、'IGAUSS'、'NORMAL'、'POISSON'です。

分布を特定する文字定数、変数または式です。有効な分布については、次の表を参照してください。

分布	引数
Bernoulli (p. 305)	'BERNOULLI' | 'BERN'
Binomial (p. 306)	'BINOMIAL' | 'BINO'
Gamma (p. 306)	'GAMMA'
逆ガウス(Wald) (p. 307)	'IGAUSS' | 'WALD'
Normal (p. 307)	'NORMAL' | 'GAUSSIAN'
Poisson (p. 307)	'POISSON' | 'POIS'

variable
数値定数、変数または式です。

shape-parameter(s)
分布の形状の特性を示す 1 つ以上の分布固有の数値パラメーターです。

ε
正規分布以外のすべての分布に使用される任意の小さい数値です。

詳細

Bernoulli 分布

DEVIANCE('BERNOULLI', *variable*, *p*<, *ε*>)

引数

variable
成功した場合は 1、失敗した場合は 0 の二値の数値の確率変数です。

p
ε ≤ *p* ≤ 1–*ε* の成功率の数値です。

ε
限界 *p* に使用される任意の正の数値です。間隔 0 ≤ *p* ≤ *ε* のすべての *p* 値は、*ε* で置換されます。間隔 1 – *ε* ≤ *p* ≤ 1 のすべての *p* 値は、1 – *ε* で置換されます。

DEVIANCE 関数は、成功が 1 の確率変数値として定義された成功率 *p* の Bernoulli 分布からデビアンスを返します。式は次のとおりです。

$$\text{DEVIANCE}('BERN', variable, p, \varepsilon) = \begin{cases} -2\log(1-p) & x = 0 \\ -2\log(p) & x = 1 \\ . & otherwise \end{cases}$$

二項分布

DEVIANCE('BINO', *variable*, *μ*, *n*<, *ε*>)

引数

variable
成功数が含まれる数値の確率変数です。

範囲　0 ≤ *variable* ≤ 1

μ
数値の平均パラメーターです。

範囲　*nε* ≤ *μ* ≤ *n*(1–*ε*)

n
Bernoulli 試行数の整数パラメーターです。

範囲　*n* ≥ 0

ε
限界 *μ* に使用される任意の正の数値です。間隔 0 ≤ *μ* ≤ *nε* のすべての *μ* 値は、*nε* で置換されます。間隔 *n*(1 – *ε*) ≤ *μ* ≤ *n* のすべての *μ* 値は、*n*(1 – *ε*)で置換されます。

DEVIANCE 関数は、成功率 *p*、独立した Bernoulli 試行数 *n* の二項分布からデビアンスを返します。正規分布の DEVIANCE 関数は次の式で表されます。*x* は確率変数です。

$$\text{DEVIANCE}('BINO', x, \mu, n) = \begin{cases} . & x < 0 \\ 2\left(x\log\left(\frac{x}{\mu}\right) + (n-x)\log\left(\frac{n-x}{n-\mu}\right)\right) & 0 \le x \le n \\ . & x > n \end{cases}$$

ガンマ分布

DEVIANCE('GAMMA', *variable*, *μ*<, *ε*>)

引数

variable
数値の確率変数です。

範囲　*variable* ≥ *ε*

μ
数値の平均パラメーターです。

範囲　*μ* ≥*ε*

ε
限界 *variable* および *μ* に使用される任意の正の数値です。間隔 0 ≤ *variable* ≤ *ε* のすべての *variable* 値は、*ε* で置換されます。間隔 0 ≤ *μ* ≤ *ε* のすべての *μ* 値は、*ε* で置換されます。

DEVIANCE 関数は、平均パラメーター *μ* のガンマ分布からデビアンスを返します。ガンマ分布の DEVIANCE 関数は次の式で表されます。*x* は確率変数です。

$$\text{DEVIANCE}('GAMMA', x, \mu) = \begin{cases} . & x < 0 \\ 2\left(-\log\left(\frac{x}{\mu}\right) + \frac{x-\mu}{\mu}\right) & x \geq \varepsilon, \mu \geq \varepsilon \end{cases}$$

逆ガウス(Wald)分布

DEVIANCE('IGAUSS' | 'WALD', *variable*, *μ*<, *ε*>)

引数

variable
数値の確率変数です。

範囲 *variable* ≥ *ε*

μ
数値の平均パラメーターです。

範囲 *μ* ≥*ε*

ε
限界 *variable* および *μ* に使用される任意の正の数値です。間隔 0 ≤ *variable* ≤ *ε* のすべての *variable* 値は、*ε* で置換されます。間隔 0 ≤ *μ* ≤ *ε* のすべての *μ* 値は、*ε* で置換されます。

DEVIANCE 関数は、平均パラメーター *μ* の逆ガウス分布からデビアンスを返します。逆ガウス分布の DEVIANCE 関数は次の式で表されます。*x* は確率変数です。

$$\text{DEVIANCE}('IGAUSS', x, \mu) = \begin{cases} . & x < 0 \\ \frac{(x-\mu)^2}{\mu^2 x} & x \geq \varepsilon, \mu \geq \varepsilon \end{cases}$$

正規分布

DEVIANCE('NORMAL' | 'GAUSSIAN', *variable*, *μ*)

引数

variable
数値の確率変数です。

μ
数値の平均パラメーターです。

DEVIANCE 関数は、平均パラメーター *μ* の正規分布からデビアンスを返します。正規分布の DEVIANCE 関数は次の式で表されます。*x* は確率変数です。

$$\text{DEVIANCE}('NORMAL', x, \mu) = (x-\mu)^2$$

Poisson 分布

DEVIANCE('POISSON', *variable*, *μ*<, *ε*>)

引数

variable
数値の確率変数です。

範囲 $variable \geq 0$

μ
数値の平均パラメーターです。

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、平均パラメーター *μ* の Poisson 分布からデビアンスを返します。Poisson 分布の DEVIANCE 関数は次の式で表されます。*x* は確率変数です。

$$\text{DEVIANCE}('POISSON', x, \mu) = \begin{cases} . & x < 0 \\ 2\left(x log\left(\frac{x}{\mu}\right) - (x - \mu)\right) & x \geq 0, \mu \geq \varepsilon \end{cases}$$

DHMS 関数

日、時、分、秒の値から SAS 日時値を返します。

返されるデータの種類: DOUBLE

構文

DHMS(*date*, *hour*, *minute*, *second*)

引数

date
SAS 日付値を表す有効な式を指定します。

データの種類 DOUBLE

hour
1 から 12 までの整数を表す数値式を指定します。

データの種類 DOUBLE

minute
1 から 59 までの整数を表す数値式を指定します。

データの種類 DOUBLE

second
1 から 59 までの整数を表す数値式を指定します。

データの種類 DOUBLE

詳細

DHMS 関数は、SAS 日時値を表す数値を返します。正または負の数値が返されます。

例: DHMS 関数の使用

```
proc cas;
  day=date();
  time=time();
  sasdt=dhms(day, 0, 0, time);
  format sasdt datetime.;
 print sasdt;
run;
```

次のように SAS ログに出力されます。

```
07MAY18:11:23:20
```

DIGAMMA 関数

ディガンマ関数の値を返します。

返されるデータの種類: DOUBLE

構文

DIGAMMA(*expression*)

引数

expression
数値に評価される有効な式を指定します。

制限事項 ゼロおよび負の整数は無効です。

データの種類 DOUBLE

詳細

DIGAMMA 関数は、次の式で求められる比率を返します。

$$\Psi(x) = \Gamma'(x)/\Gamma(x)$$

$\Gamma(.)$および$\Gamma'(.)$は、それぞれガンマ関数とその導関数を示します。*expression*>0 の場合、DIGAMMA 関数は lgamma 関数の導関数です。

例

次のステートメントは、DIGAMMA 関数を示しています。

ステートメント	結果
x=digamma(1.0);	-0.57721566490153

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。

返されるデータの種類: DOUBLE

構文

DIVIDE(*x*, *y*)

引数

x
数値に評価される有効な式を指定します。

データの種類 DOUBLE

y
数値に評価される有効な式を指定します。

データの種類 DOUBLE

詳細

DIVIDE 関数は、2 つの数値を除算して ODS 規則に適合する結果を返します。この関数は、ODS 出力の特殊欠損値を処理します。次のリストに、特定の特殊欠損値が ODS で解釈される方法を示します。

- .I は無限大として解釈
- .M は負の無限大として解釈
- ._は空白として解釈

次の表に、*x* および *y* の値に基づいて DIVIDE 関数によって返される値を示します。

図 4.1 DIVIDE 関数によって返される値

		x						
		positive	zero	negative	.I	.M	.__	other
y	positive	x/y or .I	0	x/y or .M	.I	.M	.__	x
	zero	.I	.	.M	.I	.M	.__	x
	negative	x/y or .M	0	x/y or .I	.M	.I	.__	x
	.I	0	0	0	.	.	.__	x
	.M	0	0	0	.	.	.__	x
	.__	.__	.__	.__	.__	.__	.__	.__
	other	y	y	y	y	y	.__	x

注: DIVIDE 関数は、欠損値、ゼロ除算またはオーバーフローに関するメモを SAS ログに書き込むことはありません。

例

次の例では、DIVIDE 関数を使用した結果を示します。

```
proc cas;
  a=divide(1,0);
  print "a=" a+3 "(infinity)";
  b=divide(2, .I);
  print "b=" b+3;
  c=divide(.I, -1);
  print "c=" c "(minus inifinity)";
  d=divide(constant('big'), constant('small'));
  print "d=" d+3 "(infinity because of overflow)";
run;
```

次の行が SAS ログに書き込まれます。

```
a=. (infinity)
b=3
c=M (minus inifinity)
d=. (infinity because of overflow)
```

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。

返されるデータの種類: DOUBLE

構文

DURP(*A*, *c*, *n*, *K*, k_0, *y*)

引数

A
額面価格を指定します。

範囲 $A > 0$

データの種類 DOUBLE

c
期間当たりの名目クーポンレートを分数で指定します。

範囲 $0 \le c < 1$

データの種類 DOUBLE

n
期間当たりのクーポン数を指定します。

範囲 $n > 0$ で整数

データの種類 DOUBLE

K
クーポンの残数を指定します。

範囲 $K > 0$ で整数

データの種類 DOUBLE

k_0
現在の日付から最初のクーポン日までの時間を指定します。期間数で表します。

範囲 $0 < k_0 \le 1/n$

データの種類 DOUBLE

y
期間当たりの名目有効満期利回りを指定します。分数で表します。

範囲 $y > 0$

データの種類 DOUBLE

詳細

DURP 関数は、次の式から値を返します。

$$D = \frac{1}{n} \frac{\sum_{k=1}^{K} t_k \frac{c(k)}{\left(1+\frac{y}{n}\right)^{t_k}}}{P\left(1+\frac{y}{n}\right)}$$

前述の式には次の関係が適用されます。

- $t_k = nk_0 + k - 1$
- $c(k) = \frac{c}{n}A \quad for\ k = 1, \ldots, K-1$
- $c(K) = \left(1+\frac{c}{n}\right)A$

前述の式には次の関係が適用されます。

$$P = \sum_{k=1}^{K} \frac{c(k)}{\left(1+\frac{y}{n}\right)^{t_k}}$$

例: DURP 関数の使用

```
proc cas;
  d=durp(1000, 1/100, 4, 14, .33/2, .10);
  print d;
run;
```

SAS は次の出力をログに書き込みます。

```
d=3.2649588109
```

EFFRATE 関数

有効な年利を返します。

返されるデータの種類: DOUBLE

構文

EFFRATE(*compounding-interval*, *rate*)

引数

compounding-interval
SAS 間隔です。この値は、*rate* を複利計算する頻度を表します。

データの種類 CHAR

rate
数値です。*rate* は、各利息計算期間で複利計算する名目年利(パーセント)です。

データの種類 DOUBLE

詳細

EFFRATE 関数は、有効な年利を返します。この関数は、名目年利に対応する有効な年利を計算します。

EFFRATE 関数には次の詳細が適用されます。

- rate の値は–99 以上にする必要があります。
- 名目金利と利息計算期間を考えるに当たって、*compounding-interval* が 'CONTINUOUS'の場合、EFFRATE によって返される値は $e^{rate/100}–1$ です。

  compounding-interval が'CONTINUOUS'以外で、1 年に *m* 回の利息計算期間が含まれる場合、EFFRATE によって返される値は$(1+[rate/100\ m])^{m–1}$ です。
- *compounding-interval* で有効な値は次のとおりです。
  - 'CONTINUOUS'
  - 'DAY'
  - 'SEMIMONTH'
  - 'MONTH'
  - 'QUARTER'
  - 'SEMIYEAR'
  - 'YEAR'
- 間隔が'DAY'の場合、*m*=365 です。

例

次の例では、有効な利率の計算方法を示します。

- 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);

ERF 関数

(正規)誤差関数の値を返します。

返されるデータの種類: DOUBLE

構文

ERF(*expression*)

引数

expression
数値に評価される有効な式を指定します。

データの種類 DOUBLE

詳細

ERF 関数は、次の式で求められる積分を返します。

$$\mathrm{ERF}(x) = \frac{2}{\sqrt{\pi}} \int_0^x \varepsilon^{-z^2} dz$$

ERF 関数を使用して、平均値が 0 で標準偏差が 1 の正規分布に従う確率変数の値が X よりも小さい確率(p)を求めることができます。たとえば、次のステートメントで求められる数量は PROBNORM(X)と同等です。

p=.5+.5*erf(x/sqrt(2));

例

次のステートメントは、ERF 関数を示しています。

ステートメント	結果
y=erf(1.0);	0.842701
y=erf(-1.0);	-0.8427

ERFC 関数

相補(正規)誤差関数の値を返します。

返されるデータの種類: DOUBLE

構文

ERFC(*expression*)

引数

expression
数値に評価される有効な式を指定します。

データの種類 DOUBLE

詳細

ERFC 関数は、ERF 関数(1 − ERF(*argument*))の補数を返します。

例

次のステートメントは、ERFC 関数を示しています。

ステートメント	結果
x=erfc(1.0);	0.157299
x=erfc(-1.0);	1.842701

EXP 関数

指定された累乗の e 定数の値を返します。

返されるデータの種類: DOUBLE

構文

EXP(*expression*)

引数

expression
数値に評価される有効な式を指定します。

データの種類 DOUBLE

詳細

EXP 関数は、定数 **e** (2.71828 で近似的に与えられる)に対する指定した引数の累乗を返します。この結果は、コンピューターの倍精度小数値の最大値によって制限されます。

例

次のステートメントは、EXP 関数を示しています。

ステートメント	結果
a=exp(1.0);	2.71828182845904
a=exp(0);	1

FACT 関数

階乗を計算します。

返されるデータの種類: DOUBLE

構文

FACT(*expression*)

引数

expression
数値に評価される有効な式を指定します。

データの種類 DOUBLE

詳細

FACT 関数の数学的表現は、次の式によって得られます。

$$FACT(n) = n!$$

この式では、$n \geq 0$ です。

式による計算ができない場合は欠損値が返されます。やや大きな値の場合、FACT 関数を計算できないことがあります。

例

次のステートメントは、FACT 関数を示しています。

ステートメント	結果
x=fact(5);	120

FIND 関数

文字列内の特定の部分文字列を検索します。

返されるデータの種類: CHAR

構文

FIND(*string*, *substring*<, *modifier(s)*><, *startpos*>)

FIND(*string*, *substring*<, *startpos*><, *modifier(s)*>)

引数

string
文字列を評価するか、文字列に強制変換できて、部分文字列の検索対象になる文字定数、変数、または式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

ヒント 文字のリテラル文字列を引用符で囲みます。

substring
文字列を評価するか、文字列に強制変換できる文字定数、変数、または式であり、*string* で検索する文字の部分文字列を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

ヒント 文字のリテラル文字列を引用符で囲みます。

modifier(s)
1 つ以上の修飾子を文字定数、変数または式で指定します。修飾子として使用できる文字(大文字または小文字)は、I (大文字と小文字の区別を無視する)または T (末尾の空白を取り除く)です。両方(またはすべて)の文字引数ではなく一方のみから末尾の空白を削除する場合は、FIND 関数に T 修飾子を使用するかわりに、TRIM 関数を使用します。

1 つ以上の修飾子を文字定数、変数または式で指定します。修飾子として使用できる文字(大文字または小文字)は次のとおりです。

i or I
検索時に大文字と小文字を区別しません。この修飾子を指定しないと、FIND は *substring* の文字の大文字と小文字に一致する部分文字列のみを検索します。

t or T
string および *substring* から末尾の空白を取り除きます。

注: 両方(またはすべて)の文字引数ではなく一方のみから末尾の空白を削除する場合は、FIND 関数に T 修飾子を使用するかわりに、TRIM 関数を使用します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

ヒント *modifier* が定数の場合、引用符で囲みます。一組の引用符で複数の定数を指定します。*modifier* を変数または式として表すこともできます。

startpos
数値定数、変数、または式であり、検索を開始する位置と検索の方向を指定する整数です。

データの種類 DOUBLE

詳細

FIND 関数は *string* で最初に出現する指定の *substring* を検索し、その部分文字列の位置を返します。*string* 内に部分文字列が見つからない場合、FIND は値 0 を返します。

sstartpos を指定しないと、FIND は *string* の最初から検索を開始し、左から右に *string* を検索します。*startpos* を指定すると、*startpos* の絶対値で検索の開始位置が決定されます。検索方向は、*startpos* の符号で決まります。

startpos の値	アクション
0 より大きい	*startpos* の位置から検索を開始し、右方向に検索します。*startpos* が *string* の長さよりも大きい場合、FIND は値 0 を返します。
0 より小さい	*startpos* の位置から検索を開始し、左方向に検索します。*startpos* が *string* の長さよりも大きい場合、検索は *string* の最後から開始します。
0	値 0 を返します。

比較

- FIND 関数は文字列の文字の部分文字列を検索しますが、FINDC 関数は文字列の個々の文字を検索します。
- FIND 関数と INDEX 関数はどちらも文字列の文字の部分文字列を検索します。ただし、INDEX 関数には *modifiers* 引数も *startpos* 引数もありません。

例

次のステートメントは、FIND 関数を示しています。

ステートメント	結果
whereisshe=find('She sells seashells? Yes, she does.','she '); print whereisshe;	27

ステートメント	結果
variable1='She sells seashells? Yes, she does.'; variable2='she '; variable3='i'; whereisshe_i=find(variable1,variable2,variable3); print 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);
print whereisshe_t;
	14

xyz='She sells seashells? Yes, she does.';
startposvar=22;
whereisshe_22=find(xyz,'she',startposvar);
print whereisshe_22;
	27

xyz='She sells seashells? Yes, she does.';
startposexp=1-23;
whereisShe_ineg22=find(xyz,'She','i',startposexp);
print whereisShe_ineg22;
	14

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。

返されるデータの種類: DOUBLE

構文

FINDC(*string*<, *charlist*>)

FINDC(*string*, *charlist*<, *modifier*>)

FINDC(*string*, *charlist*, *modifier*<, *startpos*>)

FINDC(*string*, *charlist*<, *startpos*><, *modifier*>)

引数

string
文字列を評価するか、文字列に強制変換できる文字定数、変数、または式であり、検索する文字列を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

ヒント　文字のリテラル文字列を引用符で囲みます。

charlist
文字リストを初期化する定数、変数または文字式です。*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。

データの種類　CHAR、NCHAR、NVARCHAR、VARCHAR

ヒント　文字のリテラル文字列を引用符で囲みます。

modifier
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。A (アルファベット文字をリストに追加)、C (制御文字を追加)、D (数字を追加)の大文字と小文字を修飾子として使用できます。他の修飾子も使用することができます。

文字定数、変数または式です。これらの各文字で FINDC 関数のアクションを変更します。修飾子として使用できる文字(大文字または小文字)は次のとおりです。

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	*startpos* 引数の符号に関係なく、左から右へではなく右から左へ検索します。
c 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	アンダースコアと英字(つまり、VALIDVARNAME=V7 を使用して SAS 変数名を開始できる文字)を文字のリストに追加します。
g 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	検索時に大文字と小文字を区別しません。
k または K	文字のリストにない文字を検索します。この修飾子を指定しないと、FINDC は文字のリストにある文字を検索します。V 修飾子と K 修飾子は同じ機能を実行します。
l 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
: *charlist* 引数および *modifier* 引数を 1 回だけ処理します。FINDC 関数の呼び出し時毎には処理されません。DS2 (WHERE 句を除く)で O 修飾子を使用すると、*charlist* 引数と *modifier* 引数が変化しないループで FINDC を呼び出すとき、FINDC がより速く実行されます。

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* 引数と *charlist* 引数から末尾の空白をトリムします。両方(またはすべて)の文字引数ではなく一方のみから末尾の空白を削除する場合は、FINDC 関数に T 修飾子を使用するかわりに、TRIM 関数を使用することに注意してください。

u または U
: 大文字を文字リストに追加します。

v または V
: 文字のリストにないすべての文字が区切り文字として扱われるようにします。V が指定されていない場合、文字のリストにあるすべての文字が区切り文字として扱われます。V 修飾子と K 修飾子は同じ機能を実行します。

w 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。

データの種類
: CHAR、NCHAR、NVARCHAR、VARCHAR

ヒント
: *modifier* が定数の場合、引用符で囲みます。一組の引用符で複数の定数を指定します。*modifier* を変数または式として表すこともできます。

startpos
: オプションの数値定数、変数、または式であり、検索を開始する位置と検索する方向を指定する整数です。

データの種類
: DOUBLE

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。数値引数はヌルにできません。

startpos を指定しないと、FINDC は、B 修飾子を使用している場合は文字列の最後から、B 修飾子を使用していない場合は文字列の最初から検索を開始します。

startpos を指定すると、*startpos* の絶対値で検索の開始位置が指定されます。B 修飾子を使用する場合、検索は常に右から左に進みます。B 修飾子を使用しない場合、*startpos* の符号で検索方向を指定します。次の表に、検索方向の要約を示します。

startpos の値	アクション
0 より大きい	検索は *startpos* の位置から開始され、右に進みます。*startpos* が文字列の長さよりも大きい場合、FINDC は値 0 を返します。
0 より小さい	検索は–*startpos* の位置から開始され、左に進みます。*startpos* が文字列の長さの負の値よりも小さい場合、検索は文字列の末尾から開始されます。
0	値 0 を返します。

比較

- FINDC 関数は文字列の個々の文字を検索しますが、FIND 関数は文字列の文字の部分文字列を検索します。
- FINDC 関数と INDEXC 関数はどちらも文字列の個々の文字を検索します。ただし、INDEXC 関数には *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。

例: 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。

```
proc cas;
    j=1;
    do until(j=0);
      j = findc('Hi, ho!','hi',j+1);
      if j= 0 then print 'The End';
      else do;
        c = substr('Hi, ho!', j, 1);
        print "j=" j;
        print "c=" c;
      end;
    end;
run;
```

SAS は次の出力をログに書き込みます。

```
j=2 c=i
j=5 c=h
The End
```

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。

返されるデータの種類: DOUBLE

構文

FINDW(*string*, *word*<, *chars*>)

FINDW(*string*, *word*, *chars*, *modifier(s)*<, *startpos*>)

FINDW(*string*, *word*, *chars*, *startpos*<, *modifier(s)*>)

FINDW(*string*, *word*, *startpos*<, *chars*<, *modifier(s)*>>)

引数

string
文字列を評価するか、文字列に強制変換できる文字定数、変数、または式であり、検索する文字列を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

ヒント 文字のリテラル文字列を引用符で囲みます。

word
文字列を評価するか、文字列に強制変換できる文字定数、変数、または式であり、検索する単語を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

ヒント 文字のリテラル文字列を引用符で囲みます。

chars
文字のリストを初期化する文字定数、変数または式です(省略可能)。リスト中の文字は単語を区切るための区切り文字です(ただし *modifier* 引数に K 修飾子を指定しない場合)。K 修飾子を指定すると、このリストにないすべての 文字が区切り文字となります。他の修飾子を使用して、このリストに文字をさらに追加できます。

文字のリストを初期化する文字定数、変数または式です(省略可能)。

リスト中の文字は単語を区切るための区切り文字です(ただし *modifier* 引数に K 修飾子を指定しない場合)。K 修飾子を指定すると、このリストにないすべての文

字が区切り文字となります。他の修飾子を使用して、このリストに文字をさらに追加できます。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

ヒント 文字のリテラル文字列を引用符で囲みます。

startpos
オプションの数値定数、変数、または式であり、検索を開始する位置と検索する方向を指定する整数です。

データの種類 DOUBLE

'*modifier(s)*'
文字定数、変数または式を指定します。これらの空白以外の各文字で FINDW 関数のアクションを変更します。A (アルファベット文字をリストに追加)、C (制御文字を追加)、D (数字を追加)の大文字と小文字を修飾子として使用できます。他の修飾子も使用することができます。

文字定数、変数または式を指定します。これらの空白以外の各文字で FINDW 関数のアクションを変更します。

次の文字を修飾子として使用できます。

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	*startpos* 引数の符号に関係なく、左から右へではなく右から左へ検索します。
c 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。
e または E	指定した単語の文字列の文字位置を決定するかわりに、指定した単語が検出されるまでにスキャンした単語を数えます。単語の断片はカウントされません。
f または F	アンダースコアと英字(つまり、VALIDVARNAME=V7 を使用して SAS 変数名を開始できる文字)を文字のリストに追加します。
g 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	文字のリストにないすべての文字が区切り文字として扱われるようにします。K を指定しない場合、文字のリストに含まれているすべての文字が区切り文字として扱われます。K 修飾子と V 修飾子は同じ機能を実行します。

l 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。

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　FINDW 関数が呼び出されるたびに、*chars* 引数と *modifier* 引数を処理するのではなく、一度のみ処理します。DS2 (WHERE 句を除く)で O 修飾子を使用すると、*chars* 引数と *modifier* 引数が変化しないループで FINDW を呼び出すと、FINDW がより速く実行されます。

p 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　*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 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、*word* 引数、*chars* 引数の末尾の空白をトリムします。

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　文字のリストにないすべての文字が区切り文字として扱われるようにします。V が指定されていない場合、文字のリストにあるすべての文字が区切り文字として扱われます。V 修飾子と K 修飾子は同じ機能を実行します。

w 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。

データの種類　CHAR、NCHAR、NVARCHAR、VARCHAR

ヒント　*modifier* 引数を使用する場合は、*chars* 引数の後に配置する必要があります。

詳細

"区切り文字"の定義

"区切り文字"は、単語を区切るために使用されるすべての文字を示します。区切り文字は、*chars* 引数、*modifier* 引数、またはその両方を使って指定できます。Q 修飾子を指定する場合、引用符で囲まれる部分文字列内の文字は区切り文字として扱われません。

"単語"の定義

"単語"は、次の両方の特徴を持つ部分文字列を示します。

- 左側にある区切り文字で区切られる部分文字列または文字列の最初の部分文字列
- 右側にある区切り文字で区切られる部分文字列または文字列の最後の部分文字列

注: 単語には区切り文字を含めることができます。その場合、FINDW 関数は、単語に区切り文字が含まれない SCAN 関数とは異なります。

文字列の検索

FINDW 関数は、指定した単語と単語の定義の両方を満たす部分文字列を検出できない場合、値 0 を返します。

指定した単語と単語の定義の両方を満たす部分文字列を FINDW 関数が検出した場合に返される値は、E 修飾子が指定されているどうかによって異なります。

- E 修飾子を指定すると、FINDW は、指定した単語の検索中にスキャンした完全な単語の数を返します。*startpos* が単語の中心の位置を指定した場合、その単語はカウントされません。
- E 修飾子を指定しないと、FINDW は検出された部分文字列の文字位置を返します。

startpos 引数を指定した場合、*startpos* の絶対値で検索を開始する位置が指定されます。*startpos* の符号で検索する方向が指定されます。

startpos の値	アクション
0 より大きい	検索は *startpos* の位置から開始され、右に進みます。*startpos* が文字列の長さよりも大きい場合、FINDW は 0 を返します。
0 より小さい	検索は–*startpos* の位置から開始され、左に進みます。*startpos* が文字列の長さの負の値よりも小さい場合、検索は文字列の末尾から開始されます。
0	FINDW は値 0 を返します。

startpos 引数または B 修飾子を指定しない場合、FINDW は文字列の先頭から左から右に検索します。B 修飾子を指定しても、*startpos* 引数を使用しない場合、FINDW は文字列の末尾から右から左に検索します。

ASCII 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、コンピューターで使用している文字(ASCII または EBCDIC)によって異なります。

- お使いのコンピューターが ASCII 文字を使用している場合、デフォルトの区切り文字は次のようになります。

  空白 ! $ % & ( ) * + , - ./ ; < ^ |

  ^文字のない ASCII 環境の場合、FINDW 関数はかわりに~文字を使用します。
- お使いのコンピューターが EBCDIC 文字を使用している場合、デフォルトの区切り文字は次のようになります。

  空白 ! $ % & ( ) * + , - ./ ; < ¬ | ¢

null 引数の使用

FINDW 関数では、文字引数を NULL に指定できます。ヌル引数は長さがゼロの文字列として扱われます。数値引数はヌルにできません。

例: 文字列の検索と、chars 引数と startpos 引数の使用

次の例には、“rain”という単語のオカレンスが 2 つ含まれています。FINDW は、25 番目の位置から検索を開始するため、2 番目の単語のみが検出されます。*chars* 引数は区切り文字としてスペースを 指定しています。

```
proc cas;
    result = findw('At least 2.5 meters of rain falls in a rain forest.',
      'rain', ' ', 25);
    print "result=" result;
  end;
run;
```

SAS は次の出力をログに書き込みます。

```
result=40
```

FLOOR 関数

数値式以下の最大の整数を返します。

返されるデータの種類:	DECIMAL、DOUBLE、NUMERIC

構文

FLOOR(*expression*)

引数

expression
数値に評価される有効な式を指定します。

データの種類 DECIMAL、DOUBLE、NUMERIC

詳細

expression が整数の 1E-12 以内の場合、関数はその整数を返します。結果が DOUBLE の範囲に収まらない数値の場合、FLOOR 関数は失敗します。

引数が DECIMAL の場合、結果は DECIMAL になります。それ以外の場合、引数は (まだ変換されていなければ) DOUBLE に変換され、結果は DOUBLE になります。

比較

FLOOR 関数は、結果が整数の 1E-12 内にある場合はその整数を返すように、結果をファジー処理します。FLOORZ 関数はゼロファジーを使用します。そのため、FLOORZ 関数では予期しない結果になる可能性があります。

例

次のステートメントは、FLOOR 関数を示しています。

ステートメント	結果
floor(1.95);	1

FMTINFO 関数

SAS 出力形式または入力形式の情報を返します。

制限事項: この関数は、SAS で提供されている出力形式の情報を返します。FORMAT プロシジャで作成されるユーザー定義出力形式には使用できません。

構文

FMTINFO('*format-name*', '*information-type*');

引数

'*format-name*'
SAS 出力形式または入力形式の名前を指定します。

要件 *format-name* は、単一引用符で囲む必要があります。

information-type
返される入力形式の種類を指定します。*format-information* には、'DESC' (簡単な説明)、'MINW' (最小幅)、'MAXW' (最大幅)のいずれかの値を指定できます。追加の情報の種類が使用可能です。

返される入力形式の種類を指定します。*format-information* には、次のいずれかの値を指定できます。

'CAT'
関数カテゴリを返します。

'TYPE'
format-name が出力形式、入力形式、またはその両方のいずれであるかを返します。

'DESC'
出力形式または入力形式の簡単な説明を返します。

'MIND'
出力形式または入力形式の小数点以下の最小桁数を返します。

'MAXD'
出力形式または入力形式の小数点以下の最大桁数を返します。

'DEFD'
出力形式または入力形式の小数点以下のデフォルト桁数を返します。

'MINW'
出力形式または入力形式の最小の幅の値を返します。

'MAXW'
出力形式または入力形式の最大の幅の値を返します。

'DEFW'
出力形式または入力形式のデフォルトの幅の値を返します。

制限事項 *information-type* 引数は 1 つだけ指定できます。

要件 *information-type* は、単一引用符で囲む必要があります。

詳細

FMTINFO 関数は、出力形式または入力形式の情報を返します。出力形式または入力形式のカテゴリ、言語要素の種類、言語要素の説明、小数点以下の値の最小/最大/デフォルト値、幅の値の最小/最大/デフォルト値を返すことができます。

FMTINFO 関数で複数の引数を指定することはできません。

FMTINFO 関数は、数値引数 MIND、MAXD、DEFD、MINW、MAXW、DEFW を含めて、すべてのデータ値に対して文字列を返します。

例

次の例では、変数 *a* の文字列を返します。

```
proc cas;
 a=fmtinfo('best', 'cat');
 print a;
run;
```

次の行が SAS ログに書き込まれます。

```
num
```

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。

返されるデータの種類: DOUBLE

構文

FLOORZ(*expression*)

引数

expression
数値に評価される有効な式を指定します。

データの種類 DOUBLE

比較

FLOOR 関数とは異なり、FLOORZ 関数はゼロファジーを使用します。引数が整数の 1E-12 内にある場合、FLOOR 関数はその整数に等しくなるように結果をファジー処理します。FLOORZ 関数は結果をファジー処理しません。そのため、FLOORZ 関数では予期しない結果になる可能性があります。

例

次のステートメントは、FLOORZ 関数を示しています。

ステートメント	結果
var1=2.1; a=floorz(var1);	2
b=floorz(-2.4);	-3
c=floorz(-1.6);	-2

FNONCT 関数

F 分布の非心度パラメーターの値を返します。

返されるデータの種類: DOUBLE

構文

FNONCT(*x*, *ndf*, *ddf*, *probability*)

必須引数

x
数値の確率変数です。

範囲 $x \geq 0$

データの種類 DOUBLE

ndf
数値の分子の自由度パラメーターです。

範囲　*ndf* > 0

データの種類　DOUBLE

ddf
数値の分母の自由度パラメーターです。

範囲　*ddf* > 0

データの種類　DOUBLE

probability
確率です。

範囲　0 < *probability* < 1

データの種類　DOUBLE

詳細

FNONCT 関数は、非心度 *F* 分布(パラメーターは *x*、*ndf*、*ddf*、*nc*)から負でない非心度パラメーターを返します。*probability* が心度 *F* 分布(パラメーターは *x*、*ndf* および *ddf*)からの確率よりも大きい場合、この問題の平方根は存在しません。この場合、欠損値が返されます。式の負でない平方根 *nc* を検索するには、ニュートン型のアルゴリズムを使用します。

$$P_f(x|ndf, ddf, nc) - prob = 0$$

前述の式には次の関係が適用されます。

$$P_f(x|ndf, ddf, nc) = \varepsilon^{\frac{-nc}{2}} \sum_{j=0}^{\infty} \frac{\left(\frac{nc}{2}\right)^j}{j!} I_{\frac{(ndf)x}{ddf+(ndf)x}}\left(\frac{ddf}{2}+j, \frac{ddf}{2}\right)$$

この式では、$I(...)$は、次の式によって得られるベータ分布の確率です。

$$I_x(a,b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)} \int_0^x t^{a-1}(1-t)^{b-1} dt$$

アルゴリズムで固定小数点を収束できない場合、欠損値が返されます。

例

```
proc cas;
   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);
```

```
    end;
  end;
   columns={'x', 'df', 'ddf', 'nc', 'prob', 'ncc'};
   coltypes={'int64' 'int64','int64', 'int64'};
   row1={x, df, ddf, nc, prob, ncc};
   row2={x, df, ddf, nc, prob, ncc};
   row3={x, df, ddf, nc, prob, ncc};
   row4={x, df, ddf, nc, prob, ncc};
   row5={x, df, ddf, nc, prob, ncc};
   mytable=newtable('mytable',columns, coltypes, row1, row2, row3, row4, row5);
   print mytable;
run;
```

アウトプット 4.4　FNONCT 関数からの出力

mytable: Results

x	df	ddf	nc	prob	ncc
2	4	5	3	0	3
2	4	5	3	0	3
2	4	5	3	0	3
2	4	5	3	0	3
2	4	5	3	0	3

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: CAS
切り捨て

返されるデータの種類: DOUBLE

構文

FUZZ(*expression*)

引数

expression
数値に評価される有効な式を指定します。

データの種類　DOUBLE

詳細

式が整数の 1E-12 以内の場合(整数と引数間の絶対差が 1E-12 よりも小さい場合)、FUZZ 関数は最も近い整数値を返します。それ以外の場合は、その式を返します。

例

次のステートメントは、FUZZ 関数を示しています。

ステートメント	結果
var1=5.9999999999999; x=print(fuzz(var1),16.14);	6
x=print(fuzz(5.99999999), 16.14);	5.99999999

GAMINV 関数

ガンマ分布から分位点を返します。

返されるデータの種類: DOUBLE

構文

GAMINV(*p*, *a*)

引数

p

数値の確率に評価される有効な式を指定します。

範囲 $0 \le p < 1$

データの種類 DOUBLE

a

数値の形状パラメーターに評価される有効な式を指定します。

範囲 $a > 0$

データの種類 DOUBLE

詳細

GAMINV 関数は、ガンマ分布(形状パラメーターは *a*)の *p*分位点を返します。ガンマ分布の行が返される分位点以下になる確率は *p* です。

注: GAMINV は、PROBGAM 関数の逆数です。

例

次のステートメントは、GAMINV 関数を示しています。

ステートメント	結果
q1=gaminv(0.5, 9);	8.668951
q2=gaminv(0.1, 2.1);	0.584193

GAMMA 関数

ガンマ関数の値を返します。

返されるデータの種類: DOUBLE

構文

GAMMA(*expression*)

引数

expression
数値に評価される有効な式を指定します。

制限事項 正でない整数は無効です。

データの種類 DOUBLE

詳細

GAMMA 関数は、次の式で求められる積分を返します。

$$\mathrm{GAMMA}(x) = \int_0^{\infty} t^{x-1} e^{-t}\, dt.$$

正の整数の場合、GAMMA(x)は(x − 1)!になります。この関数は、一般的に$\Gamma(x)$で表されます。

例

次のステートメントは、GAMMA 関数を示しています。

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: CAS
財務

返されるデータの種類: DOUBLE

構文

GARKHCLPRC(*E*, *t*, *S*, R_d, R_f, *sigma*)

引数

E
権利行使価格を指定する正の非欠損値。

要件 *E* および *S* は同じ単位で指定します。

データの種類 DOUBLE

t
満期までの時間を指定する非欠損値です。

データの種類 DOUBLE

S
通貨の現物価格を指定する非欠損値の正の値です。

要件 *S* および *E* は同じ単位で指定します。

データの種類 DOUBLE

R_d
期間 *t* の国内の安全利率を指定する非欠損値の正の分数です。

要件 単位 *t* と同じ期間の R_d の値を指定します。

データの種類 DOUBLE

R_f
期間 *t* の海外の安全利率を指定する非欠損値の正の分数です。

要件 単位 *t* と同じ期間の R_f の値を指定します。

データの種類 DOUBLE

sigma
為替相場のボラティリティを 指定する非欠損値の正の分数です。

要件 単位 *t* と同じ期間の *sigma* の値を指定します。

データの種類 DOUBLE

詳細

GARKHCLPRC 関数は、Garman-Kohlhagen モデルに基づいて、株式のヨーロピアンオプションのコール価格を計算します。この関数は次の関係に基づきます。

$$\text{CALL} = SN(d_1)\left(\varepsilon^{-R_f t}\right) - EN(d_2)\left(\varepsilon^{-R_d t}\right)$$

引数

S
通貨の現物価格を指定します。

N
累積正規密度関数を指定します。

E
オプションの権利行使価格を指定します。

t
有効期限までの時間を指定します。

R_d
期間 *t* の国内の安全利率を指定します。

R_f
　期間 t の海外の安全利率を指定します。

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

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

前述の式には次の引数が適用されます。

σ
　原資産のボラティリティを 指定します。

σ^2
　利益率の分散を指定します。

t=0 となる特別な場合には、次の式が真です。

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

比較

GARKHCLPRC 関数は、Garman-Kohlhagen モデルに基づいて、株式のヨーロピアンオプションのコール価格を計算します。GARKHPTPRC 関数は、Garman-Kohlhagen モデルに基づいて、株式のヨーロピアンオプションのプット価格を計算します。これらの関数はスカラー値を返します。

例

次のステートメントは、GARKHCLPRC 関数を示しています。

ステートメント	結果
	----+----1----+-—-2--
a=garkhclprc(40, .5, 38, .06, .04, .2);	1.44942510595479
c=garkhclprc(19, .25, 20, .05, .03, .09);	1.1304209447635

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。

返されるデータの種類: DOUBLE

構文

GARKHPTPRC(*E*, *t*, *S*, R_d, R_f, *sigma*)

引数

E
権利行使価格を指定する正の非欠損値。

要件 *E* および *S* は同じ単位で指定します。

データの種類 DOUBLE

t
満期までの時間を年単位で指定する非欠損値。

データの種類 DOUBLE

S
通貨の現物価格を指定する非欠損値の正の値です。

要件 *S* および *E* は同じ単位で指定します。

データの種類 DOUBLE

R_d
期間 *t* の国内の安全利率を指定する非欠損値の正の分数です。

要件 単位 *t* と同じ期間の R_d の値を指定します。

データの種類 DOUBLE

R_f
期間 *t* の海外の安全利率を指定する非欠損値の正の分数です。

要件 単位 *t* と同じ期間の R_t の値を指定します。

データの種類 DOUBLE

sigma
為替相場のボラティリティを 指定する非欠損値の正の分数です。

データの種類 DOUBLE

詳細

GARKHPTPRC 関数は、Garman-Kohlhagen モデルに基づいて、株式のヨーロピアンオプションのプット価格を計算します。この関数は次の関係に基づきます。

$$\text{PUT} = \text{CALL} - S\left(\varepsilon^{-R_f t}\right) + E\left(\varepsilon^{-R_d t}\right)$$

引数

S
通貨の現物価格を指定します。

E
オプションの権利行使価格を指定します。

t
有効期限までの時間を年単位で指定します。

R_d
期間 *t* の国内の安全利率を指定します。

R_f
期間 *t* の海外の安全利率を指定します。

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

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

前述の式には次の引数が適用されます。

σ
原資産のボラティリティを 指定します。

σ^2
利益率の分散を指定します。

t=0 となる特別な場合には、次の式が真です。

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

比較

GARKHPTPRC 関数は、Garman-Kohlhagen モデルに基づいて、株式のヨーロピアンオプションのプット価格を計算します。GARKHCLPRC 関数は、Garman-Kohlhagen モデルに基づいて、株式のヨーロピアンオプションのコール価格を計算します。これらの関数はスカラー値を返します。

例

次のステートメントは、GARKHPTPRC 関数を示しています。

ステートメント	結果
	----+----1----+----2--
a=garkhptprc(50, .7, 55, .05, .04, .2);	1.4050880944848
b=garkhptprc(32, .3, 33, .05, .03, .3);	1.56473205137371

GCD 関数

整数の集合に対する最大公約数を返します。

返されるデータの種類: DOUBLE

構文

GCD(*expression-1*, *expression-2* <, ...*expression-n*>)

引数

expression
数値に評価される有効な式を指定します。

要件 少なくとも 2 つの引数が必要です。

データの種類 DOUBLE

詳細

GCD (最大公約数)関数は、1 つ以上の整数の最大公約数を返します。たとえば、30 と 42 の最大公約数は 6 になります。最大公約数は、最高共通因子とも呼ばれます。

例

次のステートメントは、GCD 関数を示しています。

ステートメント	結果
x=gcd(5,15)	5
x=gcd(36,45)	9

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

返されるデータの種類: DOUBLE

構文

GEODIST(*latitude-1*, *longitude-1*, *latitude-2*, *longitude-2* <,*options*>)

引数

latitude
赤道の南北にある特定の場所の座標を指定する数値定数、変数または式です。赤道の北にある座標は正の値、赤道の南にある座標は負の値になります。

制限事項 値を度数で表す場合、90 から–90 までの値にする必要があります。値をラジアンで表す場合、pi/2 から–pi/2 までの値にする必要があります。

データの種類 DOUBLE

longitude
英国のグリニッジを通過するグリニッジ子午線の東西にある特定の場所の座標を指定する数値定数、変数または式です。グリニッジ子午線の東にある座標は正の値、グリニッジ子午線の西にある座標は負の値になります。

制限事項 値を度数で表す場合、180 から–180 までの値にする必要があります。値をラジアンで表す場合、pi から–pi までの値にする必要があります。

データの種類 DOUBLE

options
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。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

詳細

GEODIST 関数は、2 つの緯度と経度の座標間の測地距離を計算します。入力値は、度数またはラジアンで表すことができます。

例

例 1: 測地距離(キロメートル単位)の計算

次の例では、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。

```
proc cas;
    distance=geodist(30.68, -88.25, 35.43, -82.55);
    print 'Distance= ' distance 'kilometers';
run;
```

SAS は次の出力をログに書き込みます。

```
Distance = 748.6529147 kilometers
```

例 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。

```
proc cas;
    distance=geodist(30.68, -88.25, 35.43, -82.55, 'M');
    print 'Distance= ' distance 'miles';
run;
```

SAS は次の出力をログに書き込みます。

```
Distance = 465.29081088 miles
```

参考文献

Vincenty, T. 1975. “Direct and Inverse Solutions of Geodesics on the Ellipsoid with Application of Nested Equations.” *Survey Review* 22: 99-93.

GEOMEAN 関数

幾何平均を返します。

返されるデータの種類: DOUBLE

構文

GEOMEAN(*expression* <, ...*expression*>)

引数

expression
負でない数値に評価される有効な式です。

データの種類 DOUBLE

詳細

いずれかの引数が負の場合、結果は null または欠損値になります。負の引数が無効であること示すメッセージがログに表示されます。いずれかの引数がゼロの場合、幾何平均はゼロになります。すべての引数が null または欠損値の場合、結果は null または欠損値になります。それ以外の場合、結果は非 null または非欠損値の幾何平均になります。

nを非 null または非欠損値を持つ引数の数とし、$x_1, x_2, \ldots, x_n$をそれらの引数の値とします。幾何平均は、次の値の積のn^{th}の平方根です。

$$\sqrt[n]{(x_1 * x_2 * \ldots * x_n)}$$

同様に、幾何平均はこの式で示されます。

$$\exp\left(\frac{(\log(x_1) + \log(x_2) + \ldots + \log(x_n))}{n}\right)$$

多くの場合、浮動小数点の計算では僅かな数値誤差が生じます。正確に計算すればゼロになる計算でも、浮動小数点の計算が使用されると僅かな非ゼロ値になる場合があります。そのため、GEOMEAN はほぼゼロの引数の値をファジー処理します。ある引数の値が最大の引数と比較して極端に小さい場合、前者の引数はゼロとして扱われます。極端に小さい値を SAS でファジー処理しない場合、GEOMEANZ 関数を使用します。

比較

MEAN 関数は算術平均(平均)、HARMEAN 関数は調和平均、GEOMEAN 関数は非 null または非欠損値の幾何平均を返します。GEOMEANZ や GEOMEAN とは異なり、ほぼゼロの引数の値をファジー処理します。

例

次のステートメントは、GEOMEAN 関数を示しています。

ステートメント	結果
x1=geomean(1,2,2,4);	2
x2=geomean(.,2,4,8);	4

GEOMEANZ 関数

ゼロファジーを使用して、幾何平均を返します。

返されるデータの種類: DOUBLE

構文

GEOMEANZ(*expression* <, ...*expression*>)

引数

expression
負でない数値に評価される有効な式を指定します。

データの種類 DOUBLE

詳細

いずれかの引数が負の場合、結果は null または欠損値になります。負の引数が無効であること示すメッセージがログに表示されます。いずれかの引数がゼロの場合、幾何平均はゼロになります。すべての引数が null または欠損値の場合、結果は null または欠損値になります。それ以外の場合、結果は非 null または非欠損値の幾何平均になります。

nを非 null または非欠損値を持つ引数の数とし、$x_1, x_2, \ldots, x_n$をそれらの引数の値とします。幾何平均は、次の値の積のn^{th}の平方根です。

$$\sqrt[n]{(x_1 * x_2 * \ldots * x_n)}$$

同様に、幾何平均はこの式で示されます。

$$\exp\left(\frac{(\log(x_1)+\log(x_2)+\ldots+\log(x_n))}{n}\right)$$

比較

MEAN 関数は算術平均(平均)、HARMEAN 関数は調和平均、GEOMEANZ 関数は非 null または非欠損値の幾何平均を返します。GEOMEAN や GEOMEANZ とは異なり、ほぼゼロの引数の値をファジー処理しません。

例

次のステートメントは、GEOMEANZ 関数を示しています。

ステートメント	結果
x1=geomeanz(1,2,2,4);	2
x2=geomeanz(.,2,4,8);	4

HARMEAN 関数

調和平均を返します。

返されるデータの種類: DOUBLE

構文

HARMEAN(*expression* <, ...*expression*>)

引数

expression
負でない数値に評価される有効な式を指定します。

データの種類 DOUBLE

詳細

いずれかの引数が負の場合、結果は null または欠損値になります。負の引数が無効であること示すメッセージがログに表示されます。すべての引数が null または欠損値の場合、結果は null または欠損値になります。それ以外の場合、結果は非 null または非欠損値の調和平均になります。

いずれかの引数がゼロの場合、調和平均はゼロになります。それ以外の場合、調和平均は値の逆数の算術平均の逆数になります。

nを非 null または非欠損値を持つ引数の数とし、$x_1, x_2, \ldots, x_n$をそれらの引数の値とします。調和平均はこの式で示されます。

$$\frac{n}{\frac{1}{x_1} + \frac{1}{x_2} + \ldots + \frac{1}{x_n}}$$

多くの場合、浮動小数点の計算では僅かな数値誤差が生じます。正確に計算すればゼロになる計算でも、浮動小数点の計算が使用されると僅かな非ゼロ値になる場合があります。そのため、HARMEAN はほぼゼロの引数の値をファジー処理します。ある引数の値が最大の引数と比較して極端に小さい場合、前者の引数はゼロとして扱われます。極端に小さい値を SAS でファジー処理しない場合、HARMEANZ 関数を使用します。

比較

MEAN 関数は算術平均(平均)、GEOMEAN 関数は幾何平均、HARMEAN 関数は非 null または非欠損値の調和平均を返します。HARMEANZ や HARMEAN とは異なり、ほぼゼロの引数の値をファジー処理します。

例

次のステートメントは、HARMEAN 関数を示しています。

ステートメント	結果
x1=harmean(1,2,4,4);	2
x2=harmean(.,4,12,24);	8

HARMEANZ 関数

ゼロファジーを使用して、調和平均を返します。

返されるデータの種類: DOUBLE

構文

HARMEANZ(*expression* <, ...*expression*>)

引数

expression
負でない数値に評価される有効な式を指定します。

データの種類 DOUBLE

詳細

いずれかの引数が負の場合、結果は null または値になります。負の引数が無効であること示すメッセージがログに表示されます。すべての引数が null または値の場合、結果は null または値になります。それ以外の場合、結果は非 null または非欠損値の調和平均になります。

いずれかの引数がゼロの場合、調和平均はゼロになります。それ以外の場合、調和平均は値の逆数の算術平均の逆数になります。

nを非 null または非欠損値を持つ引数の数とし、$x_1, x_2, \ldots, x_n$をそれらの引数の値とします。調和平均はこの式で示されます。

$$\frac{n}{\frac{1}{x_1}+\frac{1}{x_2}+\ldots+\frac{1}{x_n}}$$

比較

MEAN 関数は算術平均(平均)、GEOMEAN 関数は幾何平均、HARMEANZ 関数は非 null または非欠損値の調和平均を返します。HARMEAN や HARMEANZ とは異なり、ほぼゼロの引数の値をファジー処理しません。

例

次のステートメントは、HARMEANZ 関数を示しています。

ステートメント	結果
x1=harmeanz(1,2,4,4);	2

ステートメント	結果
x2=harmeanz(.,4,12,24);	8

HMS 関数

SAS 時間値(時、分および秒)を返します。

返されるデータの種類: DOUBLE

構文

HMS(*hour*, *minute*, *second*)

引数

hour
1 から 12 までの整数を表す数値式を指定します。

データの種類 DOUBLE

minute
1 から 59 までの整数を表す数値式を指定します。

データの種類 DOUBLE

second
1 から 59 までの整数を表す数値式を指定します。

データの種類 DOUBLE

詳細

HMS 関数は、SAS 時間値を表す数値を返します。SAS 時間値は、現在日の午前 0 時からの秒数を表す数値です。

例

次のステートメントは、HMS 関数を示しています。

ステートメント	結果
a=hms(12,45,10); b=put(a, time.);	45910 12:45:10

HOLIDAY 関数

指定した年の特定の祝日の SAS 日付値を返します。

返されるデータの種類: DOUBLE

構文

HOLIDAY('*holiday*', *year*)

引数

'*holiday*'
次のいずれかの値を指定する文字定数、変数または式です。BOXING、CANADA、CANADAOBSERVED、CHRISTMAS、COLUMBUS、EASTER、FATHERS、HALLOWEEN、LABOR、MLK、MEMORIAL、MOTHERS、NEWYEAR、THANKSGIVING、THANKSGIVINGCANADA、USINDEPENDENCE、USPRESIDENTS、VALENTINES、VETERANS、VETERANSUSG、VETERANSUSPS、VICTORIA。*holiday* の値には、大文字または小文字を使用できます。詳細については、*SAS DS2 Language Reference* を参照してください。

次の表に記載されているいずれかの値を指定する文字定数、変数または式です。

holiday の値には、大文字または小文字を使用できます。

表 4.2 祝日の値とその説明

祝日の値	説明	祝日の日付
BOXING	ボクシングデー	12 月 26 日
CANADA	カナダ独立記念日	7 月 1 日
CANADAOBSERVED	カナダ独立記念日振替休日	7 月 1 日または 7 月 2 (7/1 が日曜日の場合)
CHRISTMAS	クリスマス	12 月 25 日
COLUMBUS	コロンブス記念日	10 月の第 2 月曜日

祝日の値	説明	祝日の日付
EASTER	復活の主日	毎年変わる
FATHERS	父の日	6 月の第 3 日曜日
HALLOWEEN	ハロウィーン	10 月 31 日
LABOR	労働祭	9 月の第 1 月曜日
MLK	マーティンルーサーキング牧師の誕生日	1986 以降の 1 月の第 3 月曜日
MEMORIAL	メモリアルデー	5 月の最終月曜日(1971 以降)
MOTHERS	母の日	5 月の第 2 日曜日
NEWYEAR	元日	1 月 1 日
THANKSGIVING	U.S. 感謝祭	11 月の第 4 木曜日
THANKSGIVINGCANADA	カナダ感謝祭	10 月の第 2 月曜日
USINDEPENDENCE	U.S. 独立記念日	7 月 4 日
USPRESIDENTS	大統領記念日の祝日	2 月の第 3 月曜日 (1971 以降)
VALENTINES	バレンタインデー	2 月 14 日
VETERANS	退役軍人の日	11 月 11 日
VETERANSUSG	退役軍人の日(米国政府の祝日)	月曜日から金曜日までにスケジュールされる米国政府の祝日
VETERANSUSPS	退役軍人の日(米国郵政公社の祝日)	月曜日から日曜日までにスケジュールされる米国政府の祝日(米国郵政公社)
VICTORIA	ビクトリアデー	5 月 24 日以前の直近の月曜日

データの種類　CHAR

year
4 桁の年を指定する数値定数、変数または式です。2 桁の年を使用する場合、YEARCUTOFF=システムオプションを指定する必要があります。

データの種類 DOUBLE

詳細

HOLIDAY 関数は、指定した年に発生する特定の祝日の日付を計算します。米国およびカナダの特定の一般的な祝日のみがこの関数で使用できるように定義されています。

多くの祝日の定義は時代とともに変わります。米国では、1971 年 2 月 11 日に出された大統領命令 11582 で、多くの米国連邦休日の遵守を定着させました。

多くの休日はそれらが設立された日付が確立していますが、現在の祝日の定義は過去および将来に無限に拡張されます。一貫した定義がこれまでにされていない休日もあります。

HOLIDAY 関数は、SAS 日付値を返します。SAS 日付値をカレンダー日付に変換するには、DATE9.形式などの有効な SAS 日付形式を使用します。

例

次のステートメントは、HOLIDAY 関数を示しています。

ステートメント	結果
proc cas; thanks = holiday('thanksgiving', 2018); format thanks date9.; print thanks;	22NOV2018
proc cas; boxing = holiday('boxing', 2018); format boxing date9.; print boxing;	26DEC2018
proc cas; easter = holiday('easter', 2018); format easter date9.; print easter;	01APR2018
proc cas; canada = holiday('canada', 2018); format canada date9.; print canada;	01JUL2018
proc cas; fathers = holiday('fathers', 2018); format fathers date9.;	17JUN2018

ステートメント	結果
print fathers;	
proc cas; valentines = holiday('valentines', 2018); format valentines date9.; print valentines;	14FEB2018
proc cas; victoria = holiday('victoria', 2018); format victoria date9.; print victoria;	21MAY2018

HOUR 関数

SAS 時間値または SAS 日時値の時間を返します。

返されるデータの種類: DOUBLE

構文

HOUR(*time* | *datetime*)

引数

time
SAS 時間値を表す有効な式を指定します。

データの種類 DOUBLE

datetime
SAS 日時値を表す有効な式を指定します。

データの種類 DOUBLE

詳細

HOUR 関数は、SAS 時間値または SAS 日時値の時間を表す数値を返します。0-23 の数値が表示されます。HOUR は常に正の数を返します。

例

次のステートメントは、HOUR 関数を示しています。

ステートメント	結果
a=hour(time());	10

IBESSEL 関数

変形ベッセル関数の値を返します。

返されるデータの種類: DOUBLE

構文

IBESSEL(*nu*, *x*, *kode*)

必須引数

nu
数値の定数、変数または式を指定します。

範囲 *nu* ≥ 0

データの種類 DOUBLE

x
数値の定数、変数または式を指定します。

範囲 *x* ≥ 0

データの種類 DOUBLE

kode
負でない整数を指定する数値定数、変数または式です。

データの種類 DOUBLE

詳細

IBESSEL 関数は、*x* で評価される順序 *nu* の変形ベッセル関数(Abramowitz, Stegun 1964; Amos, Daniel, Weston 1977)の値を返します。*kode* が 0 の場合、ベッセル関数が返されます。それ以外の場合、次の関数の値が返されます。

$$\varepsilon^{-x} I_{nu}(x)$$

例

次のステートメントは、IBESSEL 関数を示しています。

ステートメント	結果
x=ibessel(2, 2, 0);	0.68894844769873
x=ibessel(2, 2, 1);	0.09323903330473

IFC 関数

式の true、false、欠損に基づいて文字値を返します。

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 バイトの長さが割り当てられます。

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。

IFN 関数

式の真、偽、欠損に基づいて数値を返します。

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。

%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。

返されるデータの種類: INTEGER

構文

INDEX(*target-expression*, *search-expression*)

引数

target-expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

search-expression
文字列を評価するか文字列に強制変換できて、*target-expression* での検索に使用される有効な式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

ヒント 文字のリテラル文字列は単一引用符で囲みます。

詳細

INDEX 関数は、*target-expression* を左から右へ、*search-expression* で指定される文字列の最初の出現箇所を検索し、*target-expression* 内での文字列の最初の文字の位置を返します。*target-expression* 内に文字列が見つからない場合、INDEX は値 0 を返します。文字列が複数回出現する場合、INDEX は、最初の出現箇所の位置のみを返します。

比較

VERIFY 関数は、*search-expression* 式を含まない *target-expression* の最初の文字の位置を返し、INDEX 関数は、*target-expression* に存在する *search-expression* の最初の出現位置を返します。

例

次のステートメントは、INDEX ステートメントを示しています。

ステートメント	結果
a='ABC.DEF (X=Y)'; b='X=Y'; c=index(a,b);	10

INDEXC 関数

文字式から指定文字を検索し、そのいずれかの文字の最初の出現位置を返します。

返されるデータの種類: DOUBLE

構文

INDEXC(*target-expression*, *search-expression*<, ...*search-expression*>)

引数

target-expression
文字列を評価するか、文字列に強制変換できる検索先の有効な式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

search-expression
target-expression で検索する文字を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

ヒント 文字のリテラル文字列は単一引用符で囲みます。

詳細

INDEXC 関数は、*target-expression* を左から右へ、検索式に存在するいずれかの文字の最初の出現箇所を検索し、その文字の *target-expression* 内の位置を返します。*target-expression* で検索式内の文字が見つからない場合、INDEXC は値 0 を返します。

比較

INDEXC 関数が、検索式に含まれる個々の文字が最初に出現する箇所を検索するのに対し、INDEX 関数は、検索式がパターンとして最初に出現する箇所を検索します。

例

次のステートメントは、INDEXC 関数を示しています。

ステートメント	結果

a='ABC.DEP (X2=Y1)';
b='()';
c='=.';
b=indexc(a,'0123',';()=.');
b=indexc(a,b);
	4
	8
	4

ステートメント	結果
b=indexc(a,b,c);	
c='have a good day'; d=indexc(c,'pleasant','very');	2

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。

返されるデータの種類: DOUBLE

構文

INDEXW(*target-expression*, *search-expression* <, *delimiter*>)

引数

target-expression
文字列を評価するか、文字列に強制変換できる検索先の有効な式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

search-expression
文字列を評価するか文字列に強制変換できて、*target-expression* で検索される有効な式を指定します。SAS は、*search-expression* から先頭および末尾の区切り文字を削除します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

delimiter
INDEXW で文字列の単語区切り文字として使用する文字式を指定します。デフォルトの区切り文字は空白文字です。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

ヒント 空白文字が区切り文字の場合は、*delimiter* の最後の文字にならないように並べ替えてください。*delimiter* は末尾の空白から取り除かれるため、末尾の空白は無視されます。

詳細

INDEXW 関数は、*target-expression* を左から右へ、*search-expression* の最初の出現箇所を検索し、*target-expression* 内での部分文字列の最初の文字の位置を返します。*target-expression* 内に部分文字列が見つからない場合、INDEXW は値 0 を返します。文字列が複数回出現する場合、INDEXW は、最初の出現箇所の位置のみを返します。

部分文字列パターンは、単語の境界で開始および終了する必要があります。INDEXW の場合、単語の境界は区切り文字、*target-expression* の最初、および *target-expression* の最後になります。

ヒント　*search-expression* に空白が含まれている場合や長さが 0 の場合、INDEXW は次のように動作します。

- *target-expression* と *search-expression* の両方に空白のみが含まれている場合や長さが 0 の場合、INDEXW は値 1 を返します。
- *search-expression* に空白のみが含まれている場合や長さが 0 の場合、*target-expression* に文字または数値のデータが含まれていると、INDEXW は値 0 を返します。

比較

INDEXW 関数は、単語の文字列を検索しますが、INDEX 関数は区切られた単語または他の単語の一部としてパターンを検索します。INDEXC は、excerpt 内に含まれる文字を検索します。

例

次のステートメントは、INDEXW 関数を示しています。

ステートメント	結果
a='The power to know.'; word='power'; c=indexw(a,word);	5
a='The power to know.'; b=indexw(a,'know');	0
a='The power to know.'; b=indexw(a,'know','.');	0
a='abc,def@ xyz'; b=indexw(a,',', '@');	0
a="abc,def@ xyz";	5

ステートメント	結果
b=indexw(a,'def', '@,');	
x="abc,def@ xyz"; xyz=indexw(x, ' xyz', '@');	10

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。

返されるデータの種類: CHAR、NCHAR、NVARCHAR、VARCHAR

構文

INPUTC(*source*, *informat*<, *w*>)

引数

source
入力形式を適用する文字定数、変数または式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

informat
source に適用する文字の入力形式が含まれる文字定数、変数または式です。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

w
入力形式に適用する数値幅に評価される有効な式を指定します。

操作 ここで指定した幅は、入力形式での幅の指定より優先されます。

データの種類 DOUBLE

詳細

INPUTC 関数が長さの割り当てられていない変数に値を返した場合、デフォルトでは変数の長さは最初の引数の長さによって決定されます。

比較

INPUTN 関数は、実行時に数値の入力形式を指定できるようにします。

例

この例では、文字の入力形式を指定する方法を示します。

```
proc cas;
    type=inputc('positive', '$upcase15.');
    type2=inputc('positive', '$upcase15.', 3);
    print "type=" type;
    print "type2=" type2;
  end;
run;
```

次の行が SAS ログに書き込まれます。

```
type=POSITIVE
type2=POS
```

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。

返されるデータの種類: DOUBLE

構文

INPUTN(*source*, *informat*<, *w*<, *d*>>)

引数

source
入力形式を適用する文字定数、変数または式を指定します。

データの種類 CHAR

informat
source に適用する数値の入力形式が含まれる文字定数、変数または式です。

データの種類 CHAR

w
入力形式に適用する幅を指定する数値定数、変数または式です。

操作　ここで指定した幅は、入力形式での幅の指定より優先されます。

データの種類　DOUBLE

d
使用する小数点以下の桁数を指定する数値定数、変数または式です。

操作　ここで指定した桁数は、入力形式での小数点以下の桁数の指定より優先されます。

データの種類　DOUBLE

比較

INPUTC 関数は、実行時に文字の入力形式を指定できるようにします。コンパイル時に入力形式を指定するため、PUT 関数を使用した方が高速になります。

例

この例では、数値の入力形式を指定する方法を示します。

```
proc cas;
   salary = inputn('20,000.00', comma10.2);
   print "salary=" salary;
run;
```

SAS は次の出力をログに書き込みます。

```
salary=20000
```

INT 関数

予期しない浮動小数点の結果を避けるためにファジー処理された整数を返します。

返されるデータの種類:　DOUBLE

構文

INT(*expression*)

引数

expression
数値に評価される任意の式を指定します。

データの種類 DOUBLE

比較

INTZ 関数とは異なり、INT 関数は結果をファジー処理します。引数が整数の 1E-12 内にある場合、INT 関数はその整数に等しくなるように結果をファジー処理します。INTZ 関数は結果をファジー処理しません。そのため、INTZ 関数では予期しない結果になる可能性があります。

例

次のステートメントは、INT 関数を示しています。

ステートメント	結果
var1=2.1; a=int(var1);	2
a=int(-2.4);	-2
a=int(1+1.e-11);	1
a=int(-1.6);	-1

INTCINDEX 関数

周期インデックスを返します。この関数には、日付、時間またはタイムスタンプの間隔と値を指定します。

返されるデータの種類: DOUBLE

構文

INTCINDEX({*interval*<*multiple*><.*shift-index*>}, *date-time-value*)

引数

interval
文字列を評価するか、文字列に強制変換でき、WEEK、MONTH、QTR などの間隔名を含む文字定数、変数、または式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

注 *interval* の可能な値

ヒント *Interval* は、大文字または小文字で表示できます。

例 YEAR は、年ベースの間隔を指定します。

multiple
乗数を指定します(省略可能)。基本タイプの間隔の期間に対する倍数と同等の間隔を設定します。

データの種類 DOUBLE

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。

date-time-value
指定した間隔の期間を表す日付、時間またはタイムスタンプの値を指定します。

データの種類 DOUBLE

詳細

基本

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 DATE、TIME または TIMESTAMP の値を指定します。たとえば、間隔が MONTH の場合、データ内の各オブザベーションは特定の月に対応します。月単位のデータは、1 年間で周期的とみなされます。1 年には 12 か月あるため、季節周期(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。2018 年 5 月 8 日はその年の 19 番目の週で 3 番目の曜日なので、この例では 19 の値が返されます。

```
cycle_index1 = intcindex('day', 21312);
```

間隔

間隔は、基本的なものから複雑なものまであります。基本間隔は、SAS が経過時間内にカウントできる測定単位(DAY、MONTH、HOUR など)です。より複雑な間隔を指定するには、基本間隔名に乗数とシフトインデックスを組み合わせて使用できます。

間隔構文は次のとおりです。

interval<multiple><.shift-index>

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。

この例では、INTCINDEX 関数は年間通算週を返します。

```
cycle_index = intcindex('day', 21185);
```

この例では、INTINDEX 関数は曜日を返します。

```
index = intindex('day',21185);
```

この例では、INTCINDEX 関数は時刻を返します。

```
a= intcindex('minute', 50883.620553);
```

この例では、INTINDEX 関数は分を返します。

```
a= intindex('minute', 50883.620553);
```

例

次のステートメントは、INTCINDEX 関数を示しています。

ステートメント	結果

proc cas;
  cycle_index1=intcindex('day', 45910);
  print "cycle_index1=" cycle_index1;
run;
	cycle_index1=37

proc cas;
  cycle_index1=intcindex('dtqtr', 45910);
  print "cycle_index1=" cycle_index1;
run;
	cycle_index1=1

proc cas;
  cycle_index1=intcindex('tenday', 45910);
  print "cycle_index1=" cycle_index1;
run;
	cycle_index1=1

proc cas;
  cycle_index1=intcindex('tenday', 45910);
  print "cycle_index1=" cycle_index1;
run;
	cycle_index1=13

proc cas;
  cycle_index1=intcindex('semimonth', 45910);
  print "cycle_index1=" cycle_index1;
run;
	cycle_index1=1

INTCK 関数

DOUBLE としてエンコードされた 2 つの SAS 日付、時間、またはタイムスタンプ値の間にある特定の種類の間隔境界の数を返します。

返されるデータの種類: DOUBLE

構文

INTCK({*interval*<*multiple*><*.shift-index*>}, *start-date*, *end-date*<, '*method*'>)

INTCK(*start-date*, *end-date*<, '*method*'>)

引数

interval
文字列を評価するか、文字列に強制変換でき、WEEK、MONTH、QTR などの間隔名を含む文字定数、変数、または式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

ヒント *Interval* は、大文字または小文字で表示できます。

例 YEAR は、年ベースの間隔を指定します。

multiple
乗数を指定します(省略可能)。基本タイプの間隔の期間に対する倍数と同等の間隔を設定します。

データの種類 DOUBLE

例 YEAR2 は、2 年、つまり 2 年間続く間隔タイプを指定します。

shift-index
シフトインデックスを指定します(省略可能)。間隔の開始時点を指定したサブ期間にシフトします。

データの種類 DOUBLE

start-date
SAS を開始する日付、時間、またはタイムスタンプ値を表す式を指定します。

データの種類 DOUBLE

end-date
SAS を終了する日付、時間、またはタイムスタンプ値を表す式を指定します。

データの種類 DOUBLE

'*method*'
DISCRETE メソッドと CONTINUOUS メソッドのどちらを使用して間隔を数えるかを指定します。*method* は引用符で囲みます。*method* は、CONTINUOUS と DISCRETE (間隔境界をカウント)のいずれかの値になります。

DISCRETE メソッドと CONTINUOUS メソッドのどちらを使用して間隔を数えるかを指定します。

method は引用符で囲む必要があります。*Method* には次のいずれかの値を指定できます。

CONTINUOUS
連続した期間を測定単位に指定します。間隔は開始日に基づいてシフトされます。

たとえば、2013 年 1 月 15 日から 2013 年 2 月 15 日までの月単位の距離は 1 か月です。

別名　C または CONT

DISCRETE
連続しない期間を測定の単位に指定します。DISCRETE メソッドは、間隔の境界(月末など)を数えます。

デフォルトの DISCRETE メソッドは、時系列オブザベーションをビンに分類して処理する場合に役立ちます。たとえば、日単位のデータを月単位の時系列として処理するために月単位のデータに累積できます。

DISCRETE メソッドの場合、2000 年 1 月 31 日から 2000 年 2 月 1 日までの間隔を月単位で示すと、1 か月になります。

別名　D または DISC

デフォルト　DISCRETE

データの種類　CHAR、NCHAR、NVARCHAR、VARCHAR

詳細

間隔

間隔は、基本的なものから複雑なものまであります。基本間隔は、SAS が経過時間内にカウントできる測定単位(DAY、MONTH、HOUR など)です。より複雑な間隔を指定するには、基本間隔名に乗数とシフトインデックスを組み合わせて使用できます。

間隔構文は次のとおりです。

interval<multiple><.shift-index>

カレンダー間隔の計算

個々の時間間隔内に含まれる値は、すべて同等とみなします。つまり、月間隔を指定した場合、2018 年 1 月 1 日と 2018 年 1 月 15 日は同等です。これら両方の日付

は、2018 年 1 月 1 日に開始して 2018 年 1 月 31 日に終了する間隔を表します。この間隔を示すには、間隔の開始日(2018 年 1 月 1 日)または間隔の終了日(2018 年 1 月 31 日)を使用できます。これらの日付によって、月間隔内のすべての日付が表されます。

次の例では、*start-date* は SAS 値 21185 (2018 年 1 月 1 日)であり、2018 年の第 1 四半期に相当します。

```
qtr=intck('qtr', 21185, 21366);
```

end-date の SAS 日付値は 21366 (2018 年 7 月 1 日)で、2018 年の第 2 四半期に相当します。間隔数(*start-date* と *end-date* の間に間隔の開始点が含まれる回数)は 2 です。

デフォルトの DISCRETE メソッドを使用する INTCK 関数は、1 番目の日付と 2 番目の日付の間に次の間隔の開始点が含まれる回数を数えます。2 つの日付間に含まれる間隔数は計算しません。

- 次の例では、2 つの日付が同じ月内にあるため、0 が返されます。

  ```
  proc cas;
   month=intck("month", 21185, 21202);
   print "month=" month;
  run;
  ```

- 次の例では、2 つの日付が 1 か月離れた異なる月にあるため、1 が返されます。

  ```
  proc cas;
   month=intck("month", 21185, 21216);
   print "month=" month;
  run;
  ```

- 次の例では、最初の日付が 2 番目の日付よりも遅い離散間隔にあるため、-1 が返されます。(1 番目の日付が 2 番目の日付よりも後で、2 つの日付が同じ間隔内に存在しない場合、INTCK は常に負の値を返します)。

  ```
  proc cas;
   month=intck("month", 21216, 21185);
   print "month=" month;
  run;
  ```

DISCRETE メソッドを使用する場合、WEEK 間隔は *start-date* と *end-date* の間に 7 日間がいくつ含まれるかではなく、2 つの日付間に存在する日曜日(週のデフォルトの開始曜日)の数で決定されます。*start-date* と *end-date* の間に 7 日間が含まれる数を数えるには、CONTINUOUS メソッドを使用します。

multiple 引数と *shift-index* 引数は両方とも任意で、デフォルトで 1 になっています。たとえば、YEAR、YEAR1、YEAR.1、YEAR1.1 は、すべて通常のカレンダー年を指定します。

カテゴリ別の間隔

表 4.3　日付と時間の関数で使用される間隔

カテゴリ	間隔	定義	デフォルトの開始点	シフト期間	例	説明
日付	DAY	日間隔	毎日	日	DAY3	日曜日から始まる 3 日間隔
	WEEK	7 日間の週間隔	毎週日曜日	日(1=日曜日...7=土曜日)	WEEK.7	土曜日を週の最初の日とする毎週
	WEEKDAY <*days*W>	金土日を含む日間隔	毎日	日	WEEKDAY1W	日曜日を週末日、6 日間を就業日とする週
		同じ日としてカウントされます(土日を週末として 5 日間の稼働日)。*days* では、週末の曜日が数字で識別されます(1=日曜日...7=土曜日)。デフォルトでは、*days*=17 です。			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)
	TENDAY	10 日間隔(米国自動車産業の慣例)	毎月 1 日、11 日、21 日	10 日間	TENDAY4.2	TENDAY 単位の 2 番目を開始時点とする 40 日単位
	SEMIMONTH	半月間隔	毎月 1 日と 16 日	半月間	SEMIMONTH2.2	月の 16 日目から翌月の 15 日目までの間隔
	MONTH	月間隔	毎月 1 日	月	MONTH2.2	翌年の 2 月-3 月、4 月-5 月、6 月-7 月、8

カテゴリ	間隔	定義	デフォルトの開始点	シフト期間	例	説明
						月-9 月、10 月-11 月、12 月-1 月
	QTR	四半期(3 か月)間隔	1 月 1 日	月	QTR3.2	4 月 1 日、7 月 1 日、10 月 1 日、1 月 1 日から 3 か月間隔
			4 月 1 日			
			7 月 1 日			
			10 月 1 日			
	SEMIYEAR	半年(6 か月)間隔	1 月 1 日	月	SEMIYEAR.3	3 月-8 月と 9 月-2 月の 6 か月間隔
			7 月 1 日			
	YEAR	年間隔	1 月 1 日	月		
日時	日付間隔のいずれかに DT を追加	関連する日付間隔に対応する間隔	1960 年 1 月 1 日の午前 0 時		DTMONTH	
					DTWEEKDAY	
時間	SECOND	秒間隔	1 日の始まり(午前 0 時)	秒		
	MINUTE	分間隔	1 日の始まり(午前 0 時)	分		
	HOUR	時間隔	1 日の始まり(午前 0 時)	時		

小売カレンダーの間隔

小売業界では、1 年のカレンダーを 13 週間からなる 4 つの期間に分けてデータを計算することがよくあります。期間の形式は、4-4-5、4-5-4 または 5-4-4 のいずれかに基づきます。1 番目、2 番目、3 番目の数値には、それぞれ各期間の 1 か月目、2 か月目、3 か月目の週数を指定します。

例

次のステートメントは、INTCK 関数を示しています。

ステートメント	結果

proc cas;
qtr=intck('qtr', 21185, 21366);
print "qtr=" qtr;
run;
	qtr=2

proc cas;
year=intck('year', 21185, 21550);
print "year=" year;
run;
	year=1

proc cas;
year=intck('year', 21185, 21489);
print "year=" year;
run;
	year=0

proc cas;
semiyear=intck('semiyear', 21185, 21915);
print "semiyear=" semiyear;
run;
	semiyear=4

proc cas;
weekvar=intck('week2.2', 21185, 21550);
print "weekvar=" weekvar;
run;
	weekvar=26

proc cas;
wdvar=intck('weekday7w', 21185, 21216);
print "wdvar=" wdvar;
run;
	wdvar=27

proc cas;
newyears=intck('year', 17900, 21550);
print "newyears=" newyears;
run;
	newyears=10

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。

返されるデータの種類: CHAR、NCHAR、NVARCHAR、VARCHAR

構文

INTCYCLE({*interval*<*multiple*><.*shift-index*>}<, *seasonality*>)

引数

interval
文字列を評価するか、文字列に強制変換でき、WEEK、MONTH、QTR などの間隔名を含む文字定数、変数、または式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

注 *interval* の可能な値

ヒント *Interval* は、大文字または小文字で表示できます。

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。

データの種類 DOUBLE

例 YEAR2 は、2 年、つまり 2 年間続く間隔タイプを指定します。

shift-index
シフトインデックスを指定します(省略可能)。間隔の開始時点を指定したサブ期間にシフトします。

データの種類 DOUBLE

seasonality
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。

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。

デフォルト 52

データの種類 DOUBLE、CHAR、NCHAR、NVARCHAR、VARCHAR

例 次の例では、*seasonality* 引数により
INTCYCLE('MONTH', 3);
関数呼び出しで値 QTR が返されます。この関数呼び出しは
INTCYCLE('MONTH');
seasonality 引数がないため、値 YEAR を返します。

詳細

基本

INTCYCLE 関数は、日付、時間または日時の間隔に応じて、季節周期の間隔を返します。たとえば、INTCYCLE('MONTH');では、1 月から 12 月までで 1 年周期を構成するため、値 YEAR を返します。INTCYCLE('DAY');では、日曜日から土曜日までで 1 週間周期を構成するため、値 WEEK を返します。

間隔

間隔は、基本的なものから複雑なものまであります。基本間隔は、SAS が経過時間内にカウントできる測定単位(DAY、MONTH、HOUR など)です。より複雑な間隔を指定するには、基本間隔名に乗数とシフトインデックスを組み合わせて使用できます。

間隔構文は次のとおりです。

interval<multiple><.shift-index>

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。

例

次のステートメントは、INTCYCLE 関数を示しています。

ステートメント	結果

proc cas;
  cycle_year=intcycle('year');
  print "cycle_year=" cycle_year;
run;
	cycle_year=YEAR

proc cas;
  cycle_quarter=intcycle('qtr');
  print "cycle_quarter=" cycle_quarter;
run;
	cycle_quarter=YEAR

proc cas;
  cycle_month=intcycle('month', 3);
  print "cycle_month=" cycle_month;
run;
	cycle_month=QTR

proc cas;
  cycle_month=intcycle('month');
  print "cycle_month=" cycle_month;
run;
	cycle_month=YEAR

proc cas;
  cycle_weekday=intcycle('weekday');
  print "cycle_weekday=" cycle_weekday;
run;
	cycle_weekday=WEEK

ステートメント	結果

proc cas;
cycle_weekday2=intcycle('weekday', 5);
print "cycle_weekday2=" cycle_weekday2;
run;
	cycle_weekday2=WEEK

proc cas;
cycle_day=intcycle('day');
print "cycle_day=" cycle_day;
run;
	cycle_day=WEEK

proc cas;
cycle_day2=intcycle('day', 10);
print "cycle_day2=" cycle_day2;
run;
	cycle_day2=TENDAY

proc cas;
cycle_second=intcycle('second');
print "cycle_second=" cycle_second;
run;
	cycle_second=DTMINUTE

INTFIT 関数

2 つの日付に基づく時間間隔を返します。

返されるデータの種類: CHAR、NCHAR、NVARCHAR、VARCHAR

構文

INTFIT(*expresssion-1*, *expression-2*, '*type*')

引数

expression
文字列を評価するか、文字列に強制変換できて、SAS 日付または日時値を表す有効な式を指定します。

データの種類 DOUBLE

'*type*'
引数が SAS 日付値、日時値、行のいずれであるかを指定します。*type* の有効な値は、d (日付)、dt (日時)、および obs (行)です。

引数が SAS 日付値、日時値、行のいずれであるかを指定します。

type の有効な値は次のとおりです。

d *expression-1* と *expression-2* が日付値であることを指定します。

dt *expression-1* と *expression-2* が日時値であることを指定します。

obs *expression-1* と *expression-2* が行であることを指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

例

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。

```
proc cas;
  sasdate1=17900;
  sasdate2=21550;
  intfit=intfit(sasdate1, sasdate2, 'd');
  print "intfit=" intfit;
run;
```

次の行が SAS ログに書き込まれます。

```
intfit=DAY3650.3301
```

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。

返されるデータの種類: CHAR、NCHAR、NVARCHAR、VARCHAR

構文

INTGET(*date-1*, *date-2*, *date-3*)

引数

date
SAS 日付または日時の値に評価される有効な式を指定します。

データの種類 DOUBLE

詳細

INTGET 関数の間隔

INTGET 関数は、3 つの日付値または日時値に基づく時間間隔を返します。この関数は最初の 2 つの日付間で可能なすべての間隔を最初に確認し、次に 2 番目と 3 番目の日付間で可能なすべての間隔を確認します。すべての間隔が同じ場合、INTGET はその間隔を返します。1 番目と 2 番目の日付間に異なる間隔があり、2 番目と 3 番目の日付間に異なる間隔がある場合、INTGET はそれらの間隔を比較します。他の間隔の倍数となる間隔がある場合、INTGET はその小さい方の間隔を返します。それ以外の場合、INTGET は欠損値を返します。INTGET は、配置の値が BEGIN の INTNX 関数によって生成された日付に適しています。

次の例では、INTGET は間隔 DAY2 を返します。

```
interval=intget('01mar00'd, '03mar00'd, '09mar00'd);
```

2000 年 3 月 1 日から 2000 年 3 月 3 日までの日数は 2 であるため、1 番目と 2 番目の日付間の間隔は DAY2 です。2000 年 3 月 3 日から 2000 年 3 月 9 日までの日数は 6 であるため、2 番目と 3 番目の日付間の間隔は DAY6 です。DAY6 は DAY2 の倍数です。INTGET はこの 2 つの間隔のうち小さい方を返します。

次の例では、INTGET は間隔 MONTH4 を返します。

```
interval=intget('01jan00'd, '01may00'd, '01may01'd);
```

2000 年 1 月 1 日から 2000 年 5 月 1 日までの月数は 4 であるため、最初の 2 つの日付間の間隔は MONTH4 です。2 番目と 3 番目の日付間の間隔は YEAR です。YEAR は MONTH4 の倍数(YEAR には 3 つの MONTH4 間隔が含まれる)であるため、INTGET は 2 つの間隔のうち小さい方の間隔を返します。

次の例では、INTGET は欠損値を返します。

```
interval=intget('01Jan2006'd, '01Apr2006'd, '01Dec2006'd);
```

最初の 2 つの日付間の間隔は MONTH3、2 番目と 3 番目の日付間の間隔は MONTH8 です。MONTH8 は MONTH3 の倍数ではないため、INTGET は欠損値を返します。

返される間隔は有効な SAS 間隔で、間隔とシフト間隔の倍数が含まれます。有効な SAS 間隔がリストされています。

注: INTGET が一致する間隔を確認できない場合、関数は欠損値を返します。SAS ログにメッセージは書き込まれません。

例

次のステートメントは、INTGET 関数を示しています。

ステートメント	結果
sasdate1=to_double(date'2013-01-01'); sasdate2=to_double(date'2014-01-01');	MONTH4

ステートメント	結果

sasdate3=to_double(date'2014-05-01');
c=intget(sasdate1, sasdate2, sasdate3);
put c;	

sasdate1=to_double(date'2012-02-29');
sasdate2=to_double(date'2014-02-28');
sasdate3=to_double(date'2016-02-29');
c=intget(sasdate1, sasdate2, sasdate3);
put c;
	YEAR2.2

sasdate1=to_double(date'2013-02-01');
sasdate2=to_double(date'2013-02-16');
sasdate3=to_double(date'2013-03-01');
c=intget(sasdate1, sasdate2, sasdate3);
put c;
	SEMIMONTH

sasdate1=to_double(date'2013-01-02');
sasdate2=to_double(date'2014-02-02');
sasdate3=to_double(date'2015-03-02');
c=intget(sasdate1, sasdate2, sasdate3);
put c;
	MONTH13.13

sasdate1=to_double(date'2013-02-10');
sasdate2=to_double(date'2013-02-19');
sasdate3=to_double(date'2013-02-28');
c=intget(sasdate1, sasdate2, sasdate3);
put c;
	DAY9.5

sasdate1=to_double(timestamp'2014-04-01 01:03:00.0000');
sasdate2=to_double(timestamp'2014-04-01 01:04:00.0000');
sasdate3=to_double(timestamp'2014-04-01 01:05:00.0000');
c=intget(sasdate1, sasdate2, sasdate3);
put c;
	MINUTE

INTINDEX 関数

季節インデックスを返します。この関数には、日付、時間またはタイムスタンプの間隔と値を指定します。

返されるデータの種類: DOUBLE

構文

INTINDEX({*interval*<*multiple*><.*shift-index*>}, *date-value*<, *seasonality*>)

引数

interval
文字列を評価するか、文字列に強制変換でき、WEEK、MONTH、QTR などの間隔名を含む文字定数、変数、または式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

注 *interval* の可能な値

ヒント *Interval* は、大文字または小文字で表示できます。

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。

データの種類 DOUBLE

例 YEAR2 は、2 年、つまり 2 年間続く間隔タイプを指定します。

shift-index
シフトインデックスを指定します(省略可能)。間隔の開始時点を指定したサブ期間にシフトします。

制限事項 シフトインデックスは、間隔全体のサブ期間の数以下にする必要があります。たとえば、2 年間隔では 25 番目の月は存在しないため、YEAR2.24 は使用できますが YEAR2.25 はエラーになります。

デフォルトのシフト期間が間隔の種類と同じ場合、複数期間の間隔のみを任意のシフトインデックスでシフトできます。たとえば、MONTH の種類の間隔はデフォルトでは MONTH のサブ期間でシフトされるため、シフトインデックスで月間隔をシフトできません。ただし、各 MONTH2 間隔には 2 つの MONTH 間隔が含まれるため、シフトインデックスで 2 か月間隔をシフトできます。たとえば、間隔名 MONTH2.2 では、偶数月の 1 日目に開始する 2 か月の期間が指定されます。

データの種類 DOUBLE

例 YEAR.3 は、暦年の 3 月 1 日に開始し、翌年の 2 月に終了するようにシフトされた年次期間を指定します。

date-value
指定した間隔の期間を表す日付、時間またはタイムスタンプの値を指定します。

データの種類 DOUBLE

seasonality
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。

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。

データの種類 DOUBLE、CHAR、NCHAR、NVARCHAR、VARCHAR

例 この例では、次の関数で同じ結果が生成されます。

```
INTINDEX('MONTH', sasdate, 3);
INTINDEX('MONTH', sasdate, 'QTR');
```

最初の例の *seasonality* は数値(月数)で、2 番目の例の *seasonality* は周期(QTR)です。

詳細

基本

INTINDEX 関数は、季節インデックスを返します。この関数には、間隔と適切な日付、時間またはタイムスタンプの値を指定します。季節インデックスは、指定した間隔の季節周期での日付、時間またはタイムスタンプの値の位置を表す数値です。1 年周期には 12 か月があり 12 月はその年の 12 番目の月であるため、この例では 12 の値が返されます。

```
sasdate=to_double(date'2012-12-01');
x=intindex('month', sasdate);
put x;
```

次の例では、両方のステートメントが 2013 年第 1 四半期内の日付を含むため、INTINDEX は同じ値(1)を返します。

```
sasdate=to_double(date'2013-01-01');
x=intindex('qtr', sasdate);
put x;

sasdate=to_double(date'2013-03-31');
y=intindex('qtr', sasdate);
put y;
```

次の例では、日単位のデータは週単位の周期に含まれ、2012 年 12 月 7 日は金曜日で 6 番目の曜日なので、6 の値を返します。

```
sasdate=to_double(date'2012-12-07');
x=intindex('day', sasdate);
put x;
```

間隔

間隔は、基本的なものから複雑なものまであります。基本間隔は、SAS が経過時間内にカウントできる測定単位(DAY、MONTH、HOUR など)です。より複雑な間隔を指定するには、基本間隔名に乗数とシフトインデックスを組み合わせて使用できます。

間隔構文は次のとおりです。

interval<multiple><.shift-index>

Interval と *Date-Time-Value* の関係

季節インデックスを正しく識別するには、間隔が日付、時間またはタイムスタンプの値に適合する必要があります。たとえば、intindex('month', '01DEC2012'd);では、1 年の間隔には 12 か月があり 12 月はその年の 12 番目の月であるため、12 の値を返します。MONTH 間隔には SAS 日付値が必要です。週間隔には 7 日あり、2012 年 12 月 7 日は金曜日で 6 番目の曜日なので、次の例では 6 の値が返されます。

```
sasdate=to_double(date'2012-12-07');
x=intindex('day', sasdate);
put x;
```

DAY 間隔には SAS 日付値が必要です。

この例では、QTR 間隔が TIMESTAMP 値でなく SAS 日付値を日付に必要とするため、欠損値が返されます。

```
sasdate=to_double(timestamp'2013-01-01 00:00:00');
x=intindex('qtr', sasdate);
put x;
```

この例では、12 の値が返されます。DTMONTH 間隔には TIMESTAMP 値が必要です。

```
sasdate=to_double(timestamp'2013-12-01 00:00:00');
x=intindex('dtmonth', sasdate);
put x;
```

季節性

季節性は、年内の異なる間隔での周期的変動を測定する時系列の概念です。季節性の指定では、季節が最も一般的な変動ソースです。たとえば、家庭暖房用の灯油は、一般的に他の季節よりも冬の売り上げの方が高くなります。多くの場合、日単位の時系列では曜日による定期的な変動(週末のレジャーで支出が増えるなど)が発生します。INTINDEX 関数は、季節性の概念を使用して季節インデックスを返します。この関数には、日付、時間またはタイムスタンプの間隔と値を指定します。季節性と PROC FORECAST での予測手法の使用方法の詳細については、*SAS/ETS 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。

次の例では、2013 年 4 月 4 日は木曜日で週の 5 番目の曜日なので、INTINDEX 関数は 5 を返します。

```
sasdate=to_double(date'2013-04-04');
x = intindex('day', sasdate);
put x;
```

同じ日を使用しても、2013 年 4 月 4 日は年の 14 番目の週なので、INTCINDEX 関数は 14 を返します。

```
sasdate=to_double(date'2013-04-04');
x = intcindex('day', sasdate);
```

```
put x;
```

この例では、INTINDEX 関数は分を返します。

```
sasdate=to_double(timestamp'2012-09-01 06:05:04');
x = intindex('minute', sasdate);
put x;
```

同じ日付と時間を使用しても、INTCINDEX 関数は時刻を返します。

```
sasdate=to_double(timestamp'2012-09-01 06:05:04');
y = intcindex('minute', sasdate);
put y;
```

例 intseas('interval');では、INTSEAS は intindex('interval', date);によって返される最大数を返します。

例

次のステートメントは、INTINDEX 関数を示しています。

ステートメント	結果

sasdate1=to_double(date'2013-08-14');
interval1 = intindex('qtr', sasdate1);
put interval1;
	3

sasts1=to_double(timestamp'2013-12-23 15:09:19');
interval2 = intindex('dtqtr', sasts1);
put interval2;
	4

sastime1=to_double(time'09:05:15');
interval3 = intindex('hour', sastime1);
put interval3;
	10

sasdate1=to_double(date '2013-02-26');
interval4 = intindex('month', sasdate1);
put interval4;
	2

sasts1=to_double(timestamp'2013-05-28 05:15:00');
interval5 = intindex('dtmonth', sasts1);
put interval5;
	5

sasdate1=to_double(date '2013-09-09');
interval6 = intindex('week', sasdate1);
put interval6;
	37

sasdate1=to_double(date '2013-04-16');
interval7 = intindex('tenday', sasdate1);
put interval7;
	11

関連項目

その他のリファレンス:

- *SAS/ETS User's Guide*

INTNX 関数

DOUBLE としてエンコードされた SAS 日付、時間または日時の値を増分し、DOUBLE としてエンコードされた SAS 日付、時間または日時の値を返します。

返されるデータの種類: DOUBLE

構文

INTNX({*interval*<*multiple*><.*shift-index*>}, *start-from*, *increment*<, '*alignment*'>)

INTNX(*start-from*, *increment*<, '*alignment*'>)

引数

interval
文字列を評価するか、文字列に強制変換でき、WEEK、MONTH、QTR などの間隔名を含む文字定数、変数、または式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

ヒント *Interval* は、大文字または小文字で表示できます。

例 YEAR は、年ベースの間隔を指定します。

multiple
乗数を指定します(省略可能)。基本タイプの間隔の期間に対する倍数と同等の間隔を設定します。

データの種類 DOUBLE

例 YEAR2 は、2 年、つまり 2 年間続く間隔タイプを指定します。

shift-index
シフトインデックスを指定します(省略可能)。間隔の開始時点を指定したサブ期間にシフトします。

start-from
DOUBLE としてエンコードされた SAS 日付、時間または日時の値を表し、開始点を識別する式を指定します。

データの種類 DOUBLE

increment
日付、時間または日時の間隔を表す負、正またはゼロの整数を指定します。*Increment* は、*start-from* の値をシフトする間隔数です。

データの種類 DOUBLE

'*alignment*'
間隔内の SAS 日付の位置を制御します。*alignment* は引用符で囲む必要があります。*alignment* には、BEGINNING、MIDDLE、END、SAME のいずれかの値を指定します。SAME では、返された日付を入力日付と同じ場所に配置するように指定されます。詳細については、SAS DS2 言語リファレンスを参照してください。

間隔内の SAS 日付の位置を制御します。*alignment* は引用符で囲む必要があります。*Alignment* には次のいずれかの値を指定できます。

BEGINNING
返された日付値または日時値を間隔の開始点に配置するように指定します。

別名 B

MIDDLE
返された日付値または日時値を間隔の中間点(開始位置と終了位置の値の平均)に配置するように指定します。

別名 M

END
返された日付値または日時値を間隔の終了点に配置するように指定します。

別名 E

SAME
返された日付を入力日付と同じ場所に配置するように指定します。

別名 S

SAMEDAY

デフォルト BEGINNING

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

詳細

基本

INTNX 関数は、DAY、WEEK、QTR、MINUTE などの間隔、または定義したカスタム間隔単位で日付、時間または日時の値を増分します。間隔は、開始日付、時間または日時の値、および指定した時間間隔に基づきます。

INTNX 関数は、*start-from* 引数で指定した間隔の開始日付、時間または日時の値に対する SAS 日付値を返します。(日付値をカレンダー日付に変換するには、DATE9.

形式などの有効な DS2 日付形式を使用します)。2011 年 10 月 17 日の週から 6 週間後の週の開始日を確認する方法の例を次に示します。

```
sasdate=to_double(date'2011-10-17');
x=intnx('week', sasdate, 6);
print put x date9.;
```

INTNX は値 27NOV2011 を返します。

間隔

間隔は、基本的なものから複雑なものまであります。基本間隔は、SAS が経過時間内にカウントできる測定単位(DAY、MONTH、HOUR など)です。より複雑な間隔を指定するには、基本間隔名に乗数とシフトインデックスを組み合わせて使用できます。

間隔構文は次のとおりです。

interval<multiple><.shift-index>

SAS 日付出力の間隔内の配置

通常、SAS 日付値は *interval* 引数で指定した時間間隔の開始点に配置されます。

任意の *alignment* 引数を使用して、返された値の配置を指定できます。値 BEGINNING、MIDDLE または END は、それぞれ間隔の開始点、中間点または終了点に日付を配置します。

SAME 配置

alignment 引数の SAME 値を使用した場合、INTNX は指定した間隔の増分を計算した後に同じカレンダー日付を返します。同じカレンダー日付は、間隔ではなく間隔のシフト期間に基づいて配置されます。

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。

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。

日付の調整

増分された間隔の日付が存在しない場合、INTNX 関数はその日付を自動的に調整します。

例

例 1: INTNX 関数の使用

次のステートメントは、INTNX 関数を示しています。

ステートメント	結果

sasdate1 = to_double(date'2013-02-05');
yr=intnx('year', sasdate1, 3);
print yr;
print put yr date7.;
	20454
	01Jan16

sasdate1 = to_double(date'2013-01-05');
x=intnx('month', sasdate1, 0);
print x;
print put x date7.;
	19359
	01JAN13

sasdate1 = to_double(date'2013-01-01');
next=intnx('semiyear', sasdate1, 1);
print next;
print put next date7.;
	19540
	01JUL13

sasdate1 = to_double(date'2012-08-01');
past=intnx('month2', sasdate1, -1);
print past;
print put past date7.;
	19114
	01MAY12

sasdate1 = to_double(date'2013-04-01');
sm=intnx('semimonth2.2', sasdate1, 4);
print sm;
print put sm date7.;
	19555
	16JUL13

x='month';
sasdate1 = to_double(date'2013-06-01');
nextmon=intnx(x, sasdate1, 1);
print nextmon;
print put nextmon date7.;
	19540
	01JUL13

m1='month ';
m2=trim(m1);
sasdate1 = to_double(date'2012-06-15');
x=intnx(m2, sasdate1, 1);
print x;
print put x date7.;
	19175
	01JUL12

例 2: ALIGNMENT 引数の使用

任意の *alignment* 引数を使用して日付を進める例を次に示します。

ステートメント	結果

sasdate1 = to_double(date'2013-01-01');
x=intnx('month', sasdate1, 5, 'beginning');
print x;
print put x date7.;
	19510
	01JUN13

sasdate1 = to_double(date'2013-01-01');
x=intnx('month', sasdate1, 5, 'middle');
print x;
print put x date7.;
	19524
	15JUN13

sasdate1 = to_double(date'2013-01-01');
x=intnx('month', sasdate1, 5, 'end');
print x;
print put x date7.;
	19539
	30JUN13

sasdate1 = to_double(date'2013-01-01');
x=intnx('month', sasdate1, 5, 'sameday');
print x;
print put x date7.;
	19510
	01JUN13

sasdate1 = to_double(date'2013-03-15');
x=intnx('month', sasdate1, 5, 'same');
print x;
print put x date7.;
	19585
	15AUG13

interval='month';
align='m';
sasdate1 = to_double(date'2013-09-01');
x=intnx(interval, sasdate1,2, align);
print x;
print put x date7.;
	19667
	15NOV13

m1='month ';
m2=trim(m1);
sasdate1 = to_double(date'2012-09-01');
x=intnx(m2, sasdate1, 2, 'm');
print x;
print put x date7.;
	19312
	15NOV12

INTRR 関数

内部利益率を小数値で返します。

返されるデータの種類: DOUBLE

構文

INTRR(*freq*, *c0, c1*<*..., cn*>)

引数

freq
目的の内部利益率に関連付けられた指定ベース期間での支払い数を示す数値です。

範囲 *freq* > 0

データの種類 DOUBLE

ヒント *freq* = 0 は、連続複利計算を許可するフラグです。

c0, c1< ***..., cn***>
任意の現金払いを示す数値です。

データの種類 DOUBLE

詳細

INTRR 関数は、現金払いセット *c0*, *c1*,..., *cn* の指定ベース期間での内部利益率を返します。2 つの連続した支払い間の時間間隔は同じとみなされます。引数 *freq* > 0 は、指定ベース期間で発生する支払い数を表します。各インスタンスから発行される情報の数は制限されます。

内部利益率は、一連の支払いの正味現在価値が 0 となる利率です。これは次の式で求められます。

$$r = \begin{cases} \frac{1}{x^{freq}} - 1 & freq > 0 \\ -log_{\varepsilon}(x) & freq = 0 \end{cases}$$

この式では、*x* は多項式の実根です。

$$\sum_{i=0}^{n} c_i x^i = 0$$

重根の場合、1 つの実根が返され、返される内部利益率の非一意性に関する警告が発行されます。支払いの値によっては、方程式の根が存在しないことがあります。その場合、欠損値が返されます。

支払いの欠損値は 0 値として扱われます。*freq* > 0 の場合、計算される利益率は、指定ベース期間で有効な利率です。月払いでの四半期の内部利益率(ベース期間は 3 か月)を計算するには、*freq* を 3 に設定します。

freq が 0 の場合は連続複利計算とみなされ、ベース期間は 2 つの連続する支払い間の時間間隔です。計算される内部利益率は、ベース期間の名目利益率です。月単位の支払いを連続複利計算するには、*freq* を 0 に設定します。計算される内部利益率は、月単位の利率です。

比較

IRR 関数は INTRR と似ていますが、IRR 関数では内部利益率がパーセントという点で異なります。

例

当初の支払いが$400 で、次の 3 年間の予定支払い額がそれぞれ$100、$200、$300 の場合、年間内部利益率は次のように表されます。

```
rate=intrr(1,-400,100,200,300);
```

返される値は 0.19437709967 です。

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。

返されるデータの種類: DOUBLE

構文

INTSEAS({*interval*<*multiple*><.*shift-index*>}<, *seasonality*>)

引数

interval
文字列を評価するか、文字列に強制変換でき、WEEK、MONTH、QTR などの間隔名を含む文字定数、変数、または式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

ヒント *Interval* は、大文字または小文字で表示できます。

例 YEAR は、年ベースの間隔を指定します。

multiple
乗数を指定します(省略可能)。基本タイプの間隔の期間に対する倍数と同等の間隔を設定します。

データの種類 DOUBLE

例 YEAR2 は、2 年、つまり 2 年間続く間隔タイプを指定します。

shift-index
シフトインデックスを指定します(省略可能)。間隔の開始時点を指定したサブ期間にシフトします。

制限事項 シフトインデックスは、間隔全体のサブ期間の数以下にする必要があります。たとえば、2 年間隔では 25 番目の月は存在しないため、YEAR2.24 は使用できますが YEAR2.25 はエラーになります。

デフォルトのシフト期間が間隔の種類と同じ場合、複数期間の間隔のみを任意のシフトインデックスでシフトできます。たとえば、MONTH の種類の間隔はデフォルトでは MONTH のサブ期間でシフトされるため、シフトインデックスで月間隔をシフトできません。ただし、各 MONTH2 間隔には 2 つの MONTH 間隔が含まれるため、シフトインデックスで 2 か月間隔をシフトできます。たとえば、間隔名 MONTH2.2 では、偶数月の 1 日目に開始する 2 か月の期間が指定されます。

データの種類 DOUBLE

例 YEAR.3 は、暦年の 3 月 1 日に開始し、翌年の 2 月に終了するようにシフトされた年次期間を指定します。

seasonality
数値を指定します。この引数では、日付と時間の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。

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。

デフォルト 52

データの種類 DOUBLE、CHAR、NCHAR、NVARCHAR、VARCHAR

例 次の例では、*seasonality* 引数により
INTSEAS('MONTH', 'qtr');
関数呼び出しで値 3 が返されます。この関数呼び出しは
INTSEAS('MONTH');
seasonality 引数がないため、値 12 を返します。

詳細

基本

INTSEAS 関数は、季節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。

間隔

間隔は、基本的なものから複雑なものまであります。基本間隔は、SAS が経過時間内にカウントできる測定単位(DAY、MONTH、HOUR など)です。より複雑な間隔を指定するには、基本間隔名に乗数とシフトインデックスを組み合わせて使用できます。

間隔構文は次のとおりです。

interval<multiple><.shift-index>

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。

例

次のステートメントは、INTCYCLE 関数を示しています。

ステートメント	結果
cycle_years = intseas('year'); print cycle_years;	1
cycle_smiyears = intseas('semiyear'); print cycle_smiyears;	2
cycle_quarters = intseas('quarter'); print cycle_quarters;	4
cycle_number = intseas('month', 'qtr'); print cycle_number;	3
cycle_months = intseas('month'); print cycle_months;	12
cycle_smimonths = intseas('semimonth'); print cycle_smimonths;	24
cycle_tendays = intseas('tenday');	36

ステートメント	結果
print cycle_tendays;	
cycle_weeks = intseas('week'); print cycle_weeks;	52
cycle_wkdays = intseas('weekday'); print cycle_wkdays;	5
cycle_hours = intseas('hour'); print cycle_hours;	24
cycle_minutes = intseas('minute'); print cycle_minutes;	60
cycle_month2 = intseas('month2.2'); print cycle_month2;	6
cycle_week2 = intseas('week2'); print cycle_week2;	26
cycle_var1 = intseas(month4.3); print cycle_var1;	3
cycle_day1 = intseas('day1'); print cycle_day1;	7

関連項目

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。

返されるデータの種類: CHAR、NCHAR、NVARCHAR、VARCHAR

構文

INTSHIFT(*interval*<*multiple*><.*shift-index*>)

引数

interval
文字列を評価するか、文字列に強制変換でき、WEEK、MONTH、QTR などの間隔名を含む文字定数、変数、または式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

ヒント *Interval* は、大文字または小文字で表示できます。

例 YEAR は、年間隔を指定します。

multiple
乗数を指定します(省略可能)。基本タイプの間隔の期間に対する倍数と同等の間隔を設定します。

データの種類 DOUBLE

例 YEAR2 は 2 年の期間で構成されます。

shift-index
シフトインデックスを指定します(省略可能)。間隔の開始時点を指定したサブ期間にシフトします。

データの種類 DOUBLE

例 YEAR.3 は、暦年の 3 月 1 日に開始し、翌年の 2 月に終了するようにシフトされた年次期間を指定します。

詳細

基本

INTSHIFT 関数は、ベース間隔に対応するシフト間隔を返します。カスタム間隔の場合、返される値はベースカスタム間隔名です。INTSHIFT は、間隔と間隔シフトの倍数は無視します。

INTSHIFT 関数は、小売業界によるカレンダーの間隔でも使用できます。これらの間隔は ISO 8601 に準拠します。

間隔

間隔は、基本的なものから複雑なものまであります。基本間隔は、SAS が経過時間内にカウントできる測定単位(DAY、MONTH、HOUR など)です。より複雑な間隔を指定するには、基本間隔名に乗数とシフトインデックスを組み合わせて使用できます。

間隔構文は次のとおりです。

interval<multiple><.shift-index>

例

次のステートメントは、INTSHIFT 関数を示しています。

ステートメント	結果
shift1=intshift('year'); print shift1;	MONTH
shift2=intshift('dtyear'); print shift2;	DTMONTH
shift3=intshift('minute'); print shift3;	DTMINUTE
shift4 = intshift('weekdays'); print shift4;	WEEKDAY
shift5=intshift('weekday5.4'); print shift5;	WEEKDAY
shift6=intshift('qtr'); print shift6;	MONTH
shift7=intshift('dttenday'); print 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。

返されるデータの種類: DOUBLE

構文

INTTEST(*interval*<*multiple*><.*shift-index*>)

引数

interval
文字列を評価するか、文字列に強制変換でき、WEEK、MONTH、QTR などの間隔名を含む文字定数、変数、または式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

ヒント　*Interval* は、大文字または小文字で表示できます。

例　YEAR は、年ベースの間隔を指定します。

multiple
乗数を指定します(省略可能)。基本タイプの間隔の期間に対する倍数と同等の間隔を設定します。

データの種類　DOUBLE

例　YEAR2 は、2 年、つまり 2 年間続く間隔タイプを指定します。

shift-index
シフトインデックスを指定します(省略可能)。間隔の開始時点を指定したサブ期間にシフトします。

データの種類　DOUBLE

例　YEAR.3 は、暦年の 3 月 1 日に開始し、翌年の 2 月に終了するようにシフトされた年次期間を指定します。

詳細

基本

INTTEST 関数は、有効な間隔名を確認します。この関数は、*multiple* と *shift-index* の有効な値を確認するときに役立ちます。

INTTEST 関数は、小売業界によるカレンダーの間隔でも使用できます。これらの間隔は ISO 8601 に準拠します。

間隔

間隔は、基本的なものから複雑なものまであります。基本間隔は、SAS が経過時間内にカウントできる測定単位(DAY、MONTH、HOUR など)です。より複雑な間隔を指定するには、基本間隔名に乗数とシフトインデックスを組み合わせて使用できます。

間隔構文は次のとおりです。

interval<multiple><.shift-index>

例

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。

ステートメント	結果
test1 = inttest('month'); print test1;	1
test2 = inttest('week6.13'); print test2;	1
test3 = inttest('tenday'); print test3;	1
test4 = inttest('twoweeks'); print test4;	0
test5 = inttest('hour2.2'); print 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。

返されるデータの種類: DOUBLE

構文

INTZ(*expression*)

引数

expression
数値に評価される有効な式を指定します。

データの種類 DOUBLE

詳細

次の規則が適用されます。

- 引数の値が整数のみの場合、INTZ はその整数を返します。
- 引数が整数ではない正の値の場合、INTZ は引数より小さい最大の整数を返します。

- 引数が整数ではない負の値の場合、INTZ は引数より大きい最小の整数を返します。

比較

INT 関数とは異なり、INTZ 関数はゼロファジーを使用します。引数が整数の 1E-12 内にある場合、INT 関数はその整数に等しくなるように結果をファジー処理します。INTZ 関数は結果をファジー処理しません。そのため、INTZ 関数では予期しない結果になる可能性があります。

例

次のステートメントは、INTZ 関数を示しています。

ステートメント	結果
var1=2.1; a=intz(var1);	2
a=intz(-2.4);	-2
a=intz(1+1.e11);	1
a=intz(-1.6);	-1

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。

返されるデータの種類: DOUBLE

構文

IPMT(*rate*, *period*, *number-of-periods*, *principal-amount*<, *future-amount*><, *type*>)

引数

rate
支払期間ごとの利率を指定します。

データの種類 DOUBLE

period
利息の支払いを計算するための支払い期間を指定します。

要件 *period* は、*number-of-periods* の値以下の正の整数で指定する必要があります。

データの種類 INTEGER

number-of-periods
支払期間の数を指定します。

要件 *number-of-periods* は正の整数にする必要があります。

データの種類 INTEGER

principal-amount
ローンの元金を指定します。

データの種類 DOUBLE

注 欠損値が指定されている場合、ゼロとみなされます。

future-amount
将来の残高を指定します。

データの種類 DOUBLE

注 *future-amount* は、指定した支払い期間数終了後のローンの残高、または定期預金の将来の残高を指定できます。

0 を指定した場合、*future-amount* が省略されたか欠損値が指定されたとみなされます。

type
期首と期末のどちらで支払がなされるのかを指定します。0 は期末の支払を表し、1 は期首の支払を表します。

データの種類 INTEGER

注 *type* が省略された場合、または欠損値が指定されている場合は、ゼロとみなされます。

例

$8,000 のローンで名目年利が 10%、月単位の期末払いが 36 の場合、初回の定期的支払いでの利息の支払いは次のとおりです。

```
InterestPaid1 = ipmt(0.1/12, 1, 36, 8000);
```

計算によって 66.6666666666666 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 という値が返されます。
- InterestPaid3 = ipmt(0.1, 3, 3, 8000);

  計算によって 292.447129909366 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 14.8075736630449 という値が返されます。

IQR 関数

四分位範囲を返します。

返されるデータの種類: DOUBLE

構文

IQR(*expression* <, ...*expression*>)

引数

expression
数値に評価される有効な式を指定します。

データの種類 DOUBLE

詳細

すべての引数に null または欠損値がある場合、ANSI モードまたは SAS モードのどちらであるかに応じて、結果は null または欠損値になります。

それ以外の場合、結果は非 null または非欠損値の四分位範囲になります。四分位範囲の式は、Base SAS UNIVARIATE プロシジャで使用される式と同じです。

例

次のステートメントは、IQR 関数を示しています。

ステートメント	結果
a=iqr(2,4,1,3,999999);	2

IRR 関数

内部利益率をパーセントで返します。

返されるデータの種類: DOUBLE

構文

IRR(*freq*, *c1*, *c2* <..., *cn*>)

引数

freq
目的の内部利益率に関連付けられた指定ベース期間での支払い数を示す数値です。

範囲 *freq* > 0。

データの種類 DOUBLE

ヒント *freq* = 0 は、連続複利計算を許可するフラグです。

c1, *c2*< ..., *cn*>
任意の現金払いを示す数値です。

要件 最小で 2 つの現金払い値が必要です。

データの種類 DOUBLE

詳細

IRR 関数は、現金払いセット *c1*, *c2*, ..., *cn* の指定ベース期間での内部利益率を返します。2 つの連続した支払い間の時間間隔は同じとみなされます。引数 *freq* > 0 は、指定ベース期間で発生する支払い数を表します。各インスタンスから発行される情報の数は制限されます。

比較

IRR 関数は INTRR と似ていますが、IRR 関数では内部利益率がパーセントという点で異なります。

例

当初の支払いが$400 で、次の 3 年間の予定支払い額がそれぞれ$100、$200、$300 の場合、年間内部利益率はパーセントで次のように表されます。

```
rate=irr(1,-400,100,200,300);
```

返される値は 19.437709962747 です。

JBESSEL 関数

ベッセル関数の値を返します。

返されるデータの種類: DOUBLE

構文

JBESSEL(*nu*, *x*)

必須引数

nu
数値の定数、変数または式を指定します。

範囲 $nu \geq 0$

x
数値の定数、変数または式を指定します。

範囲 $x \geq 0$

詳細

JBESSEL 関数は、*x* で評価された次数 *nu* のベッセル関数の値を返します(詳細は Abramowitz and Stegun 1964 および Amos, Daniel, and Weston 1977 を参照)。

例

次のステートメントは、JBESSEL 関数を示しています。

ステートメント	結果
x=jbessel(2, 2);	0.35283402861563

JULDATE 関数

SAS 日付値からユリウス暦の日付を返します。

返されるデータの種類: DOUBLE

構文

JULDATE(*date*)

引数

date
SAS 日付値を表す有効な式を指定します。

データの種類 DOUBLE

詳細

SAS 日付値は、1960 年 1 月 1 日から特定の日付までの日数を表す数値です。JULDATE 関数は、SAS 日付値をユリウス暦の日付に変換します。*date* がシステムオプション YEARCUTOFF=で定義された 100 年の期間内にある場合、結果は 3 桁、4 桁または 5 桁です。5 桁の結果では、最初の 2 桁は年を表し、残りの 3 桁は年間通算日(1 から 365、またはうるう年の場合は 1 から 366)を表します。先頭の 0 は結果から削除されるため、ユリウス暦の日付の年は省略されたり(末尾が 00 の年の場合)、1 桁のみになったり(年の末尾が 01 から 09 の場合)することがあります。それ以外の場合、結果は 7 桁になります。最初の 4 桁は年を表し、残りの 3 桁は年間通算日を表します。

00-09 で終わる年については、Z5.出力形式を使用して 5 桁のユリウス暦の日付をフォーマットできます。

比較

関数 JULDATE7 は JULDATE に似ていますが、JULDATE7 は常に 4 桁の年を返す点で異なります。JULDATE7 は 2 桁の年を考慮する必要がなく、2000 年問題に対応しています。

例

次のステートメントは、JULDATE 関数を示しています。

ステートメント	結果
julian=juldate(mdy(12,31,2013));	7365
julian=print(juldate(mdy(12,31,2013)),z5.);	07365
julian=juldate(mdy(9,1,1999));	99244
julian=juldate(mdy(7,1,1886));	1886182

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。

返されるデータの種類: DOUBLE

構文

JULDATE7(*date*)

引数

date
SAS 日付値を表す有効な式を指定します。

データの種類 DOUBLE

詳細

SAS 日付値は、1960 年 1 月 1 日から特定の日付までの日数を表す数値です。JULDATE7 関数は、SAS 日付値から 7 桁のユリウス暦の日付を返します。最初の 4 桁は年を表し、残りの 3 桁は年間通算日を表します。

比較

関数 JULDATE7 は JULDATE に似ていますが、JULDATE7 は常に 4 桁の年を返す点で異なります。JULDATE7 は 2 桁の年を考慮する必要がなく、2000 年問題に対応しています。

例

次のステートメントは、JULDATE7 関数を示しています。

ステートメント	結果
julian=juldate7(mdy(12,31,2006));	2007365
julian=juldate7(mdy(12,31,2016));	2016366

KURTOSIS 関数

尖度を返します。

返されるデータの種類: DOUBLE

構文

KURTOSIS(*expression-1, expression-2, expression-3, expression-4* <, ...*expression-n*>)

引数

expression
数値に評価される有効な式を指定します。

要件	少なくとも 4 つの非 null または非欠損の引数が必要です。それがない場合は、関数から null または欠損値が返されます。
データの種類	DOUBLE

詳細

尖度は、主に分布の裾の重さの尺度です。尖度の値が大きい場合は、分布の裾が重いことを示しています。

null 値と欠損値は無視され、計算には含まれません。

すべての非 null または非欠損引数の値が等しい場合、尖度は数学的に定義されておらず、KURTOSIS 関数は null または欠損値を返します。

例

次のステートメントは、KURTOSIS 関数を示しています。

ステートメント	結果
a=kurtosis(5,9,3,6);	0.92799999999999
b=kurtosis(5,8,9,6,.);	-3.3
c=kurtosis(8,9,6,1);	1.5
d=kurtosis(8,1,6,1);	-4.48337950138504

LARGEST 関数

k 番目に大きい非 null または非欠損値を返します。

返されるデータの種類:	DOUBLE

構文

LARGEST(*k*, *expression* <, ...*expression*>)

引数

k
数値に評価され、最大の戻り値を表す有効な式を指定します。たとえば、*k* が 2 の場合、LARGEST 関数は、式のリストから 2 番目に大きい値を返します。

データの種類 DOUBLE

expression
数値に評価される検索先の有効な式を指定します。

データの種類 DOUBLE

詳細

k が null または欠損値、0 未満、または値の数よりも大きい場合、結果は null または欠損値になります。そうでない場合は、*k* が非 null または非欠損値の数よりも大きいと、結果は null または欠損値になります。

例

次のステートメントは、LARGEST 関数を示しています。

ステートメント	結果
k=1; largest1=largest(k, 456, 789, .Q, 123);	789
k=2; largest2=largest(k, 456, 789, .Q, 123);	456
k=3; largest3=largest(k, 456, 789, .Q, 123);	123
k=4; largest4=largest(k, 456, 789, .Q, 123);	.

LCM 関数

数の集合に対する最小公倍数を返します。

返されるデータの種類: DOUBLE

構文

LCM(*expression*-1, *expression*-2 <, ...*expression-n*>)

引数

expression
整数に評価される有効な式を指定します。

要件 少なくとも 2 つの引数が必要です。

データの種類 DOUBLE

詳細

最小公倍数は、2 つ以上の数で割り切れる最小の数です。

例

次のステートメントは、LCM 関数を示しています。

ステートメント	結果
a=lcm(1,5,3,0);	0
b=lcm(25,70,85,130);	77350
c=lcm(33,78);	858

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。

返されるデータの種類: DOUBLE

構文

LCOMB(*n*, *r*)

必須引数

n
負でない整数で要素の合計数を表します。サンプルはこの中から選ばれます。

r
負でない整数で選ばれた要素の数を表します。

制限事項 $r \leq n$

比較

LCOMB 関数は、COMB 関数の対数を計算します。

例

次のステートメントは、LCOMB 関数を示しています。

ステートメント	結果
x=lcomb(5000, 500);	1621.44113611415
y=lcomb(100, 10);	30.4823233622786

LEFT 関数

文字式を左揃えにします。

返されるデータの種類: CHAR、NCHAR、NVARCHAR、VARCHAR

構文

LEFT(*expression*)

引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

詳細

LEFT は、先頭の空白を値の末尾に移動した文字列を返します。

例

次のステートメントは、LEFT 関数を示しています。

ステートメント	結果
	----+----1----+-—-2--
a=' END-OF-YEAR'; b=left(a);	END-OF-YEAR

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。

別名: LENGTHN

返されるデータの種類: BIGINT、INTEGER

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。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

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。

例

次のステートメントは、LENGTH 関数を示しています。

ステートメント	結果
a=length('ABCDEF ');	10
b=date(); a=length(b);	5
a=length(' ');	1
a=length(.);	1

1 *b* のデータ型は NCHAR に変換されます。値は 20943 で、その長さは 5 です。
2 null または欠損値(.)のデータ型は DOUBLE から NCHAR に変換されます。値は.で、その長さは 1 です。

LENGTHC 関数

末尾の空白を含めた文字列の長さをバイト単位で返します。

返されるデータの種類: BIGINT、INTEGER

構文

LENGTHC(*expression*)

引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

詳細

固定長変数の場合、LENGTHC は、末尾の空白を含む文字列の文字長を返します。*expression* が固定長変数の場合、LENGTHC が返す値は、宣言された変数の長さと同じになります。*expression* が可変長変数の場合、LENGTHC が返す値は、宣言された変数の長さ以下になります。*expression* の値が欠損していて空白が含まれる場合、LENGTHC は *expression* の空白の数を返します。*expression* が数値式(初期化済みまたは未初期化)の場合、SAS では数値が自動的に右揃えの文字列に変換されます。

比較

- LENGTHC 関数が末尾の空白を含む文字列の長さを返すのに対し、LENGTH 関数は末尾の空白を除いた文字列の長さを返します。LENGTHC は常に LENGTH が返す値以上の値を返します。
- LENGTHC 関数は末尾の空白を含む文字列の長さを返しますが、LENGTHM 関数は文字列に割り当てられたメモリの量(バイト単位)を返します。固定長の文字列の場合、LENGTHC と LENGTHM は常に同じ値を返します。可変長の文字列の場合、LENGTHC は常に LENGTHM で返される値以下の値を返します。ただし、expression 引数がある場合は、LENGHTM が常に null 値を返すのに対して、LENGTHC は文字値の長さを文字数で返します。

例

次のステートメントは、LENGTHC 関数を示しています。

ステートメント	結果
proc cas; a=lengthc('string with trailing blanks '); print "results=" a;	results=31

proc cas;
string1='The power to know. ';
a=lengthc(string1);
print "results=" a;
	results=20

proc cas;
  a=lengthc('');
  print "results=" a;
	results=0

LENGTHM 関数

文字列に割り当て可能なメモリの量をバイト単位で返します。

返されるデータの種類: BIGINT、INTEGER

構文

LENGTHM(*variable*)

引数

variable
有効な文字列変数を指定します。文字列値を評価するか文字列値に強制変換できる式は無効な引数です。

制限事項 変数引数のみが許可されます。式とリテラルは無効であり、null バイト長を返します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

詳細

LENGTHM 関数は、文字変数に割り当て可能なメモリの最大量をバイト単位で返します。これは、NATIONAL、VARYING、CHARACTER SET などの文字データ型修飾子に関係なく当てはまります。

LENGTHM 関数によって返される値は、変数に保存されている現在の値とは無関係であり、現在割り当てられている変数のバイト長よりも大きい場合があります。これは、固定長文字変数と可変長文字変数の両方に当てはまります。たとえば、CHAR(100) CHARACTER SET UTF-8 変数または VARCHAR(100) CHARACTER SET UTF-8 変数がある場合、LENGTHM 関数は、変数に保存されている値に関係なく 400 バイトを返します。

リテラルまたは式の場合、LENGTHM 関数によって返される値は null バイト長であり、引数が関数に対して無効であるという警告が発行されます。

比較

LENGTHM 関数が文字列に割り当てられたメモリ量をバイト単位で返すのに対して、LENGTH および LENGTHC 関数は文字値の長さを文字数で返します。variable

引数がある場合は、LENGTHM は常に LENGTH および LENGTHC によって返される値以上の値を返します。expression 引数がある場合は、LENGHTM が常に null 値を返すのに対して、LENGTH および LENGTHC は文字値の長さを文字数で返します。

例

次のステートメントは、LENGTHM 関数を示しています。

ステートメント	結果
proc cas; a='ABCDEF '; b=lengthm(a); print b;	9
proc cas; string1='the power to know. '; a=lengthm(string1); print a;	20

LENGTHN 関数

末尾の空白を除いた文字列の長さを返します。

別名: LENGTH

返されるデータの種類: BIGINT、INTEGER

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。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

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 関数が末尾の空白を除いた文字列の長さを文字数で返すのに対して、LENGTHM 関数は文字列に割り当てられているメモリ量をバイト単位で返します。LENGTHN は必ず LENGTHM が返す値以下の値を返します。

例

次のステートメントは、LENGTHN 関数を示しています。

ステートメント	結果
c=lengthn(' abc '); print c;	6
d=lengthn('abc '); print d;	3
e=lengthn(18); print e;	2
f=lengthn(' '); print f;	0

LFACT 関数

FACT(階乗)関数の対数を計算します。

返されるデータの種類: DOUBLE

構文

LFACT(*n*)

必須引数

n
整数で要素の合計数を表します。サンプルはこの中から選ばれます。

詳細

LFACT 関数は、FACT 関数の対数を計算します。

例

次のステートメントは、LFACT 関数を示しています。

ステートメント	結果
x=lfact(5000);	37591.1435088767
y=lfact(100);	363.739375555563

LGAMMA 関数

ガンマ関数の自然対数を返します。

返されるデータの種類: DOUBLE

構文

LGAMMA(*expression*)

引数

expression
数値に評価される有効な式を指定します。

要件 正数にする必要があります。

データの種類 DOUBLE

例

次のステートメントは、LGAMMA 関数を示しています。

ステートメント	結果
a=lgamma(2);	0
a=lgamma(1.5);	-0.12078223763524

LOG 関数

数値式の自然対数(底 e)を返します。

返されるデータの種類: DOUBLE

構文

LOG(*expression*)

引数

expression
数値に評価される有効な式を指定します。

要件 正数にする必要があります。

データの種類 DOUBLE

例

次のステートメントは、LOG 関数を示しています。

ステートメント	結果
a=log(1.0);	0

ステートメント	結果
a=log(10.0);	2.30258509299404

LOG10 関数

数値式の 10 を底とする対数を返します。

返されるデータの種類: DOUBLE

構文

LOG10(*expression*)

引数

expression
数値に評価される有効な式を指定します。

要件 正数にする必要があります。

データの種類 DOUBLE

例

次のステートメントは、LOG10 関数を示しています。

ステートメント	結果
a=log10(1.0);	0
a=log10(10.0);	1

LOG1PX 関数

1 に引数を加えた値の対数を返します。

返されるデータの種類: DOUBLE

構文

LOG1PX(*x*)

引数

x
数値定数、変数または式を指定します。

データの種類 DOUBLE

詳細

LOG1PX 関数は、1 に引数を加えた値の対数を計算します。LOG1PX 関数は、次の式(-1 < x の場合)で数学的に定義されます。

$$LOG1PX(x) = log(1 + x)$$

x が 0 に近い場合、LOG1PX(x)は LOG(1+x)より精度が高くなることがあります。

例

例 1: LOG1PX 関数を使用した対数の計算

次の例では、1 に値 0.5 を加えた値の対数を計算します。

```
proc cas;
    x=log1px(0.5);
    print "x=" x;
run;
```

SAS は次の出力をログに書き込みます。

```
x=0.40546510810816
```

例 2: LOG1PX 関数と LOG 関数の比較

次の例では、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。

```
proc cas;
    x=log1px(1.e-5);
    print x hex16.;
    y=log(1+1.e-5);
    print y hex16.;
```

```
run;
```

SAS は次の出力をログに書き込みます。

```
9.99995E-6 hex16.
9.99995E-6 hex16.
```

LOG2 関数

数値式の 2 を底とする対数を返します。

返されるデータの種類: DOUBLE

構文

LOG2(*expression*)

引数

expression
数値に評価される有効な式を指定します。

要件 正数にする必要があります。

データの種類 DOUBLE

例

次のステートメントは、LOG2 関数を示しています。

ステートメント	結果
a=log2(8.0);	3
a=log2(4);	2

LOGBETA 関数

ベータ関数の対数を返します。

返されるデータの種類: DOUBLE

構文

LOGBETA(*a*, *b*)

引数

a
第 1 形状パラメーターです(*a*>0)。

データの種類 DOUBLE

b
第 2 形状パラメーターです(*b*>0)。

データの種類 DOUBLE

詳細

LOGBETA 関数は、式で数学的に求められます。

$$log(\beta(a,b)) = log(\Gamma(a)) + log(\Gamma(b)) - log(\Gamma(a+b))$$

この式で、$\Gamma(.)$はガンマ関数です。

式が計算できない場合、LOGBETA は欠損値を返します。

例

次の DS2 ステートメントでは、ベータ関数の対数が計算されます。第 1 形状パラメーターは 5 で、第 2 形状パラメーターは 3 です。

```
proc cas;
    y=logbeta(5,3);
    print "y=" y;
run;
```

次の行が SAS ログに書き込まれます。

```
y=-4.65396035015752
```

LOGISTIC 関数

引数のロジスティック変換を返します。

構文

LOGISTIC(*argument*)

引数

argument
数値の確率変数の値を指定する、数値変数、定数または式です。*argument* が欠損値または null 値の場合、LOGISTIC 関数は欠損値または null 値を返します。

詳細

LOGISTIC 関数は引数のロジスティック変換を返します。通常は、対数オッズ値を確率尺度の値に変換するために使用されます。この関数は、数学的に次の式で表現されます。

$$logistic = \frac{e^x}{1 + e^x}$$

引数に欠損値が含まれている場合、LOGISTIC 関数は欠損値または null 値を返します。

例

次のステートメントは、LOGISTIC 関数を示しています。

ステートメント	結果
x = LOGISTIC(.5);	0.62245933120185
x = LOGISTIC(7.3);	0.99932491726936

LOWCASE 関数

文字式のすべての文字を小文字に変換します。

別名: LOWER

返されるデータの種類: CHAR、NCHAR、NVARCHAR、VARCHAR

構文

LOWCASE(*expression*)

引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

詳細

LOWCASE 関数は、文字式をコピーし、すべての大文字を小文字に変換して、変更された値を結果として返します。

比較

UPCASE 関数は、引数内のすべての文字を大文字に変換します。LOWCASE 関数は、引数内のすべての文字を小文字に変換します。

例

次のステートメントは、LOWCASE 関数を示しています。

ステートメント	結果
a=lowcase('INTRODUCTION');	introduction

MAD 関数

中央値からの中央絶対偏差を返します。

返されるデータの種類: DOUBLE

構文

MAD(*expression-1*<, ...*expression-n*>)

引数

expression
中央値からの中央絶対偏差を計算する数値に評価される有効な式を指定します。

データの種類 DOUBLE

詳細

すべての引数が欠損値または null 値の場合、結果は欠損値または null 値になります。それ以外の場合は、結果は非欠損値または非 null 値の中央値からの中央絶対偏差になります。中央値の計算式は、UNIVARIATE プロシジャで使用するものと同じです。詳細については、Base SAS プロシジャガイド: 統計プロシジャを参照してください。

例

次のステートメントは、MAD 関数を示しています。

ステートメント	結果
c=mad(2, 4, 1, 3, 5, 999999); print c;	1.5

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。

返されるデータの種類: DOUBLE

構文

MARGRCLPRC(*X_1*, *t*, *X_2*, *sigma1*, *sigma2*, *rho12*)

引数

X_1
第 1 資産の価格を指定する正の非欠損値です。

要件 同じ単位で X_1 と X_2 を指定します。

データの種類 DOUBLE

t
有効期限までの時間を年単位で指定する非欠損値です。

データの種類 DOUBLE

X_2
第 2 資産の価格を指定する正の非欠損値です。

要件 同じ単位で X_2 と X_1 を指定します。

データの種類 DOUBLE

sigma1
第 1 資産のボラティリティを指定する非欠損の正の分数です。

データの種類 DOUBLE

sigma2
第 2 資産のボラティリティを指定する非欠損の正の分数です。

データの種類 DOUBLE

rho12
第 1 資産と第 2 資産の間の相関を指定します。

第 1 資産と第 2 資産の間の相関$\rho_{x_1x_2}$を指定します。

範囲	-1 から 1 まで
データの種類	DOUBLE

詳細

MARGRCLPRC 関数は、Margrabe モデルに基づいて、株式のヨーロピアンオプションのコール価格を計算します。この関数は次の関係に基づきます。

$$\text{CALL} = X_1 N(d_1) - X_2 N(d_2)$$

引数

X_1
第 1 資産の価格を指定します。

X_2
第 2 資産の価格を指定します。

N
累積正規密度関数を指定します。

$$d_1 = \frac{\left(\ln\left(\frac{N_1}{N_2}\right) + \left(\frac{\sigma^2}{2}\right)t\right)}{\sigma\sqrt{t}}$$

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

$$\sigma^2 = \sigma_{x1}^2 + \sigma_{x2}^2 - 2\rho_{x_1,x_2}\sigma_{x1}\sigma_{x2}$$

前述の式には次の引数が適用されます。

t
有効期限までの時間を指定します。

σ_{x1}^2
第 1 資産の分散を指定します。

σ_{x2}^2
第 2 資産の分散を指定します。

σ_{x1}
第 1 資産のボラティリティを指定します。

σ_{x2}
第 2 資産のボラティリティを指定します。

ρ_{x_1, x_2}
第 1 資産と第 2 資産の間の相関を指定します。

t=0 となる特別な場合には、次の式が真です。

$$\text{CALL} = \max((X_1 - X_2), 0)$$

注: この関数は、2 つの資産から利益配当金がないことを想定しています。

比較

MARGRCLPRC 関数は、Margrabe モデルに基づいて、株式のヨーロピアンオプションのコール価格を計算します。MARGRCLPRC 関数は、Margrabe モデルに基づいて、株式のヨーロピアンオプションのプット価格を計算します。これらの関数はスカラー値を返します。

例

次のステートメントは、MARGRCLPRC 関数を示しています。

ステートメント	結果
a=margrclprc(15, .5, 13, .06, .05, 1); print a;	2
b=margrclprc(2, .25, 1, .3, .2, 1); print b;	1

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。

返されるデータの種類: DOUBLE

構文

MARGRPTPRC(*X_1*, *t*, *X_2*, *sigma1*, *sigma2*, *rho12*)

引数

X_1

第 1 資産の価格を指定する正の非欠損値です。

要件 同じ単位で X_1 と X_2 を指定します。

データの種類 DOUBLE

t
有効期限までの時間を年単位で指定する非欠損値です。

データの種類 DOUBLE

X_2
第 2 資産の価格を指定する正の非欠損値です。

要件 同じ単位で X_2 と X_1 を指定します。

データの種類 DOUBLE

sigma1
第 1 資産のボラティリティを 指定する非欠損の正の分数です。

データの種類 DOUBLE

sigma2
第 2 資産のボラティリティを 指定する非欠損の正の分数です。

データの種類 DOUBLE

rho12
第 1 資産と第 2 資産の間の相関を指定します。

第 1 資産と第 2 資産の間の相関$\rho_{x_1y_1}$を指定します。

範囲 -1 から 1 まで

データの種類 DOUBLE

詳細

MARGRCLPRC 関数は、Margrabe モデルに基づいて、株式のヨーロピアンオプションのプット価格を計算します。この関数は次の関係に基づきます。

$$\text{PUT} = X_2 N(pd_1) - X_1 N(pd_2)$$

引数

X_1
第 1 資産の価格を指定します。

X_2
第 2 資産の価格を指定します。

N
累積正規密度関数を指定します。

$$pd_1 = \frac{\left(\ln\left(\frac{N_1}{N_2}\right) + \left(\frac{\sigma^2}{2}\right)t\right)}{\sigma\sqrt{t}}$$

$$pd_2 = pd_1 - \sigma\sqrt{t}$$

$$\sigma^2 = \sigma_{x1}^2 + \sigma_{x2}^2 - 2\rho_{x_1,\,x_2}\sigma_{x1}\sigma_{x2}$$

前述の式には次の引数が適用されます。

t
有効期限までの時間を年単位で指定する非欠損値です。

σ_{x1}^2
第 1 資産の分散を指定します。

σ_{x2}^2
第 2 資産の分散を指定します。

σ_{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。

$\rho_{x_1,\,x_2}$
第 1 資産と第 2 資産の間の相関を指定します。

t=0 となる特別な場合には、次の式が真です。

$$\text{PUT} = \max((X_2 - X_1), 0)$$

注: この関数は、2 つの資産から利益配当金がないことを想定しています。

比較

MARGRCLPRC 関数は、Margrabe モデルに基づいて、株式のヨーロピアンオプションのプット価格を計算します。MARGRCLPRC 関数は、Margrabe モデルに基づいて、株式のヨーロピアンオプションのコール価格を計算します。これらの関数はスカラー値を返します。

例

次のステートメントは、MARGRPTPRC 関数を示しています。

ステートメント	結果
a=margrptprc(2, .25, 3, .06, .2, 1); print a;	1.00000000009729

ステートメント	結果
b=margrptprc(3, .25, 4, .05, .3, 1); print b;	1.00157624907711

MAX 関数

引数のリストから最大値を返します。

返されるデータの種類: BIGINT、DECIMAL、DOUBLE

構文

MAX(*expression-1, expression-2* <, ...*expression-n*>)

引数

expression
数値に評価される有効な式です。

要件 少なくとも 2 つの引数が必要です。

データの種類 BIGINT、DECIMAL、DOUBLE

詳細

この関数の引数が数値以外の場合、その引数は DOUBLE に変換されます。いずれかの引数が DOUBLE または REAL の場合、すべての引数が(まだ変換されていなければ) DOUBLE に変換され、結果は DOUBLE になります。それ以外の場合で、いずれかの引数が DECIMAL のときは、すべての引数が(まだ変換されていなければ) DECIMAL に変換され、結果は DECIMAL になります。それ以外の場合は、すべての引数が BIGINT に変換され、結果は BIGINT になります。

比較

MAX 関数は、引数のリストから最大値を返します。MAX 演算子(<>)は、2 つのオペランドのうち大きい方を返します。

MAX 関数は、すべての引数が null または欠損の場合にのみ、null 値または欠損値を返します。MAX 演算子(<>)は、両方のオペランドが null または欠損の場合にのみ、

null 値または欠損値を返します。この場合、null 値または欠損値の並べ替え順がより高いオペランドの値が返されます。

例

次のステートメントは、MAX 関数を示しています。

ステートメント	結果
x=max(8,3);	8
x=max(2, 6, .);	6
x=max(2, -3, 1, -1);	2
x=max(3, ., -3);	3

MDY 関数

月、日および年の値から SAS 日付値を返します。

返されるデータの種類: DOUBLE

構文

MDY(*month*, *day*, *year*)

引数

month
1 から 12 までの整数を表す数値式を指定します。

データの種類 DOUBLE

day
1 から 31 までの整数を表す数値式を指定します。

データの種類 DOUBLE

year
2 桁または 4 桁の年を表す数値式を指定します。YEARCUTOFF=システムオプションは、2 桁の日付の年の値を定義します。

データの種類　DOUBLE

例

次のステートメントは、MDY 関数を示しています。

注: CASL は、誕生日を SAS 日付値として出力します。

ステートメント	結果

proc cas;
  mn=8;
  dy=27;
  yr=12;
  birthday=mdy(mn, dy, yr);
  print "birthday=" birthday;
run;
	birthday=19232

proc cas;
  mn=7;
  dy=11;
  yr=12;
  anniversary=mdy(mn, dy, yr);
  print "anniversary=" anniversary;
run;
	anniversary=19185

MEAN 関数

非 null または非欠損引数の算術平均(平均)を返します。

返されるデータの種類:　DOUBLE

構文

MEAN(*expression-1*<, ...*expression-n*>)

引数

expression
　数値に評価される有効な式を指定します。

要件　少なくとも 1 つの非 null または非欠損の引数が必要です。それがない場合は、関数から null または欠損値が返されます。

データの種類　DOUBLE

比較

GEOMEAN 関数が幾何平均を返し、HARMEAN 関数が調和平均を返すのに対し、MEAN 関数は算術平均(平均)を返します。

例

次のステートメントは、MEAN 関数を示しています。

ステートメント	結果
a=mean(2,.,.,6);	4
a=mean(1,2,3,2);	2

MEDIAN 関数

中央値を返します。

カテゴリ:　CAS

返されるデータの種類:　DOUBLE

構文

MEDIAN(*expression-1*<, ...*expression-n* >)

引数

expression
数値に評価される有効な式を指定します。

データの種類　DOUBLE

詳細

MEDIAN 関数は、非欠損値または非 null 値の中央値を返します。すべての引数が欠損値または null 値の場合、結果は欠損値または null 値になります。

注: MEDIAN 関数で使用される式は、Base SAS プロシジャガイド: 統計プロシジャに説明されている、PROC UNIVARIATE で使用される式と同じです。

比較

MEDIAN 関数が非欠損値または非 null 値の中央値を返すのに対し、MEAN 関数は算術平均(平均)を返します。

例

次のステートメントは、MEDIAN 関数を示しています。

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

MIN 関数

最小値を返します。

返されるデータの種類: BIGINT、DECIMAL、DOUBLE

構文

MIN(*expression-1, expression-2* <, ...*expression-n*>)

引数

expression
数値に評価される有効な式を指定します。

要件	少なくとも 2 つの引数が必要です。
データの種類	BIGINT、DECIMAL、DOUBLE

詳細

この関数の引数が数値以外の場合、その引数は DOUBLE に変換されます。いずれかの引数が DOUBLE または REAL の場合、すべての引数が(まだ変換されていなければ) DOUBLE に変換され、結果は DOUBLE になります。それ以外の場合で、いずれかの引数が DECIMAL のときは、すべての引数が(まだ変換されていなければ) DECIMAL に変換され、結果は DECIMAL になります。それ以外の場合は、すべての引数が BIGINT に変換され、結果は BIGINT になります。

比較

MIN 関数は、値のリストから最小値を返します。MIN 演算子(><)は、2 つのオペランドのうち小さい方を返します。

MIN 関数は、すべての引数が null または欠損の場合にのみ、null 値または欠損値を返します。MIN 演算子は、オペランドのどちらかが null または欠損の場合にのみ、null 値または欠損値を返します。この場合、null 値または欠損値の並べ替え順がより低いオペランドの値が返されます。

例

次のステートメントは、MIN 関数を示しています。

ステートメント	結果
a=min(2,.,6);	2
a=min(2,-3,1,-1);	-3

MINUTE 関数

SAS 時間または日付値から分を返します。

返されるデータの種類:	DOUBLE

構文

MINUTE(*time* | *datetime*)

引数

time
SAS 時間値を表す有効な式を指定します。

データの種類 DOUBLE

datetime
SAS 日時値を表す有効な式を指定します。

データの種類 DOUBLE

詳細

MINUTE 関数は、特定の分を表す整数を返します。MINUTE は必ず 0 から 59 までの範囲の正の値を返します。null 値または欠損値は無視されます。

例

次のステートメントは、MINUTE 関数を示しています。

ステートメント	結果
a=minute(time());	19

MISSING 関数

引数に欠損値が含まれるかどうかを示す数字を返します。

返されるデータの種類: INTEGER

構文

MISSING(*expression*)

引数

expression
値に評価される有効な式を指定します。

データの種類 すべてのデータ型

詳細

- MISSING 関数は、値が null 値または欠損値かどうかを確認し、結果を数値で返します。引数に欠損値が含まれていない場合、SAS は値 0 を返します。引数に欠損値が含まれている場合、SAS は値 1 を返します。
- SAS モードでは、空白で埋められた文字値は、SAS 欠損値として定義されています。ANSI モードでは、空白で埋められた文字値は、非欠損および非 null として定義されています。
- SAS モードでは、DOUBLE 値は SAS 欠損値(.、.A から Z)である可能性があります。その他の数値型では、SAS 欠損値はサポートされていません。
- パッケージインスタンスが存在しない場合、MISSING 関数は 1 を返します。つまり、パッケージ変数は欠損しているパッケージ参照です。パッケージ変数がパッケージインスタンスを参照している場合、MISSING 関数は 0 を返します。

比較

MISSING 関数には引数を 1 つのみ指定できます。NMISS 関数には数値引数が必要で、引数リスト内の欠損値の数が返されます。

例: MISSING 関数の使用

次の例は、MISSING 関数を示しています。

```
proc cas;
 a[1]=2;
 a[2]=4;
 a[3]=.;
 do i=1 to 3;
   if missing (a[i]) then put 'Missing';
     else print 'Not Missing';
     end;
   end;
run;
```

次の行が SAS ログに書き込まれます。

```
Not Missing
Not Missing
Missing
```

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。

返されるデータの種類: DOUBLE

構文

MOD(*dividend-expression*, *divisor-expression*)

引数

dividend-expression
数値に評価される有効な式である被除数を指定します。

データの種類 DOUBLE

divisor-expression
数値に評価される有効な式である除数を指定します。

制限事項 *divisor-expression* を 0 にすることはできません

データの種類 DOUBLE

詳細

MOD 関数は、*dividend-expression* を *divisor-expression* で除算したときの余りを返します。結果がゼロ以外のとき、結果は第 1 引数と同じ符号になります。第 2 引数の符号は無視されます。

次の条件の両方を満たす場合、MOD 関数で実行される計算は厳密になります。

- 両方の引数が厳密な整数である。
- すべての整数が、厳密に 8 バイトの浮動小数点で表されるいずれかの引数より小さい。

前述の条件のいずれかを満たしていない場合、浮動小数点計算で小さな数値誤差が発生する可能性があります。この場合、次のようになります。

- 余りが 0 か第 2 引数の値にとても近いと、MOD はゼロを返す。
- 約 3 桁以上の精度で余りを計算できなければ、MOD は null 値または欠損値を返す。この場合、ログにエラーメッセージが書き込まれる。

比較

MOD 関数と MODZ 関数の比較を次に示します。

- MOD 関数は、ファジーと呼ばれる追加の計算を実行し、厳密なゼロを返す(ファジーを実行しなければ数値誤差のため結果はゼロにならない)。
- MODZ 関数はファジーを実行しない。
- 約 3 桁以上の精度で余りを計算できなければ、MOD 関数と MODZ 関数はどちらも null 値または欠損値を返します。

例

次のステートメントは、MOD 関数を示しています。

ステートメント	結果
a=mod(10,3);	1
a=mod(.35,-.1);	0.05
a=mod(17,3);	2
a=mod(.3,-.9);	0.3

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。

返されるデータの種類: DOUBLE

構文

MODZ(*dividend-expression*, *divisor-expression*)

引数

dividend-expression
数値に評価される有効な式である被除数を指定します。

データの種類 DOUBLE

divisor-expression
数値に評価される有効な式である除数を指定します。

制限事項 *divisor-expression* を 0 にすることはできません

データの種類 DOUBLE

詳細

MODZ 関数は、*dividend-expression* を *divisor-expression* で除算したときの余りを返します。結果がゼロ以外のとき、結果は第 1 引数と同じ符号になります。第 2 引数の符号は無視されます。

次の条件の両方を満たす場合、MODZ 関数で実行される計算は厳密になります。

- 両方の引数が厳密な整数である。
- すべての整数が、厳密に 8 バイトの浮動小数点で表されるいずれかの引数より小さい。

前述の条件のいずれかを満たしていない場合、浮動小数点計算で小さな数値誤差が発生する可能性があります。たとえば、厳密算術を使用して結果がゼロのとき、MODZ は正の極小値または第 2 引数より若干小さい値を返す場合があります。

比較

MODZ 関数と MOD 関数の比較を次に示します。

- MODZ 関数はファジーを実行しない。
- MOD 関数は、ファジーを実行し、厳密なゼロを返す(ファジーを実行しなければ数値誤差のため結果はゼロにならない)。
- 約 3 桁以上の精度で余りを計算できなければ、MODZ 関数と MOD 関数はどちらも null 値または欠損値を返します。

例

次のステートメントは、MOD 関数と MODZ 関数の違いを示しています。

ステートメント	結果
a=mod(10,3); b=modz(10,3);	1 1
a=mod(.35,-.1); b=modz(.35,-.1);	0.05 0.4999999999999
a=mod(17,3);	2 2

ステートメント	結果
b=modz(17,3)	
a=mod(.3,-.9); b=modz(.3,-.9);	0.3 0.3

MONTH 関数

SAS 日付値から月を表す数字を返します。

返されるデータの種類: DOUBLE

構文

MONTH(*date*)

引数

date
SAS 日付値を表す有効な式を指定します。

範囲 1-12

データの種類 DOUBLE

例

次のステートメントは、11 月の場合の MONTH 関数を示しています。

ステートメント	結果
a=month(date());	11

MORT 関数

割賦返済パラメーターを返します。

返されるデータの種類: DOUBLE

構文

MORT(*a*, *p*, *r*, *n*)

引数

a
初期の金額に評価される有効な式を指定します。

データの種類 DOUBLE

p
定期的支払いに評価される有効な式を指定します。

データの種類 DOUBLE

r
分数で表す定期的な利率に評価される有効な式を指定します。

データの種類 DOUBLE

n
複利期間数に評価される有効な式を指定します。

範囲 $n \geq 0$

データの種類 DOUBLE

詳細

結果の計算

MORT 関数は毎期間複利計算される固定利率の割賦返済計算から 4 つの引数のリスト内の欠損引数を返します。これらの引数には次の式の関係があります。

$$p = \frac{ar(1+r)^n}{(1+r)^n - 1}$$

引数は 1 つを欠損値とする必要があります。値は残りの 3 つから計算されます。結果を変換して丸めた数字にする調整は行われません。

結果を計算するときの制限事項

次の引数の組み合わせのいずれかに該当する場合、MORT 関数は無効な引数というメモを SAS ログに返し、_ERROR_を 1 に設定します。

- rate < –1 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

例

次の例では、$50,000 を毎月複利計算される年利 10%の 30 年で借入します。

```
proc cas;
 payment=mort(50000, . , .10/12, 30*12);
 print payment;
run;
```

返される値は 438.79(丸め値)です。第 2 引数は欠損に設定されています。これは定期的支払いが計算されることを示します。名目年利 10%は、月利 0.10/12 に変換されています。rate は分数(パーセントではない)で表す複利計算期間当たりの利率です。30 年は 360 か月に変換されます。

N 関数

非 null または非欠損の数値の数を返します。

返されるデータの種類: INTEGER

構文

N(*expression* <, ...*expression*>)

引数

expression
数値に評価される有効な式を指定します。

要件 少なくとも 1 つの引数が必要です。

データの種類 DECIMAL、DOUBLE、NUMERIC

詳細

null 値は欠損値に変換され、欠損値として数えられます。

比較

N 関数は非 null 値と非欠損値を数えますが、NMISS 関数は欠損値を数えます。N 関数には数値引数が必要です。

例

次のステートメントは、N 関数を示しています。

ステートメント	結果
a=n(1,0,.,2,5,.);	4
a=n(1,2);	2

NETPV 関数

現在正味価値をパーセントで返します。

返されるデータの種類: DOUBLE

構文

NETPV(*r*, *freq*, *c0, c1, ..., cn*)

引数

r
指定したベース期間の利率を分数で表した数値。

範囲 *r* >= 0

データの種類 DOUBLE

freq
利率 *r* で指定したベース期間中の支払い回数を表す数値。

範囲 *freq* > 0

データの種類 DOUBLE

注 *freq* = 0 は、連続割引を許可するフラグです。

c0, c1 ,..., cn
時間 0,1,...n に発生する現金支出(支払い)または現金流入(収入)を表す数値のキャッシュフローです。これらのキャッシュフローは、等間隔の期間開始時の値であると想定されます。負の値は支払い、正の値は収入、値 0 はその時点でキャッシュフローがないことを表します。*c0* 引数と *c1* 引数は必須です。

データの種類 DOUBLE

詳細

NETPV 関数は、指定したベース期間中に利率 *r* を使用して、一連の現金支払い *c0*,*c1*, ...,*cn* に対する時間 0 の正味現在価値を返します。引数 *freq*>0 は、指定ベース期間で発生する支払い数を表します。

正味現在価値は次の式で求められます。

$$\mathrm{NETPV}(r, freq, c_0, c_1, ..., c_n) = \sum_{i=0}^{n} c_i x^i$$

前述の式には次の関係が適用されます。

$$x = \begin{cases} \frac{1}{(1+r)^{(1/freq)}} & freq > 0 \\ \varepsilon^{-r} & freq = 0 \end{cases}$$

支払いの欠損値は 0 値として扱われます。*freq*>0 のとき、利率 *r* は指定したベース期間の実効金利です。四半期金利 4%(ベース期間は 3 か月)の毎月現金支払いで計算するには、*freq* を 3、*r* を.04 に設定します。

freq が 0 の場合は、連続割引とみなされます。ベース期間は 2 つの連続する支払い間の時間間隔で、利率 *r* は名目金利です。

名目年利 11%、連続割引、毎月支払いで計算するには、*freq* を 0、*r* を.11/12 に設定します。

例

翌 6 年間にわたり$200、$300、$400 の年 2 回支払いおよび年間割引率 10%を返す初期投資$500 では、投資の現在正味価値は次のように表すことができます。

```
proc cas;
 value=netpv(.10, .5, -500, 200, 300, 400);
 print value;
run;
```

返される値は 95.982864829379 です。

関連項目

関数:

- “NPV 関数” (*SAS DS2 言語リファレンス*)

NMISS 関数

null および SAS 欠損数値の数を返します。

返されるデータの種類: INTEGER

構文

NMISS(*expression* <, ...*expression*>)

引数

expression
数値に評価される有効な式を指定します。

要件 少なくとも 1 つの引数が必要です。

データの種類 DECIMAL、DOUBLE、NUMERIC

詳細

null 値は SAS 欠損値に変換され、欠損値として数えられます。

比較

NMISS 関数が null 値または SAS 欠損値の数を返すのに対し、N 関数は非 null および非欠損値の数を返します。NMISS が数値を必要とし、複数の数値を使用できるのに対し、MISSING は数値か文字のいずれかの値を 1 つだけ使用できます。

例

次のステートメントは、NMISS 関数を示しています。

ステートメント	結果
a=nmiss(1,0,.,2,5,.);	2
a=nmiss(1,0);	0

NOMRATE 関数

名目年利を返します。

返されるデータの種類: DOUBLE

構文

NOMRATE(*compounding-interval*, *rate*)

引数

compounding-interval
SAS 間隔です。この値は、返される値が複利計算される頻度を表します。

データの種類 CHAR

rate
数値です。*rate* は各期間複利計算される実効年利(パーセント)です。

データの種類 DOUBLE

詳細

NOMRATE 関数は名目年利を返します。NOMRATE は実効年利に対応する名目年利を計算します。

次の詳細は NOMRATE 関数に適用されます。

- rate の値は–99 以上にする必要があります。
- 実効金利と複利計算間隔を検討するとき、*compounding-interval* が 'CONTINUOUS'であれば、NOMRATE によって返される値は $\log_e(1+[rate/100])$に等しくなります。

  compounding-interval が'CONTINUOUS'でなく、1 年に *m* 個の間隔が発生する場合、NOMRATE によって返される値は次に等しくなります。

$$m\left((1+\frac{rate}{100})^{\frac{1}{m}}-1\right)$$

- *compounding-interval* で有効な値は次のとおりです。
  - 'CONTINUOUS'
  - 'DAY'
  - 'SEMIMONTH'
  - 'MONTH'
  - 'QUARTER'
  - 'SEMIYEAR'
  - 'YEAR'
- 間隔が'DAY'の場合、*m*=365 です。

例

- 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);

NOTALNUM 関数

文字列から英数字でない文字を検索し、その文字が見つかった最初の文字位置を返します。

返されるデータの種類: DOUBLE

構文

NOTALNUM('*expression*'<, *start*>)

引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、VARCHAR、NVARCHAR

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

NOTALNUM 関数の結果は、有効になっている変換テーブルに直接依存し、システムオプションに間接的に依存します。

NOTALNUM 関数は、文字列から数字、大文字または小文字でない文字の最初の出現を検索します。対象の文字が検出されると、NOTALNUM は文字列内の該当文字の位置を返します。対象の文字が検出されない場合、NOTALNUM は値 0 を返します。

引数を 1 つのみ使用すると、NOTALNUM は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

次のいずれかの条件が満たされると、NOTALNUM は値 0 を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

NOTALNUM 関数は、文字列から英数字でない文字を検索します。ANYALNUM 関数は、文字列から英数字を検索します。

例

次の例では、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。

```
proc cas;
   string='Next = Last + 1;';
   j=1;
   do until(j=0);
     j=notalnum(string, j+1);
     if j=0 then put 'The end';
```

```
        else do;
          c=substr(string, j, 1);
          print  "j=" j "c=" c;
        end;
      end;
  run;
```

SAS は次の出力をログに書き込みます。

```
j=5 c==
j=10 c=+
j=12 c=;
The End
```

NOTALPHA 関数

文字列からアルファベットでない文字を検索し、その文字が見つかった最初の文字位置を返します。

返されるデータの種類: DOUBLE

構文

NOTALPHA('*expression*'<, *start*>)

引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、VARCHAR、NVARCHAR

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

NOTALPHA 関数は、文字列から大文字または小文字でない文字の最初の出現を検索します。対象の文字が検出されると、NOTALPHA は文字列内の該当文字の位置を返します。対象の文字が検出されない場合、NOTALPHA は値 0 を返します。

引数を 1 つのみ使用すると、NOTALPHA は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

次のいずれかの条件が満たされると、NOTALPHA は値 0 を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

NOTALPHA 関数は、文字列からアルファベットでない文字を検索します。
ANYALPHA 関数は、文字列からアルファベットを検索します。

例: 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。

```
proc cas;
    string='Next = _n_ + 12E3;';
    j=1;
    do until(j=0);
      j=notalpha(string, j+1);
      if j=0 then print 'The end';
      else do;
        c=substr(string, j, 1);
        print "j=" j "c=" c;
      end;
    end;
run;
```

SAS は次の出力をログに書き込みます。

```
j=5 c=
j=6 c==
j=7 c=
j=8 c=_
j=10 c=_
j=11 c=
j=12 c=+
j=13 c=
j=14 c=1
j=15 c=2
j=17 c=3
j=18 c=;
The end
```

NOTCNTRL 関数

文字列から制御文字でない文字を検索し、その文字が見つかった最初の文字位置を返します。

返されるデータの種類: DOUBLE

構文

NOTCNTRL('*expression*'<, *start*>)

引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、VARCHAR、NVARCHAR

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

NOTCNTRL 関数は、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。

NOTDIGIT 関数

文字列から数字でない文字を検索し、その文字が見つかった最初の文字位置を返します。

返されるデータの種類: DOUBLE

構文

NOTDIGIT('*expression*'<, *start*>)

引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、VARCHAR、NVARCHAR

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

NOTDIGIT 関数は、文字列から数字でない文字の最初の出現を検索します。対象の文字が検出されると、NOTDIGIT は文字列内の該当文字の位置を返します。対象の文字が検出されない場合、NOTDIGIT は値 0 を返します。

引数を 1 つのみ使用すると、NOTDIGIT は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

次のいずれかの条件が満たされると、NOTDIGIT は値 0 を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

NOTDIGIT 関数は、文字列から数字でない文字を検索します。ANYDIGIT 関数は、文字列から数字を検索します。

例

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。

```
proc cas;
   string='Next = _n_ + 12E3;';
    j=1;
    do until(j=0);
      j=notdigit(string, j+1);
      if j=0 then print 'The end';
      else do;
        c=substr(string, j, 1);
        print "j=" j;
        print "c=" c;
      end;
    end;
run;
```

SAS は次の出力をログに書き込みます。

```
j=2 c=e
j=3 c=x
j=4 c=t
j=5 c=
j=6 c==
j=7 c=
j=8 c=_
j=9 c=n
j=10 c=_
j=11 c=
j=12 c=+
j=13 c=
j=16 c=E
j=18 c=;
The end
```

NOTFIRST 関数

VALIDVARNAME=V7 の SAS 変数名として無効な開始文字を文字列から検索し、その文字が見つかった最初の文字位置を返します。

返されるデータの種類: DOUBLE

構文

NOTFIRST('*expression*'<, *start*>)

引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、VARCHAR、NVARCHAR

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

NOTFIRST 関数は、TRANTAB、ENCODING または LOCALE システムオプションに依存しません。

NOTFIRST 関数は、VALIDVARNAME=V7 の SAS 変数名の開始文字として有効でない文字のうち最初に出現する文字を文字列から検索します。これらの文字は、アンダ

ースコア(_)、大文字または小文字の英文字以外の文字です。対象の文字が検出されると、NOTFIRST は文字列内の該当文字の位置を返します。対象の文字が検出されない場合、NOTFIRST は値 0 を返します。

引数を 1 つのみ使用すると、NOTFIRST は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

次のいずれかの条件が満たされると、NOTFIRST は値 0 を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

NOTFIRST 関数は、VALIDVARNAME=V7 の SAS 変数名の開始文字として有効でない文字のうち最初に出現する文字を文字列から検索します。ANYFIRST 関数は、VALIDVARNAME=V7 の SAS 変数名の開始文字として有効な文字のうち最初に出現する文字を文字列から検索します。

例

次の例では、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。

```
proc cas;
   string='Next = _n_ + 12E3;';
    j=1;
    do until(j=0);
      j=notfirst(string, j+1);
      if j=0 then print 'The end';
      else do;
        c=substr(string, j, 1);
        print "j=" j;
        print "c=" c;
      end;
    end;
run;
```

SAS は次の出力をログに書き込みます。

```
j=5 c=
j=6 c==
j=7 c=
j=11 c=
j=12 c=+
j=13 c=
j=14 c=1
j=15 c=2
j=17 c=3
j=18 c=;
The end
```

NOTGRAPH 関数

文字列からグラフィカル文字でない文字を検索し、その文字が見つかった最初の文字位置を返します。

返されるデータの種類: DOUBLE

構文

NOTGRAPH('*expression*'<, *start*>)

引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、VARCHAR、NVARCHAR

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

NOTGRAPH 関数は、文字列からグラフィカル文字でない文字の最初の出現を検索します。グラフィカル文字は、空白以外の印刷可能文字として定義されます。対象の文字が検出されると、NOTGRAPH は文字列内の該当文字の位置を返します。対象の文字が検出されない場合、NOTGRAPH は値 0 を返します。

引数を 1 つのみ使用すると、NOTGRAPH は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

次のいずれかの条件が満たされると、NOTGRAPH は値 0 を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

NOTGRAPH 関数は、文字列からグラフィカル文字でない文字を検索します。
ANYGRAPH 関数は、文字列からグラフィカル文字を検索します。

例: 文字列からグラフィカルでない文字を検索

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。

```
proc cas;
    string='Next = _n_ + 12E3;';
    j=1;
    do until(j=0);
      j=notgraph(string, j+1);
      if j=0 then print 'The end';
      else do;
        c=substr(string, j, 1);
        print "j=" j;
        print "c=" c;
      end;
    end;
run;
```

SAS は次の出力をログに書き込みます。

```
j=5 c=
j=7 c=
j=11 c=
j=13 c=
The end
```

NOTLOWER 関数

文字列から小文字でない文字を検索し、その文字が見つかった最初の文字位置を返します。

返されるデータの種類: DOUBLE

構文

NOTLOWER('*expression*'<, *start*>)

引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、VARCHAR、NVARCHAR

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

NOTLOWER 関数は、文字列から小文字でない文字の最初の出現を検索します。対象の文字が検出されると、NOTLOWER は文字列内の該当文字の位置を返します。対象の文字が検出されない場合、NOTLOWER は値 0 を返します。

引数を 1 つのみ使用すると、NOTLOWER は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

次のいずれかの条件が満たされると、NOTLOWER は値 0 を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

NOTLOWER 関数は、文字列から小文字でない文字を検索します。ANYLOWER 関数は、文字列から小文字を検索します。

例

次の例では、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。

```
proc cas;
    string='Next = _n_ + 12E3;';
    j=1;
    do until(j=0);
      j=notlower(string, j+1);
      if j=0 then print 'The end';
      else do;
        c=substr(string, j, 1);
        print "j=" j;
        print "c=" c;
      end;
    end;
run;
```

SAS は次の出力をログに書き込みます。

```
j=5 c=
j=6 c==
j=7 c=
j=8 c=_
j=10 c=_
j=11 c=
j=12 c=+
j=13 c=
j=14 c=1
j=15 c=2
j=16 c=E
j=17 c=3
j=18 c=;
The end
```

NOTNAME 関数

VALIDVARNAME=V7 の SAS 変数名として無効な文字を文字列から検索し、その文字が見つかった最初の文字位置を返します。

返されるデータの種類: DOUBLE

構文

NOTNAME('*expression*'<, *start*>)

引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、VARCHAR、NVARCHAR

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

NOTNAME 関数は、TRANTAB、ENCODING または LOCALE システムオプションに依存しません。

NOTNAME 関数は、VALIDVARNAME=V7 の SAS 変数名で有効でない文字のうち最初に出現する文字を文字列から検索します。これらの文字は、アンダースコア(_)、数字、大文字または小文字の英文字以外の文字です。対象の文字が検出されると、NOTNAME は文字列内の該当文字の位置を返します。対象の文字が検出されない場合、NOTNAME は値 0 を返します。

引数を 1 つのみ使用すると、NOTNAME は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

次のいずれかの条件が満たされると、NOTNAME は値 0 を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

NOTNAME 関数は、VALIDVARNAME=V7 の SAS 変数名で有効でない文字のうち最初に出現する文字を文字列から検索します。ANYNAME 関数は、VALIDVARNAME=V7 の SAS 変数名で有効な文字のうち最初に出現する文字を文字列から検索します。

例

次の例では、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。

```
proc cas;
    string='Next = _n_ + 12E3;';
    j=1;
    do until(j=0);
      j=notname(string, j+1);
      if j=0 then print 'The end';
      else do;
        c=substr(string, j, 1);
        print "j=" j;
        print "c=" c;
      end;
    end;
run;
```

SAS は次の出力をログに書き込みます。

```
j=5 c=
j=6 c==
j=7 c=
j=11 c=
j=12 c=+
j=13 c=
j=18 c=;
The end
```

NOTPRINT 関数

文字列から印刷不可文字を検索し、その文字が見つかった最初の文字位置を返します。

返されるデータの種類: DOUBLE

構文

NOTPRINT('*expression*'<, *start*>)

引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、VARCHAR、NVARCHAR

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

NOTPRINT 関数は、文字列で最初に出現する印刷不可文字を検索します。対象の文字が検出されると、NOTPRINT は文字列内の該当文字の位置を返します。対象の文字が検出されない場合、NOTPRINT は値 0 を返します。

引数を 1 つのみ使用すると、NOTPRINT は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

次のいずれかの条件が満たされると、NOTPRINT は値 0 を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

NOTPRINT 関数は、文字列から印刷不可文字を検索します。ANYPRINT 関数は、文字列から印刷可能文字を検索します。

NOTPUNCT 関数

文字列から句読文字でない文字を検索し、その文字が見つかった最初の文字位置を返します。

返されるデータの種類: DOUBLE

構文

NOTPUNCT('*expression*'<, *start*>)

引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、VARCHAR、NVARCHAR

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

NOTPUNCT 関数は、文字列から句読文字でない文字の最初の出現を検索します。対象の文字が検出されると、NOTPUNCT は文字列内の該当文字の位置を返します。対象の文字が検出されない場合、NOTPUNCT は値 0 を返します。

引数を 1 つのみ使用すると、NOTPUNCT は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

次のいずれかの条件が満たされると、NOTPUNCT は値 0 を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

NOTPUNCT 関数は、文字列から句読文字でない文字を検索します。ANYPUNCT 関数は、文字列から句読文字を検索します。

例: 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。

```
proc cas;
   string='Next = _n_ + 12E3;';
   j=1;
```

```
    do until(j=0);
      j=notpunct(string, j+1);
      if j=0 then print 'The end';
      else do;
        c=substr(string, j, 1);
        print "j=" j;
        print "c=" c;
      end;
    end;
run;
```

SAS は次の出力をログに書き込みます。

```
j=2 c=e
j=3 c=x
j=4 c=t
j=5 c=
j=6 c==
j=7 c=
j=9 c=n
j=11 c=
j=12 c=+
j=13 c=
j=14 c=1
j=15 c=2
j=16 c=E
j=17 c=3
The end
```

NOTSPACE 関数

空白文字ではない文字(空白、水平タブと垂直タブ、キャリッジリターン、改行、およびフォームフィード)を文字列で検索し、その文字が見つかった最初の文字位置を返します。

返されるデータの種類: DOUBLE

構文

NOTSPACE('*expression*'<, *start*>)

引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、VARCHAR、NVARCHAR

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類　DOUBLE

詳細

NOTSPACE 関数は、文字列から空白、水平タブ、垂直タブ、キャリッジリターン、ラインフィード、フォームフィードでない文字の最初の出現を検索します。対象の文字が検出されると、NOTSPACE は文字列内の該当文字の位置を返します。対象の文字が検出されない場合、NOTSPACE は値 0 を返します。

引数を 1 つのみ使用すると、NOTSPACE は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

次のいずれかの条件が満たされると、NOTSPACE は値 0 を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

NOTSPACE 関数は、文字列で最初に出現する空白、水平タブ、垂直タブ、キャリッジリターン、ラインフィード、フォームフィードでない文字を検索します。
ANYSPACE 関数は、文字列で最初に出現する空白、水平タブ、垂直タブ、キャリッジリターン、ラインフィード、フォームフィードの文字を検索します。

例: 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。

```
proc cas;
   string='Next = _n_ + 12E3;';
   j=1;
   do until(j=0);
     j=notspace(string, j+1);
     if j=0 then print 'The end';
     else do;
       c=substr(string, j, 1);
       print "j=" j;
       print "c=" c;
```

```
        end;
      end;
run;
```

SAS は次の出力をログに書き込みます。

```
j=2 c=e
j=3 c=x
j=4 c=t
j=6 c==
j=8 c=_
j=9 c=n
j=10 c=_
j=12 c=+
j=14 c=1
j=15 c=2
j=16 c=E
j=17 c=3
j=18 c=;
The end
```

NOTUPPER 関数

文字列から大文字でない文字を検索し、その文字が見つかった最初の文字位置を返します。

返されるデータの種類: DOUBLE

構文

NOTUPPER('*expression*'<, *start*>)

引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、VARCHAR、NVARCHAR

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

NOTUPPER 関数は、文字列から大文字でない文字の最初の出現を検索します。対象の文字が検出されると、NOTUPPER は文字列内の該当文字の位置を返します。対象の文字が検出されない場合、NOTUPPER は値 0 を返します。

引数を 1 つのみ使用すると、NOTUPPER は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

次のいずれかの条件が満たされると、NOTUPPER は値 0 を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

NOTUPPER 関数は、文字列から大文字でない文字を検索します。ANYUPPER 関数は、文字列から大文字を検索します。

例

次の例では、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。

```
proc cas;
    string='Next = _n_ + 12E3;';
    j=1;
    do until(j=0);
      j=notupper(string, j+1);
      if j=0 then print 'The end';
      else do;
        c=substr(string, j, 1);
        print "j=" j;
        print "c=" c;
      end;
    end;
run;
```

SAS は次の出力をログに書き込みます。

```
j=2 c=e
j=3 c=x
j=4 c=t
j=5 c=
j=6 c==
j=7 c=
j=8 c=_
j=9 c=n
j=10 c=_
j=11 c=
j=12 c=+
j=13 c=
j=14 c=1
j=15 c=2
j=17 c=3
j=18 c=;
The end
```

NOTXDIGIT 関数

文字列から 16 進文字でない文字を検索し、その文字が見つかった最初の文字位置を返します。

返されるデータの種類: DOUBLE

構文

NOTXDIGIT('*expression*'<, *start*>)

引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、VARCHAR、NVARCHAR

start
数値に評価されるか強制変換できる有効な式を指定し、検索を開始する文字の位置と検索の方向を指定します。

データの種類 DOUBLE

詳細

NOTXDIGIT 関数は、文字列から数字、大文字または小文字の A、B、C、D、E、F でない文字の最初の出現を検索します。対象の文字が検出されると、NOTXDIGIT は文字列内の該当文字の位置を返します。対象の文字が検出されない場合、NOTXDIGIT は値 0 を返します。

引数を 1 つのみ使用すると、NOTXDIGIT は文字列の先頭から検索を開始します。2 つの引数を使用する場合、第 2 引数 *start* の絶対値で検索の開始位置を指定します。検索方向は次のように決まります。

- *start* の値が正の場合、検索は右方向に進みます。
- *start* の値が負の場合、検索は左方向に進みます。
- *start* の値が文字列の負の長さよりも小さい場合、文字列の末尾から検索が開始されます。

次のいずれかの条件が満たされると、NOTXDIGIT は値 0 を返します。

- 検索文字が見つからない。
- *start* の値が文字列の長さよりも大きい。
- *start* の値が 0 になっている。

比較

NOTXDIGIT 関数は、文字列から 16 進文字でない文字を検索します。ANYXDIGIT 関数は、文字列から 16 進文字を検索します。

例

次の例では、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。

```
proc cas;
   string='Next = _n_ + 12E3;';
    j=1;
    do until(j=0);
      j=notxdigit(string, j+1);
      if j=0 then print 'The end';
      else do;
        c=substr(string, j, 1);
        print "j=" j;
        print "c=" c;
      end;
    end;
run;
```

SAS は次の出力をログに書き込みます。

```
j=3 c=x
j=4 c=t
j=5 c=
j=6 c==
j=7 c=
j=8 c=_
j=9 c=n
j=10 c=_
j=11 c=
j=12 c=+
j=13 c=
j=18 c=;
The end
```

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。

返されるデータの種類: DOUBLE

構文

NPV(*r*, *freq*, *c0*, *c1*, ..., *cn*)

引数

r
指定したベース期間の利率をパーセントで表した数値。

データの種類 DOUBLE

freq
利率 *r* を使用して指定したベース期間中の支払い回数を表す数値。

範囲 *freq* > 0

データの種類 DOUBLE

注 *freq* = 0 は、連続割引を許可するフラグです。

c0, c1, ..., cn
時間 0,1,...n に発生する現金支出(支払い)または現金流入(収入)を表す数値のキャッシュフローです。これらのキャッシュフローは、等間隔の期間開始時の値であると想定されます。負の値は支払い、正の値は収入、値 0 はその時点でキャッシュフローがないことを表します。*c0* 引数と *c1* 引数は必須です。

データの種類 DOUBLE

比較

NPV 関数は、*r* 引数をパーセントで指定する点を除き、NETPV と同じです。

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。

返されるデータの種類: DOUBLE

構文

NWKDOM(*n*, *weekday*, *month*, *year*)

引数

n
指定した日を含む月の週を数値で指定します。

範囲 1-5

データの種類 DOUBLE

ヒント *N*=5 は、指定した日がその月の最終週に発生することを示します。*n*=4 と *n*=5 の結果が同じになる場合もあります。

weekday
週の曜日に対応する数値を指定します。

範囲 1-7

データの種類 DOUBLE

ヒント 日曜日は、週の最初の日として見なされ、*weekday* の値は 1 になります。

month
年の月に対応する数値を指定します。

範囲 1-12

データの種類 DOUBLE

year
4 桁のカレンダー年を指定します。

データの種類 DOUBLE

詳細

NWKDOM 関数は、指定した月および年の *n* 番目の曜日の SAS 日付値を返します。DATE9.形式などの有効な SAS 日付形式を使用して、カレンダー日付を表示します。月の最後に発生する特定の曜日の場合、*n*=5 を指定できます。

n=5 と *n*=4 の結果が同じになる場合もあります。要求された曜日がその月に 4 回しか発生しない場合にこのような結果になります。たとえば、1 月の初日が日曜日の場合、日曜日、月曜日および火曜日は 5 回発生しますが、水曜日、木曜日、金曜日、土曜日は 4 回しか発生しません。この場合、水曜日、木曜日、金曜日、土曜日に対して *n*=5 または *n*=4 を指定すると、結果が同じになります。

その年がうるう年でない場合、2 月は 28 日間あり、週の各曜日は 4 回発生します。この場合、すべての曜日で *n*=5 と *n*=4 の結果が同じになります。

比較

NWKDOM 関数では、*weekday* の値は、日曜日から始まる週の曜日の数値に対応しています。この値は、WEEKDAY 関数で使用される値と同じで、日曜日=1 (以下同様)になります。*month* の値は、1 月から始まる年の月の数値に対応します。この値は、MONTH 関数で使用される値と同じで、1 月=1 (以下同様)になります。

NWKDOM 関数を使用して、HOLIDAY 関数で定義されていないイベントを計算できます。たとえば、大学の卒業が常に 6 月の第 1 土曜日にスケジュールされている場合、次のステートメントを使用してその日付を計算できます。UnivGrad = nwkdom(1, 7, 6, *year*);

例: 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。

```
proc cas;
  /* Return the date of the third Monday in May 2012. */
  a=nwkdom(3, 2, 5, 2012);
    /* Return the date of the fourth Wednesday in November 2012. */
  b=nwkdom(4, 4, 11, 2012);
    /* Return the date of the fourth Saturday in November 2012. */
  c=nwkdom(4, 7, 11, 2012);
    /* Return the date of the first Sunday in January 2013. */
  d=nwkdom(1, 1, 1, 2013);
    /* Return the date of the second Tuesday in September 2012. */
  e=nwkdom(2, 3, 9, 2012);
    /* Return the date of the fifth Thursday in December 2012. */
  f=nwkdom(5, 5, 12, 2012);
  print "a=" a;
  print "b=" b;
  print "c=" c;
  print "d=" d;
  print "e=" e;
```

```
  print "f=" f;
end;
run;
```

SAS は次の出力をログに書き込みます。

```
a=19134
b=19325
c=19321
d=19364
e=19247
f=19354
```

ORDINAL 関数

値のリストを並べ替え、リスト内の位置に基づく値を返します。

返されるデータの種類: DOUBLE

構文

ORDINAL(*position*, *expression-1, expression-2* <, ...*expression-n*>)

引数

position
引数リストの要素数以下の整数を指定します。

要件 *position* は正数にする必要があります。

データの種類 DOUBLE

expression
数値に評価される有効な式を指定します。

要件 少なくとも 2 つの引数が必要です。

データの種類 DOUBLE

詳細

ORDINAL 関数はリストを並べ替えて、*position* で指定されたリスト内の引数を返します。欠損値は下位に並べ替えられ、数値の前に配置されます。

比較

ORDINAL 関数は、null 値、欠損値、非欠損値、非 null 値の両方を数えますが、SMALLEST 関数は非 null 値および非欠損値のみを数えます。

例

次のステートメントは、ORDINAL 関数を示しています。

ステートメント	結果
a=ordinal(4,1,.,2,3,-4,5,6,7);	2

PCTL 関数

パーセントに対応するパーセント点を返します。

返されるデータの種類: DOUBLE

構文

PCTL<*n*>(*percentage*, *expression*<, ...*expression*>)

引数

n
計算するパーセント点の定義を指定する 1 から 5 までの数字です。

デフォルト 定義 5

データの種類 DOUBLE

参照項目: *Base SAS プロシジャガイド*の"Methods for Computing Quantile Statistics"の表にある QNTLDEF= (エイリアス PCTLDEF=)の説明。

percentage
計算するパーセント点を指定します。

データの種類 DOUBLE

ヒント *percentage* は、0≤*percentage*≤100 の数値です。

percentage は、0 ≤ *percentage* ≤ 100 の数値です。

expression
数値に評価される有効な式を指定します。その値はパーセント点計算で計算されます。

データの種類 DOUBLE

詳細

PCTL 関数は、パーセントに対応する非 null 値または非欠損値のパーセント点を返します。*percentage* が null 値または欠損値、ゼロ未満、あるいは 100 を超える場合、PCTL 関数はエラーメッセージを生成します。

例

次のステートメントは、PCTL 関数を示しています。

ステートメント	結果
lower_quartile=PCTL(25,2,4,1,3);	1.5
percentile_def2=PCTL2(25,2,4,1,3);	1
lower_tertile=PCTL(100/3,2,4,1,3);	2
percentile_def3=PCTL3(100/3,2,4,1,3);	2
median=PCTL(50,2,4,1,3);	2.5
upper_tertile=PCTL(200/3,2,4,1,3);	3
upper_quartile=PCTL(75,2,4,1,3);	3.5

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。

返されるデータの種類: DOUBLE

構文

PERM(*n*<, *r*>)

引数

n

要素の合計数を表す有効な式を指定します。サンプルはこの中から選ばれます。

データの種類 DOUBLE

r

選択した要素数を表す有効な式を指定します。

制限事項 $r \le n$

データの種類 DOUBLE

注 *r* を省略すると、関数は *n* の階乗を返します。

詳細

PERM 関数の数学的表現は、次の式によって得られます。

$$PERM(n, r) = \frac{n!}{(n-r)!}$$

$n \ge 0$、$r \ge 0$ および $n \ge r$ となります。

式による計算ができない場合は欠損値が返されます。やや大きな値の場合、PERM 関数を計算できないことがあります。

例

次のステートメントは、PERM 関数を示しています。

ステートメント	結果
x=perm(5, 1);	5
x=perm(5);	120
x=perm(5, 2)	20

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。

返されるデータの種類: DOUBLE

構文

PMT(*rate*, *number-of-periods*, *principal-amount*<, *future-amount*><, *type*>)

引数

rate
支払期間ごとの利率を指定します。

データの種類 DOUBLE

number-of-periods
支払期間の数を指定します。

要件 *number-of-periods* は正の整数にする必要があります。

データの種類 DOUBLE

principal-amount
ローンの元金を指定します。null 値または欠損値が指定されている場合、ゼロとみなされます。

データの種類 DOUBLE

future-amount
将来の残高を指定します。*future-amount* は、指定した支払い期間数終了後のローンの残高、または定期預金の将来の残高を指定できます。0 を指定した場合、*future-amount* が省略されたか欠損値が指定されたとみなされます。

データの種類 DOUBLE

type
期首と期末のどちらで支払がなされるのかを指定します。0 は期末の支払を表し、1 は期首の支払を表します。*type* が省略された場合や、null 値または欠損値が指定された場合は、ゼロとみなされます。

データの種類 DOUBLE

例

- 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 1037.03208935915 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 1030.16432717796 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(.01/12., 5, 5000);

  計算によって 1002.50138831008 という値が返されます。
- 有利子期間が 18 年を超え、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-129.081160867993 という値が返されます。

POISSON 関数

Poisson 分布の確率を返します。

返されるデータの種類: DOUBLE

構文

POISSON(*m*, *n*)

引数

m
数値の平均パラメーターに評価される有効な式を指定します。

範囲 $m \geq 0$

データの種類 DOUBLE

n
確率変数に評価される有効な式を指定します。

範囲 $n \geq 0$

データの種類 DOUBLE

詳細

POISSON 関数は、平均が *m* の Poisson 分布のオブザベーションが *n* 以下となる確率を返します。オブザベーションが指定した値の *n* と等しくなる確率を計算するには、Poisson 分布からの 2 つの確率(*n* の確率と *n*-1 の確率)の差異を計算します。

例

次のステートメントは、POISSON 関数を示しています。

ステートメント	結果
x=poisson(1, 2);	0.9196986029286

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。

返されるデータの種類: DOUBLE

構文

PPMT(*rate*, *period*, *number-of-periods*, *principal-amount*<, *future-amount*><, *type*>)

引数

rate
支払期間ごとの利率を指定します。

データの種類 DOUBLE

period
元本支払いを計算するための支払い期間を指定します。

要件 *period* は、*number-of-periods* の値以下の整数で指定する必要があります。

データの種類 DOUBLE

number-of-periods
支払期間の数を指定します。

要件 *number-of-periods* は正の整数にする必要があります。

データの種類 DOUBLE

principal-amount
ローンの元金を指定します。null 値または欠損値が指定されている場合、ゼロとみなされます。

データの種類 DOUBLE

future-amount
将来の残高を指定します。*future-amount* は、指定した支払い期間数終了後のローンの残高、または定期預金の将来の残高を指定できます。*future-amount* が省略された場合や、null 値または欠損値が指定された場合は、ゼロとみなされます。

データの種類 DOUBLE

type
期首と期末のどちらで支払がなされるのかを指定します。0 は期末の支払を表し、1 は期首の支払を表します。*type* が省略された場合や、null 値または欠損値が指定された場合は、ゼロとみなされます。

データの種類 DOUBLE

例

- $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 75.6231860083663 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 640.010324505867 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 359.007807907562 が返されます。

PROBBETA 関数

ベータ分布の確率を返します。

返されるデータの種類: DOUBLE

構文

PROBBETA(*x*, *a*, *b*)

引数

x

数値の確率変数です。

範囲 $0 \le x \le 1$

データの種類 DOUBLE

a

数値の形状パラメーターです。

範囲 $a > 0$

データの種類 DOUBLE

b

数値の形状パラメーターです。

範囲 $b > 0$

データの種類 DOUBLE

詳細

PROBBETA 関数は、形状パラメーターが *a* と *b* のベータ分布のオブザベーションが *x* 以下となる確率を返します。

例

次のステートメントは、PROBBETA 関数を示しています。

ステートメント	結果
x=probbeta(.2,3,4);	0.09888

PROBBNML 関数

二項分布の確率を返します。

返されるデータの種類: DOUBLE

構文

PROBBNML(*p*, *n*, *m*)

引数

p
数値の成功率パラメーターです。

範囲 $0 \le p \le 1$

データの種類 DOUBLE

n
独立した Bernoulli 試行数のパラメーターです。この引数は整数にする必要があります。

範囲 $n > 0$

データの種類 DOUBLE

m
成功数の確率変数です。この引数は整数にする必要があります。

範囲 $0 \le m \le n$

データの種類 DOUBLE

詳細

PROBBNML 関数は、成功確率が *p*、試行数が *n*、成功数が *m* の二項分布のオブザベーションが *m* 以下となる確率を返します。オブザベーションが指定した値の *m* と等しくなる確率を計算するには、二項分布の 2 つの確率(*m* の成功と *m*-1 の成功)の差異を計算します。

例

次のステートメントは、PROBBNML 関数を示しています。

ステートメント	結果
x=probbnml(0.5,10,4);	0.376953125

PROBCHI 関数

カイ 2 乗分布の確率を返します。

返されるデータの種類: DOUBLE

構文

PROBCHI(*x*, *df*<, *nc*>)

引数

x
数値の確率変数です。

範囲 $x \geq 0$

データの種類 DOUBLE

df
数値の自由度パラメーターです。

範囲 $df > 0$

データの種類 DOUBLE

nc
数値の非心度パラメーターです(省略可能)。

範囲 $nc \geq 0$

データの種類 DOUBLE

詳細

PROBCHI 関数は、自由度が *df*、非心度パラメーターが *nc* のカイ 2 乗分布のオブザベーションが *x* 以下となる確率を返します。この関数では、整数以外の自由度パラメーター *df* を使用できます。パラメーター *nc* (省略可能)が指定されていないかゼロの場合、心度カイ 2 乗分布の値が返されます。

例

次のステートメントは、PROBCHI 関数を示しています。

ステートメント	結果
x=probchi(11.264,11);	0.5785813293173

PROBF 関数

F 分布の確率を返します。

返されるデータの種類: DOUBLE

構文

PROBF(*x*, *ndf*, *ddf*<, *nc*>)

引数

x
数値の確率変数です。

範囲 $x \geq 0$

データの種類 DOUBLE

ndf
数値の分子の自由度パラメーターです。

範囲 *ndf* > 0

データの種類 DOUBLE

ddf
数値の分母の自由度パラメーターです。

範囲 *ddf* > 0

データの種類 DOUBLE

nc
数値の非心度パラメーターです(省略可能)。

範囲 *nc* ≥ 0

データの種類 DOUBLE

詳細

PROBF 関数は、分子の自由度が *ndf*、分母の自由度が *ddf*、非心度パラメーターが *nc* の *F* 分布のオブザベーションが *x* 以下となる確率を返します。PROBF 関数では、自由度パラメーター *ndf* と *ddf* に非整数を指定できます。パラメーター *nc* (省略可能)が指定されていないかゼロの場合、心度 *F* 分布の値が返されます。

F 検定統計の有意水準は次の式で求められます。

```
p=1-probf(x,ndf,ddf);
```

例

次のステートメントは、PROBF 関数を示しています。

ステートメント	結果
x=probf(3.32,2,3);	0.82639336022431

PROBGAM 関数

ガンマ分布の確率を返します。

返されるデータの種類: DOUBLE

構文

PROBGAM(*x*, *a*)

引数

x
数値の確率変数です。

範囲 $x \geq 0$

データの種類 DOUBLE

a
数値の形状パラメーターです。

データの種類 DOUBLE

詳細

PROBGAM 関数は、形状パラメーターが *a* のガンマ分布のオブザベーションが *x* 以下となる確率を返します。

例

次のステートメントは、PROBGAM 関数を示しています。

ステートメント	結果
x=probgam(1,3);	0.08030139707139

PROBHYPR 関数

超幾何分布の確率を返します。

返されるデータの種類: DOUBLE

構文

PROBHYPR(*N*, *K*, *n*, *x*<, *r*>)

引数

N
母集団サイズパラメーターです。この引数は整数にする必要があります。

範囲　$N \geq 1$

データの種類　DOUBLE

K
対象カテゴリの項目数のパラメーターです。この引数は整数にする必要があります。

範囲　$0 \leq K \leq N$

データの種類　DOUBLE

n
サンプルサイズパラメーターです。この引数は整数にする必要があります。

範囲　$0 \leq n \leq N$

データの種類　DOUBLE

x
確率変数です。この引数は整数にする必要があります。

範囲　$max(0, K + n-N) \leq x \leq min(K,n)$

r
数値オッズ比パラメーターです。

範囲　$r \geq 0$

データの種類　DOUBLE

詳細

PROBHYPR 関数は、拡張超幾何分布(母集団サイズは *N*、項目数は *K*、サンプルサイズは *n*、オッズ比は *r*)からオブザベーションが *x* 以下となる確率を返します。パラメーター *r* (省略可能)が指定されていないか 1 に設定されている場合、一般超幾何分布の値が返されます。

例

次のステートメントは、PROBHYPR 関数を示しています。

ステートメント	結果
x=probhypr(200,50,10,2);	0.52367340812167

PROBIT 関数

標準正規分布の分位点を返します。

返されるデータの種類: DOUBLE

構文

PROBIT(*p*)

引数

p
数値の確率です。

範囲 $0 < p < 1$

データの種類 DOUBLE

詳細

PROBIT 関数は、標準正規分布の *p*番目の分位点を返します。標準正規分布のオブザベーションが、返される分位点以下となる確率は *p* です。

注意
結果は-8.222 から 7.941 の間の値に切り捨てられる場合があります。

注: PROBIT は PROBNORM 関数の逆数です。

例

次のステートメントは、PROBIT 関数を示しています。

ステートメント	結果
x=probit(.025);	-1.95996398454005
x=probit(1.e-7);	-5.19933758219281

PROBMC 関数

平均値の多重比較を行うために各種の分布から確率または分位点を返します。

返されるデータの種類: DOUBLE

構文

PROBMC(*distribution*, *q*, *prob*, *df*, *nparms*<, *parameters*>)

引数

distribution
文字列を評価するか、文字列に強制変換できる文字定数、変数、または式であり、分布を識別します。分布として、ANOM (平均の分析)、DUNNETT1、DUNNETT2、MAXMOD (最大モジュール)、PARTRANGE (パーティション分割範囲)、RANGE (Student 化範囲)、WILLIAMS が有効です。

分布を特定する文字定数、変数または式です。有効な分布は次のとおりです。

分布	引数
平均分析	ANOM
片側 Dunnett	DUNNETT1
両側 Dunnett	DUNNETT2
最大係数	MAXMOD
分割された範囲	PARTRANGE
スチューデント化された範囲	RANGE
Williams	WILLIAMS

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

q
分布からの分位点です。

制限事項 *q* または *prob* のいずれか一方を指定できます(両方指定することはできません)。

データの種類 DOUBLE

prob
分布からの左側確率です。

制限事項 *prob* または *q* のいずれか一方を指定できます(両方指定することはできません)。

データの種類 DOUBLE

df
自由度です。

注: 欠損値は無限値として解釈されます。

nparms
治療の数です。

データの種類 DOUBLE

注 DUNNETT1 と DUNNETT2 については、コントロールグループを計算に入れません。

parameters
標本サイズが等しくない場合の処理に指定する必要のある *nparms* パラメーターのセットです。*parameters* の意味は *distribution* の値によって異なります。*parameters* が指定されていない場合、サンプルサイズは揃っているものと見なされます。通常、帰無仮説のケースはサンプルサイズが揃っています。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

詳細

概要

PROBMC 関数は、分散推定値に有限および無限の自由度を使用して、各種の分布の確率または分位点を返します。

prob 引数は確率変数が *q* 未満となる確率です。したがって、*p*-values は 1– *prob* のように計算できます。たとえば、有意水準 5%の臨界値を計算するには、*prob*= 0.95 と設定します。計算された確率の精度は $O(10^{-8})$ (絶対誤差)、計算された分位点の精度は $O(10^{-5})$です。

注: 自由度が有限のときと、サンプルサイズが不揃いのときは、スチューデント化された範囲は計算されません。

注: Williams 検定はサンプルサイズが揃っているときにのみ計算されます。

式とパラメーター

ここに掲げる等式は、各種の分布と、それぞれの分布におけるさまざまな状況で確率 *prob* と分位点 *q* を関連付ける等式に使用する式を定義します。これらの式では、*v* を自由度 *df* とします。

$$d\mu_\nu(x) = \frac{\nu^{\frac{\nu}{2}}}{\Gamma\left(\frac{\nu}{2}\right)2^{\frac{\nu}{2}-1}} x^{\nu-1}\varepsilon^{-\frac{\nu x^2}{2}}dx$$

$$\phi(x) = \frac{1}{\sqrt{2\,\pi}}\varepsilon^{-\frac{x^2}{2}}$$

$$\Phi(x) = \int_{-\infty}^{x} \phi(u)du$$

平均の分析の計算

平均の分析(ANOM)は、*k* (ガウス)標本として編成されるデータに適用されます。ここでは、*i*番目の標本のサイズが n_i となります。$I = \sqrt{-1}$であるとします。分布関数 [1, 2, 3, 4, 5]は、多次元 *k* 多変量$\mathbb{T}$ベクトルの最大絶対値、ν 自由度、関連付けられた相関行列$\rho_{ij} = -\alpha_i\alpha_j$の CDF です。この式は次のように記述することができます。

$$\begin{aligned} prob &= r(|t_1| < h, |t_2| < h, ..., |t_k| < h) \\ &= \int_0^\infty \left\{ \int_0^\infty \prod_{j=0}^{j=k} g\left(sh, y, \alpha_j\right)\phi(y)dy \right\} d\mu_\nu(s) \end{aligned}$$

前述の式には次の関係が適用されます。

$$g\left(sh, y, \alpha_j\right) = \Phi\left(\frac{sh - y\alpha_j}{\sqrt{1+\alpha_j^2}}\right) - \Phi\left(\frac{-sh - y\alpha_j}{\sqrt{1+\alpha_j^2}}\right)$$

この式では、$\Gamma(.)$、$\phi(.)$、$\Phi(.)$はそれぞれ、標準正規分布からのガンマ関数、密度、および CDF です。

$\nu = \infty$では、分布はこの式になります。

$$r(|t_1| < h, |t_2| < h, ..., |t_k| < h) = \int_0^\infty \prod_{j=0}^{j=k} g\left(h, y, \alpha_j\right)\phi(y)dy$$

前述の式には次の関係が適用されます。

$$g\left(h, y, \alpha_j\right) = \Phi\left(\frac{h - y\alpha_j}{\sqrt{1+\alpha_j^2}}\right) - \Phi\left(\frac{-h - y\alpha_j}{\sqrt{1+\alpha_j^2}}\right)$$

バランスの取れている場合、分布は次の式になります。

$$r(|t_1| < h, |t_2| < h, ..., |t_n| < h) = \int_0^\infty f(h, y, \rho)^n \phi(y) dy$$

前述の式には次の関係が適用されます。

$$f(h, y, \rho) = \Phi\left(\frac{h - y\sqrt{\rho}}{\sqrt{1+\rho}}\right) - \Phi\left(\frac{-h - y\sqrt{\rho}}{\sqrt{1+\rho}}\right)$$

$$\rho = \frac{1}{n-1}$$

この分布の構文は次のとおりです。

x=*probmc*('*anom*', *q*, *p*, *nu*, *n*[, *alpha*$_1$, ..., *alpha*$_n$]);

引数

x
返される結果の数値です。

q
分位点を示す数値です。

p
確率を示す数値です。*p* と *q* は、いずれかが欠損値である必要があります。

nu
自由度を示す数値です。

n
サンプル数を示す数値です。

alpha$_i$, *i* = 1, ...,*k*
この分布の最初の式のアルファ値を示すオプションの数値です。“平均の分析の計算”を参照してください。

多対 1 の *t* 統計量: Dunnett の片側検定

- 自由度が有限の不揃いなケースで確率 *prob* と分位点 *q* を相関させます。*parameters* は $\lambda_1, ..., \lambda_k$、*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。

$$prob = \int_0^\infty \int_{-\infty}^\infty \phi(y) \prod_{i=1}^k \Phi\left(\frac{\lambda_i y + qx}{\sqrt{1-\lambda_i^2}}\right) dy\, du_\nu(x)$$

- 自由度が有限の揃っているケースで確率 *prob* と分位点 *q* を相関させます。*parameters* は渡されず、$\left(\lambda = \sqrt{\frac{1}{2}}\right)$*nparms* の値は *k* に設定され、*df* の値は *v* に設定されます。式は次のとおりです。

$$prob = \int_0^\infty \int_{-\infty}^\infty \phi(y)[\Phi(y + \sqrt{2qx})]^k dy\, du_\nu(x)$$

- 自由度が無限の不揃いなケースで確率 *prob* と分位点 *q* を相関させます。*parameters* は $\lambda_1, ..., \lambda_k$、*nparms* の設定値は *k*、*df* の設定値は欠損です。式は次のとおりです。

$$prob = \int_{-\infty}^{\infty} \phi(y) \prod_{i=1}^{k} \Phi\left(\frac{\lambda_i y + q}{\sqrt{1-\lambda_i^2}}\right) dy$$

- 自由度が無限の揃っているケースで確率 *prob* と分位点 *q* を相関させます。*parameters* は渡されず、$\left(\lambda = \sqrt{\frac{1}{2}}\right)$*nparms* の値は *k* に設定され、*df* の値は欠損値に設定されます。式は次のとおりです。

$$prob = \int_{-\infty}^{\infty} \phi(y)[\Phi(y + \sqrt{2q})]^k dy$$

多対 1 の *t* 統計量: Dunnett の両側検定

- 自由度が有限の不揃いなケースで確率 *prob* と分位点 *q* を相関させます。*parameters* は $\lambda_1, \ldots, \lambda_k$、*nparms* の設定値は *k*、*df* の設定値は *ν* です。式は次のとおりです。

$$prob = \int_{0}^{\infty} \int_{-\infty}^{\infty} \phi(y) \prod_{i=1}^{k} \left[\Phi\left(\frac{\lambda_i y + qx}{\sqrt{1-\lambda_i^2}}\right) - \Phi\left(\frac{\lambda_i y - qx}{\sqrt{1-\lambda_i^2}}\right)\right] dy\, du_\nu(x)$$

- 自由度が有限の揃っているケースで確率 *prob* と分位点 *q* を相関させます。*parameters* は渡されず、*nparms* の設定は *k*、*df* の設定値は *ν* です。式は次のとおりです。

$$prob = \int_{0}^{\infty} \int_{-\infty}^{\infty} \phi(y)[\Phi(y + \sqrt{2qx}) - \Phi(y - \sqrt{2qx})]^k dy\, du_\nu(x)$$

- 自由度が無限の不揃いなケースで確率 *prob* と分位点 *q* を相関させます。*parameters* は $\lambda_1, \ldots, \lambda_k$、*nparms* の設定値は *k*、*df* の設定値は欠損です。式は次のとおりです。

$$prob = \int_{-\infty}^{\infty} \phi(y) \prod_{i=1}^{k} \left[\Phi\left(\frac{\lambda_i y + q}{\sqrt{1-\lambda_i^2}}\right) - \Phi\left(\frac{\lambda_i y - q}{\sqrt{1-\lambda_i^2}}\right)\right] dy$$

- 自由度が無限の揃っているケースで確率 *prob* と分位点 *q* を相関させます。*parameters* は渡されず、*nparms* の設定は *k*、*df* の設定値は欠損です。式は次のとおりです。

$$prob = \int_{-\infty}^{\infty} \phi(y)[\Phi(y + \sqrt{2q}) - \Phi(y - \sqrt{2q})]^k dy$$

パーティション分割範囲の計算

RANGE は、*n* のグループ平均のスチューデント化範囲の分布で使用されます。PARTRANGE は、分割されたスチューデント化範囲の分布で使用されます。*n* グループを、大きさが $n_1 + \ldots + n_k = n$ の *k* 個のサブセットに分割します。このとき、分割された範囲はそれぞれのサブセットのスチューデント化範囲の最大です。スチューデント化係数はすべてのケースで 同一です。

$$prob = \int_{0}^{\infty} \prod_{i=1}^{i=k} \left(\int_{-\infty}^{\infty} k\phi(y)(\Phi(y) - \Phi(y - qx))^{k-1} dy \right)^{n_i} d\mu_\nu(x)$$

この分布の構文は次のとおりです。

x=*probmc*('*partrange*', *q*, *p*,*nu*, *k*, n_1 ,...,n_k);

引数

x
返される結果の数値です(確率または分位点)。

q
分位点を示す数値です。

p
確率を示す数値です。*p* と *q* は、いずれかが欠損値である必要があります。

nu
自由度を示す数値です。

k
グループ数を示す数値です。

n_i *i*=1,...,*k*
この分布の式の *n* 値を示すオプションの数値です。"パーティション分割範囲の計算"を参照してください。

Student 化範囲

- 自由度が有限の揃っているケースで確率 *prob* と分位点 *q* を相関させます。*parameters* は渡されず、*nparms* の設定は *k*、*df* の設定値は *v* です。式は次のとおりです。

$$prob = \int_0^{\infty} \int_{-\infty}^{\infty} k\phi(y)[\Phi(y) - \Phi(y - qx)]^{k-1} dy\, du_\nu(x)$$

- 自由度が無限の不揃いなケースで確率 *prob* と分位点 *q* を相関させます。*parameters* は σ_1, ..., σ_k、*nparms* の設定値は *k*、*df* の設定値は欠損です。式は次のとおりです。

$$prob = \int_{-\infty}^{\infty} \sum_{j=1}^{k} \left\{ \prod_{i=1}^{k} \left[\Phi\left(\frac{y}{\sigma_i}\right) - \Phi\left(\frac{y-q}{\sigma_i}\right) \right] \right\} \phi\left(\frac{y}{\sigma_j}\right) \frac{1}{\sigma_j} dy$$

- 自由度が無限の揃っているケースで確率 *prob* と分位点 *q* を相関させます。*parameters* は渡されず、*nparms* の設定は *k*、*df* の設定値は欠損です。式は次のとおりです。

$$prob = \int_{-\infty}^{\infty} k\phi(y)[\Phi(y) - \Phi(y - q)]^{k-1} dy$$

Student 化最大モジュール

- 自由度が有限の不揃いなケースで確率 *prob* と分位点 *q* を相関させます。*parameters* は σ_1, ..., σ_k、*nparms* の設定値は *k*、*df* の設定値は *v* です。式は次のとおりです。

$$prob = \int_0^{\infty} \prod_{i=1}^{k} \left[2\Phi\left(\frac{qx}{\sigma_i}\right) - 1 \right] d\mu_\nu(x)$$

- 自由度が有限の揃っているケースで確率 *prob* と分位点 *q* を相関させます。*parameters* は渡されず、*nparms* の設定は *k*、*df* の設定値は *ν* です。式は次のとおりです。

$$prob = \int_0^{\infty} [2\Phi(qx) - 1]^k d\mu_\nu(x)$$

- 自由度が無限の不揃いなケースで確率 *prob* と分位点 *q* を相関させます。*parameters* は σ_1, ..., σ_k、*nparms* の設定値は *k*、*df* の設定値は欠損です。式は次のとおりです。

$$prob = \prod_{i=1}^{k} \left[2\Phi\left(\frac{q}{\sigma_i}\right) - 1\right]$$

- 自由度が無限の揃っているケースで確率 *prob* と分位点 *q* を相関させます。*parameters* は渡されず、*nparms* の設定は *k*、*df* の設定値は欠損です。式は次のとおりです。

$$prob = [2\Phi(q) - 1]^k$$

Williams の検定

Williams の検定は、投与治療平均を対照平均と比較して、治療の最小有効量を決定するときに必要になります。

注: Williams 検定はサンプルサイズが揃っているときにのみ計算されます。

X_1, X_2, ..., X_k は同一の独立している N(0,1)の確率変数であるものとします。ここで、Y_k は次の式で与えられる平均値を表します。

$$Y_k = \frac{X_1 + X_2 + ... + X_k}{k}$$

次の値の分布を計算する必要があります。

$$(Y_k - Z)/S$$

引数

Y_k
前述の定義のとおりです。

Z
N(0,1)の独立した確率変数です。

S
½νS² が χ^2 変数です(自由度 ν)。

1 Y_k の分布を計算します。分布の計算はこの演算の基本的な(高コストな)部分で、Y_k の密度と確率の両方を求めるのに使われます。U_i をこの式に定義するとします。

$$U_i = \frac{X_1 + X_2 + ... + X_i}{i},\ i = 1, 2, ..., k$$

$Y_k > d$ の確率の再帰式を書くことができます。*d* の値は任意の実数になります。

$$\begin{aligned}\Pr(Y_k > d) &= \Pr(U_1 > d)\\ &\quad +\Pr(U_2 > d, U_1 < d)\\ &\quad +\Pr(U_3 > d, U_2 < d, U_1 < d)\\ &\quad +\ldots\\ &\quad +\Pr(U_k > d, U_{k-1} < d, \ldots, U_1 < d)\\ &\quad = \Pr(Y_{k-1} > d) + \Pr(X_k + (k-1)U_{k-1} > kd)\end{aligned}$$

この確率を計算するには、N(0,1)密度関数から開始します。

$$D(U_1 = x) = \phi(x)$$

そして、畳み込みを再帰的に計算します。

$$D(U_k = x, U_{k-1} < d, \ldots, U_1 < d) =$$

$$\int_{-\infty}^{d} D(U_{k-1} = y, U_{k-2} < d, \ldots, U_1 < d)(k-1)\phi(kx - (k-1)y)dy$$

この順次畳み込みから、前に示した$\Pr(Y_k < d)$の再帰方程式のすべての要素を計算することができます。

2 $Y_k - Z$ の分布を計算します。この計算には、確率を計算するための別の畳み込みが必要です。

$$\Pr((Y_k - Z) > d) = \int_{-\infty}^{\infty} \Pr(Y_k > \sqrt{2}d + y)\phi(y)dy$$

3 $(Y_k - Z)/S$ の分布を計算します。この計算には、確率を計算するための別の畳み込みが含まれます。

$$\Pr((Y_k - Z) > tS) = \int_{0}^{\infty} \Pr((Y_k - Z) > ty)d\mu_\nu(y)$$

ν = ∞のときは、第 3 段階は必要ありません。演算が複雑なため、この冗長なアルゴリズムは有限および無限の自由度 ν の両方で $k \le 15$ のとき、高速なアルゴリズムに置き換えられます。$k \ge 16$ のときは、冗長計算が実行されます。この計算は、アルゴリズムが複雑なため、きわめて高コストで非常に時間がかかります。

比較

SAS/STAT ソフトウェアの GLM プロシジャの MEANS ステートメントは次の検定を計算します。

- Dunnett の片側検定
- Dunnett の両側検定
- スチューデント化された範囲

例: Williams の検定の計算

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	Mean
X_0	10.4
X_1	9.9
X_2	10.0
X_3	10.6
X_4	11.4
X_5	11.9
X_6	11.7

自由度が (7 – 1)(8 – 1) = 42 である平均平方は、s^2 = 1.16 です。

平均化プロセスを経て最大尤度推定 M_i を求めます。

- $X_0 > X_1$ であるため、$X_{0,1} = (X_0 + X_1)/2$ = 10.15 となります。
- $X_{0,1} > X_2$ であるため、$X_{0,1,2} = (X_0 + X_1 + X_2)/3 = (2X_{0,1} + X_2)/3$ = 10.1 となります。
- $X_{0,1,2} < X_3 < X_4 < X_5$
- $X_5 > X_6$ であるため、$X_{5,6} = (X_5 + X_6)/2$ = 11.8 となります。

これで順序制約が満たされました。

対立仮説の最尤推定値は、次のようになります。

- $M_0 = M_1 = M_2 = X_{0,1,2}$ = 10.1
- $M_3 = X_3$ = 10.6
- $M_4 = X_4$ = 11.4
- $M_5 = M_6 = X_{5,6}$ = 11.8

$t = (11.8 - 10.4)/\sqrt{2s^2/8} = 2.60$を計算します。$k$ = 6、ν = 42、t = 2.60 の確率は、.9924467341 です。この物質に対する反応があるという強力な証拠を示しています。また、次の表に示すように、上の 5%および裾の 1%の分位点も計算できます。

```
proc cas;
    prob=probmc('WILLIAMS',2.6,.,42,6);
    print "prob=" prob;
```

```
    quant5=probmc('WILLIAMS',.,.95,42,6);
    print "quant5=" quant5;
    quant1=probmc('WILLIAMS',.,.99,42,6);
    print "quant1=" quant1;
run;
```

ステートメント	結果
prob=probmc('williams',2.6,.,42,6);	0.99244668715827
quant5=probmc('williams',.,.95,42,6);	1.80656253603889
quant1=probmc('williams',.,.99,42,6);	2.49090827298686

参考文献

Guirguis, G. H., and R. D. Tobias. 2004. “On the Computation of the Distribution for the Analysis of Means.” *Communications in Statistics: Simulation and Computation* 33: 861-887.

Nelson, P. R. 1981. “Numerical evaluation of an equicorrelated multivariate non-central t distribution.” *Communications in Statistics: Part B - Simulation and Computation* 10: 41-50.

Nelson, P.R. 1982a. “An Approximation for the Complex Normal Probability Integral.” *BIT* 22(1): 94-100.

Nelson, P. R. 1982b. “Exact critical points for the analysis of means.” *Communications in Statistics: Part A - Theory and Methods* 11: 699-709.

Nelson, P. R. 1988. “Application of the Analysis of Means.” *Proceedings of the SAS Users Group International Conference* 13: 225-230.

Nelson, P. R. 1991. “Numerical evaluation of multivariate normal integrals with correlations.” *The Frontiers of Statistical Scientific Theory and Industrial Applications* 2: 97-114.

Nelson, P. R. 1993. “Additional Uses for the Analysis of Means and Extended Tables of Critical Values.” *Technometrics* 35: 61-71.

PROBNEGB 関数

負の二項分布の確率を返します。

返されるデータの種類: DOUBLE

構文

PROBNEGB(*p*, *n*, *m*)

引数

p
数値の成功率パラメーターです。

範囲 $0 \le p \le 1$

データの種類 DOUBLE

n
成功数パラメーターです。この引数は整数にする必要があります。

範囲 $n \ge 1$

データの種類 DOUBLE

m
確率変数の失敗数です。この引数は正の整数にする必要があります。

範囲 $m \ge 0$

データの種類 DOUBLE

詳細

PROBNEGB 関数は、成功確率が *p*、成功数が *n* の負の二項分布のオブザベーションが *m* 以下となる確率を返します。

オブザベーションが指定した値の *m* と等しくなる確率を計算するには、負の二項分布からの 2 つの確率(*m* の確率と *m*-1 の確率)の差異を計算します。

例

次のステートメントは、PROBNEGB 関数を示しています。

ステートメント	結果
x=probnegb(0.5,2,1);	0.5

PRXCHANGE 関数

パターンマッチングの置換を実行します。

返されるデータの種類: CHAR

構文

PRXCHANGE(*perl-regular-expression* | *regular-expression-id*, *times*, *source*)

引数

perl-regular-expression
Perl 正規表現を値とする文字定数、変数または式を指定します。

データの種類 CHAR

regular-expression-id
PRXPARSE 関数によって返されるパターン識別子の値が含まれる数値変数を指定します。

制限事項 この引数を使用する場合、PRXPARSE 関数も使用する必要があります。

データの種類 INTEGER

times
一致を検索して一致するパターンを置換する回数を指定する数値の定数、変数または式です。

データの種類 INTEGER

ヒント *times* の値が–1 の場合、*source* の末尾に到達するまで一致したパターンが置換され続けます。

source
検索する文字定数、変数または式を指定します。

データの種類 CHAR

詳細

基本

regular-expression-id を使用すると、PRXCHANGE 関数は PRXPARSE から返された *regular-expression-id* で *source* を検索します。*source* 中の値は正規表現で指定された変更が加えられて返されます。一致がない場合、PRXCHANGE は未変更の *source* の値を返します。

perl-regular-expression を使用すると、PRXCHANGE は *perl-regular-expression* を使って *source* を検索します。PRXPARSE を呼び出す必要はありません。PRXCHANGE は WHERE 句と PROC SQL の中の *perl-regular-expression* で使用できます。

注: DS2 の PRX 関数には次の制限が適用されます。

- PRX 形式の/.../.../で使用できるのは **m**、**i**、**s**、**x** のみで、前にも後にも 1 文字を置くことができます。
- マッチングモード修飾子の **p**、**o**、**c**、**a**、**l** はサポートされていません。
- マッチングモード修飾子 **g** がサポートされています。

例: DATA ステップを使用して姓名の順序を入れ替える

次の例では、名と姓の順序を入れ替えます。

```
proc cas;
/*Create four varibles that contain four different names. Include both first and last
names*/
  name1="Jones, Fred";
  name2="Kavich, Kate";
  name3="Turley, Ron";
  name4="Dulix, Yolanda";
/*Reverse the first and last name*/
  nameA=prxchange('s/(\w+), (\w+)/$2 $1/', -1, name1);
  nameB=prxchange('s/(\w+), (\w+)/$2 $1/', -1, name2);
  nameC=prxchange('s/(\w+), (\w+)/$2 $1/', -1, name3);
  nameD=prxchange('s/(\w+), (\w+)/$2 $1/', -1, name4);
  columns={'name(before)', 'name(after)'};
  coltypes={'string', 'string'};
  row1={name1, nameA};
  row2={name2, nameB};
  row3={name3, nameC};
  row4={name4, nameD};
  mytable=newtable('mytable', columns, coltypes, row1, row2, row3, row4);
  print mytable;
run;
```

アウトプット 4.5　PRXCHANGE 関数の結果

mytable: Results

name(before)	name(after)
Jones, Fred	Fred Jones
Kavich, Kate	Kate Kavich
Turley, Ron	Ron Turley
Dulix, Yolanda	Yolanda Dulix

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。

返されるデータの種類: INTEGER

構文

PRXMATCH(*perl-regular-expression*, *source*)

引数

perl-regular-expression
Perl 正規表現を値とする文字定数、変数または式を指定します。

データの種類　CHAR

source
検索する文字定数、変数または式を指定します。

データの種類　CHAR

詳細

基本

perl-regular-expression を使用すると、PRXMATCH 関数は *perl-regular-expression* で *source* を検索し、文字列の開始位置を返します。一致が見つからない場合、PRXMATCH は 0 を返します。

WHERE 句と PROC SQL で、PRXMATCH に Perl 正規表現を使用できます。

注: DS2 の PRX 関数には次の制限が適用されます。

- PRX 形式の/.../.../で使用できるのは **m**、**i**、**s**、**x** のみで、前にも後にも 1 文字を置くことができます。
- マッチングモード修飾子の **p**、**o**、**c**、**a**、**l** はサポートされていません。

Perl 正規表現のコンパイル

perl-regular-expression が定数の場合、または/o オプションが指定されている場合、一度コンパイルされた Perl 正規表現が PRXMATCH の使用のたびに再利用されます。*perl-regular-expression* が定数ではなく、/o オプションを使用しない場合、PRXMATCH を呼び出すたびに Perl 正規表現が再コンパイルされます。

例: 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。

```
proc cas;
  position=prxmatch('/world/', 'Hello world!');
  print "position=" position;
run;
```

SAS は次の出力をログに書き込みます。

```
position=7
```

PRXPARSE 関数

文字値のパターンマッチングに使用できる Perl 正規表現(PRX)をコンパイルします。

制限事項: 他の Perl 正規表現とともに使用します。

返されるデータの種類: INTEGER

構文

regular-expression-id=**PRXPARSE**(*perl-regular-expression*)

引数

regular-expression-id
PRXPARSE 関数によって返される数値のパターン識別子です。

データの種類 INTEGER

perl-regular-expression
Perl 正規表現を値とする文字、定数、変数、式のいずれかを指定します。

データの種類 CHAR

詳細

基本

PRXPARSE 関数は、他の Perl 関数でパターンをマッチングするために使用するパターン識別子番号を返します。正規表現の解析でエラーが発生すると、SAS は欠損値を返します。

PRXPARSE は、Perl 正規表現の構成にメタ文字を使用します。

Perl 正規表現のコンパイル

perl-regular-expression が定数の場合、Perl 正規表現は 1 回のみコンパイルされます。後続の PRXPARSE の呼び出しでは再コンパイルされず、すでにコンパイル済みの正規表現の *regular-expression-id* が返されます。この動作では、初期化ブロック (IF _N_ =1)を使用して Perl 正規表現を初期化する必要がないため、コードが簡素化されます。

例: Perl 正規表現のコンパイル

次の例では、PRXPARSE を使用して Perl 正規表現をコンパイルします。

```
proc cas;
   /* 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!');
   print "position=" position;
run;
```

```
position=7
```

PRXPOSN 関数

キャプチャバッファーの値が含まれる文字列を返します。

返されるデータの種類: CHAR

構文

PRXPOSN(*regular-expression-id*, *capture-buffer*, *source*)

引数

regular-expression-id
PRXPARSE 関数によって返されるパターン識別子の値が含まれる数値変数を指定します。

制限事項

データの種類 INTEGER

capture-buffer
値を取得するキャプチャバッファーを識別する数値定数、変数または式です。*capture-buffer* の値が 0 の場合、PRXPOSN は一致全体を返します。*capture-buffer* の値が 1 から正規表現の開始かっこ数までの間の場合、PRXPOSN はそのキャプチャバッファーの値を返します。*capture-buffer* の値が開始かっこの数よりも大きい場合、PRXPOSN は欠損値を返します。

値を取得するキャプチャバッファーを識別する数値定数、変数または式です。

- *capture-buffer* の値が 0 の場合、PRXPOSN は一致全体を返します。
- *capture-buffer* の値が 1 から正規表現の開始かっこ数までの間の場合、PRXPOSN はそのキャプチャバッファーの値を返します。
- *capture-buffer* の値が開始かっこの数よりも大きい場合、PRXPOSN は欠損値を返します。

データの種類 INTEGER

source
キャプチャバッファーを抽出するテキストを指定します。

データの種類 CHAR

詳細

PRXPOSN 関数は、PRXMATCH または PRXCHANGE の結果を使用してキャプチャバッファーを返します。PRXPOSN が有益な情報を返すには、これらの関数のいずれかで一致が見つかる必要があります。

注: DS2 の PRX 関数には次の制限が適用されます。

- PRX 形式の/.../.../で使用できるのは **m**、**i**、**s**、**x** のみで、前にも後にも 1 文字を置くことができます。
- マッチングモード修飾子の **p**、**o**、**c**、**a**、**l** はサポートされていません。

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。

構文

PUTC(*value*, *format-specification* <, *w*>)

必須引数

value
フォーマットされる文字値を指定します。

format-specification
value に適用する文字出力形式です。

有効な出力形式の指定方法は次のようになります。

- *format-name*
- *format-name.*
- *format-namew.*

format-name を除き、*format-specification* に–L、–R、および–C を使用して、左揃え、右揃え、中央揃えで出力を配置できます。たとえば、第 2 引数 *format-specification* の値には'upcase.-c'を使用できます。

オプション引数

w
出力形式に適用する幅を指定する数値定数、変数または式です。

操作　ここで指定した幅は、出力形式での幅の指定より優先されます。

詳細

まだ長さが割り当てられていない変数に PUTC 関数から値が返される場合、その変数の長さは最初の引数の長さによって決定されます。

比較

PUTN 関数は、実行時に数値の出力形式を指定できるようにします。

PUT 関数は実行時ではなくコンパイル時に出力形式を指定できるため、PUTC よりも高速です。

例: 文字値の配置

この例では、A の値を配置する、出力形式と配置文字の使用方法を示します。

```
proc cas;
   a='experiment';
   y=putc(a, 'upcase.-r', 20);
   print '*' y '*';
   print '*' a '*';
run;
```

```
*EXPERIMENT       *
*experiment*
```

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。

構文

PUTN(*value*, *format-specification* <, *w* <, *d*>>)

必須引数

value
フォーマットされる数値を指定します。

format-specification
value に適用する数値出力形式です。

有効な出力形式の指定方法は次のようになります。

- *format-name*
- *format-name.*
- *format-namew.*
- *format-namew.d*

format-name を除き、*format-specification* に–L、–R、および–C を使用して、左揃え、右揃え、中央揃えで出力を配置できます。たとえば、第 2 引数 *format-specification* の値には'weekdate.-c'を使用できます。

オプション引数

w
出力形式に適用する幅を指定する数値定数、変数または式です。

操作　ここで指定した幅は、出力形式での幅の指定より優先されます。

d
使用する小数点以下の桁数を指定する数値定数、変数または式です。

操作　ここで指定した桁数は、出力形式での小数点以下の桁数の指定より優先されます。

詳細

まだ長さが割り当てられていない変数に PUTN 関数から値が返される場合、変数にはデフォルトで長さ 200 バイトが割り当てられます。

比較

PUTC 関数は、実行時に文字の出力形式を指定できるようにします。

PUT 関数は実行時ではなくコンパイル時に出力形式を指定できるため、PUTN よりも高速です。

例: PUTN 関数による出力の配置

この例では、値を左揃えにする、出力形式と配置文字の使用方法を示します。

```
proc cas;
   y=putn(today(), 'weekdate.-1', 30);
   print '*' y '*';
run;
```

例のコード 4.1　配置結果

```
*       Friday, May 11, 2018*
```

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。

返されるデータの種類: DOUBLE

構文

PVP(*A*, *c*, *n*, *K*, k_0, *y*)

引数

A

額面価格を指定します。

範囲 $A > 0$

A > 0

データの種類 DOUBLE

c

1 年当たりの名目クーポン率を分数で指定します。

範囲 $0 \le c < 1$

0 ≤ *c* < 1

データの種類 DOUBLE

n

1 年当たりのクーポン数を指定します。

範囲 $n > 0$

n > 0

データの種類 DOUBLE

K

クーポンの残数を指定します。

範囲 $K > 0$

K > 0

データの種類 DOUBLE

k_0

現在の日付から最初のクーポン日までの時間を指定します。年数で表します。

範囲 $0 < k_0 \leq \frac{1}{n}$

$0 < k_0 \leq 1/n$

データの種類 DOUBLE

y

1 年当たりの名目最終利回りを指定します。分数で表します。

範囲 $y > 0$

$y > 0$

データの種類 DOUBLE

詳細

PVP 関数は次の関係に基づきます。

$$P = \sum_{k=1}^{K} \frac{c(k)}{\left(1 + \frac{y}{n}\right)^{t_k}}$$

前述の式には次の関係が適用されます。

- $t_k = nk_0 + k - 1$
- $c(k) = \frac{c}{n}A \quad for\ k = 1, \ldots, K-1$
- $c(K) = \left(1 + \frac{c}{n}\right)A$

例

```
proc cas;
   p=pvp(1000,.01,4,14,.33/2,.10);
   print "p=" p;
run;
```

返される値は 743.167613519067 です。

QTR 関数

SAS 日付値から年の四半期を返します。

返されるデータの種類: DOUBLE

構文

QTR(*date*)

引数

date
SAS 日付値を表す有効な式を指定します。

データの種類 DOUBLE

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。

例

次のステートメントは、QTR 関数を示しています。

ステートメント	結果

a=17180;
b=print(a,date7.);
c=qtr(a);
	14JAN07
	1

QUOTE 関数

文字値に二重引用符を付加します。

返されるデータの種類: CHAR、NCHAR、NVARCHAR、VARCHAR

構文

QUOTE(*expression*)

引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

詳細

QUOTE 関数は、文字値にデフォルト文字である二重引用符を付加します。引数内から二重引用符が検出されると、さらに二重にして出力されます。

受け取る変数には、引数(末尾の空白も含む)、先頭および末尾の引用符、二重にされた埋め込み引用符を含む十分な長さが必要です。たとえば、引数が ABC でその末尾に 3 つの空白が続く場合、“ABC###”を保持するために受け取る変数の長さは 8 文字以上であることが必要です。(文字#は空白を表します)。受け取るフィールドが長すぎると、QUOTE 関数は空白の文字列を返し、無効な引数というメモを SAS ログに書き込みます。

二重引用符で囲まれた文字列は DS2 識別子であり、文字定数ではありません。二重引用符は識別子の一部になります。引用符で囲まれた識別子を使用して、出力テーブルに列名を作成することはできません。

例

次のステートメントは、QUOTE 関数を示しています。

ステートメント	結果
a='A"B'; b=quote(a);	"A""B"
a='A"B'; b=quote(a);	"A'B"
a='Paul"s Catering Service '; b=quote(trim(a));	"Paul's Catering Service"

RAND 関数

指定する分布から乱数を生成します。

構文

RAND(*distribution*, *parameter-1*, ..., *parameter-k*)

必須引数

distribution
分布を特定する文字定数、変数または式です。

有効な分布は、次のとおりです。

分布	関数呼び出し	パラメーター値
Bernoulli	RAND('BERNoulli', prob)	ここでは、(0 <= prob <= 1)
ベータ	RAND('BETA', alpha, beta)	ここでは、(0.01 <= alpha <= 1.5e6)および(0.20 <= beta <= 1.5e6)
二項	RAND('BINOmial', prob, trials)	ここでは、(0 <= prob <= 1)および(0 <= trials)
Cauchy	RAND('CAUChy')	
カイ 2 乗	RAND('CHISquare', df)	ここでは、0.03 <= df <= 4e9
Erlang	RAND('ERLAng', alpha)	ここでは、1 <= alpha <= 2e9
指数	RAND('EXPOnential', scale)	ここでは、0 <= scale <= 1e300
極値	RAND('EXTReme')	デフォルト m = 0、scale = 1、shape = 0
	RAND('EXTReme', mu)	ここでは、–1e300 <= mu <= 1e300
	RAND('EXTReme', mu, scale)	ここでは、scale > 0
	RAND('EXTReme', mu, scale, shape)	ここでは、(scale > 0)および(–0.5 <= shape <= 5)
F	RAND('F', numdf, dendf)	ここでは、(0.025 <= numdf <= 1e7)および(0.1 <= dendf <= 2e9)
ガンマ	RAND('GAMMa', alpha)	ここでは 0.015 <= alpha <= 2e9、デフォルト scale = 1
	RAND('GAMMa', alpha, scale)	ここでは、(1e–80 <= scale <= 1e295)または (scale = 0)
幾何	RAND('GEOMetric', prob)	ここでは、0 < prob <= 1
Gompertz	RAND('GOMPertz')	デフォルト shape = 1、scale = 1

分布	関数呼び出し	パラメーター値
	RAND('GOMPertz' shape)	ここでは、shape >= 1e–300
	RAND('GOMPertz' shape , scale)	ここでは、(shape >= 1e–300)および(scale >= 0)
Gumbel	RAND('GUMBel')	デフォルト mu = 0、scale = 1
	RAND('GUMBel', mu)	ここでは、–1e300 <= mu <= 1e300
	RAND('GUMBel', mu, scale)	ここでは、(-1e300 <= mu <= 1e300)および (scale >= 0)
超幾何	RAND('HYPErgeometric', pop_size, cat_size, samp_size)	ここでは、1 <= pop_size <= 9e15 および 0 <= cat_size <= pop_size および 1 <= samp_size <= pop_size
一様整数	RAND('INTEger', int)	ここでは、–2147483648 <= int <= 2147483647
Laplace	RAND('LAPLace')	デフォルト mu = 0、scale = 1
	RAND('LAPLace', mu)	ここでは、–1e300 <= mu <= 1e300
	RAND('LAPLace', mu, scale)	ここでは、(–1e300 <= mu <= 1e300)および (scale >= 0)
ロジスティック	RAND('LOGIstic')	デフォルト mu = 0、scale = 1
	RAND('LOGIstic', mu)	ここでは、–1e300 <= mu <= 1e300
	RAND('LOGIstic', mu, scale)	ここでは、(–1e300 <= mu <= 1e300)および (scale >= 0)
対数正規	RAND('LOGNormal')	デフォルト mu = 0、sigma = 1
	RAND('LOGNormal', mu)	ここでは、ABS(mu) <= 702
	RAND('LOGNormal', mu, sigma)	ここでは、(ABS(mu) + 7*sigma <= 709)および ((sigma >= max(1, ABS(mu)) * 1e–13)または (sigma = 0))
負の二項	RAND('NEGBinomial', prob, n)	ここでは、(0 < prob <= 1)および(n >= 1)
正規またはガウス	RAND('NORMal')	デフォルト mu = 0、sigma = 1
	RAND('NORMal', mu)	ここでは、ABS(mu) <= 1e14
	RAND('NORMal', mu, sigma)	ここでは、(ABS(mu) <= 1e14*sigma)または (sigma = 0)

分布	関数呼び出し	パラメーター値
パレート	RAND('PAREto')	デフォルト shape = 1、scale = 1
	RAND('PAREto', shape)	ここでは、shape .04 <= xi <= 1e10
	RAND('PAREto', shape, scale)	ここでは、(shape .04 <= xi <= 1e10)および(0 <= scale <=1e100)
Poisson	RAND('POISson', mean)	ここでは、0 <= mean <= 1e300
シフトされた Gompertz	RAND('SHGOmpertz')	デフォルト shape = 1、inverse_scale = 1
	RAND('SHGOmpertz', shape)	ここでは、shape >= 1e–300
	RAND('SHGOmpertz', shape, inverse_scale)	ここでは、(shape >= 1e–300)および (inverse_scale >= 1e-300)
T	RAND('T', df)	ここでは、df >= 0.05
テーブル	RAND('TABLe', p_1, ..., p_n)	ここでは、(0 <= pi <= 1)および(p_1 + ... + p_n <= 1)
三角	RAND('TRIAngular', height)	ここでは、0 <= height <= 1
一様	RAND('UNIForm')	デフォルト a = 1、b = 0
	RAND('UNIForm', a)	(min(a,0) max(a,0))で一様
	RAND('UNIForm', a, b)	(min(a,b) max(a,b))で一様
Wald (逆ガウス)	RAND('WALD', shape)	ここでは 1e–18 <= shape <= 1e17、デフォルト mu = 1
	RAND('WALD', shape, mu)	ここでは、(shape >= 1e–18)および(mu >= 1e–10)および(1e–21 <= shape/mu <= 1e17)
Weibull	RAND('WEIBull', shape)	ここでは、0.02 <= shape <= 1e13、デフォルト scale = 1
	RAND('WEIBull', shape, scale)	ここでは、(0.02 <= shape <= 1e13)および(1e–100 <= scale <= 1e20)

注: F および T を除き、最初の 4 文字で分布を最小限に識別できます。

注: *distribution* 引数の値を指定しない場合は、一様分布が使用されます。

parameter-1, ...,parameter-k
特定の分布に適した*形状*、*位置*または*尺度*パラメーターの値を指定する数値定数、変数または式です(省略可能)。

例

この例では、RAND 関数と STREAMINIT 関数を使用しています。

```
proc cas;
  streaminit(1234);
run;
  do i=1 to 1000;
    z[i]=0;
  end;
  loops=100000;
  maxnum=1000;
  do i=1 to loops;
    x=RAND('UNIFORM');
    y=(int64)(x*maxnum)+1;
    z[y]=z[y]+1;
  end;
  sleep(1);
  print "Expected mean: " loops/maxnum;
  mean=mean(z);
  std=std(z);
  print "Mean       : " put(mean,d6.3);
  print "Std        : " put(std,d6.3);
  print "std/mean   : " put(std/mean,d6.3);
run;
```

アウトプット 4.6　SAS ログ

```
Expected mean: 100
Mean         :  100.0
Std          :  9.759
std/mean     : 0.0976
```

RANGE 関数

最大値と最小値の差を返します。

返されるデータの種類: DOUBLE

構文

RANGE(*expression* <, ...*expression*>)

引数

expression
数値に評価される有効な式を指定します。

要件	少なくとも 1 つの非 null または非欠損の引数が必要です。それがない場合は、関数から null または欠損値が返されます。
データの種類	DOUBLE

詳細

RANGE 関数は、最大と最小の非 null または非欠損引数の差を返します。

例

次のステートメントは、RANGE 関数を示しています。

ステートメント	結果
a=range(.,.);	.
a=range(-2,6,3);	8
a=range(2,6,3,.);	4
a=range(1,6,3,1);	5

RANK 関数

ASCII または EBCDIC 照合順序で文字の位置を返します。

返されるデータの種類:	DOUBLE

構文

RANK(*expression*)

引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

詳細

RANK 関数は、文字式の最初の文字の位置を表す整数を返します。

例

次のステートメントは、RANK 関数を示しています。

ステートメント	結果	
	ASCII	EBCDIC
a=rank('A');	65	193

REPEAT 関数

文字式を繰り返します。

返されるデータの種類: CHAR、NCHAR、NVARCHAR、VARCHAR

構文

REPEAT(*expression*, *n*)

引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

n
expression を繰り返す回数を指定します。

制限事項　*n* は、0 以上である必要があります。

データの種類　BIGINT、DOUBLE

詳細

REPEAT 関数は、*n* 回繰り返された第 1 引数で構成される文字値を返します。つまり、第 1 引数が *n*+1 回、結果に表示されます。

例

次のステートメントは、REPEAT 関数を示しています。

ステートメント	結果
a=repeat('ONE',2);	ONEONEONE

REVERSE 関数

文字式を逆にします。

返されるデータの種類:　CHAR、NCHAR、NVARCHAR、VARCHAR

構文

REVERSE(*expression*)

引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類　CHAR、NCHAR、NVARCHAR、VARCHAR

詳細

REVERSE 関数が返す文字値では、式の最後の文字が結果の最初の文字になり、式の最後から 2 番目の文字が結果の 2 番目の文字になります(以下同様です)。

注: 式の末尾の空白は結果の先頭の空白になります。

例

次のステートメントは、REVERSE 関数を示しています。

ステートメント	結果
	----+----1
a=reverse('xyz ');	zyx

RIGHT 関数

文字式を右揃えにします。

返されるデータの種類: CHAR、NCHAR、NVARCHAR、VARCHAR

構文

RIGHT(*expression*)

引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

詳細

RIGHT 関数は、末尾の空白を値の先頭に移動した引数を返します。引数の長さは変わりません。

例

次のステートメントは、RIGHT 関数を示しています。

ステートメント	結果
	----+----1----+

a='Due Date ';
b=print(a, $10.);
c=print(right(a),$10.);
	Due Date
	  Due Date

RMS 関数

2 乗平均平方根を返します。

返されるデータの種類: DOUBLE

構文

RMS(*expression* <, ...*expression*>)

引数

expression
数値に評価される有効な式を指定します。

データの種類 DOUBLE

詳細

2 乗平均平方根は、値の 2 乗の算術平均の平方根です。すべての引数が null または欠損値の場合、結果は null または欠損値になります。それ以外の場合、結果は非 null または非欠損値の 2 乗平均平方根になります。

nを非 null または非欠損値を持つ引数の数とし、$x_1, x_2, \ldots, x_n$をそれらの引数の値とします。2 乗平均平方根は次のように計算されます。

$$\sqrt{\frac{x_1^2 + x_2^2 + \ldots + x_n^2}{n}}$$

例

次のステートメントは、RMS 関数を示しています。

ステートメント	結果
x1=rms(1,7);	5
x2=rms(.,1,5,11);	7

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。

返されるデータの種類: DOUBLE

構文

ROUND(*expression* <, *rounding-unit*>)

引数

expression
数値に評価される丸め対象の有効な式を指定します。

データの種類 DOUBLE

rounding-unit
丸め単位を示す正の数値式を指定します。

データの種類 DOUBLE

詳細

基本概念

ROUND 関数は、第 1 引数を第 2 引数の倍数の近似値に丸めます。結果は第 2 引数の正確な倍数でない場合があります。

2 進算術演算と 10 進算術演算の差異

コンピューターは、有限精度の 2 進算術演算を使用します。通常、厳密な 2 進表現がない数値を扱う場合、コンピューターによって生成される結果は 10 進算術演算で生成される結果とわずかに異なります。

たとえば、10 進値の 0.1 および 0.3 には厳密な 2 進表現はありません。10 進算術演算の 3*0.1 は厳密に 0.3 になりますが、2 進算術演算ではこの等式は成り立ちません。

次の例が示すように、**a** が float で **b** が REAL の場合、2 つの値には差があります。

```
proc cas;
    a=0.3;
    b=3*0.1;
    diff=a-b;
    print "a=" a;
    print "b=" b;
    print "diff=" diff;
run;
```

次の行が SAS ログに書き込まれます。

```
a=0.3
b=0.3
diff=-5.55112E-17
```

動作環境の情報: 前述の例は、Windows 環境で実行されています。他の動作環境を使用する場合、結果は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

DS2 プログラムで文字"0.3"が定数として示されている場合、値は 3/10 として計算されます。ROUND(0.33,0.1)は、標準の入力形式にあわせて結果を 3/10 として計算します。次のステートメントでは期待される結果が生成されます。

```
if round(x,0.1) = 0.3 then
  ... more DS2 statements ...
```

ただし、次のステートメントのように定数 0.3 のかわりに変数 Y を使用する場合、Y の計算方法によっては予期しない結果が生じる可能性があります。

```
if round(x,0.1) = y then
  ... more DS2 statements ...
```

ROUND が標準の入力形式を使用して Y を文字"0.3"として読み込む場合、IF ステートメントに定数 0.3 が示されている場合と同じ結果になります。ROUND が別の入力形式を使用して Y を読み込む場合や、SAS 以外のプログラムが Y を読み込む場合、

文字"0.3"では厳密な 3/10 の値が生成されない可能性があります。また、厳密な 2 進表現ではない数値が計算に含まれている場合や、浮動小数点表現が異なる別の動作環境にテーブルを移植する場合も、正確な結果を得られない可能性があります。

Y が小数第 1 位の 10 進数だとわかっているが、標準の入力形式で生成される値とまったく同じかどうかわからない場合、次のステートメントを使用することをお勧めします。

```
if round(x,0.1) = round(y,0.1) then
  ... more DS2 statements ...
```

期待される結果の生成

通常、ROUND(expression, rounding-unit)は、結果の有効桁数が 9 以下で、次のどの条件にも該当しない場合、10 進算術演算で期待される結果を生成します。

- 丸め単位が整数である。
- 丸め単位が 1e-15 以上の 10 のべき乗である。 [1]
- 10 進算術演算で期待される結果が小数第 4 位以下である。

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 [2]、ROUND 関数は整数で除算して結果を計算します。そのため、丸め単位の倍数ではなく 2 つの整数の比率を使用して ROUND の結果を比較できます。

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。

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。

1. 丸め単位が 1 よりも小さい場合、ROUND は、丸め単位の逆数が 10 のべき乗との違いが最下位から第 3 または第 4 ビットまでであれば、丸め単位を 10 のべき乗として扱います。
2. ROUND は、丸め単位の逆数と整数の違いが最下位から第 3 または第 4 ビットまでであれば、丸め単位を整数の逆数として扱います。

比較

ROUND 関数は ROUNDE 関数とほぼ同じですが、第 1 引数が第 2 引数の最も近い 2 つの倍数の中間にある場合に ROUNDE が偶数の倍数を返す点が異なります。
ROUND は、絶対値が大きい方の倍数を返します。

ROUNDZ 関数は、10 進算術演算で計算された結果と結果を一致させようとせずに、丸め単位の倍数を返します。

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。

返されるデータの種類: DOUBLE

構文

ROUNDE(*expression* <, *rounding-unit*>)

引数

expression
数値に評価される丸め対象の有効な式を指定します。

データの種類 DOUBLE

rounding-unit
丸め単位を示す正の数値式です。

デフォルト 1

データの種類 DOUBLE

詳細

ROUNDE 関数は、第 1 引数を第 2 引数の最も近い倍数に丸めます。

比較

ROUNDE 関数は ROUND 関数とほぼ同じですが、第 1 引数が第 2 引数の最も近い 2 つの倍数の中間にある場合に ROUNDE が偶数の倍数を返す点が異なります。ROUND は、絶対値が大きい方の倍数を返します。

例

次の例では、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。

```
proc cas;
  x=0;
    do x=0 to 4 by .25;
    rounde=rounde(x);
    round=round(x);
  print "rounde=" rounde;
  print "round=" round;
    end;
run;
```

次のように SAS ログに出力されます。

```
rounde=0    round=0
rounde=0    round=0
rounde=0    round=1
rounde=1    round=1
rounde=1    round=1
rounde=1    round=1
rounde=2    round=2
rounde=2    round=2
rounde=2    round=2
rounde=2    round=2
rounde=2    round=3
rounde=3    round=3
rounde=3    round=3
rounde=3    round=3
rounde=4    round=4
rounde=4    round=4
rounde=4    round=4
```

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。

返されるデータの種類: DOUBLE

構文

ROUNDZ(*expression* <, *rounding-unit*>)

引数

expression
数値に評価される有効な式を指定します。

データの種類 DOUBLE

rounding-unit
数値式に評価され、丸め単位を示す有効な式を指定します。

デフォルト 1

要件 正の値のみが有効です。

データの種類 DOUBLE

詳細

ROUNDZ 関数は、第 1 引数を第 2 引数の最も近い倍数に丸めます。

比較

ROUNDZ 関数は、ROUND 関数と同じですが、次の点が異なります。

- ROUNDZ は、第 1 引数が第 2 引数の最も近い 2 つの倍数のちょうど中間にある場合には偶数の倍数を返します。ROUND は、第 1 引数が最も近い 2 つの倍数のほぼ中間にある場合には絶対値が大きい方の倍数を返します。
- 丸め単位が 1 よりも小さい場合や整数の逆数でない場合、ROUNDZ によって返される結果は 10 進算術演算の結果と完全には一致しない可能性があります。ROUNDZ は、結果をファジー処理しません。ROUND は、結果が 10 進算術演算に一致するようにファジーと呼ばれる追加の計算を実行します。

例

例 1: ROUNDZ 関数と ROUND 関数の結果の比較

次の例では、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。

```
proc cas;
   do i=10 to 17;
     Value=2.5 - 10**(-i);
```

```
    Roundz1=roundz(value);
    Round1=round(value);
    print "Roundz1=" Roundz1;
    print "Round1=" Round1;
  end;
  do i=16 to 12 by -1;
    value=2.5 + 10**(-i);
    roundz=roundz(value);
    round=round(value);
    print "roundz=" roundz;
    print "round=" round;
  end;
run;
```

次の出力は結果を示しています。

```
Roundz1=2    Round1=3
Roundz1=2    Round1=3
Roundz1=2    Round1=3
Roundz1=2    Round1=3
Roundz1=2    Round1=3
Roundz1=2    Round1=3
Roundz1=2    Round1=3
Roundz1=2    Round1=3
roundz=2     round=3
roundz=2     round=3
roundz=2     round=3
roundz=2     round=3
roundz=2     round=3
```

例 2: ROUNDZ 関数の出力例

次のステートメントは、ROUNDZ 関数を示しています。

ステートメント	結果

a=223.456;
b=roundz(a,1);
c=(put(b), 9.5);
	223.00000

var2=223.456;
x=roundz(var2,.01);
print x 9.5;
	223.46000

x=roundz(223.456,100); print x 9.5;	200.00000
x=roundz(223.456,.3); print x 9.5;	223.50000

SAVING 関数

定期預金の将来価値を返します。

返されるデータの種類: DOUBLE

構文

SAVING(*f*, *p*, *r*, *n*)

引数

f
数値の将来総額です(*n* 期間の終了時)。

範囲 $f \geq 0$

データの種類 DOUBLE

p
数値の固定定期的支払いです。

範囲 $p \geq 0$

データの種類 DOUBLE

r
10 進で表される数値の定期利率です。

範囲 $r \geq 0$

データの種類 DOUBLE

n
整数の複利期間数です。

範囲 $n \geq 0$

データの種類 DOUBLE

詳細

SAVING 関数は、定期預金の 4 つの引数のリストの欠損引数を返します。これらの引数には次の式の関係があります。

$$f = \frac{p(1+r)\left((1+r)^n - 1\right)}{r}$$

引数は 1 つを欠損値とする必要があります。次に、残りの 3 つから計算されます。結果を変換して丸めた数字にする調整は行われません。

例

ある定期預金口座は 5%の名目年利が支払われ、毎月複利計算されます。毎月$100 を入金する場合、少なくとも累計が$12,000 になるために必要な支払い回数は、次のように表すことができます。

```
number=saving(12000, 100, .05/12, .);
```

返される値は 97.18 か月です。第 4 引数には欠損値が設定されます。これは、支払い回数を計算することを示します。5%の名目年利は、0.05/12 の月利に変換されます。rate は分数(パーセントではない)で表す複利計算期間当たりの利率です。

SAVINGS 関数

変動金利を使用して定期預金の残高を返します。

返されるデータの種類: DOUBLE

構文

SAVINGS(*base-date*, *initial-deposit-date*, *deposit-amount*, *deposit-number*, *deposit-interval*, *compounding-interval*, *date-1*, *rate-1*<, ...*date-n*, *rate-n*>)

引数

base-date
返される値が基準日の預金残高になるように指定します。

要件 *base-date* は SAS 日付です。

データの種類 DOUBLE

initial-deposit-date
初回入金日を指定します。後続の入金間隔の開始時に次の入金が行われます。

要件 *initial-deposit-date* は SAS 日付です。

データの種類 DOUBLE

deposit-amount
各入金の値を指定します。すべての入金は定数を前提としています。

データの種類 DOUBLE

deposit-number
入金の回数を指定します。

データの種類 DOUBLE

deposit-interval
入金の頻度を指定します。

要件 *deposit-interval* は SAS 間隔です。

データの種類 CHAR

compounding-interval
複利間隔を指定します。

要件 *compounding-interval* は SAS 間隔です。

データの種類 CHAR

date
rate が有効になるタイミングを指定します。各日付は利率とペアになります。

要件 *date* は SAS 日付です。

データの種類 DOUBLE

rate
date に開始する利率を数値のパーセンテージで指定します。各利率は日付とペアになります。

データの種類 DOUBLE

詳細

SAVINGS 関数には、次の詳細情報が適用されます。

- 利率の値は–99 から 120 までの値にする必要があります。
- *deposit-interval* は'CONTINUOUS'にすることはできません。
- 日付-利率ペアのリストは日付順にする必要はありません。
- 1 日に複数の利率が変更される場合、SAVINGS 関数は、その日付でリストされている最後の利率のみを適用します。
- 一部の期間には単利が適用されます。
- *initial-deposit-date* と *base-date* の両方の日付またはその前の日付となる有効な日付-利率ペアが必要になります。

例

- 年利 4%で年 4 回複利計算される口座に 2 年間毎月$300 を入金する場合、5 年後の口座残高は次のように表すことができます。

```
proc cas;
    bd=16437;
    idd=14612;
    d=14612;
    amount_base1=savings(bd, idd, 300, 24, 'month', 'qtr', d, 4.00);
    print "amount_base1=" amount_base1;
run;
```

次の行が SAS ログに書き込まれます。

```
amount_base1=8458.7145034
```

- 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。

```
proc cas;
    bd=14976;
    idd=14610;
    d=14610;
  amount_base3 = savings(bd, idd, 300, 24, 'month', 'qtr', d, 4);
    print "amount_base3=" amount_base3;
run;
```

次の行が SAS ログに書き込まれます。

```
amount_base3=3978.6903712
```

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

文字式から *n* 番目の単語を返します。

返されるデータの種類: CHAR、NCHAR、NVARCHAR、VARCHAR

構文

SCAN(*expression*, *n* <, *delimiters*<, *modifier*>>)

引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

n

SCAN で選択する文字式の単語番号を指定するゼロ以外の数値式です。次のルールが適用されます。*n* が正の場合、SCAN は文字列内の単語を左から右にカウントします。*n* が負の場合、SCAN は文字列内の単語を右から左にカウントします。*n* が *expression* の単語数より大きい場合、SCAN は空白値を返します。

SCAN で選択する文字式の単語番号を指定するゼロ以外の数値式です。次の規則が適用されます。

- *n* が正の場合、SCAN は文字列内の単語を左から右にカウントします。
- *n* が負の場合、SCAN は文字列内の単語を右から左にカウントします。
- *n* が *expression* の単語数より大きい場合、SCAN は空白値を返します。

delimiters

文字列を評価するか文字列に強制変換できて、SCAN で式の単語区切りとして使用される有効な式を指定します。

デフォルト

要件 *delimiter* が定数の場合は、*delimiter* を単一引用符で囲みます。

操作 ASCII のデフォルトの区切り文字は、空白 ! $ % & ( ) * + , – . / ; < | です。^文字のない環境では、SCAN はかわりに~文字を使用します。

EBCDIC のデフォルトの区切り文字は、空白 ! $ % & ( ) * + , – . / ; < ¬ | ¢ です。

修飾子を指定すると、*delimiter* の文字を変更できます。たとえば、*modifier* 引数で K 修飾子を指定すると、*delimiter* に含まれていないすべての文字が区切り文字として使用されます。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

ヒント 他の修飾子を使用して、*delimiter* にさらに文字を追加できます。

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します。

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、右から左に逆方向にスキャンします。

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 が指定されている場合、文字のリストにある文字は、区切り文字であるために省略されるのではなく、返された値に保持されます。K を指定しない場合、文字のリストに含まれているすべての文字が区切り文字として扱われます。

l 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、および英字を追加します。

o または O
: *charlist* および *modifier* 引数を 1 回だけ処理します。SCAN 関数の呼び出し時毎には処理されません。DATA ステップ(WHERE 句以外)または SQL プロシジャで O 修飾子を使用すると、*character-list* 引数および *modifier* 引数に変更がないループで SCAN が呼び出されるとき、より迅速に実行できます。O 修飾子は SAS コードの SCAN 関数の各インスタンスに個別に適用され、SCAN 関数のすべてのインスタンスで同じ区切り文字および修飾子が使用されるようには*なりません*。

p 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、対になっていない引用符が含まれている場合、左から右へのスキャンと、右から左へのスキャンとでは異なる単語が生成されます。

r 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 修飾子を指定すると、SCAN 関数は単語から先頭および末尾の空白をまず削除します。次に、SCAN は単語が引用符で始まる場合には単語から引用符の層も 1 つ削除します。

s 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トリムします。両方の文字引数ではなく 1 つの文字引数からのみ末尾の空白を削除する場合、T 修飾子を指定して SCAN 関数を使用するかわりに TRIM 関数を使用します。

u 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。

制限事項　この引数は、SAS Viya プラットフォームと CAS サーバーでのみサポートされています。

ヒント　*modifier* 引数が文字定数の場合は、引数を引用符で囲みます。一組の引用符で複数の修飾子を指定します。引数 *modifier* には、文字変数や文字式も指定できます。

詳細

"区切り文字"と"単語"の定義

*区切り文字*とは、単語を区切るために使用される複数の文字のどれかです。*delimiter* 引数と *modifier* 引数で区切り文字を指定できます。

Q 修飾子を指定すると、引用符で囲まれた部分文字列内部の区切り文字は無視されます。

SCAN 関数では、"単語"は、次の特徴をすべて備えた部分文字列を示します。

- 左側が区切り文字で囲まれているか、文字列の先頭である
- 右側が区切り文字で囲まれているか、文字列の末尾である
- 区切り文字が含まれていない

文字列の先頭または末尾に区切り文字がある場合、または文字列に 2 つ以上の連続する区切り文字が含まれている場合、単語の長さがゼロになることがあります。ただし、SCAN 関数では、M 修飾子を指定しなければ、長さがゼロの単語は無視されます。

注: ”単語”の定義は SCAN 関数と COUNTW 関数に共通します。

ASCII および EBCDIC 環境でのデフォルト区切り文字の使用

2 つの引数のみを指定して SCAN 関数を使用する場合、デフォルトの区切り文字は、コンピューターで使用している文字(ASCII または EBCDIC)によって異なります。

- コンピューターで 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。

- お使いのコンピューターが EBCDIC 文字を使用している場合、デフォルトの区切り文字は次のようになります。

  空白 ! $ % & ( ) * + , - ./ ; < ¬ | ¢

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、*delimiter* 引数によって明示的に指定された区切り文字のリストに追加されます。修飾子は、デフォルトの修飾子のリストには追加しません。

結果の長さ

式の最初の単語の前にある先頭の区切り文字は、SCAN に影響しません。連続する区切り文字が 2 つ以上ある場合、SCAN はそれらを 1 つとして扱います。

DS2 では、SCAN 関数が、まだ長さが指定されていない変数に値を返す場合、その変数には第 1 引数の長さが指定されます。SCAN 関数で第 1 引数の長さとは異なる値を変数に割り当てる必要がある場合は、SCAN 関数を使用するステートメントの前に、その変数の DECLARE ステートメントを使用します。

SCAN 関数によって返される単語の最小長は、M 修飾子が指定されているかどうかによって異なります。

M 修飾子ありで SCAN 関数を使用

M 修飾子を指定すると、文字列内の単語数は文字列内の区切り文字数に 1 を足した数になります。ただし、Q 修飾子を指定すると、引用符内の区切り文字は無視されます。

M 修飾子を指定すると、SCAN 関数は次のいずれかの条件に該当する場合に長さがゼロの単語を返します。

- 文字列の先頭が区切り文字であり、ユーザーが最初の単語を要求した場合
- 文字列の末尾が区切り文字であり、ユーザーが最後の単語を要求した場合
- 文字列が 2 つの連続する区切り文字を含んでおり、ユーザーがこれら 2 つの区切り文字間にある単語を要求した場合

M 修飾子なしで SCAN 関数を使用

M 修飾子を指定しない場合、文字列内の単語数は連続する非区切り文字の最大部分文字列数になります。ただし、Q 修飾子を指定すると、引用符内の区切り文字は無視されます。

M 修飾子を指定しないと、SCAN 関数は次のように動作します。

- 文字列の最初または最後の区切り文字は無視されます。
- 2 つ以上の連続する区切り文字を、単一の区切り文字のように扱います。

文字列に区切り文字以外の文字が含まれていない場合や、文字列の単語数よりも絶対値の大きい count を指定する場合、SCAN 関数は次のいずれかの項目を返します。

- 1 つの空白(DATA ステップから SCAN 関数を呼び出す場合)
- 長さがゼロの文字列(マクロプロセッサから SCAN 関数を呼び出す場合)

null 引数の使用

SCAN 関数では、文字引数を NULL に指定できます。ヌル引数は長さがゼロの文字列として扱われます。数値引数はヌルにできません。

SBCS および DBCS データの処理

SCAN 関数は、SBCS データを処理するように設計されていますが、DBCS データを処理することもできます。基準は次のとおりです。

- *expression* が VARCHAR として宣言されておらず、シングルバイトデータを処理している場合、SCAN は SBCS を処理します。
- *string* が VARCHAR として宣言されていて、マルチバイトデータを処理している場合、SCAN は DBCS を処理します。

例: 基本的な SCAN 関数の使用法

次のステートメントは、SCAN 関数を示しています。

ステートメント	結果
expr='ABC.DEF(X=y)'; word=scan(expr,3);	X=Y
scan('ABC.DEF(X=Y)',-3);	ABC

SEC 関数

正割を返します。

返されるデータの種類: DOUBLE

構文

SEC(*expression*)

引数

expression
数値に評価される有効な式をラジアンで表して指定します。

制限事項 *expression* は、PI/2 の奇数倍にはできません。

比較

SEC 関数は COS 関数と関係があります。

sec(x)=1/cos(x)

例

次のステートメントは、SEC 関数を示しています。

ステートメント	結果
x=sec(0.5);	1.13949392732454
y=sec(0);	1
z=sec(3.14159/3);	1.99999693590391

SECOND 関数

SAS 時間または日時値から秒を返します。

返されるデータの種類: DOUBLE

構文

SECOND(*time* | *datetime*)

引数

time
SAS 時間値を表す有効な式を指定します。

データの種類 DOUBLE

datetime
SAS 日時値を表す有効な式を指定します。

データの種類 DOUBLE

詳細

SECOND 関数は、特定の秒部分を表す数値を生成します。結果は、>= 0 かつ< 60 の数値になります。

例

次のステートメントは、SECOND 関数を示しています。

ステートメント	結果
a=hms(3,19,24); b=second(a);	11964 24
a=hms(6,25,65); s=second(a);	23165 5
a=hms(3,19,60); b=second(a);	12000 0

SIGN 関数

数値式の符号を示す数字を返します。

返されるデータの種類: DOUBLE

構文

SIGN(*expression*)

引数

expression
数値に評価される有効な式を指定します。

データの種類 すべての数値型

詳細

SIGN 関数は次の値を返します。

-1
expression < 0 の場合

0
expression = 0 の場合

1
expression > 0 の場合

例

次のステートメントは、SIGN 関数を示しています。

ステートメント	結果
a=sign(-5);	-1
a=sign(5);	1
a=sign(0);	0

SIN 関数

三角関数の正弦を返します。

返されるデータの種類: DOUBLE

構文

SIN(*expression*)

引数

expression
数値に評価される有効な式を指定します。

データの種類 DOUBLE

例

次のステートメントは、SIN 関数を示しています。

ステートメント	結果
a=sin(25.6)	0.45044059427538
a=sin(5);	-0.95892427466313

SINH 関数

双曲線正弦を返します。

返されるデータの種類: DOUBLE

構文

SINH(*expression*)

引数

expression
数値に評価される有効な式を指定します。

データの種類 DOUBLE

詳細

SINH 関数は、次の式によって得られる引数の双曲線正弦を返します。

$$e^{argument} - e^{-argument}/2$$

例

次のステートメントは、SINH 関数を示しています。

ステートメント	結果
a=sinh(0);	0
a=sinh(1);	1.1752011936438
a=sinh(-1.0);	-1.1752011936438

SKEWNESS 関数

歪度を返します。

返されるデータの種類: DOUBLE

構文

SKEWNESS(*expression-1, expression-2, expression-3* <, ...*expression-n*>)

引数

expression
数値に評価される有効な式を指定します。

要件 少なくとも 3 つの非 null または非欠損の引数が必要です。それがない場合は、関数から null または欠損値が返されます。

データの種類 DOUBLE

詳細

すべての非 null または非欠損引数の値が等しい場合、歪度は数学的に定義されておらず、SKEWNESS 関数は null または欠損値を返します。

例

次のステートメントは、SKEWNESS 関数を示しています。

ステートメント	結果
x1=skewness(0,1,1);	-1.73205080756887
x2=skewness(2,4,6,3,1);	0.59012865638436
x3=skewness(2,0,0);	1.73205080756887

SLEEP 関数

指定した期間、この関数を呼び出すプログラムの実行を中断します。

返されるデータの種類: DOUBLE

構文

SLEEP(*number-of-time-units* <, *time-unit*>)

引数

number-of-time-units
数値に評価され、プログラムの実行を中断する時間の単位数を示す有効な式を指定します。

範囲 $n \geq 0$

データの種類 DOUBLE

time-unit
number-of-time-units に適用される時間の単位を 10 のべき乗で指定します。たとえば、1 は秒に対応し、0.001 はミリ秒に対応します。

デフォルト Windows PC 環境では 1、その他の環境では 0.001

データの種類 DOUBLE

詳細

SLEEP 関数は、この関数を呼び出したプログラムの実行を指定した期間中断します。SLEEP 関数の最大スリープ期間は 46 日です。

例

例 1: 指定した期間の実行中断

次の例では、実行を 20 秒間遅らせます。

```
proc cas;
  ...CASL statements...
  time_slept=sleep(20,1);
```

```
  ...more CASL statements...
  print "time_slept=" time_slept;
run;
```

例 2: スリープ時間の計算に基づいた実行中断

次に、2011 年 6 月 15 日の午前 0 時まで実行を中断するように SAS に指示する例を示します。DS2 では、対象日とコードの実行開始日時に基づいて中断の長さが計算されます。

```
proc cas;
  ...CASL statements...;
  sleeptime=dhms(mdy(06,15,2011),00,00,00)-datetime();
  time_calc=sleep(sleeptime,1);
  ...more CASL statements...;
  print "time_calc=" time_calc;
run;
```

SMALLEST 関数

k 番目に小さい非 null 値または非欠損値を返します。

カテゴリ: CAS
記述統計

返されるデータの種類: DOUBLE

構文

SMALLEST(*k*, *expression* <, ...*expression*>)

引数

k
数値に評価されて返される有効な式を指定します。

データの種類 DOUBLE

expression
数値に評価されて処理される有効な式を指定します。

データの種類 DOUBLE

詳細

k が null または欠損値、0 未満、または値の数よりも大きい場合、結果は null または欠損値になります。

比較

SMALLEST 関数と ORDINAL 関数の違いは、SMALLEST は null 値と欠損値を無視しますが、ORDINAL は null 値と欠損値を数える点です。

SQRT 関数

値の平方根を返します。

返されるデータの種類: DOUBLE

構文

SQRT(*expression*)

引数

expression
負でない数値に評価される有効な式を指定します。

データの種類 DOUBLE

例

次のステートメントは、SQRT 関数を示しています。

ステートメント	結果
a=sqrt(36);	6
a=sqrt(25);	5
a=sqrt(4.4);	2.0976176963403

STD 関数

標準偏差を返します。

返されるデータの種類: DOUBLE

構文

STD(*expression-1, expression-2* <, ...*expression-n*>)

引数

expression
数値に評価される有効な式を指定します。

要件 少なくとも 2 つの非 null または非欠損の引数が必要です。それがない場合は、関数から null または欠損値が返されます。

データの種類 DOUBLE

例

次のステートメントは、STD 関数を示しています。

ステートメント	結果
a=std(2,6);	2.82842712474619
a=std(2,6,.);	2.82842714274619
a=std(2,4,6,3,1);	1.92353840616713

STDERR 関数

平均の標準誤差を返します。

カテゴリ: CAS
記述統計

返されるデータの種類: DOUBLE

構文

STDERR(*expression-1, expression-2* <, ...*expression-n*>)

引数

expression
数値に評価される有効な式を指定します。

要件 少なくとも 2 つの非 null または非欠損の引数が必要です。それがない場合は、関数から null または欠損値が返されます。

データの種類 DOUBLE

例

次の例は、STDERR 関数を示しています。

ステートメント	結果
a=stderr(2,6);	2
a=stderr(2,6,.);	2
a=stderr(2,4,6,3,1);	0.86023252670426

STRIP 関数

先頭と末尾の空白を削除して文字列を返します。

返されるデータの種類: CHAR、NCHAR、NVARCHAR、VARCHAR

構文

STRIP(*expression*)

引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

詳細

STRIP 関数は引数の先頭と末尾から空白をすべて削除して返します。引数が空白の場合、STRIP は長さがゼロの文字列を返します。

調整された値が受け取る変数の長さより短い場合、値の末尾に新しい空白が埋め込まれます。

注: 連結演算子では末尾の空白が削除されないため、STRIP 関数は連結を行う場合に便利です。

比較

STRIP 関数と TRIMN 関数の比較を次のリストに示します。

- 空白の文字列の場合、STRIP 関数と TRIM 関数はどちらも長さがゼロの文字列を返します。
- 先頭に空白がない文字列では、STRIP 関数と TRIM 関数は同一値を返します。

例

次の例では、STRIP 関数を使用して先頭と末尾の空白を削除します。

```
proc cas;
  string1="abcd";
  string2=" abcd ";
  string3=" abcd";
  string4="abcdefgh";
  string5="x y z";
  original1='*' || string1 || '*';
  original2='*' || string2 || ' *';
  original3=' *' || string3 || '*';
  original4='*' || string4 || '*';
  original5='*' || string5 || '*';
  stripped1='*' || strip (string1) || '*';
  stripped2='*' || strip (string4) || '*';
  stripped3='*' || strip (string5) || '*';

  columns={'string', 'original', 'stripped'};
  coltypes={'string', 'string', 'string'};
```

```
  row1={string1, original1, stripped1};
  row2={string2, original2, stripped1};
  row3={string3, original3, stripped1};
  row4={string4, original4, stripped3};
  row5={string5, original5, stripped3};
  striptab=newtable('striptab', columns, coltypes, row1, row2, row3, row4, row5);
  print striptab;
run;
```

アウトプット 4.7　STRIP 関数の結果

striptab: Results

string	original	stripped
abcd	*abcd*	*abcd*
abcd	* abcd *	*abcd*
abcd	* abcd*	*abcd*
abcdefgh	*abcdefgh*	*x y z*
x y z	*x y z*	*x y z*

SUBSTR (=の右) 関数

引数から部分文字列を返します。

返されるデータの種類: CHAR、NCHAR、NVARCHAR、VARCHAR

構文

SUBSTR(*string*, *position* <, *length*>)

必須引数

string
文字定数、文字変数、または文字式を指定します。

注: *string* を文字定数として指定する場合は、値を引用符で囲む必要があります。string を文字変数または文字式として指定する場合、値を引用符で囲む必要はありません。

参照項目: “例 1: SUBSTR=関数での string 値の指定”

position
抽出される部分文字列の開始位置を指定します。*position* は、数値定数、数値変数、または数式にすることができます。

参照項目: “例 2: SUBSTR=関数での position 値の指定”

オプション引数

length
抽出される部分文字列の長さを指定します。*length* は、数値定数、数値変数、または数式にすることができます。

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。SAS では、また、*length* 引数が無効であることを示す WARNING もログに出力します。

ヒント *length* を省略すると、式の残りの部分が抽出されます。“SUBSTR=関数での position 値の指定: position を数値定数として指定”の例を参照してください。

参照項目: “例 3: SUBSTR=関数での length 値を指定”

例

例 1: SUBSTR=関数での string 値の指定

例	結果

/* *string* を文字定数として指定 */
proc cas;
  zcode=substr("zip28413",4,5);
  put zcode;
run;
	28413

/* *string* を文字変数として指定 */
proc cas;
  z="zip28413";
  zcode=substr(z,4);
  put zcode;
run;
	28413

/* *string* を文字式として指定 */
proc cas;
  z="zip";
  code="28413";
  zcode=substr(z||code,4,5);
  put zcode;
run;
	28413

例 2: SUBSTR=関数での position 値の指定

例	結果

/* *position* を数値定数として指定 */
proc cas;
  name=substr("nameaddress",1,4);
  put name;
run;
	name

/* *position* を数値変数として指定
proc cas;
  position=1;
  name=substr("nameaddress",position,4);
  put name;
run;
	name

/* *position* を数値式として指定
proc cas;
  position=1+4;
  address=substr("name123Main",position,7);
  put address;
run;
	123Main

例 3: SUBSTR=関数での length 値を指定

表 4.4　SUBSTR=関数で *length* 値を指定

例	結果

/* *length* を数値定数として指定 */
proc cas;
  car="make##model";
  type=substr(car,7,5);
  put type;
run;
	model

/* *length* を数値変数として指定 */
proc cas;
  car="make##model";
  length=5;
  position=7;
  model=substr(car,position,length);
  put model;
run;
	model

/* *length* を数値式として指定
proc cas;
  length=1+4;
  model=substr("make##model",7,length);
  put model;
	model

例	結果
run;	

例 4: 無効な length 値の指定

例	結果
/* *length* を 0 に設定 */ proc cas; car="make##model"; type=substr(car,7,0); put type; run;	model WARNING: Argument 3 is an invalid argument
/* *length* を負の整数に設定 */ proc cas; car="make##model"; type=substr(car,7,-5); put type; run;	model WARNING: Argument 3 is an invalid argument.
/* *length* が *position* 値の適用後の *string* より長い*/ proc cas; car="make##model"; type=substr(car,7,9); put type; run;	model WARNING: Argument 3 is an invalid argument.

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。

返されるデータの種類: CHAR、NCHAR、NVARCHAR、VARCHAR

構文

SUBSTRN(*character-expression*, *position*<, *length*>)

必須引数

expression
文字列に評価されるか強制変換できる有効な式または数値を指定します。

データの種類　CHAR、NCHAR、NVARCHAR、VARCHAR

注　*expression* が数値の場合は、BEST32.出力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。

制限事項　*position* がない場合は空白スペースが返され、SAS ログに警告が書き込まれます。

position が文字列の最後を超えている場合は、空の文字列が返されます。

注　指定する *position* 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。

position と *length* の合計が文字列の長さよりも大きい場合は、文字列の最後までの位置が返されます。

position と *length* の合計が 0 より小さい場合は、空の文字列が返されます。

オプション引数

length
部分文字列の長さを指定する整数です。

制限事項　*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。

length が負の場合、SUBSTRN 関数は空の文字列を返します。

注　指定した *length* が文字列の最後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。

比較

次の表に、SUBSTRN 関数と SUBSTR 関数の比較を示します。

表 4.5　比較: SUBSTRN 関数と SUBSTR 関数

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す

条件	関数	結果
position の値が正でない	SUBSTR	SAS ログに警告を書き込み、空白の文字列を返す
length の値が正でない	SUBSTRN	長さがゼロの結果を返す
length の値が正でない	SUBSTR	SAS ログに警告を書き込み、空白の文字列を返す
指定した部分文字列が文字列の末尾を超える	SUBSTRN	結果を切り捨てる
指定した部分文字列が文字列の末尾を超える	SUBSTR	SAS ログに警告を書き込み、空白の文字列を返す

例

例 1: 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。

例のコード 4.1　SUBSTR の例

```
proc cas;
substr_result="*" || substr(1234.5678,2,6) || "*";
print substr_result;
run;
```

例のコード 4.2　SUBSTRN の例

```
substrn_result="*" || substrn(1234.5678,2,6)  || "*";
print substrn_result;
run;
```

SAS は次の出力をログに書き込みます。

例のコード 4.2　SUBSTR の例の結果

```
WARNING: Argument 1 should be a string.
* *
```

例のコード 4.3　SUBSTRN の例の結果

```
*234.56*
```

例 2: SUBSTRN 関数を使用した文字列の操作

この例では、SUBSTRN 関数を使用して文字列を操作する方法を示します。

```
proc cas;
columns = {"Position", "Length", "Result"};
coltypes={"integer", "integer", "string"};
table = newtable("Test", columns, coltypes);

xyz="abcd";
do position=-1 to 6;
  do length=max(-1,-position) to 7-position;
  substrn_result="*" || substrn(xyz, position, length) || "*";
  addrow(table,{position, length, substrn_result});
  end;
end;
run;
print table;
quit;
```

アウトプット 4.8　SUBSTRN 関数からの部分出力

table: Results

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*

関連項目

“SUBSTR (=の右) 関数” (552 ページ)

SUM 関数

非 null または非欠損引数の合計を返します。

返されるデータの種類: BIGINT、DECIMAL、DOUBLE

構文

SUM(*expression-1*, *expression-2* <, ...*expression-n*>)

引数

expression
数値に評価される有効な式を指定します。

要件 少なくとも 2 つの引数が必要です。

データの種類 BIGINT、DECIMAL、DOUBLE

詳細

null 値と欠損値は無視され、計算には含まれません。すべての引数が欠損値の場合、結果は欠損値になります。すべての引数が null 値の場合、結果は null 値になります。

この関数の引数が数値以外の場合、その引数は DOUBLE に変換されます。いずれかの引数が DOUBLE または REAL の場合、すべての引数が(まだ変換されていなければ) DOUBLE に変換され、結果は DOUBLE になります。それ以外の場合で、いずれかの引数が DECIMAL のときは、すべての引数が(まだ変換されていなければ) DECIMAL に変換され、結果は DECIMAL になります。それ以外の場合は、すべての引数が BIGINT に変換され、結果は BIGINT になります。

例

次のステートメントは、SUM 関数を示しています。

ステートメント	結果
a=sum(4,9,3,8);	24
a=sum(4,9,3,8,.);	24

SUMABS 関数

非欠損引数の絶対値の合計を返します。

返されるデータの種類: DOUBLE

構文

SUMABS(*value-1*<, ...*value-n*>)

引数

value
数値に評価される有効な式を指定します。

データの種類 DOUBLE

詳細

すべての引数が null または欠損値の場合、結果は null または欠損値になります。それ以外の場合は、結果は非欠損値の絶対値の合計です。

例

例 1: 絶対値の合計の計算

次の例では、非欠損引数の絶対値の合計を返します。

```
proc cas;
  x=sumabs(1,.,-2,0,3,.q,-4);
  print "x=" x;
run;
```

SAS は次の結果をログに書き込みます。

```
x=10
```

例 2: 変数リスト使用時の絶対値の合計の計算

次の例では、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。

```
proc cas;
```

```
x1=1;
x2=3;
x3=4;
x4=3;
x5=1;
x = sumabs(x1, x2, x3, x4, x5);
print "x=" x;
run;
```

SAS は次の結果をログに書き込みます。

```
x=12
```

TAN 関数

正接を返します。

返されるデータの種類: DOUBLE

構文

TAN(*expression*)

引数

expression
ラジアン単位の数値に評価される有効な式を指定します。

制限事項 *expression* は、π/2 の奇数倍にはできません。

データの種類 DOUBLE

例

次のステートメントは、TAN 関数を示しています。

ステートメント	結果
a=tan(0.5);	0.54630248984379
a=tan(0);	0
a=tan(3.14159/3);	1.73204726945457

TANH 関数

双曲線正接を返します。

返されるデータの種類: DOUBLE

構文

TANH(*expression*)

引数

expression
数値に評価される有効な式を指定します。

制限事項 *expression* は、π/2 の奇数倍にはできません。

データの種類 DOUBLE

詳細

TANH 関数は、次の式によって得られる引数の双曲線正接を返します。

$$\frac{e^{argument} - e^{-argument}}{e^{argument} + e^{-argument}}$$

例

次のステートメントは、TANH 関数を示しています。

ステートメント	結果
a=tanh(0);	0
a=tanh(0.5);	0.46211715726
a=tanh(-0.5);	-0.46211715726

TIME 関数

数値の SAS 時間値として現在時刻を返します。

返されるデータの種類: DOUBLE

構文

TIME( )

詳細

TIME 関数は引数を取りません。SAS 時間値の形式で現在時刻を生成します。

例

次のステートメントは、TIME 関数を示しています。

ステートメント	結果
a=time();	56526.0399990081
a=put(time(),time.);	15:42:06

TIMEPART 関数

SAS 日時値から時間値を抽出します。

返されるデータの種類: DOUBLE

構文

TIMEPART(*datetime*)

引数

datetime
SAS 日時値を表す有効な式を指定します。

データの種類 DOUBLE

例

次のステートメントは、TIMEPART 関数を示しています。

ステートメント	結果

proc cas;
ddtm=datetime();
tm=put(timepart(ddtm),time.);
print "ddtm=" ddtm;
print "tm=" tm;
run;
	ddtm=1841668215.1
	
	tm=14:30:15

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。

返されるデータの種類: DOUBLE

構文

TIMEVALUE(*base-date*, *reference-date*, *reference-amount*, *compounding-interval*, *date-1*, *rate-1*<, ...*date-n*, *rate-n*>)

引数

base-date
base-date における *reference-amount* の時間値を指定します。

要件 *base-date* は SAS 日付です。

データの種類 DOUBLE

reference-date
reference-amount の日付を指定します。

要件 *reference-date* は SAS 日付です。

データの種類 DOUBLE

reference-amount
reference-date における金額を指定します。

データの種類 DOUBLE

compounding-interval
複利間隔を指定します。

要件 *compounding-interval* は SAS 間隔です。

データの種類 CHAR

date
rate が有効になるタイミングを指定します。各日付は利率とペアになります。

要件 *date* は SAS 日付です。

データの種類 DOUBLE

rate
date に開始する利率を数値のパーセンテージで指定します。各利率は日付とペアになります。

データの種類 DOUBLE

詳細

TIMEVALUE 関数には次の詳細が適用されます。

- 利率の値は–99 から 120 までの値にする必要があります。
- 日付-利率ペアのリストは、日付順に並べ替える必要はありません。
- 1 日に複数の利率が変更される場合、TIMEVALUE 関数は、その日付でリストされている最後の利率のみを適用します。
- 一部の期間には単利が適用されます。
- *reference-date* と *base-date* の両方の日付またはその前の日付となる有効な日付-利率ペアが必要になります。

例

- 名目金利が 10%で 1 年間毎月複利計算される$1,000 の投資の累計額は、次のように表すことができます。

```
proc cas;
  bd=16437;
```

```
  rd=14612;
  d=14612;
  amount_base1=timevalue(bd, rd, 1000, 'month', d, 10);
  print "amount_base1=" amount_base1;
run;
```

- 半年後に利率が 20%に上昇した場合の計算は次のとおりです。

```
proc cas;
   bd=16437;
   rd=14610;
   d1=14610;
   d2=14976;
   amount_base2 = timevalue(bd, rd, 1000, 'month', d1, 10, d2, 20);
   print "amount_base2=" amount_base2;
run;
```

- 日付-利率ペアは、日付順に並べ替える必要はありません。この柔軟性があるため、amount_base2 と amount_base3 は同じ値になります。

```
proc cas;
   bd=16437;
   rd=14610;
   d1=14610;
   d2=14910;
   amount_base3 = timevalue(bd, rd, 1000, 'month', d1, 10, d2, 20);
   print "amount_base3=" amount_base3;
run;
```

TINV 関数

t 分布から分位点を返します。

返されるデータの種類: DOUBLE

構文

TINV(*p*, *df*<, *nc*>)

引数

p
数値の確率に評価される有効な式を指定します。

範囲 $0 < p < 1$

データの種類 DOUBLE

df
数値の自由度パラメーターに評価される有効な式を指定します。

範囲 *df* > 0

データの種類 DOUBLE

nc
数値の非心度パラメーターに評価される有効な式を指定します。

データの種類 DOUBLE

詳細

TINV 関数は、自由度が *df*、非心度パラメーターが *nc* のスチューデントの *t* 分布から *p*番目の分位点を返します。*t* 分布のオブザベーションが、返される分位点以下となる確率は *p* です。

TINV では、整数以外の自由度パラメーター *df* を指定できます。このパラメーター *nc* (省略可能)が指定されていないか 0 の場合、心度 *t* 分布から分位点が返されます。

注意
nc の値が大きいとアルゴリズムが失敗する場合があります。 その場合、欠損値が返されます。

比較

TINV は PROBT 関数の逆数です。

例

次のステートメントは、TINV 関数を示しています。

ステートメント	結果
x=tinv(.95, 2);	2.91998558035372
x=tinv(.95, 2.5, 3);	11.0338336251942

TODAY 関数

数値の SAS 日付値として現在の日付を返します。

返されるデータの種類: DOUBLE

構文

TODAY( )

詳細

TODAY 関数は引数を取りません。SAS 日付値(1960 年 1 月 1 日からの日数)の形式で現在の日付を生成します。

例

次のステートメントは、TODAY 関数を示しています。

ステートメント	結果

proc cas;
td=today();
tday=put(td, date.);
print "td=" td;
print "tday=" tday;
run;
	td=21315
	tday=11MAY18

TRANSLATE 関数

文字式内の特定の文字を置き換えます。

返されるデータの種類: CHAR、NCHAR、NVARCHAR、VARCHAR

構文

TRANSLATE(*expression*, *to-characters*, *from-characters*)

引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。*expression* には元の文字値が含まれます。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

to-characters
TRANSLATE で代替文字として使用する文字を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

from-characters
TRANSLATE で置換する文字を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

詳細

to-characters と *from-characters* の値は、個々の文字ごとに対応しています。たとえば、TRANSLATE では、*from-characters* の最初の文字が *to-characters* の最初の文字に変更されます。*to-characters* の文字数が *from-characters* の文字数より少ない場合、TRANSLATE は *from-characters* の余った文字を空白に変更します。*to-characters* の文字数が *from-characters* の文字数より多い場合、TRANSLATE は *to-characters* の余った文字を無視します。

比較

TRANSTRN 関数は、置換文字列にゼロの長さを使用できるという点で TRANWRD 関数と異なります。TRANWRD では、置換文字列の長さがゼロ場合はかわりに 1 つの空白を使用します。

TRANSLATE 関数は、ユーザーが指定したすべての文字を別の文字に変換します。TRANSLATE は 1 回の呼び出しで複数の文字をスキャンできます。ただし、このスキャンを実行すると、TRANSLATE は文字列に含まれる個々のすべての文字を検索します。つまり、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、単語(または文字のパターン)をスキャンし、それらの単語を 2 番目の単語(または文字のパターン)で置き換える点で TRANSLATE とは異なります。

例

次のステートメントは、TRANSLATE 関数を示しています。

ステートメント	結果

proc cas;
  a=translate('XYZW', 'AB', 'VW');
  print "a=" a;
run;
	a=XYZB

proc cas;
  string1='AABBAABABB';
  a=translate(string1, '12', 'AB');
  print "a=" a;
run;
	a=1122112122

TRANSTRN 関数

文字列に含まれる特定の部分文字列をすべて置き換えるか取り除きます。

返されるデータの種類: CHAR、NCHAR、NVARCHAR、VARCHAR

構文

TRANSTRN(*source-expression*, *target-expression*, *replacement-expression*)

引数

source-expression
文字列に評価されるか強制変換できて、変換する文字を含む有効な式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

target-expression
文字列に評価されるか強制変換できて、その文字が *source-expression* で検索される有効な式を指定します。

要件 *target-expression* の長さにはゼロより大きい値を指定する必要があります。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

replacement-expression
文字列に評価されるか強制変換できて、*target-expression* を置換する有効な式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

詳細

TRANSTRN 関数は、文字列に含まれる特定の部分文字列をすべて置き換えるか取り除きます。TRANSTRN 関数では、*target-expression* 文字列と *replacement-expression* 文字列の末尾の空白は取り除きません。出現する *target* をすべて取り除くには、*replacement-expression* に TRIMN(' ')を指定します。

比較

TRANSTRN 関数は、置換文字列にゼロの長さを使用できるという点で TRANWRD 関数と異なります。TRANWRD では、置換文字列の長さがゼロ場合はかわりに 1 つの空白を使用します。

TRANSLATE 関数は、ユーザーが指定したすべての文字を別の文字に変換します。TRANSLATE は 1 回の呼び出しで複数の文字をスキャンできます。ただし、このスキャンを実行すると、TRANSLATE は文字列に含まれる個々のすべての文字を検索します。つまり、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。

例

例 1: 単語のすべての出現箇所の置換

この例では、TRANSTRN 関数を使用して、**Mrs.**と **Miss** を **Ms.**に置き換えています。

```
proc cas;
   name1='Mrs.  Joan Smith';
   nameA=transtrn(name1, 'Mrs.', 'Ms.');
   print "nameA=" nameA;
   name2='Miss Alice Cooper';
   nameB=transtrn(name2, 'Miss', 'Ms.');
   print "nameB=" nameB;
run;
```

次の行が SAS ログに書き込まれます。

```
nameA=Ms.  Joan Smith
nameB=Ms. Alice Cooper
```

例 2: 検索文字列からの空白の除外

この例では、TRIM 関数を *target* とともに使用して、空白を除外しています。TRIM 関数を含めなかった場合、ターゲット文字列に空白が含まれるため、TRANSTRN 関数はソース文字列を置き換えません。

```
proc cas;
   salelist='CATFISH';
   target='FISH';
```

```
    replacement='NIP';
    salelist2=transtrn(salelist, trim(target), replacement);
    print "salelist2=" salelist2;
run;
```

次のように SAS ログに書き込まれます。

```
salelist2=CATNIP
```

例 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。DS2 では、2 つの引用符とその間にある空白で構成される文字定数は、長さがゼロの文字列ではなく 1 つの空白を表します。次の例では、*string1* の結果は *string2* の結果とは異なります。

```
proc cas;
   string1='*' || transtrn('abcxabc', 'abc', trimn(' ')) || '*';
   print "string1=" string1;
   string2='*' || transtrn('abcxabc', 'abc', ' ') || '*';
   print "string2=" string2;
run;
```

SAS は次の出力をログに書き込みます。

```
string1=*x*
string2=* x *
```

TRIGAMMA 関数

トリガンマ関数の値を返します。

返されるデータの種類: DOUBLE

構文

TRIGAMMA(*expression*)

引数

expression
数値に評価される有効な式を指定します。

制限事項 正でない整数は無効です。

データの種類 DOUBLE

詳細

TRIGAMMA 関数は、ディガンマ関数の導関数を返します。*expression* > 0 の場合、TRIGAMMA 関数は lgamma 関数の第 2 導関数です。

例

次のステートメントは、TRIGAMMA 関数を示しています。

ステートメント	結果
a=trigamma(3);	0.39493406684822

TRIM 関数

文字式から末尾の空白を取り除きます。文字式が欠損値の場合は、長さがゼロの文字列を返します。

別名: TRIMN

返されるデータの種類: CHAR、NCHAR、NVARCHAR、VARCHAR

構文

TRIM('*expression*')

引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

詳細

TRIM 関数では、文字の引数をコピーし、末尾の空白を取り除いて、その結果調整された引数を返します。引数が空白の場合、TRIM は長さがゼロの文字列を返します。連結では末尾の空白は取り除かれないため、TRIM は連結後に便利です。

注: TRIM 関数は、Unicode 標準で定義されている空白とスペースの 両方の文字を取り除きます。したがって、TRIM 関数は DBCS 空白とシフトアウト/シフトイン (SO/SI)エスケープコードも処理します。Unicode スペース文字標準の詳細については、Wikipedia: Unicode character property を参照してください。

例

次のステートメントは、TRIM 関数を示しています。

ステートメント	結果

proc cas;
  string1='Testscore ';
  string2='File.xls';
  result=trim(string1)||(string2);
  print "result=" result;
run;
	result=TestscoreFile.xls

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。

構文

TRIMN(*argument*)

必須引数

argument
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取り除いて、その結果調整された引数を取り除きます。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取り除かれないため、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。

例

```
data new;
  x="A"||trimn("")||"B";
  z=" ";
  y=">"||trimn(z)||"<";
  put x= y=;
run;
```

これらのステートメントでは、次の結果が生成されます。

```
x=AB
y=><
```

TRUNC 関数

数値を指定した長さに切り捨てます。

返されるデータの種類: DOUBLE

構文

TRUNC(*expression*, *length-expression*)

引数

expression
数値に評価される有効な式を指定します。

データの種類 DOUBLE

length-expression
数値に評価される有効な式を指定します。

範囲 3-8

データの種類 DOUBLE

詳細

TRUNC 関数では、完全な長さの数値式(DOUBLE として保存)を *length-expression* で指定したより小さいバイト数に切り捨て、切り捨てられたバイトを 0 で埋め込みます。切り捨てとその後の展開は、最初に完全な長さより短い数を保存した結果を複製し、次にそれを読み込みます。

例

次のステートメントは、TRUNC 関数を示しています。

ステートメント	結果
x=trunc(3.1,3);	3.099609375
x=trunc(3.1,4);	3.09999847412109
x=trunc(3.1,5);	3.09999999403953
x=trunc(3.1,6);	3.09999999997671
x=trunc(3.1,7);	3.0999999999999
x=trunc(3.1,8);	3.1

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
CAS

別名: RANUNI

制限事項: *この関数は非推奨です。*この関数は、小さなサンプルや、高度な乱数ジェネレーターを必要としないアプリケーションに適しています。並列および分散処理には適していません。より要求の厳しいアプリケーションの場合は、STREAMINIT サブルーチンと RANDRAND('Normal')関数を使用してください。

参照項目: “RANUNI 関数” (*SAS 関数と CALL ルーチン: リファレンス*)“RAND 関数” (*SAS 関数と CALL ルーチン: リファレンス*)“CALL STREAMINIT ルーチン” (*SAS 関数と CALL ルーチン: リファレンス*)

UPCASE 関数

引数内のすべての文字を大文字に変換します。

別名: UPPER

返されるデータの種類: CHAR、NCHAR、NVARCHAR、VARCHAR

構文

UPCASE(*expression*)

引数

expression
文字列に評価されるか強制変換できる有効な式を指定します。

データの種類 CHAR、NCHAR

詳細

UPCASE 関数は、文字式をコピーし、すべての小文字を大文字に変換して、変更された値を結果として返します。

比較

LOWCASE 関数は、引数内のすべての文字を小文字に変換します。UPCASE 関数は、引数内のすべての文字を大文字に変換します。

例

次のステートメントは、UPCASE 関数を示しています。

ステートメント	結果
name=upcase('John B. Smith');	JOHN B. SMITH

USS 関数

無修正平方和を返します。

返されるデータの種類: DOUBLE

構文

USS(*expression* <, ...*expression-n*>)

引数

expression
数値に評価される有効な式を指定します。

要件: 少なくとも 1 つの非 null または非欠損の引数が必要です。それがない場合は、関数から null または欠損値が返されます。

データの種類: DOUBLE

例

次のステートメントは、USS 関数を示しています。

ステートメント	結果
a=uss(4,2,3.5,6);	68.25
a=uss(4,2,3.5,6,.);	68.25

UUIDGEN 関数

ユニバーサル一意識別子(UUID)の短い形式を返します。

返されるデータの種類: CHAR、NCHAR、NVARCHAR、VARCHAR

構文

UUIDGEN( )

引数なし

UUIDGEN 関数には引数がありません。

詳細

UUIDGEN 関数は、各呼び出しに対して UUID (固有値)を返します。デフォルトの結果は、次のように 36 文字になります。

5ab6fa40–426b-4375–bb22–2d0291f43319

例

次の例では、UUID を返します。36 文字の変数宣言が必要であることに注意してください。

```
proc cas;
  x=uuidgen();
  print x;
run;
```

次の値が SAS ログに書き込まれます。各 UUID は一意です。

```
25C752D5-AFA1-4932-BEE6-39E4006C2AAB
```

VAR 関数

分散を返します。

返されるデータの種類: DOUBLE

構文

VAR(*expression-1*, *expression-2* < ,...*expression-n*>)

引数

expression
数値に評価される有効な式を指定します。引数リストは変数リストで構成することができます。

要件	少なくとも 2 つの非 null または非欠損の引数が必要です。それがない場合は、関数から null または欠損値が返されます。
データの種類	DOUBLE

例

次の例では、VAR 関数を使用して、分散を返します。

```
proc cas;
  val1=4;
  val2=2;
  val3=3.5;
  val4=6;
  x1=var(val1, val2, val3);
  x2=var(val1, val4);
  x3=var(x1, x2);
  print "x1=" x1;
  print "x2=" x2;
  print "x3=" x3;
run;
```

次のように SAS ログに出力されます。

```
x1=1.0833333333
x2=2
x3=0.4201388889
```

VERIFY 関数

式に固有の最初の文字の位置を返します。

返されるデータの種類:	DOUBLE

構文

VERIFY(*target-expression*, *search–expression*)

引数

target-expression
文字列に評価されるか強制変換できる検索先の有効な式を指定します。

要件 リテラル文字列は、単一引用符で囲む必要があります。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

search-expression
文字列に評価されるか強制変換できる有効な式を指定します。

要件 リテラル文字列は、単一引用符で囲む必要があります。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

詳細

VERIFY 関数は、*target-expression* における *search-expression* に存在しない最初の文字の位置を返します。*target-expression* に *search-expression* と異なる文字がない場合、VERIFY は 0 を返します。

比較

INDEX 関数は、*target-expression* に存在する *search-expression* の最初の出現位置を返し、VERIFY 関数は、*target-expression* における *search-expression* が含まれていない最初の文字の位置を返します。

例

次のステートメントは、VERIFY 関数を示しています。

ステートメント	結果

proc cas;
  x='abc';
  y='abcdef';
  z=verify(y,x);
  print "z=" z;
run;
	z=4

proc cas;
  x='abcdef';
  y='abcdef';
  z=verify(y,x);
  print "z=" z;
run;
	z=0

WEEK 関数

週番号の値を返します。

返されるデータの種類: DOUBLE

構文

WEEK(<*sas-date*>, <'*descriptor*'>)

引数

sas-date
SAS 日付値を指定します。*sas-date* 引数を指定しない場合、WEEK 関数は現在の日付の週番号の値を返します。

データの種類 DOUBLE

descriptor
ディスクリプターの値を指定します。次のディスクリプターを大文字または小文字で指定できます。

U
1 年のうちの週番号を指定します。日曜日が週の第 1 日とみなされます。週番号値は、0 から 53 の範囲の 10 進数として表されます。第 53 週に特別な意味はありません。week('31dec2013'd, 'u')の値は 53 です。U はデフォルト値です。

ヒント U および W ディスクリプターは似ていますが、U ディスクリプターが日曜日を週の第 1 日とみなすのに対し、W ディスクリプターは月曜日を週の第 1 日とみなします。

V
1 から 53 の範囲の 10 進数として表される週番号値を指定します。月曜日が週の第 1 日とみなされ、年の第 1 週は、1 月 4 日とその年の最初の木曜日の両方を含む週です。1 月の最初の月曜日が 2 日、3 日または 4 日の場合、それより前の日は前年の最後の週に組み込まれます。

W
1 年のうちの週番号を指定します。月曜日が週の第 1 日とみなされます。週番号値は、0 から 53 の範囲の 10 進数として表されます。第 53 週に特別な意味はありません。week('31dec2013'd, 'w')の値は 53 です。

ヒント U および W ディスクリプターは似ていますが、U ディスクリプターが日曜日を週の第 1 日とみなすのに対し、W ディスクリプターは月曜日を週の第 1 日とみなします。

デフォルト　U

データの種類　CHAR、NCHAR、NVARCHAR、VARCHAR

詳細

基本

WEEK 関数は、SAS 日付値を読み取り、週番号を返します。WEEK 関数はロケールに依存せず、計算にはグレゴリオ暦のみを使用します。

U 記述子

U ディスクリプターを使用する WEEK 関数は、SAS 日付値を読み取り、1 年のうちの週番号を返します。週番号値は、先頭に 0 を付けた 0 から 53 までの範囲の 10 進数として表され、最大値は 53 になります。第 0 週は、週の第 1 日が前年になるということを意味します。年の第 5 週は 05 と表されます。

日曜日が週の第 1 日とみなされます。たとえば、week('01jan2013'd, 'u')は 0 です。

V 記述子

V ディスクリプターを使用する WEEK 関数は、SAS 日付値を読み取り、週番号を返します。週番号値は、01 から 53 の範囲の 10 進数として表されます。10 進数の先頭に 0 を使用し、最大値は 53 です。各週は月曜日から始まり、年の第 1 週は 1 月 4 日とその年の最初の木曜日の両方を含む週です。1 月の最初の月曜日が 2 日、3 日または 4 日の場合、それより前の日は前年の最後の週に組み込まれます。次の例では、01jan2014 と 31dec2013 は同じ週になります。この週の第 1 日(月曜日)は 30dec2013 です。したがって、week('01jan2014'd, 'v')と week('30dec2013'd, 'v')の両方は 53 を返します。つまり、両方の日付が 2013 年の第 53 週に含まれるという意味です。

W 記述子

W ディスクリプターを使用する WEEK 関数は、SAS 日付値を読み取り、1 年のうちの週番号を返します。週番号値は、先頭に 0 を付けた 0 から 53 までの範囲の 10 進数として表され、最大値は 53 になります。第 0 週は、週の第 1 日が前年になるということを意味します。年の第 5 週は 05 と表されます。

月曜日が週の第 1 日とみなされます。したがって、week('30dec2013'd, 'w')の値は 1 です。

記述子の比較

U はデフォルトのディスクリプターです。範囲は 0 から 53 で、週の第 1 日は日曜日です。V ディスクリプターの範囲は 1 から 53 で、週の第 1 日は月曜日です。W ディスクリプターの範囲は 0 から 53 で、週の第 1 日は月曜日です。

ディスクリプターおよび関連付けられた週を次のリストに示します。

- 第 0 週:

U　現在のグレゴリオ暦年の第 1 週より前の日を示します。

V　該当しません。

W　現在のグレゴリオ暦年の第 1 週より前の日を示します。

- 第 1 週:

U　グレゴリオ暦年の最初の日曜日から始まります。

V　グレゴリオ暦による前年 12 月 29 日から当年 1 月 4 日までの間の月曜日から始まります。ISO 形式の最初の週は、グレゴリオ暦による前年と当年をまたぐことがあります。

W　グレゴリオ暦年の最初の月曜日から始まります。

- 年末の週:

U　1 年の最後の週(第 52 または 53)の日数を 7 日未満にできることを指定します。グレゴリオ暦による連続する 2 年をまたぐ日曜日から土曜日までの期間は、第 52 週と第 0 週、または第 53 週と第 0 週として指定されます。

V　ISO 形式の年の最後の週(第 52 または 53)の日数が 7 日であることを指定します。ただし、ISO 形式の年の最後の週は、グレゴリオ暦の当年と翌年をまたぐことがあります。

W　1 年の最後の週(第 52 または 53)の日数を 7 日未満にできることを指定します。グレゴリオ暦による連続する 2 年をまたぐ月曜日から日曜日までの期間は、第 52 と第 0 週、または第 53 週と第 0 週として指定されます。

例

次の例は、2013 年 5 月 12 日の日付の U、V、および W 記述子の値を示しています。

```
proc cas;
  sasdate=(19490);
  x=week(sasdate, 'u');
  y=week(sasdate, 'v');
  z=week(sasdate, 'w');
  print "x=" x;
  print "y=" y;
  print "z=" z;
run;
```

次の行が SAS ログに書き込まれます。

```
 x=19
 y=19
 z=18
```

WEEKDAY 関数

SAS 日付値から、曜日に対応する整数を返します。

返されるデータの種類: DOUBLE

構文

WEEKDAY(*expression*)

引数

expression
SAS 日付値を表す有効な式を指定します。

データの種類 DOUBLE

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)。

例

次のステートメントは、現在日が日曜日の場合の WEEKDAY 関数を示しています。

ステートメント	結果
x=weekday(today());	1

WHICHC 関数

文字列のリストから文字列の最初の位置を返します。

返されるデータの種類: DOUBLE

構文

WHICHC(*search-expression*, *expression-list-item-1*, *expression-list-item-2* <, ...*expression-list-item-n*>)

引数

search-expression
文字列式のリストと比較される文字列に評価されるか強制変換できる有効な式を指定します。

要件 リテラル文字列は、単一引用符で囲む必要があります。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

expression-list-item
文字列に評価されるか強制変換できて、文字列式のリストのメンバーである有効な式を指定します。

要件 リテラル文字列は、単一引用符で囲む必要があります。

リストには少なくとも 2 つの式が必要です。

データの種類 CHAR、NCHAR、NVARCHAR、VARCHAR

詳細

WHICHC 関数は、文字式リストを左から右に検索して、検索式に一致する最初の式を探します。一致するものが見つかった場合、WHICHC は式リスト内のその位置を返します。いずれの式も検索式に一致しない場合、WHICHC は値 0 を返します。

例

次の例では、'Spain'がリストに 2 回出現します。WHICHC 関数は、リスト内の最初の'Spain'の位置を返します。

ステートメント	結果
x=whichc('Spain', 'Denmark', 'Germany', 'Austria', 'Spain', 'China', 'Egypt', 'Spain', 'France');	4

WHICHN 関数

数値のリストから数値の最初の位置を返します。

返されるデータの種類: DOUBLE

構文

WHICHN(*search-expression*, *expression-list-item-1*, *expression-list-item-2* <, ...*expression-list-item-n*>)

引数

search-expression
数値に評価され、数値式のリストと比較される有効な式を指定します。

データの種類 DOUBLE

expression-list-item
数値に評価され、リストの一部である有効な式を指定します。

要件 リストには少なくとも 2 つの式が必要です。

データの種類 DOUBLE

詳細

WHICHN 関数は、数値式リストを左から右に検索して、検索式に一致する最初の式を探します。一致するものが見つかった場合、WHICHN は式リスト内のその位置を返します。いずれの式も検索式に一致しない場合、WHICHN は値 0 を返します。WHICHN 関数の引数には、任意の数値データ型を指定できます。

例

次の例では、4.5 がリストに 2 回出現します。WHICHN 関数は、リスト内の最初の 4.5 の位置を返します。

ステートメント	結果
x=whichn(4.5,7.3, 8.6, 4.5, 4.5, 2.1, 6.4);	3

YEAR 関数

SAS 日付値から年を返します。

返されるデータの種類: DOUBLE

構文

YEAR(*date*)

引数

date
SAS 日付値を表す有効な式を指定します。

データの種類 DOUBLE

詳細

YEAR 関数は、年を表す 4 桁の数値を生成します。

例

次のステートメントは、年が 2007 の場合の YEAR 関数を示しています。

ステートメント	結果
date=today(); y=year(date);	2007

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。

返されるデータの種類: DOUBLE

構文

YIELDP(*A*, *c*, *n*, *K*, k_0, *p*)

引数

A

額面価格を指定します。

範囲 $A > 0$

A > 0

データの種類 DOUBLE

c

分数で表す、年間の名目クーポン率を指定します。

範囲 $0 \leq c < 1$

0 ≤ *c* < 1

データの種類 DOUBLE

n

1 年当たりのクーポン数を指定します。

範囲 $n > 0$

n > 0

データの種類 DOUBLE

K

決済日から満期までの残りのクーポン数を指定します。

範囲 $K > 0$

K > 0

データの種類 DOUBLE

k_0

決算日から次のクーポンまでの時間を、1 年ごとの分数で指定します。

範囲 $0 < k_0 \leq \frac{1}{n}$

$0 < k_0 \leq 1/n$

データの種類 DOUBLE

p

未払い利息込みの価格を指定します。

範囲	$p > 0$
	$p > 0$
データの種類	DOUBLE

詳細

YIELDP 関数は次の関係に基づきます。

$$P = \sum_{k=1}^{K} c(k) \frac{1}{\left(1 + \frac{y}{n}\right)^{t_k}}$$

前述の式には次の関係が適用されます。

- $t_k = nk_0 + k - 1$
- $c(k) = \frac{c}{n}A \quad for\ k = 1, \ldots, K-1$
- $c(K) = \left(1 + \frac{c}{n}\right)A$

YIELDP 関数は、y に対して解決します。

例

次の例では、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。返される値は 0.0775031248 です。

```
proc cas;
  y=yieldp(1000, 0.01, 4, 14, .165, 800);
  print "y=" y;
run;
```

SAS は次の出力をログに書き込みます。

```
y=0.0775031253
```

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。

返されるデータの種類:	DOUBLE

構文

YRDIF(*start-date*, *end-date*<, *basis*>)

引数

start-date
開始日を識別する SAS 日付値を指定します。

データの種類 DOUBLE

end-date
終了日を識別する SAS 日付値を指定します。

データの種類 DOUBLE

basis
SAS で日付の差(人の年齢)が計算される方法を記述する文字定数、変数または式を指定します。有効な文字列は、'30/360' (30 日の月と 360 日の年)、'ACT/ACT'、'ACT/360',、'ACT/365'、'AGE'です。たとえば、'ACT/360'では、年数の計算に日付間の実際の日数を使用します。この値は、各年の実際の日数に関係なく、日数を 360 で除算して計算されます。詳細については、SAS DS2 言語リファレンスを参照してください。

SAS で日付の差(人の年齢)が計算される方法を記述する文字定数、変数または式を指定します。有効な文字列は次のとおりです。

'30/360'
年数の計算で 30 日の月と 360 日の年を指定します。各月または年の実際の日数に関係なく、各月は 30 日、各年は 360 日とみなされます。

別名 '360'

ヒント いずれかの日付が月末になる場合、その日は 30 日の月の最終日として扱われます。

'ACT/ACT'
年数の計算には、日付間の実際の日数を使用します。この値は、365 日の年の日数を 365 で除算して得た値と、366 日の年の日数を 366 で除算して得た値を足して計算されます。

別名 'Actual'

'ACT/360'
年数の計算には、日付間の実際の日数を使用します。この値は、各年の実際の日数に関係なく、日数を 360 で除算して計算されます。

'ACT/365'
年数の計算には、日付間の実際の日数を使用します。この値は、各年の実際の日数に関係なく、日数を 365 で除算して計算されます。

'AGE'
人の年齢が計算されることを指定します。

第 3 引数を指定しない場合、AGE が *basis* のデフォルト値になります。

データの種類　CHAR、NCHAR、NVARCHAR、VARCHAR

詳細

会計アプリケーションでの YRDIF の使用

基本

YRDIF 関数は、第 3 引数 *basis* が存在する場合に、確定利付証券の利息を計算するために使用できます。指定した日数計算規則に従って 2 つの日付の差を返します。

ACT/ACT 基準を使用する計算

ACT/ACT 基準を使用する YRDIF 計算では、365 日の年と 366 日の年の両方が考慮されます。たとえば、n365 が 365 日の年の開始日から終了日までの日数と等しく、n366 が 366 日の年の開始日から終了日までの日数と等しい場合、YRDIF 計算は YRDIF=n365/365.0 + n366/366.0 として計算されます。この計算は、金融関係の文献に記されている一般に理解されている *ACT/ACT* 日数計算基準に対応します。*basis* の値には、他に *30/360*、*ACT/360* および *ACT/365* があります。各基準には、特定の金融商品の利息の支払いを計算するために従う必要がある明確な意味があります。

人の年齢の計算

YRDIF 関数で人の年齢を計算できます。最初の 2 つの引数 *start-date* と *end-date* は必須です。*basis* の値が AGE の場合、YRDIF は年齢を計算します。年齢の計算では、うるう年が考慮されます。*basis* のその他の値は、人の年齢の計算には無効です。

例

例 1: basis に基づいた年の差の計算

次の例では、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。

```
proc cas;
  sdate=(19490); /*12MAY2013*/
  edate=(19990); /*24SEP2014*/
  y30360=yrdif(sdate, edate, '30/360');
  yactact=yrdif(sdate, edate, 'ACT/ACT');
  yact360=yrdif(sdate, edate, 'ACT/360');
  yact365=yrdif(sdate, edate, 'ACT/365');
  print "y30360=" y30360;
  print "yactact=" yactact;
  print "yact360=" yact360;
  print "yact365=" yact365;
run;
```

SAS は次の結果をログに書き込みます。

```
y30360=1.3666666667
yactact=1.3698630137
yact360=1.3888888889
yact365=1.3698630137
```

例 2: 人の年齢の計算

YRDID 関数で 3 つの引数を使用して人の年齢を計算できます。第 3 引数 *basis* の値は AGE にする必要があります。

```
proc cas;
  sdate=(7572); /*24SEP1980*/
  edate=(21451); /*24SEP2018*/
  age=yrdif(sdate, edate, 'AGE');
  print "Age=" age 'years';
run;
```

SAS は次の結果をログに書き込みます。

```
Age=38 years
```

参考文献

"Day count convention." *Wikipedia*, 2010. URI: Day count convention. Accessed on May 1, 2013..

ISDA International Swaps and Derivatives Association, Inc "EMU and Market Conventions: Recent Developments." 1998. ISDA. URI: EMU and Market Conventions: Recent Developments.

Mayle, Jan. 1994. *Standard Securities Calculation Methods – Fixed Income Securities Formulas for Analytic Measures*. 2 巻 New York, New York: Securities Industry Association.

YYQ 関数

年と四半期の値から SAS 日付値を返します。

返されるデータの種類: DOUBLE

構文

YYQ(*year*, *quarter*)

引数

year
年を表す 2 桁または 4 桁の整数に評価される有効な式を指定します。

操作 YEARCUTOFF=システムオプションは、2 桁の日付の年の値を定義します。

データの種類 DOUBLE

quarter
年の四半期(1、2、3、4)を指定します。

データの種類 DOUBLE

詳細

YYQ 関数は、指定した四半期の最初の日に対応する SAS 日付値を返します。*year* または *quarter* のいずれかが null または欠損している場合、または四半期の値が無効である場合、結果は null 値または欠損値になります。

例

次のステートメントは、YYQ 関数を示しています。

ステートメント	結果

proc cas;
DateValue=yyq(2016,3);
Date7Value=put(DateValue, date7.);
Date9Value=put(DateValue, date9.);
print "DateValue=" DateValue;
print "Date7Value=" Date7Value;
print "Date9Value=" Date9Value;
run;
	DateValue=20636
	Date7Value=01JUL16
	Date9Value=01JUL2016

proc cas;
StartOfQuarter=yyq(2016,4);
StartofQuarter9=put(StartOfQuarter, date9.);
print "StartofQuarter=" StartofQuarter;
print "StartofQuarter9=" StartofQuarter9;
run;
	StartofQuarter=20728
	StartofQuarter9=01OCT2016