CHAPTER

4

Functions and CALL Routines

Definitions

Definition of Functions

A *SAS function* performs a computation or system manipulation on arguments and returns a value. Most functions use arguments supplied by the user, but a few obtain their arguments from the operating environment.

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

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

Definition of CALL Routines

A *CALL routine* alters variable values or performs other system functions. CALL routines are similar to functions, but differ from functions in that you cannot use them in assignment statements.

All SAS CALL routines are invoked with CALL statements; that is, the name of the routine must appear after the keyword CALL on the CALL statement.

Syntax

Syntax of Functions

The syntax of a function is

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

function-name (OF *variable-list*)

function-name (OF *array-name*{*})

where

function-name
names the function.

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

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

Examples:

- `x=max(cash,credit);`

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

(OF *variable-list*)
can be any form of a SAS variable list, including individual variable names. If more than one variable list appears, separate them with a space.

Examples:

- `a=sum(of x  y  z);`
- The following two examples are equivalent.
  -
```
a=sum(of x1-x10 y1-y10 z1-z10);
a=sum(of x1-x10, of y1-y10, of z1-z10);
```
- `z=sum(of y1-y10);`

(OF *array-name*{*})
names a currently defined array. Specifying an array in this way causes SAS to treat the array as a list of the variables instead of processing only one element of the array at a time.

Examples:

-
```
array y{10} y1-y10;
x=sum(of y{*});
```

Syntax of CALL Routines

The syntax of a CALL routine is

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

where

routine-name
names a SAS CALL routine.

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

Examples:

- `call rxsubstr(rx,string,position);`
- `call set(dsid);`

- `call ranbin(Seed_1,n,p,X1);`
- `call label(abc{j},lab);`

Using Functions

Restrictions on Function Arguments

If the value of an argument is invalid, SAS prints an error message and sets the result to a missing value. Here are some common restrictions on function arguments:

- Some functions require that their arguments be restricted within a certain range. For example, the argument of the LOG function must be greater than 0.
- Most functions do not permit missing values as arguments. Exceptions include some of the descriptive statistics functions and financial functions.
- In general, the allowed range of the arguments is platform-dependent, such as with the EXP function.
- For some probability functions, combinations of extreme values can cause convergence problems.

Characteristics of Target Variables

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

Table 4.1 Target Variables

Function	Target Variable Type	Target Variable Length (bytes)
BYTE	character	1
COMPRESS	character	length of first argument
INPUT	character	width of informat
	numeric	8
LEFT	character	length of argument
PUT	character	width of format
REVERSE	character	length of argument
RIGHT	character	length of argument
SUBSTR	character	length of first argument
TRANSLATE	character	length of first argument
TRIM	character	length of argument
UPCASE, LOWCASE	character	length of argument
VTYPE, VTYPEX	character	1

Notes on Descriptive Statistic Functions

SAS provides functions that return descriptive statistics. Except for the MISSING function, the functions correspond to the statistics produced by the MEANS procedure. The computing method for each statistic is discussed in "SAS Elementary Statistics Procedures" in of the *SAS Procedures Guide*. SAS calculates descriptive statistics for the nonmissing values of the arguments.

Notes on Financial Functions

SAS provides a group of functions that perform financial calculations. The functions are grouped into the following types:

Table 4.2 Types of Financial Functions

Function type	Functions	Description
Cashflow	CONVX, CONVXP	calculates convexity for cashflows
	DUR, DURP	calculates modified duration for cashflows
	PVP, YIELDP	calculates present value and yield-to-maturity for a periodic cashflow
Parameter calculations	COMPOUND	calculates compound interest parameters
	MORT	calculates amortization parameters
Internal rate of return	INTRR, IRR	calculates the internal rate of return
Net present and future value	NETPV, NPV	calculates net present and future values
	SAVING	calculates the future value of periodic saving
Depreciation	DACCxx	calculates the accumulated depreciation up to the specified period
	DEPxxx	calculates depreciation for a single period

Special Considerations for Depreciation Functions

The period argument for depreciation functions can be fractional for all of the functions except DEPDBSL and DACCDBSL. For fractional arguments, the depreciation is prorated between the two consecutive time periods preceding and following the fractional period.

CAUTION:

Verify the depreciation method for fractional periods. You must verify whether this method is appropriate to use with fractional periods because many depreciation schedules, specified as tables, have special rules for fractional periods. △

Using DATA Step Functions within Macro Functions

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

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

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

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

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

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

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

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

You can use these functions to call all of the DATA step SAS functions except those that pertain to DATA step variables or processing. These prohibited functions are: DIF, DIM, HBOUND, INPUT, IORCMSG, LAG, LBOUND, MISSING, PUT, RESOLVE, SYMGET, and all of the variable information functions (for example, VLABEL).

Using Functions to Manipulate Files

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

When you use external files, the FOPEN function allocates a buffer called the File Data Buffer (FDB) and opens the external file for reading or updating. The FREAD function reads a record from the external file and copies the data into the FDB. The FGET function then moves the data to the DATA step variables. The function returns a value that you can check with statements or other functions in the DATA step to determine how to further process your data. After the records are processed, the FWRITE function writes the contents of the FDB to the external file, and the FCLOSE function closes the file.

When you use SAS data sets, the OPEN function opens the data set. The FETCH and FETCHOBS functions read observations from an open SAS data set into the Data Set Data Vector (DDV). The GETVARC and GETVARN functions then move the data to DATA step variables. The functions return a value that you can check with statements or other functions in the DATA step to determine how you want to further process your data. After the data is processed, the CLOSE function closes the data set.

For a complete listing of functions and CALL routines, see "Functions and CALL Routines by Category" on page 213 . For complete descriptions and examples, see the dictionary section of this book.

Using Random-Number Functions and CALL Routines

Seed Values

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

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

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

Comparison of Random-Number Functions and CALL Routines

Except for the NORMAL and UNIFORM functions, which are equivalent to the RANNOR and RANUNI functions, respectively, SAS provides a CALL routine that has the same name as each random-number function. Using CALL routines gives you greater control over the seed values. As an illustration of this control over seed values, all the random-number CALL routines show an example of interrupting the stream of random numbers to change the seed value.

With a CALL routine, you can generate multiple streams of random numbers within a single DATA step. If you supply a different seed value to initialize each of the seed variables, the streams of the generated random numbers are computationally independent. With a function, however, you cannot generate more than one stream by supplying multiple seeds within a DATA step. The following two examples illustrate the difference.

Examples

Example 1: Generating Multiple Streams from a CALL Routine

This example uses the CALL RANUNI routine to generate three streams of random numbers from the uniform distribution with ten numbers each. See the results in Output 4.1 on page 212.

```
options nodate pageno=1 linesize=80 pagesize=60;

data multiple(drop=i);
   retain Seed_1 1298573062 Seed_2 447801538
          Seed_3 631280;
   do i=1 to 10;
      call ranuni (Seed_1,X1);
      call ranuni (Seed_2,X2);
      call ranuni (Seed_3,X3);
      output;
   end;
run;

proc print data=multiple;
   title 'Multiple Streams from a CALL Routine';
run;
```

Output 4.1 The CALL Routine Example

```
                     Multiple Streams from a CALL Routine                    1

Obs      Seed_1        Seed_2        Seed_3         X1         X2         X3

  1    1394231558     512727191     367385659    0.64924    0.23876    0.17108
  2    1921384255    1857602268    1297973981    0.89471    0.86501    0.60442
  3     902955627     422181009     188867073    0.42047    0.19659    0.08795
  4     440711467     761747298     379789529    0.20522    0.35472    0.17685
  5    1044485023    1703172173     591320717    0.48638    0.79310    0.27536
  6    2136205611    2077746915     870485645    0.99475    0.96753    0.40535
  7    1028417321    1800207034    1916469763    0.47889    0.83829    0.89243
  8    1163276804     473335603     753297438    0.54169    0.22041    0.35078
  9     176629027    1114889939    2089210809    0.08225    0.51916    0.97286
 10    1587189112     399894790     284959446    0.73909    0.18622    0.13269
```

Example 2: Assigning Values from a Single Stream to Multiple Variables

Using the same three seeds that were used in Example 1, this example uses a function to create three variables. The results that are produced are different from those in Example 1 because the values of all three variables are generated by the first seed. When you use an individual function more than once in a DATA step, the function accepts only the first seed value that you supply and ignores the rest.

```
options nodate pageno=1 linesize=80 pagesize=60;

data single(drop=i);
   do i=1 to 3;
      Y1=ranuni(1298573062);
      Y2=ranuni(447801538);
      Y3=ranuni(631280);
      output;
   end;
run;

proc print data=single;
   title 'A Single Stream across Multiple Variables';
run;
```

Output 4.2 on page 213 shows the results. The values of Y1, Y2, and Y3 in this example come from the same random-number stream generated from the first seed. You can see this by comparing the values by observation across these three variables with the values of X1 in Output 4.1 on page 212.

Output 4.2 The Function Example

```
                    A Single Stream across Multiple Variables                   1

                       Obs       Y1          Y2          Y3

                        1     0.64924     0.89471     0.42047
                        2     0.20522     0.48638     0.99475
                        3     0.47889     0.54169     0.08225
```

Pattern Matching Using Regular Expression (RX) Functions and CALL Routines

You can use a special group of functions and CALL routines to match or change data according to a specific pattern that you specify. By using these functions and CALL routines, you can determine whether a given character string is in a set denoted by a pattern, or you can search a given character string for a substring in a set denoted by a pattern. You can also change a matched substring to a different substring.

This group consists of CALL RXCHANGE, CALL RXFREE, CALL RXSUBSTR, RXMATCH, and RXPARSE, and comprises the character string matching category for functions and CALL routines. For a description of these functions, see "Functions and CALL Routines by Category" on page 213.

Base SAS Functions for Web Applications

Four functions that manipulate Web-related content are available in base SAS software. HTMLENCODE and URLENCODE return encoded strings. HTMLDECODE and URLDECODE return decoded strings. For information about Web-based SAS tools, follow the Web Enablement link on the SAS Institute home page, at www.sas.com.

Functions and CALL Routines by Category

Table 4.3 Categories and Descriptions of Functions and CALL Routines

Category	Functions and CALL Routine	Description
Array	"DIM" on page 330	Returns the number of elements in an array
	"HBOUND" on page 392	Returns the upper bound of an array
	"LBOUND" on page 434	Returns the lower bound of an array

Bitwise Logical Operations	"BAND" on page 236	Returns the bitwise logical AND of two arguments
	"BLSHIFT" on page 238	Returns the bitwise logical left shift of two arguments
	"BNOT" on page 238	Returns the bitwise logical NOT of an argument
	"BOR" on page 239	Returns the bitwise logical OR of two arguments
	"BRSHIFT" on page 239	Returns the bitwise logical right shift of two arguments
	"BXOR" on page 240	Returns the bitwise logical EXCLUSIVE OR of two arguments
Character String Matching	"CALL RXCHANGE" on page 265	Changes one or more substrings that match a pattern
	"CALL RXFREE" on page 267	Frees memory allocated by other regular expression (RX) functions and CALL routines
	"CALL RXSUBSTR" on page 268	Finds the position, length, and score of a substring that matches a pattern
	"RXMATCH" on page 526	Finds the beginning of a substring that matches a pattern and returns a value
	"RXPARSE" on page 527	Parses a pattern and returns a value
Character	"BYTE" on page 241	Returns one character in the ASCII or the EBCDIC collating sequence
	"COLLATE" on page 291	Returns an ASCII or EBCDIC collating sequence character string
	"COMPBL" on page 294	Removes multiple blanks from a character string
	"COMPRESS" on page 296	Removes specific characters from a character string
	"DEQUOTE" on page 323	Removes quotation marks from a character value
	"INDEX" on page 399	Searches a character expression for a string of characters
	"INDEXC" on page 400	Searches a character expression for specific characters
	"INDEXW" on page 401	Searches a character expression for a specified string as a word
	"LEFT" on page 435	Left aligns a SAS character expression
	"LENGTH" on page 436	Returns the length of an argument
	"LOWCASE" on page 441	Converts all letters in an argument to lowercase
	"MISSING" on page 446	Returns a numeric result that indicates whether the argument contains a missing value
	"QUOTE" on page 510	Adds double quotation marks to a character value
	"RANK" on page 516	Returns the position of a character in the ASCII or EBCDIC collating sequence
	"REPEAT" on page 522	Repeats a character expression
	"REVERSE" on page 523	Reverses a character expression
	"RIGHT" on page 524	Right aligns a character expression
	"SCAN" on page 541	Selects a given word from a character expression
	"SOUNDEX" on page 546	Encodes a string to facilitate searching

	"SPEDIS" on page 547	Determines the likelihood of two words matching, expressed as the asymmetric spelling distance between the two words
	"SUBSTR (left of =)" on page 554	Replaces character value contents
	"SUBSTR (right of =)" on page 555	Extracts a substring from an argument
	"TRANSLATE" on page 567	Replaces specific characters in a character expression
	"TRANWRD" on page 568	Replaces or removes all occurrences of a word in a character string
	"TRIM" on page 570	Removes trailing blanks from character expressions and returns one blank if the expression is missing
	"TRIMN" on page 572	Removes trailing blanks from character expressions and returns a null string (zero blanks) if the expression is missing
	"UPCASE" on page 574	Converts all letters in an argument to uppercase
	"VERIFY" on page 589	Returns the position of the first character that is unique to an expression
DBCS	"KCOMPARE" on page 418	Returns the result of a comparison of character strings
	"KCOMPRESS" on page 418	Removes specific characters from a character string
	"KCOUNT" on page 419	Returns the number of double-byte characters in a string
	"KINDEX" on page 419	Searches a character expression for a string of characters
	"KINDEXC" on page 420	Searches a character expression for specific characters
	"KLEFT" on page 421	Left aligns a SAS character expression by removing unnecessary leading DBCS blanks and SO/SI
	"KLENGTH" on page 421	Returns the length of an argument
	"KLOWCASE" on page 422	Converts all letters in an argument to lowercase
	"KREVERSE" on page 422	Reverses a character expression
	"KRIGHT" on page 423	Right aligns a character expression by trimming trailing DBCS blanks and SO/SI
	"KSCAN" on page 423	Selects a given word from a character expression
	"KSTRCAT" on page 424	Concatenates two or more character strings
	"KSUBSTR" on page 425	Extracts a substring from an argument
	"KSUBSTRB" on page 425	Extracts a substring from an argument based on byte position
	"KTRANSLATE" on page 426	Replaces specific characters in a character expression
	"KTRIM" on page 427	Removes trailing DBCS blanks and SO/SI from character expressions
	"KTRUNCATE" on page 428	Truncates a numeric value to a specified length

	"KUPCASE" on page 428	Converts all single-byte letters in an argument to uppercase
	"KUPDATE" on page 429	Inserts, deletes, and replaces character value contents
	"KUPDATEB" on page 430	Inserts, deletes, and replaces character value contents based on byte unit
	"KVERIFY" on page 431	Returns the position of the first character that is unique to an expression
Date and Time	"DATDIF" on page 311	Returns the number of days between two dates
	"DATE" on page 312	Returns the current date as a SAS date value
	"DATEJUL" on page 313	Converts a Julian date to a SAS date value
	"DATEPART" on page 314	Extracts the date from a SAS datetime value
	"DATETIME" on page 315	Returns the current date and time of day as a SAS datetime value
	"DAY" on page 315	Returns the day of the month from a SAS date value
	"DHMS" on page 327	Returns a SAS datetime value from date, hour, minute, and second
	"HMS" on page 394	Returns a SAS time value from hour, minute, and second values
	"HOUR" on page 395	Returns the hour from a SAS time or datetime value
	"INTCK" on page 408	Returns the integer number of time intervals in a given time span
	"INTNX" on page 410	Advances a date, time, or datetime value by a given interval, and returns a date, time, or datetime value
	"JULDATE" on page 416	Returns the Julian date from a SAS date value
	"JULDATE7" on page 417	Returns a seven-digit Julian date from a SAS date value
	"MDY" on page 443	Returns a SAS date value from month, day, and year values
	"MINUTE" on page 445	Returns the minute from a SAS time or datetime value
	"MONTH" on page 451	Returns the month from a SAS date value
	"QTR" on page 509	Returns the quarter of the year from a SAS date value
	"SECOND" on page 542	Returns the second from a SAS time or datetime value
	"TIME" on page 563	Returns the current time of day
	"TIMEPART" on page 564	Extracts a time value from a SAS datetime value
	"TODAY" on page 566	Returns the current date as a SAS date value
	"WEEKDAY" on page 617	Returns the day of the week from a SAS date value
	"YEAR" on page 618	Returns the year from a SAS date value
	"YRDIF" on page 620	Returns the difference in years between two dates
	"YYQ" on page 621	Returns a SAS date value from the year and quarter
Descriptive Statistics	"CSS" on page 304	Returns the corrected sum of squares
	"CV" on page 305	Returns the coefficient of variation
	"KURTOSIS" on page 431	Returns the kurtosis

	"MAX" on page 442	Returns the largest value
	"MEAN" on page 444	Returns the arithmetic mean (average)
	"MIN" on page 444	Returns the smallest value
	"MISSING" on page 446	Returns a numeric result that indicates whether the argument contains a missing value
	"N" on page 455	Returns the number of nonmissing values
	"NMISS" on page 457	Returns the number of missing values
	"ORDINAL" on page 462	Returns any specified order statistic
	"RANGE" on page 515	Returns the range of values
	"SKEWNESS" on page 545	Returns the skewness
	"STD" on page 549	Returns the standard deviation
	"STDERR" on page 550	Returns the standard error of the mean
	"SUM" on page 556	Returns the sum of the nonmissing arguments
	"USS" on page 577	Returns the uncorrected sum of squares
	"VAR" on page 578	Returns the variance
External Files	"DCLOSE" on page 316	Closes a directory that was opened by the DOPEN function and returns a value
	"DINFO" on page 331	Returns information about a directory
	"DNUM" on page 333	Returns the number of members in a directory
	"DOPEN" on page 334	Opens a directory and returns a directory identifier value
	"DOPTNAME" on page 335	Returns directory attribute information
	"DOPTNUM" on page 336	Returns the number of information items that are available for a directory
	"DREAD" on page 337	Returns the name of a directory member
	"DROPNOTE" on page 338	Deletes a note marker from a SAS data set or an external file and returns a value
	"FAPPEND" on page 347	Appends the current record to the end of an external file and returns a value
	"FCLOSE" on page 348	Closes an external file, directory, or directory member, and returns a value
	"FCOL" on page 349	Returns the current column position in the File Data Buffer (FDB)
	"FDELETE" on page 350	Deletes an external file or an empty directory
	"FEXIST" on page 354	Verifies the existence of an external file associated with a fileref and returns a value
	"FGET" on page 355	Copies data from the File Data Buffer (FDB) into a variable and returns a value
	"FILEEXIST" on page 357	Verifies the existence of an external file by its physical name and returns a value
	"FILENAME" on page 358	Assigns or deassigns a fileref for an external file, directory, or output device and returns a value

	"FILEREF" on page 360	Verifies that a fileref has been assigned for the current SAS session and returns a value
	"FINFO" on page 361	Returns the value of a file information item
	"FNOTE" on page 369	Identifies the last record that was read and returns a value that FPOINT can use
	"FOPEN" on page 371	Opens an external file and returns a file identifier value
	"FOPTNAME" on page 373	Returns the name of an item of information about a file
	"FOPTNUM" on page 374	Returns the number of information items that are available for an external file
	"FPOINT" on page 375	Positions the read pointer on the next record to be read and returns a value
	"FPOS" on page 377	Sets the position of the column pointer in the File Data Buffer (FDB) and returns a value
	"FPUT" on page 378	Moves data to the File Data Buffer (FDB) of an external file, starting at the FDB's current column position, and returns a value
	"FREAD" on page 379	Reads a record from an external file into the File Data Buffer (FDB) and returns a value
	"FREWIND" on page 380	Positions the file pointer to the start of the file and returns a value
	"FRLEN" on page 382	Returns the size of the last record read, or, if the file is opened for output, returns the current record size
	"FSEP" on page 383	Sets the token delimiters for the FGET function and returns a value
	"FWRITE" on page 385	Writes a record to an external file and returns a value
	"MOPEN" on page 452	Opens a file by directory id and member name, and returns the file identifier or a 0
	"PATHNAME" on page 463	Returns the physical name of a SAS data library or of an external file, or returns a blank
	"SYSMSG" on page 558	Returns the text of error messages or warning messages from the last data set or external file function execution
	"SYSRC" on page 561	Returns a system error number
External Routines	"CALL MODULE" on page 244	Calls the external routine without any return code
	"CALL MODULEI" on page 246	Calls the external routine without any return code (in IML environment only)
	"MODULEC" on page 448	Calls an external routine and returns a character value
	"MODULEIC" on page 449	Calls an external routine and returns a character value (in IML environment only)
	"MODULEIN" on page 449	Calls an external routine and returns a numeric value (in IML environment only)
	"MODULEN" on page 450	Calls an external routine and returns a numeric value
Financial	"COMPOUND" on page 295	Returns compound interest parameters

	"CONVX" on page 300	Returns the convexity for an enumerated cashflow
	"CONVXP" on page 301	Returns the convexity for a periodic cashflow stream, such as a bond
	"DACCDB" on page 306	Returns the accumulated declining balance depreciation
	"DACCDBSL" on page 307	Returns the accumulated declining balance with conversion to a straight-line depreciation
	"DACCSL" on page 308	Returns the accumulated straight-line depreciation
	"DACCSYD" on page 309	Returns the accumulated sum-of-years-digits depreciation
	"DACCTAB" on page 310	Returns the accumulated depreciation from specified tables
	"DEPDB" on page 318	Returns the declining balance depreciation
	"DEPDBSL" on page 319	Returns the declining balance with conversion to a straight-line depreciation
	"DEPSL" on page 320	Returns the straight-line depreciation
	"DEPSYD" on page 321	Returns the sum-of-years-digits depreciation
	"DEPTAB" on page 322	Returns the depreciation from specified tables
	"DUR" on page 340	Returns the modified duration for an enumerated cashflow
	"DURP" on page 341	Returns the modified duration for a periodic cashflow stream, such as a bond
	"INTRR" on page 412	Returns the internal rate of return as a fraction
	"IRR" on page 415	Returns the internal rate of return as a percentage
	"MORT" on page 454	Returns amortization parameters
	"NETPV" on page 456	Returns the net present value as a fraction
	"NPV" on page 460	Returns the net present value with the rate expressed as a percentage
	"PVP" on page 508	Returns the present value for a periodic cashflow stream, such as a bond
	"SAVING" on page 540	Returns the future value of a periodic saving
	"YIELDP" on page 618	Returns the yield-to-maturity for a periodic cashflow stream, such as a bond
Hyperbolic	"COSH" on page 303	Returns the hyperbolic cosine
	"SINH" on page 544	Returns the hyperbolic sine
	"TANH" on page 563	Returns the hyperbolic tangent
Macro	"CALL EXECUTE" on page 242	Resolves an argument and issues the resolved value for execution
	"CALL SYMPUT" on page 271	Assigns DATA step information to a macro variable
	"RESOLVE" on page 522	Returns the resolved value of an argument after it has been processed by the macro facility

	"SYMGET" on page 557	Returns the value of a macro variable during DATA step execution
Mathematical	"ABS" on page 226	Returns the absolute value
	"AIRY" on page 227	Returns the value of the airy function
	"CNONCT" on page 290	Returns the noncentrality parameter from a chi-squared distribution
	"COMB" on page 293	Computes the number of combinations of *n* elements taken *r* at a time and returns a value
	"CONSTANT" on page 297	Computes some machine and mathematical constants and returns a value
	"DAIRY" on page 311	Returns the derivative of the airy function
	"DEVIANCE" on page 323	Computes the deviance and returns a value
	"DIGAMMA" on page 329	Returns the value of the DIGAMMA function
	"ERF" on page 342	Returns the value of the (normal) error function
	"ERFC" on page 343	Returns the value of the complementary (normal) error function
	"EXP" on page 345	Returns the value of the exponential function
	"FACT" on page 346	Computes a factorial and returns a value
	"FNONCT" on page 368	Returns the value of the noncentrality parameter of an F distribution
	"GAMMA" on page 387	Returns the value of the Gamma function
	"IBESSEL" on page 398	Returns the value of the modified bessel function
	"JBESSEL" on page 415	Returns the value of the bessel function
	"LGAMMA" on page 437	Returns the natural logarithm of the Gamma function
	"LOG" on page 439	Returns the natural (base e) logarithm
	"LOG10" on page 440	Returns the logarithm to the base 10
	"LOG2" on page 440	Returns the logarithm to the base 2
	"MOD" on page 447	Returns the remainder value
	"PERM" on page 480	Computes the number of permutations of *n* items taken *r* at a time and returns a value
	"SIGN" on page 543	Returns the sign of a value
	"SQRT" on page 549	Returns the square root of a value
	"TNONCT" on page 565	Returns the value of the noncentrality parameter from the student's *t* distribution
	"TRIGAMMA" on page 570	Returns the value of the TRIGAMMA function
Probability	"CDF" on page 273	Computes cumulative distribution functions
	"LOGPDF" on page 441	Computes the logarithm of a probability (mass) function
	"LOGSDF" on page 441	Computes the logarithm of a survival function
	"PDF" on page 464	Computes probability density (mass) functions
	"POISSON" on page 482	Returns the probability from a Poisson distribution

	"PROBBETA" on page 484	Returns the probability from a beta distribution
	"PROBBNML" on page 484	Returns the probability from a binomial distribution
	"PROBBNRM" on page 485	Computes a probability from the bivariate normal distribution and returns a value
	"PROBCHI" on page 486	Returns the probability from a chi-squared distribution
	"PROBF" on page 487	Returns the probability from an *F* distribution
	"PROBGAM" on page 488	Returns the probability from a gamma distribution
	"PROBHYPR" on page 489	Returns the probability from a hypergeometric distribution
	"PROBMC" on page 490	Computes a probability or a quantile from various distributions for multiple comparisons of means, and returns a value
	"PROBNEGB" on page 501	Returns the probability from a negative binomial distribution
	"PROBNORM" on page 502	Returns the probability from the standard normal distribution
	"PROBT" on page 502	Returns the probability from a *t* distribution
	"SDF" on page 542	Computes a survival function
Quantile	"BETAINV" on page 237	Returns a quantile from the beta distribution
	"CINV" on page 288	Returns a quantile from the chi-squared distribution
	"FINV" on page 362	Returns a quantile from the *F* distribution
	"GAMINV" on page 386	Returns a quantile from the gamma distribution
	"PROBIT" on page 490	Returns a quantile from the standard normal distribution
	"TINV" on page 564	Returns a quantile from the *t* distribution
Random Number	"CALL RANBIN" on page 248	Returns a random variate from a binomial distribution
	"CALL RANCAU" on page 250	Returns a random variate from a Cauchy distribution
	"CALL RANEXP" on page 252	Returns a random variate from an exponential distribution
	"CALL RANGAM" on page 254	Returns a random variate from a gamma distribution
	"CALL RANNOR" on page 256	Returns a random variate from a normal distribution
	"CALL RANPOI" on page 257	Returns a random variate from a Poisson distribution
	"CALL RANTBL" on page 259	Returns a random variate from a tabled probability distribution
	"CALL RANTRI" on page 262	Returns a random variate from a triangular distribution
	"CALL RANUNI" on page 264	Returns a random variate from a uniform distribution

	"NORMAL" on page 458	Returns a random variate from a normal distribution
	"RANBIN" on page 511	Returns a random variate from a binomial distribution
	"RANCAU" on page 512	Returns a random variate from a Cauchy distribution
	"RANEXP" on page 513	Returns a random variate from an exponential distribution
	"RANGAM" on page 514	Returns a random variate from a gamma distribution
	"RANNOR" on page 516	Returns a random variate from a normal distribution
	"RANPOI" on page 517	Returns a random variate from a Poisson distribution
	"RANTBL" on page 518	Returns a random variate from a tabled probability
	"RANTRI" on page 520	Random variate from a triangular distribution
	"RANUNI" on page 521	Returns a random variate from a uniform distribution
	"UNIFORM" on page 574	Random variate from a uniform distribution
SAS File I/O	"ATTRC" on page 230	Returns the value of a character attribute for a SAS data set
	"ATTRN" on page 233	Returns the value of a numeric attribute for the specified SAS data set
	"CEXIST" on page 287	Verifies the existence of a SAS catalog or SAS catalog entry and returns a value
	"CLOSE" on page 289	Closes a SAS data set and returns a value
	"CUROBS" on page 304	Returns the observation number of the current observation
	"DROPNOTE" on page 338	Deletes a note marker from a SAS data set or an external file and returns a value
	"DSNAME" on page 339	Returns the SAS data set name that is associated with a data set identifier
	"EXIST" on page 344	Verifies the existence of a SAS data library member
	"FETCH" on page 352	Reads the next nondeleted observation from a SAS data set into the Data Set Data Vector (DDV) and returns a value
	"FETCHOBS" on page 353	Reads a specified observation from a SAS data set into the Data Set Data Vector (DDV) and returns a value
	"GETVARC" on page 390	Returns the value of a SAS data set character variable
	"GETVARN" on page 391	Returns the value of a SAS data set numeric variable
	"IORCMSG" on page 414	Returns a formatted error message for _IORC_
	"LIBNAME" on page 437	Assigns or deassigns a libref for a SAS data library and returns a value
	"LIBREF" on page 438	Verifies that a libref has been assigned and returns a value
	"NOTE" on page 458	Returns an observation ID for the current observation of a SAS data set
	"OPEN" on page 460	Opens a SAS data set and returns a value

	"PATHNAME" on page 463	Returns the physical name of a SAS data library or of an external file, or returns a blank
	"POINT" on page 481	Locates an observation identified by the NOTE function and returns a value
	"REWIND" on page 523	Positions the data set pointer at the beginning of a SAS data set and returns a value
	"SYSMSG" on page 558	Returns the text of error messages or warning messages from the last data set or external file function execution
	"SYSRC" on page 561	Returns a system error number
	"VARFMT" on page 578	Returns the format assigned to a SAS data set variable
	"VARINFMT" on page 580	Returns the informat assigned to a SAS data set variable
	"VARLABEL" on page 581	Returns the label assigned to a SAS data set variable
	"VARLEN" on page 582	Returns the length of a SAS data set variable
	"VARNAME" on page 583	Returns the name of a SAS data set variable
	"VARNUM" on page 584	Returns the number of a variable's position in a SAS data set
	"VARTYPE" on page 587	Returns the data type of a SAS data set variable
Special	"ADDR" on page 227	Returns the memory address of a variable
	"CALL POKE" on page 247	Writes a value directly into memory
	"CALL SYSTEM" on page 271	Submits an operating environment command for execution
	"DIF" on page 328	Returns differences between the argument and its *n*th lag
	"GETOPTION" on page 388	Returns the value of a SAS system or graphics option
	"INPUT" on page 402	Returns the value produced when a SAS expression that uses a specified informat expression is read
	"INPUTC" on page 404	Enables you to specify a character informat at run time
	"INPUTN" on page 405	Enables you to specify a numeric informat at run time
	"LAG" on page 432	Returns values from a queue
	"PEEK" on page 476	Stores the contents of a memory address into a numeric variable
	"PEEKC" on page 477	Stores the contents of a memory address into a character variable
	"POKE" on page 483	Writes a value directly into memory
	"PUT" on page 503	Returns a value using a specified format
	"PUTC" on page 505	Enables you to specify a character format at run time
	"PUTN" on page 506	Enables you to specify a numeric format at run time
	"SYSGET" on page 557	Returns the value of the specified operating environment variable
	"SYSPARM" on page 559	Returns the system parameter string
	"SYSPROD" on page 560	Determines if a product is licensed

	"SYSTEM" on page 561	Issues an operating environment command during a SAS session
State and ZIP Code	"FIPNAME" on page 364	Converts FIPS codes to uppercase state names
	"FIPNAMEL" on page 365	Converts FIPS codes to mixed case state names
	"FIPSTATE" on page 366	Converts FIPS codes to two-character postal codes
	"STFIPS" on page 551	Converts state postal codes to FIPS state codes
	"STNAME" on page 552	Converts state postal codes to uppercase state names
	"STNAMEL" on page 553	Converts state postal codes to mixed case state names
	"ZIPFIPS" on page 622	Converts ZIP codes to FIPS state codes
	"ZIPNAME" on page 623	Converts ZIP codes to uppercase state names
	"ZIPNAMEL" on page 624	Converts ZIP codes to mixed case state names
	"ZIPSTATE" on page 625	Converts ZIP codes to state postal codes
Trigonometric	"ARCOS" on page 228	Returns the arccosine
	"ARSIN" on page 229	Returns the arcsine
	"ATAN" on page 230	Returns the arctangent
	"COS" on page 302	Returns the cosine
	"SIN" on page 544	Returns the sine
	"TAN" on page 562	Returns the tangent
Truncation	"CEIL" on page 286	Returns the smallest integer that is greater than or equal to the argument
	"FLOOR" on page 367	Returns the largest integer that is less than or equal to the argument
	"FUZZ" on page 384	Returns the nearest integer if the argument is within 1E−12
	"INT" on page 407	Returns the integer value
	"ROUND" on page 525	Rounds to the nearest round-off unit
	"TRUNC" on page 573	Truncates a numeric value to a specified length
Variable Control	"CALL LABEL" on page 242	Assigns a variable label to a specified character variable
	"CALL SET" on page 269	Links SAS data set variables to DATA step or macro variables that have the same name and data type
	"CALL VNAME" on page 272	Assigns a variable name as the value of a specified variable
Variable Information	"VARRAY" on page 585	Returns a value that indicates whether the specified name is an array
	"VARRAYX" on page 586	Returns a value that indicates whether the value of the specified argument is an array
	"VFORMAT" on page 590	Returns the format that is associated with the specified variable
	"VFORMATD" on page 591	Returns the format decimal value that is associated with the specified variable

"VFORMATDX" on page 592	Returns the format decimal value that is associated with the value of the specified argument
"VFORMATN" on page 593	Returns the format name that is associated with the specified variable
"VFORMATNX" on page 594	Returns the format name that is associated with the value of the specified argument
"VFORMATW" on page 595	Returns the format width that is associated with the specified variable
"VFORMATWX" on page 596	Returns the format width that is associated with the value of the specified argument
"VFORMATX" on page 597	Returns the format that is associated with the value of the specified argument
"VINARRAY" on page 598	Returns a value that indicates whether the specified variable is a member of an array
"VINARRAYX" on page 599	Returns a value that indicates whether the value of the specified argument is a member of an array
"VINFORMAT" on page 600	Returns the informat that is associated with the specified variable
"VINFORMATD" on page 601	Returns the informat decimal value that is associated with the specified variable
"VINFORMATDX" on page 602	Returns the informat decimal value that is associated with the value of the specified argument
"VINFORMATN" on page 603	Returns the informat name that is associated with the specified variable
"VINFORMATNX" on page 604	Returns the informat name that is associated with the value of the specified argument
"VINFORMATW" on page 605	Returns the informat width that is associated with the specified variable
"VINFORMATWX" on page 606	Returns the informat width that is associated with the value of the specified argument
"VINFORMATX" on page 607	Returns the informat that is associated with the value of the specified argument
"VLABEL" on page 608	Returns the label that is associated with the specified variable
"VLABELX" on page 609	Returns the variable label for the value of a specified argument
"VLENGTH" on page 611	Returns the compile-time (allocated) size of the specified variable
"VLENGTHX" on page 612	Returns the compile-time (allocated) size for the value of the specified argument
"VNAME" on page 613	Returns the name of the specified variable
"VNAMEX" on page 614	Validates the value of the specified argument as a variable name
"VTYPE" on page 615	Returns the type (character or numeric) of the specified variable

	"VTYPEX" on page 616	Returns the type (character or numeric) for the value of the specified argument
Web Tools	"HTMLDECODE" on page 396	Decodes a string containing HTML numeric character references or HTML character entity references and returns the decoded string
	"HTMLENCODE" on page 397	Encodes characters using HTML character entity references and returns the encoded string
	"URLDECODE" on page 575	Returns a string that was decoded using the URL escape syntax
	"URLENCODE" on page 576	Returns a string that was encoded using the URL escape syntax

Dictionary

ABS

Returns the absolute value

Category: Mathematical

Syntax

ABS (*argument*)

Arguments

argument
is numeric.

Details

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

Examples

SAS Statements	Results
`x=abs(2.4);`	**`2.4`**
`x=abs(-3);`	**`3`**

ADDR

Returns the memory address of a variable

Category: Special

Syntax

ADDR(*variable*)

Arguments

variable
specifies a variable name.

Details

The return value is always numeric. Because the storage location of a variable may vary from one execution to the next, based on many factors, the value returned by ADDR may vary. This function is used mostly in combination with the PEEK and PEEKC functions and the POKE CALL routine.

Examples

The following example returns the address at which the variable FIRST is stored:

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

See Also

CALL Routine:
“CALL POKE” on page 247
Functions:
“PEEK” on page 476
“PEEKC” on page 477

AIRY

Returns the value of the airy function

Category: Mathematical

Syntax

AIRY(*x*)

Arguments

x
is numeric.

Details

The AIRY function returns the value of the airy function (Abramowitz and Stegun 1964; Amos, Daniel and Weston 1977) (See “References” on page 626). It is the solution of the differential equation

$$w^{(2)} - xw = 0$$

with the conditions

$$w(0) = \frac{1}{3^{\frac{2}{3}}\Gamma\left(\frac{2}{3}\right)}$$

and

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

Examples

SAS Statements	Results
x=airy(2.0);	**0.0349241304**
x=airy(-2.0);	**0.2274074282**

ARCOS

Returns the arccosine

Category: Trigonometric

Syntax

ARCOS (*argument*)

Arguments

argument
is numeric.

Range: between –1 and 1

Details

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

Examples

SAS Statements	Results
`x=arcos(1);`	`0`
`x=arcos(0);`	`1.5707963268`
`x=arcos(-0.5);`	`2.0943951024`

ARSIN

Returns the arcsine

Category: Trigonometric

Syntax

ARSIN (*argument*)

Arguments

argument
is numeric.

Range: between –1 and 1

Details

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

Examples

SAS Statements	Results
`x=arsin(0);`	**`0`**
`x=arsin(1);`	**`1.5707963268`**
`x=arsin(--0.5);`	**`-0.523598776`**

ATAN

Returns the arctangent

Category: Trigonometric

Syntax

ATAN (*argument*)

Arguments

argument
is numeric.

Details

The ATAN function returns the arctangent (inverse tangent) of the argument. The value returned is in radians.

Examples

SAS Statements	Results
`x=atan(0);`	**`0`**
`x=atan(1);`	**`0.7853981634`**
`x=atan(-9.0);`	**`-1.460139106`**

ATTRC

Returns the value of a character attribute for a SAS data set

Category: SAS File I/O

Syntax

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

Arguments

data-set-id
specifies the data set identifier that the OPEN function returns.

attr-name
is an attribute name. If *attr-name* is invalid, a missing value is returned.

Valid values for use with *attr-name* are:

CHARSET
returns a value for the character set of the machine that created the data set.

empty string	Data set not sorted
ASCII	ASCII character set
EBCDIC	EBCDIC character set
ANSI	OS/2 ANSI standard ASCII character set
OEM	OS/2 OEM code format

ENCRYPT
returns 'YES' or 'NO' depending on whether the SAS data set is encrypted.

ENGINE
returns the name of the engine that is used to access the data set.

LABEL
returns the label assigned to the data set.

LIB
returns the libref of the SAS data library in which the data set resides.

MEM
returns the SAS data set name.

MODE
returns the mode in which the SAS data set was opened, such as:

I	INPUT mode allows random access if the engine supports it; otherwise, it defaults to IN mode.
IN	INPUT mode reads sequentially and allows revisiting observations.
IS	INPUT mode reads sequentially but does not allow revisiting observations.
N	NEW mode creates a new data set.
U	UPDATE mode allows random access if the engine supports it; otherwise, it defaults to UN mode.
UN	UPDATE mode reads sequentially and allows revisiting observations.

US	UPDATE mode reads sequentially but does not allow revisiting observations.
V	UTILITY mode allows modification of variable attributes and indexes associated with the data set.

MTYPE
returns the SAS data library member type.

SORTEDBY
returns an empty string if the data set is not sorted. Otherwise, it returns the names of the BY variables in the standard BY statement format.

SORTLVL
returns a value that indicates how a data set was sorted:

Empty string	Data set is not sorted.
WEAK	Sort order of the data set was established by the user (for example, through the SORTEDBY data set option). The system cannot validate its correctness, so the order of observations cannot be depended on.
STRONG	Sort order of the data set was established by the software (for example, through PROC SORT or the OUT= option in the CONTENTS procedure).

SORTSEQ
returns an empty string if the data set is sorted on the native machine or if the sort collating sequence is the default for the operating environment. Otherwise, it returns the name of the alternate collating sequence used to sort the file.

TYPE
returns the SAS data set type.

Examples

- This example generates a message if the SAS data set has not been opened in INPUT SEQUENTIAL mode. The message is written to the SAS log as follows:

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

- This example tests whether a data set has been sorted and writes the result to the SAS log.

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

See Also

Functions:

“ATTRN” on page 233

“OPEN” on page 460

ATTRN

Returns the value of a numeric attribute for the specified SAS data set

Category: SAS File I/O

Syntax

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

Arguments

data-set-id
specifies the data set identifier that the OPEN function returns.

attr-name
is a numeric attribute, as listed in the section below. If the value of *attr-name* is invalid, a missing value is returned.

Valid numeric values used with *attr-name* are:

ALTERPW
specifies whether a password is required to alter the data set.

1	the data set is alter protected.
0	the data set is not alter protected.

ANOBS
specifies whether the engine knows the number of observations.

1	the engine knows the number of observations.
0	the engine does not know the number of observations.

ANY
specifies whether the data set has observations or variables.

–1	the data set has no observations or variables.
0	the data set has no observations.
1	the data set has observations and variables.

Alias: VAROBS

ARAND
specifies whether the engine supports random access.

1	the engine supports random access.

0	the engine does not support random access.

Alias: RANDOM

ARWU
specifies whether the engine can manipulate files.

1	the engine is not read-only. It can create or update SAS files.
0	the engine is read-only.

CRDTE
specifies the date that the data set was created. The value returned is the internal SAS datetime value for the creation date.

Tip: Use the DATETIME. format to display this value.

ICONST
returns information about the existenece of integrity constraints for a SAS data set.

0	no integrity constraints.
1	one or more general integrity constraints.
2	one or more referential integrity constraints.
3	both one or more general integrity constraints and one or more referential integrity constraints

INDEX
specifies whether the data set supports indexing.

1	indexing is supported.
0	indexing is not supported.

ISINDEX
specifies whether the data set is indexed.

1	at least one index exists for the data set.
0	the data set is not indexed.

ISSUBSET
specifies whether the data set is a subset.

1	at least one WHERE clause is active.
0	no WHERE clause is active.

LRECL
specifies the logical record length.

LRID
specifies the length of the record ID.

MAXGEN
specifies the maximum number of generations.

MAXRC
specifies whether an application checks return codes.

1	an application checks return codes.
0	an application does not check return codes.

MODTE
specifies the last date and time the data set was modified. The value returned is the internal SAS datetime value.

Tip: Use the DATETIME. format to display this value.

NDEL
specifies the number of observations in the data set that are marked for deletion.

NEXTGEN
specifies the next generation number to generate.

NLOBS
specifies the number of logical observations (those not marked for deletion). An active WHERE clause does not affect this number.

–1	the number of observations is not available.

NLOBSF
specifies the number of logical observations (those not marked for deletion) by forcing a read of each observation and taking the FIRSTOBS and OBS system options, and the WHERE clauses into account.

Tip: Passing NLOBSF to ATTRN requires the engine to read every observation from the data set that matches the WHERE clause. Based on the file type and size, this can be a time-consuming process.

NOBS
specifies the number of physical observations (including those that are marked for deletion). An active WHERE clause does not affect this number.

–1	the number of observations is not available.

NVARS
specifies the number of variables in the data set.

PW
specifies whether a password is required to access the data set.

1	the data set is protected.
0	the data set is not protected.

RADIX
specifies whether access by observation number (radix addressability) is allowed.

1	access by observation number is allowed.
0	access by observation number is not allowed.

Note: A data set on a tape engine is index addressable although it cannot be accessed by observation number.

READPW
specifies whether a password is required to read the data set.

1	the data set is read protected.
0	the data set is not read protected.

TAPE
specifies the data set tape file status.

1	the data set is a sequential file.
0	the data set is not a sequential file.

WHSTMT
specifies the active WHERE clauses.

0	no WHERE clause is active.
1	a permanent WHERE clause is active.

2	a temporary WHERE clause is active.
3	both permanent and temporary WHERE clauses are active.

WRITEPW
specifies whether a password is required to write to the data set.

1	the data set is write protected.
0	the data set is not write protected.

Examples

□ This example checks whether a WHERE clause is currently active for a data set.

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

□ This example checks whether a data set is protected with a password.

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

See Also

Functions:
“ATTRC” on page 230
“OPEN” on page 460

BAND

Returns the bitwise logical AND of two arguments

Category: Bitwise Logical Operations

Syntax

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

Arguments

argument-1,argument-2
are numeric, nonnegative, and nonmissing. Separate the arguments with a comma.

Range: 0 to the largest 32-bit unsigned integer

Examples

SAS Statements	Results
`x=band(0Fx,05x);` `put x=hex.;`	`x=00000005`

BETAINV

Returns a quantile from the beta distribution

Category: Quantile

Syntax

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

Arguments

p
: is a numeric probability.

Range: $0 \le p \le 1$

a
: is a numeric shape parameter.

Range: $a > 0$

b
: is a numeric shape parameter.

Range: $b > 0$

Details

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

Note: BETAINV is the inverse of the PROBBETA function. △

Examples

SAS Statements	Results
`x=betainv(0.001,2,4);`	`0.0101017879`

BLSHIFT

Returns the bitwise logical left shift of two arguments

Category: Bitwise Logical Operations

Syntax

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

Arguments

argument-1
is numeric, nonnegative, and nonmissing.

Range: 0 to the largest 32-bit unsigned integer

argument-2
is numeric, nonnegative, and nonmissing.

Range: 0 to 31, inclusive

Examples

SAS Statements	Results

`x=blshift(07x,2);`
`put x=hex.;`
	
	`x=0000001C`

BNOT

Returns the bitwise logical NOT of an argument

Category: Bitwise Logical Operations

Syntax

BNOT(*argument*)

Arguments

argument
is numeric, nonnegative, and nonmissing.

Range: 0 to the largest 32-bit unsigned integer

Examples

SAS Statements	Results
x=bnot(0F000000Fx); **put x=hex.;**	**x=0FFFFFF0**

BOR

Returns the bitwise logical OR of two arguments

Category: Bitwise Logical Operations

Syntax

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

Arguments

argument-1,argument-2
are numeric, non-negative, and nonmissing. Separate the arguments with a comma.

Range: 0 to the largest 32-bit unsigned integer

Examples

SAS Statements	Results
x=bor(01x,0F4x); **put x=hex.;**	**x=000000F5**

BRSHIFT

Returns the bitwise logical right shift of two arguments

Category: Bitwise Logical Operations

Syntax

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

Arguments

argument-1
is numeric, nonnegative, and nonmissing.

Range: 0 to the largest 32-bit unsigned integer

argument-2
is numeric, nonnegative, and nonmissing.

Range: 0 to 31, inclusive

Examples

SAS Statements	Results

`x=brshift(01Cx,2);`
`put x=hex.;`
	`x=00000007`

BXOR

Returns the bitwise logical EXCLUSIVE OR of two arguments

Category: Bitwise Logical Operations

Syntax

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

Arguments

argument-1, argument-2
are numeric, nonnegative, and nonmissing. Separate the arguments with a comma.

Range: 0 to the largest 32-bit unsigned integer

Examples

SAS Statements	Results
`x=bxor(03x,01x);` `put x=hex.;`	`x=00000002`

BYTE

Returns one character in the ASCII or the EBCDIC collating sequence

Category: Character

Syntax

BYTE (*n*)

Arguments

n
specifies an integer that represents a specific ASCII or EBCDIC character.

Range: 0–255

Details

For EBCDIC collating sequences, *n* is between 0 and 255. For ASCII collating sequences, the characters that correspond to values between 0 and 127 represent the standard character set. Other ASCII characters that correspond to values between 128 and 255 are available on certain ASCII operating environments, but the information those characters represent varies from host environment.

Examples

SAS Statements	Results	
	ASCII	EBCDIC
`x=byte(80);`		
`put x;`	`P`	`&`

See Also

Functions:
- "COLLATE" on page 291
- "RANK" on page 516

CALL EXECUTE

Resolves an argument and issues the resolved value for execution

Category: Macro

Syntax

CALL EXECUTE(*argument*);

Arguments

argument
specifies a character expression or a constant that yields a macro invocation or a SAS statement. *Argument* can be:

- □ a character string, enclosed in quotation marks
- □ the name of a DATA step character variable. Do not enclose the name of the DATA step variable in quotation marks.
- □ a character expression that the DATA step resolves to a macro text expression or a SAS statement.

Details

If *argument* resolves to a macro invocation, the macro executes immediately and DATA step execution pauses while the macro executes. If *argument* resolves to a SAS statement or if execution of the macro generates SAS statements, the statement(s) execute after the end of the DATA step that contains the CALL EXECUTE routine. CALL EXECUTE is fully documented in *SAS Macro Language: Reference*.

CALL LABEL

Assigns a variable label to a specified character variable

Category: Variable Control

Syntax

CALL LABEL(*variable-1*,*variable-2*);

Arguments

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

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

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

Details

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

Examples

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

```
data new;
   set old;
      /* lab is not in either array */
   length lab $256;
      /* all character variables in old */
   array abc{*} _character_;
      /* all numeric variables in old */
   array def{*} _numeric_;
   do i=1 to dim(abc);
         /* get label of character variable */
      call label(abc{i},lab);
         /* write label to an observation */
      output;
   end;
   do j=1 to dim(def);
         /* get label of numeric variable */
      call label(def{j},lab);
         /* write label to an observation */
      output;
   end;
   stop;
   keep lab;
```

```
run;
```

See Also

Function:

"VLABEL" on page 608

CALL MODULE

Calls the external routine without any return code

Category: External Routines

Syntax

CALL MODULE(<*cntl-string,*>*module-name*<*,argument-1, ..., argument-n*>);

Arguments

cntl-string

is an optional control string whose first character must be an asterisk (*), followed by any combination of the following characters:

I	prints the hexadecimal representations of all arguments to the CALL MODULE routine. You can use this option to help diagnose problems caused by incorrect arguments or attribute tables. If you specify the I option, the E option is implied.
E	prints detailed error messages. Without the E option (or the I option, which supersedes it), the only error message that the CALL MODULE routine generates is "Invalid argument to function," which is usually not enough information to determine the cause of the error. The E option is useful for a production environment, while the I option is preferable for a development or debugging environment.
H	provides brief help information about the syntax of the CALL MODULE routine, the attribute file format, and suggested SAS formats and informats.

module-name

is the name of the external module to use.

argument

is one or more arguments to pass to the requested routine.

CAUTION:

Be sure to use the correct arguments and attributes. If you use incorrect arguments or attributes, you can cause the SAS System, and possibly your operating system, to fail. △

Details

The CALL MODULE routine executes a routine *module-name* that resides in an external library with the specified arguments.

CALL MODULE builds a parameter list using the information in the arguments and a routine description and argument attribute table that you define in a separate file. The attribute table is a sequential text file that contains descriptions of the routines that you can invoke with the CALL MODULE routine. The purpose of the table is to define how CALL MODULE should interpret its supplied arguments when it builds a parameter list to pass to the external routine. The attribute table should contain a description for each external routine that you intend to call, and descriptions of each argument associated with that routine.

Before you invoke CALL MODULE, you must define the fileref of SASCBTBL to point to the external file that contains the attribute table. You can name the file whatever you want when you create it. This way, you can use SAS variables and formats as arguments to CALL MODULE and ensure that these arguments are properly converted before being passed to the external routine. If you do not define this fileref, CALL MODULE calls the requested routine without altering the arguments.

CAUTION:

Using the CALL MODULE routine without a defined attribute table can cause the SAS System to fail or force you to reset your computer. You need to use an attribute table for all external functions that you want to invoke. △

Comparisons

The two CALL routines and four functions share identical syntax:

- The MODULEN and MODULEC functions return a number and a character, respectively, while the routine CALL MODULE does not return a value.
- The CALL MODULEI routine and the functions MODULEIC and MODULEIN permit vector and matrix arguments. Their return values are scalar. You can invoke CALL MODULEI, MODULEIC, and MODULEIN only from the IML procedure.

Examples

Example 1: Using the CALL MODULE Routine This example calls the **xyz** routine. Use the following attribute table:

```
routine xyz minarg=2 maxarg=2;
arg 1 input num byvalue format=ib4.;
arg 2 output char format=$char10.;
```

The following is the sample SAS code that calls the **xyz** function:

```
data _null_;
   call module('xyz',1,x);
run;
```

Example 2: Using the MODULEIN Function in the IML Procedure This example invokes the **changi** routine from the TRYMOD.DLL module on a Windows platform. Use the following attribute table:

```
routine changi module=trymod returns=long;
arg 1 input num format=ib4. byvalue;
arg 2 update num format=ib4.;
```

The following PROC IML code calls the **changi** function:

```
proc iml;
   x1=J(4,5,0);
   do i=1 to 4;
      do j=1 to 5;
         x1[i,j]=i*10+j+3;
      end;
   end;
   y1=x1;
   x2=x1;
   y2=y1;
   rc=modulein('*i','changi',6,x2);
```

Example 3: Using the MODULEN Function This example calls the **Beep** routine, which is part of the Win32 API in the USER32 Dynamic Link Library on a Windows platform. Use the following attribute table:

```
routine Beep
   minarg=2
   maxarg=2
   stackpop=called
   callseq=byvalue
   module=user32;
arg 1 num format=pib4.;
arg 2 num format=pib4.;
```

The following is the sample SAS code that calls the **Beep** function:

```
filename sascbtbl 'sascbtbl.dat';
data _null_;
   rc=modulen("Beep",1380,1000);
run;
```

The previous code causes the computer speaker to beep.

See Also

CALL Routine:

“CALL MODULEI” on page 246

Functions:

“MODULEC” on page 448
“MODULEIC” on page 449
“MODULEIN” on page 449
“MODULEN” on page 450

CALL MODULEI

Calls the external routine without any return code (in IML environment only)

Category: External Routines

Restriction: CALL MODULEI can only be invoked from within the IML procedure

See: “CALL MODULE” on page 244

Syntax

CALL MODULEI(<*cntl-string,*>*module-name*<,*argument-1*, ..., *argument-n*>);

Details

For details on CALL MODULEI, see “CALL MODULE” on page 244.

See Also

CALL Routine:

“CALL MODULE” on page 244

Functions:

“MODULEC” on page 448
“MODULEIC” on page 449
“MODULEIN” on page 449
“MODULEN” on page 450

CALL POKE

Writes a value directly into memory

Category: Special

Syntax

CALL POKE(*source*,*pointer*<,*length*>);

Arguments

source
specifies a SAS expression that contains a value to write into memory.

pointer
specifies a numeric SAS expression that contains the virtual address of the data that the CALL POKE routine alters.

length
specifies a numeric SAS expression that contains the number of bytes to write from the *source* to the address indicated by *pointer*. If you omit *length*, the action that the

CALL POKE routine takes depends on whether *source* is a character value or a numeric value:

- □ If *source* is a character value, the CALL POKE routine copies the entire value of *source* to the specified memory location.
- □ If *source* is a numeric value, the CALL POKE routine converts *source* into a long integer and writes into memory the number of bytes that constitute a pointer.

  Operating Environment Information: Under OS/390, pointers are 3 or 4 bytes long, depending on the situation. △

Details

CAUTION:

The POKE routine is intended only for experienced programmers in specific cases. If you plan to use this routine, use extreme care both in your programming and in your typing. Writing directly into memory can cause devastating problems. This routine bypasses the normal safeguards that prevent you from destroying a vital element in your SAS session or in another piece of software that is active at the time. △

If you do not have access to the memory location that you specify, the POKE routine returns an "Invalid argument" error.

See Also

Functions:

“ADDR” on page 227
“PEEK” on page 476
“PEEKC” on page 477
“POKE” on page 483

CALL RANBIN

Returns a random variate from a binomial distribution

Category: Random Number

Syntax

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

Arguments

seed

is the seed value. For more information about seeds, see “Seed Values” in “Using Random-Number Functions and CALL Routines” on page 211. A new value for *seed* is returned each time CALL RANBIN is executed.

Range: $seed < 2^{31} - 1$

Note: If $seed \leq 0$, the time of day is used to initialize the seed stream.

n

is an integer number of independent Bernoulli trials.

Range: $n > 0$

p

is a numeric probability of success parameter.

Range: $0<p<1$

x

is a numeric SAS variable. A new value for the random variate *x* is returned each time CALL RANBIN is executed.

Details

The CALL RANBIN routine updates *seed* and returns a variate *x* that is generated from a binomial distribution with mean *np* and variance $np(1–p)$. If $n\leq 50$, $np\leq 5$, or $n(1–p)\leq 5$, SAS uses an inverse transform method applied to a RANUNI uniform variate. If $n>50$, $np>5$, and $n(1–p)>5$, SAS uses the normal approximation to the binomial distribution. In that case, the Box-Muller transformation of RANUNI uniform variates is used.

By adjusting the seeds, you can force streams of variates to agree or disagree for some or all of the observations in the same, or in subsequent, DATA steps.

Comparisons

The CALL RANBIN routine gives greater control of the seed and random number streams than does the RANBIN function.

Examples

This example uses the CALL RANBIN routine:

```
options nodate pageno=1 linesize=80 pagesize=60;

data case;
   retain Seed_1 Seed_2 Seed_3 45;
   n=2000;
   p=.2;
   do i=1 to 10;
      call ranbin(Seed_1,n,p,X1);
      call ranbin(Seed_2,n,p,X2);
      X3=ranbin(Seed_3,n,p);
      if i=5 then
         do;
            Seed_2=18;
            Seed_3=18;
         end;
      output;
   end;
run;

proc print;
   id i;
   var Seed_1-Seed_3 X1-X3;
run;
```

Output 4.3 on page 250 shows the results.

Output 4.3 The RANBIN Example

```
                                The SAS System                              1

           i      Seed_1           Seed_2     Seed_3     X1      X2      X3

           1     1404437564    1404437564       45      385     385     385
           2     1445125588    1445125588       45      399     399     399
           3     1326029789    1326029789       45      384     384     384
           4     1988843719    1988843719       45      421     421     421
           5     2137808851            18       18      430     430     430
           6     1233028129     991271755       18      392     374     392
           7       50049159    1437043694       18      424     384     424
           8      802575599     959908645       18      371     383     371
           9      100573943    1225034217       18      428     388     428
          10      414117170     425626811       18      402     403     402
```

Changing Seed_2 for the CALL RANBIN statement, when I=5, forces the stream of the variates for X2 to deviate from the stream of the variates for X1. Changing Seed_3 on the RANBIN function, however, has no effect.

See Also

Function:
“RANBIN” on page 511

CALL RANCAU

Returns a random variate from a Cauchy distribution

Category: Random Number

Syntax

CALL RANCAU(*seed,x*);

Arguments

seed
is the seed value. For more information on seeds, see “Seed Values” in “Using Random-Number Functions and CALL Routines” on page 211. A new value for *seed* is returned each time CALL RANCAU is executed.

Range: *seed* < 2^{31} - 1

Note: If *seed* ≤ 0, the time of day is used to initialize the seed stream.

x
is a numeric SAS variable. A new value for the random variate *x* is returned each time CALL RANCAU is executed.

Details

The CALL RANCAU routine updates *seed* and returns a variate *x* that is generated from a Cauchy distribution that has a location parameter of 0 and scale parameter of 1.

By adjusting the seeds, you can force streams of variates to agree or disagree for some or all of the observations in the same, or in subsequent, DATA steps.

An acceptance-rejection procedure applied to RANUNI uniform variates is used. If u and v are independent uniform (–1/2, 1/2) variables and $u^2+v^2 \le 1/4$, then u/v is a Cauchy variate.

Comparisons

The CALL RANCAU routine gives greater control of the seed and random number streams than does the RANCAU function.

Examples

This example uses the CALL RANCAU routine:

```
options nodate pageno=1 linesize=80 pagesize=60;

data case;
   retain Seed_1 Seed_2 Seed_3 45;
   do i=1 to 10;
      call rancau(Seed_1,X1);
      call rancau(Seed_2,X2);
      X3=rancau(Seed_3);
      if i=5 then
          do;
             Seed_2=18;
             Seed_3=18;
          end;
      output;
   end;
run;

proc print;
   id i;
   var Seed_1-Seed_3 X1-X3;
run;
```

Output 4.4 on page 251 shows the results.

Output 4.4 The RANCAU Example

```
                              The SAS System                              1

   i      Seed_1          Seed_2     Seed_3       X1          X2          X3

   1    1404437564     1404437564      45      -1.14736    -1.14736    -1.14736
   2    1326029789     1326029789      45      -0.23735    -0.23735    -0.23735
   3    1988843719     1988843719      45      -0.15474    -0.15474    -0.15474
   4    1233028129     1233028129      45       4.97935     4.97935     4.97935
   5      50049159             18      18       0.20402     0.20402     0.20402
   6     802575599      991271755      18       3.43645     4.44427     3.43645
   7    1233458739     1437043694      18       6.32808    -1.79200     6.32808
   8      52428589      959908645      18       0.18815    -1.67610     0.18815
   9    1216356463     1225034217      18       0.80689     3.88391     0.80689
  10    1711885541      425626811      18       0.92971    -1.31309     0.92971
```

Changing Seed_2 for the CALL RANCAU statement, when I=5, forces the stream of the variates for X2 to deviate from the stream of the variates for X1. Changing Seed_3 on the RANCAU function, however, has no effect.

See Also

Function:

“RANCAU” on page 512

CALL RANEXP

Returns a random variate from an exponential distribution

Category: Random Number

Syntax

CALL RANEXP(*seed*,*x*);

Arguments

seed

is the seed value. For more information about seeds, see “Seed Values” in “Using Random-Number Functions and CALL Routines” on page 211. A new value for *seed* is returned each time CALL RANEXP is executed.

Range: *seed* $< 2^{31} - 1$

Note: If *seed* ≤ 0, the time of day is used to initialize the seed stream.

x

is a numeric variable. A new value for the random variate *x* is returned each time CALL RANEXP is executed.

Details

The CALL RANEXP routine updates *seed* and returns a variate *x* that is generated from an exponential distribution that has a parameter of 1.

By adjusting the seeds, you can force streams of variates to agree or disagree for some or all of the observations in the same, or in subsequent, DATA steps.

The CALL RANEXP routine uses an inverse transform method applied to a RANUNI uniform variate.

Comparisons

The CALL RANEXP routine gives greater control of the seed and random number streams than does the RANEXP function.

Examples

This example uses the CALL RANEXP routine:

```
options nodate pageno=1 linesize=80 pagesize=60;

data case;
   retain Seed_1 Seed_2 Seed_3 45;
   do i=1 to 10;
      call ranexp(Seed_1,X1);
      call ranexp(Seed_2,X2);
      X3=ranexp(Seed_3);
      if i=5 then
         do;
            seed_2=18;
            seed_3=18;
         end;
      output;
   end;
   run;

proc print;
   id i;
   var Seed_1-Seed_3 X1-X3;
run;
```

Output 4.5 on page 253 shows the results.

Output 4.5 The RANEXP Example

```
                              The SAS System                              1

   i      Seed_1        Seed_2      Seed_3       X1         X2         X3

   1    694315054     694315054       45      1.12913    1.12913    1.12913
   2   1404437564    1404437564       45      0.42466    0.42466    0.42466
   3   2130505156    2130505156       45      0.00794    0.00794    0.00794
   4   1445125588    1445125588       45      0.39610    0.39610    0.39610
   5   1013861398            18       18      0.75053    0.75053    0.75053
   6   1326029789     417047966       18      0.48211    0.57102    0.48211
   7    932142747     850344656       18      0.83457    0.92566    0.83457
   8   1988843719    2067665501       18      0.07674    0.16730    0.07674
   9    516966271     607886093       18      1.42407    0.51513    1.42407
  10   2137808851    1550198721       18      0.00452    0.91543    0.00452
```

Changing Seed_2 for the CALL RANEXP statement, when I=5, forces the stream of the variates for X2 to deviate from the stream of the variates for X1. Changing Seed_3 on the RANEXP function, however, has no effect.

See Also

Function:

"RANEXP" on page 513

CALL RANGAM

Returns a random variate from a gamma distribution

Category: Random Number

Syntax

CALL RANGAM(*seed*,*a*,*x*);

Arguments

seed

is the seed value. For more information about seeds, see "Seed Values" in "Using Random-Number Functions and CALL Routines" on page 211. A new value for *seed* is returned each time CALL RANGAM is executed.

Range: *seed* < 2^{31} - 1

Note: If *seed* ≤ 0, the time of day is used to initialize the seed stream.

a

is a numeric shape parameter.

Range: $a > 0$

x

is a numeric variable. A new value for the random variate *x* is returned each time CALL RANGAM is executed.

Details

The CALL RANGAM routine updates *seed* and returns a variate *x* that is generated from a gamma distribution with parameter *a*.

By adjusting the seeds, you can force streams of variates to agree or disagree for some or all of the observations in the same, or in subsequent, DATA steps.

For a>1, an acceptance-rejection method (Cheng 1977) (See "References" on page 626) is used. For $a \leq 1$, an acceptance-rejection method due to (Fishman 1978) (See "References" on page 626) is used.

Comparisons

The CALL RANGAM routine gives greater control of the seed and random number streams than does the RANGAM function.

Examples

This example uses the CALL RANGAM routine:

```
options nodate pageno=1 linesize=80 pagesize=60;

data case;
   retain Seed_1 Seed_2 Seed_3 45;
   a=2;
   do i=1 to 10;
      call rangam(Seed_1,a,X1);
      call rangam(Seed_2,a,X2);
      X3=rangam(Seed_3,a);
      if i=5 then
         do;
            Seed_2=18;
            Seed_3=18;
         end;
      output;
   end;
run;

proc print;
   id i;
   var Seed_1-Seed_3 X1-X3;
run;
```

Output 4.6 on page 255 shows the results.

Output 4.6 The RANGAM Example

```
                              The SAS System                              1

     i      Seed_1          Seed_2    Seed_3       X1         X2         X3

     1    1404437564    1404437564      45      1.30569    1.30569    1.30569
     2    1326029789    1326029789      45      1.87514    1.87514    1.87514
     3    1988843719    1988843719      45      1.71597    1.71597    1.71597
     4      50049159      50049159      45      1.59304    1.59304    1.59304
     5     802575599            18      18      0.43342    0.43342    0.43342
     6     100573943     991271755      18      1.11812    1.32646    1.11812
     7    1986749826    1437043694      18      0.68415    0.88806    0.68415
     8      52428589     959908645      18      1.62296    2.46091    1.62296
     9    1216356463    1225034217      18      2.26455    4.06596    2.26455
    10     805366679     425626811      18      2.16723    6.94703    2.16723
```

Changing Seed_2 for the CALL RANGAM statement, when I=5, forces the stream of the variates for X2 to deviate from the stream of the variates for X1. Changing Seed_3 on the RANGAM function, however, has no effect.

See Also

Function:
"RANGAM" on page 514

CALL RANNOR

Returns a random variate from a normal distribution

Category: Random Number

Syntax

CALL RANNOR(*seed*,*x*);

Arguments

seed
is the seed value. For more information about seeds, see "Seed Values" in "Using Random-Number Functions and CALL Routines" on page 211. A new value for *seed* is returned each time CALL RANNOR is executed.

Range: *seed* $< 2^{31} - 1$

Note: If *seed* ≤ 0, the time of day is used to initialize the seed stream.

x
is a numeric variable. A new value for the random variate *x* is returned each time CALL RANNOR is executed.

Details

The CALL RANNOR routine updates *seed* and returns a variate *x* that is generated from a normal distribution, with mean 0 and variance 1.

By adjusting the seeds, you can force streams of variates to agree or disagree for some or all of the observations in the same, or in subsequent, DATA steps.

The CALL RANNOR routine uses the Box-Muller transformation of RANUNI uniform variates.

Comparisons

The CALL RANNOR routine gives greater control of the seed and random number streams than does the RANNOR function.

Examples

This example uses the CALL RANNOR routine:

```
options nodate pageno=1 linesize=80 pagesize=60;
```

```
data case;
   retain Seed_1 Seed_2 Seed_3 45;
   do i=1 to 10;
      call rannor(Seed_1,X1);
      call rannor(Seed_2,X2);
      X3=rannor(Seed_3);
      if i=5 then
         do;
            Seed_2=18;
            Seed_3=18;
         end;
      output;
   end;
run;

proc print;
   id i;
   var Seed_1-Seed_3 X1-X3;
run;
```

Output 4.7 on page 257 shows the results.

Output 4.7 The RANNOR Example

```
                              The SAS System                              1

   i      Seed_1          Seed_2     Seed_3       X1          X2          X3

   1    1404437564    1404437564      45      -0.85252    -0.85252    -0.85252
   2    1445125588    1445125588      45      -0.05865    -0.05865    -0.05865
   3    1326029789    1326029789      45      -0.90628    -0.90628    -0.90628
   4    1988843719    1988843719      45       1.15526     1.15526     1.15526
   5    2137808851            18      18       1.68697     1.68697     1.68697
   6    1233028129     991271755      18      -0.47276    -1.44726    -0.47276
   7      50049159    1437043694      18       1.33423    -0.87677     1.33423
   8     802575599     959908645      18      -1.63511    -0.97261    -1.63511
   9     100573943    1225034217      18       1.55410    -0.64742     1.55410
  10     414117170     425626811      18       0.10736     0.14963     0.10736
```

Changing Seed_2 for the CALL RANNOR statement, when I=5, forces the stream of the variates for X2 to deviate from the stream of the variates for X1. Changing Seed_3 on the RANNOR function, however, has no effect.

See Also

Function:

“RANNOR” on page 516

CALL RANPOI

Returns a random variate from a Poisson distribution

Category: Random Number

Syntax

CALL RANPOI(*seed,m,x*);

Arguments

seed
is the seed value. For more information about seeds, see "Seed Values" in "Using Random-Number Functions and CALL Routines" on page 211. A new value for *seed* is returned each time CALL RANPOI is executed.

Range: $seed < 2^{31} - 1$

Note: If $seed \leq 0$, the time of day is used to initialize the seed stream.

m
is a numeric mean parameter.

Range: $m \geq 0$

x
is a numeric variable. A new value for the random variate *x* is returned each time CALL RANPOI is executed.

Details

The CALL RANPOI routine updates *seed* and returns a variate *x* that is generated from a Poisson distribution, with mean *m*.

By adjusting the seeds, you can force streams of variates to agree or disagree for some or all of the observations in the same, or in subsequent, DATA steps.

For $m < 85$, an inverse transform method applied to a RANUNI uniform variate is used. (Fishman 1976) (See "References" on page 626). For $m \geq 85$, the normal approximation of a Poisson random variable is used. To expedite execution, internal variables are calculated only on initial calls (that is, with each new *m*).

Comparisons

The CALL RANPOI routine gives greater control of the seed and random number streams than does the RANPOI function.

Examples

This example uses the CALL RANPOI routine:

```
options nodate pageno=1 linesize=80 pagesize=60;

data case;
   retain Seed_1 Seed_2 Seed_3 45;
   m=120;
   do i=1 to 10;
      call ranpoi(Seed_1,m,X1);
      call ranpoi(Seed_2,m,X2);
      X3=ranpoi(Seed_3,m);
      if i=5 then
         do;
            Seed_2=18;
            Seed_3=18;
         end;
      output;
   end;
run;

proc print;
   id i;
   var Seed_1-Seed_3 X1-X3;
run;
```

Output 4.8 on page 259 shows the results.

Output 4.8 The RANPOI Example

```
                              The SAS System                              1

       i      Seed_1          Seed_2    Seed_3     X1      X2      X3

       1    1404437564    1404437564      45       111     111     111
       2    1445125588    1445125588      45       119     119     119
       3    1326029789    1326029789      45       110     110     110
       4    1988843719    1988843719      45       133     133     133
       5    2137808851            18      18       138     138     138
       6    1233028129     991271755      18       115     104     115
       7      50049159    1437043694      18       135     110     135
       8     802575599     959908645      18       102     109     102
       9     100573943    1225034217      18       137     113     137
      10     414117170     425626811      18       121     122     121
```

Changing Seed_2 for the CALL RANPOI statement, when I=5, forces the stream of the variates for X2 to deviate from the stream of the variates for X1. Changing Seed_3 on the RANPOI function, however, has no effect.

See Also

Function:

“RANPOI” on page 517

CALL RANTBL

Returns a random variate from a tabled probability distribution

Category: Random Number

Syntax

CALL RANTBL(*seed*,p_1, . . . p_i, . . . , p_n,*x*);

Arguments

seed
is the seed value. For more information about seeds, see “Seed Values” in “Using Random-Number Functions and CALL Routines” on page 211. A new value for *seed* is returned each time CALL RANTBL is executed.

Range: *seed* $< 2^{31} - 1$

Note: If *seed* ≤ 0, the time of day is used to initialize the seed stream.

p_i
is a numeric SAS value.

Range: $0 \leq p_i \leq 1$ for $0 < i \leq n$

x
is a numeric SAS variable. A new value for the random variate *x* is returned each time CALL RANTBL is executed.

Details

The CALL RANTBL routine updates *seed* and returns a variate *x* generated from the probability mass function defined by p_1 through p_n.

By adjusting the seeds, you can force streams of variates to agree or disagree for some or all of the observations in the same, or in subsequent, DATA steps.

An inverse transform method applied to a RANUNI uniform variate is used. The CALL RANTBL routine returns these data:

$$\begin{array}{ll}
1 & \text{with probability } p_1 \\
2 & \text{with probability } p_2 \\
. & \\
. & \\
. & \\
\text{n} & \text{with probability } p_n \\
\text{n} + 1 & \text{with probability } 1 - \sum_{i=1}^{n} p_i \quad \text{if } \sum_{i=1}^{n} p_i \leq 1
\end{array}$$

If, for some index $j<n$,

$$\sum_{i=1}^{j} p_i \geq 1$$

RANTBL returns only the indices 1 through *j*, with the probability of occurrence of the index *j* equal to

$$1 - \sum_{i=1}^{j-1} p_i$$

Comparisons

The CALL RANTBL routine gives greater control of the seed and random number streams than does the RANTBL function.

Examples

This example uses the CALL RANTBL routine:

```
options nodate pageno=1 linesize=80 pagesize=60;

data case;
    retain Seed_1 Seed_2 Seed_3 45;
    input p1-p9;
    do i=1 to 10;
       call rantbl(Seed_1,of p1-p9,X1);
       call rantbl(Seed_2,of p1-p9,X2);
       X3=rantbl(Seed_3,of p1-p9);
       if i=5 then
          do;
             Seed_2=18;
             Seed_3=18;
          end;
       output;
    end;
    datalines;
.02 .04 .06 .08 .1 .12 .14 .16 .18
;

proc print;
   id i;
   var Seed_1-Seed_3 X1-X3;
run;
```

Output 4.9 on page 261 shows the results.

Output 4.9 The RANTBL Example

```
                              The SAS System                                 1
          i      Seed_1          Seed_2     Seed_3     X1     X2     X3

          1     694315054     694315054       45        6      6      6
          2    1404437564    1404437564       45        8      8      8
          3    2130505156    2130505156       45       10     10     10
          4    1445125588    1445125588       45        8      8      8
          5    1013861398            18       18        7      7      7
          6    1326029789     707222751       18        8      6      8
          7     932142747     991271755       18        7      7      7
          8    1988843719     422705333       18       10      4     10
          9     516966271    1437043694       18        5      8      5
         10    2137808851    1264538018       18       10      8     10
```

Changing Seed_2 for the CALL RANTBL statement, when I=5, forces the stream of variates for X2 to deviate from the stream of variates for X1. Changing Seed_3 on the RANTBL function, however, has no effect.

See Also

Function:
"RANTBL" on page 518

CALL RANTRI

Returns a random variate from a triangular distribution

Category: Random Number

Syntax

CALL RANTRI(*seed*,*h*,*x*);

Arguments

seed
is the seed value. For more information about seeds, see "Seed Values" in "Using Random-Number Functions and CALL Routines" on page 211. A new value for *seed* is returned each time CALL RANTRI is executed.

Range: $seed < 2^{31} - 1$

Note: If $seed \le 0$, the time of day is used to initialize the seed stream.

h
is a numeric SAS value.

Range: $0<h<1$

x
is a numeric SAS variable. A new value for the random variate *x* is returned each time CALL RANTRI is executed.

Details

The CALL RANTRI routine updates *seed* and returns a variate *x* generated from a triangular distribution on the interval (0,1) with parameter *h*, which is the modal value of the distribution.

By adjusting the seeds, you can force streams of variates to agree or disagree for some or all of the observations in the same, or in subsequent, DATA steps.

The CALL RANTRI routine uses an inverse transform method applied to a RANUNI uniform variate.

Comparisons

The CALL RANTRI routine gives greater control of the seed and random number streams than does the RANTRI function.

Examples

This example uses the CALL RANTRI routine:

```
options nodate pageno=1 linesize=80 pagesize=60;

data case;
   retain Seed_1 Seed_2 Seed_3 45;
   h=.2;
   do i=1 to 10;
      call rantri(Seed_1,h,X1);
      call rantri(Seed_2,h,X2);
      X3=rantri(Seed_3,h);
      if i=5 then
         do;
            Seed_2=18;
            Seed_3=18;
         end;
      output;
   end;
run;


proc print;
   id i;
   var Seed_1-Seed_3 X1-X3;
run;
```

Output 4.10 on page 263 shows the results.

Output 4.10 The RANTRI Example

```
                                The SAS System                               1

     i       Seed_1          Seed_2     Seed_3        X1          X2          X3

     1     694315054      694315054       45      0.26424     0.26424     0.26424
     2    1404437564     1404437564       45      0.47388     0.47388     0.47388
     3    2130505156     2130505156       45      0.92047     0.92047     0.92047
     4    1445125588     1445125588       45      0.48848     0.48848     0.48848
     5    1013861398             18       18      0.35015     0.35015     0.35015
     6    1326029789      707222751       18      0.44681     0.26751     0.44681
     7     932142747      991271755       18      0.32713     0.34371     0.32713
     8    1988843719      422705333       18      0.75690     0.19841     0.75690
     9     516966271     1437043694       18      0.22063     0.48555     0.22063
    10    2137808851     1264538018       18      0.93997     0.42648     0.93997
```

Changing Seed_2 for the CALL RANTRI statement, when I=5, forces the stream of the variates for X2 to deviate from the stream of the variates for X1. Changing Seed_3 on the RANTRI function has, however, no effect.

See Also

Function:

"RANTRI" on page 520

CALL RANUNI

Returns a random variate from a uniform distribution

Category: Random Number

Syntax

CALL RANUNI(*seed*,*x*);

Arguments

seed

is the seed value. For more information about seeds, see "Seed Values" in "Using Random-Number Functions and CALL Routines" on page 211. A new value for *seed* is returned each time CALL RANUNI is executed.

Range: *seed* $< 2^{31} - 1$

Note If *seed* ≤ 0, the time of day is used to initialize the seed stream.

x

is a numeric variable. A new value for the random variate *x* is returned each time CALL RANUNI is executed.

Details

The CALL RANUNI routine updates *seed* and returns a variate *x* that is generated from the uniform distribution on the interval (0,1), using a prime modulus multiplicative generator with modulus 2^{31}–1 and multiplier 397204094 (Fishman and Moore 1982) (See "References" on page 626).

By adjusting the seeds, you can force streams of variates to agree or disagree for some or all of the observations in the same, or in subsequent, DATA steps.

Comparisons

The CALL RANUNI routine gives greater control of the seed and random number streams than does the RANUNI function.

Examples

This example uses the CALL RANUNI routine:

```
options nodate pageno=1 linesize=80 pagesize=60;

data case;
```

```
   retain Seed_1 Seed_2 Seed_3 45;
   do i=1 to 10;
      call ranuni(Seed_1,X1);
      call ranuni(Seed_2,X2);
      X3=ranuni(Seed_3);
      if i=5 then
         do;
            Seed_2=18;
            Seed_3=18;
         end;
      output;
   end;
run;

proc print;
   id i;
   var Seed_1-Seed_3 X1-X3;
run;
```

Output 4.11 on page 265 shows the results.

Output 4.11 The RANUNI Example

```
                                The SAS System                              1

     i      Seed_1         Seed_2     Seed_3       X1         X2         X3

     1     694315054      694315054      45      0.32332    0.32332    0.32332
     2    1404437564     1404437564      45      0.65399    0.65399    0.65399
     3    2130505156     2130505156      45      0.99209    0.99209    0.99209
     4    1445125588     1445125588      45      0.67294    0.67294    0.67294
     5    1013861398             18      18      0.47212    0.47212    0.47212
     6    1326029789      707222751      18      0.61748    0.32933    0.61748
     7     932142747      991271755      18      0.43406    0.46160    0.43406
     8    1988843719      422705333      18      0.92613    0.19684    0.92613
     9     516966271     1437043694      18      0.24073    0.66918    0.24073
    10    2137808851     1264538018      18      0.99549    0.58885    0.99549
```

Changing Seed_2 for the CALL RANUNI statement, when I=5, forces the stream of the variates for X2 to deviate from the stream of the variates for X1. Changing Seed_3 on the RANUNI function, however, has no effect.

See Also

Function:

“RANUNI” on page 521

CALL RXCHANGE

Changes one or more substrings that match a pattern

Category: Character String Matching

Restriction: Use with the RXPARSE function

Syntax

CALL RXCHANGE (*rx, times, old-string*<, *new-string*>);

Arguments

rx
specifies a numeric value that is returned from the RXPARSE function.

Tip: The value of *rx* points to an expression that is parsed by RXPARSE. CALL RXCHANGE uses the expression to find and change a matching substring.

times
is a numeric value that specifies the maximum number of times to change matching substrings.

Restriction: Maximum value of *times* is 2,147,483,647.

old-string
specifies the character expression to be searched and changed as a result of the change operation.

new-string
is a character variable that contains the result of the change operation.

Details

If *old-string* is a character variable, *new-string* is optional. If *new-string* is omitted, *old-string* is changed in place.

If *old-string* is an expression that is not a character variable, you must specify *new-string*.

Comparisons

The regular expression (RX) functions and CALL routines work together to manipulate strings that match patterns. Use the RXPARSE function to parse a pattern you specify. Use the RXMATCH function and the CALL RXCHANGE and CALL RXSUBSTR routines to match or modify your data. Use the CALL RXFREE routine to free allocated space.

Example

See the RXPARSE function "Example" on page 538.

See Also

Functions and CALL routines:

- "CALL RXFREE" on page 267
- "CALL RXSUBSTR" on page 268
- "RXMATCH" on page 526
- "RXPARSE" on page 527

CALL RXFREE

Frees memory allocated by other regular expression (RX) functions and CALL routines

Category: Character String Matching

Restriction: Use with the RXPARSE function

Syntax

CALL RXFREE (*rx*);

Arguments

rx
specifies a numeric value that is returned from the RXPARSE function.

Comparisons

The regular expression (RX) functions and CALL routines work together to manipulate strings that match patterns. Use the RXPARSE function to parse a pattern you specify. Use the RXMATCH function and the CALL RXCHANGE and CALL RXSUBSTR routines to match or modify your data. Use the CALL RXFREE routine to free allocated space.

Example

See the RXPARSE function "Example" on page 538.

See Also

Functions and CALL routines:

“CALL RXCHANGE” on page 265
“CALL RXSUBSTR” on page 268
“RXMATCH” on page 526
“RXPARSE” on page 527

CALL RXSUBSTR

Finds the position, length, and score of a substring that matches a pattern

Category: Character String Matching

Restriction: Use with the RXPARSE function

Syntax

CALL RXSUBSTR (*rx, string, position*);

CALL RXSUBSTR (*rx, string, position, length*);

CALL RXSUBSTR (*rx, string, position, length, score*);

Arguments

rx
specifies a numeric value that is returned from the RXPARSE function.

string
specifies the character expression to be searched.

position
specifies a numeric position in a string where the substring that is matched by the pattern begins. If there is no match, the result is zero.

length
specifies a numeric value that is the length of the substring that is matched by the pattern.

score
specifies an integer value based on the number of matches for a particular pattern in a substring.

Details

CALL RXSUBSTR searches the variable *string* for the pattern from RXPARSE, returns the position of the start of the string, indicates the length of the matched string, and returns a score value. By default, when a pattern matches more than one substring beginning at a specific position, the longest substring is selected.

Comparisons

CALL RXSUBSTR performs the same matching as RXMATCH, but CALL RXSUBSTR additionally allows you to use the *length* and *score* arguments to receive more information about the match.

The regular expression (RX) functions and CALL routines work together to manipulate strings that match patterns. Use the RXPARSE function to parse a pattern you specify. Use the RXMATCH function and the CALL RXCHANGE and CALL RXSUBSTR routines to match or modify your data. Use the CALL RXFREE routine to free allocated space.

Example

See the RXPARSE function “Example” on page 538.

See Also

Functions and CALL routines:

“CALL RXCHANGE” on page 265

“CALL RXFREE” on page 267

“RXMATCH” on page 526

“RXPARSE” on page 527

CALL SET

Links SAS data set variables to DATA step or macro variables that have the same name and data type

Category: Variable Control

Syntax

CALL SET(*data-set-id*);

Arguments

data-set-id

is the identifier that is assigned by the OPEN function when the data set is opened.

Details

Using SET can significantly reduce the coding that is required for accessing variable values for modification or verification when you use functions to read or to manipulate a SAS file. After a CALL SET, whenever a read is performed from the SAS data set, the values of the corresponding macro or DATA step variables are set to the values of the matching SAS data set variables. If the variable lengths do not match, the values are truncated or padded according to need. If you do not use SET, then you must use the

GETVARC and GETVARN functions to move values explicitly between data set variables and macro or DATA step variables.

As a general rule, use CALL SET immediately following OPEN if you want to link the data set and the macro and DATA step variables.

Examples

This example uses the CALL SET routine:

- The following statements automatically set the values of the macro variables PRICE and STYLE when an observation is fetched:

```
%macro setvar;
   %let dsid=%sysfunc(open(sasuser.houses,i));
      /* No leading ampersand with %SYSCALL */
   %syscall set(dsid);
   %let rc=%sysfunc(fetchobs(&dsid,10));
   %let rc=%sysfunc(close(&dsid));
%mend setvar;

%global price style;
%setvar
%put _global_;
```

- The %PUT statement writes these lines to the SAS log:

```
GLOBAL PRICE 127150
GLOBAL STYLE CONDO
```

- The following statements obtain the values for the first 10 observations in SASUSER.HOUSES and store them in MYDATA:

```
data mydata;
      /* create variables for assignment */
      /*by CALL SET */
   length style $8 sqfeet bedrooms baths 8
      street $16 price 8;
   drop rc dsid;
   dsid=open("sasuser.houses","i");
   call set (dsid);
   do i=1 to 10;
      rc=fetchobs(dsid,i);
      output;
   end;
run;
```

See Also

Functions:

“FETCH” on page 352

“FETCHOBS” on page 353

“GETVARC” on page 390

“GETVARN” on page 391

CALL SYMPUT

Assigns DATA step information to a macro variable

Category: Macro

Syntax

CALL SYMPUT(*argument-1*,*argument-2*);

Arguments

argument-1
specifies a character expression that identifies the macro variable that is assigned a value. If the macro variable does not exist, the routine creates it.

argument-2
specifies a character expression that contains the value that is assigned.

Details

The CALL SYMPUT routine either creates a macro variable whose value is information from the DATA step or assigns a DATA step value to an existing macro variable. CALL SYMPUT is fully documented in *SAS Macro Language: Reference*.

See Also

Function:

“SYMGET” on page 557

CALL SYSTEM

Submits an operating environment command for execution

Category: Special

Syntax

CALL SYSTEM(*command*);

Arguments

command
specifies any of the following: a system command enclosed in quotation marks (explicit character string), an expression whose value is a system command, or the name of a character variable whose value is a system command that is executed.

Comparisons

The behavior of the CALL SYSTEM routine is similar to that of the X command, the X statement, and the SYSTEM function. It is useful in certain situations because it can be conditionally executed, it accepts an expression as an argument, and it is executed at run time.

See Also

Function:
“SYSTEM” on page 561

CALL VNAME

Assigns a variable name as the value of a specified variable

Category: Variable Control

Syntax

CALL VNAME(*variable-1*,*variable-2*);

Arguments

variable-1
specifies any SAS variable.

variable-2
specifies any SAS character variable. Because SAS variable names can contain up to 32 characters, the length of *variable-2* should be at least 32.

Details

The CALL VNAME routine assigns the name of the *variable-1* variable as the value of the *variable-2* variable.

Examples

This example uses the CALL VNAME routine with array references to return the names of all variables in the data set OLD:

```
data new(keep=name);
   set old;
      /* all character variables in old */
   array abc{*} _character_;
      /* all numeric variables in old */
   array def{*} _numeric_;
      /* name is not in either array */
   length name $32;
   do i=1 to dim(abc);
         /* get name of character variable */
      call vname(abc{i},name);
         /* write name to an observation */
      output;
   end;
   do j=1 to dim(def);
         /* get name of numeric variable */
      call vname(def{j},name);
         /* write name to an observation */
      output;
   end;
   stop;
run;
```

See Also

Functions:

“VNAME” on page 613

“VNAMEX” on page 614

CDF

Computes cumulative distribution functions

Category: Probability

Syntax

CDF (*’dist’*,*quantile*,*parm-1*, . . . ,*parm-k*)

Arguments

’dist’

is a character string that identifies the distribution. Valid distributions are as follows:

Distribution	Argument
Bernoulli	**'BERNOULLI'**
Beta	**'BETA'**
Binomial	**'BINOMIAL'**
Cauchy	**'CAUCHY'**
Chi-squared	**'CHISQUARED'**
Exponential	**'EXPONENTIAL'**
F	**'F'**
Gamma	**'GAMMA'**
Geometric	**'GEOMETRIC'**
Hypergeometric	**'HYPERGEOMETRIC'**
Laplace	**'LAPLACE'**
Logistic	**'LOGISTIC'**
Lognormal	**'LOGNORMAL'**
Negative binomial	**'NEGBINOMIAL'**
Normal	**'NORMAL' | 'GAUSS'**
Pareto	**'PARETO'**
Poisson	**'POISSON'**
T	**'T'**
Uniform	**'UNIFORM'**
Wald (inverse Gaussian)	**'WALD' | 'IGAUSS'**
Weibull	**'WEIBULL'**

Note: Except for *T* and *F*, any distribution can be minimally identified by its first four characters. △

quantile
is a numeric random variable.

parm-1, . . . ,parm-k
are *shape*, *location*, or *scale* parameters appropriate for the specific distribution. See the description for each distribution in "Details" for complete information about these parameters.

Details

Bernoulli Distribution

CDF('BERNOULLI',*x*,*p*)

where

x
is a numeric random variable.

p
is a numeric probability of success.

Range: $0 \le p \le 1$

The CDF function for the Bernoulli distribution returns the probability that an observation from a Bernoulli distribution, with probability of success equal to *p*, is less than or equal to *x*. The equation follows:

$$CDF\left('BERN', x, p\right) = \begin{cases} 0 & x < 0 \\ 1 - p & 0 \leq x < 1 \\ 1 & x \geq 1 \end{cases}$$

Note: There are no *location* or *scale* parameters for this distribution. △

Beta Distribution

CDF('BETA',*x*,*a*,*b*<,*l*,*r*>)

where

x
: is a numeric random variable.

a
: is a numeric shape parameter.

Range: *a* > 0

b
: is a numeric shape parameter, with *b* > 0.

Range: *b* > 0

l
: is an optional numeric left location parameter.

r
: is an optional right location parameter.

Range: *r* > *l*

The CDF function for the beta distribution returns the probability that an observation from a beta distribution, with shape parameters *a* and *b*, is less than or equal to *x*. The following equation describes the CDF function of the Beta distribution:

$$CDF\left('BETA', x, a, b, l, r\right) = \begin{cases} 0 & x \leq 1 \\ \frac{1}{\beta(a,b)} \int\limits_{l}^{x} \frac{(x-1)^{a-1}(r-x)^{b-1}}{(r-l)^{a+b-1}} dx & l < x \leq r \\ 1 & x > r \end{cases}$$

where

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

and

$$\Gamma\left(a\right) = \int\limits_{0}^{\infty} x^{a-1} e^{-x} dx$$

Note: The default values for *l* and *r* are 0 and 1, respectively. △

Binomial Distribution

CDF('BINOMIAL',*m*,*p*,*n*)

where

m
: is an integer random variable that counts the number of successes.

p
: is a numeric parameter that is the probability of success.

Range: $0 \le p \le 1$

n
: is an integer parameter that counts the number of independent Bernoulli trials.

Range: $n > 0$

The CDF function for the binomial distribution returns the probability that an observation from a binomial distribution, with parameters *p* and *n*, is less than or equal to *m*. The equation follows:

$$CDF\left('BINOM', m, p, n\right) = \begin{cases} 0 & m < 0 \\ \sum_{j=0}^{m} \binom{n}{i} p^j \left(1-p\right)^{n-j} & 0 \le m \le n \\ 1 & m > n \end{cases}$$

Note: There are no *location* or *scale* parameters for the binomial distribution. △

Cauchy Distribution

CDF('CAUCHY',*x*<,*θ*,*λ*>)

where

x
: is a numeric random variable.

θ
: is an optional numeric location parameter.

λ
: is an optional numeric scale parameter.

Range: $\lambda > 0$

The CDF function for the Cauchy distribution returns the probability that an observation from a Cauchy distribution, with location parameter θ and scale parameter λ, is less than or equal to *x*. The equation follows:

$$CDF\left('CAUCHY', x, \theta, \lambda\right) = \frac{1}{2} + \frac{1}{\pi} tan^{-1}\left(\frac{x-\theta}{\lambda}\right)$$

Note: The default values for θ and λ are 0 and 1, respectively. △

Chi-squared Distribution

CDF('CHISQUARED',*x*,*df* <,*nc*>)

where

x
: is a numeric random variable.

df
: is a numeric degrees of freedom parameter.

  Range: *df* > 0

nc
: is an optional numeric noncentrality parameter.

  Range: *nc* ≥ 0

The CDF function for the chi-squared distribution returns the probability that an observation from a chi-squared distribution, with *df* degrees of freedom and noncentrality parameter *nc*, is less than or equal to *x*. This function accepts noninteger degrees of freedom. If *nc* is omitted or equal to zero, the value returned is from the central chi-squared distribution. The following equation describes the CDF function of the chi-squared distribution:

$$CDF\left('CHISQ', x, \nu, \lambda\right) = \begin{cases} 0 & x < 0 \\ \sum\limits_{j=0}^{\infty} e^{-\frac{\lambda}{2}} \frac{\left(\frac{\lambda}{2}\right)^j}{j!} P_c\left(x, \nu + 2j\right) & x \geq 0 \end{cases}$$

where $P_c(.,.)$ denotes the probability from the central chi-squared distribution:

$$P_c\left(x, a\right) = P_g\left(\frac{x}{2}, \frac{a}{2}\right)$$

and where $P_g(y,b)$ is the probability from the Gamma distribution given by

$$P_g\left(y, b\right) = \frac{}{\Gamma\left(b\right)} \int\limits_0^y e^{-v} v^{b-} dv$$

Exponential Distribution

CDF('EXPONENTIAL',*x* <,λ>)

where

x
: is a numeric random variable.

λ
: is an optional scale parameter.

  Range: λ > 0

The CDF function for the exponential distribution returns the probability that an observation from an exponential distribution, with scale parameter λ, is less than or equal to *x*. The equation follows:

$$CDF\left('EXPO', x, \lambda\right) = \begin{cases} 0 & x < 0 \\ 1 - e^{-\frac{x}{\lambda}} & x \geq 0 \end{cases}$$

Note: The default value for λ is 1. △

F Distribution

CDF('F',*x*,*ndf*,*ddf* <,*nc*>)

where

x
: is a numeric random variable.

ndf
: is a numeric numerator degrees of freedom parameter.

  Range: *ndf* > 0

ddf
: is a numeric denominator degrees of freedom parameter.

  Range: *ddf* > 0

nc
: is a numeric noncentrality parameter.

  Range: *nc* ≥ 0

The CDF function for the *F* distribution returns the probability that an observation from an *F* distribution, with *ndf* numerator degrees of freedom, *ddf* denominator degrees of freedom, and noncentrality parameter *nc*, is less than or equal to *x*. This function accepts noninteger degrees of freedom for *ndf* and *ddf*. If *nc* is omitted or equal to zero, the value returned is from a central *F* distribution. The following equation describes the CDF function of the *F* distribution:

$$CDF\left('F', x, v_1, v_2, \lambda\right) = \begin{cases} 0 & x < 0 \\ \sum_{j=0}^{\infty} e^{-\frac{\lambda}{2}} \frac{\left(\frac{\lambda}{2}\right)^j}{j!} P_f\left(f, v_1 + 2j, v_2\right) & x \geq 0 \end{cases}$$

where $P_f(f,u_1,u_2)$ is the probability from the central *F* distribution with

$$P_f\left(f, u_1, u_2\right) = P_B\left(\frac{u_1 f}{u_1 f + u_2}, \frac{u_1}{2}, \frac{u_2}{2}\right)$$

and $P_B(x,a,b)$ is the probability from the standard Beta distribution.

Note: There are no *location* or *scale* parameters for the *F* distribution. △

Gamma Distribution

CDF('GAMMA',*x*,*a*<,λ>)

where

x
: is a numeric random variable.

a
: is a numeric shape parameter.

  Range: *a* > 0

λ

is an optional numeric scale parameter.

Range: $\lambda > 0$

The CDF function for the Gamma distribution returns the probability that an observation from a Gamma distribution, with shape parameter *a* and scale parameter *λ*, is less than or equal to *x*. The equation follows:

$$CDF\left('GAMMA', x, a, \lambda\right) = \begin{cases} 0 & x < 0 \\ \frac{1}{\lambda^a \Gamma(a)} \int\limits_0^x v^{a-1} e^{-\frac{v}{\lambda}} dv & x \geq 0 \end{cases}$$

Note: The default value for λ is 1. △

Geometric Distribution

CDF('GEOMETRIC',*m*,*p*)

where

m

is a numeric random variable that denotes the number of failures.

Range: $m \geq 0$

p

is a numeric probability.

Range: $0 \leq p \leq 1$

The CDF function for the geometric distribution returns the probability that an obervation from a geometric distribution, with parameter *p*, is less than or equal to *m*. The equation follows:

$$CDF\left('GEOM', m, p\right) = \begin{cases} 0 & m < 0 \\ \sum\limits_{j=0}^{j \leq m} p\left(1-p\right)^j & m \geq 0 \end{cases}$$

Note: There are no *location* or *scale* parameters for this distribution. △

Hypergeometric Distribution

CDF('HYPER',*x*,*m*,*k*,*n*<,*r*>)

where

x

is an integer random variable.

m

is an integer population size parameter, with $m \geq 1$.

Range:

k

is an integer number of items in the category of interest.

Range: $0 \leq k \leq m$

n

is an integer sample size parameter.

Range: $0 \le n \le m$

r

is an optional numeric odds ratio parameter.

Range: $r > 0$

The CDF function for the hypergeometric distribution returns the probability that an observation from an extended hypergeometric distribution, with population size *m*, number of items *k*, sample size *n*, and odds ratio *r*, is less than or equal to *x*. If *r* is omitted or equal to 1, the value returned is from the usual hypergeometric distribution. The equation follows:

$$CDF\left('HYPER', x, m, k, n, r\right) = \begin{cases} 0 & x < max\left(0, k+n-m\right) \\ \dfrac{\sum_{i=0}^{x} \binom{k}{i}\binom{m-k}{n-i} r^{i}}{\sum_{j=max(0,k+n-m)}^{min(k,n)} \binom{k}{j}\binom{m-k}{n-j} r^{j}} & max\left(0, k+n-m\right) \le x \le min\left(k, n\right) \\ 1 & x > min\left(k, n\right) \end{cases}$$

Laplace Distribution

CDF('LAPLACE',*x*<,θ,λ>)

where

x

is a numeric random variable.

θ

is an optional numeric location parameter.

λ

is an optional numeric scale parameter.

Range: $\lambda > 0$

The CDF function for the Laplace distribution returns the probability that an observation from the Laplace distribution, with location parameter θ and scale parameter λ, is less than or equal to *x*. The equation follows:

$$CDF\left('LAPLACE', x, \theta, \lambda\right) = \begin{cases} \frac{1}{2}e^{\frac{(x-\theta)}{\lambda}} & x > 0 \\ 1 - \frac{1}{2}e^{\left(-\frac{(x-\theta)}{\lambda}\right)} & x \ge 0 \end{cases}$$

Note: The default values for θ and λ are 0 and 1, respectively. △

Logistic Distribution

CDF('LOGISTIC',*x*<,θ,λ>)

where

x

is a numeric random variable.

θ

is an optional numeric location parameter

λ

is an optional numeric scale parameter.

Range: $\lambda > 0$

The CDF function for the logistic distribution returns the probability that an observation from a logistic distribution, with a location parameter θ and a scale parameter λ, is less than or equal to *x*. The equation follows:

$$CDF\left('LOGISTIC', x, \theta, \lambda\right) = \frac{1}{1 + e^{-\frac{x-\theta}{\lambda}}}$$

Note: The default values for θ and λ are 0 and 1, respectively. △

Lognormal Distribution

CDF('LOGNORMAL',*x*<,θ,λ>)

where

x

is a numeric random variable.

θ

is an optional numeric location parameter

λ

is an optional numeric scale parameter.

Range: $\lambda > 0$

The CDF function for the lognormal distribution returns the probability that an observation from a lognormal distribution, with location parameter θ and scale parameter λ, is less than or equal to *x*. The equation follows:

$$CDF\left('LOGN', x, \theta, \lambda\right) = \begin{cases} 0 & x \le 0 \\ \frac{1}{\lambda\sqrt{2\pi}} \int\limits_{o}^{log(x)} exp\left(-\frac{(v-\theta)^2}{2\lambda^2}\right) dv & x > 0 \end{cases}$$

Note: The default values for θ and λ are 0 and 1, respectively. △

Negative Binomial Distribution

CDF('NEGBINOMIAL',*m*,*p*,*n*)

where

m

is a positive integer random variable that counts the number of failures.

Range: $m \ge 0$

p

is a numeric probability of success parameter.

Range: $0 \le p \le 1$

n

is an integer parameter that counts the number of successes.

Range: $n \ge 1$

The CDF function for the negative binomial distribution returns the probability that an observation from a negative binomial distribution, with probability of success *p* and number of successes *n*, is less than or equal to *m*. The equation follows:

$$CDF\left('NEGB', m, p, n\right) = \begin{cases} 0 & m < 0 \\ p^n \sum\limits_{j=0}^{m} \binom{n+j-1}{j} (1-p)^j & m \geq 0 \end{cases}$$

Note: There are no *location* or *scale* parameters for the negative binomial distribution. △

Normal Distribution

CDF('NORMAL',*x*<,θ,λ>)

where

x
: is a numeric random variable.

θ
: is an optional numeric location parameter.

λ
: is an optional numeric scale parameter.

Range: $\lambda > 0$

The CDF function for the normal distribution returns the probability that an observation from the normal distribution, with location parameter θ and scale parameter λ, is less than or equal to *x*. The equation follows:

$$CDF\left('NORMAL', x, \theta, \lambda\right) = \frac{1}{\lambda\sqrt{2\pi}} \int_{-\infty}^{x} exp\left(-\frac{(v-\theta)^2}{2\lambda^2}\right) dv$$

Note: The default values for θ and λ are 0 and 1, respectively. △

Pareto Distribution

CDF('PARETO',*x*,*a*<,*k*>)

where

x
: is a numeric random variable.

a
: is a numeric shape parameter.

Range: $a > 0$

k
: is an optional numeric scale parameter.

Range: $k > 0$

The CDF function for the Pareto distribution returns the probability that an observation from a Pareto distribution, with shape parameter *a* and scale parameter *k*, is less than or equal to *x*. The equation follows:

$$CDF\left('PARETO',x,a,k\right)=\begin{cases}0 & x<k\\ 1-\left(\frac{k}{x}\right)^{a} & x\geq k\end{cases}$$

Note: The default value for *k* is 1. △

Poisson Distribution

CDF('POISSON',*n*,*m*)

where

n
: is an integer random variable.

m
: is a numeric mean parameter.

Range: *m* > 0

The CDF function for the Poisson distribution returns the probability that an observation from a Poisson distribution, with mean *m*, is less than or equal to *n*. The equation follows:

$$CDF\left('POISSON',n,m\right)=\begin{cases}0 & n<0\\ \sum_{i=0}^{n} exp\left(-m\right)\frac{m^{i}}{i!} & n\geq 0\end{cases}$$

Note: There are no *location* or *scale* parameters for the Poisson distribution. △

T Distribution

CDF('T',*t*,*df*<,*nc*>)

where

t
: is a numeric random variable.

df
: is a numeric degrees of freedom parameter

Range: *df* > 0

nc
: is an optional numeric noncentrality parameter.

The CDF function for the *T* distribution returns the probability that an observation from a *T* distribution, with degrees of freedom *df* and noncentrality parameter *nc*, is less than or equal to *x*. This function accepts noninteger degrees of freedom. If *nc* is omitted or equal to zero, the value returned is from the central *T* distribution. The equation follows:

$$CDF\left('T',t,v,\delta\right)=\frac{}{2^{\left(\frac{1}{2}v-\right)}\Gamma\left(\frac{v}{2}\right)}\int_{0}^{\infty}x^{v-1}e^{-\frac{1}{2}x^{2}}\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\frac{tx}{\sqrt{v}}}e^{-\frac{1}{2}(u-\delta)^{2}}du\,dx$$

Note: There are no *location* or *scale* parameters for the *T* distribution. △

Uniform Distribution

CDF('UNIFORM',*x*<,*l*,*r*>)

where

x
: is a numeric random variable.

l
: is an optional numeric left location parameter.

r
: is an optional numeric right location parameter.

Range: $r > l$

The CDF function for the uniform distribution returns the probability that an observation from a uniform distribution, with left location parameter *l* and right location parameter *r*, is less than or equal to *x*. The equation follows:

$$CDF\left('UNIFORM', x, l, r\right) = \begin{cases} 0 & x < l \\ \frac{x-l}{r-l} & l \le x < r \\ & x \ge r \end{cases}$$

Note: The default values for *l* and *r* are 0 and 1, respectively. △

Wald (Inverse Gaussian) Distribution

CDF('WALD',*x*,*d*)

CDF('IGAUSS',*x*,*d*)

where

x
: is a numeric random variable.

d
: is a numeric shape parameter.

Range: $d > 0$

The CDF function for the Wald distribution returns the probability that an observation from a Wald distribution, with shape parameter *d*, is less than or equal to *x*. The equation follows:

$$CDF\left('WALD', x, d\right) = \begin{cases} 0 & x \le 0 \\ \Phi\left((x-1)\sqrt{\frac{d}{x}}\right) + e^{2d}\Phi\left(-(x+1)\sqrt{\frac{d}{x}}\right) & x > 0 \end{cases}$$

where Φ(.) denotes the probability from the standard normal distribution.

Note: There are no *location* or *scale* parameters for the Wald distribution. △

Weibull Distribution

CDF('WEIBULL',*x*,*a*<,*λ*>)

where

x
: is a numeric random variable.

a

is a numeric shape parameter.

Range: $a > 0$

λ

is an optional numeric scale parameter.

Range: $\lambda > 0$

The CDF function for the Weibull distribution returns the probability that an observation from a Weibull distribution, with shape parameter *a* and scale parameter λ is less than or equal to *x*. The equation follows:

$$CDF\left('WEIBULL', x, a, \lambda\right) = \begin{cases} 0 & x < 0 \\ 1 - e^{-\left(\frac{x}{\lambda}\right)^{a}} & x \geq 0 \end{cases}$$

Note: The default value for λ is 1. △

Examples

SAS Statements	Results
`y=cdf('BERN',0,.25);`	`0.75`
`y=cdf('BERN',1,.25);`	`1.0`
`y=cdf('BETA',0.2,3,4);`	`0.09888`
`y=cdf('BINOM',4,.5,10);`	`0.37695`
`y=cdf('CAUCHY',2);`	`0.85242`
`y=cdf('CHISQ',11.264,11);`	`0.57858`
`y=cdf('EXPO',1);`	`0.63212`
`y=cdf('F',3.32,2,3);`	`0.82639`
`y=cdf('GAMMA',1,3);`	`0.080301`
`y=cdf('HYPER',2,200,50,10);`	`0.52367`
`y=cdf('LAPLACE',1);`	`0.81606`
`y=cdf('LOGISTIC',1);`	`0.73106`
`y=cdf('LOGNORMAL',1);`	`0.5`
`y=cdf('NEGB',1,.5,2);`	`0.5`
`y=cdf('NORMAL',1.96);`	`0.97500`
`y=cdf('PARETO',1,1);`	`0`
`y=cdf('POISSON',2,1);`	`0.91970`
`y=cdf('T',.9,5);`	`0.79531`
`y=cdf('UNIFORM',0.25);`	`0.25`

SAS Statements	Results
`y=cdf('WALD',1,2);`	**0.62770**
`y=cdf('WEIBULL',1,2);`	**0.63212**

CEIL

Returns the smallest integer that is greater than or equal to the argument

Category: Truncation

Syntax

CEIL (*argument*)

Arguments

argument
is numeric.

Details

If the argument is within 10**(–12) of an integer, the function returns that integer.

Examples

The following SAS statements produce these results:

SAS Statements	Results
`var1=2.1;` `a=ceil(var1);` `put a=;`	**3**
`var2=3;` `b=ceil(var2);` `put b=;`	**3**
`c=ceil(-2.4);` `put c=;`	**–2**
`d=ceil(1+1.e-11);` `put d=;`	**2**

SAS Statements	Results

`e=ceil(-1+1.e-11);`
`put e=;`
	0

`f=ceil(1+1.e-13);`
`put f=;`
	1

CEXIST

Verifies the existence of a SAS catalog or SAS catalog entry and returns a value

Category: SAS File I/O

Syntax

CEXIST(*entry*<,'U'>)

Arguments

entry
specifies a SAS catalog, or the name of an entry in a catalog. If the *entry* value is a one- or two-level name, then it is assumed to be the name of a catalog. Use a three- or four-level name to test for the existence of an entry within a catalog.

'U'
tests whether the catalog can be opened for updating.

Details

CEXIST returns 1 if the SAS catalog or catalog entry exists, or 0 if the SAS catalog or catalog entry does not exist.

Examples

- This example verifies the existence of the entry X.PROGRAM in LIB.CAT1:

```
data _null_;
   if cexist("lib.cat1.x.program") then
   put "Entry X.PROGRAM exists";
run;
```

- This example tests whether the catalog LIB.CAT1 exists and can be opened for update (see Example Code 4.1 on page 288). If the catalog does not exist, a message is written to the SAS log. Note that in a macro statement you do not enclose character strings in quotation marks.

Example Code 4.1 Determining if LIB.CAT1 Can Be Opened for Update

```
%if %sysfunc(cexist(lib.cat1,u)) %then
   %put The catalog LIB.CAT1 exists and can be opened for update.;
%else
   %put %sysfunc(sysmsg());
```

See Also

Function:

“EXIST” on page 344

CINV

Returns a quantile from the chi-squared distribution

Category: Quantile

Syntax

CINV (*p*,*df*<,*nc*>)

Arguments

p
is a numeric probability.

Range: $0 \leq p < 1$

df
is a numeric degrees of freedom parameter.

Range: $df > 0$

nc
is a numeric noncentrality parameter.

Range: $nc \geq 0$

Details

The CINV function returns the p^{th} quantile from the chi-square distribution with degrees of freedom *df* and a noncentrality parameter *nc*. The probability that an observation from a chi-square distribution is less than or equal to the returned quantile is *p*. This function accepts a noninteger degrees of freedom parameter *df*.

If the optional parameter *nc* is not specified or has the value 0, the quantile from the central chi-square distribution is returned. The noncentrality parameter *nc* is defined such that if X is a normal random variable with mean μ and variance 1, X^2 has a noncentral chi-square distribution with $df=1$ and $nc = \mu^2$.

CAUTION:
For large values of *nc*, the algorithm could fail; in that case, a missing value is returned. △

Note: CINV is the inverse of the PROBCHI function. △

Examples

These statements show how to find the 95th percentile from a central chi-square distribution with 3 degrees of freedom and the 95th percentile from a noncentral chi-square distribution with 3.5 degrees of freedom and a noncentrality parameter equal to 4.5.

SAS Statements	Results
`q1=cinv(.95,3);`	`7.8147279033`
`a2=cinv(.95,3.5,4.5);`	`7.504582117`

CLOSE

Closes a SAS data set and returns a value

Category: SAS File I/O

Syntax

CLOSE(*data-set-id*)

Arguments

data-set-id
specifies the data set identifier that the OPEN function returns.

Details

CLOSE returns 0 if the operation was successful, ≠0 if it was not successful. Close all SAS data sets as soon as they are no longer needed by the application.

Note: All data sets opened within a DATA step are closed automatically at the end of the DATA step. △

Examples

- □ This example uses OPEN to open the SAS data set PAYROLL. If the data set opens successfully, indicated by a positive value for the variable PAYID, the example uses CLOSE to close the data set.

```
%let payid=%sysfunc(open(payroll,is));
   macro statements
%if &payid > 0 %then
   %let rc=%sysfunc(close(&payid));
```

□ This example opens the SAS data set MYDATA within a DATA step. MYDATA is closed automatically at the end of the DATA step. You do not need to use CLOSE to close the file.

```
data _null_;
   dsid=open('mydata','i');
   if dsid > 0 then do;
      ...more statements...
   end;
run;
```

See Also

Function:
"OPEN" on page 460

CNONCT

Returns the noncentrality parameter from a chi-squared distribution

Category: Mathematical

Syntax

CNONCT(*x*,*df*,*prob*)

Arguments

x
is a numeric random variable.
Range: $x \geq 0$

df
is a numeric degrees of freedom parameter.
Range: $df > 0$

prob
is a probability.
Range: $0 < prob < 1$

Details

The CNONCT function returns the nonnegative noncentrality parameter from a noncentral chi-square distribution whose parameters are *x*, *df*, and *nc*. If *prob* is greater than the probability from the central chi-square distribution with the parameters *x* and *df*, a root to this problem does not exist. In this case a missing value is returned. A Newton-type algorithm is used to find a nonnegative root *nc* of the equation

$$P_c\left(x|df, nc\right) - prob = 0$$

where

$$P_c\left(x|df,nc\right) = e^{\frac{-nc}{2}} \sum_{j=0}^{\infty} \frac{\left(\frac{nc}{2}\right)^j}{j!} P_g\left(\frac{x}{2}|\frac{df}{2}+j\right)$$

where $P_g\left(x|a\right)$ is the probability from the gamma distribution given by

$$P_g\left(x|a\right) = \frac{1}{\Gamma\left(a\right)} \int_0^x t^{a-1}e^{-t}dt$$

If the algorithm fails to converge to a fixed point, a missing value is returned.

Examples

```
data work;
   x=2;
   df=4;
   do nc=1 to 3 by .5;
      prob=probchi(x,df,nc);
      ncc=cnonct(x,df,prob);
      output;
   end;
run;
proc print;
run;
```

Output 4.12 Computations of the Noncentrality Parameters from the Chi-squared Distribution

```
            OBS    x    df     nc      prob      ncc

             1     2     4    1.0    0.18611     1.0
             2     2     4    1.5    0.15592     1.5
             3     2     4    2.0    0.13048     2.0
             4     2     4    2.5    0.10907     2.5
             5     2     4    3.0    0.09109     3.0
```

COLLATE

Returns an ASCII or EBCDIC collating sequence character string

Category: Character

Syntax

COLLATE (*start-position*<,*end-position*>) |

(*start-position*<,,*length*>)

Arguments

start-position

specifies the numeric position in the collating sequence of the first character to be returned.

Interaction: If you specify only *start-position*, COLLATE returns consecutive characters from that position to the end of the collating sequence or up to 255 characters, whichever comes first.

end-position

specifies the numeric position in the collating sequence of the last character to be returned.

The maximum *end-position* for the EBCDIC collating sequence is 255. For ASCII collating sequences, the characters that correspond to *end-position* values between 0 and 27 represent the standard character set. Other ASCII characters that correspond to *end-position* values between 128 and 255 are available on certain ASCII operating environments, but the information those characters represents varies from host environment.

Tip: *end-position* must be larger than *start-position*

Tip: If you specify *end-position*, COLLATE returns all character values in the collating sequence between *start-position* and *end-position*, inclusive.

Tip: If you omit *end-position* and use *length*, mark the *end-position* place with a comma.

length

specifies the number of characters in the collating sequence.

Default: 200

Tip: If you omit *end-position*, use *length* to specify the length of the result explicitly.

Details

If you specify both *end-position* and *length*, COLLATE ignores *length*. If you request a string longer than the remainder of the sequence, COLLATE returns a string through the end of the sequence.

Examples

SAS Statements	Results
ASCII	`----+----1----+-----2--`

`x=collate(48,,10);`
`y=collate(48,57);`
`put @1 x @14 y;`
	`0123456789   0123456789`

SAS Statements	Results
EBCDIC	
`x=collate(240,,10);` `y=collate(240,249);` `put @1 x @14 y;`	`0123456789   0123456789`

See Also

Functions:

"BYTE" on page 241

"RANK" on page 516

COMB

Computes the number of combinations of *n* elements taken *r* at a time and returns a value

Category: Mathematical

Syntax

COMB(*n*, *r*)

Arguments

n

is an integer that represents the total number of elements from which the sample is chosen.

r

is an integer that represents the number of chosen elements.

Restriction: $r \le n$

Details

The mathematical representation of the COMB function is given by the following equation:

$$COMB(n,r) = \binom{n}{r} = \frac{n!}{r!\,(n-r)!}$$

with $n \ge 0$, $r \ge 0$, and $n \ge r$.

If the expression cannot be computed, a missing value is returned.

Examples

SAS Statements	Result
`x=comb(5,1);`	**5**

See Also

Functions:

“FACT” on page 346

“PERM” on page 480

COMPBL

Removes multiple blanks from a character string

Category: Character

Syntax

COMPBL(*source*)

Arguments

source
specifies the source string to compress.

Details

The COMPBL function removes multiple blanks in a character string by translating each occurrence of two or more consecutive blanks into a single blank.

The value that the COMPBL function returns has a default length of 200. You can use the LENGTH statement, before calling COMPBL, to set the length of the value.

Comparisons

The COMPRESS function removes every occurrence of the specific character from a string. If you specify a blank as the character to remove from the source string, the COMPRESS function is similar to the COMPBL function. However, the COMPRESS function removes all blanks from the source string, while the COMPBL function compresses multiple blanks to a single blank and has no affect on a single blank.

Examples

SAS Statements	Results
	`----+----1----+-----2--`
`string='Hey`	
` Diddle  Diddle';`	
`string=compbl(string);`	
`put string;`	`Hey Diddle Diddle`
`string='125    E Main St';`	
`length address $10;`	
`address=compbl(string);`	
`put address;`	`125 E Main`

See Also

Function:

“COMPRESS” on page 296

COMPOUND

Returns compound interest parameters

Category: Financial

Syntax

COMPOUND(*a*,*f*,*r*,*n*)

Arguments

a

is numeric, the initial amount.

Range: $a \geq 0$

f

is numeric, the future amount (at the end of *n* periods).

Range: $f \geq 0$

r

is numeric, the periodic interest rate expressed as a fraction.

Range: $r \geq 0$

n

is an integer, the number of compounding periods.

Range: $n \geq 0$

Details

The COMPOUND function returns the missing argument in the list of four arguments from a compound interest calculation. The arguments are related by

$$f = a\,(1 + r)^n$$

One missing argument must be provided. It is then calculated from the remaining three. No adjustment is made to convert the results to round numbers.

Examples

The accumulated value of an investment of $2000 at a nominal annual interest rate of 9 percent, compounded monthly after 30 months, can be expressed as

```
future=compound(2000,.,0.09/12,30);
```

The value returned is 2502.54. The second argument has been set to missing, indicating that the future amount is to be calculated. The 9 percent nominal annual rate has been converted to a monthly rate of 0.09/12. The rate argument is the fractional (not the percentage) interest rate per compounding period.

COMPRESS

Removes specific characters from a character string

Category: Character

Syntax

COMPRESS(*source*<, *characters-to-remove*>)

Arguments

source
specifies a source string that contains the characters to remove.

characters-to-remove
specifies the character or characters that SAS removes from the character string.

Tip: Enclose a literal string of characters in quotation marks.

Tip: If you specify nothing, SAS removes blanks from *source*.

Examples

Example 1: Compressing Blanks

SAS Statements	Results
	`----+----1`
`a='AB C D ';`	
`b=compress(a);`	
`put b;`	`ABCD`

Example 2: Compressing Special Characters

SAS Statements	Results
	`----+----1`

`x='A.B (C=D);';`
`y=compress(x,'.;()');`
`put y;`
	`AB C=D`

See Also

Functions:
- "COMPBL" on page 294
- "LEFT" on page 435
- "TRIM" on page 570

CONSTANT

Computes some machine and mathematical constants and returns a value

Category: Mathematical

Syntax

CONSTANT(*constant*<, *parameter*>)

Arguments

constant
is a character string that identifies the constant. Valid constants are

Constant	Argument
The natural base	**'E'**
Euler constant	**'EULER'**
Pi	**'PI'**
Exact integer	**'EXACTINT' <,nbytes>**
The largest double-precision number	**'BIG'**
The log with respect to *base* of BIG	**'LOGBIG' <,base>**
The square root of BIG	**'SQRTBIG'**
The smallest double-precision number	**'SMALL'**
The log with respect to *base* of SMALL	**'LOGSMALL' <,base>**
The square root of SMALL	**'SQRTSMALL'**
Machine precision constant	**'MACEPS'**

Constant	Argument
The log with respect to *base* of MACEPS	**'LOGMACEPS' <,base>**
The square root of MACEPS	**'SQRTMACEPS'**

parameter
is an optional numeric parameter. Some of the constants specified in *constant* have an optional argument that alters the functionality of the CONSTANT function.

Details

The natural base

CONSTANT('E')

The natural base is described by the following equation:

$$\lim_{x \to 0} (1+x)^{\frac{1}{x}} \approx 2.718281828459045$$

Euler constant

CONSTANT('EULER')

Euler's constant is described by the following equation:

$$\lim_{n \to \infty} \left\{ \sum_{j=0}^{j=n} \frac{1}{j} - \log(n) \right\} \approx 0.577215664901532860$$

Pi

CONSTANT('PI')

Pi is the well-known constant in trigonometry that is the ratio between the circumference and the diameter of a circle. Many expressions exist for computing this constant. One such expression for the series is described by the following equation:

$$4 \sum_{j=0}^{j=\infty} \frac{(-1)^j}{2j+1} \approx 3.14159265358979323846$$

Exact Integer

CONSTANT('EXACTINT' <, *nbytes*>)

where

nbytes
is a numeric value that is the number of bytes.

Range: 2 ≤ *nbytes* ≤ 8

Default: 8

The exact integer is the largest integer *k* such that all integers less than or equal to *k* in absolute value have an exact representation in a SAS numeric variable of length *nbytes*. This information can be useful to know before you trim a SAS numeric variable from the default 8 bytes of storage to a lower number of bytes to save storage.

The largest double-precision number

CONSTANT('BIG')

This case returns the largest double-precision floating point number (8-bytes) that is representable on your computer.

CONSTANT('LOGBIG' <, *base*>)

where

base
is a numeric value that is the base of the logarithm.

Restriction: The *base* that you specify must be greater than the value of 1+SQRTMACEPS.

Default: the natural base, E.

This case returns the logarithm with respect to *base* of the largest double-precision floating point number (8-bytes) that is representable on your computer.

It is safe to exponentiate the given *base* raised to a power less than or equal to CONSTANT('LOGBIG', *base*) by using the power operation (**) without causing any overflows.

It is safe to exponentiate any floating point number less than or equal to CONSTANT('LOGBIG') by using the exponential function, EXP, without causing any overflows.

CONSTANT('SQRTBIG')

This case returns the square root of the largest double-precision floating point number (8-bytes) that is representable on your computer.

It is safe to square any floating point number less than or equal to CONSTANT('SQRTBIG') without causing any overflows.

The smallest double-precision number

CONSTANT('SMALL')

This case returns the smallest double-precision floating point number (8-bytes) that is representable on your computer.

CONSTANT('LOGSMALL' <, *base*>)

where

base
is a numeric value that is the base of the logarithm.

Restriction: The *base* that you specify must be greater than the value of 1+SQRTMACEPS.

Default: the natural base, E.

This case returns the logarithm with respect to *base* of the smallest double-precision floating point number (8-bytes) that is representable on your computer.

It is safe to exponentiate the given *base* raised to a power greater than or equal to CONSTANT('LOGSMALL', *base*) by using the power operation (**) without causing any underflows or 0.

It is safe to exponentiate any floating point number greater than or equal to CONSTANT('LOGSMALL') by using the exponential function, EXP, without causing any underflows or 0.

CONSTANT('SQRTSMALL')

This case returns the square root of the smallest double-precision floating point number (8-bytes) that is representable on the machine.

It is safe to square any floating point number greater than or equal to CONSTANT('SQRTBIG') without causing any underflows or 0.

Machine precision

CONSTANT('MACEPS')

This case returns the smallest double-precision floating point number (8-bytes) $\epsilon = 2^{-j}$ for some integer *j*, such that $1 + \epsilon > 1$.

This constant is important in finite precision computations. A number n_1 is considered larger than another number n_2 if the (8-byte) representation of $n_1 + n_2$ is identical to n_1. This constant can be used in summing series to implement a machine dependent stopping criterion.

CONSTANT('LOGMACEPS' <, *base*>)

where

base

is a numeric value that is the base of the logarithm.

Restriction: The *base* that you specify must be greater than the value of 1+SQRTMACEPS.

Default: the natural base, E.

This case returns the logarithm with respect to *base* of CONSTANT('MACEPS').

CONSTANT('SQRTMACEPS')

This case returns the square root of CONSTANT('MACEPS').

CONVX

Returns the convexity for an enumerated cashflow

Category: Financial

Syntax

CONVX(*y,f,c(1), ... ,c(k)*)

Arguments

y

the effective per-period yield-to-maturity, expressed as a fraction.

Range: $0 < y < 1$

f

the frequency of cashflows per period.

Range: $f > 0$

c(1), ... ,c(k)
a list of cashflows.

Details

The CONVX function returns the value

$$C = \frac{\sum_{k=1}^{K} \frac{k}{f} \frac{c(k)}{(1+fy)^{\frac{k}{f}}}}{P\left(1+y\right)^{2}}$$

where

$$P = \sum_{k=1}^{K} \frac{c(k)}{(1+fy)^{\frac{k}{f}}}$$

Examples

```
c=convx(1/20,1,.33,.44,.55,.49,.50,.22,.4,.8,.01,.36,.2,.4);
```

The value returned is 42.3778.

CONVXP

Returns the convexity for a periodic cashflow stream, such as a bond

Category: Financial

Syntax

CONVXP(A,c,n,K,k_0,y)

Arguments

A
the par value.
Range: $A > 0$

c
the nominal per-period coupon rate, expressed as a fraction.
Range: $0 < c < 1$

n
the number of coupons per period.

Range: $n > 0$ and is an integer

K

the number of remaining coupons.

Range: $K > 0$ and is an integer

k_0

the time from present to the first coupon date, expressed in terms of the number of periods.

Range: $0 < k_0 > 1/n$

y

the nominal per-period yield-to-maturity, expressed as a fraction.

Range: $y > 0$

Details

The CONVXP function returns the value

$$C = \frac{1}{n} \frac{\sum_{k=1}^{K} t_k \frac{c(k)}{\left(1+\frac{y}{n}\right)^{t_k}}}{P\left(1+\frac{y}{n}\right)^2}$$

where

$t_k = k - (1 - nk_0)$

$c(k) = \frac{c}{n}A$ for k=1, ..., K-1

$c(K) = \left(1 + \frac{c}{n}\right)A$

and where

$$P = \sum_{k=1}^{K} \frac{c(k)}{(1+y)^{t_k}}$$

Examples

```
c=convxp(1000,1/100, 4,14,.33/2,.10);
```

The value returned is 11.6023.

COS

Returns the cosine

Category: Trigonometric

Syntax

COS (*argument*)

Arguments

argument
is numeric and is specified in radians.

Examples

SAS Statements	Results
`x=cos(0.5);`	`0.8775825619`
`x=cos(0);`	`1`
`x=cos(3.14159/3);`	`0.500000766`

COSH

Returns the hyperbolic cosine

Category: Hyperbolic

Syntax

COSH(*argument*)

Arguments

argument
is numeric.

Details

The COSH function returns the hyperbolic cosine of the argument, given by

$$\left(e^{argument} + e^{-argument}\right)/2$$

Examples

SAS Statements	Results
`x=cosh(0);`	`1`
`x=cosh(-5.0);`	`74.209948525`
`x=cosh(0.5);`	`1.1276259652`

CSS

Returns the corrected sum of squares

Category: Descriptive Statistics

Syntax

CSS(*argument*,*argument*, . . .)

Arguments

argument
is numeric. At least two arguments are required. The argument list may consist of a variable list, which is preceded by OF.

Examples

SAS Statements	Results
`x1=css(5,9,3,6);`	`18.75`
`x2=css(5,8,9,6,.);`	`0`
`x3=css(8,9,6,.);`	`4.6666666667`
`x4=css(of x1-x3);`	`101.11574074`

CUROBS

Returns the observation number of the current observation

Category: SAS File I/O

Syntax

CUROBS(*data-set-id*)

Arguments

data-set-id
specifies the data set identifier that the OPEN function returns.

Details

CAUTION:
Use this function only with an uncompressed SAS data set that is accessed using a native library engine. △

If the engine being used does not support observation numbers, the function returns a missing value.

With a SAS data view, the function returns the relative observation number, that is, the number of the observation within the SAS data view (as opposed to the number of the observation within any related SAS data set).

Examples

This example uses the FETCHOBS function to fetch the tenth observation in the data set MYDATA. The value of OBSNUM returned by CUROBS is 10.

```
%let dsid=%sysfunc(open(mydata,i));
%let rc=%sysfunc(fetchobs(&dsid,10));
%let obsnum=%sysfunc(curobs(&dsid));
```

See Also

Functions:
“FETCHOBS” on page 353
“OPEN” on page 460

CV

Returns the coefficient of variation

Category: Descriptive Statistics

Syntax

CV(*argument,argument*, . . .)

Arguments

argument
is numeric. At least two arguments are required. The argument list may consist of a variable list, which is preceded by OF.

Examples

SAS Statements	Results
`x1=cv(5,9,3,6);`	43.47826087
`x2=cv(5,8,9,6,.);`	26.082026548
`x3=cv(8,9,6,.);`	19.924242152
`x4=cv(of x1-x3);`	40.953539216

DACCDB

Returns the accumulated declining balance depreciation

Category: Financial

Syntax

DACCDB(*p*,*v*,*y*,*r*)

Arguments

p
is numeric, the period for which the calculation is to be done. For noninteger *p* arguments, the depreciation is prorated between the two consecutive time periods that precede and follow the fractional period.

v
is numeric, the depreciable initial value of the asset.

y
is numeric, the lifetime of the asset.

Range: $y > 0$

r
is numeric, the rate of depreciation expressed as a decimal.

Range: $r > 0$

Details

The DACCDB function returns the accumulated depreciation by using a declining balance method. The formula is

$$\mathrm{DACCDB}(p,v,y,r)=\begin{cases}0 & p\le 0\\ v\left(-\left(1-\frac{r}{y}\right)^{int(p)}\right)\left(1-\left(p-int\left(p\right)\right)\frac{r}{y}\right) & p>0\end{cases}$$

Note that int(*p*) is the integer part of *p*. The *p* and *y* arguments must be expressed by using the same units of time. A double-declining balance is obtained by setting *r* equal to 2.

Examples

An asset has a depreciable initial value of $1000 and a fifteen-year lifetime. Using a 200 percent declining balance, the depreciation throughout the first 10 years can be expressed as

```
a=daccdb(10,1000,15,2);
```

The value returned is 760.93. The first and the third arguments are expressed in years.

DACCDBSL

Returns the accumulated declining balance with conversion to a straight-line depreciation

Category: Financial

Syntax

DACCDBSL(*p*,*v*,*y*,*r*)

Arguments

p
is numeric, the period for which the calculation is to be done.

v
is numeric, the depreciable initial value of the asset.

y
is an integer, the lifetime of the asset.

Range: $y > 0$

r
is numeric, the rate of depreciation that is expressed as a fraction.

Range: $r > 0$

Details

The DACCDBSL function returns the accumulated depreciation by using a declining balance method, with conversion to a straight-line depreciation function that is defined by

$$\mathrm{DACCDBSL}\left(p, v, y, r\right) = \sum_{i=1}^{p} \mathrm{DEPDBSL}\left(i, v, y, r\right)$$

The declining balance with conversion to a straight-line depreciation chooses for each time period the method of depreciation (declining balance or straight-line on the remaining balance) that gives the larger depreciation. The *p* and *y* arguments must be expressed by using the same units of time.

Examples

An asset has a depreciable initial value of $1,000 and a ten-year lifetime. Using a declining balance rate of 150 percent, the accumulated depreciation of that asset in its fifth year can be expressed as

```
y5=daccdbsl(5,1000,10,1.5);
```

The value returned is 564.99. The first and the third arguments are expressed in years.

DACCSL

Returns the accumulated straight-line depreciation

Category: Financial

Syntax

DACCSL(*p,v,y*)

Arguments

p
is numeric, the period for which the calculation is to be done. For fractional *p*, the depreciation is prorated between the two consecutive time periods that precede and follow the fractional period.

v
is numeric, the depreciable initial value of the asset.

y
is numeric, the lifetime of the asset.

Range: *y* > 0

Details

The DACCSL function returns the accumulated depreciation by using the straight-line method, which is given by

$$\mathrm{DACCSL}\left(p,v,y\right)=\begin{cases}0 & p<0\\ v\left(\frac{p}{y}\right) & 0\le p\le y\\ v & p>y\end{cases}$$

The *p* and *y* arguments must be expressed by using the same units of time.

Example

An asset, acquired on 01APR86, has a depreciable initial value of $1000 and a ten-year lifetime. The accumulated depreciation in the value of the asset through 31DEC87 can be expressed as

```
a=daccsl(1.75,1000,10);
```

The value returned is 175.00. The first and the third arguments are expressed in years.

DACCSYD

Returns the accumulated sum-of-years-digits depreciation

Category: Financial

Syntax

DACCSYD(*p,v,y*)

Arguments

p
is numeric, the period for which the calculation is to be done. For noninteger *p* arguments, the depreciation is prorated between the two consecutive time periods that precede and follow the fractional period.

v
is numeric, the depreciable initial value of the asset.

y
is numeric, the lifetime of the asset.

Range: $y > 0$

Details

The DACCSYD function returns the accumulated depreciation by using the sum-of-years-digits method. The formula is

$$\mathrm{DACCSYD}\,(p,v,y) = \begin{cases} 0 & p < 0 \\ v\dfrac{int(p)\left(y-\frac{int(p)-1}{2}\right)+(p-int(p))(y-int(p))}{int(y)\left(y-\frac{int(y)-1}{2}\right)+(y-int(y))^2} & 0 \le p \le y \\ v & p > y \end{cases}$$

Note that int(*y*) indicates the integer part of *y*. The *p* and *y* arguments must be expressed by using the same units of time.

Examples

An asset, acquired on 01OCT86, has a depreciable initial value of $1,000 and a five-year lifetime. The accumulated depreciation of the asset throughout 01JAN88 can be expressed as

```
y2=daccsyd(15/12,1000,5);
```

The value returned is 400.00. The first and the third arguments are expressed in years.

DACCTAB

Returns the accumulated depreciation from specified tables

Category: Financial

Syntax

DACCTAB(*p*,*v*,*t1*, . . . ,*tn*)

Arguments

p
is numeric, the period for which the calculation is to be done. For noninteger *p* arguments, the depreciation is prorated between the two consecutive time periods that precede and follow the fractional period.

v
is numeric, the depreciable initial value of the asset.

t1,t2, . . . ,tn
are numeric, the fractions of depreciation for each time period.

Details

The DACCTAB function returns the accumulated depreciation by using user-specified tables. The formula for this function is

$$DACCTAB\left(p,v,t_1,t_2,...,t_n\right)=\begin{cases}0 & p\leq 0\\ v\left(t_1+t_2+...+t_{int(p)}+\left(p-int\left(p\right)\right)t_{int(p)+1}\right) & 0<p<n\\ v & p\geq n\end{cases}$$

For a given *p*, only the arguments $t_1, t_2, \ldots, t_k$ need to be specified with **`k=ceil(p)`**.

Examples

An asset has a depreciable initial value of $1000 and a five-year lifetime. Using a table of the annual depreciation rates of .15, .22, .21, .21, and .20 during the first, second, third, fourth, and fifth years, respectively, the accumulated depreciation throughout the third year can be expressed as

```
   y3=dacctab(3,1000,.15,.22,.21,.24,.20);
```

The value returned is 580.00. The fourth rate, .24, and the fifth rate, .20, can be omitted because they are not needed in the calculation.

DAIRY

Returns the derivative of the airy function

Category: Mathematical

Syntax

DAIRY(*x*)

Arguments

x
is numeric.

Details

The DAIRY function returns the value of the derivative of the airy function (Abramowitz and Stegun 1964; Amos, Daniel, and Weston 977).

Examples

SAS Statements	Results
x=dairy(2.0);	**-0.053090384**
x=dairy(-2.0);	**0.6182590207**

DATDIF

Returns the number of days between two dates

Category: Date and Time

Syntax

DATDIF(*sdate*,*edate*,*basis*)

Arguments

sdate
specifies a SAS date value that identifies the starting date.

edate
specifies a SAS date value that identifies the ending date.

basis
identifies a character constant or variable that describes how SAS calculates the date difference. The following character strings are valid:

'30/360'
specifies a 30 day month and a 360 day year. Each month is considered to have 30 days, and each year 360 days, regardless of the actual number of days in each month or year.

Alias: '360'

Tip: If either date falls at the end of a month, SAS treats the date as if it were the last day of a 30-day month.

'ACT/ACT'
uses the actual number of days between dates.

Alias: *'Actual'*

Examples

In the following example, DATDIF returns the actual number of days between two dates, and the number of days based on a 30-month and 360-day year.

```
data _null;
   sdate='16oct78'd;
   edate='16feb96'd;
   actual=datdif(sdate, edate, 'act/act');
   days360=datdif(sdate, edate, '30/360');
   put actual= days360=;
run;
```

SAS Statements	Results
put actual=;	**6332**
put days360=;	**6240**

See Also

Functions:
"YRDIF" on page 620

DATE

Returns the current date as a SAS date value

Category: Date and Time

Syntax

DATE()

Details

The DATE function produces the current date in the form of a SAS date value, which is the number of days since January 1, 1960.

Examples

These statements illustrate a practical use of the DATE function:

```
tday=date();
if (tday-datedue)> 15 then
   do;
      put 'As of ' tday date9. ' Account #'
           account 'is more than 15 days overdue.';
   end;
```

DATEJUL

Converts a Julian date to a SAS date value

Category: Date and Time

Syntax

DATEJUL(*julian-date*)

Arguments

julian-date

specifies a SAS numeric expression that represents a Julian date. A Julian date in SAS is a date in the form *yyddd* or *yyyyddd*, where *yy* or *yyyy* is a two- or four-digit integer that represents the year and *ddd* is the number of the day of the year. The value of *ddd* must be between and 365 (or 366 for a leap year).

Examples

The following SAS statements produce these results:

SAS Statements	Results

`Xstart=datejul(94365);`
`put Xstart / Xstart date9.;`
	
	`12783`
	`31DEC1994`

`Xend=datejul(2001001);`
`put Xend / Xend date9.;`
	
	`14976`
	`01JAN2001`

See Also

Function:
 "JULDATE" on page 416

DATEPART

Extracts the date from a SAS datetime value

Category: Date and Time

Syntax

DATEPART(*datetime*)

Arguments

datetime
 specifies a SAS expression that represents a SAS datetime value.

Examples

The following SAS statements produce this result:

SAS Statements	Results

`conn='01feb94:8:45'dt;`
`servdate=datepart(conn);`
`put servdate worddate.;`
	`February 1, 1994`

See Also

Functions:
- "DATETIME" on page 315
- "TIMEPART" on page 564

DATETIME

Returns the current date and time of day as a SAS datetime value

Category: Date and Time

Syntax

DATETIME()

Examples

This example returns a SAS value that represents the number of seconds between January 1, 1960 and the current time:

```
when=datetime();
put when=;
```

See Also

Functions:
- "DATE" on page 312
- "TIME" on page 563

DAY

Returns the day of the month from a SAS date value

Category: Date and Time

Syntax

DAY(*date*)

Arguments

date
specifies a SAS expression that represents a SAS date value.

Details

The DAY function produces an integer from 1 to 31 that represents the day of the month.

Examples

The following SAS statements produce this result:

SAS Statements	Results
`now='05may97'd;`	
`d=day(now);`	
`put d;`	`5`

See Also

Functions:
“MONTH” on page 451
“YEAR” on page 618

DCLOSE

Closes a directory that was opened by the DOPEN function and returns a value

Category: External Files

Syntax

DCLOSE(*directory-id*)

Argument

directory-id
specifies the identifier that was assigned when the directory was opened, generally by the DOPEN function.

Details

DCLOSE returns 0 if the operation was successful, ≠0 if it was not successful. The DCLOSE function closes a directory that was previously opened by the DOPEN function. DCLOSE also closes any open members.

Note: All directories or members opened within a DATA step are closed automatically when the DATA step ends. △

Examples

Example 1: Using DCLOSE to Close a Directory This example opens the directory to which the fileref MYDIR has previously been assigned, returns the number of members, and then closes the directory:

```
%macro memnum(filrf,path);
%let rc=%sysfunc(filename(filrf,&path));
%if %sysfunc(fileref(&filrf)) = 0 %then
   %do;
         /* Open the directory. */
      %let did=%sysfunc(dopen(&filrf));
      %put did=&did;
         /* Get the member count. */
      %let memcount=%sysfunc(dnum(&did));
      %put &memcount members in &filrf.;
         /* Close the directory. */
      %let rc= %sysfunc(dclose(&did));
   %end;
%else %put Invalid FILEREF;
%mend;
%memnum(MYDIR,physical-filename)
```

Example 2: Using DCLOSE within a DATA Step This example uses the DCLOSE function within a DATA step:

```
%let filrf=MYDIR;
data _null_;
  rc=filename("&filrf","physical-filename");
  if fileref("&filrf") = 0 then
     do;
           /* Open the directory. */
        did=dopen("&filrf");
           /* Get the member count. */
        memcount=dnum(did);
        put memcount "members in &filrf";
           /* Close the directory. */
        rc=dclose(did);
     end;
  else put "Invalid FILEREF";
run;
```

See Also

Functions:

"DOPEN" on page 334
"FCLOSE" on page 348
"FOPEN" on page 371
"MOPEN" on page 452

DEPDB

Returns the declining balance depreciation

Category: Financial

Syntax

DEPDB(*p*,*v*,*y*,*r*)

Arguments

p
is numeric, the period for which the calculation is to be done. For noninteger *p* arguments, the depreciation is prorated between the two consecutive time periods that precede and follow the fractional period.

v
is numeric, the depreciable initial value of the asset.

y
is numeric, the lifetime of the asset.

Range: $y > 0$

r
is numeric, the rate of depreciation that is expressed as a fraction.

Range: $r \geq 0$

Details

The DEPDB function returns the depreciation by using the declining balance method, which is given by

$$\begin{aligned}\mathrm{DEPDB}(p,v,y,r) &= \mathrm{DACCDB}(p,v,y,r)\\ &\quad - \mathrm{DACCDB}(p-1,v,y,r)\end{aligned}$$

The *p* and *y* arguments must be expressed by using the same units of time. A double-declining balance is obtained by setting *r* equal to 2.

Examples

An asset has an initial value of $1,000 and a fifteen-year lifetime. Using a declining balance rate of 200 percent, the depreciation of the value of the asset for the tenth year can be expressed as

```
y10=depdb(10,1000,15,2);
```

The value returned is 36.78. The first and the third arguments are expressed in years.

DEPDBSL

Returns the declining balance with conversion to a straight-line depreciation

Category: Financial

Syntax

DEPDBSL(*p*,*v*,*y*,*r*)

Arguments

p
is an integer, the period for which the calculation is to be done.

v
is numeric, the depreciable initial value of the asset.

y
is an integer, the lifetime of the asset.
Range: $y > 0$

r
is numeric, the rate of depreciation that is expressed as a fraction.
Range: $r \geq 0$

Details

The DEPDBSL function returns the depreciation by using the declining balance method with conversion to a straight-line depreciation, which is given by

$$\mathrm{DEPDBSL}\left(p,v,y,r\right)=\begin{cases}0 & p<0\\ v\frac{r}{y}\left(1-\frac{r}{y}\right)^{p-1} & p<t\\ v\left(-\frac{r}{y}\right)^{\frac{t-1}{y-t+1}} & p\geq t\\ 0 & p>y\end{cases}$$

where

$$t = int\left(y - \frac{y}{r} + 1\right)$$

and *int*( )denotes the integer part of a numeric argument.

The *p* and *y* arguments must be expressed by using the same units of time. The declining balance that changes to a straight-line depreciation chooses for each time period the method of depreciation (declining balance or straight-line on the remaining balance) that gives the larger depreciation.

Examples

An asset has a depreciable initial value of $1,000 and a ten-year lifetime. Using a declining balance rate of 150 percent, the depreciation of the value of the asset in the fifth year can be expressed as

```
y5=depdbsl(5,1000,10,1.5);
```

The value returned is 87.00. The first and the third arguments are expressed in years.

DEPSL

Returns the straight-line depreciation

Category: Financial

Syntax

DEPSL(*p*,*v*,*y*)

Arguments

p
is numeric, the period for which the calculation is to be done. For fractional *p*, the depreciation is prorated between the two consecutive time periods that precede and follow the fractional period.

v
is numeric, the depreciable initial value of the asset.

y
is numeric, the lifetime of the asset.

Range: $y > 0$

Details

The DEPSL function returns the straight-line depreciation, which is given by

$$\mathrm{DEPSY}\,(p,v,y) = \mathrm{DACCSL}\,(p,v,y) - \mathrm{DACCSL}\,(p-1,v,y)$$

The *p* and *y* arguments must be expressed by using the same units of time.

Examples

An asset, acquired on 01APR86, has a depreciable initial value of $1,000 and a ten-year lifetime. The depreciation in the value of the asset for the year 1986 can be expressed as

```
d=depsl(9/12,1000,10);
```

The value returned is 75.00. The first and the third arguments are expressed in years.

DEPSYD

Returns the sum-of-years-digits depreciation

Category: Financial

Syntax

DEPSYD(*p,v,y*)

Arguments

p
is numeric, the period for which the calculation is to be done. For noninteger *p* arguments, the depreciation is prorated between the two consecutive time periods that precede and follow the fractional period.

v
is numeric, the depreciable initial value of the asset.

y
is numeric, the lifetime of the asset in number of depreciation periods.

Range: $y > 0$

Details

The DEPSYD function returns the sum-of-years-digits depreciation, which is given by

$$\mathrm{DEPSYD}\,(p,v,y) = \mathrm{DACCSYD}\,(p,v,y) - \mathrm{DACCSYD}\,(p-1,v,y)$$

The *p* and *y* arguments must be expressed by using the same units of time.

Examples

An asset, acquired on 01OCT86, has a depreciable initial value of $1,000 and a five-year lifetime. The depreciations in the value of the asset for the years 1986 and 1987 can be expressed as

```
   y1=depsyd(3/12,1000,5);
   y2=depsyd(15/12,1000,5);
```

The values returned are 83.33 and 316.67, respectively. The first and the third arguments are expressed in years.

DEPTAB

Returns the depreciation from specified tables

Category: Financial

Syntax

DEPTAB(*p*,*v*,*t1*, . . . ,*tn*)

Arguments

p
is numeric, the period for which the calculation is to be done. For noninteger *p* arguments, the depreciation is prorated between the two consecutive time periods that precede and follow the fractional period.

v
is numeric, the depreciable initial value of the asset.

t1,t2, . . . ,tn
are numeric, the fractions of depreciation for each time period.

Details

The DEPTAB function returns the depreciation by using specified tables. The formula is

$$\begin{aligned} DEPTAB\left(p, v, t_1, t_2, ..., t_n\right) &= DACCTAB\left(p, v, t_1, t_2, ..., t_n\right) \\ &\quad - DACCTAB\left(p-1, v, t_1, t_2, ..., t_n\right) \end{aligned}$$

For a given *p*, only the arguments $t_1, t_2, \ldots, t_k$ need to be specified with **`k=ceil(p)`**.

Examples

An asset has a depreciable initial value of $1,000 and a five-year lifetime. Using a table of the annual depreciation rates of .15, .22, .21, .21, and .21 during the first, second, third, fourth, and fifth years, respectively, the depreciation in the third year can be expressed as

```
    y3=deptab(3,1000,.15,.22,.21,.21,.21);
```

The value returned is 210.00.

DEQUOTE

Removes quotation marks from a character value

Category: Character

Syntax

DEQUOTE(*argument*)

Arguments

argument
specifies a character value.

Details

The DEQUOTE function removes single or double quotation marks from a character value. SAS also reduces any multiple quotation marks within the *character-expression*.

Examples

SAS Statements	Results

x="A'B";
y=dequote(x);
put y;
	A'B

x="A""B";
y=dequote(x);
put y;
	A"B

x='A''B';
y=dequote(x);
put y;
	A'B

DEVIANCE

Computes the deviance and returns a value

Category: Mathematical

Syntax

DEVIANCE(*distribution, variable, shape-parameter(s)*<,*ε*>)

Arguments

distribution

is a character string that identifies the distribution. Valid distributions are

Distribution	Argument
Bernoulli	**'BERNOULLI' | 'BERN'**
Binomial	**'BINOMIAL' | 'BINO'**
Gamma	**'GAMMA'**
Inverse Gauss (Wald)	**'IGAUSS' | 'WALD'**
Normal	**'NORMAL' | 'GAUSSIAN'**
Poisson	**'POISSON' | 'POIS'**

variable

is a numeric random variable.

shape-parameter(s)

are one or more distribution-specific numeric parameters that characterize the shape of the distribution.

ε

is an optional numeric small value used for all of the distributions, except for the normal distribution.

Details

The Bernoulli Distribution

DEVIANCE('BERNOULLI', *variable, p*<, *ε*>)

where

variable

is a binary numeric random variable that has the value of 1 for success and 0 for failure.

p

is a numeric probability of success with $\varepsilon \le p \le 1-\varepsilon$.

ε

is an optional positive numeric value that is used to bound *p*. Any value of *p* in the interval $0 \le p \le \varepsilon$ is replaced by *ε*. Any value of *p* in the interval $1 - \varepsilon \le p \le 1$ is replaced by $1 - \varepsilon$.

The DEVIANCE function returns the deviance from a Bernoulli distribution with a probability of success *p*, where success is defined as a random variable value of 1. The equation follows:

$$\mathrm{DEVIANCE}\left('BERN', variable, p, \epsilon\right) = \begin{cases} -2\log\left(1-p\right) & x = 0 \\ -2\log\left(p\right) & x = 1 \\ . & otherwise \end{cases}$$

The Binomial Distribution

DEVIANCE('BINO', *variable,* μ*, n*<, ε>)

where

variable
: is a numeric random variable that contains the number of successes.

  Range: 0 ≤ *variable* ≤ 1

μ
: is a numeric mean parameter.

  Range: $n\varepsilon \le \mu \le n(1-\varepsilon)$

n
: is an integer number of Bernoulli trials parameter

  Range: $n \ge 0$

ε
: is an optional positive numeric value that is used to bound μ. Any value of μ in the interval $0 \le \mu \le n\varepsilon$ is replaced by $n\varepsilon$. Any value of μ in the interval $n(1-\varepsilon) \le \mu \le n$ is replaced by $n(1-\varepsilon)$.

The DEVIANCE function returns the deviance from a binomial distribution, with a probability of success *p*, and a number of independent Bernoulli trials *n*. The following equation describes the DEVIANCE function for the Binomial distribution, where *x* is the random variable.

$$\mathrm{DEVIANCE}\left('BINO', x, \mu, n\right) = \begin{cases} . & x < 0 \\ 2\left(x\log\left(\frac{x}{\mu}\right) + (n-x)\log\left(\frac{n-x}{n-\mu}\right)\right) & 0 \le x \le n \\ . & x > n \end{cases}$$

The Gamma Distribution

DEVIANCE('GAMMA', *variable,* μ <, ε>)

where

variable
: is a numeric random variable.

  Range: *variable* ≥ ε

μ
: is a numeric mean parameter.

  Range: $\mu \ge \varepsilon$

ε
: is an optional positive numeric value that is used to bound *variable* and μ. Any value of *variable* in the interval 0 ≤ *variable* ≤ ε is replaced by ε. Any value of μ in the interval $0 \le \mu \le \varepsilon$ is replaced by ε.

The DEVIANCE function returns the deviance from a gamma distribution with a mean parameter μ. The following equation describes the DEVIANCE function for the gamma distribution, where *x* is the random variable:

$$\text{DEVIANCE}\left('GAMMA', x, \mu\right) = \begin{cases} . & x < 0 \\ 2\left(-\log\left(\frac{x}{\mu}\right) + \frac{x-\mu}{\mu}\right) & x \geq \epsilon, \mu \geq \epsilon \end{cases}$$

The Inverse Gauss (Wald) Distribution

DEVIANCE('IGAUSS' | 'WALD', *variable*, μ<, ε>)

where

variable
is a numeric random variable.
Range: *variable* ≥ ε

μ
is a numeric mean parameter.
Range: $\mu \geq \varepsilon$

ε
is an optional positive numeric value that is used to bound *variable* and μ. Any value of *variable* in the interval 0 ≤ *variable* ≤ ε is replaced by ε. Any value of μ in the interval $0 \leq \mu \leq \varepsilon$ is replaced by ε.

The DEVIANCE function returns the deviance from an inverse Gaussian distribution with a mean parameter μ. The following equation describes the DEVIANCE function for the inverse Gaussian distribution, where *x* is the random variable:

$$\text{DEVIANCE}\left('IGAUSS', x, \mu\right) = \begin{cases} . & x < 0 \\ \frac{(x-\mu)^2}{\mu^2 x} & x \geq \epsilon, \mu \geq \epsilon \end{cases}$$

The Normal Distribution

DEVIANCE('NORMAL' | 'GAUSSIAN', *variable*, μ)

where

variable
is a numeric random variable.

μ
is a numeric mean parameter.

The DEVIANCE function returns the deviance from a normal distribution with a mean parameter μ. The following equation describes the DEVIANCE function for the normal distribution, where *x* is the random variable:

$$\text{DEVIANCE}\left('NORMAL', x, \mu\right) = (x - \mu)^2$$

The Poisson Distribution

DEVIANCE('POISSON', *variable*, μ<, ε>)

where

variable
is a numeric random variable.

Range: *variable* ≥ 0

μ
: is a numeric mean parameter.

Range: $\mu \geq \varepsilon$

ε
: is an optional positive numeric value that is used to bound μ. Any value of μ in the interval $0 \leq \mu \leq \varepsilon$ is replaced by ε.

The DEVIANCE function returns the deviance from a Poisson distribution with a mean parameter μ. The following equation describes the DEVIANCE function for the Poisson distribution, where *x* is the random variable:

$$\mathrm{DEVIANCE}\left('POISSON', x, \mu\right) = \begin{cases} \cdot & x < 0 \\ 2\left(x \log\left(\frac{x}{\mu}\right) - (x - \mu)\right) & x \geq 0, \mu \geq \epsilon \end{cases}$$

DHMS

Returns a SAS datetime value from date, hour, minute, and second

Category: Date and Time

Syntax

DHMS(*date,hour,minute,second*)

Arguments

date
: specifies a SAS expression that represents a SAS date value.

hour
: specifies a SAS expression that represents an integer from 0 through 23.

minute
: specifies a SAS expression that represents an integer from 0 through 59.

second
: specifies a SAS expression that represents an integer from 0 through 59.

Examples

The following SAS statements produce these results:

SAS Statements	Results
`dtid=dhms('01jan89'd,15,30,15);`	
`put dtid / dtid datetime.;`	`915291015`
	`01JAN89:15:30:15`

The following SAS statements show how to combine a SAS date value with a SAS time value into a SAS datetime value. If you execute these statements on August 13, 1997 at the time of 15:55:50, it produces these results:

SAS Statements	Result
`day=date();`	
`time=time();`	`13AUG97:15:55:50`
`sasdt=dhms(day,0,0,time);`	
`put sasdt datetime.;`	

See Also

Function:
“HMS” on page 394

DIF

Returns differences between the argument and its *n*th lag

Category: Special

Syntax

DIF<*n*>(*argument*)

Arguments

n
specifies the number of lags.

argument
is numeric.

Details

The DIF functions, DIF1, DIF2, . . . , DIF100, return the first differences between the argument and its *n*th lag. DIF1 can also be written as DIF. DIF*n* is defined as DIFN(*x*)=*x*-LAGN(*x*) .

For details on storing and returning values from the LAG*n* queue, see the LAG function.

Comparisons

The function DIF2(X) is not equivalent to the second difference DIF(DIF(X)).

Examples

This example demonstrates the difference between the LAG and DIF functions.

```
data two;
   input X @@;
   Z=lag(x);
   D=dif(x);
   datalines;
 2 6 4 7
;
proc print data=two;
run;
```

Results of the PROC PRINT step follow:

```
OBS         X             Z                D
 1          1             .                .
 2          2             1                1
 3          6             2                4
 4          4             6              - 2
 5          7             4                3
```

See Also

Function:
  "LAG" on page 432

DIGAMMA

Returns the value of the DIGAMMA function

Category: Mathematical

Syntax

DIGAMMA(*argument*)

Arguments

argument
is numeric.

Restriction: Nonpositive integers are invalid.

Details

The DIGAMMA function returns the ratio that is given by

$$\Psi\left(x\right) = \Gamma'\left(x\right)/\Gamma\left(x\right)$$

where $\Gamma(.)$ and $\Gamma'(.)$ denote the Gamma function and its derivative, respectively. For *argument*>0, the DIGAMMA function is the derivative of the LGAMMA function.

Example

SAS Statements	Results
x=digamma(1.0);	**-0.577215665**

DIM

Returns the number of elements in an array

Category: Array

Syntax

DIM<*n*>(*array-name*)

DIM(*array-name*,*bound-n*)

Arguments

n
specifies the dimension, in a multidimensional array, for which you want to know the number of elements. If no *n* value is specified, the DIM function returns the number of elements in the first dimension of the array.

array-name
specifies the name of an array that was previously defined in the same DATA step.

bound-n
specifies the dimension, in a multidimensional array, for which you want to know the number of elements. Use *bound-n* only when *n* is not specified.

Details

The DIM function returns the number of elements in a one-dimensional array or the number of elements in a specified dimension of a multidimensional array when the lower bound of the dimension is 1. Use DIM in array processing to avoid changing the upper bound of an iterative DO group each time you change the number of array elements.

Comparisons

- DIM always returns a total count of the number of elements in an array dimension.
- HBOUND returns the literal value of the upper bound of an array dimension.

Note: This distinction is important when the lower bound of an array dimension has a value other than 1 and the upper bound has a value other than the total number of elements in the array dimension. △

Examples

Example 1: One-dimensional Array In this example, DIM returns a value of 5. Therefore, SAS repeats the statements in the DO loop five times.

```
array big{5} weight sex height state city;
do i=1 to dim(big);
  more SAS statements;
end;
```

Example 2: Multidimensional Array This example shows two ways of specifying the DIM function for multidimensional arrays. Both methods return the same value for DIM, as shown in the table that follows the SAS code example.

```
array mult{5,10,2} mult1-mult100;
```

Syntax	Alternative Syntax	Value
DIM(MULT)	DIM(MULT,1)	5
DIM2(MULT)	DIM(MULT,2)	10
DIM3(MULT)	DIM(MULT,3)	2

See Also

Functions:
- "HBOUND" on page 392
- "LBOUND" on page 434

Statements:
- "ARRAY" on page 755
- "Array Reference" on page 759

"Array Processing" in *SAS Language Reference: Concepts*

DINFO

Returns information about a directory

Category: External Files

Syntax

DINFO(*directory-id*,*info-item*)

Arguments

directory-id
specifies the identifier that was assigned when the directory was opened, generally by the DOPEN function.

info-item
specifies the information item to be retrieved. DINFO returns a blank if the value of the *info-item* argument is invalid. The information available varies according to the operating environment. This is a character value.

Details

Use DOPTNAME to determine the names of the available system-dependent directory information items. Use DOPTNUM to determine the number of directory information items available.

Operating Environment Information: DINFO returns the value of a system-dependent directory parameter. See the SAS documentation for your operating environment for information about system-dependent directory parameters. △

Examples

Example 1: Using DINFO to Return Information about a Directory This example opens the directory MYDIR, determines the number of directory information items available, and retrieves the value of the last one:

```
%let filrf=MYDIR;
%let rc=%sysfunc(filename(filrf,physical-name));
%let did=%sysfunc(dopen(&filrf));
%let numopts=%sysfunc(doptnum(&did));
%let foption=%sysfunc(doptname(&did,&numopts));
%let charval=%sysfunc(dinfo(&did,&foption));
%let rc=%sysfunc(dclose(&did));
```

Example 2: Using DINFO within a DATA Step This example creates a data set that contains the name and value of each directory information item:

```
data diropts;
   length foption $ 12 charval $ 40;
   keep foption charval;
   rc=filename("mydir","physical-name");
   did=dopen("mydir");
   numopts=doptnum(did);
   do i=1 to numopts;
      foption=doptname(did,i);
      charval=dinfo(did,foption);
      output;
   end;
run;
```

See Also

Functions:

“DOPEN” on page 334
“DOPTNAME” on page 335
“DOPTNUM” on page 336
“FINFO” on page 361
“FOPTNAME” on page 373
“FOPTNUM” on page 374

DNUM

Returns the number of members in a directory

Category: External Files

Syntax

DNUM(*directory-id*)

Argument

directory-id
specifies the identifier that was assigned when the directory was opened, generally by the DOPEN function.

Details

You can use DNUM to determine the highest possible member number that can be passed to DREAD.

Examples

Example 1: Using DNUM to Return the Number of Members This example opens the directory MYDIR, determines the number of members, and closes the directory:

```
%let filrf=MYDIR;
%let rc=%sysfunc(filename(filrf,physical-name));
%let did=%sysfunc(dopen(&filrf));
%let memcount=%sysfunc(dnum(&did));
%let rc=%sysfunc(dclose(&did));
```

Example 2: Using DNUM within a DATA Step This example creates a DATA step that returns the number of members in a directory called MYDIR:

```
data _null_;
   rc=filename("mydir","physical-name");
   did=dopen("mydir");
```

```
      memcount=dnum(did);
      rc=dclose(did);
   run;
```

See Also

Functions:
- “DOPEN” on page 334
- “DREAD” on page 337

DOPEN

Opens a directory and returns a directory identifier value

Category: External Files

Syntax

DOPEN(*fileref*)

Argument

fileref
specifies the fileref assigned to the directory.

Restriction: You must associate a fileref with the directory before calling DOPEN.

Details

DOPEN opens a directory and returns a directory identifier value (a number greater than 0) that is used to identify the open directory in other SAS external file access functions. If the directory could not be opened, DOPEN returns 0. The directory to be opened must be identified by a fileref. You can assign filerefs using the FILENAME statement or the FILENAME external file access function. Under some operating environments, you can also assign filerefs using system commands.

Operating Environment Information: The term *directory* used in the description of this function and related SAS external file access functions refers to an aggregate grouping of files managed by the operating environment. Different operating environments identify such groupings with different names, such as directory, subdirectory, MACLIB, or partitioned data set. For details, see the SAS documentation for your operating environment. △

Examples

Example 1: Using DOPEN to Open a Directory This example assigns the fileref MYDIR to a directory. It uses DOPEN to open the directory. DOPTNUM determines the number of system-dependent directory information items available, and DCLOSE closes the directory:

```
%let filrf=MYDIR;
%let rc=%sysfunc(filename(filrf,physical-name));
%let did=%sysfunc(dopen(&filrf));
%let infocnt=%sysfunc(doptnum(&did));
%let rc=%sysfunc(dclose(&did));
```

Example 2: Using DOPEN within a DATA Step This example opens a directory for processing within a DATA step.

```
data _null_;
   drop rc did;
   rc=filename("mydir","physical-name");
   did=dopen("mydir");
   if did > 0 then do;
      ...more statements...
   end;
run;
```

See Also

Functions:
- "DCLOSE" on page 316
- "DOPTNUM" on page 336
- "FOPEN" on page 371
- "MOPEN" on page 452

DOPTNAME

Returns directory attribute information

Category: External Files

Syntax

DOPTNAME(*directory-id*,*nval*)

Arguments

directory-id
specifies the identifier that was assigned when the directory was opened, generally by the DOPEN function.

Restriction: To use DOPTNAME on a directory, the directory must have been previously opened by using DOPEN.

nval
specifies the sequence number of the option.

Details

Operating Environment Information: The number, names, and nature of the directory information varies between operating environments. The number of options that are available for a directory varies depending on the operating environment. For details, see the SAS documentation for your operating environment. △

Examples

Example 1: Using DOPTNAME to Retrieve Directory Attribute Information This example opens the directory with the fileref MYDIR, retrieves all system-dependent directory information items, writes them to the SAS log, and closes the directory:

```
%let filrf=mydir;
%let rc=%sysfunc(filename(filrf,physical-name));
%let did=%sysfunc(dopen(&filrf));
%let infocnt=%sysfunc(doptnum(&did));
%do j=1 %to &infocnt;
   %let opt=%sysfunc(doptname(&did,&j));
   %put Directory information=&opt;
%end;
%let rc=%sysfunc(dclose(&did));
```

Example 2: Using DOPTNAME within a DATA Step This example creates a data set that contains the name and value of each directory information item:

```
data diropts;
   length optname $ 12 optval $ 40;
   keep optname optval;
   rc=filename("mydir","physical-name");
   did=dopen("mydir");
   numopts=doptnum(did);
   do i=1 to numopts;
      optname=doptname(did,i);
      optval=dinfo(did,optname);
      output;
   end;
   run;
```

See Also

Functions:
- "DINFO" on page 331
- "DOPEN" on page 334
- "DOPTNUM" on page 336

DOPTNUM

Returns the number of information items that are available for a directory

Category: External Files

Syntax

DOPTNUM(*directory-id*)

Argument

directory-id

specifies the identifier that was assigned when the directory was opened, generally by the DOPEN function.

Restriction: The directory must have been previously opened by using DOPEN.

Details

Operating Environment Information: The number, names, and nature of the directory information varies between operating environments. The number of options that are available for a directory varies depending on the operating environment. For details, see the SAS documentation for your operating environment. △

Examples

Example 1: Retrieving the Number of Information Items This example retrieves the number of system-dependent directory information items that are available for the directory MYDIR and closes the directory:

```
%let filrf=mydir;
%let rc=%sysfunc(filename(filrf,physical-name));
%let did=%sysfunc(dopen(&filrf));
%let infocnt=%sysfunc(doptnum(&did));
%let rc=%sysfunc(dclose(&did));
```

Example 2: Using DOPTNUM within a DATA Step This example creates a data set that retrieves the number of system-dependent information items that are available for the MYDIR directory:

```
data _null_;
   rc=filename("mydir","physical-name");
   did=dopen("mydir");
   infocnt=doptnum(did);
   rc=dclose(did);
run;
```

See Also

Functions:

“DINFO” on page 331

“DOPEN” on page 334

“DOPTNAME” on page 335

DREAD

Returns the name of a directory member

Category: External Files

Syntax

DREAD(*directory-id*,*nval*)

Arguments

directory-id
specifies the identifier that was assigned when the directory was opened, generally by the DOPEN function.

Restriction: The directory must have been previously opened by using DOPEN.

nval
specifies the sequence number of the member within the directory.

Details

DREAD returns a blank if an error occurs (such as when *nval* is out-of-range). Use DNUM to determine the highest possible member number that can be passed to DREAD.

Examples

This example opens the directory identified by the fileref MYDIR, retrieves the number of members, and places the number in the variable MEMCOUNT. It then retrieves the name of the last member, places the name in the variable LSTNAME , and closes the directory:

```
%let filrf=mydir;
%let rc=%sysfunc(filename(filrf,physical-name));
%let did=%sysfunc(dopen(&filrf));
%let lstname=;
%let memcount=%sysfunc(dnum(&did));
%if &memcount > 0 %then
   %let lstname=%sysfunc(dread(&did,&memcount));
%let rc=%sysfunc(dclose(&did));
```

See Also

Functions:

“DNUM” on page 333

“DOPEN” on page 334

DROPNOTE

Deletes a note marker from a SAS data set or an external file and returns a value

Category: SAS File I/O

Category: External Files

Syntax

DROPNOTE(*data-set-id* | *file-id*,*note-id*)

Arguments

data-set-id* | *file-id
specifies the identifier that was assigned when the data set or external file was opened, generally by the OPEN function or the FOPEN function.

note-id
specifies the identifier that was assigned by the NOTE or FNOTE function.

Details

DROPNOTE deletes a marker set by NOTE or FNOTE. It returns a 0 if successful and ≠0 if not successful.

Examples

This example opens the SAS data set MYDATA, fetches the first observation, and sets a note ID at the beginning of the data set. It uses POINT to return to the first observation, and then uses DROPNOTE to delete the note ID:

```
%let dsid=%sysfunc(open(mydata,i));
%let rc=%sysfunc(fetch(&dsid));
%let noteid=%sysfunc(note(&dsid));
   more macro statements
%let rc=%sysfunc(point(&dsid,&noteid));
%let rc=%sysfunc(fetch(&dsid));
%let rc=%sysfunc(dropnote(&dsid,&noteid));
```

See Also

Functions:
- "FETCH" on page 352
- "FNOTE" on page 369
- "FOPEN" on page 371
- "FPOINT" on page 375
- "NOTE" on page 458
- "OPEN" on page 460
- "POINT" on page 481

DSNAME

Returns the SAS data set name that is associated with a data set identifier

Category: SAS File I/O

Syntax

DSNAME(*data-set-id*)

Arguments

data-set-id
specifies the data set identifier that the OPEN function returns.

Details

DSNAME returns the data set name that is associated with a data set identifier, or a blank if the data set identifier is not valid.

Examples

This example determines the name of the SAS data set that is associated with the variable DSID and displays the name in the SAS log.

```
%let dsid=%sysfunc(open(sasuser.houses,i));
%put The current open data set
is %sysfunc(dsname(&dsid)).;
```

See Also

Function:
“OPEN” on page 460

DUR

Returns the modified duration for an enumerated cashflow

Category: Financial

Syntax

DUR(*y,f,c(1), ... ,c(k)*)

Arguments

y
the effective per-period yield-to-maturity, expressed as a fraction.

Range: $y > 0$

f
the frequency of cashflows per period.

Range: $f > 0$

c(1), ... ,c(k)
a list of cashflows.

Details

The DUR function returns the value

$$D = \frac{\sum_{k=1}^{K} \frac{k}{f} \frac{c(k)}{(1+fy)^{\frac{k}{f}}}}{P(1+y)}$$

where

$$P = \sum_{k=1}^{K} \frac{c(k)}{(1+fy)^{\frac{k}{f}}}$$

Examples

```
d=dur(1/20,1,.33,.44,.55,.49,.50,.22,.4,.8,.01,.36,.2,.4);
```

The value returned is 5.28402.

DURP

Returns the modified duration for a periodic cashflow stream, such as a bond

Category: Financial

Syntax

DURP(*A,c,n,K,k₀,y*)

Arguments

A
the par value.

Range: $A > 0$

c
the nominal per-period coupon rate, expressed as a fraction.

Range: $0 < c < 1$

n

the number of coupons per period.

Range: $n > 0$ and is an integer

K

the number of remaining coupons.

Range: $K > 0$ and is an integer

k_0

the time from present to the first coupon date, expressed in terms of the number of periods.

Range: $0 < k_0 < 1/n$

y

the nomimal per-period yield-to-maturity, expressed as a fraction.

Range: $y > 0$

Details

The DURP function returns the value

$$D = \frac{1}{n}\frac{\sum_{k=1}^{K} t_k \frac{c(k)}{\left(1+\frac{y}{n}\right)^{t_k}}}{P\left(1+\frac{y}{n}\right)}$$

where

$t_k = k - (1 - nk_0)$

$c(k) = \frac{c}{n}A$ for k=1, ..., K-1

$c(K) = \left(1 + \frac{c}{n}\right)A$

and where

$$P = \sum_{k=1}^{K} \frac{c(k)}{\left(1+\frac{y}{n}\right)^{t_k}}$$

Examples

```
d=durp(1000,1/100,4,14,.33/2,.10);
```

The value returned is 3.26496.

ERF

Returns the value of the (normal) error function

Category: Mathematical

Syntax

ERF(*argument*)

Arguments

argument
is numeric.

Details

The ERF function returns the integral, given by

$$\mathrm{ERF}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-z^2} dz$$

Examples

You can use the ERF to find the probability (p) that a normally distributed random variable with mean 0 and standard deviation will take on a value less than X. For example, the quantity that is given by the following statement is equivalent to PROBNORM(X):

```
p=.5+.5*erf(x/sqrt(2));
```

SAS Statements	Results
`y=erf(1.0);`	**0.8427007929**
`y=erf(-1.0);`	**-0.842700793**

ERFC

Returns the value of the complementary (normal) error function

Category: Mathematical

Syntax

ERFC(*argument*)

Arguments

argument
is numeric.

Details

The ERFC function returns the complement to the ERF function (that is, 1 – ERF(*argument*)).

Examples

SAS Statements	Results
`x=erfc(1.0);`	`0.1572992071`
`x=erfc(-1.0);`	`.8427007929`

EXIST

Verifies the existence of a SAS data library member

Category: SAS File I/O

Syntax

EXIST(*member-name*<,*member-type*>)

Arguments

member-name
specifies the SAS data library member. If *member-name* is blank or a null string, EXIST uses the member specified by the system variable _LAST_ .

member-type
specifies the type of SAS data library member:

ACCESS	an access descriptor created using SAS/ACCESS software.
CATALOG	a SAS catalog or catalog entry.
DATA	a SAS data file (default).
VIEW	a SAS data view.

Details

EXIST returns 1 if the library member exists, or 0 if *member-name* does not exist or *member-type* is invalid. Use CEXIST to verify the existence of an entry in a catalog.

Examples

- This example verifies the existence of a data set. If the data set does not exist, the example displays a message in the log.

```
%let dsname=sasuser.houses;
%if %sysfunc(exist(&dsname)) %then
   %let dsid=%sysfunc(open(&dsname,i));
%else %put Data set &dsname does not exist.;
```

- This example verifies the existence of the SAS data view TEST.MYVIEW. If the view does not exist, the example displays a message in the log.

```
data _null_;
dsname="test.myview";
   if (exist(dsname,"VIEW")) then
      dsid=open(dsname,"i");
   else put dsname 'does not exist.';
run;
```

See Also

Functions:
- "CEXIST" on page 287
- "FEXIST" on page 354
- "FILEEXIST" on page 357

EXP

Returns the value of the exponential function

Category: Mathematical

Syntax

EXP(*argument*)

Arguments

argument
is numeric.

Details

The EXP function raises the constant *e*, which is approximately given by 2.71828, to the power that is supplied by the argument. The result is limited by the maximum value of a floating-point decimal value on the computer.

Examples

SAS Statements	Results
`x=exp(1.0);`	`2.7182818285`
`x=exp(0);`	`1`

FACT

Computes a factorial and returns a value

Category: Mathematical

Syntax

FACT(*n*)

Arguments

n
is an integer that represents the number of elements for which the factorial is computed.

Details

The mathematical representation of the FACT function is given by the following equation:

$$FACT(n) = n!$$

with $n \geq 0$.

If the expression cannot be computed, a missing value is returned.

Examples

SAS Statements	Result
`x=fact(5);`	120

See Also

Functions:
- "COMB" on page 293
- "PERM" on page 480

FAPPEND

Appends the current record to the end of an external file and returns a value

Category: External Files

Syntax

FAPPEND(*file-id*<,*cc*>)

Arguments

file-id
specifies the identifier that was assigned when the file was opened, generally by the FOPEN function.

cc
specifies a carriage control character:

blank	indicates that the record starts a new line.
0	skips one blank line before this new line.
-	skips two blank lines before this new line.
1	specifies that the line starts a new page.
+	specifies that the line overstrikes a previous line.
P	specifies that the line is a terminal prompt.
=	specifies that the line contains carriage control information.
all else	specifies that the line record starts a new line.

Details

FAPPEND adds the record that is currently contained in the File Data Buffer (FDB) to the end of an external file. FAPPEND returns a 0 if the operation was successful and ≠0 if it was not successful.

Examples

This example assigns the fileref MYFILE to an external file and attempts to open the file. If the file is opened successfully, it moves data into the File Data Buffer, appends a record, and then closes the file. Note that in a macro statement you do not enclose character strings in quotation marks.

```
%let filrf=myfile;
%let rc=%sysfunc(filename(filrf,
   physical-filename));
%let fid=%sysfunc(fopen(&filrf,a));
%if &fid > 0 %then
   %do;
      %let rc=%sysfunc(fput(&fid,
                       Data for the new record));
      %let rc=%sysfunc(fappend(&fid));
      %let rc=%sysfunc(fclose(&fid));
   %end;
%else
   %do;
      /* unsuccessful open processing */
   %end;
```

See Also

Functions:
- "DOPEN" on page 334
- "FCLOSE" on page 348
- "FGET" on page 355
- "FOPEN" on page 371
- "FPUT" on page 378
- "FWRITE" on page 385
- "MOPEN" on page 452

FCLOSE

Closes an external file, directory, or directory member, and returns a value

Category: External Files

Syntax

FCLOSE(*file-id*)

Argument

file-id
specifies the identifier that was assigned when the file was opened, generally by the FOPEN function.

Details

FCLOSE returns a 0 if the operation was successful and ≠0 if it was not successful. If you open a file within a DATA step, it is closed automatically when the DATA step ends.

Operating Environment Information: On some operating environments you must close the file with the FCLOSE function at the end of the DATA step. For details, see the SAS documentation for your operating environment. △

Examples

This example assigns the fileref MYFILE to an external file, and attempts to open the file. If the file is opened successfully, indicated by a positive value in the variable FID, the program reads the first record, closes the file, and deassigns the fileref:

```
%let filrf=myfile;
%let rc=%sysfunc(filename(filrf,
   physical-filename));
%let fid=%sysfunc(fopen(&filrf));
%if &fid > 0 %then
   %do;
      %let rc=%sysfunc(fread(&fid));
      %let rc=%sysfunc(fclose(&fid));
   %end;
%else
   %do;
      %put %sysfunc(sysmsg());
%end;
%let rc=%sysfunc(filename(filrf));
```

See Also

Functions:
- "DCLOSE" on page 316
- "DOPEN" on page 334
- "FOPEN" on page 371
- "FREAD" on page 379
- "MOPEN" on page 452

FCOL

Returns the current column position in the File Data Buffer (FDB)

Category: External Files

Syntax

FCOL(*file-id*)

Argument

file-id
specifies the identifier that was assigned when the file was opened, generally by the FOPEN function.

Details

Use FCOL combined with FPOS to manipulate data in the File Data Buffer (FDB).

Examples

This example assigns the fileref MYFILE to an external file and attempts to open the file. If the file is successfully opened, indicated by a positive value in the variable FID, it puts more data into the FDB relative to position POS, writes the record, and closes the file:

```
%let filrf=myfile;
%let rc=%sysfunc(filename(filrf,
   physical-filename));
%let fid=%sysfunc(fopen(&filrf,o));
%if (&fid > 0) %then
   %do;
      %let record=This is data for the record.;
      %let rc=%sysfunc(fput(&fid,&record));
      %let pos=%sysfunc(fcol(&fid));
      %let rc=%sysfunc(fpos(&fid,%eval(&pos+1)));
      %let rc=%sysfunc(fput(&fid,more data));
      %let rc=%sysfunc(fwrite(&fid));
      %let rc=%sysfunc(fclose(&fid));
   %end;
%let rc=%sysfunc(filename(filrf));
```

The new record written to the external file is

```
This is data for the record. more data
```

See Also

Functions:
- "FCLOSE" on page 348
- "FOPEN" on page 371
- "FPOS" on page 377
- "FPUT" on page 378
- "FWRITE" on page 385
- "MOPEN" on page 452

FDELETE

Deletes an external file or an empty directory

Category: External Files

Syntax

FDELETE(*fileref* | *directory*)

Argument

fileref

specifies the fileref that you assigned to the external file. You can assign filerefs by using the FILENAME statement or the FILENAME external file access function.

Restriction: The fileref that you use with FDELETE can not be a concatenation.

Operating Environment Information: In some operating environments, you can specify a fileref that was assigned with an environment variable. You can also assign filerefs using system commands. For details, see the SAS documentation for your operating environment. △

directory

specifies an empty directory that you want to delete.

Restriction: You must have authorization to delete the directory.

Details

FDELETE returns 0 if the operation was successful or ≠0 if it was not successful.

Examples

Example 1: Deleting an External File This example generates a fileref for an external file in the variable FNAME. Then it calls FDELETE to delete the file and calls the FILENAME function again to deassign the fileref.

```
data _null_;
    fname="tempfile";
    rc=filename(fname,"physical-filename");
    if rc = 0 and fexist(fname) then
       rc=fdelete(fname);
    rc=filename(fname);
run;
```

Example 2: Deleting a Directory This example uses FDELETE to delete an empty directory to which you have write access. If the directory is not empty, the optional SYSMSG function returns an error message stating that SAS is unable to delete the file.

```
filename testdir 'physical-filename';
data _null_;
   rc=fdelete('testdir');
   put rc=;
   msg=sysmsg();
   put msg=;
run;
```

See Also

Functions:
  "FEXIST" on page 354
  "FILENAME" on page 358
Statement:
  "FILENAME" on page 821

FETCH

Reads the next nondeleted observation from a SAS data set into the Data Set Data Vector (DDV) and returns a value

Category: SAS File I/O

Syntax

FETCH(*data-set-id* <,'NOSET'>)

Arguments

data-set-id
specifies the data set identifier that the OPEN function returns.

'NOSET'
prevents the automatic passing of SAS data set variable values to macro or DATA step variables even if the SET routine has been called.

Details

FETCH returns a 0 if the operation was successful, ≠0 if it was not successful, and – 1 if the end of the data set is reached. FETCH skips observations marked for deletion.

If the SET routine has been called previously, the values for any data set variables are automatically passed from the DDV to the corresponding DATA step or macro variables. To override this behavior temporarily so that fetched values are not automatically copied to the DATA step or macro variables, use the NOSET option.

Examples

This example fetches the next observation from the SAS data set MYDATA. If the end of the data set is reached or if an error occurs, SYSMSG retrieves the appropriate message and writes it to the SAS log. Note that in a macro statement you do not enclose character strings in quotation marks.

```
%let dsid=%sysfunc(open(mydata,i));
%let rc=%sysfunc(fetch(&dsid));
%if &rc ne 0 %then
  %put %sysfunc(sysmsg());
%else
```

```
      %do;
         ...more macro statements...
      %end;
   %let rc=%sysfunc(close(&dsid));
```

See Also

CALL Routine:
  "CALL SET" on page 269

Functions:
  "FETCHOBS" on page 353
  "GETVARC" on page 390
  "GETVARN" on page 391

FETCHOBS

Reads a specified observation from a SAS data set into the Data Set Data Vector (DDV) and returns a value

Category: SAS File I/O

Syntax

FETCHOBS(*data-set-id*,*obs-number*<,*options*>)

Arguments

data-set-id
specifies the data set identifier that the OPEN function returns.

obs-number
specifies the number of the observation to read. FETCHOBS treats the observation value as a relative observation number unless you specify the ABS option. The relative observation numbert may or may not coincide with the physical observation number on disk, because the function skips observations marked for deletion. When a WHERE clause is active, the function counts only observations that meet the WHERE condition.

Default: FETCHOBS skips deleted observations.

options
names one or more options, separated by blanks and enclosed in quotation marks:

'ABS'	specifies that the value of *obs-number* is absolute; that is, deleted observations are counted.
'NOSET'	prevents the automatic passing of SAS data set variable values to DATA step or macro variables even if the SET routine has been called.

Details

FETCHOBS returns 0 if the operation was successful, ≠0 if it was not successful, and –1 if the end of the data set is reached. To retrieve the error message that is associated with a nonzero return code, use the SYSMSG function. If the SET routine has been called previously, the values for any data set variables are automatically passed from the DDV to the corresponding DATA step or macro variables. To override this behavior temporarily, use the NOSET option.

If *obs-number* is less than 1, the function returns an error condition. If *obs-number* is greater than the number of observations in the SAS data set, the function returns an end-of-file condition.

Examples

This example fetches the tenth observation from the SAS data set MYDATA. If an error occurs, the SYSMSG function retrieves the error message and writes it to the SAS log. Note that in a macro statement you do not enclose character strings in quotation marks.

```
%let rc = %sysfunc(fetchobs(&mydataid,10));
%if &rc = -1 %then
   %put End of data set has been reached.;
%if &rc > 0 %then %put %sysfunc(sysmsg());
```

See Also

CALL Routine:

“CALL SET” on page 269

Functions:

“FETCH” on page 352

“GETVARC” on page 390

“GETVARN” on page 391

FEXIST

Verifies the existence of an external file associated with a fileref and returns a value

Category: External Files

Syntax

FEXIST(*fileref*)

Argument

fileref

specifies the fileref assigned to an external file.

Restriction: The *fileref* must have been previously assigned.

Operating Environment Information: In some operating environments, you can specify a fileref that was assigned with an environment variable. For details, see the SAS documentation for your operating environment. △

Details

FEXIST returns 1 if the external file that is associated with *fileref* exists, and 0 if the file does not exist. You can assign filerefs by using the FILENAME statement or the FILENAME external file access function. In some operating environments, you can also assign filerefs by using system commands.

Comparison

FILEEXIST verifies the existence of a file based on its physical name.

Examples

This example verifies the existence of an external file and writes the result to the SAS log:

```
%if %sysfunc(fexist(&fref)) %then
   %put The file identified by the fileref
      &fref exists.;
%else
   %put %sysfunc(sysmsg());
```

See Also

Functions:
- "EXIST" on page 344
- "FILEEXIST" on page 357
- "FILENAME" on page 358
- "FILEREF" on page 360

Statement:
- "FILENAME" on page 821

FGET

Copies data from the File Data Buffer (FDB) into a variable and returns a value

Category: External Files

Syntax

FGET(*file-id*,*variable*<,*length*>)

Arguments

file-id
specifies the identifier that was assigned when the file was opened, generally by the FOPEN function.

variable
in a DATA step, specifies a character variable to hold the data. In a macro, specifies a macro variable to hold the data. If *variable* is a macro variable and it does not exist, it is created.

length
specifies the number of characters to retrieve from the FDB. If *length* is specified, only the specified number of characters is retrieved (or the number of characters remaining in the buffer i f that number is less than length). If *length* is omitted, all characters in the FDB from the current column position to the next delimiter are returned. The default delimiter is a blank. The delimiter is not retrieved. See "FSEP" on page 383 for more information on delimiters.

Details

FGET returns 0 if the operation was successful, or –1 if the end of the FDB was reached or no more tokens were available.

After FGET is executed, the column pointer moves to the next read position in the FDB.

Examples

This example assigns the fileref MYFILE to an external file and attempts to open the file. If the file is opened successfully, it reads the first record into the File Data Buffer, retrieves the first token of the record and stores i t in the variable MYSTRING, and then closes the file. Note that in a macro statement you do not enclose character strings in quotation marks.

```
%let filrf=myfile;
%let rc=%sysfunc(filename(filrf,
   physical-filename));
%let fid=%sysfunc(fopen(&filrf));
%if &fid > 0 %then
   %do;
      %let rc=%sysfunc(fread(&fid));
      %let rc=%sysfunc(fget(&fid,mystring));
      %put &mystring;
      %let rc=%sysfunc(fclose(&fid));
   %end;
%let rc=%sysfunc(filename(filrf));
```

See Also

Functions:

“FCLOSE” on page 348
“FILENAME” on page 358
“FOPEN” on page 371
“FPOS” on page 377
“FREAD” on page 379
“FSEP” on page 383
“MOPEN” on page 452

FILEEXIST

Verifies the existence of an external file by its physical name and returns a value

Category: External Files

Syntax

FILEEXIST(*file-name*)

Argument

file-name
specifies a fully qualified physical filename of the external file in the operating environment. In a DATA step, *file-name* can be a character string in quotation marks or it can be a DATA ste p variable. In a macro, *file-name* is a macro variable.

Details

FILEEXIST returns 1 if the external file exists and 0 if the external file does not exist. The specification of the physical name for *file-name* varies according to the operating environment.

Although your operating environment utilities may recognize partial physical filenames, you must always use fully qualified physical filenames with FILEEXIST.

Examples

This example verifies the existence of an external file. If the file exists, FILEEXIST opens the file. If the file does not exist, FILEEXIST displays a message in the SAS log. Note that in a macro statement you do not enclose chara cter strings in quotation marks.

```
%if %sysfunc(fileexist(&myfilerf)) %then
    %let fid=%sysfunc(fopen(&myfilerf));
%else
   %put The external file &myfilerf does not exist.;
```

See Also

Functions:
- "EXIST" on page 344
- "FEXIST" on page 354
- "FILENAME" on page 358
- "FILEREF" on page 360
- "FOPEN" on page 371

FILENAME

Assigns or deassigns a fileref for an external file, directory, or output device and returns a value

Category: External Files

Syntax

FILENAME(*fileref*,*file-name*<,*device-type*<,*host-options*<,*dir-ref*>>>)

Arguments

fileref
in a DATA step, specifies the fileref to assign to the external file. In a macro (for example, in the %SYSFUNC function), *fileref* is the name of a macro variable (without an ampersand) whose value contains the fileref to assign to the external file.

Tip: A blank *fileref* (`''`)causes an error. If the fileref is a DATA step character variable with a blank value and a minimum length of 8 characters, a fileref is generated for you.

Tip: If a macro variable named in *fileref* has a null value, a fileref is generated for you.

file-name
specifies the external file. Specifying a blank *file-name* deassigns one that was assigned previously.

device-type
specifies the type of device or the access method that is used if the fileref points to an input or output device or location that is not a physical file:

DISK	specifies that the device is a disk drive. Tip: When you assign a fileref to a file on disk, you are not required to specify DISK. Alias: BASE
DUMMY	specifies that the output to the file is discarded. Tip: Specifying DUMMY can be useful for testing.
GTERM	indicates that the output device-type is a graphics device that will be receiving graphics data.

PIPE	specifies an unnamed pipe. *Note:* Some operating environments do not support pipes. △
PLOTTER	specifies an unbuffered graphics output device.
PRINTER	specifies a printer or printer spool file.
TAPE	specifies a tape drive.
TEMP	creates a temporary file that exists only as long as the filename is assigned. The temporary file can be accessed only through the logical name and is available only while the logical name exists. Restriction: Do not specify a physical pathname. If you do, SAS returns an error. Tip: Files manipulated by the TEMP device can have the same attributes and behave identically to DISK files.
TERMINAL	specifies the user's terminal.

The FILENAME function also supports operating environment specific devices. For details, see the SAS documentation for your operating environment.

host-options
specifies host-specific details such as file attributes and processing attributes. For details, see the SAS documentation for your operating environment.

dir-ref
specifies the fileref that was assigned to the directory or partitioned data set in which the external file resides.

Details

FILENAME returns 0 if the operation was successful, ≠0 if it was not successful. The name associated with the file or device is called a *fileref* (file reference name). Other system functions that manipulate external files and directories require that the files be identified by fileref rather than by physical filename.

Operating Environment Information: The term *directory* in this description refers to an aggregate grouping of files managed by the operating environment. Different operating environments identify such groupings with different names, such as directory, subdirectory, MACLIB, or partitioned data set. For details, see the SAS documentation for your operating environment.

Under some operating environments, you can also assign filerefs by using system commands. Depending on the operating environment, FILENAME may be unable to change or deassign filerefs assigned outside the SAS System. △

The association between a fileref and a physical file lasts only for the duration of the current SAS session or until you change or discontinue the association by using FILENAME. You can deassign filerefs by specifying a null string for the *file-name* argument in FILENAME.

Examples

Example 1: Assigning a Fileref to an External File This example assigns the fileref MYFILE to an external file, then deassigns the fileref. Note that in a macro statement you do not enclose character strings in quotation marks.

```
%let filrf=myfile;
%let rc=%sysfunc(filename(filrf, physical-filename));
```

```
%if &rc ne 0 %then
   %put %sysfunc(sysmsg());
%let rc=%sysfunc(filename(filrf));
```

Example 2: Assigning a System-Generated Fileref This example assigns a system-generated fileref to an external file. The fileref is stored in the variable FNAME. Note that in a macro statement you do not enclose character strings in quotation marks.

```
%let rc=%sysfunc(filename(fname, physical-filename));
%if &rc %then
   %put %sysfunc(sysmsg());
%else
   %do;
      more macro statements
   %end;
```

Example 3: Assigning a Fileref to a Pipe File This example assigns the fileref MYPIPE for a pipe file with the output from the UNIX command LS, which lists the files in the directory /u/myid. Note that in a macro statement you do not enclose character strings in quotation marks.

```
%let filrf=mypipe;
%let rc=%sysfunc(filename(filrf, %str(ls /u/myid), pipe));
```

See Also

Functions:

“FEXIST” on page 354

“FILEEXIST” on page 357

“FILEREF” on page 360

“SYSMSG” on page 558

FILEREF

Verifies that a fileref has been assigned for the current SAS session and returns a value

Category: External Files

Syntax

FILEREF(*fileref*)

Argument

fileref
specifies the fileref to be validated.

Range: 1 to 8 characters

Details

A negative return code indicates that the fileref exists but the physical file associated with the fileref does not exist. A positive value indicates that the fileref is not assigned. A value of zero indicates that the fileref and external file both exist.

A fileref can be assigned to an external file by using the FILENAME statement or the FILENAME function.

Operating Environment Information: Under some operating environments, filerefs can also be assigned by using system commands. For details, see the SAS documentation for your operating environment. △

Examples

Example 1: Verifying that a Fileref is Assigned This example tests whether the fileref MYFILE is currently assigned to an external file. A system error message is issued if the fileref is not currently assigned:

```
%if %sysfunc(fileref(myfile))>0 %then
   %put MYFILE is not assigned;
```

Example 2: Verifying that Both a Fileref and a File Exist This example tests for a zero value to determine if both the fileref and the file exist:

```
%if %sysfunc(fileref(myfile)) ne 0 %then
   %put %sysfunc(sysmsg());
```

See Also

Functions:
- "FEXIST" on page 354
- "FILEEXIST" on page 357
- "FILENAME" on page 358
- "SYSMSG" on page 558

Statement:
- "FILENAME" on page 821

FINFO

Returns the value of a file information item

Category: External Files

Syntax

FINFO(*file-id*,*info-item*)

Arguments

file-id
specifies the identifier that was assigned when the file was opened, generally by the FOPEN function.

info-item
specifies the name of the file information item to be retrieved. This is a character value.

Details

FINFO returns the value of a system-dependent information item for an external file. FINFO returns a blank if the value given for *info-item* is invalid.

Operating Environment Information: The information available on files depends on the operating environment. △

Comparisons

- □ FOPTNAME determines the names of the available file information items.
- □ FOPTNUM determines the number of system-dependent information items available.

Examples

This example stores information items about an external file in a SAS data set:

```
data info;
   length infoname infoval $60;
   drop rc fid infonum i close;
   rc=filename('abc','physical-filename');
   fid=fopen('abc');
   infonum=foptnum(fid);
   do i=1 to infonum;
      infoname=foptname(fid,i);
      infoval=finfo(fid,infoname);
      output;
   end;
   close=fclose(fid);
run;
```

See Also

Functions:

“FCLOSE” on page 348
“FOPTNUM” on page 374
“MOPEN” on page 452

FINV

Returns a quantile from the *F* distribution

Category: Quantile

Syntax

FINV (*p*, *ndf*, *ddf* <,*nc*>)

Arguments

p
is a numeric probability.

Range: $0 \leq p < 1$

ndf
is a numeric numerator degrees of freedom parameter.

Range: $ndf > 0$

ddf
is a numeric denominator degrees of freedom parameter.

Range: $ddf > 0$

nc
is an optional numeric noncentrality parameter.

Range: $nc \geq 0$

Details

The FINV function returns the p^{th} quantile from the *F* distribution with numerator degrees of freedom *ndf*, denominator degrees of freedom *ddf*, and noncentrality parameter *nc*. The probability that an observation from the *F* distribution is less than the quantile is *p*. This function accepts noninteger degrees of freedom parameters *ndf* and *ddf*.

If the optional parameter *nc* is not specified or has the value 0, the quantile from the central *F* distribution is returned. The noncentrality parameter *nc* is defined such that if X and Y are normal random variables with means μ and 0, respectively, and variance 1, then X^2/Y^2 has a noncentral *F* distribution with $nc = \mu^2$.

CAUTION:
For large values of *nc*,the algorithm could fail; in that case, a missing value is returned. △

Note: FINV is the inverse of the PROBF function. △

Examples

These statements compute the 95th quantile value of a central *F* distribution with 2 and 10 degrees of freedom and a noncentral *F* distribution with 2 and 10 degrees of freedom and a noncentrality parameter equal to 3.2:

SAS Statements	Results
`q1=finv(.95,2,10);`	4.1028210151
`q2=finv(.95,2,10.3,2);`	7.583766024

FIPNAME

Converts FIPS codes to uppercase state names

Category: State and ZIP Code

Syntax

FIPNAME(*expression*)

Arguments

expression
specifies a numeric expression that represents a U.S. FIPS code.

Details

The FIPNAME function converts a U.S. Federal Information Processing Standards (FIPS) code to the corresponding state or U.S. territory name in uppercase, returning a value of up to 20 characters.

Comparisons

The FIPNAME, FIPNAMEL, and FIPSTATE functions take the same argument but return different values. FIPNAME returns uppercase state names. FIPNAMEL returns mixed case state names. FIPSTATE returns a two-character state postal code (or world-wide GSA geographic code for U.S. territories) in uppercase.

Examples

The examples show the differences when using FIPNAME, FIPNAMEL, and FIPSTATE.

SAS Statements	Results

x=fipname(37);
put x;
	NORTH CAROLINA

x=fipnamel(37);
put x;
	North Carolina

x=fipstate(37);
put x;
	NC

See Also

Functions:
- "FIPNAMEL" on page 365
- "FIPSTATE" on page 366
- "STFIPS" on page 551
- "STNAME" on page 552
- "STNAMEL" on page 553

FIPNAMEL

Converts FIPS codes to mixed case state names

Category: State and ZIP Code

Syntax

FIPNAMEL(*expression*)

Arguments

expression
specifies a numeric expression that represents a U.S. FIPS code.

Details

The FIPNAMEL function converts a U.S. Federal Information Processing Standards (FIPS) code to the corresponding state or U.S. territory name in mixed case, returning a value of up to 20 characters.

Comparisons

The FIPNAME, FIPNAMEL, and FIPSTATE functions take the same argument but return different values. FIPNAME returns uppercase state names. FIPNAMEL returns mixed case state names. FIPSTATE returns a two-character state postal code (or world-wide GSA geographic code for U.S. territories) in uppercase.

Examples

The examples show the differences when using FIPNAME, FIPNAMEL, and FIPSTATE.

SAS Statements	Results

x=fipname(37);
put x;
	NORTH CAROLINA

x=fipnamel(37);
put x;
	North Carolina

x=fipstate(37);
put x;
	NC

See Also

Functions:
- "FIPNAME" on page 364
- "FIPSTATE" on page 366
- "STFIPS" on page 551
- "STNAME" on page 552
- "STNAMEL" on page 553

FIPSTATE

Converts FIPS codes to two-character postal codes

Category: State and ZIP Code

Syntax

FIPSTATE(*expression*)

Arguments

expression
specifies a numeric expression that represents a U.S. FIPS code.

Details

The FIPSTATE function converts a U.S. Federal Information Processing Standards (FIPS) code to a two-character state postal code (or world-wide GSA geographic code for U.S. territories) in uppercase.

Comparisons

The FIPNAME, FIPNAMEL, and FIPSTATE functions take the same argument but return different values. FIPNAME returns uppercase state names. FIPNAMEL returns mixed case state names. FIPSTATE returns a two-character state postal code (or world-wide GSA geographic code for U.S. territories) in uppercase.

Examples

The examples show the differences when using FIPNAME, FIPNAMEL, and FIPSTATE.

SAS Statements	Results

`x=fipname(37);`
`put x;`
	`NORTH CAROLINA`

`x=fipnamel(37);`
`put x;`
	`North Carolina`

`x=fipstate(37);`
`put x;`
	`NC`

See Also

Functions:

“FIPNAME” on page 364

“FIPNAMEL” on page 365

“STFIPS” on page 551

“STNAME” on page 552

“STNAMEL” on page 553

FLOOR

Returns the largest integer that is less than or equal to the argument

Category: Truncation

Syntax

FLOOR(*argument*)

Arguments

argument
is numeric.

Details

If the argument is within 10**(–12) of an integer, the function returns that integer.

Examples

SAS Statements	Results

`var1=2.1;`
`a=floor(var1);`
`put a;`
	2

`var2=-2.4;`
`b=floor(var2);`
`put b;`
	-3

`c=floor(3);`
`put c;`
	3

`d=floor(-1.6);`
`put d;`
	-2

`e=floor(1.-1.e-13);`
`put e;`
	1

FNONCT

Returns the value of the noncentrality parameter of an F distribution

Category: Mathematical

Syntax

FNONCT(*x*,*ndf*,*ddf*,*prob*)

Arguments

x
: is a numeric random variable.

  Range: $x \geq 0$

ndf
: is a numeric numerator degrees-of-freedom parameter.

  Range: $ndf > 0$

ddf
: is a numeric denominator degrees-of-freedom parameter.

  Range: $ddf > 0$

prob
: is a probability.

  Range: $0 < prob < 1$

Details

The FNONCT function returns the nonnegative noncentrality parameter from a noncentral *F* distribution whose parameters are *x*, *ndf*, *ddf*, and *nc*. If *prob* is greater than the probability from the central *F* distribution whose parameters are *x*, *ndf*, and *ddf*, a root to this problem does not exist. In this case a missing value is returned. A Newton-type algorithm is used to find a nonnegative root *nc* of the equation

$$P_f\left(x|ndf, ddf, nc\right) - prob = 0$$

where

$$P_f\left(x|ndf, ddf, nc\right) = e^{\frac{-nc}{2}} \sum_{j=0}^{\infty} \frac{\left(\frac{nc}{2}\right)^j}{j!} I_{\frac{(ndf)x}{ddf+(ndf)x}} \left(\frac{ddf}{2} + j, \frac{ddf}{2}\right)$$

where *I* (. . .) is the probability from the beta distribution that is given by

$$I_x\left(a, b\right) = \frac{}{\frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}} \int_0^x t^{a-1}\left(1-t\right)^{b-1} dt$$

If the algorithm fails to converge to a fixed point, a missing value is returned.

Examples

```
data work;
   x=2;
   df=4;
   ddf=5;
   do nc=1 to 3 by .5;
      prob=probf(x,df,ddf,nc);
      ncc=fnonct(x,df,ddf,prob);
      output;
   end;
run;
proc print;
run;
```

```
OBS    x    df    ddf    nc      prob      ncc
 1     2     4     5     1.0    0.69277    1.0
 2     2     4     5     1.5    0.65701    1.5
 3     2     4     5     2.0    0.62232    2.0
 4     2     4     5     2.5    0.58878    2.5
 5     2     4     5     3.0    0.55642    3.0
```

FNOTE

Identifies the last record that was read and returns a value that FPOINT can use

Category: External Files

Syntax

FNOTE(*file-id*)

Argument

file-id
specifies the identifier that was assigned when the file was opened, generally by the FOPEN function.

Details

You can use FNOTE like a bookmark, marking the position in the file so that your application can later return to that position using FPOINT. The value returned by FNOTE is required by the FPOINT function to reposition the file pointe r on a specific record.

To free the memory associated with each FNOTE identifier, use DROPNOTE.

Examples

This example assigns the fileref MYFILE to an external file and attempts to open the file. If the file is opened successfully, indicated by a positive value in the variable FID, then it reads the records, stores in the variable NOTE 3 the position of the third record read, and then later uses FPOINT to point back to NOTE3 to update the file. After updating the record, it closes the file:

```
%let
fref=MYFILE;
%let rc=%sysfunc(filename(fref,
   physical-filename));
%let fid=%sysfunc(fopen(&fref,u));
%if &fid > 0 %then
   %do;
      %let rc=%sysfunc(fread(&fid));
         /* Read second record. */
      %let rc=%sysfunc(fread(&fid));
         /* Read third record. */
      %let rc=%sysfunc(fread(&fid));
         /* Note position of third record. */
      %let note3=%sysfunc(fnote(&fid));
         /* Read fourth record. */
      %let rc=%sysfunc(fread(&fid));
         /* Read fifth record. */
      %let rc=%sysfunc(fread(&fid));
         /* Point to third record. */
      %let rc=%sysfunc(fpoint(&fid,&note3));
         /* Read third record. */
      %let rc=%sysfunc(fread(&fid));
         /* Copy new text to FDB. */
      %let rc=%sysfunc(fput(&fid,New text));
```

```
            /* Update third record  */
            /* with data in FDB. */
         %let rc=%sysfunc(fwrite(&fid));
            /* Close file. */
         %let rc=%sysfunc(fclose(&fid));
      %end;
   %let rc=%sysfunc(filename(fref));
```

See Also

Functions:

- "DROPNOTE" on page 338
- "FCLOSE" on page 348
- "FILENAME" on page 358
- "FOPEN" on page 371
- "FPOINT" on page 375
- "FPUT" on page 378
- "FREAD" on page 379
- "FREWIND" on page 380
- "FWRITE" on page 385
- "MOPEN" on page 452

FOPEN

Opens an external file and returns a file identifier value

Category: External Files

Syntax

FOPEN(*fileref*<,*open-mode*<,*record-length*<,*record-format*>>>)

Arguments

fileref
specifies the fileref assigned to the external file.

open-mode
specifies the type of access to the file:

A	APPEND mode allows writing new records after the current end of the file.
I	INPUT mode allows reading only (default).
O	OUTPUT mode defaults to the OPEN mode specified in the host option in the FILENAME statement or function. If no host option is specified, it allows writing new records at the beginning of the file.

S	Sequential input mode is used for pipes and other sequential devices such as hardware ports.
U	UPDATE mode allows both reading and writing.

Default: I

record-length
specifies the logical record length of the file. To use the existing record length for the file, specify a length of 0, or do not provide a value here.

record-format
specifies the record format of the file. To use the existing record format, do not specify a value here. Valid values are:

B	data are to be interpreted as binary data.
D	use default record format.
E	use editable record format.
F	file contains fixed length records.
P	file contains printer carriage control in host-dependent record format. *Note:* For OS/390 data sets with FBA or VBA record format, specify 'P' for the *record-format* argument.
V	file contains variable length records.

Details

CAUTION:
Use OUTPUT mode with care. Opening an existing file for output overwrites the current contents of the file without warning. △

The FOPEN function opens an external file for reading or updating and returns a file identifier value that is used to identify the open file to other functions. You must associate a fileref with the external file before calling the F OPEN function. FOPEN returns a 0 if the file could not be opened. You can assign filerefs by using the FILENAME statement or the FILENAME external file access function. Under some operating environments, you can also assign filerefs by using system comman ds.

Operating Environment Information: On some operating environments you must close the file at the end o f the DATA step using the FCLOSE function. For details, see the SAS documentation for your operating environment. △

Examples

Example 1: Opening a File Using Defaults This example assigns the fileref MYFILE to an external file and attempts to open the file for input using all defaults. Note that in a macro statement you do not enclose character strings in quotation marks.

```
%let filrf=myfile;
%let rc=%sysfunc(filename(filrf,
   physical-filename));
%let fid=%sysfunc(fopen(&filrf));
```

Example 2: Opening a File without Using Defaults This example attempts to open a file for input without using defaults. Note that in a macro statement you do not enclose character strings in quotation marks.

```
%let fid=%sysfunc(fopen(file2,o,132,e));
```

See Also

Functions:
- "DOPEN" on page 334
- "FCLOSE" on page 348
- "FILENAME" on page 358
- "FILEREF" on page 360
- "MOPEN" on page 452

Statement:
- "FILENAME" on page 821

FOPTNAME

Returns the name of an item of information about a file

Category: External Files

Syntax

FOPTNAME(*file-id*,*nval*)

Arguments

file-id
specifies the identifier that was assigned when the file was opened, generally by the FOPEN function.

nval
specifies the number of the information item.

Details

FOPTNAME returns a blank if an error occurred.

Operating Environment Information: The number, value, and type of information items that are available depend on the operating environment. △

Examples

Example 1: Retrieving File Information Items and Writing Them to the Log This example retrieves the system-dependent file information items that are available and writes them to the log:

```
%let filrf=myfile;
%let rc=%sysfunc(filename(filrf,
   physical-filename));
%let fid=%sysfunc(fopen(&filrf));
%let infonum=%sysfunc(foptnum(&fid));
%do j=1 %to &infonum;
```

```
      %let name=%sysfunc(foptname(&fid,&j));
      %let value=%sysfunc(finfo(&fid,&name));
      %put File attribute &name equals &value;
   %end;
   %let rc=%sysfunc(fclose(&fid));
   %let rc=%sysfunc(filename(filrf));
```

Example 2: Creating a Data Set with Names and Values of File Attributes This example creates a data set that contains the name and value of the available file attributes:

```
data fileatt;
   length name $ 20 value $ 40;
   drop rc fid j infonum;
   rc=filename("myfile","physical-filename");
   fid=fopen("myfile");
   infonum=foptnum(fid);
   do j=1 to infonum;
      name=foptname(fid,j);
      value=finfo(fid,name);
      put 'File attribute ' name
       'has a value of ' value;
      output;
   end;
   rc=filename("myfile");
run;
```

See Also

Functions:
- "DINFO" on page 331
- "DOPTNAME" on page 335
- "DOPTNUM" on page 336
- "FCLOSE" on page 348
- "FILENAME" on page 358
- "FINFO" on page 361
- "FOPEN" on page 371
- "FOPTNUM" on page 374
- "MOPEN" on page 452

FOPTNUM

Returns the number of information items that are available for an external file

Category: External Files

Syntax

FOPTNUM(*file-id*)

Argument

file-id
specifies the identifier that was assigned when the file was opened, generally by the FOPEN function.

Details

Operating Environment Information: The number, value, and type of information items that are available depend on the operating environment. △

Comparisons

- □ Use FOPTNAME to determine the names of the items that are available for a particular operating environment.
- □ Use FINFO to retrieve the value of a particular information item.

Examples

This example opens the external file with the fileref MYFILE and determines the number of system-dependent file information items available:

```
%let fid=%sysfunc(fopen(myfile));
%let infonum=%sysfunc(foptnum(&fid));
```

See Also

Functions:
- "DINFO" on page 331
- "DOPTNAME" on page 335
- "DOPTNUM" on page 336
- "FINFO" on page 361
- "FOPEN" on page 371
- "FOPTNAME" on page 373
- "MOPEN" on page 452

See the "Examples" on page 373 in the FOPTNAME Function.

FPOINT

Positions the read pointer on the next record to be read and returns a value

Category: External Files

Syntax

FPOINT(*file-id*,*note-id*)

Arguments

file-id
specifies the identifier that was assigned when the file was opened, generally by the FOPEN function.

note-id
specifies the identifier that was assigned by the FNOTE function.

Details

FPOINT returns 0 if the operation was successful, or ≠0 if it was not successful. FPOINT determines only the record to read next. It has no impact on which record is written next. When you open the file for update, FWRITE writes to the most recently read record.

Examples

This example assigns the fileref MYFILE to an external file and attempts to open the file. If the file is opened successfully, it reads the records and uses NOTE3 to store the position of the third record read. Later, it points back to NOTE3 to update the file, and closes the file afterward:

```
%let filrf=myfile;
%let rc=%sysfunc(filename(filrf,
   physical-filename));
%let fid=%sysfunc(fopen(&filrf,u));
%if &fid > 0 %then
   %do;
         /* Read first record. */
      %let rc=%sysfunc(fread(&fid));
         /* Read second record. */
      %let rc=%sysfunc(fread(&fid));
         /* Read third record. */
      %let rc=%sysfunc(fread(&fid));
         /* Note position of third record. */
      %let note3=%sysfunc(fnote(&fid));
         /* Read fourth record. */
      %let rc=%sysfunc(fread(&fid));
         /* Read fifth record. */
      %let rc=%sysfunc(fread(&fid));
         /* Point to third record. */
      %let rc=%sysfunc(fpoint(&fid,&note3));
         /* Read third record. */
      %let rc=%sysfunc(fread(&fid));
         /* Copy new text to FDB. */
      %let rc=%sysfunc(fput(&fid,New text));
         /* Update third record  */
         /* with data in FDB. */
      %let rc=%sysfunc(fwrite(&fid));
         /* Close file. */
      %let rc=%sysfunc(fclose(&fid));
   %end;
%let rc=%sysfunc(filename(filrf));
```

See Also

Functions:

“DROPNOTE” on page 338
“FCLOSE” on page 348
“FILENAME” on page 358
“FNOTE” on page 369
“FOPEN” on page 371
“FPUT” on page 378
“FREAD” on page 379
“FREWIND” on page 380
“FWRITE” on page 385
“MOPEN” on page 452

FPOS

Sets the position of the column pointer in the File Data Buffer (FDB) and returns a value

Category: External Files

Syntax

FPOS(*file-id*,*nval*)

Arguments

file-id
specifies the identifier that was assigned when the file was opened, generally by the FOPEN function.

nval
specifies the column at which to set the pointer.

Details

FPOS returns 0 if the operation was successful, ≠0 if it was not successful. If the specified position is past the end of the current record, the size of the record is increased appropriately. However, in a fixed block or VBA file, if you specify a column position beyond the end of the record, the record size does not change and the text string is not written to the file.

Examples

This example assigns the fileref MYFILE to an external file and attempts to open the file. If the file is opened successfully, indicated by a positive value in the variable FID, it places data into the file’s buffer at column 12, writes the record, and closes the file:

```
%let filrf=myfile;
%let rc=%sysfunc(filename(filrf,
   physical-filename));
%let fid=%sysfunc(fopen(&filrf,o));
%if (&fid > 0) %then
   %do;
      %let dataline=This is some data.;
         /* Position at column 12 in the FDB. */
      %let rc=%sysfunc(fpos(&fid,12));
         /* Put the data in the FDB. */
      %let rc=%sysfunc(fput(&fid,&dataline));
         /* Write the record. */
      %let rc=%sysfunc(fwrite(&fid));
         /* Close the file. */
      %let rc=%sysfunc(fclose(&fid));
   %end;
%let rc=%sysfunc(filename(filrf));
```

See Also

Functions:

“FCLOSE” on page 348
“FCOL” on page 349
“FILENAME” on page 358
“FOPEN” on page 371
“FPUT” on page 378
“FWRITE” on page 385
“MOPEN” on page 452

FPUT

Moves data to the File Data Buffer (FDB) of an external file, starting at the FDB’s current column position, and returns a value

Category: External Files

Syntax

FPUT(*file-id*,*cval*)

Arguments

file-id
specifies the identifier that was assigned when the file was opened, generally by the FOPEN function.

cval
specifies the file data. In a DATA step, *cval* can be a character string in quotation marks or a DATA step variable. In a macro, *cval* is a macro variable.

Details

FPUT returns 0 if the operation was successful, ≠0 if it was not successful. The number of bytes moved to the FDB is determined by the length of the variable. The value of the column pointer is then increased to one position pas t the end of the new text.

Examples

This example assigns the fileref MYFILE to an external file and attempts to open the file in APPEND mode. If the file is opened successfully, indicated by a positive value in the variable FID, it moves data to the FDB using FPUT, ap pends a record using FWRITE, and then closes the file. Note that in a macro statement you do not enclose character strings in quotation marks.

```
%let filrf=myfile;
%let rc=%sysfunc(filename(filrf,
   physical-filename));
%let fid=%sysfunc(fopen(&filrf,a));
%if &fid > 0 %then
   %do;
      %let mystring=This is some data.;
      %let rc=%sysfunc(fput(&fid,&mystring));
      %let rc=%sysfunc(fwrite(&fid));
      %let rc=%sysfunc(fclose(&fid));
   %end;
%else
   %put %sysfunc(sysmsg());
%let rc=%sysfunc(filename(filrf));
```

See Also

Functions:
- “FCLOSE” on page 348
- “FILENAME” on page 358
- “FNOTE” on page 369
- “FOPEN” on page 371
- “FPOINT” on page 375
- “FPOS” on page 377
- “FWRITE” on page 385
- “MOPEN” on page 452
- “SYSMSG” on page 558

FREAD

Reads a record from an external file into the File Data Buffer (FDB) and returns a value

Category: External Files

Syntax

FREAD(*file-id*)

Argument

file-id
specifies the identifier that was assigned when the file was opened, generally by the FOPEN function.

Details

FREAD returns 0 if the operation was successful, ≠0 if it was not successful. The position of the file pointer is updated automatically after the read operation so that successive FREAD functions read successive file records.

To position the file pointer explicitly, use FNOTE, FPOINT, and FREWIND.

Examples

This example assigns the fileref MYFILE to an external file and attempts to open the file. If the file opens successfully, it lists all of the file's records in the log:

```
%let filrf=myfile;
%let rc=%sysfunc(filename(filrf,
   physical-filename));
%let fid=%sysfunc(fopen(&filrf));
%if &fid > 0 %then
   %do %while(%sysfunc(fread(&fid)) = 0);
      %let rc=%sysfunc(fget(&fid,c,200));
           %put &c;
      %end;
   %let rc=%sysfunc(fclose(&fid));
%let rc=%sysfunc(filename(filrf));
```

See Also

Functions:

"FCLOSE" on page 348
"FGET" on page 355
"FILENAME" on page 358
"FNOTE" on page 369
"FOPEN" on page 371
"FREWIND" on page 380
"FREWIND" on page 380
"MOPEN" on page 452

FREWIND

Positions the file pointer to the start of the file and returns a value

Category: External Files

Syntax

FREWIND(*file-id*)

Argument

file-id
specifies the identifier that was assigned when the file was opened, generally by the FOPEN function.

Details

FREWIND returns 0 if the operation was successful, ≠0 if it was not successful. FREWIND has no effect on a file opened with sequential access.

Examples

This example assigns the fileref MYFILE to an external file. Then it opens the file and reads the records until the end of the file is reached. The FREWIND function then repositions the pointer to the beginning of the file. The first record is read again and stored in the File Data Buffer (FDB). The first token is retrieved and stored in the macro variable VAL:

```
%let filrf=myfile;
%let rc=%sysfunc(filename(filrf,
   physical-filename));
%let fid=%sysfunc(fopen(&filrf));
%let rc=0;
%do %while (&rc ne -1);
   /* Read a record. */
   %let rc=%sysfunc(fread(&fid));
%end;
   /* Reposition pointer to beginning of file. */
%if &rc = -1 %then
  %do;
    %let rc=%sysfunc(frewind(&fid));
       /* Read first record. */
    %let rc=%sysfunc(fread(&fid));
       /* Read first token  */
       /* into macro variable VAL. */
    %let rc=%sysfunc(fget(&fid,val));
    %put val=&val;
  %end;
%else
  %put Error on fread=%sysfunc(sysmsg());
%let rc=%sysfunc(fclose(&fid));
%let rc=%sysfunc(filename(filrf));
```

See Also

Functions:
- "FCLOSE" on page 348
- "FGET" on page 355
- "FILENAME" on page 358
- "FOPEN" on page 371
- "FREAD" on page 379
- "MOPEN" on page 452
- "SYSMSG" on page 558

FRLEN

Returns the size of the last record read, or, if the file is opened for output, returns the current record size

Category: External Files

Syntax

FRLEN(*file-id*)

Argument

file-id
specifies the identifier that was assigned when the file was opened, generally by the FOPEN function.

Examples

This example opens the file that is identified by the fileref MYFILE. It determines the minimum and maximum length of records in the external file and writes the results to the log:

```
%let fid=%sysfunc(fopen(myfile));
%let min=0;
%let max=0;
%if (%sysfunc(fread(&fid)) = 0) %then
   %do;
      %let min=%sysfunc(frlen(&fid));
      %let max=&min;
      %do %while(%sysfunc(fread(&fid)) = 0);
         %let reclen=%sysfunc(frlen(&fid));
         %if (&reclen > &max) %then
            %let max=&reclen;
         %if (&reclen < &min) %then
            %let min=&reclen;
      %end;
```

```
%end;
%let rc=%sysfunc(fclose(&fid));
%put max=&max min=&min;
```

See Also

Functions:

“FCLOSE” on page 348

“FOPEN” on page 371

“FREAD” on page 379

“MOPEN” on page 452

FSEP

Sets the token delimiters for the FGET function and returns a value

Category: External Files

Syntax

FSEP(*file-id*,*character(s)*)

Arguments

file-id
specifies the identifier that was assigned when the file was opened, generally by the FOPEN function.

character
specifies one or more delimiters that separate items in the File Data Buffer (FDB). Each character listed is a delimiter. That is, if *character* is #@, either # or @ can separate items. Multiple consecutive delimiters, such as @#@, are treated as a single delimiter.

Default: blank

Details

FSEP returns 0 if the operation was successful, ≠0 if it was not successful.

Examples

An external file has data in this form:

```
John J. Doe,Male,25,Weight Lifter
Pat O'Neal,Female,22,Gymnast
```

Note that each field is separated by a comma.

This example reads the file that is identified by the fileref MYFILE, using the comma as a separator, and writes the values for NAME, GENDER, AGE, and WORK to the

SAS log. Note that in a macro statement you do not enclose character strings in quotation marks.

```
%let fid=%sysfunc(fopen(myfile));
%let rc=%sysfunc(fsep(&fid,%str(,)));
%do %while(%sysfunc(fread(&fid)) = 0);
   %let rc=%sysfunc(fget(&fid,name));
   %let rc=%sysfunc(fget(&fid,gender));
   %let rc=%sysfunc(fget(&fid,age));
   %let rc=%sysfunc(fget(&fid,work));
   %put name=%bquote(&name) gender=&gender
      age=&age work=&work;
%end;
%let rc=%sysfunc(fclose(&fid));
```

See Also

Functions:
- "FCLOSE" on page 348
- "FGET" on page 355
- "FOPEN" on page 371
- "FREAD" on page 379
- "MOPEN" on page 452

FUZZ

Returns the nearest integer if the argument is within 1E–12

Category: Truncation

Syntax

FUZZ(*argument*)

Arguments

argument
is numeric.

Details

The FUZZ function returns the nearest integer value if the argument is within 1E–12 of the integer (that is, if the absolute difference between the integer and argument is less than 1E–12). Otherwise, the argument is returned.

Examples

SAS Statements	Results

`var1=5.9999999999999;`
`x=fuzz(var1);`
`put x 16.14`
	`6.000000000000000`

`x=fuzz(5.99999999);`
`put x 16.14;`
	`5.999999990000000`

FWRITE

Writes a record to an external file and returns a value

Category: External Files

Syntax

FWRITE(*file-id*<,*cc*>)

Arguments

file-id
specifies the identifier that was assigned when the file was opened, generally by the FOPEN function.

cc
specifies a carriage-control character:

blank	starts the record on a new line.
0	skips one blank line before a new line.
-	skips two blank lines before a new line.
1	starts the line on a new page.
+	overstrikes the line on a previous line.
P	interprets the line as a terminal prompt.
=	interprets the line as carriage control information.
all else	starts the line record on a new line.

Details

FWRITE returns 0 if the operation was successful, ≠0 if it was not successful. FWRITE moves text from the File Data Buffer (FDB) to the external file. In order to use the carriage control characters, you must open the file with a record format of **`P`** (print format) in FOPEN.

Examples

This example assigns the fileref MYFILE to an external file and attempts to open the file. If the file is opened successfully, it writes the numbers 1 to 50 to the external file, skipping two blank lines. Note that in a macro statement you do not enclose character strings in quotation marks.

```
%let filrf=myfile;
%let rc=%sysfunc(filename(filrf,
   physical-filename));
%let fid=%sysfunc(fopen(&filrf,o,0,P));

%do i=1 %to 50;
   %let rc=%sysfunc(fput(&fid,
      %sysfunc(putn(&i,2.))));

   %if (%sysfunc(fwrite(&fid,-)) ne 0) %then
     %put %sysfunc(sysmsg());
%end;

%let rc=%sysfunc(fclose(&fid));
```

See Also

Functions:
- "FAPPEND" on page 347
- "FCLOSE" on page 348
- "FGET" on page 355
- "FILENAME" on page 358
- "FOPEN" on page 371
- "FPUT" on page 378
- "SYSMSG" on page 558

GAMINV

Returns a quantile from the gamma distribution

Category: Quantile

Syntax

GAMINV(*p*,*a*)

Arguments

p
is a numeric probability.
Range: $0 \le p < 1$

a
is a numeric shape parameter.
Range: $a > 0$

Details

The GAMINV function returns the p^{th} quantile from the gamma distribution, with shape parameter *a*. The probability that an observation from a gamma distribution is less than or equal to the returned quantile is *p*.

Note: GAMINV is the inverse of the PROBGAM function. △

Examples

SAS Statements	Results
`q1=gaminv(0.5,9);`	`8.6689511844`
`q2=gaminv(0.1,2.1);`	`0.5841932369`

GAMMA

Returns the value of the Gamma function

Category: Mathematical

Syntax

GAMMA(*argument*)

Arguments

argument
is numeric.

Restriction: Nonpositive integers are invalid.

Details

The GAMMA function returns the integral, which is given by

$$\mathrm{GAMMA}\,(x) = \int_0^\infty t^{x-1} e^{-1}\, dt.$$

For positive integers, GAMMA(x) is (x – 1)!. This function is commonly denoted by $\Gamma\,(x)$.

Example

SAS Statements	Results
x=gamma(6);	**120**

GETOPTION

Returns the value of a SAS system or graphics option

Category: Special

Syntax

GETOPTION(*option-name*<,*reporting-options*<,...>>)

Arguments

option-name
is the name of the system option.

Note: Do not put an equal sign after the name. For example, write PAGESIZE= as PAGESIZE.

reporting-options
specifies the reporting options. You can separate the options with blanks, or you can specify each reporting option as a separate argument to the GETOPTION function. The reporting options are

IN	reports graphic units of measure in inches.
CM	reports graphic units of measure in centimeters.
KEYWORD	returns option values in a KEYWORD= format that would be suitable for direct use in the SAS OPTIONS or GOPTIONS global statements.

Examples

□ This example saves the initial value of the YEARCUTOFF option and then resets the value to 1920. The DATA step that follows verifies the option setting and performs date processing. When the DATA step ends, the YEARCUTOFF option is set to its original value.

```
%let cutoff=%sysfunc(getoption
                    (yearcutoff,keyword));
options yearcutoff=1920;
data ages;
  if getoption('yearcutoff') = '1920' then
     do;
        ...more statements...
     end;
  else put 'Set Option YEARCUTOFF to 1920';
run;

options &cutoff;
```

□ This example defines a macro to illustrate the use of the GETOPTION function to obtain the value of system and graphics options by using different reporting options.

```
%macro showopts;
  %put PAGESIZE= %sysfunc(
     getoption(PAGESIZE));
  %put PS= %sysfunc(
     getoption(PS));
  %put LS= %sysfunc(
     getoption(LS));
  %put PS(keyword form)= %sysfunc(
     getoption(PS,keyword));
  %put LS(keyword form)= %sysfunc(
     getoption(LS,keyword));
  %put FORMCHAR= %sysfunc(
     getoption(FORMCHAR));
  %put HSIZE= %sysfunc(
     getoption(HSIZE));
  %put VSIZE= %sysfunc(
     getoption(VSIZE));
  %put HSIZE(in/keyword form)= %sysfunc(
     getoption(HSIZE,in,keyword));
  %put HSIZE(cm/keyword form)= %sysfunc(
     getoption(HSIZE,cm,keyword));
  %put VSIZE(in/keyword form)= %sysfunc(
     getoption(VSIZE,in,keyword));
  %put HSIZE(cm/keyword form)= %sysfunc(
     getoption(VSIZE,cm,keyword));
%mend;
goptions VSIZE=8.5 in HSIZE=11 in;

%showopts;
```

The following is SAS output from the example.

```
PAGESIZE= 23
PS= 23
LS= 76
PS(keyword form)= PS=23
LS(keyword form)= LS=76
FORMCHAR= |----|+|---+=|-/\<>*
HSIZE= 11.0000 in.
VSIZE= 8.5000 in.
HSIZE(in/keyword form)= HSIZE=11.0000 in.
HSIZE(cm/keyword form)= HSIZE=27.9400 cm.
VSIZE(in/keyword form)= VSIZE=8.5000 in.
HSIZE(cm/keyword form)= VSIZE=21.5900 cm.
```

Note: The default settings for pagesize and linesize depend on the mode you use to run SAS. △

GETVARC

Returns the value of a SAS data set character variable

Category: SAS File I/O

Syntax

GETVARC(*data-set-id*,*var-num*)

Arguments

data-set-id
specifies the data set identifier that the OPEN function returns.

var-num
is the number of the variable in the Data Set Data Vector (DDV).

Tip: You can obtain this value by using the VARNUM function.

Tip: This value is listed next to the variable when you use the CONTENTS procedure.

Details

Use VARNUM to obtain the number of a variable in a SAS data set. VARNUM can be nested or it can be assigned to a variable that can then be passed as the second argument, as shown in the following examples. GETVARC reads the value of a character variable from the current observation in the Data Set Data Vector (DDV) into a macro or DATA step variable.

Examples

□ This example opens the SASUSER.HOUSES data set and gets the entire tenth observation. The data set identifier value for the open data set is stored in the macro variable MYDATAID. This example nests VARNUM to return the position of the variable in the DDV, and reads in the value of the character variable STYLE.

```
%let mydataid=%sysfunc(open
                  (sasuser.houses,i));
%let rc=%sysfunc(fetchobs(&mydataid,10));
%let style=%sysfunc(getvarc(&mydataid,
                %sysfunc(varnum
                (&mydataid,STYLE))));
%let rc=%sysfunc(close(&mydataid));
```

□ This example assigns VARNUM to a variable that can then be passed as the second argument. This example fetches data from observation 10.

```
%let namenum=%sysfunc(varnum(&mydataid,NAME));
%let rc=%sysfunc(fetchobs(&mydataid,10));
%let user=%sysfunc(getvarc
                  (&mydataid,&namenum));
```

See Also

Functions:
- "FETCH" on page 352
- "FETCHOBS" on page 353
- "GETVARN" on page 391
- "VARNUM" on page 584

GETVARN

Returns the value of a SAS data set numeric variable

Category: SAS File I/O

Syntax

GETVARN(*data-set-id*,*var-num*)

Arguments

data-set-id
specifies the data set identifier that the OPEN function returns.

var-num
is the number of the variable in the Data Set Data Vector (DDV).

Tip: You can obtain this value by using the VARNUM function.

Tip: This value is listed next to the variable when you use the CONTENTS procedure.

Details

Use VARNUM to obtain the number of a variable in a SAS data set. You can nest VARNUM or you can assign it to a variable that can then be passed as the second argument, as shown in the "Examples" section. GETVARN reads the value of a numeric variable from the current observation in the Data Set Data Vector (DDV) into a macro variable or DATA step variable.

Examples

- This example obtains the entire tenth observation from a SAS data set. The data set must have been previously opened using OPEN. The data set identifier value for the open data set is stored in the variable MYDATAID. This example nests VARNUM, and reads in the value of the numeric variable PRICE from the tenth observation of an open SAS data set.

```
%let rc=%sysfunc(fetchobs(&mydataid,10));
%let price=%sysfunc(getvarn(&mydataid,
                   %sysfunc(varnum
                        (&mydataid,price))));
```

- This example assigns VARNUM to a variable that can then be passed as the second argument. This example fetches data from observation 10.

```
%let pricenum=%sysfunc(varnum
                      (&mydataid,price));
%let rc=%sysfunc(fetchobs(&mydataid,10));
%let price=%sysfunc(getvarn
                   (&mydataid,&pricenum));
```

See Also

Functions:

“FETCH” on page 352
“FETCHOBS” on page 353
“GETVARC” on page 390
“VARNUM” on page 584

HBOUND

Returns the upper bound of an array

Category: Array

Syntax

HBOUND<*n*>(*array-name*)

HBOUND(*array-name,bound-n*)

Arguments

n
specifies the dimension for which you want to know the upper bound. If no *n* value is specified, the HBOUND function returns the upper bound of the first dimension of the array.

array-name
specifies the name of an array defined previously in the same DATA step.

bound-n
specifies the dimension for which you want to know the upper bound. Use *bound-n* only if *n* is not specified.

Details

The HBOUND function returns the upper bound of a one-dimensional array or the upper bound of a specified dimension of a multidimensional array. Use HBOUND in array processing to avoid changing the upper bound of an iterative DO group each time you change the bounds of the array. HBOUND and LBOUND can be used together to return the values of the upper and lower bounds of an array dimension.

Comparisons

- □ HBOUND returns the literal value of the upper bound of an array dimension.
- □ DIM always returns a total count of the number of elements in an array dimension.

  Note: This distinction is important when the lower bound of an array dimension has a value other than 1 and the upper bound has a value other than the total number of elements in the array dimension. △

Examples

Example 1: One-dimensional Array In this example, HBOUND returns the upper bound of the dimension, a value of 5. Therefore, SAS repeats the statements in the DO loop five times.

```
array big{5} weight sex height state city;
do i=1 to hbound(big5);
   more SAS statements;
end;
```

Example 2: Multidimensional Array This example shows two ways of specifying the HBOUND function for multidimensional arrays. Both methods return the same value for HBOUND, as shown in the table that follows the SAS code example.

```
array mult{2:6,4:13,2} mult1-mult100;
```

Syntax	Alternative Syntax	Value
HBOUND(MULT)	HBOUND(MULT,1)	6
HBOUND2(MULT)	HBOUND(MULT,2)	13
HBOUND3(MULT)	HBOUND(MULT,3)	2

See Also

Functions:
 “DIM” on page 330
 “LBOUND” on page 434
Statements:
 “ARRAY” on page 755
 “Array Reference” on page 759
“Array Processing ”in *SAS Language Reference: Concepts*

HMS

Returns a SAS time value from hour, minute, and second values

Category: Date and Time

Syntax

HMS(*hour,minute,second*)

Arguments

hour
specifies a SAS expression that represents an integer from 0 through 23.

minute
specifies a SAS expression that represents an integer from 0 through 59.

second
specifies a SAS expression that represents an integer from 0 through 59.

Examples

The following SAS statements produce these results:

SAS Statements	Results
`hrid=hms(12,45,10);`	
`put hrid`	`45910`
`/ hrid time.;`	`2:45:10`

See Also

Functions:

- "DHMS" on page 327
- "HOUR" on page 395
- "MINUTE" on page 445
- "SECOND" on page 542

HOUR

Returns the hour from a SAS time or datetime value

Category: Date and Time

Syntax

HOUR(<*time* | *datetime*>)

Arguments

time
specifies a SAS expression that represents a SAS time value.

datetime
specifies a SAS expression that represents a SAS datetime value.

Details

The HOUR function returns a numeric value that represents the hour from a SAS time or datetime value. Numeric values can range from 0 through 23. HOUR always returns a positive number.

Examples

The following SAS statements produce these results:

SAS Statements	Results
`now='1:30't;`	
`h=hour(now);`	
`put h;`	`1`

See Also

Functions:
- "MINUTE" on page 445
- "SECOND" on page 542

HTMLDECODE

Decodes a string containing HTML numeric character references or HTML character entity references and returns the decoded string

Category: Web Tools

Syntax

HTMLDECODE(*argument*)

Arguments

argument
specifies any character expression.

Details

The HTMLDECODE function recognizes the following character entity references:

Character entity reference ...	decodes as ...
&amp;	&
&lt;	<
&gt;	>
&quot;	"
&nbsp;	(space)

Numeric character references have one of the following forms:

&#nnn;	where *nnn* specifies a decimal number that contains one or more digits.

&#Xnnn;	where *nnn* specifies a hexadecimal number that contains one or more digits.

Operating Environment Information: Numeric character references greater than zero and less than or equal to 255 are converted to the corresponding character in the character set supported by your operating environment. In all operating environments, references greater than 255 are converted to a question mark (?). △

Operating Environment Information: In operating environments that use EBCDIC, SAS performs an extra translation step after it recognizes an HTML numeric character reference. The specified reference is assumed to be an ASCII encoding. SAS uses the transport-to-local translation table to convert this character to an EBCDIC character in operating environments that use EBCDIC. If the translation table does not specify a corresponding EBCDIC character, SAS inserts a question mark (?). For more information see "TRANTAB=" on page 1171. △

Examples

SAS Statements	Results

`x1=htmldecode ('not a &lt;tag&gt;');`
`put x1;`
	`not a <tag>`

`x2=htmldecode ('&amp;');`
`put x2;`
	`&`

`x3=htmldecode ('&#65;&#66;&#67;');`
`put x3;`
	`ABC`

See Also

Function:

"HTMLENCODE" on page 397

HTMLENCODE

Encodes characters using HTML character entity references and returns the encoded string

Category: Web Tools

Syntax

HTMLENCODE(*argument*)

Arguments

argument
specifies any character expression.

Details

HTMLENCODE can encode the following three characters:

Character	encodes as ...
&	&amp;
<	&lt;
>	&gt;

Note: The encoded string may be longer than the original string. Ensure that you consider the additional length when you use this function. △

Examples

SAS Statements	Results
`x1=htmlencode ('not a <tag>'); put x1;`	`not a &lt;tag&gt;`
`x2=htmlencode ('&'); put x2;`	`&amp;`
`x3=htmlencode ('normal text'); put x3;`	`normal text`

See Also

Function:

"HTMLDECODE" on page 396

IBESSEL

Returns the value of the modified bessel function

Category: Mathematical

Syntax

IBESSEL(*nu*,*x*,*kode*)

Arguments

nu

is numeric.

Range: $nu \geq 0$

x
is numeric.

Range: $x \geq 0$

kode
is a nonnegative integer.

Details

The IBESSEL function returns the value of the modified bessel function of order *nu* evaluated at *x* (Abramowitz, Stegun 1964; Amos, Daniel, Weston 1977). When *kode* equals 0, the bessel function is returned. Otherwise, the value of the following function is returned:

$$e^{-x} I_{nm}(x)$$

Examples

SAS Statements	Results
`x=ibessel(2,2,0);`	**0.6889484477**
`x=ibessel(2,2,1);`	**0.0932390333**

INDEX

Searches a character expression for a string of characters

Category: Character

Syntax

INDEX(*source,excerpt*)

Arguments

source
specifies the character expression to search.

excerpt
specifies the string of characters to search for in the character expression.

Tip: Enclose a literal string of characters in quotation marks.

Details

The INDEX function searches *source*, from left to right, for the first occurrence of the string specified in *excerpt*, and returns the position in *source* of the string's first character. If the string is not found in *source*, INDEX returns a value of 0. If there are multiple occurrences of the string, INDEX returns only the position of the first occurrence.

Examples

SAS Statements	Results
`a='ABC.DEF (X=Y)';`	
`b='X=Y';`	
`x=index(a,b);`	
`put x;`	`10`

See Also

Functions:
"INDEXC" on page 400
"INDEXW" on page 401

INDEXC

Searches a character expression for specific characters

Category: Character

Syntax

INDEXC(*source,excerpt-1*<,... *excerpt-n*>)

Arguments

source
specifies the character expression to search.

excerpt
specifies the characters to search for in the character expression.
Tip: If you specify more than one excerpt, separate them with a comma.

Details

The INDEXC function searches *source*, from left to right, for the first occurrence of any character present in the excerpts and returns the position in *source* of that character. If none of the characters in *excerpt-1* through *excerpt-n* in *source* are found, INDEXC returns a value of 0.

Comparisons

The INDEXC function searches for the first occurrence of any individual character that is present within the character string, whereas the INDEX function searches for the first occurrence of the character string as a pattern.

Examples

SAS Statements	Results

`a='ABC.DEP (X2=Y1)';`
`x=indexc(a,'0123',';()=.');`
`put x;`
	`4`

`b='have a good day';`
`x=indexc(b,'pleasant','very');`
`put x;`
	`2`

See Also

Functions:

"INDEX" on page 399

"INDEXW" on page 401

INDEXW

Searches a character expression for a specified string as a word

Category: Character

Syntax

INDEXW(*source, excerpt*)

Arguments

source
specifies the character expression to search.

excerpt
specifies the string of characters to search for in the character expression. SAS removes the leading and trailing blanks from *excerpt*.

Details

The INDEXW function searches *source*, from left to right, for the first occurrence of *excerpt* and returns the position in *source* of the substring's first character. If the substring is not found in *source*, INDEXW returns a value of 0. If there are multiple occurrences of the string, INDEXW returns only the position of the first occurrence.

The substring pattern must begin and end on a word boundary. For INDEXW, word boundaries are blanks, the beginning of *source*, and the end of *source*. Punctuation marks are not word boundaries.

Comparisons

The INDEXW function searches for strings that are words, whereas the INDEX function searches for patterns as separate words or as parts of other words. INDEX searches for any characters present in the excerpts.

Examples

SAS Statements	Results

`s='asdf adog dog';`
`p='dog  ';`
`x=indexw(s,p);`
`put x;`
	`11`

`s='abcdef x=y';`
`p='def';`
`x=indexw(s,p);`
`put x;`
	`0`

See Also

Functions:

"INDEX" on page 399

"INDEXC" on page 400

INPUT

Returns the value produced when a SAS expression that uses a specified informat expression is read

Category: Special

Syntax

INPUT(*source*, <? | ??>*informat.*)

Arguments

source
contains the SAS character expression to which you want to apply a specific informat.

? or ??
The optional question mark (?) and double question mark (??) format modifiers suppress the printing of both the error messages and the input lines when invalid data values are read. The ? modifier suppresses the invalid data message. The ?? modifier also supresses the invalid data message and, in addition, prevents the automatic variable _ERROR_ from being set to 1 when invalid data are read.

informat.
is the SAS informat that you want to apply to the source.

Details

The INPUT function enables you to read the value of *source* by using a specified informat. The informat determines whether the result is numeric or character. Use INPUT to convert character values to numeric values.

Comparisons

The INPUT function returns the value produced when a SAS expression is read using a specified informat. You must use an assignment statement to store that value in a variable. The INPUT statement uses an informat to read a data value and then optionally stores that value in a variable.

Examples

Example 1: Converting Character Values to Numeric Values This example uses the INPUT function to convert a character value to a numeric value and store it in another variable. The COMMA9. informat reads the value of the SALE variable, stripping the commas. The resulting value, 2115353, is stored in FMTSALE.

```
data testin;
   input sale $9.;
   fmtsale=input(sale,comma9.);
   datalines;
2,115,353
;
```

Example 2: Using PUT and INPUT Functions In this example, PUT returns a numeric value as a character string. The value 122591 is assigned to the CHARDATE variable. INPUT returns the value of the character string as a SAS date value using a SAS date informat. The value 11681 is stored in the SASDATE variable.

```
numdate=122591;
chardate=put(numdate,z6.);
sasdate=input(chardate,mmddyy6.);
```

Example 3: Suppressing Error Messages In this example, the question mark (?) format modifier tells SAS not to print the invalid data error message if it finds data errors. The automatic variable _ERROR_ is set to 1 and input data lines are written to the SAS log.

```
y=input(x,? 3.1);
```

Because the double question mark (??) format modifier suppresses printing of error messages and input lines and prevents the automatic variable _ERROR_ from being set to 1 when invalid data are read, the following two examples produce the same result:

- **`y=input(x,?? 2.);`**
- **`y=input(x,? 2.); _error_=0;`**

See Also

Functions:
- "INPUTC" on page 404
- "INPUTN" on page 405
- "PUT" on page 503
- "PUTC" on page 505
- "PUTN" on page 506

Statements:
- "INPUT" on page 876

INPUTC

Enables you to specify a character informat at run time

Category: Special

Syntax

INPUTC(*source*, *informat.*<,*w*>)

Arguments

source
is the SAS expression to which you want to apply the informat.

informat.
is an expression that contains the character informat you want to apply to *source*.

w
specifies a width to apply to the informat.

Interaction: If you specify a width here, it overrides any width specification in the informat.

Comparisons

The INPUTN function enables you to specify a numeric informat at run time.

Examples

Example 1: Specifying Character Informats The PROC FORMAT step in this example creates a format, TYPEFMT., that formats the variable values 1, 2, and 3 with the name of one of the three informats that this step also creates. The informats store responses of "positive," "negative," and "neutral" as different words, depending on the type of question. After PROC FORMAT creates the format and informats, the DATA step creates a SAS data set from raw data consisting of a number identifying the type of question and a response. After reading a record, the DATA step uses the value of TYPE to create a variable, RESPINF, that contains the value of the appropriate informat for the current type of question. The DATA step also creates another variable, WORD, whose value is the appropriate word for a response. The INPUTC function assigns the value of WORD based on the type of question and the appropriate informat.

```
proc format;
   value typefmt 1='$groupx'
                 2='$groupy'
                 3='$groupz';
   invalue $groupx 'positive'='agree'
                   'negative'='disagree'
                   'neutral'='notsure';
   invalue $groupy 'positive'='accept'
                   'negative'='reject'
                   'neutral'='possible';

   invalue $groupz 'positive'='pass'
                   'negative'='fail'
                   'neutral'='retest';
run;

data answers;
   input type response $;
   respinformat = put(type, typefmt.);
   word = inputc(response, respinformat);
   datalines;
1 positive
1 negative
1 neutral
2 positive
2 negative
2 neutral
3 positive
3 negative
3 neutral
;
```

The value of WORD for the first observation is **`agree`**. The value of WORD for the last observation is **`retest`**.

See Also

Functions:
- "INPUT" on page 402
- "INPUTN" on page 405
- "PUT" on page 503
- "PUTC" on page 505
- "PUTN" on page 506

INPUTN

Enables you to specify a numeric informat at run time

Category: Special

Syntax

INPUTN(*source*, *informat.*<,*w*<,*d*>>)

Arguments

source
is the SAS expression to which you want to apply the informat.

informat.
is an expression that contains the numeric informat you want to apply to *source*.

w
specifies a width to apply to the informat.

Interaction: If you specify a width here, it overrides any width specification in the informat.

d
specifies the number of decimal places to use.

Interaction: If you specify a number here, it overrides any decimal-place specification in the informat.

Comparisons

The INPUTC function enables you to specify a character informat at run time.

Examples

Example 1: Specifying Numeric Informats The PROC FORMAT step in this example creates a format, READDATE., that formats the variable values 1 and 2 with the name of a SAS date informat. The DATA step creates a SAS data set from raw data originally from two different sources (indicated by the value of the variable SOURCE). Each source specified dates differently. After reading a record, the DATA step uses the value of SOURCE to create a variable, DATEINF, that contains the value of the appropriate informat for reading the date. The DATA step also creates a new variable, NEWDATE, whose value is a SAS date. The INPUTN function assigns the value of NEWDATE based on the source of the observation and the appropriate informat.

```
proc format;
   value readdate 1='date7.'
                  2='mmddyy8.';
run;

options yearcutoff=1920;

data fixdates (drop=start dateinformat);
   length jobdesc $12;
   input source id lname $ jobdesc $ start $;
   dateinformat=put(source, readdate.);
   newdate = inputn(start, dateinformat);
   datalines;
```

```
1 1604 Ziminski writer 09aug90
1 2010 Clavell editor 26jan95
2 1833 Rivera writer 10/25/92
2 2222 Barnes proofreader 3/26/98
;
```

See Also

Functions:
- "INPUT" on page 402
- "INPUTC" on page 404
- "PUT" on page 503
- "PUTC" on page 505
- "PUTN" on page 506

INT

Returns the integer value

Category: Truncation

Syntax

INT(*argument*)

Arguments

argument
is numeric.

Details

The INT function returns the integer portion of the argument (truncates the decimal portion). If the argument's value is within 10**(-12) of an integer, the function results in that integer. If the value of *argument* is positive, INT(*argument*) has the same result as FLOOR(*argument*). If the value of *argument* is negative, INT(*argument*) has the same result as CEIL(*argument*).

Examples

SAS Statement	Results

`var1=2.1;`
`x=int(var1);`
`put x=;`
	`2`

`var2=-2.4;`
`y=int(var2);`
`put y=;`
	`-2`

`a=int(3);`
`put a=;`
	`3`

`b=int(-1.6);`
`put b=;`
	`-1`

See Also

Functions:
“CEIL” on page 286
“FLOOR” on page 367

INTCK

Returns the integer number of time intervals in a given time span

Category: Date and Time

Syntax

INTCK(’*interval*’,*from*,*to*)

Arguments

’interval’
specifies a character constant or variable. *Interval* can appear in upper- or lowercase. The value of the character constant or variable must be one of those listed in this table:

Date Intervals	Datetime Intervals	Time Intervals
DAY	DTDAY	HOUR
WEEKDAY	DTWEEKDAY	MINUTE
WEEK	DTWEEK	SECOND

Date Intervals	Datetime Intervals	Time Intervals
TENDAY	DTTENDAY	
SEMIMONTH	DTSEMIMONTH	
MONTH	DTMONTH	
QTR	DTQTR	
SEMIYEAR	DTSEMIYEAR	
YEAR	DTYEAR	

Requirement: The type of interval (date, datetime, or time) must match the type of value in *from*.

from
specifies a SAS expression that represents a SAS date, time, or datetime value that identifies the beginning of the time span.

to
specifies a SAS expression that represents a SAS date, time, or datetime value that identifies the end of the time span.

Details

The INTCK function counts intervals by using a fixed starting point for the interval as opposed to counting in multiples of the interval unit. Partial intervals are not counted. For example, WEEK intervals are determined by the number of Sundays that begin between the *from* and the *to*, not by how many seven-day periods fall in between the *from* and the *to*.

Examples

SAS Statements	Results
`qtr=intck('qtr','10jan95'd,'01jul95'd);` `put qtr;`	**2**
`year=intck('year','31dec94'd,` `'01jan95'd);` `put year;`	**1**
`year=intck('year','01jan94'd,` `'31dec94'd);` `put year;`	**0**
`semi=intck('semiyear','01jan95'd,` `'01jan98'd);` `put semi;`	**6**

SAS Statements	Results

`weekvar=intck('week2.2','01jan97'd,`
`        '31mar97'd);`
`put weekvar;`
	6

`wdvar=intck('weekday7w','01jan97'd,`
`        '01feb97'd);`
`put wdvar;`
	26

In the second example, INTCK returns a value of 1 even though only one day has elapsed. This is because the interval from December 31, 1994 to January , 1995 contains the starting point for the YEAR interval. In the third example, however, a value of 0 is returned even though 364 days have elapsed. This is because the period between the two dates does not contain the starting point for the interval.

In the fourth example, SAS returns a value of 6 because the time period contains six semiyearly intervals. (Note that if the ending date were December 31, 1997, SAS would count five intervals.) In the fifth example, SAS returns a value of 6 because there are six two-week intervals beginning on a first Monday during the period. In the sixth example, SAS returns the value 26. That indicates that beginning with January 1, 1997, and counting only Saturdays as weekend days, the period contains 26 weekdays.

See Also

Function:

"INTNX" on page 410

INTNX

Advances a date, time, or datetime value by a given interval, and returns a date, time, or datetime value

Category: Date and Time

Syntax

INTNX(*'interval',start-from,increment*<,*'alignment'*>)

Arguments

'interval'

specifies a character constant or variable. The argument *interval* can appear in upper- or lowercase.

The value of the character constant or variable must be one of those listed in this table:

Date Intervals	Datetime Intervals	Time Intervals
DAY	DTDAY	HOUR
WEEKDAY	DTWEEKDAY	MINUTE
WEEK	DTWEEK	SECOND
TENDAY	DTTENDAY	
SEMIMONTH	DTSEMIMONTH	
MONTH	DTMONTH	
QTR	DTQTR	
SEMIYEAR	DTSEMIYEAR	
YEAR	DTYEAR	

Requirement: The type of interval (date, datetime, or time) must match the type of value in *start-from* and *increment*.

start-from
specifies a SAS expression that represents a SAS date, time, or datetime value identifying a starting point.

increment
specifies a negative or positive integer that represents the specific number of time *intervals*.

'*alignment*'
specifies one of these values:

BEGINNING
specifies that the returned date is aligned to the beginning of the interval.

Alias: B

MIDDLE
specifies that the returned date is aligned to the midpoint of the interval.

Alias: M

END
specifies that the returned date is aligned to the end of the interval.

Alias: E

Default: BEGINNING

Examples

SAS Statements	Results

`yr=intnx('year','05feb94'd,3);`
`put yr / yr date7.;`
	13515
	01JAN97

`x=intnx('month','05jan95'd,0);`
`put x / x date7.;`
	12784
	01JAN95

`next=intnx('semiyear','01jan97'd,1);`
`put next / next date7.;`
	13696
	01JUL97

`past=intnx('month2','01aug96'd,-1);`
`put past / past date7;`
	13270
	01MAY96

`sm=intnx('semimonth2.2',`
`'01apr97'd,4);`
`put sm / sm date7.;`
	13711
	6JUL97

These examples illustrate advancing a date using an alignment value:

SAS Statements	Results

`date1=intnx('month','01jan95'd,5,`
`      'beginning');`
` put date1 / date1 date7.;`
	12935
	01JUN95

`date2=intnx('month','01jan95'd,5,`
`      'middle');`
`put date2 / date2 date7.;`
	12949
	5JUN95

`date3=intnx('month','01jan95'd,5,`
`      'end');`
`put date3 / date3 date7.;`
	12964
	30JUN95

See Also

Function:

“INTCK” on page 408

INTRR

Returns the internal rate of return as a fraction

Category: Financial

Syntax

INTRR(*freq*,*c0*, *c1*, . . . ,*cn*)

Arguments

freq

is numeric, the number of payments over a specified base period of time that is associated with the desired internal rate of return.

Range: *freq* > 0

Exception: The case *freq* = 0 is a flag to allow continuous compounding.

c0,c1, . . . ,cn

are numeric, the optional cash payments.

Details

The INTRR function returns the internal rate of return over a specified base period of time for the set of cash payments *c0*, *c1*, ..., *cn*. The time intervals between any two consecutive payments are assumed to be equal. The argument *freq*>0 describes the number of payments that occur over the specified base period of time.

The internal rate of return is the interest rate such that the sequence of payments has a 0 net present value (see the NETPV function). It is given by

$$r = \begin{cases} \frac{1}{x^{freq}} & freq > 0 \\ -log_e(x) & freq = 0 \end{cases}$$

where *x* is the real root, nearest to 1, of the polynomial

$$\sum_{i=0}^{n} c_i x^i = 0$$

The routine uses Newton's method to look for the internal rate of return nearest to 0. Depending on the value of payments, a root for the equation does not always exist; in that case, a missing value is returned.

Missing values in the payments are treated as 0 values. When *freq*>0, the computed rate of return is the effective rate over the specified base period. To compute a quarterly internal rate of return (the base period is three months) with monthly payments, set *freq* to 3.

If *freq* is 0, continuous compounding is assumed and the base period is the time interval between two consecutive payments. The computed internal rate of return is the nominal rate of return over the base period. To compute with continuous compounding and monthly payments, set *freq* to 0. The computed internal rate of return will be a monthly rate.

Examples

For an initial outlay of $400 and expected payments of $100, $200, and $300 over the following three years, the annual internal rate of return can be expressed as

```
rate=intrr(1,-400,100,200,300);
```

The value returned is 0.19438.

IORCMSG

Returns a formatted error message for _IORC_

Category: SAS File I/O

Syntax

character-variable=**IORCMSG**()

Arguments

character-variable
specifies a character variable.

Tip: If the length has been previously defined, the result will be truncated or padded as needed.

Default: The default length is 200 characters.

Details

The IORCMSG function returns the formated error message associated with the current value of the automatic variable _IORC_ . The _IORC_ variable is created when you use the MODIFY statement, or when you use the SET statement with the KEY= option.

Examples

In the following program, observations are either rewritten or added to the updated master file that contains bank accounts and current bank balance. The program queries the _IORC_ variable and returns a formatted error message if the _IORC_ value is unexpected.

```
libname bank 'SAS-data-library';

data bank.master;
   set bank.trans;
   modify bank.master key=Accountnum;
   if (_IORC_ EQ %sysrc(_SOK)) then
      do;
         balance=balance+deposit;
         replace;
      end;
else
   if (_IORC_ = %sysrc(_DSENOM)) then
      do;
         balance=deposit;
         output;
         _error_=0;
      end;
```

```
else
   do;
      errmsg=IORCMSG();
      put 'Unknown error condition:'
      errmsg;
   end;
run;
```

IRR

Returns the internal rate of return as a percentage

Category: Financial

Syntax

IRR(*freq*,*c0*,*cl*, . . . ,*cn*)

Arguments

freq
is numeric, the number of payments over a specified base period of time associated with the desired internal rate of return.

Range: *freq* > 0.

Exception: The case *freq* = 0 is a flag to allow continuous compounding.

c0,c1, . . . ,cn
are numeric, the optional cash payments.

Comparisons

The IRR function is identical to INTRR, except that the rate returned is a percentage.

JBESSEL

Returns the value of the bessel function

Category: Mathematical

Syntax

JBESSEL(*nu*,*x*)

Arguments

nu
is numeric.

Range: *nu* ≥ 0

x
is numeric.

Range: *x* ≥ 0

Details

The JBESSEL function returns the value of the bessel function of order *nu* evaluated at *x* (For more information, see Abramowitz and Stegun 1964; Amos, Daniel, and Weston 1977).

Examples

SAS Statements	Results
`x=jbessel(2,2);`	`0.3528340286`

JULDATE

Returns the Julian date from a SAS date value

Category: Date and Time

Syntax

JULDATE(*date*)

Arguments

date
specifies a SAS date value.

Details

The JULDATE function converts a SAS date value to a five- or seven-digit Julian date. If *date* falls within the 100-year span defined by the system option YEARCUTOFF=, the result has five digits: the first two digits represent the year, and the next three digits represent the day of the year (1 to 365, or 1 to 366 for leap years). Otherwise, the result has seven digits: the first four digits represent the year, and the next three digits represent the day of the year. For example, if YEARCUTOFF=1920, JULDATE would return 97001 for January 1, 1997, and return 1878365 for December 31, 1878.

Comparisons

The function JULDATE7 is similar to JULDATE except that JULDATE7 always returns a four digit year. Thus JULDATE7 is year 2000 compliant because it eliminates the need to consider the implications of a two digit year.

Examples

The following SAS statements produce these results:

SAS Statements	Results
`julian=juldate('31dec99'd);`	**99365**
`julian=juldate('01jan2099'd);`	**2099001**

See Also

Function:
    "DATEJUL" on page 313
    "JULDATE7" on page 417
System Option:
    "YEARCUTOFF=" on page 1177

JULDATE7

Returns a seven-digit Julian date from a SAS date value

Category: Date and Time

Syntax

JULDATE7(*date*)

Arguments

date
specifies a SAS date value.

Details

The JULDATE7 function returns a seven digit Julian date from a SAS date value. The first four digits represent the year, and the next three digits represent the day of the year.

Comparisons

The function JULDATE7 is similar to JULDATE except that JULDATE7 always returns a four digit year. Thus JULDATE7 is year 2000 compliant because it eliminates the need to consider the implications of a two digit year.

Examples

The following SAS statements produce these results:

SAS Statements	Results
`julian=juldate7('31dec96'd);`	**1996366**
`julian=juldate7('01jan2099'd);`	**2099001**

See Also

Function:
“JULDATE” on page 416

KCOMPARE

Returns the result of a comparison of character strings

Category: DBCS

Syntax

KCOMPARE(*source*,<*pos*, <*count*,>>*findstr*)

Arguments

source
specifies the character string to be compared.

pos
specifies the starting position in *source* to begin the comparison. If *pos* is omitted, the entire *source* is compared. If *pos* is less than 0, *source* is assumed as extended DBCS data that does not contain any SO/SI characters.

count
specifies the number of bytes to compare. If *count* is omitted, all of *source* that follows *pos* is compared, except for any trailing blanks.

findstr
specifies the character string to compare to *source*.

Details

KCOMPARE returns

- -1 if *source* is greater than *findstr*
- 0 if *source* is equal to *findstr*
- 1 if *source* is less than *findstr*.

KCOMPRESS

Removes specific characters from a character string

Category: DBCS

Syntax

KCOMPRESS(*source*<, *characters-to-remove*>)

Arguments

source
specifies a character string that contains the characters to remove. When only *source* is specified, KCOMPRESS returns this string with all of the single- and double-byte blanks removed.

characters-to-remove
specifies the character or characters that KCOMPRESS removes from the character string.

Tip: Enclose a literal string of characters in quotation marks.

See Also

Functions:

“KLEFT” on page 421

“KTRIM” on page 427

KCOUNT

Returns the number of double-byte characters in a string

Category: DBCS

Syntax

KCOUNT(*source*)

Arguments

source
specifies the character string to count.

KINDEX

Searches a character expression for a string of characters

Category: DBCS

Syntax

KINDEX(*source, excerpt*)

Arguments

source
specifies the character expression to search.

excerpt
specifies the string of characters to search for in the character expression.

Tip: Enclose a literal string of characters in quotation marks.

Details

The KINDEX function searches *source*, from left to right, for the first occurrence of the string specified in *excerpt*, and returns the position in *source* of the string's first character. If the string is not found in *source*, KINDEX returns a value of 0. If there are multiple occurrences of the string, KINDEX returns only the position of the first occurrence.

See Also

Functions:
"KINDEXC" on page 420

KINDEXC

Searches a character expression for specific characters

Category: DBCS

Syntax

KINDEXC(*source,excerpt-1<,... excerpt-n>*)

Arguments

source
specifies the character expression to search.

excerpt
specifies the characters to search for in the character expression.

Tip: If you specify more than one excerpt, separate them with a comma.

Details

The KINDEXC function searches *source*, from left to right, for the first occurrence of any character present in the excerpts and returns the position in *source* of that character. If none of the characters in *excerpt-1* through *excerpt-n* in *source* are found, KINDEXC returns a value of 0 .

Comparisons

The KINDEXC function searches for the first occurrence of any individual character that is present within the character string, whereas the KINDEX function searches for the first occurrence of the character string as a pattern.

See Also

Functions:
  "KINDEX" on page 419

KLEFT

Left aligns a SAS character expression by removing unnecessary leading DBCS blanks and SO/SI

Category: DBCS

Syntax

KLEFT(*argument*)

Arguments

argument
specifies any SAS character expression.

Details

KLEFT returns an argument with leading blanks moved to the end of the value. The argument's length does not change.

See Also

Functions:
  "KCOMPRESS" on page 418
  "KRIGHT" on page 423
  "KTRIM" on page 427

KLENGTH

Returns the length of an argument

Category: DBCS

Syntax

KLENGTH(*argument*)

Arguments

argument
specifies any SAS expression.

Details

The KLENGTH function returns an integer that represents the position of the right-most nonblank character in the argument. If the value of the argument is missing, KLENGTH returns a value of 1. If the argument is an uninitialized numeric variable, KLENGTH returns a value of 12 and prints a note in the SAS log that the numeric values have been converted to character values.

KLOWCASE

Converts all letters in an argument to lowercase

Category: DBCS

Syntax

KLOWCASE(*argument*)

Arguments

argument
specifies any SAS character expression.

Details

The KLOWCASE function copies a character argument, converts all uppercase letters to lowercase letters, and returns the altered value as a result.

KREVERSE

Reverses a character expression

Category: DBCS

Syntax

KREVERSE(*argument*)

Arguments

argument
specifies any SAS character expression.

KRIGHT

Right aligns a character expression by trimming trailing DBCS blanks and SO/SI

Category: DBCS

Syntax

KRIGHT(*argument*)

Arguments

argument
specifies any SAS character expression.

Details

The KRIGHT function returns an argument with trailing blanks moved to the start of the value. The argument's length does not change.

See Also

Functions:
“KCOMPRESS” on page 418
“KLEFT” on page 421
“KTRIM” on page 427

KSCAN

Selects a given word from a character expression

Category: DBCS

Syntax

KSCAN(*argument,n*<, *delimiters*>)

Arguments

argument
specifies any character expression.

n
specifies a numeric expression that produces the number of the word in the character string you want KSCAN to select.

Tip: If *n* is negative, KSCAN selects the word in the character string starting from the end of the string. If |*n*| is greater than the number of words in the character string, KSCAN returns a blank value.

delimiters
specifies a character expression that produces characters that you want KSCAN to use as word separators in the character string.

Default: If you omit *delimiters* in an ASCII environment, SAS uses the following characters:

blank . < (+ & ! $ *); ^ – / , % |

In ASCII environments without the ^ character, KSCAN uses the ~ character instead.

If you omit *delimiters* on an EBCDIC environment, SAS uses the following characters:

blank . < (+ | & ! $ *); ¬ – / , % | ¢

Tip: If you represent *delimiters* as a constant, enclose *delimiters* in quotation marks.

Details

Leading delimiters before the first word in the character string do not effect KSCAN. If there are two or more contiguous delimiters, KSCAN treats them as one.

KSTRCAT

Concatenates two or more character strings

Category: DBCS

Syntax

KSTRCAT(*argument-1, argument-2*<, ... *argument-n*>)

Arguments

argument
specifies any single-byte or double-byte character string.

Details

KSTRCAT concatenates two or more single-byte or double-byte character strings. It also removes unnecessary SO/SI pairs between the strings.

KSUBSTR

Extracts a substring from an argument

Category: DBCS

Syntax

KSUBSTR(*argument,position*<,*n*>)

Arguments

argument
specifies any SAS character expression.

position
specifies a numeric expression that is the beginning character position.

n
specifies a numeric expression that is the length of the substring to extract.

Interaction: If *n* is larger than the length of the expression that remains in *argument* after *position*, SAS extracts the remainder of the expression.

Tip: If you omit *n*, SAS extracts the remainder of the expression.

Details

The KSUBSTR function returns a portion of an expression that you specify in *argument*. The portion begins with the character specified by *position* and is the number of characters specified by *n*.

A variable that is created by KSUBSTR obtains its length from the length of *argument*.

See Also

Function:
“KSUBSTRB” on page 425

KSUBSTRB

Extracts a substring from an argument based on byte position

Category: DBCS

Syntax

KSUBSTRB(*argument,position*<,*n*>)

Arguments

argument
specifies any SAS character expression.

position
specifies the beginning character position in byte units.

n
specifies the length of the substring to extract in byte units.

Interaction: If *n* is larger than the length (in byte units) of the expression that remains in *argument* after *position*, SAS extracts the remainder of the expression.

Tip: If you omit *n*, SAS extracts the remainder of the expression.

Details

The KSUBSTRB function returns a portion of an expression that you specify in *argument*. The portion begins with the byte unit specified by *position* and is the number of byte units specified by *n*.

A variable that is created by KSUBSTRB obtains its length from the length of *argument*.

See Also

Function:
“KSUBSTR” on page 425

KTRANSLATE

Replaces specific characters in a character expression

Category: DBCS

Syntax

KTRANSLATE(*source,to-1,from-1*<,…*to-n,from-n*>)

Arguments

source
specifies the SAS expression that contains the original character value.

to

specifies the characters that you want KTRANSLATE to use as substitutes.

from

specifies the characters that you want KTRANSLATE to replace.

Interaction: Values of *to* and *from* correspond on a character-by-character basis; KTRANSLATE changes character one of *from* to character one of *to*, and so on. If *to* has fewer characters than *from*, KTRANSLATE changes the extra *from* characters to blanks. If *to* has more characters than *from*, KTRANSLATE ignores the extra *to* characters.

Operating Environment Information: You must have pairs of *to* and *from* arguments on some operating environments. On other operating environments, a segment of the collating sequence replaces null *from* arguments. See the SAS documentation for your operating environment for more information. △

Details

You can use KTRANSLATE to translate a single-byte character expression to a double-byte character expression, or translate a double-byte character expression to a single-byte character expression.

The maximum number of pairs of *to* and *from* arguments that KTRANSLATE accepts depends on the operating environment you use to run SAS. There is no functional difference between using several pairs of short arguments, or fewer pairs of longer arguments.

KTRIM

Removes trailing DBCS blanks and SO/SI from character expressions

Category: DBCS

Syntax

KTRIM(*argument*)

Arguments

argument

specifies any SAS character expression.

Details

KTRIM copies a character argument, removes all trailing blanks, and returns the trimmed argument as a result. If the argument is blank, KTRIM returns one blank. KTRIM is useful for concatenating because concatenation does not remove trailing blanks.

Assigning the results of KTRIM to a variable does not affect the length of the receiving variable. If the trimmed value is shorter than the length of the receiving variable, SAS pads the value with new blanks as it assigns it to the variable.

See Also

Functions:
- "KCOMPRESS" on page 418
- "KLEFT" on page 421
- "KRIGHT" on page 423

KTRUNCATE

Truncates a numeric value to a specified length

Category: DBCS

Syntax

KTRUNCATE(*number*,*length*)

Arguments

number
is numeric.

length
is numeric and integer.

Details

The KTRUNCATE function truncates a full-length *number* (stored as a double) to a smaller number of bytes, as specified in *length* and pads the truncated bytes with 0s. The truncation and subsequent expansion duplicate the effect of storing numbers in less than full length and then reading them.

KUPCASE

Converts all single-byte letters in an argument to uppercase

Category: DBCS

Syntax

KUPCASE(*argument*)

Arguments

argument
specifies any SAS character expression.

Details

The KUPCASE function copies a character argument, converts all single-byte lowercase letters to uppercase letters, and returns the altered value as a result.

KUPDATE

Inserts, deletes, and replaces character value contents

Category: DBCS

Syntax

KUPDATE(*argument,position,n*<, *characters-to-replace*>)

KUPDATE(*argument,position*<,*n*>, *characters-to-replace*)

Arguments

argument
specifies a character variable.

position
specifies a numeric expression that is the beginning character position.

n
specifies a numeric expression that is the length of the substring to be replaced.

Restriction: *n* can not be larger than the length of the expression that remains in *argument* after *position*.

Restriction: *n* is optional, but you cannot omit both *n* and *characters-to-replace* from the function.

Tip: If you omit *n*, SAS uses all of the characters in *characters-to-replace* to replace the values of *argument*.

characters-to-replace
specifies a character expression that will replace the contents of *argument*.

Restriction: *characters-to-replace* is optional, but you cannot omit both *characters-to-replace* and *n* from the function.

Tip: Enclose a literal string of characters in quotation marks.

Details

The KUPDATE function replaces the value of *argument* with the expression in *characters-to-replace*. KUPDATE replaces *n* characters starting at the character you specify in *position*.

See Also

Function:

“KUPDATEB” on page 430

KUPDATEB

Inserts, deletes, and replaces character value contents based on byte unit

Category: DBCS

Syntax

KUPDATEB(*argument,position,n*<*,characters-to-replace*>)

KUPDATEB(*argument,position* <, *n*>, *characters-to-replace*)

Arguments

argument
specifies a character variable.

position
specifies the beginning character position in byte units.

n
specifies the length of the substring to be replaced in byte units.

Restriction: *n* can not be larger than the length (in bytes) of the expression that remains in *argument* after *position*.

Restriction: *n* is optional, but you cannot omit both *n* and *characters-to-replace* from the function.

Tip: If you omit *n*, SAS uses all of the characters in *characters-to-replace* to replace the values of *argument*.

characters-to-replace
specifies a character expression to replace the contents of *argument*.

Restriction: *characters-to-replace* is optional, but you cannot omit both *characters-to-replace* and *n* from the function.

Tip: Enclose a literal string of characters in quotation marks.

Details

The KUPDATEB function replaces the value of *argument* with the expression in *characters-to-replace*. KUPDATEB replaces *n* byte units starting at the byte unit that you specify in *position*.

See Also

Function:
  "KUPDATE" on page 429

KURTOSIS

Returns the kurtosis

Category: Descriptive Statistics

Syntax

KURTOSIS(*argument,argument, . . .*)

Arguments

argument
is numeric. At least four arguments are required. The argument list may consist of a variable list, which is preceded by OF.

Examples

SAS Statements	Results
`x1=kurtosis(5,9,3,6);`	`0.928`
`x2=kurtosis(5,8,9,6,.);`	`-3.3`
`x3=kurtosis(8,9,6,1);`	`1.5`
`x4=kurtosis(8,1,6,1);`	`-4.483379501`
`x5=kurtosis(of x1-x4);`	`-5.065692754`

KVERIFY

Returns the position of the first character that is unique to an expression

Category: DBCS

Syntax

KVERIFY(*source,excerpt-1<,...excerpt-n>*)

Arguments

source
specifies any SAS character expression.

excerpt
specifies any SAS character expression. If you specify more than one excerpt, separate them with a comma.

Details

The KVERIFY function returns the position of the first character in *source* that is not present in any *excerpt*. If KVERIFY finds every character in *source* in at least one *excerpt*, it returns a 0.

LAG

Returns values from a queue

Category: Special

Syntax

LAG<*n*>(*argument*)

Arguments

n
specifies the number of lagged values.

argument
is numeric or character.

Details

The LAG functions, LAG1, LAG2, . . . , LAG100 return values from a queue. LAG1 can also be written as LAG. A LAG*n* function stores a value in a queue and returns a value stored previously in that queue. Each occurrence of a LAG*n* function in a program generates its own queue of values.

The queue for LAG*n* is initialized with *n* missing values, where *n* is the length of the queue (for example, a LAG2 queue is initialized with two missing values). When LAG*n* is executed, the value at the top of the queue is removed and returned, the remaining values are shifted upwards, and the new value of the argument is placed at the bottom of the queue. Hence, missing values are returned for the first *n* executions of LAG*n*, after which the lagged values of the argument begin to appear.

Note: Storing values at the bottom of the queue and returning values from the top of the queue occurs only when the function is executed. A LAG*n* function that is executed conditionally will store and return values only from the observations for which the condition is satisfied. See Example 2 on page 433 . △

If the argument of LAG*n* is an array name, a separate queue is maintained for each variable in the array.

Examples

Example 1: Creating a Data Set The following program creates a data set that contains the values for X, Y, and Z.

```
options pagesize=25 linesize=64 nodate pageno=1;


data one;
   input X @@;
   Y=lag1(x);
   Z=lag2(x);
   datalines;
1 2 3 4 5 6
;
proc print;
   title 'Lag Output';
run;
```

```
                          Lag Output                          1

                      Obs     X     Y     Z

                       1      1     .     .
                       2      2     1     .
                       3      3     2     1
                       4      4     3     2
                       5      5     4     3
                       6      6     5     4
```

LAG1 returns one missing value and the values of X (lagged once). LAG2 returns two missing values and the values of X (lagged twice).

Example 2: Storing Every Other Lagged Value This example shows the difference in output when you use conditional and unconditional logic in your program. Because the LAG function stores values on the queue only when it is called, you must call LAG unconditionally to get the correct answers.

```
options pagesize=25 linesize=64 nodate pageno=1;

title 'Store Every Other Lagged Value';

data test;
   input x @@;
   if mod(x,2)=0 then a=lag(x);
   b=lag(x);
   if mod(x,2)=0 then c=b;
   label a='(WRONG) a' c='(RIGHT) c';
   datalines;
1 2 3 4 5 6 7 8
;

proc print label data=test;
run;
```

```
                 Store Every Other Lagged Value              1

                          (WRONG)          (RIGHT)
             Obs     x       a        b       c

              1      1       .        .       .
              2      2       .        1       1
              3      3       .        2       .
              4      4       2        3       3
              5      5       .        4       .
              6      6       4        5       5
              7      7       .        6       .
              8      8       6        7       7
```

See Also

Function:
  "DIF" on page 328

LBOUND

Returns the lower bound of an array

Category: Array

Syntax

LBOUND<*n*>(*array-name*)

LBOUND(*array-name*,*bound-n*)

Arguments

n
specifies the dimension for which you want to know the lower bound. If no *n* value is specified, the LBOUND function returns the lower bound of the first dimension of the array.

array-name
specifies the name of an array defined previously in the same DATA step.

bound-n
specifies the dimension for which you want to know the lower bound. Use *bound-n* only if *n* is not specified.

Details

The LBOUND function returns the lower bound of a one-dimensional array or the lower bound of a specified dimension of a multidimensional array. Use LBOUND in array processing to avoid changing the lower bound of an iterative DO group each time you change the bounds of the array. LBOUND and HBOUND can be used together to return the values of the lower and upper bounds of an array dimension.

Examples

Example 1: One-dimensional Array In this example, LBOUND returns the lower bound of the dimension, a value of 2. SAS repeats the statements in the DO loop five times.

```
array big{2:6} weight sex height state city;
do i=lbound(big) to hbound(big);
   ...more SAS statements...;
end;
```

Example 2: Multidimensional Array This example shows two ways of specifying the LBOUND function for multidimensional arrays. Both methods return the same value for LBOUND, as shown in the table that follows the SAS code example.

```
array mult{2:6,4:13,2} mult1-mult100;
```

Syntax	Alternative Syntax	Value
LBOUND(MULT)	LBOUND(MULT,1)	2
LBOUND2(MULT)	LBOUND(MULT,2)	4
LBOUND3(MULT)	LBOUND(MULT,3)	1

See Also

Functions:
- “DIM” on page 330
- “HBOUND” on page 392

Statements:
- “ARRAY” on page 755
- “Array Reference” on page 759

“Array Processing” in *SAS Language Reference: Concepts*

LEFT

Left aligns a SAS character expression

Category: Character

Syntax

LEFT(*argument*)

Arguments

argument
specifies any SAS character expression.

Details

LEFT returns an argument with leading blanks moved to the end of the value. The argument's length does not change.

Examples

SAS Statements	Results
	`----+----1----+`
`a='  DUE DATE';`	
`b=left(a);`	
`put a;`	`DUE DATE`

See Also

Functions:
- "COMPRESS" on page 296
- "RIGHT" on page 524
- "TRIM" on page 570

LENGTH

Returns the length of an argument

Category: Character

Syntax

LENGTH(*argument*)

Arguments

argument
specifies any SAS expression.

Details

The LENGTH function returns an integer that represents the position of the right-most nonblank character in the argument. If the value of the argument is missing, LENGTH returns a value of 1. If the argument is a numeric variable (either initialized or uninitialized), LENGTH returns a value of 12 and prints a note in the SAS log that the numeric values have been converted to character values.

Examples

SAS Statements	Results

`len=length('ABCDEF');`
`put len;`
	6

LGAMMA

Returns the natural logarithm of the Gamma function

Category: Mathematical

Syntax

LGAMMA(*argument*)

Arguments

argument
is numeric.

Range: must be positive.

Examples

SAS Statements	Results
`x=lgamma(2);`	**0**
`x=lgamma(1.5);`	**-0.120782238**

LIBNAME

Assigns or deassigns a libref for a SAS data library and returns a value

Category: SAS File I/O

Syntax

LIBNAME(*libref*<,*SAS-data-library*<,*engine*<,*options*>>>)

Arguments

libref
specifies the libref that is assigned to a SAS data library.

SAS-data-library
specifies the physical name of the SAS data library that is associated with the libref. Specify this name as required by the host operating environment.

engine
specifies the engine that is used to access SAS files opened in the data library. If you are specifying a SAS/SHARE server, then the engine should be REMOTE.

options
names one or more options honored by the specified engine, delimited with blanks.

Details

LIBNAME returns 0 if the operation was successful, ≠ 0 if it was not successful.

Operating Environment Information: Some systems allow a *SAS-data-library* value of `' '`(with a space) to assign a libref to the current directory. The behavior of LIBNAME when a single space is specified for *SAS-data-library* is host dependent.
If no value is provided for *SAS-data-library* or if *SAS-data-library* has a value of `''`(with no space), LIBNAME disassociates the libref from the data library.
Under some operating environments, the user can assign librefs using system commands outside the SAS session. △

Examples

- This example attempts to assign the libref NEW to the SAS data library MYLIB. If an error or warning occurs, the message is written to the SAS log. Note that in a macro statement you do not enclose character strings in quotation marks.

```
%if (%sysfunc(libname(new,MYLIB))) %then
  %put %sysfunc(sysmsg());
```

- This example deassigns the libref NEW that has been previously associated with the data library MYLIB in the preceding example. If an error or warning occurs, the message is written to the SAS log. Note that in a macro statement you do not enclose character strings in quotation marks.

```
%if (libname(new)) %then
  %put %sysfunc(sysmsg());
```

See Also

Function:
“LIBREF” on page 438

LIBREF

Verifies that a libref has been assigned and returns a value

Category: SAS File I/O

Syntax

LIBREF(*libref*)

Arguments

libref
specifies the libref to be verified.

Details

LIBREF returns 0 if the operation was successful, ≠0 if it was not successful.

Examples

This example verifies a libref. If an error or warning occurs, the message is written to the log. Under some operating environments, the user can assign librefs by using system commands outside the SAS session.

```
%if (%sysfunc(libref(sashelp))) %then
  %put %sysfunc(sysmsg());
```

See Also

Function:
“LIBNAME” on page 437

LOG

Returns the natural (base e) logarithm

Category: Mathematical

Syntax

LOG(*argument*)

Arguments

argument
is numeric.

Range: must be positive.

Examples

SAS Statements	Results
`x=log(1.0);`	**0**
`x=log(10.0);`	**2.302585093**

LOG10

Returns the logarithm to the base 10

Category: Mathematical

Syntax

LOG10(*argument*)

Arguments

argument
is numeric.
Range: must be positive.

Examples

SAS Statements	Results
`x=log10(1.0);`	**0**
`x=log10(10.0);`	**1**
`x=log10(100.0);`	**2**

LOG2

Returns the logarithm to the base 2

Category: Mathematical

Syntax

LOG2(*argument*)

Arguments

argument
is numeric.

Range: must be positive (>0).

Examples

SAS Statements	Results
`x=log2(2.0);`	`1`
`x=log2(0.5);`	`-1`

LOGPDF

Computes the logarithm of a probability (mass) function

Category: Probability
Alias: LOGPMF
See: "PDF" on page 464

Syntax

LOGPDF(*'dist',quantile,parm-1, . . . ,parm-k*)

The LOGPDF function computes the logarithm of the probability density (mass) function.

LOGSDF

Computes the logarithm of a survival function

Category: Probability
See: "CDF" on page 273

Syntax

LOGSDF(*'dist',quantile,parm-1, . . . ,parm-k*)

The LOGSDF function computes the logarithm of the survival function.

LOWCASE

Converts all letters in an argument to lowercase

Category: Character

Syntax

LOWCASE(*argument*)

Arguments

argument
specifies any SAS character expression.

Details

The LOWCASE function copies a character argument, converts all uppercase letters to lowercase letters, and returns the altered value as a result.

Examples

SAS Statements	Results
`x='INTRODUCTION';`	
`y=lowcase(x);`	
`put y;`	`introduction`

MAX

Returns the largest value

Category: Descriptive Statistics

Syntax

MAX(*argument,argument, . . .*)

Arguments

argument
is numeric. At least two arguments are required. The argument list may consist of a variable list, which is preceded by OF.

Comparisons

The MAX function returns a missing value (.) only if all arguments are missing.

The MAX operator (<>) returns a missing value only if both operands are missing. In this case, it returns the value of the operand that is higher in the sort order for missing values.

Examples

SAS Statements	Results
`x=max(8,3);`	**8**
`x1=max(2,6,.);`	**6**
`x2=max(2.-3,1,-1);`	**2**
`x3=max(3,.,-3);`	**3**
`x4=max(of x1-x3);`	**6**

MDY

Returns a SAS date value from month, day, and year values

Category: Date and Time

Syntax

MDY(*month*,*day*,*year*)

Arguments

month
specifies a numeric expression that represents an integer from 1 through 12.

day
specifies a numeric expression that represents an integer from 1 through 31.

year
specifies a numeric expression that represents an integer identifying a specific year. Use the YEARCUTOFF system option to define the year range.

Examples

SAS Statements	Result

`m=8;`
`d=27;`
`y=90;`
`birthday=mdy(m,d,y);`
`put birthday= worddate.;`
	`birthday=August 27, 1990`

See Also

Functions:
- "DAY" on page 315
- "MONTH" on page 451
- "YEAR" on page 618

MEAN

Returns the arithmetic mean (average)

Category: Descriptive Statistics

Syntax

MEAN(*argument*,*argument*, . . .)

Arguments

argument
is numeric. At least one argument is required. The argument list may consist of a variable list, which is preceded by OF.

Examples

SAS Statements	Results
`x1=mean(2,.,.,6);`	**4**
`x2=mean(1,2,3,2);`	**2**
`x3=mean(of x1-x2);`	**3**

MIN

Returns the smallest value

Category: Descriptive Statistics

Syntax

MIN(*argument,argument*, . . .)

Arguments

argument
is numeric. At least two arguments are required. The argument list may consist of a variable list, which is preceded by OF.

Comparisons

The MIN function returns a missing value (.) only if all arguments are missing.

The MIN operator (><) returns a missing value only if either operand is missing. In this case, it returns the value of the operand that is lower in the sort order for missing values.

Examples

SAS Statements	Results
`x=min(7,4);`	**4**
`x1=min(2,.,6);`	**2**
`x2=min(2,-3,1,-1);`	**-3**
`x3=min(0,4);`	**0**
`x4=min(of x1-x3);`	**-3**

MINUTE

Returns the minute from a SAS time or datetime value

Category: Date and Time

Syntax

MINUTE(*time* | *datetime*)

Arguments

time
specifies a SAS expression that represents a SAS time value.

datetime
specifies a SAS expression that represents a SAS datetime value.

Details

The MINUTE function returns an integer that represents a specific minute of the hour. MINUTE always returns a positive number. Missing values are ignored.

Examples

SAS Statements	Results
`time='3:19:24't;`	
`m=minute(time);`	
`put m;`	`19`

See Also

Functions:

"HOUR" on page 395

"SECOND" on page 542

MISSING

Returns a numeric result that indicates whether the argument contains a missing value

Category: Descriptive Statistics

Category: Character

Syntax

MISSING(*numeric-expression* | *character-expression*)

Arguments

numeric-expression
specifies numeric data.

character-expression
is the name of a character variable or an expression that evaluates to a character value.

Details

- The MISSING function checks a numeric or character expression for a missing value, and returns a numeric result. If the argument does not contain a missing value, SAS returns a value of 0. If the argument contains a missing value, SAS returns a value of 1.
- A *character-expression* is defined as having a missing value if the result of the expression contains all blank spaces.

□ A *numeric-expression* is defined as having a missing value if the result of the expression is missing (.), or if the expression contains special characters you used to differentiate among missing values. The special characters are the letters A through Z and the underscore, preceded by a period.

Comparisons

The NMISS function requires a numeric argument and returns the number of missing values in the list of arguments.

Examples

This example uses the MISSING function to check whether the input variables contain missing values.

```
data values;
   input @1 var1 3. @5 var2 3.;
   if missing(var1) then
      do;
         put 'Variable 1 is Missing.';
      end;
   else if missing(var2) then
      do;
         put 'Variable 2 is Missing.';
      end;
   datalines;
127
988 195
;
```

In this example, the following message appears in the SAS log.

```
Variable 2 is Missing.
```

See Also

Function:
“NMISS” on page 457

MOD

Returns the remainder value

Category: Mathematical

Syntax

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

Arguments

argument-1
is numeric.

argument-2
is numeric.

Restriction: cannot be 0

Details

The MOD function returns the remainder when the integer quotient of *argument-1* divided by *argument-2* is calculated.

Examples

SAS Statement	Results
`x1=mod(6,3);`	`0`
`x2=mod(10,3);`	`1`
`x3=mod(11,3.5);`	`0.5`
`x4=mod(10,-3);`	`1`

MODULEC

Calls an external routine and returns a character value

Category: External Routines

See: "CALL MODULE" on page 244

Syntax

MODULEC(<*cntl-string*,>*module-name*<,*argument-1*, ..., *argument-n*>)

Details

For details on the MODULEC function, see "CALL MODULE" on page 244.

See Also

CALL Routines:
- "CALL MODULE" on page 244
- "CALL MODULEI" on page 246

Functions:
- "MODULEIC" on page 449
- "MODULEIN" on page 449
- "MODULEN" on page 450

MODULEIC

Calls an external routine and returns a character value (in IML environment only)

Category: External Routines

Restriction: MODULEIC can only be invoked from within the IML procedure

See: "CALL MODULE" on page 244

Syntax

MODULEIC(<*cntl-string,*>*module-name*<,*argument-1, ..., argument-n*>)

Details

For details on the MODULEIC function, see "CALL MODULE" on page 244.

See Also

CALL Routines:
- "CALL MODULE" on page 244
- "CALL MODULEI" on page 246

Functions:
- "MODULEC" on page 448
- "MODULEIN" on page 449
- "MODULEN" on page 450

MODULEIN

Calls an external routine and returns a numeric value (in IML environment only)

Category: External Routines

Restriction: MODULEIN can only be invoked from within the IML procedure

See: "CALL MODULE" on page 244

Syntax

MODULEIN(<*cntl-string*,>*module-name*<,*argument-1*, ..., *argument-n*>)

Details

For details on the MODULEIN function, see "CALL MODULE" on page 244.

See Also

CALL Routines:
- "CALL MODULE" on page 244
- "CALL MODULEI" on page 246

Functions:
- "MODULEC" on page 448
- "MODULEIC" on page 449
- "MODULEN" on page 450

MODULEN

Calls an external routine and returns a numeric value

Category: External Routines

See: "CALL MODULE" on page 244

Syntax

MODULEN(<*cntl-string*,>*module-name*<,*argument-1*, ..., *argument-n*>)

Details

For details on the MODULEN function, see "CALL MODULE" on page 244.

See Also

CALL Routines:

“CALL MODULE” on page 244
“CALL MODULEI” on page 246

Functions:

“MODULEC” on page 448
“MODULEIC” on page 449
“MODULEIN” on page 449

MONTH

Returns the month from a SAS date value

Category: Date and Time

Syntax

MONTH(*date*)

Arguments

date
specifies a SAS expression that represents a SAS date value.

Details

The MONTH function returns a numeric value that represents the month from a SAS date value. Numeric values can range from 1 through 2.

Examples

SAS Statements	Results
`date='25jan94'd;`	
`m=month(date);`	
`put m;`	`1`

See Also

Functions:

"DAY" on page 315

"YEAR" on page 618

MOPEN

Opens a file by directory id and member name, and returns the file identifier or a 0

Category: External Files

Syntax

MOPEN(*directory-id,member-name<open-mode<,record-length<,record-format>>>*)

Arguments

directory-id
specifies the identifier that was assigned when the directory was opened, generally by the DOPEN function.

member-name
specifies the member name in the directory.

open-mode
specifies the type of access to the file:

A	APPEND mode allows writing new records after the current end of the file.
I	INPUT mode allows reading only. (Default)
O	OUTPUT mode defaults to the OPEN mode specified in the host option in the FILENAME statement or function. If no host option is specified, it allows writing new records at the beginning of the file.
S	Sequential input mode is used for pipes and other sequential devices such as hardware ports.
U	UPDATE mode allows both reading and writing.
W	Sequential update mode is used for pipes and other sequential devices such as ports.

Default: I

record-length
specifies a new logical record length for the file. To use the existing record length for the file, specify a length of 0 or do not provide a value here.

record-format
specifies a new record format for the file. To use the existing record format, do not specify a value here. Valid values are:

B	specifies that data is to be interpreted as binary data.
D	specifies the default record format.
E	specifies the editable record format.
F	specifies that the file contains fixed-length records.
P	specifies that the file contains printer carriage control in host-dependent record format.
V	specifies that the file contains variable-length records.

Details

MOPEN returns the identifier for the file, or 0 if the file could not be opened. You can use a *file-id* that is returned by the MOPEN function as you would use a *file-id* returned by the FOPEN function.

CAUTION:
Use OUTPUT mode with care. Opening an existing file for output may overwrite the current contents of the file without warning. △

The member is identified by *directory-id* and *member-name* instead of by a fileref. You can also open a directory member by using FILENAME to assign a fileref to the member, followed by a call to FOPEN. However, when you use MOPEN, you do not have to use a separate fileref for each member.

If the file already exists, the output and update modes default to the host option (append or replace) specified with the FILENAME statement or function. For example,

```
%let rc=%sysfunc(filename(file,physical-name,,mod));
%let did=%sysfunc(dopen(&file));
%let fid=%sysfunc(mopen(&did,member-name,o,0,d));
%let rc=%sysfunc(fput(&fid,This is a test.));
%let rc=%sysfunc(fwrite(&fid));
%let rc=%sysfunc(fclose(&fid));
```

If **'file'** already exists, FWRITE appends the new record instead of writing it at the beginning of the file. However, if there is no a host option specified with the FILENAME function, the output mode implies that the record be replace.

If the open fails, use SYSMSG to retrieve the message text.

Operating Environment Information: The term *directory* in this description refers to an aggregate grouping of files that are managed by the operating environment. Different host operating environments identify such groupings with different names, such as directory, subdirectory, MACLIB, or partitioned data set. For details, see the SAS documentation for your operating environment.

Opening a directory member for output or append is not possible in some operating environments. △

Examples

This example assigns the fileref MYDIR to a directory. Then it opens the directory, determines the number of members, retrieves the name of the first member, and opens that member. The last three arguments to MOPEN are the defaults. Note that in a macro statement you do not enclose character strings in quotation marks.

```
%let filrf=mydir;
%let rc=%sysfunc(filename(filrf,physical-name));
%let did=%sysfunc(dopen(&filrf));
%let frstname=' ';
%let memcount=%sysfunc(dnum(&did));
%if (&memcount > 0) %then
   %do;
      %let frstname =
         %sysfunc(dread(&did,1));
      %let fid =
         %sysfunc(mopen(&did,&frstname,i,0,d));
      macro statements to process the member

      %let rc=%sysfunc(fclose(&fid));
   %end;
%else
   %put %sysfunc(sysmsg());
%let rc=%sysfunc(dclose(&did));
```

See Also

Functions:
- "DCLOSE" on page 316
- "DNUM" on page 333
- "DOPEN" on page 334
- "DREAD" on page 337
- "FCLOSE" on page 348
- "FILENAME" on page 358
- "FOPEN" on page 371
- "FPUT" on page 378
- "FWRITE" on page 385
- "SYSMSG" on page 558

MORT

Returns amortization parameters

Category: Financial

Syntax

MORT(*a*,*p*,*r*,*n*)

Arguments

a
is numeric, the initial amount.

p
is numeric, the periodic payment.

r
is numeric, the periodic interest rate that is expressed as a fraction.

n
is an integer, the number of compounding periods.

Range: $n \geq 0$

Details

The MORT function returns the missing argument in the list of four arguments from an amortization calculation with a fixed interest rate that is compounded each period. The arguments are related by

$$p = \frac{ar(1+r)^n}{(1+r)^n - 1}$$

One missing argument must be provided. It is then calculated from the remaining three. No adjustment is made to convert the results to round numbers.

Examples

An amount of $50,000 is borrowed for 30 years at an annual interest rate of 10 percent compounded monthly. The monthly payment can be expressed as

```
payment=mort(50000, . , .10/12,30*12);
```

The value returned is 438.79. The second argument has been set to missing, which indicates that the future value is to be calculated. The 10 percent nominal annual rate has been converted to a monthly rate of 0.10/12. The rate is the fractional (not the percentage) interest rate per compounding period. The 30 years are converted into 360 months.

N

Returns the number of nonmissing values

Category: Descriptive Statistics

Syntax

N(*argument*,*argument*, . . .)

Arguments

argument
is numeric. At least one argument is required. The argument list may consist of a variable list, which is preceded by OF.

Examples

SAS Statements	Results
`x1=n(1,0,.,2,5,.);`	**4**
`x2=n(1,2);`	**2**
`x3=n(of x1-x2);`	**2**

NETPV

Returns the net present value as a fraction

Category: Financial

Syntax

NETPV(*r,freq,c0,c1, . . . ,cn*)

r
is numeric, the interest rate over a specifed base period of time expressed as a fraction.

freq
is numeric, the number of payments during the base period of time that is specified with the rate *r*.

Range: *freq* > 0

Exception: The case *freq* = 0 is a flag to allow continuous discounting.

c0,c1, . . . ,cn
are numeric cash flows that represent cash outlays (payments) or cash inflows (income) occurring at times 0, 1, ... n. These cash flows are assumed to be equally spaced, beginning-of-period values. Negative values represent payments, positive values represent income, and values of 0 represent no cash flow at a given time. The *c0* argument and the *c1* argument are required.

Details

The NETPV function returns the net present value at time 0 for the set of cash payments *c0,c1, . . . ,cn*, with a rate *r* over a specified base period of time. The argument *freq*>0 describes the number of payments that occur over the specified base period of time.

The net present value is given by

$$\mathrm{NETPV}\left(r, freq, c_0, c_1, ..., c_n\right) = \sum_{i=0}^{n} c_i x^i$$

where

$$x = \begin{cases} \frac{1}{(1+r)^{(1/freq)}} & freq > 0 \\ e^{-r} & freq = 0 \end{cases}$$

Missing values in the payments are treated as 0 values. When *freq*>0, the rate *r* is the effective rate over the specified base period. To compute with a quarterly rate (the base period is three months) of 4 percent with monthly cash payments, set *freq* to 3 and set *r* to .04.

If *freq* is 0, continuous discounting is assumed. The base period is the time interval between two consecutive payments, and the rate *r* is a nominal rate.

To compute with a nominal annual interest rate of 11 percent discounted continuously with monthly payments, set *freq* to 0 and set *r* to .11/12.

Examples

For an initial investment of $500 that returns biannual payments of $200, $300, and $400 over the succeeding 6 years and an annual discount rate of 10 percent, the net present value of the investment can be expressed as

```
value=netpv(.10,.5,-500,200,300,400);
```

The value returned is 95.98.

NMISS

Returns the number of missing values

Category: Descriptive Statistics

Syntax

NMISS(*argument*,*argument*, . . .)

Arguments

argument
is numeric. At least one argument is required. The argument list may consist of a variable list, which is preceded by OF.

Examples

SAS Statements	Results
`x1=nmiss(1,0,.,2,5,.);`	2
`x2=nmiss(1,0);`	0
`x3=nmiss(of x1-x2);`	0

NORMAL

Returns a random variate from a normal distribution

Category: Random Number

See: "RANNOR" on page 516

Syntax

NORMAL(*seed*)

Arguments

seed

is an integer.

Range: $seed < 2^{31} - 1$

Note: If $seed \le 0$, the time of day is used to initialize the seed stream.

NOTE

Returns an observation ID for the current observation of a SAS data set

Category: SAS File I/O

Syntax

NOTE(*data-set-id*)

Arguments

data-set-id

specifies the data set identifier that the OPEN function returns.

Details

You can use the observation ID value to return to the current observation by using POINT. Observations can be marked by using NOTE and then returned to later by using POINT. Each observation ID is a unique numeric value.

To free the memory that is associated with an observation ID, use DROPNOTE.

Examples

This example calls CUROBS to display the observation number, calls NOTE to mark the observation, and calls POINT to point to the observation that corresponds to NOTEID.

```
%let dsid=%sysfunc(open(sasuser.fitness,i));
  /* Go to observation 10 in data set */
%let rc=%sysfunc(fetchobs(&dsid,10));
%if %sysfunc(abs(&rc)) %then
  %put FETCHOBS FAILED;
%else
  %do;
      /* Display observation number     */
      /* in the Log                     */
    %let cur=%sysfunc(curobs(&dsid));
    %put CUROBS=&cur;
      /* Mark observation 10 using NOTE */
    %let noteid=%sysfunc(note(&dsid));
      /* Rewind pointer to beginning    */
      /* of data                        */
      /* set using REWIND               */
    %let rc=%sysfunc(rewind(&dsid));
      /* FETCH first observation into DDV */
    %let rc=%sysfunc(fetch(&dsid));
      /* Display first observation number */
    %let cur=%sysfunc(curobs(&dsid));
    %put CUROBS=&cur;
      /* POINT to observation 10 marked  */
      /* earlier by NOTE                 */
    %let rc=%sysfunc(point(&dsid,&noteid));
      /* FETCH observation into DDV */
    %let rc=%sysfunc(fetch(&dsid));
      /* Display observation number 10   */
      /* marked by NOTE                  */
    %let cur=%sysfunc(curobs(&dsid));
    %put CUROBS=&cur;
  %end;
%if (&dsid > 0) %then
  %let rc=%sysfunc(close(&dsid));
```

The output produced by this program is:

```
CUROBS=10
CUROBS=1
CUROBS=10
```

See Also

Functions:

"DROPNOTE" on page 338
"OPEN" on page 460
"POINT" on page 481
"REWIND" on page 523

NPV

Returns the net present value with the rate expressed as a percentage

Category: Financial

Syntax

NPV(*r,freq,c0,c1, . . . ,cn*)

Arguments

r
is numeric, the interest rate over a specifed base period of time expressed as a percentage.

freq
is numeric, the number of payments during the base period of time specified with the rate *r*.

Range: *freq* > 0

Exception: The case *freq* = 0 is a flag to allow continuous discounting.

c0,c1, . . . ,cn
are numeric cash flows that represent cash outlays (payments) or cash inflows (income) occurring at times 0, 1, ... n. These cash flows are assumed to be equally spaced, beginning-of-period values. Negative values represent payments, positive values represent income, and values of 0 represent no cash flow at a given time. The *c0* argument and the *c1* argument are required.

Comparisons

The NPV function is identical to NETPV, except that the *r* argument is provided as a percentage.

OPEN

Opens a SAS data set and returns a value

Category: SAS File I/O

Syntax

OPEN(<*data-set-name*<,*mode*>>)

Arguments

data-set-name
specifies the SAS data set to be opened. The name should be of the form

<*libref.*>*member-name*<(*data-set-options*)>

Default: The default value for *data-set-name* is _LAST_.

Restriction: If you specify the FIRSTOBS= and OBS= data set, they are ignored. All other data set options are valid.

mode
specifies the type of access to the data set:

I	opens the data set in INPUT mode (default). Values can be read but not modified. **'I'** uses the strongest access mode available in the engine. That is, if the engine supports random access, OPEN defaults to random access. Otherwise, the file is opened in **'IN'** mode automatically. Files are opened with sequential access and a system level warning is set.
IN	opens the data set in INPUT mode. Observations are read sequentially, and you are allowed to revisit an observation.
IS	opens the data set in INPUT mode. Observations are read sequentially, but you are not allowed to revisit an observation.

Default: I

Details

OPEN opens a SAS data set (a SAS data set or a SAS SQL view) and returns a unique numeric data set identifier, which is used in most other data set access functions. OPEN returns 0 if the data set could not be opened.

By default, a SAS data set is opened with a control level of RECORD. For details, see the data set option "CNTLLEV=" on page 11. An open SAS data set should be closed when it is no longer needed. If you open a data set within a DATA step, it will be closed automatically when the DATA step ends.

OPEN defaults to the strongest access mode available in the engine. That is, if the engine supports random access, OPEN defaults to random access when data sets are opened in INPUT or UPDATE mode. Otherwise, data sets are opened with sequential access, and a system-level warning is set.

Examples

- This example opens the data set PRICES in the library MASTER using INPUT mode. Note that in a macro statement you do not enclose character strings in quotation marks.

```
%let dsid=%sysfunc(open(master.prices,i));
%if (&dsid = 0) %then
   %put %sysfunc(sysmsg());
%else
   %put PRICES data set has been opened;
```

- This example passes values from macro or DATA step variables to be used on data set options. It opens the data set SASUSER.HOUSES, and uses the WHERE= data set option to apply a permanent WHERE clause. Note that in a macro statement you do not enclose character strings in quotation marks.

```
%let choice = style="RANCH";
%let dsid=%sysfunc(open(sasuser.houses
                        (where=(&choice)),i));
```

See Also

Function:

“CLOSE” on page 289

ORDINAL

Returns any specified order statistic

Category: Descriptive Statistics

Syntax

ORDINAL(*count,argument,argument*, . . .)

Arguments

count
is an integer that is less than the number of elements in the list of arguments.

argument
is numeric. At least two arguments are required. The argument list may consist of a variable list, preceded by OF.

Details

The ORDINAL function sorts the list and returns the *count*th argument in the list.

Examples

SAS Statements	Results
`x1=ordinal(4,1,2,3,-4,5,6,7);`	**3**

PATHNAME

Returns the physical name of a SAS data library or of an external file, or returns a blank

Category: SAS File I/O

Category: External Files

Syntax

PATHNAME(*fileref* | *libref*)

Arguments

fileref
specifies the fileref assigned to an external file.

libref
specifies the libref assigned to a SAS library.

Details

PATHNAME returns the physical name of an external file or SAS library, or blank if *fileref* or *libref* is invalid. The default length of the target variable in the DATA step is 200 characters.

A fileref can be assigned to an external file by using the FILENAME statement or the FILENAME function.

Operating Environment Information: Under some operating environments, filerefs can also be assigned by using system commands. For details, see the SAS documentation for your operating environment. △

You can assign a libref to a SAS library using the LIBNAME statement or the LIBNAME function. Some operating environments allow you to assign a libref using system commands.

Examples

This example uses the FILEREF function to verify that the fileref MYFILE is associated with an external file. Then it uses PATHNAME to retrieve the actual name of the external file:

```
data _null_;
  length fname $ 100;
  rc=fileref("myfile");
```

```
    if (rc=0) then
    do;
      fname=pathname("myfile");
      put fname=;
    end;
  run;
```

See Also

Functions:

- "FEXIST" on page 354
- "FILEEXIST" on page 357
- "FILENAME" on page 358
- "FILEREF" on page 360

Statements:

- "LIBNAME" on page 913
- "FILENAME" on page 821

PDF

Computes probability density (mass) functions

Category: Probability

Alias: PMF

Syntax

PDF (*'dist',quantile,parm-1, . . . ,parm-k*)

Arguments

'dist'

is a character string that identifies the distribution. Valid distributions are as follows:

Distribution	Argument
Bernoulli	**'BERNOULLI'**
Beta	**'BETA'**
Binomial	**'BINOMIAL'**
Cauchy	**'CAUCHY'**
Chi-squared	**'CHISQUARED'**
Exponential	**'EXPONENTIAL'**
F	**'F'**
Gamma	**'GAMMA'**

Distribution	Argument
Geometric	**'GEOMETRIC'**
Hypergeometric	**'HYPERGEOMETRIC'**
Laplace	**'LAPLACE'**
Logistic	**'LOGISTIC'**
Lognormal	**'LOGNORMAL'**
Negative binomial	**'NEGBINOMIAL'**
Normal	**'NORMAL'|'GAUSS'**
Pareto	**'PARETO'**
Poisson	**'POISSON'**
T	**'T'**
Uniform	**'UNIFORM'**
Wald (inverse Gaussian)	**'WALD'|'IGAUSS'**
Weibull	**'WEIBULL'**

Note: Except for *T* and *F*, any distribution can be minimally identified by its first four characters. △

quantile
is a numeric random variable.

parm-1, . . . ,parm-k
are shape, location, or scale parameters appropriate for the specific distribution. See the description for each distribution in "Details" for complete information about these parameters.

Details

Bernoulli Distribution

PDF('BERNOULLI',*x*,*p*)

where

x
is a numeric random variable.

p
is a numeric probability of success.
Range: $0 \le p \le 1$

The PDF function for the Bernoulli distribution returns the probability density function of a Bernoulli distribution, with probability of success equal to *p*, which is evaluated at the value *x*. The equation follows:

$$PDF\left('BERN', x, p\right) = \begin{cases} 0 & x < 0 \\ 1 - p & x = 0 \\ 0 & 0 < x < 1 \\ p & x = 1 \\ 0 & x > 1 \end{cases}$$

Note: There are no *location* or *scale* parameters for this distribution. △

Beta Distribution

PDF('BETA',*x*,*a*,*b*<,*l*,*r*>)

where

x
: is a numeric random variable.

a
: is a numeric shape parameter.

Range: $a > 0$

b
: is a numeric shape parameter.

Range: $b > 0$

l
: is an optional numeric left location parameter.

r
: is an optional right location parameter.

Range: $r > l$

The PDF function for the beta distribution returns the probability density function of a beta distribution, with shape parameters *a* and *b*, which is evaluated at the value *x*. The equation follows:

$$PDF\left('BETA', x, a, b, l, r\right) = \begin{cases} 0 & x < 1 \\ \frac{1}{\beta(a,b)} \frac{(x-l)^{a-1}(x-r)^{b-1}}{(r-l)^{a+b-1}} & l \le x \le r \\ 0 & x > r \end{cases}$$

Note: The quantity $\frac{x-l}{r-l}$ is forced to be $\epsilon \le \frac{x-l}{r-l} \le 1 - 2\epsilon$. The default values for *l* and *r* are 0 and 1, respectively. △

Binomial Distribution

PDF('BINOMIAL',*m*,*p*,*n*)

where

m
: is an integer random variable that counts the number of successes.

p
: is a numeric parameter that is the probability of success.

Range: $0 \le p \le 1$

n
: is an integer parameter that counts the number of independent Bernoulli trials.

Range: $n > 0$

The PDF function for the binomial distribution returns the probability density function of a binomial distribution, with parameters *p* and *n*, which is evaluated at the value *m*. The equation follows:

$$PDF\left('BINOM', m, p, n\right) = \begin{cases} 0 & m < 0 \\ \binom{n}{m} p^m (1-p)^{n-m} & 0 \le m \le n \\ 0 & m > n \end{cases}$$

Note: There are no *location* or *scale* parameters for the binomial distribution. △

Cauchy Distribution

PDF('CAUCHY',*x*<,θ,λ>)

where

x
: is a numeric random variable.

θ
: is an optional numeric location parameter.

λ
: is an optional numeric scale parameter.
: **Range:** $\lambda > 0$

The PDF function for the Cauchy distribution returns the probability density function of a Cauchy distribution, with location parameter θ and scale parameter λ, which is evaluated at the value *x*. The equation follows:

$$PDF\left('CAUCHY',x,\theta,\lambda\right) = \frac{1}{\pi}\left(\frac{\lambda}{\lambda^2 + (x-\theta)^2}\right)$$

Note: The default values for θ and λ are 0 and 1, respectively. △

Chi-squared Distribution

PDF('CHISQUARED',*x*,*df* <,*nc*>)

where

x
: is a numeric random variable.

df
: is a numeric degrees of freedom parameter.
: **Range:** *df* > 0

nc
: is an optional numeric noncentrality parameter.
: **Range:** *nc* ≥ 0

The PDF function for the chi-squared distribution returns the probability density function of a chi-squared distribution, with *df* degrees of freedom and noncentrality parameter *nc*, which is evaluated at the value *x*. This function accepts noninteger degrees of freedom. If *nc* is omitted or equal to zero, the value returned is from the central chi-squared distribution. The following equation describes the PDF function of the chi–squared distribution,

$$PDF\left('CHISQ',x,v,\lambda\right) = \begin{cases} 0 & x < 0 \\ \sum\limits_{j=0}^{\infty} e^{-\frac{\lambda}{2}} \frac{\left(\frac{\lambda}{2}\right)^j}{j!} p_c\left(x, v+2j\right) & x \geq 0 \end{cases}$$

where $p_c(.,.)$ denotes the density from the central chi-squared distribution:

$$p_c\left(x,a\right) = \frac{}{2} p_g\left(\frac{x}{2},\frac{a}{2}\right)$$

and where $p_g(y,b)$ is the density from the Gamma distribution, which is given by

$$p_g\left(y,b\right) = \frac{}{\Gamma\left(b\right)} e^{-y} y^{b-1}$$

Exponential Distribution

PDF('EXPONENTIAL',*x* <,λ>)

where

x
: is a numeric random variable.

λ
: is an optional scale parameter.

  Range: $\lambda > 0$

The PDF function for the exponential distribution returns the probability density function of an exponential distribution, with scale parameter λ, which is evaluated at the value *x*. The equation follows:

$$PDF\left('EXPO', x, \lambda\right) = \begin{cases} 0 & x < 0 \\ \frac{1}{\lambda} exp\left(-\frac{x}{\lambda}\right) & x \geq 0 \end{cases}$$

Note: The default value for λ is 1. △

F Distribution

PDF('F',*x*,*ndf*,*ddf*<,*nc*>)

where

x
: is a numeric random variable.

ndf
: is a numeric numerator degrees of freedom parameter.

  Range: *ndf* > 0

ddf
: is a numeric denominator degrees of freedom parameter.

  Range: *ddf* > 0

nc
: is a numeric noncentrality parameter.

  Range: *nc* ≥ 0

The PDF function for the *F* distribution returns the probability density function of an *F* distribution, with *ndf* numerator degrees of freedom, *ddf* denominator degrees of freedom, and noncentrality parameter *nc*, which is evaluated at the value *x*. This function accepts noninteger degrees of freedom for *ndf* and *ddf*. If *nc* is omitted or equal to zero, the value returned is from a central *F* distribution. The following equation describes the PDF function of the *F* distribution,

$$PDF\left('F', x, v_1, v_2, \lambda\right) = \begin{cases} 0 & x < 0 \\ \sum\limits_{j=0}^{\infty} e^{-\frac{\lambda}{2}} \frac{\left(\frac{\lambda}{2}\right)^j}{j!} p_f\left(f, v_1 + 2j, v_2\right) & x \geq 0 \end{cases}$$

where $p_f(f,u_1,u_2)$ is the density from the central *F* distribution with

$$p_f\left(f, u_1, u_2\right) = p_B\left(\frac{u_1 f}{uf + u_2}, \frac{u_1}{2}, \frac{u_2}{2}\right) \frac{u_1 u_2}{\left(u_2 + u_1 f\right)^2}$$

and where $p_B(x,a,b)$ is the density from the standard beta distribution.

Note: There are no *location scale* parameters for the *F* distribution. △

Gamma Distribution

PDF('GAMMA',*x*,*a*<,λ>)

where

x
: is a numeric random variable.

a
: is a numeric shape parameter.

Range: $a > 0$

λ
: is an optional numeric scale parameter.

Range: $\lambda > 0$

The PDF function for the gamma distribution returns the probability density function of a gamma distribution, with shape parameter *a* and scale parameter λ, which is evaluated at the value *x*. The equation follows:

$$PDF\left('GAMMA', x, a, \lambda\right) = \begin{cases} 0 & x < 0 \\ \frac{1}{\lambda^a \Gamma(a)} x^{a-1} exp\left(-\frac{x}{\lambda}\right) & x \geq 0 \end{cases}$$

Note: The default value for λ is 1. △

Geometric Distribution

PDF('GEOMETRIC',*m*,*p*)

where

m
: is a numeric random variable that denotes the number of failures.

Range: $m \geq 0$

p
: is a numeric probability.

Range: $0 \leq p \leq 1$

The PDF function for the geometric distribution returns the probability density function of a geometric distribution, with parameter *p*, which is evaluated at the value *m*. The equation follows:

$$PDF\left('GEOM', m, p\right) = \begin{cases} 0 & m < 0 \\ p\left(1-p\right)^{m} & m \geq 0 \end{cases}$$

Note: There are no *location* or *scale* parameters for this distribution. △

Hypergeometric Distribution

PDF('HYPER',*x*,*m*,*k*,*n*<,*r*>)

where

x
: is an integer random variable.

m
: is an integer population size parameter.
: **Range:** $m \geq 1$

k
: is an integer number of items in the category of interest.
: **Range:** $0 \leq k \leq m$

n
: is an integer sample size parameter.
: **Range:** $0 \leq n \leq m$

r
: is an optional numeric odds ratio parameter.
: **Range:** $r > 0$

The PDF function for the hypergeometric distribution returns the probability density function of an extended hypergeometric distribution, with population size *m*, number of items *k*, sample size *n*, and odds ratio *r*, which is evaluated at the value *x*. If *r* is omitted or equal to 1, the value returned is from the usual hypergeometric distribution. The equation follows:

$$PDF\left('HYPER', x, m, k, n, r\right) = \begin{cases} 0 & x < max\left(0, k+n-m\right) \\ \dfrac{\binom{k}{x}\binom{m-k}{n-x}r^{x}}{\sum_{j=max(0,k+n-m)}^{min(k,n)}\binom{k}{j}\binom{m-k}{n-j}r^{j}} & max\left(0, k+n-m\right) \leq x \leq min\left(k, n\right) \\ 0 & x > min\left(k, n\right) \end{cases}$$

Laplace Distribution

PDF('LAPLACE',*x*<,*θ*,*λ*>)

where

x
: is a numeric random variable.

θ
: is an optional numeric location parameter.

λ
: is an optional numeric scale parameter.

Range: $\lambda > 0$

The PDF function for the Laplace distribution returns the probability density function of the Laplace distribution, with location parameter θ and scale parameter λ, which is evaluated at the value *x*. The equation follows:

$$PDF\left('LAPLACE', x, \theta, \lambda\right) = \frac{1}{2}exp\left(\frac{|x-\theta|}{\lambda}\right)$$

Note: The default values for θ and λ are 0 and 1, respectively. △

Logistic Distribution

PDF('LOGISTIC',*x*<,θ,λ>)

where

x
: is a numeric random variable.

θ
: is an optional numeric location parameter.

λ
: is an optional numeric scale parameter.

Range: $\lambda > 0$

The PDF function for the logistic distribution returns the probability density function of a logistic distribution, with a location parameter θ and a scale parameter λ, which is evaluated at the value *x*. The equation follows:

$$PDF\left('LOGISTIC', x, \theta, \lambda\right) = \frac{1}{\lambda\left(1 + exp\left(\frac{x-\theta}{\lambda}\right)\right)}$$

Note: The default values for θ and λ are 0 and 1, respectively. △

Lognormal Distribution

PDF('LOGNORMAL',*x*<,θ,λ>)

where

x
: is a numeric random variable.

θ
: is an optional numeric location parameter.

λ
: is an optional numeric scale parameter.

Range: $\lambda > 0$

The PDF function for the lognormal distribution returns the probability density function of a lognormal distribution, with location parameter θ and scale parameter λ, which is evaluated at the value *x*. The equation follows:

$$PDF\left('LOGN', x, \theta, \lambda\right) = \begin{cases} 0 & x \le 0 \\ \frac{1}{x\sqrt{2\pi\lambda}} exp\left(-\frac{(log(x)-\theta)^2}{2\lambda^2}\right) & x > 0 \end{cases}$$

Note: The default values for θ and λ are 0 and 1, respectively. △

Negative Binomial Distribution

PDF('NEGBINOMIAL',*m*,*p*,*n*)

where

m
: is a positive integer random variable that counts the number of failures.

Range: $m \ge 0$

p
: is a numeric probability of success parameter.

Range: $0 \le p \le 1$

n
: is an integer parameter that counts the number of successes.

Range: $n \ge 1$

The PDF function for the negative binomial distribution returns the probability density function of a negative binomial distribution, with probability of success *p* and number of successes *n*, which is evaluated at the value *m*. The equation follows:

$$PDF\left('NEGB', m, p, n\right) = \begin{cases} 0 & m < 0 \\ p^n \binom{n+m-1}{m} (1-p)^m & m \ge 0 \end{cases}$$

Note: There are no *location* or *scale* parameters for the negative binomial distribution. △

Normal Distribution

PDF('NORMAL',*x*<,θ,λ>)

where

x
: is a numeric random variable.

θ
: is an optional numeric location parameter.

λ
: is an optional numeric scale parameter.

Range: $\lambda > 0$

The PDF function for the normal distribution returns the probability density function of a normal distribution, with location parameter θ and scale parameter λ, which is evaluated at the value *x*. The equation follows:

$$PDF\left('NORMAL', x, \theta, \lambda\right) = \frac{1}{\lambda\sqrt{2\pi}} exp\left(-\frac{(x-\theta)^2}{2\lambda^2}\right)$$

Note: The default values for θ and λ are 0 and 1, respectively. △

Pareto Distribution

PDF('PARETO',*x*,*a*<,*k*>)

where

x
: is a numeric random variable.

a
: is a numeric shape parameter.

Range: $a > 0$

k
: is an optional numeric scale parameter.

Range: $k > 0$

The PDF function for the Pareto distribution returns the probability density function of a Pareto distribution, with shape parameter *a* and scale parameter *k*, which is evaluated at the value *x*. The equation follows:

$$PDF\left('PARETO', x, a, k\right) = \begin{cases} 0 & x < k \\ \frac{a}{k}\left(\frac{k}{x}\right)^{a+1} & x \geq k \end{cases}$$

Note: The default value for *k* is 1. △

Poisson Distribution

PDF('POISSON',*n*,*m*)

where

n
: is an integer random variable.

m
: is a numeric mean parameter.

Range: $m > 0$

The PDF function for the Poisson distribution returns the probability density function of a Poisson distribution, with mean *m*, which is evaluated at the value *n*. The equation follows:

$$PDF\left('POISSON', n, m\right) = \begin{cases} 0 & n < 0 \\ e^{-m}\frac{m^n}{n!} & n \geq 0 \end{cases}$$

Note: There are no *location* or *scale* parameters for the Poisson distribution. △

T distribution

PDF('T',*t*,*df*<,*nc*>)

where

t
: is a numeric random variable.

df
: is a numeric degrees of freedom parameter.

  range: *df* > 0

nc
: is an optional numeric noncentrality parameter.

The PDF function for the *T* distribution returns the probability density function of a *T* distribution, with degrees of freedom *df* and noncentrality parameter *nc*, which is evaluated at the value *x*. This function accepts noninteger degrees of freedom. If *nc* is omitted or equal to zero, the value returned is from the central *T* distribution. The equation follows:

$$PDF\left('T', t, v, \delta\right) = \frac{}{2^{\left(\frac{1}{2}v-\right)}\Gamma\left(\frac{v}{2}\right)} \int_{0}^{\infty} x^{v-1} e^{-\frac{1}{2}x^2} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{tx}{\sqrt{v}}-\delta\right)^2} \frac{x}{\sqrt{v}} dx$$

Note: There are no *location* or *scale* parameters for the *T* distribution. △

Uniform Distribution

PDF('UNIFORM',*x*<,*l*,*r*>)

where

x
: is a numeric random variable.

l
: is an optional numeric left location parameter.

r
: is an optional numeric right location parameter.

  Range: *r* > *l*

The PDF function for the uniform distribution returns the probability density function of a uniform distribution, with left location parameter *l* and right location parameter *r*, which is evaluated at the value *x*. The equation follows:

$$PDF\left('UNIFORM', x, l, r\right) = \begin{cases} 0 & x < l \\ \frac{1}{r-l} & l \le x \le r \\ 0 & x > r \end{cases}$$

Note: The default values for *l* and *r* are 0 and 1, respectively. △

Wald (Inverse Gaussian) Distribution

PDF('WALD',*x*,*d*)

PDF('IGAUSS',*x*,*d*)

where

x
: is a numeric random variable.

d
: is a numeric shape parameter.

Range: $d > 0$

The PDF function for the Wald distribution returns the probability density function of a Wald distribution, with shape parameter *d*, which is evaluated at the value *x*. The equation follows:

$$PDF\left('WALD', x, d\right) = \begin{cases} 0 & x \le 0 \\ \sqrt{\frac{d}{2\pi x}} exp\left(-\frac{d}{2}x + d - \frac{d}{2x}\right) & x > 0 \end{cases}$$

Note: There are no *location* or *scale* parameters for the Wald distribution. △

Weibull Distribution

PDF('WEIBULL',*x*,*a*<,λ>)

where

x
: is a numeric random variable.

a
: is a numeric shape parameter.

  Range: $a > 0$

λ
: is an optional numeric scale parameter.

  Range: $\lambda > 0$

The PDF function for the Weibull distribution returns the probability density function of a Weibull distribution, with shape parameter *a* and scale parameter λ, which is evaluated at the value *x*. The equation follows:

$$PDF\left('WEIBULL', x, a, \lambda\right) = \begin{cases} 0 & x < 0 \\ exp\left(-\left(\frac{x}{\lambda}\right)^a\right)\frac{a}{\lambda}\left(\frac{x}{\lambda}\right)^{a-1} & x \ge 0 \end{cases}$$

Note: The default value for λ is 1. △

Examples

SAS Statements	Results
`y=pdf('BERN',0,.25);`	0.75
`y=pdf('BERN',1,.25);`	0.25
`y=pdf('BETA',0.2,3,4);`	1.2288
`y=pdf('BINOM',4,.5,10);`	0.20508
`y=pdf('CAUCHY',2);`	0.063662
`y=pdf('CHISQ',11.264,11);`	0.081686
`y=pdf('EXPO',1);`	0.36788
`y=pdf('F',3.32,2,3);`	0.054027
`y=pdf('GAMMA',1,3);`	0.18394
`y=pdf('HYPER',2,200,50,10);`	0.28685

SAS Statements	Results
`y=pdf('LAPLACE',1);`	0.18394
`y=pdf('LOGISTIC',1);`	0.19661
`y=pdf('LOGNORMAL',1);`	0.39894
`y=pdf('NEGB',1,.5,2);`	0.25
`y=pdf('NORMAL',1.96);`	0.058441
`y=pdf('PARETO',1,1);`	1
`y=pdf('POISSON',2,1);`	0.18394
`y=pdf('T',.9,5);`	0.24194
`y=pdf('UNIFORM',0.25);`	1
`y=pdf('WALD',1,2);`	0.56419
`y=pdf('WEIBULL',1,2);`	0.73576

PEEK

Stores the contents of a memory address into a numeric variable

Category: Special

Syntax

PEEK(*address*<,*length*>)

Arguments

address
specifies the memory address.

length
specifies the data length.

Default: a 4-byte address pointer

Range: 2 to 8

Details

If you do not have access to the memory storage location that you are requesting, the PEEK function returns an "Invalid argument" error.

Comparisons

The PEEK function stores the contents of a memory address into a *numeric* variable. The PEEKC function stores the contents of a memory address into a *character* variable.

Examples

The following example, specific to the OS/390 operating environment, returns a numeric value that represents the address of the Communication Vector Table (CVT).

```
data _null_;
      /* 16 is the location of of the CVT address */
   y=16;
   x=peek(y);
   put 'x= ' x hex8.;
run;
```

See also the second example in the PEEKC function description.

See Also

Functions:

"ADDR" on page 227

"PEEKC" on page 477

CALL Routine:

"CALL POKE" on page 247

PEEKC

Stores the contents of a memory address into a character variable

Category: Special

Syntax

PEEKC(*address*<,*length*>)

Arguments

address
specifies the memory address.

length
specifies the data length.

Default: 8, unless the variable length has already been set (by the LENGTH statement, for example)

Range: 1 to 32,767

Details

If you do not have access to the memory storage location that you are requesting, the PEEKC function returns an "Invalid argument" error.

Comparisons

The PEEKC function stores the contents of a memory address into a *character* variable. The PEEK function stores the contents of a memory address into a *numeric* variable.

Examples

Example 1: Listing ASCB Bytes The following example, specific to the OS/390 operating environment, uses both PEEK and PEEKC, and prints the first four bytes of the Address Space Control Block (ASCB).

```
data _null_;
   length y $4;
      /* 220x is the location of the ASCB pointer */
   x=220x;
   y=peekc(peek(x));
   put 'y= ' y;
run;
```

Example 2: Creating a DATA Step View This example, specific to the OS/390 operating environment, also uses both the PEEK and PEEKC functions. It creates a DATA step view that accesses the entries in the Task Input Output Table (TIOT). The PRINT procedure is then used to print the entries. Entries in the TIOT include the three components outlined in the following list. In this example, TIOT represents the starting address of the TIOT entry.

TIOT+4	is the DDname. This component takes up 8 bytes.
TIOT+12	is a 3-byte pointer to the Job File Control Block (JFCB).
TIOT+134	is the volume serial number (volser) of the data set. This component takes up 6 bytes.

Here is the program:

```
   /* Create a DATA step view of the contents    */
   /* of the TIOT. The code steps through each   */
   /* TIOT entry to extract the DDname, JFCB,    */
   /* and volser of each DDname that has been    */
   /* allocated for the current task. The data   */
   /* set name is also extracted from the JFCB.  */

data save.tiot/view=save.tiot;
   length ddname $8 volser $6 dsname $44;
      /* Get the TCB (Task Control Block)address */
      /* from the PSATOLD variable in the PSA    */
      /* (Prefixed Save Area).  The address of   */
      /* the PSA is 21CX. Add 12 to the address  */
      /* of the TCB to get the address of the    */
      /* TIOT. Add 24 to bypass the 24-byte      */
      /* header, so that TIOTVAR represents the  */
      /* start of the TIOT entries.              */

   tiotvar=peek(peek(021CX)+12)+24;

      /* Loop through all TIOT entries until the */
      /* TIOT entry length (indicated by the     */
      /* value of the first byte) is 0.          */
   do while(peek(tiotvar,1));

      /* Check to see whether the current TIOT   */
      /* entry is a freed TIOT entry (indicated  */
      /* by the high order bit of the second     */
      /* byte of the TIOT entry). If it is not   */
      /* freed, then proceed.                    */

      if peek(tiotvar+1,1)NE'1.......'B then do;
         ddname=peekc(tiotvar+4);
         jfcb=peek(tiotvar+12,3);
         volser=peekc(jfcb+134);
            /* Add 16 to the JFCB value to get  */
            /* the data set name.  The data set */
            /* name is 44 bytes.                */

         dsname=peekc(jfcb+16);
         output;
         end;

         /* Increment the TIOTVAR value to point */
         /* to the next TIOT entry. This is done */
         /* by adding the length of the current  */
         /* TIOT entry (indicated by first byte  */
         /* of the entry) to the current value   */
         /* of TIOTVAR.                          */

      tiotvar+peek(tiotvar,1);
   end;

      /* The final DATA step view does not       */
      /* contain the TIOTVAR and JFCB variables. */

   keep ddname volser dsname;
run;

     /* Print the TIOT entries.                  */
proc print data=save.tiot uniform width=minimum;
run;
```

In the PROC PRINT statement, the UNIFORM option ensures that each page of the output is formatted exactly the same way. WIDTH=MINIMUM causes the PRINT procedure to use the minimum column width for each variable on the page. The column width is defined by the longest data value in that column.

See Also

CALL Routine:
    "CALL POKE" on page 247
Functions:
    "ADDR" on page 227
    "PEEK" on page 476

PERM

Computes the number of permutations of *n* items taken *r* at a time and returns a value

Category: Mathematical

Syntax

PERM(*n*<,*r*>)

Arguments

n

is an integer that represents the total number of elements from which the sample is chosen.

r

is an optional integer value that represents the number of chosen elements. If *r* is omitted, the function returns the factorial of *n*.

Restriction: $r \le n$

Details

The mathematical representation of the PERM function is given by the following equation:

$$PERM\,(n, r) = \frac{n!}{(n-r)!}$$

with $n \ge 0$, $r \ge 0$, and $n \ge r$.

If the expression cannot be computed, a missing value is returned.

Examples

SAS Statements	Result
`x=perm(5,1);`	5
`x=perm(5);`	120
`x=perm(5,2)`	20

See Also

Functions:
"COMB" on page 293
"FACT" on page 346

POINT

Locates an observation identified by the NOTE function and returns a value

Category: SAS File I/O

Syntax

POINT(*data-set-id*,*note-id*)

Arguments

data-set-id
specifies the data set identifier that the OPEN function returns.

note-id
specifies the identifier assigned to the observation by the NOTE function.

Details

POINT returns 0 if the operation was successful, ≠0 if it was not successful. POINT prepares the program to read from the SAS data set. The Data Set Data Vector is not updated until a read is done using FETCH or FETCHOBS.

Examples

This example calls NOTE to obtain an observation ID for the most recently read observation of the SAS data set MYDATA. It calls POINT to point to that observation, and calls FETCH to return the observation marked by the pointer.

```
%let dsid=%sysfunc(open(mydata,i));
%let rc=%sysfunc(fetch(&dsid));
```

```
%let noteid=%sysfunc(note(&dsid));
   ...more macro statements...
%let rc=%sysfunc(point(&dsid,&noteid));
%let rc=%sysfunc(fetch(&dsid));
   ...more macro statements...
%let rc=%sysfunc(close(&dsid));
```

See Also

Functions:
- "DROPNOTE" on page 338
- "NOTE" on page 458
- "OPEN" on page 460

POISSON

Returns the probability from a Poisson distribution

Category: Probability

See: "CDF" on page 273

Syntax

POISSON(*m*,*n*)

Arguments

m
is a numeric mean parameter.

Range: $m \geq 0$

n
is an integer random variable.

Range: $n \geq 0$

Details

The POISSON function returns the probability that an observation from a Poisson distribution, with mean *m*, is less than or equal to *n*. To compute the probability that an observation is equal to a given value, *n*, compute the difference of two probabilities from the Poisson distribution for *n* and *n*– .

Examples

SAS Statements	Results
`x=poisson(1,2);`	**`0.9196986029`**

POKE

Writes a value directly into memory

Category: Special

Syntax

POKE(*source*,*pointer*<,*length*>)

Arguments

source
specifies a SAS expression that contains a value to write into memory.

pointer
specifies a numeric SAS expression that contains the virtual address of the data that the POKE function alters.

length
specifies a numeric SAS expression that contains the number of bytes to write from the *source* to the address indicated by *pointer*. If you omit *length*, the action that the POKE function takes depends on whether *source* is a character value or a numeric value:

- If *source* is a character value, then the POKE routine copies the entire value of *source* to the specified memory location.
- If *source* is a numeric value, then the POKE function converts *source* into a long integer and writes into memory the number of bytes that constitute a pointer.

  Operating Environment Information: Under OS/390, pointers are 3 or 4 bytes long, depending on the situation. △

Details

CAUTION:
The POKE function is intended only for experienced programmers in specific cases. If you plan to use this function, use extreme care both in your programming and in your typing. *Writing directly into memory can cause devastating problems.* It bypasses the normal safeguards that prevent you from destroying a vital element in your SAS session or in another piece of software that is active at the time. △

If you do not have access to the memory location that you specify, the POKE function returns an "Invalid argument" error.

See Also

Functions:
- "ADDR" on page 227
- "PEEK" on page 476
- "PEEKC" on page 477

CALL Routine:
- "CALL POKE" on page 247

PROBBETA

Returns the probability from a beta distribution

Category: Probability
See: "CDF" on page 273

Syntax

PROBBETA(*x*,*a*,*b*)

Arguments

x
is a numeric random variable.
Range: $0 \le x \le 1$

a
is a numeric shape parameter.
Range: $a > 0$

b
is a numeric shape parameter.
Range: $b > 0$

Details

The PROBBETA function returns the probability that an observation from a beta distribution, with shape parameters *a* and *b*, is less than or equal to *x*.

Example

SAS Statements	Results
`x=probbeta(.2,3,4);`	`0.09888`

PROBBNML

Returns the probability from a binomial distribution

Category: Probability

See: “CDF” on page 273, “PDF” on page 464

Syntax

PROBBNML(*p*,*n*,*m*)

Arguments

p

is a numeric probability of success parameter.

RANGE: $0 \leq p \leq 1$

n

is an integer number of independent Bernoulli trials parameter.

RANGE: $n > 0$

m

is an integer number of successes random variable.

RANGE: $0 \leq m \leq n$

Details

The PROBBNML function returns the probability that an observation from a binomial distribution, with probability of success *p*, number of trials *n*, and number of successes *m*, is less than or equal to *m*. To compute the probability that an observation is equal to a given value *m*, compute the difference of two probabilities from the binomial distribution for *m* and *m*–1 successes.

Examples

SAS Statements	Results
`x=probbnml(0.5,10,4);`	`0.376953125`

PROBBNRM

Computes a probability from the bivariate normal distribution and returns a value

Category: Probability

Syntax

PROBBNRM(*x, y, r*)

Arguments

x
is a numeric variable.

y
is a numeric variable.

r
is a numeric correlation coefficient.

Range: -1 ≤ *r* ≤ 1

Details

The PROBBNRM function returns the probability that an observation (X, Y) from a standardized bivariate normal distribution with mean 0, variance 1, and a correlation coefficient *r*, is less than or equal to (*x*, *y*). That is, it returns the probability that X≤*x* and Y≤*y*. The following equation describes the PROBBNRM function, where *u* and *v* represent the random variables *x* and *y*, respectively.

$$\mathrm{PROBBNRM}(x,y,r) = \frac{1}{2\pi\sqrt{1-\mathrm{r}^2}} \int_{-\infty}^{x}\int_{-\infty}^{y} \exp\left[-\frac{u^2-2ruv+v^2}{2\left(1-r^2\right)}\right] dv\, du$$

Examples

SAS Statements	Result
`p=probbnrm(.4, -3, .2);`	**0.2783183345**

PROBCHI

Returns the probability from a chi-squared distribution

Category: Probability

See: “CDF” on page 273

Syntax

PROBCHI(*x*,*df*<,*nc*>)

Arguments

x
is a numeric random variable.

Range: $x \geq 0$

df

is a numeric degrees of freedom parameter.

Range: $df > 0$

nc

is an optional numeric noncentrality parameter.

Range: $nc \geq 0$

Details

The PROBCHI function returns the probability that an observation from a chi-square distribution, with degrees of freedom *df* and noncentrality parameter *nc*, is less than or equal to *x*. This function accepts a noninteger degrees of freedom parameter *df*. If the optional parameter *nc* is not specified or has the value 0, the value returned is from the central chi-square distribution.

Examples

SAS Statements	Results
`x=probchi(11.264,11);`	`0.5785813293`

PROBF

Returns the probability from an *F* distribution

Category: Probability

See: "CDF" on page 273

Syntax

PROBF(*x,ndf,ddf*<,nc>)

Arguments

x

is a numeric random variable.

Range: $x \geq 0$

ndf

is a numeric numerator degrees of freedom parameter.

Range: $ndf > 0$

ddf

is a numeric denominator degrees of freedom parameter.

Range: *ddf* > 0

nc

is an optional numeric noncentrality parameter.

Range: *nc* ≥ 0

Details

The PROBF function returns the probability that an observation from an *F* distribution, with numerator degrees of freedom *ndf*, denominator degrees of freedom *ddf*, and noncentrality parameter *nc*, is less than or equal to *x*. The PROBF function accepts noninteger degrees of freedom parameters *ndf* and *ddf*. If the optional parameter *nc* is not specified or has the value 0, the value returned is from the central *F* distribution.

The significance level for an *F* test statistic is given by

```
p=1-probf(x,ndf,ddf);
```

Examples

SAS Statements	Results
x=probf(3.32,2,3);	**0.8263933602**

PROBGAM

Returns the probability from a gamma distribution

Category: Probability

See: "CDF" on page 273

Syntax

PROBGAM(*x*,*a*)

Arguments

x

is a numeric random variable.

Range: *x* ≥ 0

a

is a numeric shape parameter.

Range: *a* > 0

Details

The PROBGAM function returns the probability that an observation from a gamma distribution, with shape parameter *a*, is less than or equal to *x*.

Examples

SAS Statements	Results
`x=probgam(1,3);`	`0.0803013971`

PROBHYPR

Returns the probability from a hypergeometric distribution

Category: Probability

See: "CDF" on page 273

Syntax

PROBHYPR(*N*,*K*,*n*,*x*<,*r*>)

Arguments

N

is an integer population size parameter, with $N \geq 1$.

Range:

K

is an integer number of items in the category of interest parameter.

Range: $0 \leq K \leq N$

n

is an integer sample size parameter.

Range: $0 \leq n \leq N$

x

is an integer random variable.

Range: $max(0, K + n - N) \leq x \leq min(K,n)$

r

is an optional numeric odds ratio parameter.

Range: $r \geq 0$

Details

The PROBHYPR function returns the probability that an observation from an extended hypergeometric distribution, with population size *N*, number of items *K*, sample size *n*, and odds ratio *r*, is less than or equal to *x*. If the optional parameter *r* is not specified or is set to 1, the value returned is from the usual hypergeometric distribution.

Examples

SAS Statements	Results
`x=probhypr(200,50,10,2);`	`0.5236734081`

PROBIT

Returns a quantile from the standard normal distribution

Category: Quantile

Syntax

PROBIT(*p*)

Arguments

p
is a numeric probability.

Range: $0 \le p < 1$

Details

The PROBIT function returns the p^{th} quantile from the standard normal distribution. The probability that an observation from the standard normal distribution is less than or equal to the returned quantile is *p*.

CAUTION:
The result could be truncated to lie between -8.222 and 7.941. △

Note: PROBIT is the inverse of the PROBNORM function. △

Examples

SAS Statements	Results
`x=probit(.025);`	`-1.959963985`
`x=probit(1.e-7);`	`-5.199337582`

PROBMC

Computes a probability or a quantile from various distributions for multiple comparisons of means, and returns a value

Category: Probability

Syntax

PROBMC(*distribution, q, prob, df, nparms*<, *parameters*>)

Arguments

distribution
is a character string that identifies the distribution. Valid distributions are

Distribution	Argument
One-sided Dunnett	**'DUNNETT1'**
Two-sided Dunnett	**'DUNNETT2'**
Maximum Modulus	**'MAXMOD'**
Studentized Range	**'RANGE'**
Williams	**'WILLIAMS'**

q
is the quantile from the distribution.

Restriction: Either *q* or *prob* can be specified, but not both.

prob
is the left probability from the distribution.

Restriction: Either *prob* or *q* can be specified, but not both.

df
is the degrees of freedom.

Note: A missing value is interpreted as an infinite value. △

nparms
is the number of treatments.

Note: For DUNNETT1 and DUNNETT2, the control group is not counted. △

parameters
is an optional set of *nparms* parameters that must be specified to handle the case of unequal sample sizes. The meaning of *parameters* depends on the value of *distribution*. If *parameters* is not specified, equal sample sizes are assumed; this is usually the case for a null hypothesis.

Details

The PROBMC function returns the probability or the quantile from various distributions with finite and infinite degrees of freedom for the the variance estimate.

The *prob* argument is the probability that the random variable is less than *q*. Therefore, *p*-values can be computed as 1– *prob*. For example, to compute the critical value for a 5% significance level, set *prob*= 0.95. The precision of the computed probability is $O(10^{-8})$ (absolute error); the precision of computed quantile is $O(10^{-5})$.

Note: The studentized range is not computed for finite degrees of freedom and unequal sample sizes. △

Note: Williams' test is computed only for equal sample sizes. △

Formulas and Parameters The equations listed here define expressions used in equations that relate the probability, *prob*, and the quantile, *q*, for different distributions and different situations within each distribution. For these equations, let ν be the degrees of freedom, *df*.

$$d\mu_\nu(x) = \frac{\nu^{\frac{\nu}{2}}}{\Gamma\left(\frac{\nu}{2}\right) 2^{\frac{\nu}{2}-1}} x^{\nu-1} e^{-\frac{\nu x^2}{2}} dx$$

$$\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$$

$$\Phi(x) = \int_{-\infty}^{x} \phi(u)\, du$$

Many-One *t*-Statistics: Dunnett's One-Sided Test

- □ This case relates the probability, *prob*, and the quantile, *q*, for the unequal case with finite degrees of freedom. The *parameters* are $\lambda_1, \ldots, \lambda_k$, the value of *nparms* is set to *k*, and the value of *df* is set to ν. The equation follows:

$$prob = \int_0^\infty \int_{-\infty}^\infty \phi(y) \prod_{i=1}^k \Phi\left(\frac{\lambda_i y + qx}{\sqrt{1-\lambda_i^2}}\right) dy\, du_\nu(x)$$

- □ This case relates the probability, *prob*, and the quantile, *q*, for the equal case with finite degrees of freedom. No *parameters* are passed $\left(\lambda = \sqrt{\frac{1}{2}}\right)$, the value of *nparms* is set to *k*, and the value of *df* is set to ν. The equation follows:

$$prob = \int_0^\infty \int_{-\infty}^\infty \phi(y) \left[\Phi\left(y + \sqrt{2}qx\right)\right]^k dy\, du_\nu(x)$$

- □ This case relates the probability, *prob*, and the quantile, *q*, for the unequal case with infinite degrees of freedom. The *parameters* are $\lambda_1, \ldots, \lambda_k$, the value of *nparms* is set to *k*, and the value of *df* is set to missing. The equation follows:

$$prob = \int_{-\infty}^\infty \phi(y) \prod_{i=1}^k \Phi\left(\frac{\lambda_i y + q}{\sqrt{1-\lambda_i^2}}\right) dy$$

- This case relates the probability, *prob*, and the quantile, *q*, for the equal case with infinite degrees of freedom. No *parameters* are passed $\left(\lambda = \sqrt{\frac{1}{2}}\right)$, the value of *nparms* is set to *k*, and the value of *df* is set to missing. The equation follows:

$$prob = \int_{-\infty}^{\infty} \phi(y) \left[\Phi\left(y + \sqrt{2}q\right)\right]^k dy$$

Many-One *t*-Statistics: Dunnett's Two-sided Test

- This case relates the probability, *prob*, and the quantile, *q*, for the unequal case with finite degrees of freedom. The *parameters* are λ_1, ..., λ_k, the value of *nparms* is set to *k*, and the value of *df* is set to ν. The equation follows:

$$prob = \int_{0}^{\infty}\int_{-\infty}^{\infty} \phi(y) \prod_{i=1}^{k} \left[\Phi\left(\frac{\lambda_i y + qx}{\sqrt{1-\lambda_i^2}}\right) - \Phi\left(\frac{\lambda_i y - qx}{\sqrt{1-\lambda_i^2}}\right)\right] dy\, du_\nu(x)$$

- This case relates the probability, *prob*, and the quantile, *q*, for the equal case with finite degrees of freedom. No *parameters* are passed, the value of *nparms* is set to *k*, and the value of *df* is set to ν. The equation follows:

$$prob = \int_{0}^{\infty}\int_{-\infty}^{\infty} \phi(y) \left[\Phi\left(y + \sqrt{2}qx\right) - \Phi\left(y - \sqrt{2}qx\right)\right]^k dy\, du_\nu(x)$$

- This case relates the probability, *prob*, and the quantile, *q*, for the unequal case with infinite degrees of freedom. The *parameters* are λ_1, ..., λ_k, the value of *nparms* is set to *k*, and the value of *df* is set to missing. The equation follows:

$$prob = \int_{-\infty}^{\infty} \phi(y) \prod_{i=1}^{k} \left[\Phi\left(\frac{\lambda_i y + q}{\sqrt{1-\lambda_i^2}}\right) - \Phi\left(\frac{\lambda_i y - q}{\sqrt{1-\lambda_i^2}}\right)\right] dy$$

- This case relates the probability, *prob*, and the quantile, *q*, for the equal case with infinite degrees of freedom. No *parameters* are passed, the value of *nparms* is set to *k*, and the value of *df* is set to missing. The equation follows:

$$prob = \int_{-\infty}^{\infty} \phi(y) \left[\Phi\left(y + \sqrt{2}q\right) - \Phi\left(y - \sqrt{2}q\right)\right]^k dy$$

The Studentized Range

Note: The studentized range is not computed for finite degrees of freedom and unequal sample sizes. △

- □ This case relates the probability, *prob*, and the quantile, *q*, for the equal case with finite degrees of freedom. No *parameters* are passed, the value of *nparms* is set to *k*, and the value of *df* is set to ν. The equation follows:

$$prob = \int_{0}^{\infty} \int_{-\infty}^{\infty} k\phi(y)\left[\Phi(y) - \Phi(y - qx)\right]^{k-1} dy\, du_{\nu}(x)$$

- □ This case relates the probability, *prob*, and the quantile, *q*, for the unequal case with infinite degrees of freedom. The *parameters* are σ_1, ..., σ_k, the value of *nparms* is set to *k*, and the value of *df* is set to missing. The equation follows:

$$prob = \int_{-\infty}^{\infty} \sum_{j=1}^{k} \left\{ \prod_{i=1}^{k} \left[\Phi\left(\frac{y}{\sigma_i}\right) - \Phi\left(\frac{y-q}{\sigma_i}\right) \right] \right\} \phi\left(\frac{y}{\sigma_j}\right) \frac{1}{\sigma_j} dy$$

- □ This case relates the probability, *prob*, and the quantile, *q*, for the equal case with infinite degrees of freedom. No *parameters* are passed, the value of *nparms* is set to *k*, and the value of *df* is set to missing. The equation follows:

$$prob = \int_{-\infty}^{\infty} k\phi(y)\left[\Phi(y) - \Phi(y - q)\right]^{k-1} dy$$

The Studentized Maximum Modulus

- □ This case relates the probability, *prob*, and the quantile, *q*, for the unequal case with finite degrees of freedom. The *parameters* are σ_1, ..., σ_k, the value of *nparms* is set to *k*, and the value of *df* is set to ν. The equation follows:

$$prob = \int_{0}^{\infty} \prod_{i=1}^{k} \left[2\Phi\left(\frac{qx}{\sigma_i}\right) - 1 \right] d\mu_{\nu}(x)$$

- □ This case relates the probability, *prob*, and the quantile, *q*, for the equal case with finite degrees of freedom. No *parameters* are passed, the value of *nparms* is set to *k*, and the value of *df* is set to ν. The equation follows:

$$prob = \int_{0}^{\infty} \left[2\Phi(qx) - 1\right]^{k} d\mu_{\nu}(x)$$

- □ This case relates the probability, *prob*, and the quantile, *q*, for the unequal case with infinite degrees of freedom. The *parameters* are σ_1, ..., σ_k, the value of *nparms* is set to *k*, and the value of *df* is set to missing. The equation follows:

$$prob = \prod_{i=1}^{k} \left[2\Phi\left(\frac{q}{\sigma_i}\right) - 1 \right]$$

- □ This case relates the probability, *prob*, and the quantile, *q*, for the equal case with infinite degrees of freedom. No *parameters* are passed, the value of *nparms* is set to *k*, and the value of *df* is set to missing. The equation follows:

$$prob = \left[2\Phi(q) - 1\right]^k$$

Williams' Test PROBMC computes the probabilities or quantiles from the distribution defined in Williams (1971, 1972) (See "References" on page 626). It arises when you compare the dose treatment means with a control mean to determine the lowest effective dose of treatment.

Note: Williams' Test is computed only for equal sample sizes. △

Let X_1, X_2, ..., X_k be identical independent N(0,1) random variables. Let Y_k denote their average given by

$$Y_k = \frac{X_1 + X_2 + ... + X_k}{k}$$

It is required to compute the distribution of

$$\left(Y_k - Z\right)/S$$

where

Y_k	is as defined previously
Z	is a N(0,1) independent random variable
S	is such that $\frac{1}{2}\ \nu S^2$ is a χ^2 variable with ν degrees of freedom.

As described in Williams (1971) (See "References" on page 626), the full computation is extremely lengthy and is carried out in three stages.

1. Compute the distribution of Y_k. It is the fundamental (expensive) part of this operation and it can be used to find both the density and the probability of Y_k. Let U_i be defined as

$$U_i = \frac{X_1 + X_2 + ... + X_i}{i},\ i = 1, 2, ..., k$$

   You can write a recursive expression for the probability of $Y_k > d$, with d being any real number.

$$\begin{aligned}\Pr(Y_k > d) &= \Pr(U_1 > d) \\ &+ \Pr(U_2 > d, U_1 < d) \\ &+ \Pr(U_3 > d, U_2 < d, U_1 < d) \\ &+ \ldots \\ &+ \Pr(U_k > d, U_{k-1} < d, \ldots, U_1 < d) \\ &= \Pr(Y_{k-1} > d) + \Pr(X_k + (k-1)U_{k-1} > kd)\end{aligned}$$

To compute this probability, start from a N(0,1) density function

$$D(U_1 = x) = \phi(x)$$

and recursively compute the convolution

$$D(U_k = x, U_{k-1} < d, \ldots, U_1 < d) = \int_{-\infty}^{d} D(U_{k-1} = y, U_{k-2} < d, \ldots, U_1 < d)(k-1)\phi(kx - (k-1)y)\,dy$$

From this sequential convolution, it is possible to compute all the elements of the recursive equation for $\Pr(Y_k < d)$, shown previously.

2 Compute the distribution of $Y_k - Z$. This involves another convolution to compute the probability

$$\Pr((Y_k - Z) > d) = \int_{-\infty}^{\infty} \Pr\left(Y_k > \sqrt{2d} + y\right)\phi(y)\,dy$$

3 Compute the distribution of $(Y_k - Z)/S$. This involves another convolution to compute the probability

$$\Pr((Y_k - Z) > tS) = \int_{0}^{\infty} \Pr((Y_k - Z) > ty)\,d\mu_\nu(y)$$

The third stage is not needed when $\nu = \infty$. Due to the complexity of the operations, this lengthy algorithm is replaced by a much faster one when $k \leq 15$ for both finite and infinite degrees of freedom ν. For $k \geq 16$, the lengthy computation is carried out. It is extremely expensive and very slow due to the complexity of the algorithm.

Comparisons

The MEANS statement in the GLM Procedure of SAS/STAT Software computes the following tests:

- □ Dunnett's one-sided test

- □ Dunnett's two-sided test
- □ Studentized Range.

Examples

Example 1: Using PROBMC to Compute Probabilities This example shows how to use PROBMC in a DO loop to compute probabilities:

```
data probs;
   array par{5};
      par{1}=.5;
      par{2}=.51;
      par{3}=.55;
      par{4}=.45;
      par{5}=.2;
   df=40;
   q=1;
   do test="dunnett1","dunnett2", "maxmod";
      prob=probmc(test, q, ., df, 5, of par1--par5);
      put test $10. df q e18.13 prob e18.13;
   end;
run;
```

Output 4.13 on page 497 shows the results of this DATA step that are printed to the SAS log.

Output 4.13 Probabilities from PROBMC

```
DUNNETT1  40  1.00000000000E+00 4.82992188740E-01
DUNNETT2  40  1.00000000000E+00 1.64023099613E-01
MAXMOD    40  1.00000000000E+00 8.02784203408E-01
```

Example 2: Comparing Means This example shows how to compare group means to find where the significant differences lie. The data for this example is taken from a paper by Duncan (1955) (See "References" on page 626) and can also be found in Hochberg and Tamhane (1987) (See "References" on page 626). The group means are

49.6

71.2

67.6

61.5

71.3

58.1

61.0

For this data, the mean square error is $s^2 = 79.64$ ($s = 8.924$) with $\nu = 30$.

```
data duncan;
   array tr{7}$;
   array mu{7};
   n=7;
   do i=1 to n;
      input tr{i} $1. mu{i};
   end;
```

```
      input df s alpha;
      prob= 1--alpha;
         /* compute the interval */
      x = probmc("RANGE", ., prob, df, 7);
      w = x * s / sqrt(6);
         /* compare the means */
      do i = 1 to n;
         do j = i + 1 to n;
            dmean = abs(mu{i} - mu{j});
            if dmean >= w then do;
               put tr{i} tr{j} dmean;
            end;
         end;
      end;
      datalines;
   A 49.6
   B 71.2
   C 67.6
   D 61.5
   E 71.3
   F 58.1
   G 61.0
    30 8.924 .05
   ;
```

Output 4.14 on page 498 shows the results of this DATA step that are printed to the SAS log.

Output 4.14 Group Differences

```
A B 21.6
A C 18
A E 21.7
```

Example 3: Computing Confidence Intervals This example shows how to compute 95% one-sided and two-sided confidence intervals of Dunnett's test. This example and the data come from Dunnett (1955) (See "References" on page 626) and can also be found in Hochberg and Tamhane (1987) (See "References" on page 626). The data are blood count measurements on three groups of animals. As shown in the following table, the third group serves as the control, while the first two groups were treated with different drugs. The numbers of animals in these three groups are unequal.

Treatment Group:	Drug A	Drug B	Control
	9.76	12.80	7.40
	8.80	9.68	8.50
	7.68	12.16	7.20
	9.36	9.20	8.24
		10.55	9.84
			8.32

Treatment Group:	Drug A	Drug B	Control
Group Mean	8.90	10.88	8.25
n	4	5	6

The mean square error $s^2 = 1.3805$ ($s = 1.175$) with $\nu = 12$.

```
data a;
   array drug{3}$;
   array count{3};
   array mu{3};
   array lambda{2};
   array delta{2};
   array left{2};
   array right{2};

      /* input the table */
   do i = 1 to 3;
      input drug{i} count{i} mu{i};
   end;

      /* input the alpha level,    */
      /* the degrees of freedom,   */
      /* and the mean square error */
   input alpha df s;

      /* from the sample size, */
      /* compute the lambdas   */
   do i = 1 to 2;
      lambda{i} = sqrt(count{i}/
        (count{i} + count{3}));
   end;

      /* run the one-sided Dunnett's test */
   test="dunnett1";
      x = probmc(test, ., 1 - alpha, df,
                 2, of lambda1--lambda2);
      do i = 1 to 2;
         delta{i} = x * s *
            sqrt(1/count{i} + 1/count{3});
         left{i} = mu{i} - mu{3} - delta{i};
      end;
   put test $10. x left{1} left{2};

      /* run the two-sided Dunnett's test */
   test="dunnett2";
      x = probmc(test, ., 1 - alpha, df,
                 2, of lambda1--lambda2);
      do i=1 to 2;
         delta{i} = x * s *
            sqrt(1/count{i} + 1/count{3});
         left{i} = mu{i} - mu{3} - delta{i};
         right{i} = mu{i} - mu{3} + delta{i};
      end;
   put test $10. left{1} right{1};
   put test $10. left{2} right{2};
   datalines;
A 4 8.90
B 5 10.88
C 6 8.25
0.05 12 1.175
;
```

Output 4.15 on page 500 shows the results of this DATA step that are printed to the SAS log.

Output 4.15 Confidence Intervals

```
DUNNETT1  2.1210786586 -0.958751705 1.1208571303
DUNNETT2  -1.256411895 2.5564118953
DUNNETT2  0.8416271203 4.4183728797
```

Example 4: Computing Williams' Test Suppose that a substance has been tested at seven levels in a randomized block design of eight blocks. The observed treatment means are as follows:

Treatment	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

The mean square, with (7 – 1)(8 – 1) = 42 degrees of freedom, is $s^2 = 1.16$. Determine the maximum likelihood estimates M_i through the averaging process.

- □ Because $X_0 > X_1$, form $X_{0,1} = (X_0 + X_1)/2 = 10.15$.
- □ Because $X_{0,1} > X_2$, form $X_{0,1,2} = (X_0 + X_1 + X_2)/3 = (2X_{0,1} + X_2)/3 = 10.1$.
- □ $X_{0,1,2} < X_3 < X_4 < X_5$
- □ Because $X_5 > X_6$, form $X_{5,6} = (X_5 + X_6)/2 = 11.8$.

Now the order restriction is satisfied.

The maximum likelihood estimates under the alternative hypothesis are

$M_0 = M_1 = M_2 = X_{0,1,2} = 10.1$

$M_3 = X_3 = 10.6$

$M_4 = X_4 = 11.4$

$M_5 = M_6 = X_{5,6} = 11.8$

Now compute $t = (11.8 - 10.4) / \sqrt{2s^2/8} = 2.60$, and the probability that corresponds to $k = 6$, $\nu = 42$, and $t = 2.60$ is .9924467341, which shows strong evidence that there is a response to the substance. You can also compute the quantiles for the upper 5% and 1% tails, as shown in the following table.

SAS Statements	Results
`prob=probmc("williams",2.6,.,42,6);`	0.99244673
`quant5=probmc("williams",.,.95,42,6);`	1.80654052
`quant1=probmc("williams",.,.99,42,6);`	2.49087829

PROBNEGB

Returns the probability from a negative binomial distribution

Category: Probability

See: "CDF" on page 273

Syntax

PROBNEGB(*p*,*n*,*m*)

Arguments

p
is a numeric probability of success parameter.

Range: $0 \le p \le 1$

n
is an integer number of successes parameter.

Range: $n \ge 1$

m
is a positive integer random variable, the number of failures.

Range: $m \ge 0$

Details

The PROBNEGB function returns the probability that an observation from a negative binomial distribution, with probability of success *p* and number of successes *n*, is less than or equal to *m*.

To compute the probability that an observation is equal to a given value *m*, compute the difference of two probabilities from the negative binomial distribution for *m* and $m-1$.

Examples

SAS Statements	Results
x=probnegb(0.5,2,1);	**0.5**

PROBNORM

Returns the probability from the standard normal distribution

Category: Probability

See: "CDF" on page 273,

Syntax

PROBNORM(*x*)

Arguments

x
is a numeric random variable.

Details

The PROBNORM function returns the probability that an observation from the standard normal distribution is less than or equal to *x*.

Examples

SAS Statements	Results
x=probnorm(1.96);	**0.9750021049**

PROBT

Returns the probability from a *t* distribution

Category: Probability

See: "CDF" on page 273, "PDF" on page 464

Syntax

PROBT(*x*,*df*<,*nc*>)

Arguments

x
is a numeric random variable.

df
is a numeric degrees of freedom parameter.

Range: *df* > 0

nc
is an optional numeric noncentrality parameter.

Details

The PROBT function returns the probability that an observation from a Student's *t* distribution, with degrees of freedom *df* and noncentrality parameter *nc*, is less than or equal to *x*. This function accepts a noninteger degree of freedom parameter *df*. If the optional parameter, *nc*, is not specified or has the value 0, the value that is returned is from the central Student's *t* distribution.

The significance level of a two-tailed *t* test is given by

```
p=(1-probt(abs(x),df))*2;
```

Examples

SAS Statements	Results
x=probt(0.9,5);	**0.7953143998**

PUT

Returns a value using a specified format

Category: Special

Syntax

PUT(*source, format.*)

Arguments

source
identifies the SAS variable or constant whose value you want to reformat. The *source* argument can be character or numeric.

format.
contains the SAS format that you want applied to the variable or constant that is specified in the source. To override the default alignment, you can add an alignment specification to a format:

-L	left aligns the value.

-C	centers the value.
-R	right aligns the value.

Restriction: The *format.* must be of the same type as the source, either character or numeric.

Details

The format must be the same type (numeric or character) as the value of *source*. The result of the PUT function is always a character string. If the source is numeric, the resulting string is right aligned. If the source is character, the result is left aligned.

Use PUT to convert a numeric value to a character value. PUT writes (or produces a reformatted result) only while it is executing. To preserve the result, assign it to a variable.

Comparisons

The PUT statement and the PUT function are similar. The PUT function returns a value using a specified format. You must use an assignment statement to store the value in a variable. The PUT statement writes a value to an external destination (either the SAS log or a destination you specify).

Examples

Example 1: Converting Numeric Values to Character Value In this example, the first statement converts the values of CC, a numeric variable, into the four-character hexadecimal format, and the second writes the same value that the PUT function returns.

```
cchex=put(cc,hex4.);
put cc hex4.;
```

Example 2: Using PUT and INPUT Functions In this example, the PUT function returns a numeric value as a character string. The value 122591 is assigned to the CHARDATE variable. The INPUT function returns the value of the character string as a SAS date value using a SAS date informat. The value 11681 is stored in the SASDATE variable.

```
numdate=122591;
chardate=put(numdate,z6.);
sasdate=input(chardate,mmddyy6.);
```

See Also

Functions:
  "INPUT" on page 402
  "INPUTC" on page 404
  "INPUTN" on page 405
  "PUTC" on page 505,
  "PUTN" on page 506
Statement:
  "PUT" on page 962

PUTC

Enables you to specify a character format at run time

Category: Special

Syntax

PUTC(*source*, *format.*<,*w*>)

Arguments

source
is the SAS expression to which you want to apply the format.

format.
is an expression that contains the character format you want to apply to *source*.

w
specifies a width to apply to the format.

Interaction: If you specify a width here, it overrides any width specification in the format.

Comparisons

The PUTN function enables you to specify a numeric format at run time.

Examples

Example 1: Specifying a Character Format The PROC FORMAT step in this example creates a format, TYPEFMT., that formats the variable values 1, 2, and 3 with the name of one of the three other formats that this step creates. These three formats output responses of "positive," "negative," and "neutral" as different words, depending on the type of question. After PROC FORMAT creates the formats, the DATA step creates a SAS data set from raw data consisting of a number identifying the type of question and a response. After reading a record, the DATA step uses the value of TYPE to create a variable, RESPFMT, that contains the value of the appropriate format for the current type of question. The DATA step also creates another variable, WORD, whose value is the appropriate word for a response. The PUTC function assigns the value of WORD based on the type of question and the appropriate format.

```
proc format;
   value typefmt 1='$groupx'
                 2='$groupy'
                 3='$groupz';
   value $groupx 'positive'='agree'
                 'negative'='disagree'
                 'neutral'='notsure ';
   value $groupy 'positive'='accept'
                 'negative'='reject'
                 'neutral'='possible';

   value $groupz 'positive'='pass    '
                 'negative'='fail'
                 'neutral'='retest';
run;

data answers;
   length word $ 8;
   input type response $;
   respfmt = put(type, typefmt.);
   word = putc(response, respfmt);
   datalines;
 positive
 negative
 neutral
2 positive
2 negative
2 neutral
3 positive
3 negative
3 neutral
;
```

The value of the variable WORD is **`agree`** for the first observation. The value of the variable WORD is **`retest`** for the last observation.

See Also

Functions:
- "INPUT" on page 402
- "INPUTC" on page 404
- "INPUTN" on page 405
- "PUT" on page 503,
- "PUTN" on page 506

PUTN

Enables you to specify a numeric format at run time

Category: Special

Syntax

PUTN(*source*, *format.*<,*w*<,*d*>>)

Arguments

source
is the SAS expression to which you want to apply the format.

format.
is an expression that contains the numeric format you want to apply to *source*.

w
specifies a width to apply to the format.

Interaction: If you specify a width here, it overrides any width specification in the format.

d
specifies the number of decimal places to use.

Interaction: If you specify a number here, it overrides any decimal-place specification in the format.

Comparisons

The PUTC function enables you to specify a character format at run time.

Examples

Example 1: Specifying a Numeric Format The PROC FORMAT step in this example creates a format, WRITFMT., that formats the variable values 1 and 2 with the name of a SAS date format. The DATA step creates a SAS data set from raw data consisting of a number and a key. After reading a record, the DATA step uses the value of KEY to create a variable, DATEFMT, that contains the value of the appropriate date format. The DATA step also creates a new variable, DATE, whose value is the formatted value of the date. PUTN assigns the value of DATE based on the value of NUMBER and the appropriate format.

```
proc format;
   value writfmt 1='date9.'
                 2='mmddyy10.';
run;
data dates;
   input number key;
   datefmt=put(key,writfmt.);
   date=putn(number,datefmt);
   datalines;
15756 1
14552 2
;
```

See Also

Functions:

“INPUT” on page 402
“INPUTC” on page 404
“INPUTN” on page 405
“PUT” on page 503
“PUTC” on page 505

PVP

Returns the present value for a periodic cashflow stream, such as a bond

Category: Financial

Syntax

PVP(A,c,n,K,k_0,y)

Arguments

A
the par value.

Range: $A > 0$

c
the nominal per-period coupon rate, expressed as a fraction.

Range: $0 < c < 1$

n
the number of coupons per period.

Range: $n > 0$ and is an integer

K
the number of remaining coupons.

Range: $K > 0$ and is an integer

k_0
the time from present to the first coupon date, expressed in terms of the number of periods.

Range: $0 < k_0 < 1/n$

y
the nomimal per-period yield-to-maturity, expressed as a fraction.

Range: $y > 0$

Details

The PVP function is based on the relationship

$$P = \sum_{k=1}^{K} c(k) \frac{1}{\left(1 + \frac{y}{n}\right)^{t_k}}$$

where

$t_k = k - (1 - nk_0)$

$c(k) = \frac{c}{n}A$ for k=1, ..., K-1

$c(K) = \left(1 + \frac{c}{n}\right)A$

Examples

```
p=pvp(1000,1/100,4,14,.33/2,.10);
```

The value returned is 743.168.

QTR

Returns the quarter of the year from a SAS date value

Category: Date and Time

Syntax

QTR(*date*)

Arguments

date
specifies a SAS expression that represents a SAS date value.

Details

The QTR function returns a value of 1, 2, 3, or 4 from a SAS date value to indicate the quarter of the year in which a date value falls.

Examples

The following SAS statements produce these results:

SAS Statements	Results

`x='20jan94'd;`
`y=qtr(x);`
`put y=;`
	`y=1`

See Also

Function:
"YYQ" on page 621

QUOTE

Adds double quotation marks to a character value

Category: Character

Syntax

QUOTE(*argument*)

Arguments

argument
is a character value.

Details

The QUOTE function adds double quotation marks, the default character, to a character value. If double quotation marks are found within the *argument*, they are doubled in the output.

Examples

SAS Statements	Results

`x='A"B';`
`y=quote(x);`
`put y;`
	`"A""B"`

`x='A''B';`
`y=quote(x);`
`put y;`
	`"A'B"`

`x='Paul''s';`
`y=quote(x);`
`put y;`
	`"Paul's"`

RANBIN

Returns a random variate from a binomial distribution

Category: Random Number

Tip: If you want to change the seed value during execution, you must use the CALL RANBIN routine instead of the RANBIN function.

Syntax

RANBIN(*seed,n,p*)

Arguments

seed
is an integer. For more information on seeds, see "Seed Values" in "Using Random-Number Functions and CALL Routines" on page 211.

Range: $seed < 2^{31}-1$

Note: If $seed \le 0$, the time of day is used to initialize the seed stream.

n
is an integer number of independent Bernoulli trials parameter.

Range: $n > 0$

p
is a numeric probability of success parameter.

Range: $0 < p < 1$

Details

The RANBIN function returns a variate that is generated from a binomial distribution with mean np and variance $np(1-p)$. If $n \le 50$, $np \le 5$, or $n(1-p) \le 5$, an inverse transform method applied to a RANUNI uniform variate is used. If $n > 50$, $np > 5$, and $n(1-p) > 5$, the normal approximation to the binomial distribution is used. In that case, the Box-Muller transformation of RANUNI uniform variates is used.

Comparisons

The CALL RANBIN routine, an alternative to the RANBIN function, gives greater control of the seed and random number streams.

See Also

Call routine:

“CALL RANBIN” on page 248

RANCAU

Returns a random variate from a Cauchy distribution

Category: Random Number

Tip: If you want to change the seed value during execution, you must use the CALL RANCAU routine instead of the RANCAU function.

Syntax

RANCAU(*seed*)

Arguments

seed
is an integer. For more information on seeds, see “Seed Values” in “Using Random-Number Functions and CALL Routines” on page 211.

Range: *seed* < 2^{31} –1

Note: If *seed* ≤ 0, the time of day is used to initialize the seed stream.

Details

The RANCAU function returns a variate that is generated from a Cauchy distribution with location parameter 0 and scale parameter 1. An acceptance-rejection procedure applied to RANUNI uniform variates is used. If *u* and *v* are independent uniform (–1/2, 1/2) variables and $u^2 + v^2 \leq 1/4$, then *u*/*v* is a Cauchy variate. A Cauchy variate X with location parameter ALPHA and scale parameter BETA can be generated:

```
x=alpha+beta*rancau(seed);
```

Comparisons

The CALL RANCAU routine, an alternative to the RANCAU function, gives greater control of the seed and random number streams.

See Also

Call routine:

“CALL RANCAU” on page 250

RANEXP

Returns a random variate from an exponential distribution

Category: Random Number

Tip: If you want to change the seed value during execution, you must use the CALL RANEXP routine instead of the RANEXP function.

Syntax

RANEXP(*seed*)

Arguments

seed

is an integer. For more information on seeds, see “Seed Values” in “Using Random-Number Functions and CALL Routines” on page 211.

Range: *seed* $< 2^{31}-1$

Note: If *seed* ≤ 0, the time of day is used to initialize the seed stream.

Details

The RANEXP function returns a variate that is generated from an exponential distribution with parameter 1. An inverse transform method applied to a RANUNI uniform variate is used.

An exponential variate X with parameter LAMBDA can be generated:

```
x=ranexp(seed)/lambda;
```

An extreme value variate X with location parameter ALPHA and scale parameter BETA can be generated:

```
x=alpha-beta*log(ranexp(seed));
```

A geometric variate X with parameter P can be generated as follows:

```
x=floor(-ranexp(seed)/log(1-p));
```

Comparisons

The CALL RANEXP routine, an alternative to the RANEXP function, gives greater control of the seed and random number streams.

See Also

Call routine:

"CALL RANEXP" on page 252

RANGAM

Returns a random variate from a gamma distribution

Category: Random Number

Tip: If you want to change the seed value during execution, you must use the CALL RANGAM routine instead of the RANGAM function.

Syntax

RANGAM(*seed*,*a*)

Arguments

seed
is an integer. For more information on seeds, see "Seed Values" in "Using Random-Number Functions and CALL Routines" on page 211.

Range: *seed* $< 2^{31}–1$

Note: If *seed* ≤ 0, the time of day is used to initialize the seed stream.

a
is a numeric shape parameter.

Range: $a > 0$.

Details

The RANGAM function returns a variate that is generated from a gamma distribution with parameter *a*. For $a > 1$, an acceptance-rejection method due to Cheng (1977) (See "References" on page 626) is used. For $a \le 1$, an acceptance-rejection method due to Fishman is used (1978, Algorithm G2) (See "References" on page 626).

A gamma variate X with shape parameter ALPHA and scale BETA can be generated:

```
x=beta*rangam(seed,alpha);
```

If 2*ALPHA is an integer, a chi-square variate X with 2*ALPHA degrees of freedom can be generated:

```
x=2*rangam(seed,alpha);
```

If N is a positive integer, an Erlang variate X can be generated:

```
x=beta*rangam(seed,N);
```

It has the distribution of the sum of N independent exponential variates whose means are BETA.

And finally, a beta variate X with parameters ALPHA and BETA can be generated:

```
y1=rangam(seed,alpha);
y2=rangam(seed,beta);
x=y1/(y1+y2);
```

Comparisons

The CALL RANGAM routine, an alternative to the RANGAM function, gives greater control of the seed and random number streams.

See Also

Call routine:
“CALL RANGAM” on page 254

RANGE

Returns the range of values

Category: Descriptive Statistics

Syntax

RANGE(*argument*,*argument*, . . .)

Arguments

argument
is numeric. At least two arguments are required. The argument list may consist of a variable list, which is preceded by OF.

Details

The RANGE function returns the difference between the largest and the smallest of the nonmissing arguments.

Examples

SAS Statements	Results
x0=range(.,.);	**.**
x1=range(-2,6,3);	**8**
x2=range(2,6,3,.);	**4**

SAS Statements	Results
`x3=range(1,6,3,1);`	**5**
`x4=range(of x1-x3);`	**4**

RANK

Returns the position of a character in the ASCII or EBCDIC collating sequence

Category: Character

Syntax

RANK(*x*)

Arguments

x
specifies a character expression.

Details

The RANK function returns an integer that represents the position of the first character in the character expression. The result depends on your operating environment.

Examples

SAS Statements	Results	
	ASCII	EBCDIC
`n=rank('A');` `put n;`	**65**	**193**

See Also

Functions:
“BYTE” on page 241
“COLLATE” on page 291

RANNOR

Returns a random variate from a normal distribution

Category: Random Number

Tip: If you want to change the seed value during execution, you must use the CALL RANNOR routine instead of the RANNOR function.

Syntax

RANNOR(*seed*)

Arguments

seed

is an integer. For more information on seeds, see “Seed Values” in “Using Random-Number Functions and CALL Routines” on page 211.

Range: *seed* < 2^{31}–1

Note: If *seed* ≤ 0, the time of day is used to initialize the seed stream.

Details

The RANNOR function returns a variate that is generated from a normal distribution with mean 0 and variance 1. The Box-Muller transformation of RANUNI uniform variates is used.

A normal variate X with mean MU and variance S2 can be generated with this code:

```
x=MU+sqrt(S2)*rannor(seed);
```

A lognormal variate X with mean *exp*(MU + S2/2) and variance *exp*(2*MU + 2*S2) –*exp*(2*MU + S2) can be generated with this code:

```
x=exp(MU+sqrt(S2)*rannor(seed));
```

Comparisons

The CALL RANNOR routine, an alternative to the RANNOR function, gives greater control of the seed and random number streams.

See Also

Call routine:

“CALL RANNOR” on page 256

RANPOI

Returns a random variate from a Poisson distribution

Category: Random Number

Tip: If you want to change the seed value during execution, you must use the CALL RANPOI routine instead of the RANPOI function.

Syntax

RANPOI(*seed,m*)

Arguments

seed

is an integer. For more information on seeds, see "Seed Values" in "Using Random-Number Functions and CALL Routines" on page 211.

Range: *seed* $< 2^{31}-1$

Note: If *seed* ≤ 0, the time of day is used to initialize the seed stream.

m

is a numeric mean parameter.

Range: $m \geq 0$

Details

The RANPOI function returns a variate that is generated from a Poisson distribution with mean *m*. For $m < 85$, an inverse transform method applied to a RANUNI uniform variate is used (Fishman 1976) (See "References" on page 626). For $m \geq 85$, the normal approximation of a Poisson random variable is used. To expedite execution, internal variables are calculated only on initial calls (that is, with each new *m*).

Comparisons

The CALL RANPOI routine, an alternative to the RANPOI function, gives greater control of the seed and random number streams.

See Also

Call routine:

"CALL RANPOI" on page 257

RANTBL

Returns a random variate from a tabled probability

Category: Random Number

Tip: If you want to change the seed value during execution, you must use the CALL RANTBL routine instead of the RANTBL function.

Syntax

RANTBL(*seed,p*1 ,... p_i... ,p_n)

Arguments

seed

is an integer. For more information on seeds, see “Seed Values” in “Using Random-Number Functions and CALL Routines” on page 211.

Range: *seed* < $2^{31}-1$

Note: If *seed* ≤ 0, the time of day is used to initialize the seed stream.

p_i

is numeric.

Range: $0 \le p_i \le 1$ for $0 < i \le n$

Details

The RANTBL function returns a variate that is generated from the probability mass function defined by p_1 through p_n . An inverse transform method applied to a RANUNI uniform variate is used. RANTBL returns

$$\begin{array}{ll} 1 & with\, probability\, p_1 \\ 2 & with\, probability\, p_2 \\ . & \\ . & \\ . & \\ n & with\, probability\, p_n \\ n+1 & with\, probability\;\; 1-\sum_{i=1}^{n} p_i \;\; if \sum_{i=1}^{n} p_i \le 1 \end{array}$$

If, for some index $j<n$, $\sum_{i=1}^{j} p_i \ge 1$, RANTBL returns only the indices 1 through *j* with the probability of occurrence of the index *j* equal to $1-\sum_{i=1}^{j-1} p_i$.

Let n=3 and P1, P2, and P3 be three probabilities with P1+P2+P3=1, and M1, M2, and M3 be three variables. The variable X in these statements

```
array m{3} m1-m3;
x=m{rantbl(seed,of p1-p3)};
```

will be assigned one of the values of M1, M2, or M3 with probabilities of occurrence P1, P2, and P3, respectively.

Comparisons

The CALL RANTBL routine, an alternative to the RANTBL function, gives greater control of the seed and random number streams.

See Also

Call routine:

“CALL RANTBL” on page 259

RANTRI

Random variate from a triangular distribution

Category: Random Number

Tip: If you want to change the seed value during execution, you must use the CALL RANTRI routine instead of the RANTRI function.

Syntax

RANTRI(*seed,h*)

Arguments

seed

is an integer. For more information on seeds, see “Seed Values” in “Using Random-Number Functions and CALL Routines” on page 211.

Range: *seed* $< 2^{31}-1$

Note: If *seed* ≤ 0, the time of day is used to initialize the seed stream.

h

is numeric.

range: $0 < h < 1$

Details

The RANTRI function returns a variate that is generated from the triangular distribution on the interval (0,1) with parameter *h*, which is the modal value of the distribution. An inverse transform method applied to a RANUNI uniform variate is used.

A triangular distribution X on the interval [A,B] with mode C, where A ≤ C ≤ B, can be generated:

```
x=(b-a)*rantri(seed,(c-a)/(b-a))+a;
```

Comparisons

The CALL RANTRI routine, an alternative to the RANTRI function, gives greater control of the seed and random number streams.

See Also

Call routine:

"CALL RANTRI" on page 262

RANUNI

Returns a random variate from a uniform distribution

Category: Random Number

Tip: If you want to change the seed value during execution, you must use the CALL RANUNI routine instead of the RANUNI function.

Syntax

RANUNI(*seed*)

Arguments

seed

is an integer. For more information on seeds, see "Seed Values" in "Using Random-Number Functions and CALL Routines" on page 211.

Range: *seed* $< 2^{31}-1$

Note: If *seed* ≤ 0, the time of day is used to initialize the seed stream.

Details

The RANUNI function returns a number that is generated from the uniform distribution on the interval (0,1) using a prime modulus multiplicative generator with modulus $2^{31}-$ and multiplier 397204094 (Fishman and Moore 1982) (See "References" on page 626).

You can use a multiplier to change the length of the interval and an added constant to move the interval. For example,

```
random_variate=a*ranuni(seed)+b;
```

returns a number that is generated from the uniform distribution on the interval (b,a+b).

Comparisons

The CALL RANUNI routine, an alternative to the RANUNI function, gives greater control of the seed and random number streams.

See Also

Call routine:

"CALL RANUNI" on page 264

REPEAT

Repeats a character expression

Category: Character

Syntax

REPEAT(*argument*,*n*)

Arguments

argument
specifies any SAS character expression.

n
specifies the number of times to repeat *argument*.

Restriction: *n* must be greater than or equal to 0.

Details

The REPEAT function returns a character value consisting of the first argument repeated *n* times. Thus, the first argument appears *n*+1 times in the result.

Examples

SAS Statements	Results

`x=repeat('ONE',2);`
`put x;`
	ONEONEONE

RESOLVE

Returns the resolved value of an argument after it has been processed by the macro facility

Category: Macro

Syntax

RESOLVE(*argument*)

Arguments

argument
represents a macro expression.

Details

RESOLVE is fully documented in *SAS Macro Language: Reference*.

See Also

Function:
“SYMGET” on page 557

REVERSE

Reverses a character expression

Category: Character

Syntax

REVERSE(*argument*)

Arguments

argument
specifies any SAS character expression.

Examples

SAS Statements	Results
	`----+----1`

`backward=reverse('xyz  ');`
`put backward $5.;`
	`  zyx`

REWIND

Positions the data set pointer at the beginning of a SAS data set and returns a value

Category: SAS File I/O

Syntax

REWIND(*data-set-id*)

Arguments

data-set-id
specifies the data set identifier that the OPEN function returns.

Restriction: The data set cannot be opened in IS mode.

Details

REWIND returns 0 if the operation was successful, ≠0 if it was not successful. After a call to REWIND, a call to FETCH reads the first observation in the data set.

If there is an active WHERE clause, REWIND moves the data set pointer to the first observation that satisfies the WHERE condition.

Examples

This example calls FETCHOBS to fetch the tenth observation in the data set MYDATA, then calls REWIND to return to the first observation and fetch the first observation.

```
%let dsid=%sysfunc(open(mydata,i));
%let rc=%sysfunc(fetchobs(&dsid,10));
%let rc=%sysfunc(rewind(&dsid));
%let rc=%sysfunc(fetch(&dsid));
```

See Also

Functions:
- “FETCH” on page 352
- “FETCHOBS” on page 353
- “FREWIND” on page 380
- “NOTE” on page 458
- “OPEN” on page 460

RIGHT

Right aligns a character expression

Category: Character

Syntax

RIGHT(*argument*)

Arguments

argument
specifies any SAS character expression.

Details

The RIGHT function returns an argument with trailing blanks moved to the start of the value. The argument's length does not change.

Examples

SAS Statements	Results
	`----+----1----+`
`a='Due Date  ';`	
`b=right(a);`	
`put a $10.;`	**`Due Date`**
`put b $10.;`	**`  Due Date`**

See Also

Functions:

“COMPRESS” on page 296

“LEFT” on page 435

“TRIM” on page 570

ROUND

Rounds to the nearest round-off unit

Category: Truncation

Syntax

ROUND(*argument,round-off-unit*)

Arguments

argument
is numeric.

round-off-unit
is numeric and nonnegative.

Details

The ROUND function returns a value rounded to the nearest round-off unit. If *round-off-unit* is not provided, a default value of 1 is used and *argument* is rounded to the nearest integer.

Examples

SAS Statement	Results

`var1=223.456;`
`x=round(var1,1);`
`put x 9.5;`
	`223.00000`

`var2=223.456;`
`x=round(var2,.01);`
`put x 9.5;`
	`223.46000`

`x=round(223.456,100);`
`put x 9.5;`
	`200.00000`

`x=round(223.456);`
`put x 9.5;`
	`223.00000`

`x=round(223.456,.3);`
`put x 9.5;`
	`223.33333`

RXMATCH

Finds the beginning of a substring that matches a pattern and returns a value

Category: Character String Matching

Restriction: Use with the RXPARSE function

Syntax

position=**RXMATCH** (*rx, string*)

Arguments

position
specifies a numeric position in a string where the substring that is matched by the pattern begins. If there is no match, the result is zero.

rx
specifies a numeric value that is returned from the RXPARSE function.

string
specifies the character expression to be searched.

Details

RXMATCH searches the variable *string* for the pattern from RXPARSE and returns the position of the start of the string.

Comparisons

The regular expression (RX) functions and CALL routines work together to manipulate strings that match patterns. Use the RXPARSE function to parse a pattern you specify. Use the RXMATCH function and the CALL RXCHANGE and CALL RXSUBSTR routines to match or modify your data. Use the CALL RXFREE routine to free allocated space.

Example

See the RXPARSE function "Example" on page 538.

See Also

Functions and CALL routines:

"CALL RXCHANGE" on page 265
"CALL RXFREE" on page 267
"RXPARSE" on page 527
"CALL RXSUBSTR" on page 268

RXPARSE

Parses a pattern and returns a value

Category: Character String Matching

Syntax

rx=**RXPARSE**(*pattern-expression*)

Syntax Description

Arguments

rx
specifies a numeric value that is passed to other regular expression (RX) functions and call routines.

pattern-expression
specifies a character constant, variable, or expression whose value is a literal or a pattern expression. A *pattern-expression* is composed of the following elements:

string-in-quotation-marks
matches a substring consisting of the characters in the string.

letter
matches the upper- or lowercase letter in a substring.

digit
matches the digit in a substring.

period (.)
matches a period (.) in a substring.

underscore (_)
matches an underscore (_) in a substring.

?
matches any one character in a substring.

colon (:)
matches any sequence of zero or more characters in a substring.

$'*pattern*' or $"*pattern*"
matches any one character in a substring.

Tip:
Ranges of alphanumeric variables are indicated by the hyphen (-).

Example:
To match any lowercase letter, use

```
rx=rxparse("$'a-z'");
```

See:
"User-defined Character Classes" on page 531

~'*character-class*' or ^'*character-class*' or ~"*character-class*" or ^"*character-class*"
matches any one character that is *not* matched by the corresponding character class.

Tip:
Ranges of alphanumeric variables are indicated by a hyphen (-).

Example:
To *exclude* the letters a-d from the match, use

```
rx=rxparse("^'a-d'");
```

See:
"Character Class Complements" on page 531

pattern1 pattern2 or *pattern1* | | *pattern2*
selects any substring matched by pattern1 followed immediately by any substring matched by pattern2 (with no intervening blanks).

pattern1 | *pattern2*
selects any substring matched by pattern1 or any substring matched by pattern2.

Tip:	You can use an exclamation point (!) instead of a vertical bar (|).

(*pattern*)
matches a substring that contains a pattern. You can use parentheses to indicate the order in which operations are performed.

[*pattern*] or {*pattern*}
matches a substring that contains a pattern or null string.

*pattern**
matches zero or more consecutive strings matched by a pattern.

pattern+
matches one or more consecutive strings matched by a pattern.

@int
matches the position of a variable if the next character is located in the column specified by *int*. @0 matches end-of-line. If *int* is negative, it matches -*int* positions from end-of-line.

reuse-character-class
reuses a *character-class* you previously defined.

See: “Reusing Character Classes” on page 532

pattern-abbreviaton
specifies ways to shorten pattern representation.

See: “Pattern Abbreviations” on page 533, “Default Character Classes” on page 530

balanced-symbols
specifies the number of nested parentheses, brackets, braces, or less-than/greater-than symbols in a mathematical expression.

See: “Matching Balanced Symbols” on page 533

special-symbol
specifies a position in a string, or a score value.

See: “Special Symbols” on page 534

score-value
selects the pattern with the highest score value.

See: “Scores” on page 535

<pattern>
retrieves a matched substring for use in a change expression.

See: “Tag Expression” on page 535

change-expression
specifies a pattern change operation that replaces a string containing a matched substring by concatenating values to the replacement string.

See: “Change Expressions” on page 536

change-item
specifies items used for string manipulation.

See: “Change Items” on page 536

Character Classes

Using a character class element is a shorthand method for specifying a range of values for matching. In pattern matching, you can

- use default character classes
- define your own character classes
- use character class complements
- reuse character classes.

Default Character Classes You specify a default character class with a dollar sign ($) followed by a single upper- or lowercase letter. In the following list, the character class is listed in the left column and the definition is listed in the right column.

$a or $A	matches any alphabetic upper- or lowercase letter in a substring ($'a-zA-Z').
$c or $C	matches any character allowed in a version 6 SAS name that is found in a substring ($'0-9a-zA-Z_').
$d or $D	matches any digit in a substring ($'0-9').
$i or $I	matches any initial character in a version 6 SAS name that is found in a substring ($'a-zA-Z_').
$l or $L	matches any lowercase letter in a substring ($'a-z').
$u or $U	matches any uppercase letter in a substring ($'A-Z').
$w or $W	matches any white space character, such as blank, tab, backspace, carriage return, etc., in a substring.
See also:	"Character Class Complements" on page 531

Note: A hyphen appearing at the beginning or end of a character class is treated as a member of the class rather than as a range symbol. △

This statement and these values produce these matches.

```
rx=rxparse("$character-class");
```

Pattern	Input string	Position of match	Value of match
$L or $l	**3+Y STRIkeS**	9	**k**
$U or $u	**0*5x49XY**	7	**X (uppercase)**

The following example shows how to use a default character class in a DATA step.

```
data _null_;
   stringA='3+Y STRIkeS';
   rx=rxparse("$L");
   matchA = rxmatch(rx,stringA);
   valueA=substr(stringA,matchA,1);
   put 'Example A: ' matchA = valueA= ;
run;

data _null_;
   stringA2='0*5x49XY';
   rx=rxparse("$u");
   matchA2 = rxmatch(rx,stringA2);
   valueA2 = substr(stringA2, matchA2,1);
   put 'Example A2: ' matchA2 = valueA2= ;
run;
```

The SAS log shows the following results:

```
Example A: matchA=9 valueA=k
Example A2: matchA2=7 valueA2=X
```

User-defined Character Classes A user-defined character class begins with a dollar sign ($) and is followed by a string in quotation marks. A character class matches any one character within the quotation marks.

Note: Ranges of values are indicated by a hyphen (-). △

This statement and these values produce these matches.

```
rx=rxparse("$'pattern'");
```

Pattern	Input string	Position of match	Value of match
$'abcde'	**3+yE strikes**	11	**e**
$'1-9'	**z0*549xy**	4	**5**

The following example shows how to use a user-defined character class in a DATA step.

```
data _null_;
   stringB='3+yE strikes';
   rx=rxparse("$'abcde'");
   matchB = rxmatch(rx,stringB);
   valueB=substr(stringB,matchB,1);
   put 'Example B: ' matchB= valueB= ;
run;

data _null_;
   stringB2='z0*549xy';
   rx=rxparse("$'1-9'");
   matchB2=rxmatch(rx,stringB2);
   valueB2=substr(stringB2,matchB2,1);
   put 'Example B2: ' matchB2= valueB2= ;
run;
```

The SAS log shows the following results:

```
Example B: matchB=11 valueB=e
Example B2: matchB2=4 valueB2=5
```

You can also define your own character class complements. For details about character class complements, see "Character Class Complements" on page 531.

Character Class Complements A character class complement begins with a caret (^) or a tilde (~) and is followed by a string in quotation marks. A character class complement matches any one character that is *not* matched by the corresponding character class. For details about character classes, see "Character Classes" on page 529.

This statement and these values produce these matches.

```
rx=rxparse(^character-class | ~character-class);
```

Pattern	Input string	Position of match	Value of match
^u or ~u	**0*5x49XY**	1	**0**
^'A-z' or ~'A-z'	**Abc de45**	4	**the first space**

The following example shows how to use a character class complement in a DATA step.

```
data _null_;
   stringC='0*5x49XY';
   rx=rxparse('^u');
   matchC = rxmatch(rx,stringC);
   valueC=substr(stringC,matchC,1);
   put 'Example C: ' matchC = valueC=;
run;

data _null_;
   stringC2='Abc de45';
   rx=rxparse("~'A-z'");
   matchC2=rxmatch(rx,stringC2);
   valueC2=substr(stringC2,matchC2,1);
   put 'Example C2: ' matchC2= valueC2= ;
run;
```

The SAS log shows the following results:

```
Example C: matchC=1 valueC=0
Example C2: matchC2=4 valueC2=
```

Reusing Character Classes You can reuse character classes you previously defined by using one of the following patterns:

$*int*
reuses the *int*th character class.

Restriction: *int* is a nonzero integer.

Example: If you defined a character class in a pattern and want to use the same character class again in the same pattern, use $*int* to refer to the *int*th character class you defined. If *int* is negative, count backwards from the last pattern to identify the character class for -*int*. For example,

```
rx=rxparse("$'AB' $1 $'XYZ' $2 $-2");
```

is equivalent to

```
rx=rxparse("$'AB' $'AB' $'XYZ' $'XYZ' $'AB'");
```

- The $1 element in the first code sample is replaced by AB in the second code sample, because AB was the first pattern defined.
- The $2 element in the first code sample is replaced by XYZ in the second code sample, because XYZ was the second pattern defined.
- The $-2 element in the first code sample is replaced by AB in the second code sample, because AB is the second-to-the-last pattern defined.

~*int* or ^*int*
reuses the complement of the *int*'th character class.

Restriction: *int* is a nonzero integer.

Example: This example shows character-class elements ($'Al', $'Jo', $'Li') and reuse numbers ($1, $2, $3, ~2):

```
rx=rxparse($'Al' $1 $'Jo' $2 $'Li' $3 ~2);
```

is equivalent to

```
rx=rxparse($'Al' $'Al' $'Jo' $'Jo'
           $'Li' $'Li' $'Al' $'Li');
```

The ~2 matches patterns 1 (Al) and 3 (Li), and excludes pattern 2 (Jo).

Pattern Abbreviations

You can use the following list of elements in your pattern:

$f or $F	matches a floating point number.
$n or $N	matches a SAS name.
$p or $P	indicates a prefix option.
$q or $Q	matches a string in quotation marks.
$s or $S	indicates a suffix option.

This statement and input string produce these matches.

`rx=rxparse(`*`$pattern-abbreviation pattern`*`);`

Pattern	Input string	Position of match	Value of match
`$p wood`	**`woodchucks eat wood`**	1	**`characters "wood" in woodchucks`**
`wood $s`	**`woodchucks eat wood`**	20	**`wood`**

The following example shows how to use a pattern abbreviation in a DATA step.

```
data _null_;
  stringD='woodchucks eat firewood';
  rx=rxparse("$p 'wood'");
  PositionOfMatchD=rxmatch(rx,stringD);
  call rxsubstr(rx,stringD,positionD,lengthD);
  valueD=substr(stringD,PositionOfMatchD);
  put 'Example D: ' lengthD= valueD= ;
run;

data _null_;
  stringD2='woodchucks eat firewood';
  rx=rxparse("'wood' $s");
  PositionOfMatchD2=rxmatch(rx,stringD2);
  call rxsubstr(rx,stringD2,positionD2,lengthD2);
  valueD2=substr(stringD2,PositionOfMatchD2);
  put 'Example D2: ' lengthD2= valueD2= ;
run;
```

The SAS log shows the following results:

```
Example D: lengthD=4 valueD=woodchucks eat firewood
Example D2: lengthD2=4 valueD2=wood
```

Matching Balanced Symbols

You can match mathematical expressions containing multiple sets of balanced parentheses, brackets, braces, and less-than/greater-than symbols. Both the symbols and the expressions within the symbols are part of the match:

$(*int*) or $[*int*] or ${*int*} or $<*int*>
indicates the *int* level of nesting you specify.

Restriction: *int* is a positive integer.

Tip: Using smaller values increases the efficiency of finding a match.

Example: This statement and input string produces this match.

```
rx=rxparse("$(2)");
```

Input string	Position of match	Value of match
(((a+b)*5)/43)	2	**((a+b)*5)**

The following example shows how to use mathematical symbol matching in a DATA step.

```
data _null_;
   stringE='(((a+b)*5)/43)';
      rx=rxparse("$(2)");
      call rxsubstr(rx,stringE,positionE,lengthE);
      PositionOfMatchE=rxmatch(rx,stringE);
      valueE=substr(stringE,PositionOfMatchE);
      put 'Example E: ' lengthE= valueE= ;
run;
```

The SAS log shows the following results:

```
Example E: lengthE=9 valueE=((a+b)*5)/43)
```

Special Symbols

You can use the following list of special symbols in your pattern:

\ sets the beginning of a match to the current position.

/ sets the end of a match to the current position.

Restriction: If you use a backward slash (\) in one alternative of a union (|), you must use a forward slash (/) in all alternatives of the union, or in a position preceding or following the union.

$# requests the match with the highest score, regardless of the starting position.

Tip: The position of this symbol within the pattern is not significant.

$- scans a string from right to left.

Tip: The position of this symbol within the pattern is not significant.

Tip: Do not confuse a hyphen (-) used to scan a string with a hyphen used in arithmetic operations.

$@ requires the match to begin where the scan of the text begins.

Tip: The position of this symbol within the pattern is not significant.

The following table shows how a pattern matches an input string.

Pattern	Input string	Value of match
c\ow	**How now brown cow?**	**characters "ow" in cow**
ow/n	**How now brown cow?**	**characters "ow" in brown**
@3:\ow	**How now brown cow?**	**characters "ow" in now**

The following example shows how to use special symbol matching in a DATA step.

```
data _null_;
   stringF='How now brown cow?';
   rx=rxparse("$'c\ow'");
   matchF=rxmatch(rx,stringF);
   valueF=substr(stringF,matchF,2);
   put 'Example F= ' matchF= valueF= ;
run;

data _null_;
   stringF2='How now brown cow?';
   rx=rxparse("@3:\ow");
   matchF2=rxmatch(rx,stringF2);
   valueF2=substr(stringF2,matchF2,2);
   put 'Example F2= ' matchF2= valueF2= ;
run;
```

The SAS log shows the following results:

```
Example F= matchF=2 valueF=ow
Example F2= matchF2=6 valueF2=ow
```

Scores

When a pattern is matched by more than one substring beginning at a specific position, the longest substring is selected. To change the selection criterion, assign a score value to each substring by using the pound sign (#) special symbol followed by an integer.

The score for any substring begins at zero. When *#int* is encountered in the pattern, the value of *int* is added to the score. If two or more matching substrings begin at the same leftmost position, SAS selects the substring with the highest score value. If two substrings begin at the same leftmost position and have the same score value, SAS selects the longer substring. The following is a list of score representations:

#*int*	adds *int* to the score, where *int* is a positive or negative integer.
#**int*	multiplies the score by nonnegative *int*.
#/*int*	divides the score by positive *int*.
#=*int*	assigns the value of *int* to the score.
#>*int*	finds a match if the current score exceeds *int*.

Tag Expression

You can assign a substring of the string being searched to a character variable with the expression ***name=<pattern>***, where *pattern* specifies any pattern expression. The substring matched by this expression is assigned to the variable *name*.

If you enclose a pattern in less-than/greater-than symbols (<>) and do not specify a variable name, SAS automatically assigns the pattern to a variable. SAS assigns the variable _1 to the first occurrence of the pattern, _2 to the second occurrence, etc. This assignment is called tagging. SAS tags the corresponding substring of the matched string.

The following shows the syntax of a tag expression:

<*pattern*>
: specifies a pattern expression. SAS assigns a variable to each occurrence of *pattern* for use in a change expression.

Change Expressions

If you find a substring that matches a pattern, you can change the substring to another value. You must specify the pattern expression, use the TO keyword, and specify the change expression in the argument for RXPARSE. You can specify a list of pattern change expressions by separating each expression with a comma.

A pattern change operation replaces a matched string by concatenating values to the replacement string. The operation concatenates

- all characters to the left of the match
- the characters specified in the change expression
- all characters to the right of the match.

You can have multiple parallel operations within the RXPARSE argument. In the following example,

```
rx=rxparse("x TO y, y TO x");
```

x in a substring is substituted for **y**, and **y** in a substring is substituted for **x**.

A change expression can include the items in the following list. Each item in the list is followed by the description of the value concatenated to the replacement string at the position of the pointer.

string in quotation marks
: concatenates the contents of the string.

name
: concatenates the name, possibly in a different case.

number
: concatenates the number.

period (.)
: concatenates the period (.).

underscore (_)
: concatenates the underscore (_).

=*int*
: concatenates the value of the *int*th tagged substring if *int* is positive, or the -*int*th-from-the-last tagged substring if *int* is negative. In a parallel change expression, the *int*th or -*int*th-from-the-last tag is counted within the component of the parallel change expression that yielded the match, and not over the entire parallel change expression.

==
: concatenates the entire matched substring.

Change Items

You can use the items in the following list to manipulate the replacement string. The items position the cursor without affecting the replacement string.

@*int*	moves the pointer to column *int* where the next string added to the replacement string will start.
@=	moves the pointer one column past the end of the matched substring.
>*int*	moves the pointer to the right to column *int*. If the pointer is already to the right of column *int*, the pointer is not moved.
>=	moves the pointer to the right, one column past the end of the matched substring.
<*int*	moves pointer to the left to column *int*. If the pointer is already to the left of column *int*, the pointer is not moved.
<=	moves the pointer to the left, one column past the end of the matched substring.
+*int*	moves the pointer *int* columns to the right.
-*int*	moves the pointer *int* columns to the left.
-L	left-aligns the result of the previous item or expression in parentheses.
-R	right-aligns the result of the previous item or expression in parentheses.
-C	centers the result of the previous item or expression in parentheses.
**int*	repeats the result of the previous item or expression in parentheses *int*-1 times, producing a total of *int* copies.

Details

General Information

- When creating a pattern for matching, make the pattern as short as possible for greater efficiency. The time required for matching is roughly proportional to the length of the pattern times the length of the string that is searched.
- The algorithm used by the regular expression (RX) functions and CALL routines is a nondeterministic finite automaton.

Using Quotation Marks in Expressions

- To specify a literal that begins with a single quotation mark, use two single quotation marks instead of one.
- Literals inside a pattern must be enclosed by another layer of quotation marks. For example, `'' 'O' '' connor''` matches an uppercase O, followed by a single quotation mark, followed by the letters "connor" in either upper or lower case.

Comparisons

The regular expression (RX) functions and CALL routines work together to manipulate strings that match patterns. Use the RXPARSE function to parse a pattern you specify. Use the RXMATCH function and the CALL RXCHANGE and CALL RXSUBSTR routines to match or modify your data. Use the CALL RXFREE routine to free allocated space.

Note: Use RXPARSE only with other regular expression (RX) functions and CALL routines. △

Example

The following example uses RXPARSE to parse an input string and change the value of the string.

```
data test;
   input string $;
   datalines;
abcxyzpq
xyyzxyZx
x2z..X7z
;

data _null_;
  set;
  length to $20;
  if _n_=1 then
     rx=rxparse("' x < ? > 'z' to ABC =1 '@#%'");
  retain rx;
  drop rx;
  put string=;
  match=rxmatch(rx,string);
     put @3 match=;
  call rxsubstr(rx,string,position);
     put @3 position=;
  call rxsubstr(rx,string,position,length,score);
     put @3 position= Length= Score=;
  call rxchange(rx,999,string,to);
     put @3 to=;
  call rxchange(rx,999,string);
     put @3 'New ' string=;
```

```
 run;
```

```
      cpu time            0.05 seconds

1  data test;
2     input string $;
3     datalines;
NOTE: The data set WORK.TEST has 3 observations and 1 variables.
NOTE: DATA statement used:
      real time           0.34 seconds
      cpu time            0.21 seconds

7  ;
8
9  data _null_;
10    set;
11    length to $20;
12    if _n_=1 then
13       rx=rxparse("‘ x < ? > ’z’ to ABC =1 ’@#%’");
14    retain rx;
15    drop rx;
16    put string=;
17    match=rxmatch(rx,string);
18       put @3 match=;
19    call rxsubstr(rx,string,position);
20       put @3 position=;
21    call rxsubstr(rx,string,position,length,score);
22       put @3 position= Length= Score=;
23    call rxchange(rx,999,string,to);
24       put @3 to=;
25    call rxchange(rx,999,string);
26       put @3 ’New ’ string=;
27  run;
string=abcxyzpq
  match=4
  position=4
  position=4 length=3 score=0
  to=abcabcy@#%pq
  New string=abcabcy@
string=xyyzxyZx
  match=0
  position=0
  position=0 length=0 score=0
  to=xyyzxyZx
  New string=xyyzxyZx
string=x2z..X7z
  match=1
  position=1
  position=1 length=3 score=0
  to=abc2@#%..Abc7@#%
  New string=abc2@#%.
NOTE: DATA statement used:
      real time           0.67 seconds
      cpu time            0.45 seconds
```

See Also

Functions and CALL routines:
- "CALL RXCHANGE" on page 265
- "CALL RXFREE" on page 267
- "RXMATCH" on page 526
- "CALL RXSUBSTR" on page 268

Aho, Hopcroft, and Ullman, Chapter 9 (See "References" on page 626)

SAVING

Returns the future value of a periodic saving

Category: Financial

Syntax

SAVING(*f*,*p*,*r*,*n*)

Arguments

f
is numeric, the future amount (at the end of *n* periods).
Range: $f \geq 0$

p
is numeric, the fixed periodic payment.
Range: $p \geq 0$

r
is numeric, the periodic interest rate expressed as a decimal.
Range: $r \geq 0$

n
is an integer, the number of compounding periods.
Range: $n \geq 0$

Details

The SAVING function returns the missing argument in the list of four arguments from a periodic saving. The arguments are related by

$$f = \frac{p\,(1+r)\left((1+r)^n - 1\right)}{r}$$

One missing argument must be provided. It is then calculated from the remaining three. No adjustment is made to convert the results to round numbers.

Examples

A savings account pays a 5 percent nominal annual interest rate, compounded monthly. For a monthly deposit of $100, the number of payments that are needed to accumulate at least $12,000, can be expressed as

```
number=saving(12000,100,.05/12,.);
```

The value returned is 97.18 months. The fourth argument is set to missing, which indicates that the number of payments is to be calculated. The 5 percent nominal annual rate is converted to a monthly rate of 0.05/12. The rate is the fractional (not the percentage) interest rate per compounding period.

SCAN

Selects a given word from a character expression

Category: Character

Syntax

SCAN(*argument,n*<, *delimiters*>)

Arguments

argument
specifies any character expression.

n
specifies a numeric expression that produces the number of the word in the character string you want SCAN to select.

Tip: If *n* is negative, SCAN selects the word in the character string starting from the end of the string. If |*n*| is greater than the number of words in the character string, SCAN returns a blank value.

delimiters
specifies a character expression that produces characters that you want SCAN to use as word separators in the character string.

Default: If you omit *delimiters* in an ASCII environment, SAS uses the following characters:

blank . < (+ & ! $ *); ^ – / , % |

In ASCII environments without the ^ character, SCAN uses the ~ character instead.

If you omit *delimiters* on an EBCDIC environment, SAS uses the following characters:

blank . < (+ | & ! $ *); ¬ – / , % | ¢

Tip: If you represent *delimiters* as a constant, enclose *delimiters* in quotation marks.

Details

Leading delimiters before the first word in the character string do not effect SCAN. If there are two or more contiguous delimiters, SCAN treats them as one.

Examples

SAS Statements	Results

`arg='ABC.DEF(X=Y)';`
`word=scan(arg,3);`
`put word;`
	`X=Y`

`word=scan(arg,-3);`
`put word;`
	`ABC`

SDF

Computes a survival function

Category: Probability

See: "CDF" on page 273

Syntax

SDF('*dist*',*quantile*,*parm-1*, . . . ,*parm-k*)

The SDF function computes the survival function (upper tail) for specified distributions.

SECOND

Returns the second from a SAS time or datetime value

Category: Date and Time

Syntax

SECOND(*time* | *datetime*)

Arguments

time
specifies a SAS expression that represents a SAS time value.

datetime
specifies a SAS expression that represents a SAS datetime value.

Details

The SECOND function produces a positive, numeric value that represents a specific second of the minute. The result ranges from 0 through 59.

Examples

SAS Statements	Result

`time='3:19:24't;`
`s=second(time);`
`put s;`
	`24`

See Also

Functions:
“HOUR” on page 395
“MINUTE” on page 445

SIGN

Returns the sign of a value

Category: Mathematical

Syntax

SIGN(*argument*)

Required argument

argument
is numeric.

Details

The SIGN function returns a value of

-1	if $x < 0$
0	if $x = 0$
1	if $x > 0$.

Examples

SAS Statements	Results
`x=sign(-5);`	`-1`
`x=sign(5);`	`1`
`x=sign(0);`	`0`

SIN

Returns the sine

Category: Trigonometric

Syntax

SIN(*argument*)

Arguments

argument
is numeric and is specified in radians.

Examples

SAS Statements	Results
`x=sin(0.5);`	`0.4794255386`
`x=sin(0);`	`0`
`x=sin(3.14159/4);`	`.7071063121`

SINH

Returns the hyperbolic sine

Category: Hyperbolic

Syntax

SINH(*argument*)

Arguments

argument
is numeric.

Details

The SINH function returns the hyperbolic sine of the argument, which is given by

$$\left(e^{argument} - e^{-argument}\right)/2$$

Examples

SAS Statements	Results
`x=sinh(0);`	**`0`**
`x=sinh(1);`	**`1.1752011936`**
`x=sinh(-1.0);`	**`-1.175201194`**

SKEWNESS

Returns the skewness

Category: Descriptive Statistics

Syntax

SKEWNESS(*argument,argument,argument, . . .*)

Arguments

argument
is numeric. At least three arguments are required. The argument list may consist of a variable list, which is preceded by OF.

Examples

SAS Statements	Results
`x1=skewness(0,1,1);`	`-1.732050808`
`x2=skewness(2,4,6,3,1);`	`0.5901286564`
`x3=skewness(2,0,0);`	`1.7320508076`
`x4=skewness(of x1-x3);`	`-0.953097714`

SOUNDEX

Encodes a string to facilitate searching

Category: Character

Syntax

SOUNDEX(*argument*)

Arguments

argument
specifies any SAS character expression.

Details

The SOUNDEX function encodes a character string according to an algorithm originally developed by Margaret K. Odell and Robert C. Russel (US Patents 1261167 (1918) and 1435663 (1922)). The algorithm is described in Knuth, *The Art of Computer Programming, Volume 3* (See "References" on page 626).

The SOUNDEX function returns a copy of the *argument* encoded by using the following steps.

1 Retain the first letter in the *argument* and discard the following letters:

A E H I O U W Y

2 Assign the following numbers to these classes of letters:

1: B F P V

2: C G J K Q S X Z

3: D T

4: L

5: M N

6: R

3 If two or more adjacent letters have the same classification from Step 2, discard all but the first. (Adjacent refers to the position in the word prior to discarding letters.)

The algorithm described in Knuth adds trailing zeroes and truncates the result to the length of 4. You can perform these operations with other SAS functions.

Examples

SAS Statements	Results
`x=soundex('Paul');` `put x;`	`P4`
`word='amnesty';` `x=soundex(word);` `put x;`	`A523`

SPEDIS

Determines the likelihood of two words matching, expressed as the asymmetric spelling distance between the two words

Category: Character

Syntax

SPEDIS(*query*,*keyword*)

Arguments

query
identifies the word to query for the likelihood of a match. SPEDIS removes trailing blanks before comparing the value.

keyword
specifies a target word for the query. SPEDIS removes trailing blanks before comparing the value.

Details

SPEDIS returns the distance between the query and a keyword, a nonnegative value usually less than 100, never greater than 200 with the default costs.

SPEDIS computes an asymmetric spelling distance between two words as the normalized cost for converting the keyword to the query word via a sequence of operations. SPEDIS(QUERY, KEYWORD) is NOT the same as SPEDIS(KEYWORD, QUERY).

Costs for each operation that is required to convert the keyword to the query are

Operation	Cost	Explanation
match	0	no change
singlet	25	delete one of a double letter
doublet	50	double a letter
swap	50	reverse the order of two consecutive letters
truncate	50	delete a letter from the end
append	35	add a letter to the end
delete	50	delete a letter from the middle
insert	100	insert a letter in the middle
replace	100	replace a letter in the middle
firstdel	100	delete the first letter
firstins	200	insert a letter at the beginning
firstrep	200	replace the first letter

The distance is the sum of the costs divided (in integer arithmetic) by the length of the query.

Examples

```
options nodate pageno=1 linesize=64;
data words;
   input oper $ query $ keyword $;
   dist = spedis(query,keyword);
   cost = dist * length(query);
   put oper $10. query $10. keyword $10.
       dist 5. cost 5.;
datalines;
match       fuzzy        fuzzy
singlet     fuzy         fuzzy
doublet     fuuzzy       fuzzy
swap        fzuzy        fuzzy
truncate    fuzz         fuzzy
append      fuzzys       fuzzy
delete      fzzy         fuzzy
insert      fluzzy       fuzzy
replace     fizzy        fuzzy
firstdel    uzzy         fuzzy
firstins    pfuzzy       fuzzy
firstrep    wuzzy        fuzzy
several     floozy       fuzzy
;

proc print data = words;
run;
```

The output from the DATA step is as follows:

```
                       The SAS System                          1
      OBS    OPER        QUERY     KEYWORD    DIST    COST

        1    match       fuzzy      fuzzy       0        0
        2    singlet     fuzy       fuzzy       6       24
        3    doublet     fuuzzy     fuzzy       8       48
        4    swap        fzuzy      fuzzy      10       50
        5    truncate    fuzz       fuzzy      12       48
        6    append      fuzzys     fuzzy       5       30
        7    delete      fzzy       fuzzy      12       48
        8    insert      fluzzy     fuzzy      16       96
        9    replace     fizzy      fuzzy      20      100
       10    firstdel    uzzy       fuzzy      25      100
       11    firstins    pfuzzy     fuzzy      33      198
       12    firstrep    wuzzy      fuzzy      40      200
       13    several     floozy     fuzzy      50      300
```

SQRT

Returns the square root of a value

Category: Mathematical

Syntax

SQRT(*argument*)

Arguments

argument
is numeric and must be nonnegative.

Examples

SAS Statements	Results
`x=sqrt(36);`	**6**
`x=sqrt(25);`	**5**
`x=sqrt(4.4);`	**2.0976176963**

STD

Returns the standard deviation

Category: Descriptive Statistics

Syntax

STD(*argument,argument*, . . .)

Arguments

argument
is numeric. At least two arguments are required. The argument list may consist of a variable list, which is preceded by OF.

Examples

SAS Statements	Results
`x1=std(2,6);`	**2.8284271247**
`x2=std(2,6,.);`	**2.8284271427**
`x3=std(2,4,6,3,1);`	**1.9235384062**
`x4=std(of x1-x3);`	**0.5224377453**

STDERR

Returns the standard error of the mean

Category: Descriptive Statistics

Syntax

STDERR(*argument,argument*, . . .)

Arguments

argument
is numeric. At least two arguments are required. The argument list may consist of a variable list, which is preceded by OF.

Examples

SAS Statements	Results
`x1=stderr(2,6);`	`2`
`x2=stderr(2,6,.);`	`2`
`x3=stderr(2,4,6,3,1);`	`0.8602325267`
`x4=stderr(of x1-x3);`	`0.3799224911`

STFIPS

Converts state postal codes to FIPS state codes

Category: State and ZIP Code

Syntax

STFIPS(*postal-code*)

Arguments

postal-code
specifies a character expression that contains the two-character standard state postal code. Characters can be mixed case. The function ignores trailing blanks, but generates an error if the expression contains leading blanks.

Details

The STFIPS function converts a two-character state postal code (or world-wide GSA geographic code for U.S. territories) to the corresponding numeric U.S. Federal Information Processing Standards (FIPS) code.

Comparisons

The STFIPS, STNAME, and STNAMEL functions take the same argument but return different values. STFIPS returns a numeric U.S. Federal Information Processing Standards (FIPS) code. STNAME returns an uppercase state name. STNAMEL returns a mixed case state name.

Examples

The examples show the differences when using STFIPS, STNAME, and STNAMEL.

SAS Statements	Results

`fips=stfips ('NC');`
`put fips;`
	37

`state=stname('NC');`
`put state;`
	NORTH CAROLINA

`state=stnamel('NC');`
`put state;`
	North Carolina

See Also

Functions:
- "FIPNAME" on page 364
- "FIPNAMEL" on page 365
- "FIPSTATE" on page 366
- "STNAME" on page 552,
- "STNAMEL" on page 553

STNAME

Converts state postal codes to uppercase state names

Category: State and ZIP Code

Syntax

STNAME(*postal-code*)

Arguments

postal-code
specifies a character expression that contains the two-character standard state postal code. Characters can be mixed case. The function ignores trailing blanks, but generates an error if the expression contains leading blanks.

Details

The STNAME function converts a two-character state postal code (or world-wide GSA geographic code for U.S. territories) to the corresponding state name in uppercase. Returned values can contain up to 20 characters.

Comparisons

The STFIPS, STNAME, and STNAMEL functions take the same argument but return different values. STFIPS returns a numeric U.S. Federal Information Processing Standards (FIPS) code. STNAME returns an uppercase state name. STNAMEL returns a mixed case state name.

Examples

SAS Statements	Results

`fips=stfips ('NC');`
`put fips;`
	`37`

`state=stname('NC');`
`put state;`
	`NORTH CAROLINA`

`state=stnamel('NC');`
`put state;`
	`North Carolina`

See Also

Functions:
- "FIPNAME" on page 364
- "FIPNAMEL" on page 365
- "FIPSTATE" on page 366
- "STFIPS" on page 551, "STNAMEL" on page 553

STNAMEL

Converts state postal codes to mixed case state names

Category: State and ZIP Code

Syntax

STNAMEL(*postal-code*)

Arguments

postal-code
specifies a character expression that contains the two-character standard state postal code. Characters can be mixed case. The function ignores trailing blanks, but generates an error if the expression contains leading blanks.

Details

The STNAMEL function converts a two-character state postal code (or world-wide GSA geographic code for U.S. territories) to the corresponding state name in mixed case. Returned values can contain up to 20 characters.

Comparisons

The STFIPS, STNAME, and STNAMEL functions take the same argument but return different values. STFIPS returns a numeric U.S. Federal Information Processing

Standards (FIPS) code. STNAME returns an uppercase state name. STNAMEL returns a mixed case state name.

Examples

The examples show the differences when using STFIPS, STNAME, and STNAMEL.

SAS Statements	Results

`fips=stfips ('NC');`
`put fips;`
	37

`state=stname('NC');`
`put state;`
	NORTH CAROLINA

`state=stnamel('NC');`
`put state;`
	North Carolina

See Also

Functions:
- "FIPNAME" on page 364
- "FIPNAMEL" on page 365
- "FIPSTATE" on page 366
- "STFIPS" on page 551

SUBSTR (left of =)

Replaces character value contents

Category: Character

Syntax

SUBSTR(*argument,position*<,*n*>)=*characters-to-replace*

Arguments

argument
specifies a character variable.

position
specifies a numeric expression that is the beginning character position.

n
specifies a numeric expression that is the length of the substring that will be replaced.

Restriction: *n* can not be larger than the length of the expression that remains in *argument* after *position*.

Tip: If you omit *n* SAS uses all of the characters on the right side of the assignment statement to replace the values of *argument*.

characters-to-replace
specifies a character expression that will replace the contents of *argument*.

Tip: Enclose a literal string of characters in quotation marks.

Details

When you use the SUBSTR function on the left side of an assignment statement, SAS places the value of *argument* with the expression on right side. SUBSTR replaces *n* characters starting at the character you specify in *position*.

Examples

SAS Statements	Results
`a='KIDNAP';`	
`substr(a,1,3)='CAT';`	
`put a;`	`CATNAP`
`b=a;`	
`substr(b,4)='TY';`	
`put b;`	`CATTY`

See Also

Function:
"SUBSTR (right of =)" on page 555

SUBSTR (right of =)

Extracts a substring from an argument

Category: Character

Syntax

<*variable*=>**SUBSTR**(*argument,position*<*,n*>)

Arguments

variable
specifies a valid SAS variable name.

argument
specifies any SAS character expression.

position
specifies a numeric expression that is the beginning character position.

n

specifies a numeric expression that is the length of the substring to extract.

Interaction: If *n* is larger than the length of the expression that remains in *argument* after *position*, SAS extracts the remainder of the expression.

Tip: If you omit *n*, SAS extracts the remainder of the expression.

Details

The SUBSTR function returns a portion of an expression that you specify in *argument*. The portion begins with the character specified by *position* and is the number of characters specified by *n*.

A variable that is created by SUBSTR obtains its length from the length of *argument*.

Examples

SAS Statements	Results
	`----+----1----+----2`
`date='06MAY98';`	
`month=substr(date,3,3);`	
`year=substr(date,6,2);`	
`put @1 month @5 year;`	`MAY 98`

See Also

Function:

“SUBSTR (left of =)” on page 554

SUM

Returns the sum of the nonmissing arguments

Category: Descriptive Statistics

Syntax

SUM(*argument*,*argument*, ...)

Arguments

argument

is numeric. The argument list can consist of a variable list, which is preceded by OF.

Examples

SAS Statements	Results
`x1=sum(4,9,3,8);`	`24`
`x2=sum(4,9,3,8,.);`	`24`
`x3=sum(of x1-x2);`	`48`
`x4=sum(of x1-x3, 5);`	`101`

`y1=20;`
`y2=30;`
`x5=sum(of y:);`
	`50`

SYMGET

Returns the value of a macro variable during DATA step execution

Category: Macro

Syntax

SYMGET(*argument*)

Arguments

argument
is a character expression that identifies the macro variable whose value you want to retrieve.

Details

The SYMGET function returns the value of a macro variable during DATA step execution. SYMGET is documented in *SAS Macro Language: Reference.*

See Also

SAS Macro Language: Reference

SYSGET

Returns the value of the specified operating environment variable

Category: Special

Syntax

SYSGET(*operating-environment-variable*)

Arguments

operating-environment-variable
is the name of an operating environment variable. The case of *operating-environment-variable* must agree with the case that is stored in the operating environment. Trailing blanks in the argument of SYSGET are significant. Use the TRIM function to remove them.

Operating Environment Information: The term *operating-environment-variable* used in the description of this function refers to a name that represents a numeric, character, or logical value in the operating environment. Refer to the SAS documentation for your operating environment for details. △

Details

If the value of the operating environment variable is truncated or the variable is not defined in the operating environment, SYSGET displays a warning message in the SAS log.

Examples

This example obtains the value of two environment variables in the UNIX environment:

```
data _null_;
   length result $200;
   input env_var $;
   result=sysget(trim(env_var));
   put env_var= result=;
   datalines;
USER
PATH
;
```

Executing this DATA step for user ABCDEF displays these lines:

```
ENV_VAR=USER RESULT=abcdef
ENV_VAR=PATH RESULT=path-for-abcdef
```

SYSMSG

Returns the text of error messages or warning messages from the last data set or external file function execution

Category: SAS File I/O

Category: External Files

Syntax

SYSMSG()

Details

SYSMSG returns the text of error messages or warning messages that are produced when a data set or external file access function encounters an error condition. If no error message is available, the returned value is blank. The internally stored error message is reset to blank after a call to SYSMSG, so subsequent calls to SYSMSG before another error condition occurs return blank values.

Example

This example uses SYSMSG to write to the SAS log the error message generated if FETCH cannot copy the next observation into the Data Set Data Vector. The return code is 0 only when a record is fetched successfully:

```
%let rc=%sysfunc(fetch(&dsid));
%if &rc ne 0 %then
   %put %sysfunc(sysmsg());
```

See Also

Functions:
- "FETCH" on page 352
- "SYSRC" on page 561

SYSPARM

Returns the system parameter string

Category: Special

Syntax

SYSPARM()

Details

SYSPARM allows you to access a character string specified with the SYSPARM= system option at SAS invocation or in an OPTIONS statement.

Note: If the SYSPARM= system option is not specified, the SYSPARM function returns a null string. △

Example

This example shows the SYSPARM= system option and the SYSPARM function.

```
options sysparm='yes';
data a;
```

```
      If sysparm()='yes' then
         do;
         ...SAS Statements...
         end;
run;
```

See Also

System option:
  "SYSPARM=" on page 1165

SYSPROD

Determines if a product is licensed

Category: Special

Syntax

SYSPROD(*product-name*)

Arguments

product-name
specifies a character expression that resolves to the name of a SAS product.

Details

The SYSPROD function returns 1 if a specific SAS Institute software product is licensed, 0 if it is a SAS Institute software product but not licensed for your system, and -1 if the product name is not recognized. Use SYSPROD in the DATA step, in an IML step, or in an SCL program.

If SYSPROD indicates that a product is licensed, it means that the final license expiration date has not passed. Use the SETINIT procedure to determine the final expiration date for the product.

It is possible for a SAS software product to exist on your system even though the product is no longer licensed. However, SAS cannot access this product.

You can enter the product name in uppercase, in lowercase, or in mixed case. You can prefix the product with 'SAS/'. You can prefix SAS/ACCESS product names with 'ACC-'. Use the SETINIT procedure to obtain a list of products available on your system.

Examples

- `x=sysprod('graph');`The value returned is 1 if SAS/GRAPH software is currently licensed. The value returned is 0 if SAS/GRAPH software is not currently licensed.
- `x=sysprod('abc');`The value returned is -1 because ABC is not a valid product name.

- □ `x=sysprod('base');`The value returned is always 1 because the Base product must be licensed for the DATA step to run successfully.

SYSRC

Returns a system error number

Category: SAS File I/O

Category: External Files

Syntax

SYSRC()

Details

SYSRC returns the error number for the last system error encountered by a call to one of the data set functions or external file functions.

Example

This example determines the error message if FILEREF does not exist:

```
%if %sysfunc(fileref(myfile)) ne 0 %then
   %put %sysfunc(sysrc()) - %sysfunc(sysmsg());
```

See Also

Functions:
"FILEREF" on page 360
"SYSMSG" on page 558

SYSTEM

Issues an operating environment command during a SAS session

Category: Special

Syntax

SYSTEM(*command*)

Arguments

command

Operating Environment Information: See the SAS documentation for your operating environment for information on what you can specify. △

Comparisons

The SYSTEM function is similar to the X statement, the X command, and the CALL SYSTEM routine. In most cases, the X statement, X command, or %SYSEXEC macro statement are preferable because they require less overhead. However, the SYSTEM function can be executed conditionally, and accepts expressions as arguments. The X statement is a global statement and executes as a DATA step is being compiled, regardless of whether SAS encounters a conditional statement.

Example

Execute the host command TIMEDATA if the macro variable SYSDAY is **`Friday`**.

```
data _null_;
   if "&sysday"="Friday" then do;
      rc=system("timedata");
   end;
   else rc=system("errorck");
run;
```

See Also

CALL Routine:
"CALL SYSTEM" on page 271

Statement:
"X" on page 1043

TAN

Returns the tangent

Category: Trigonometric

Syntax

TAN(*argument*)

Arguments

argument
is numeric and is specified in radians.

Restriction: cannot be an odd multiple of π /2

Examples

SAS Statements	Results
`x=tan(0.5);`	**`0.5463024898`**
`x=tan(0);`	**`0`**
`x=tan(3.14159/3);`	**`1.7320472695`**

TANH

Returns the hyperbolic tangent

Category: Hyperbolic

Syntax

TANH(*argument*)

Arguments

argument
is numeric.

Details

The TANH function returns the hyperbolic tangent of the argument, which is given by

$$\frac{\left(e^{argument} - e^{-argument}\right)}{\left(e^{argument} + e^{-argument}\right)}$$

Examples

SAS Statements	Results
`x=tanh(0);`	**`0`**
`x=tanh(0.5);`	**`0.4621171573`**
`x=tanh(-0.5);`	**`-0.462117157`**

TIME

Returns the current time of day

Category: Date and Time

Syntax

TIME()

Example

SAS assigns CURRENT a SAS time value corresponding to 4:32:00 if the following statements are executed exactly at 2:32 PM:

```
current=time();
put current=time.;
```

TIMEPART

Extracts a time value from a SAS datetime value

Category: Date and Time

Syntax

TIMEPART(*datetime*)

Arguments

datetime
specifies a SAS expression that represents a SAS datetime value.

Example

SAS assigns TIME a SAS value that corresponds to 10:40:17 if the following statements are executed exactly at 10:40:17 AM on any date:

```
datim=datetime();
time=timepart(datim);
```

TINV

Returns a quantile from the *t* distribution

Category: Quantile

Syntax

TINV(*p*,*df*<,*nc*>)

Arguments

p

is a numeric probability.

Range: $0 \leq p < 1$

df

is a numeric degrees of freedom parameter.

Range: $df > 0$

nc

is an optional numeric noncentrality parameter.

Details

The TINV function returns the p^{th} quantile from the Student's *t* distribution with degrees of freedom *df* and a noncentrality parameter *nc*. The probability that an observation from a *t* distribution is less than or equal to the returned quantile is *p*.

TINV accepts a noninteger degree of freedom parameter *df*. If the optional parameter *nc* is not specified or is 0, the quantile from the central *t* distribution is returned.

CAUTION:

For large values of *nc*, the algorithm can fail; in that case, a missing value is returned. △

Note: TINV is the inverse of the PROBT function. △

Examples

SAS Statements	Results
`x=tinv(.95,2);`	`2.9199855804`
`x=tinv(.95,2.5,3);`	`1.033833625`

TNONCT

Returns the value of the noncentrality parameter from the student's *t* distribution

Category Mathematical

Syntax

TNONCT(*x*,*df*,*prob*)

Arguments

x

is a numeric random variable.

df
is a numeric degrees-of-freedom parameter, with *df* > 0.

prob
is a probability, with 0 < *prob* < .

Details

The TNONCT function returns the nonnegative noncentrality parameter from a noncentral *t* distribution whose parameters are *x*, *df*, and *nc*. A Newton-type algorithm is used to find a root *nc* of the equation

$$P_t\left(x|df, nc\right) - prob = 0$$

where

$$P_t\left(x|df, nc\right) = \frac{1}{\Gamma\left(\frac{df}{2}\right)} \int_0^\infty v^{\frac{df}{2}-1} e^{-v} \int_{-\infty}^{x\sqrt{\frac{2v}{df}}} e^{-\frac{(u-nc)^2}{2}} du dv$$

If the algorithm fails to converge to a fixed point, a missing value is returned.

Example

```
data work;
   x=2;
   df=4;
   do nc=1 to 3 by .5;
      prob=probt(x,df,nc);
      ncc=tnonct(x,df,prob);
      output;
   end;
run;
proc print;
run;
```

Output 4.16 Computations of the Noncentrality Parameter from the *t* Distribution

```
         OBS     x     df      nc       prob       ncc

          1      2      4     1.0     0.76457     1.0
          2      2      4     1.5     0.61893     1.5
          3      2      4     2.0     0.45567     2.0
          4      2      4     2.5     0.30115     2.5
          5      2      4     3.0     0.17702     3.0
```

TODAY

Returns the current date as a SAS date value

Category: Date and Time

Syntax

TODAY()

TODAY is identical to the DATE function. See “DATE” on page 312.

TRANSLATE

Replaces specific characters in a character expression

Category: Character

Syntax

TRANSLATE(*source,to-1,from-1*<,...*to-n,from-n*>)

Arguments

source
specifies the SAS expression that contains the original character value.

to
specifies the characters that you want TRANSLATE to use as substitutes.

from
specifies the characters that you want TRANSLATE to replace.

Interaction: Values of *to* and *from* correspond on a character-by-character basis; TRANSLATE changes character one of *from* to character one of *to*, and so on. If *to* has fewer characters than *from*, TRANSLATE changes the extra *from* characters to blanks. If *to* has more characters than *from*, TRANSLATE ignores the extra *to* characters.

Operating Environment Information: You must have pairs of *to* and *from* arguments on some operating environments. On other operating environments, a segment of the collating sequence replaces null *from* arguments. See the SAS documentation for your operating environment for more information. △

Details

The maximum number of pairs of *to* and *from* arguments that TRANSLATE accepts depends on the operating environment you use to run SAS. There is no functional difference between using several pairs of short arguments, or fewer pairs of longer arguments.

Comparisons

The TRANWRD function differs from TRANSLATE in that it scans for words (or patterns of characters) and replaces those words with a second word (or pattern of characters).

Examples

SAS Statements	Results

`x=translate('XYZW','AB','VW');`
`put x;`
	`XYZB`

See Also

Function:
"TRANWRD" on page 568

TRANWRD

Replaces or removes all occurrences of a word in a character string

Category: Character

Syntax

TRANWRD(*source,target,replacement*)

Arguments

source
specifies the source string that you want to translate.

target
specifies the string searched for in *source*.

replacement
specifies the string that replaces *target*.

Details

The TRANWRD function replaces or removes all occurrences of a given word (or a pattern of characters) within a character string. The TRANWRD function does not remove trailing blanks in the *target* string and the *replacement* string.

The value that the TRANWRD function returns has a default length of 200. You can use the LENGTH statement, before calling TRANWRD, to change the length of the value.

Comparisons

The TRANSLATE function converts every occurrence of a user-supplied character to another character. TRANSLATE can scan for more than one character in a single call. In doing this, however, TRANSLATE searches for every occurrence of any of the individual characters within a string. That is, if any letter (or character) in the target string is found in the source string, it is replaced with the corresponding letter (or character) in the replacement string.

The TRANWRD function differs from TRANSLATE in that it scans for words (or patterns of characters) and replaces those words with a second word (or pattern of characters).

Examples

Example 1: Replacing All Ocurrences of a Word These statements and these values produce these results:

```
name=tranwrd(name, "Mrs.", "Ms.");
name=tranwrd(name, "Miss", "Ms.");
put name;
```

Values	Results
Mrs. Joan Smith	**Ms. Joan Smith**
Miss Alice Cooper	**Ms. Alice Cooper**

Example 2: Removing Blanks From the Search String In this example, the TRANWRD function does not replace the source string because the target string contains blanks.

```
data list;
   input salelist $;
   length target $10 replacement $3;
   target='FISH';
   replacement='NIP';
   salelist=tranwrd(salelist,target,replacement);
   put salelist;
   datalines;
CATFISH
;
```

The LENGTH statement left-aligns TARGET and pads it with blanks to the length of 10. This causes the TRANWRD function to search for the character string **'FISH   '** in SALELIST. Because the search fails, this line is written to the SAS log:

```
CATFISH
```

You can use the TRIM function to exclude trailing blanks from a target or replacement variable. Use the TRIM function with TARGET:

```
salelist=tranwrd(salelist,trim(target),replacement);
put salelist;
```

Now, this line is written to the SAS log:

```
CATNIP
```

See Also

Function:
 "TRANSLATE" on page 567

TRIGAMMA

Returns the value of the TRIGAMMA function

Category: Mathematical

Syntax

TRIGAMMA(*argument*)

Arguments

argument
is numeric.

Restriction: Nonpositive integers are invalid.

Details

The TRIGAMMA function returns the derivative of the DIGAMMA function. For *argument* > 0, the TRIGAMMA function is the second derivative of the LGAMMA function.

Examples

SAS Statements	Results
`x=trigamma(3);`	**`0.3949340668`**

TRIM

Removes trailing blanks from character expressions and returns one blank if the expression is missing

Category: Character

Syntax

TRIM(*argument*)

Arguments

argument
specifies any SAS character expression.

Details

TRIM copies a character argument, removes all trailing blanks, and returns the trimmed argument as a result. If the argument is blank, TRIM returns one blank. TRIM is useful for concatenating because concatenation does not remove trailing blanks.

Assigning the results of TRIM to a variable does not affect the length of the receiving variable. If the trimmed value is shorter than the length of the receiving variable, SAS pads the value with new blanks as it assigns it to the variable.

Comparisons

The TRIM and TRIMN functions are similar. TRIM returns one blank for a blank string. TRIMN returns a null string (zero blanks) for a blank string.

Examples

Example 1: Removing Trailing Blanks These statements and this data line produce these results:

```
data test;
   input part1 $ 1-10 part2 $ 11-20;
   hasblank=part1||part2;
   noblank=trim(part1)||part2;
   put hasblank;
   put noblank;
   datalines;
```

Data Line	Results
`apple     sauce`	`----+----1----+----2`
	`apple     sauce`
	`applesauce`

Example 2: Concatenating a Blank Character Expression

SAS Statements	Results
`x="A"||trim(" ")||"B"; put x;`	`A B`
`x="    "; y=">"||trim(x)||"<"; put y;`	`> <`

See Also

Functions:
- "COMPRESS" on page 296
- "LEFT" on page 435
- "RIGHT" on page 524

TRIMN

Removes trailing blanks from character expressions and returns a null string (zero blanks) if the expression is missing

Category: Character

Syntax

TRIMN(*argument*)

Arguments

argument
specifies any SAS character expression.

Details

TRIMN copies a character argument, removes all trailing blanks, and returns the trimmed argument as a result. If the argument is blank, TRIMN returns a null string. TRIMN is useful for concatenating because concatenation does not remove trailing blanks.

Assigning the results of TRIMN to a variable does not affect the length of the receiving variable. If the trimmed value is shorter than the length of the receiving variable, SAS pads the value with new blanks as it assigns it to the variable.

Comparisons

The TRIMN and TRIM functions are similar. TRIMN returns a null string (zero blanks) for a blank string. TRIM returns one blank for a blank string.

Examples

SAS Statements	Results

`x="A"||trimn("")||"B";`
`put x;`
	AB

`x="   ";`
`z=">"||trimn(x)||"<";`
`put z;`
	><

See Also

Functions:
- "COMPRESS" on page 296
- "LEFT" on page 435
- "RIGHT" on page 524
- "TRIM" on page 570

TRUNC

Truncates a numeric value to a specified length

Category: Truncation

Syntax

TRUNC(*number*,*length*)

Arguments

number
is numeric.

length
is numeric and integer.

Details

The TRUNC function truncates a full-length *number* (stored as a double) to a smaller number of bytes, as specified in *length* and pads the truncated bytes with 0s. The truncation and subsequent expansion duplicate the effect of storing numbers in less than full length and then reading them.

Comparisons

The ROUND function returns a value rounded to the nearest round-off unit. If a round-off unit is not provided, a default value of 1 is used, and the argument is rounded to the nearest integer.

Examples

```
data test;
  length x 3;
  x=1/5;
run;
data test2;
  set test;
  if x ne 1/5 then
     put 'x ne 1/5';
  if x eq trunc(1/5,3) then
     put 'x eq trunc(1/5,3)';
run;
```

The variable X is stored with a length of 3 and, therefore, each of the above comparisons is true.

UNIFORM

Random variate from a uniform distribution

Category: Random Number

See: "RANUNI" on page 521

Syntax

UNIFORM(*seed*)

seed
is numeric.

Range: *seed* < 2^{31}–1

Note: If *seed* ≤ 0, the time of day is used to initialize the seed stream.

UPCASE

Converts all letters in an argument to uppercase

Category: Character

Syntax

UPCASE(*argument*)

Arguments

argument
specifies any SAS character expression.

Details

The UPCASE function copies a character argument, converts all lowercase letters to uppercase letters, and returns the altered value as a result.

Examples

SAS Statements	Results

`name=upcase('John B. Smith');`
`put name;`
	`JOHN B. SMITH`

URLDECODE

Returns a string that was decoded using the URL escape syntax

Category: Web Tools

Syntax

URLDECODE(*argument*)

Arguments

argument
specifies any character expression that contains a URL escape sequence, which is a three character string of the form *%nn*.

Details

The URL escape syntax is used to hide characters that may otherwise be significant when used in a URL. URLDECODE also converts plus (+) characters to spaces.

Operating Environment Information: In operating environments that use EBCDIC, SAS performs an extra translation step after it recognizes an escape sequence. The specified character is assumed to be an ASCII encoding. SAS uses the transport-to-local translation table to convert this character to an EBCDIC character in operating environments that use EBCDIC. For more information see <MAKR LINK> the TRANTAB= system option. △

Examples

SAS Statements	Results

`x1=urldecode ('abc+def');`
`put x1;`
	`abc def`

`x2=urldecode ('why%3F');`
`put x2;`
	`why?`

`x3=urldecode ('%41%42%43%23%31');`
`put x3;`
	`ABC#1`

See Also

Function:
  "URLENCODE" on page 576

URLENCODE

Returns a string that was encoded using the URL escape syntax

Category: Web Tools

Syntax

URLENCODE(*argument*)

Arguments

argument
  specifies any character expression.

Details

The URLENCODE function encodes characters that may otherwise be significant when used in a URL. This function encodes all characters except for the following:

- all alphanumeric characters
- dollar sign ($)
- dash (-)
- underscore (_)
- at sign (@)
- period (.)
- exclamation point (!)
- asterisk (*)
- left parenthesis (( )and right parenthesis ( ))
- comma (,).

Note: The encoded string may be longer than the original string. Ensure that you consider the additional length when you use this function. △

Examples

SAS Statements	Results

`x1=urlencode ('abc def');`
`put x1;`
	`abc%20def`

`x2=urlencode ('why?');`
`put x2;`
	`why%3F`

`x3=urlencode ('ABC#1');`
`put x3;`
	`ABC%231`

See Also

Function:
"URLDECODE" on page 575

USS

Returns the uncorrected sum of squares

Category: Descriptive Statistics

Syntax

USS(*argument,argument, . . .*)

Arguments

argument
is numeric. At least two arguments are required. The argument list may consist of a variable list, which is preceded by OF.

Examples

SAS Statements	Results
`x1=uss(4,2,3.5,6);`	68.25
`x2=uss(4,2,3.5,6,.);`	68.25
`x3=uss(of x1-x2);`	9316.125

VAR

Returns the variance

Category: Descriptive Statistics

Syntax

VAR(*argument*,*argument*, . . .)

Arguments

argument
is numeric. At least two arguments are required. The argument list may consist of a variable list, which is preceded by OF.

Examples

SAS Statements	Results
`x1=var(4,2,3.5,6);`	`2.7291666667`
`x2=var(4,6,.);`	`2`
`x3=var(of x1-x2);`	`0.2658420139`

VARFMT

Returns the format assigned to a SAS data set variable

Category: SAS File I/O

Syntax

VARFMT(*data-set-id*,*var-num*)

Arguments

data-set-id
specifies the data set identifier that the OPEN function returns.

var-num
specifies the number of the variable's position in the SAS data set.

Tip: This number is next to the variable in the list that is produced by the CONTENTS procedure.

Tip: The VARNUM function returns this number.

Details

If no format has been assigned to the variable, a blank string is returned.

Examples

- □ This example obtains the format of the variable NAME in the SAS data set MYDATA.

```
%let dsid=%sysfunc(open(mydata,i));
%if &dsid %then
   %do;
      %let fmt=%sysfunc(varfmt(&dsid,
                        %sysfunc(varnum
                        (&dsid,NAME))));
      %let rc=%sysfunc(close(&dsid));
   %end;
```

- □ This example creates a data set that contains the name and formatted content of each numeric variable in the SAS data set MYDATA.

```
data vars;
   length name $ 8 content $ 12;
   drop dsid i num rc fmt;
   dsid=open("mydata","i");
   num=attrn(dsid,"nvars");
   do while (fetch(dsid)=0);
      do i=1 to num;
         name=varname(dsid,i);
         if (vartype(dsid,i)='N') then do;
            fmt=varfmt(dsid,i);
            if fmt='' then fmt="BEST12.";
            content=putc(putn(getvarn
                        (dsid,i),fmt),"$char12.");
            output;
            end;
      end;
   end;
   rc=close(dsid);
run;
```

See Also

Functions:

“VARINFMT” on page 580

“VARNUM” on page 584

VARINFMT

Returns the informat assigned to a SAS data set variable

Category: SAS File I/O

Syntax

VARINFMT(*data-set-id*,*var-num*)

Arguments

data-set-id
specifies the data set identifier that the OPEN function returns.

var-num
specifies the number of the variable’s position in the SAS data set.

Tip: This number is next to the variable in the list produced by the CONTENTS procedure.

Tip: The VARNUM function returns this number.

Details

If no informat has been assigned to the variable, a blank string is returned.

Examples

□ This example obtains the informat of the variable NAME in the SAS data set MYDATA.

```
%let dsid=%sysfunc(open(mydata,i));
%if &dsid %then
   %do;
      %let fmt=%sysfunc(varinfmt(&dsid,
                        %sysfunc(varnum
                           (&dsid,NAME))));
      %let rc=%sysfunc(close(&dsid));
   %end;
```

□ This example creates a data set that contains the name and informat of the variables in MYDATA.

```
data vars;
   length name $ 8 informat $ 10 ;
   drop dsid i num rc;
   dsid=open("mydata","i");
   num=attrn(dsid,"nvars");
   do i=1 to num;
      name=varname(dsid,i);
      informat=varinfmt(dsid,i);
      output;
   end;
   rc=close(dsid);
run;
```

See Also

Functions:
- "OPEN" on page 460
- "VARFMT" on page 578
- "VARNUM" on page 584

VARLABEL

Returns the label assigned to a SAS data set variable

Category: SAS File I/O

Syntax

VARLABEL(*data-set-id*,*var-num*)

Arguments

data-set-id
specifies the data set identifier that the OPEN function returns.

var-num
specifies the number of the variable's position in the SAS data set.

Tip: This number is next to the variable in the list that is produced by the CONTENTS procedure.

Tip: The VARNUM function returns this number.

Details

If no label has been assigned to the variable, a blank string is returned.

Comparisons

VLABEL returns the label that is associated with the given variable.

Examples

This example obtains the label of the variable NAME in the SAS data set MYDATA.

Example Code 4.2 Obtaining the Label of the Variable NAME

```
%let dsid=%sysfunc(open(mydata,i));
%if &dsid %then
   %do;
      %let fmt=%sysfunc(varlabel(&dsid,
                       %sysfunc(varnum
                               (&dsid,NAME))));
      %let rc=%sysfunc(close(&dsid));
   %end;
```

See Also

Functions:
"OPEN" on page 460
"VARNUM" on page 584

VARLEN

Returns the length of a SAS data set variable

Category: SAS File I/O

Syntax

VARLEN(*data-set-id*,*var-num*)

Arguments

data-set-id
specifies the data set identifier that the OPEN function returns.

var-num
specifies the number of the variable's position in the SAS data set.

Tip: This number is next to the variable in the list that is produced by the CONTENTS procedure.

Tip: The VARNUM function returns this number.

Comparisons

VLENGTH returns the compile-time (allocated) size of the given variable.

Examples

- This example obtains the length of the variable ADDRESS in the SAS data set MYDATA.

```
%let dsid=%sysfunc(open(mydata,i));
%if &dsid %then
   %do;
      %let len=%sysfunc(varlen(&dsid,
                        %sysfunc(varnum
                        (&dsid,ADDRESS))));
      %let rc=%sysfunc(close(&dsid));
   %end;
```

- This example creates a data set that contains the name, type, and length of the variables in MYDATA.

```
data vars;
   length name $ 8 type $ 1;
   drop dsid i num rc;
   dsid=open("mydata","i");
   num=attrn(dsid,"nvars");
   do i=1 to num;
      name=varname(dsid,i);
      type=vartype(dsid,i);
      length=varlen(dsid,i);
      output;
   end;
   rc=close(dsid);
run;
```

See Also

Functions:

“OPEN” on page 460

“VARNUM” on page 584

VARNAME

Returns the name of a SAS data set variable

Category: SAS File I/O

Syntax

VARNAME(*data-set-id*,*var-num*)

Arguments

data-set-id
specifies the data set identifier that the OPEN function returns.

var-num
specifies the number of the variable's position in the SAS data set.

Tip: This number is next to the variable in the list that is produced by the CONTENTS procedure.

Tip: The VARNUM function returns this number.

Examples

This example copies the names of the first five variables in the SAS data set CITY (or all of the variables if there are fewer than five) into a macro variable.

```
%let dsid=%sysfunc(open(city,i));
%let varlist=;
%do i=1 %to
      %sysfunc(min(5,%sysfunc(attrn
                            (&dsid,nvars))));
   %let varlist=&varlist %sysfunc(varname
                                (&dsid,&i));
%end;
%put varlist=&varlist;
%mend;
```

See Also

Functions:
“OPEN” on page 460
“VARNUM” on page 584

VARNUM

Returns the number of a variable's position in a SAS data set

Category: SAS File I/O

Syntax

VARNUM(*data-set-id*,*var-name*)

Arguments

data-set-id
specifies the data set identifier that the OPEN function returns.

var-name
specifies the variable's name.

Details

VARNUM returns the number of a variable's position in a SAS data set, or 0 if the variable is not in the SAS data set. This is the same variable number that is next to the variable in the output from PROC CONTENTS.

Examples

- This example obtains the number of a variable's position in the SAS data set CITY, given the name of the variable.

```
%let dsid=%sysfunc(open(city,i));
%let citynum=%sysfunc(varnum(&dsid,CITYNAME));
%let rc=%sysfunc(fetch(&dsid));
%let cityname=%sysfunc(getvarc
                      (&dsid,&citynum));
```

- This example creates a data set that contains the name, type, format, informat, label, length, and position of the variables in SASUSER.HOUSES.

```
data vars;
   length name $ 8 type $ 1
          format informat $ 10 label $ 40;
   drop dsid i num rc;
   dsid=open("sasuser.houses","i");
   num=attrn(dsid,"nvars");
   do i=1 to num;
      name=varname(dsid,i);
      type=vartype(dsid,i);
      format=varfmt(dsid,i);
      informat=varinfmt(dsid,i);
      label=varlabel(dsid,i);
      length=varlen(dsid,i);
      position=varnum(dsid,name);
      output;
   end;
   rc=close(dsid);
run;
```

See Also

Functions:
  "OPEN" on page 460
  "VARNAME" on page 583

VARRAY

Returns a value that indicates whether the specified name is an array

Category: Variable Information

Syntax

VARRAY (*name*)

Arguments

name
specifies a name expressed as a scalar or as an array reference.

Restriction: You cannot use an expression as an argument.

Details

VARRAY returns 1 if the given name is an array; it returns 0 if the given name is not an array.

Comparisons

- VARRAY returns a value that indicates whether the specified name is an array. VARRAYX returns a value that indicates whether the value of the specified expression is an array.
- VARRAY does not accept an expression as an argument. VARRAYX accepts expressions, but the value of the specified variable cannot denote an array reference.
- Related functions return the value of other variable attributes, such as the variable name, informat, format, among others. For a list, see the “Variable Information” functions in “Functions and CALL Routines by Category” on page 213 .

Examples

SAS Statements	Results

`array x(3) x1-x3;`
`y=varray(x);`
`Z=varray(x1);`
`put y=;`
`put Z=;`
	`y=1`
	`z=0`

See Also

Functions:

“Variable Information” functions in “Functions and CALL Routines by Category” on page 213

VARRAYX

Returns a value that indicates whether the value of the specified argument is an array

Category: Variable Information

Syntax

VARRAYX (*expression*)

Arguments

expression
specifies any SAS character expression.

Restriction: The value of the specified expression cannot denote an array reference.

Details

VARRAYX returns 1 if the value of the given argument is the name of an array; it returns 0 if the value of the given argument is not the name of an array.

Comparisons

- VARRAY returns a value that indicates whether the specified name is the name of an array. VARRAYX returns a value that indicates whether the value of the specified expression is the name of an array.
- VARRAY does not accept an expression as an argument. VARRAYX accepts expressions, but the value of the specified variable cannot denote an array reference.
- Related functions return the value of other variable attributes, such as the variable name, informat, format, among others. For a list, see the "Variable Information" functions in "Functions and CALL Routines by Category" on page 213 .

Examples

SAS Statements	Results

`array x(3) x1-x3;`
`array vx(4) $6 vx1 vx2 vx3 vx4`
`   ('x' 'x1' 'x2' 'x3');`
`y=varrayx(vx(1));`
`z=varrayx(vx(2));`
`put y=;`
`put z=;`
	`y=1`
	`z=0`

See Also

Functions:

"Variable Information" functions in "Functions and CALL Routines by Category" on page 213

VARTYPE

Returns the data type of a SAS data set variable

Category: SAS File I/O

Syntax

VARTYPE(*data-set-id*,*var-num*)

Arguments

data-set-id
specifies the data set identifier that the OPEN function returns.

var-num
specifies the number of the variable's position in the SAS data set.

Tip: This number is next to the variable in the list that is produced by the CONTENTS procedure.

Tip: The VARNUM function returns this number.

Details

VARTYPE returns C for a character variable or N for a numeric variable.

Examples

- This example places the names of all the numeric variables of the SAS data set MYDATA into a macro variable.

```
%let dsid=%sysfunc(open(mydata,i));
%let varlist=;
%do i=1 %to %sysfunc(attrn(&dsid,nvars));
  %if (%sysfunc(vartype(&dsid,&i)) = N) %then
     %let varlist=&varlist %sysfunc(varname
                                (&dsid,&i));
%end;
%let rc=%sysfunc(close(&dsid));
```

- This example creates a data set that contains the name and formatted contents of each character variable in the SAS data set MYDATA.

```
data vars;
   length name $ 8 content $ 20;
   drop dsid i num fmt rc;
   dsid=open("mydata","i");
   num=attrn(dsid,"nvars");
   do while (fetch(dsid)=0);
      do i=1 to num;
         name=varname(dsid,i);
         fmt=varfmt(dsid,i);
         if (vartype(dsid,i)='C') then do;
            content=getvarc(dsid,i);
            if (fmt ne '' ) then
             content=left(putc(content,fmt));
            output;
            end;
      end;
   end;
   rc=close(dsid);
run;
```

See Also

Function:
    "VARNUM" on page 584

VERIFY

Returns the position of the first character that is unique to an expression

Category: Character

Syntax

VERIFY(*source,excerpt-1*<,...*excerpt-n*>)

Arguments

source
specifies any SAS character expression.

excerpt
specifies any SAS character expression. If you specify more than one excerpt, separate them with a comma.

Details

The VERIFY function returns the position of the first character in *source* that is not present in any *excerpt*. If VERIFY finds every character in *source* in at least one *excerpt*, it returns a 0.

Examples

SAS Statements	Results

`data scores;`
`   input Grade : $1. @@;`
`   check='abcdf';`
`   if verify(grade,check)>0 then`
`      put @1 'INVALID ' grade=;`
`   datalines;`
`a b c b c d f a a q a b d d b`
`;`
	`INVALID Grade=q`

VFORMAT

Returns the format that is associated with the specified variable

Category: Variable Information

Syntax

VFORMAT (*var*)

Arguments

var
specifies a variable, expressed as a scalar or as an array reference.

Restriction: You cannot use an expression as an argument.

Details

VFORMAT returns the complete format name, which includes the width and the period (for example, $CHAR20.).

Comparisons

□ VFORMAT returns the format that is associated with the specified variable. VFORMATX, however, evaluates the argument to determine the variable name. The function then returns the format that is associated with that variable name.

□ VFORMAT does not accept an expression as an argument. VFORMATX accepts expressions, but the value of the specified expression cannot denote an array reference.

□ Related functions return the value of other variable attributes, such as the variable name, type, length, among others. For a list, see the “Variable Information” functions in “Functions and CALL Routines by Category” on page 213 .

Examples

SAS Statements	Results

`array x(3) x1-x3;`
`format x1 best6.;`
`y=vformat(x(1));`
`put y=;`
	`y=BEST6.`

See Also

Functions:

"Variable Information" functions in "Functions and CALL Routines by Category" on page 213

VFORMATD

Returns the format decimal value that is associated with the specified variable

Category: Variable Information

Syntax

VFORMATD (*var*)

Arguments

var

specifies a variable, expressed as a scalar or as an array reference.

Restriction: You cannot use an expression as an argument.

Comparisons

- VFORMATD returns the format decimal value that is associated with the specified variable. VFORMATDX, however, evaluates the argument to determine the variable name. The function then returns the format decimal value that is associated with that variable name.
- VFORMATD does not accept an expression as an argument. VFORMATDX accepts expressions, but the value of the specified expression cannot denote an array reference.
- Related functions return the value of other variable attributes, such as the variable name, type, and length, among others. For a list, see the "Variable Information" functions in "Functions and CALL Routines by Category" on page 213 .

Examples

SAS Statements	Results

`array x(3) x1-x3;`
`format x1 comma8.2;`
`y=vformatd(x(1));`
`put y=;`
	`y=2`

See Also

Functions:

“Variable Information” functions in “Functions and CALL Routines by Category” on page 213

VFORMATDX

Returns the format decimal value that is associated with the value of the specified argument

Category: Variable Information

Syntax

VFORMATDX (*expression*)

Arguments

expression
specifies any SAS character expression that evaluates to a variable name.

Restriction: The value of the specified expression cannot denote an array reference.

Comparisons

- VFORMATD returns the format decimal value that is associated with the specified variable. VFORMATDX, however, evaluates the argument to determine the variable name. The function then returns the format decimal value that is associated with that variable name.
- VFORMATD does not accept an expression as an argument. VFORMATDX accepts expressions, but the value of the specified expression cannot denote an array reference.
- Related functions return the value of other variable attributes, such as the variable name, length, type, among others. For a list, see the “Variable Information” functions in “Functions and CALL Routines by Category” on page 213 .

Examples

SAS Statements	Results
`array x(3) x1-x3;`	
`format x1 comma8.2;`	
`array vx(3) $6 vx1 vx2 vx3`	
`   ('x1' 'x2' 'x3');`	
`y=vformatdx(vx(1));`	
`z=vformatdx('x'‖'1');`	
`put y=;`	`y=2`
`put z=;`	`z=2`

See Also

Functions:

"Variable Information" functions in "Functions and CALL Routines by Category" on page 213

VFORMATN

Returns the format name that is associated with the specified variable

Category: Variable Information

Syntax

VFORMATN (*var*)

Arguments

var

specifies a variable, expressed as a scalar or as an array reference.

Restriction: You cannot use an expression as an argument.

Details

VFORMATN returns only the format name, which does not include the width or the period (for example, $CHAR).

Comparisons

- VFORMATN returns the format name that is associated with the specified variable. VFORMATNX, however, evaluates the argument to determine the variable name. The function then returns the format name that is associated with that variable name.
- VFORMATN does not accept an expression as an argument. VFORMATNX accepts expressions, but the value of the specified expression cannot denote an array reference.

- Related functions return the value of other variable attributes, such as the variable name, type, and length, among others. For a list, see the "Variable Information" functions in "Functions and CALL Routines by Category" on page 213 .

Examples

SAS Statements	Results

`array x(3) x1-x3;`
`format x1 best6.;`
`y=vformatn(x(1));`
`put y=;`
	`y=BEST`

See Also

Functions:

"Variable Information" functions in "Functions and CALL Routines by Category" on page 213

VFORMATNX

Returns the format name that is associated with the value of the specified argument

Category: Variable Information

Syntax

VFORMATNX (*expression*)

Arguments

expression

specifies any SAS character expression that evaluates to a variable name.

Restriction: The value of the specified expression cannot denote an array reference.

Details

VFORMATNX returns only the format name, which does not include the length or the period (for example, $CHAR).

Comparisons

- VFORMATN returns the format name that is associated with the specified variable. VFORMATNX, however, evaluates the argument to determine the variable name. The function then returns the format name that is associated with that variable name.

- □ VFORMATN does not accept an expression as an argument. VFORMATNX accepts expressions, but the value of the specified expression cannot denote an array reference.
- □ Related functions return the value of other variable attributes, such as the variable name, length, and type, among others. For a list, see the "Variable Information" functions in "Functions and CALL Routines by Category" on page 213 .

Examples

SAS Statements	Results

`array x(3) x1-x3;`
`format x1 best6.;`
`array vx(3) $6 vx1 vx2 vx3`
`   ('x1' 'x2' 'x3');`
`y=vformatnx(vx(1));`
`put y=;`
	`y=BEST`

See Also

Functions:

"Variable Information" functions in "Functions and CALL Routines by Category" on page 213

VFORMATW

Returns the format width that is associated with the specified variable

Category: Variable Information

Syntax

VFORMATW (*var*)

Arguments

var

specifies a variable, expressed as a scalar or as an array reference.

Restriction: You cannot use an expression as an argument.

Comparisons

- □ VFORMATW returns the format width that is associated with the specified variable. VFORMATWX, however, evaluates the argument to determine the variable name. The function then returns the format width that is associated with that variable name.

- □ VFORMATW does not accept an expression as an argument. VFORMATWX accepts expressions, but the value of the specified expression cannot denote an array reference.
- □ Related functions return the value of other variable attributes, such as the variable name, type, and length, among others. For a list, see the "Variable Information" functions in “Functions and CALL Routines by Category” on page 213 .

Examples

SAS Statements	Results

`array x(3) x1-x3;`
`format x1 best6.;`
`y=vformatw(x(1));`
`put y=;`
	`y=6`

See Also

Functions:

"Variable Information" functions in “Functions and CALL Routines by Category” on page 213

VFORMATWX

Returns the format width that is associated with the value of the specified argument

Category: Variable Information

Syntax

VFORMATWX (*expression*)

Arguments

expression
specifies any SAS character expression that evaluates to a variable name.

Restriction: The value of the specified expression cannot denote an array reference.

Comparisons

- □ VFORMATW returns the format width that is associated with the specified variable. VFORMATWX, however, evaluates the argument to determine the variable name. The function then returns the format width that is associated with that variable name.

- □ VFORMATW does not accept an expression as an argument. VFORMATWX accepts expressions, but the value of the specified expression cannot denote an array reference.
- □ Related functions return the value of other variable attributes, such as the variable name, length, and type, among others. For a list, see the "Variable Information" functions in “Functions and CALL Routines by Category” on page 213 .

Examples

SAS Statements	Results
`array x(3) x1-x3;`	
`format x1 best6.;`	
`array vx(3) $6 vx1 vx2 vx3`	
`   ('x1' 'x2' 'x3');`	
`y=vformatwx(vx(1));`	
`put y=;`	`y=6`

See Also

Functions:

"Variable Information" functions in “Functions and CALL Routines by Category” on page 213

VFORMATX

Returns the format that is associated with the value of the specified argument

Category: Variable Information

Syntax

VFORMATX (*expression*)

Arguments

expression
specifies any SAS character expression that evaluates to a variable name.

Restriction: The value of the specified expression cannot denote an array reference.

Details

VFORMATX returns the complete format name which includes the width and the period (for example, $CHAR20.).

Comparisons

- VFORMAT returns the format that is associated with the specified variable. VFORMATX, however, evaluates the argument to determine the variable name. The function then returns the format that is associated with that variable name.
- VFORMAT does not accept an expression as an argument. VFORMATX accepts expressions, but the value of the specified expression cannot denote an array reference.
- Related functions return the value of other variable attributes, such as the variable name, length, and type, among others. For a list, see the "Variable Information" functions in “Functions and CALL Routines by Category” on page 213 .

Examples

SAS Statements	Results
`array x(3) x1-x3;`	
`format x1 best6.;`	
`format x2 20.10;`	
`array vx(3) $6 vx1 vx2 vx3`	
`   ('x1' 'x2' 'x3');`	
`y=vformatx(vx(1));`	
`z=vformatx(vx(2));`	
`put y=;`	`y=BEST6.`
`put z=;`	`z=F20.10`

See Also

Functions:

"Variable Information" functions in “Functions and CALL Routines by Category” on page 213

VINARRAY

Returns a value that indicates whether the specified variable is a member of an array

Category: Variable Information

Syntax

VINARRAY (*var*)

Arguments

var

specifies a variable, expressed as a scalar or as an array reference.

Restriction: You cannot use an expression as an argument.

Details

VINARRAY returns 1 if the given variable is a member of an array; it returns 0 if the given variable is not a member of an array.

Comparisons

- VINARRAY returns a value that indicates whether the specified variable is a member of an array. VINARRAYX, however, evaluates the argument to determine the variable name. The function then returns a value that indicates whether the variable name is a member of an array.
- VINARRAY does not accept an expression as an argument. VINARRAYX accepts expressions, but the value of the specified expression cannot denote an array reference.
- Related functions return the value of other variable attributes, such as the variable name, informat, and format, among others. For a list, see the "Variable Information" functions in “Functions and CALL Routines by Category” on page 213 .

Examples

SAS Statements	Results
`array x(3) x1-x3;`	
`y=vinarray(x);`	
`Z=vinarray(x1);`	
`put y=;`	`y=0`
`put Z=;`	`z=1`

See Also

Functions:
"Variable Information" functions in “Functions and CALL Routines by Category” on page 213

VINARRAYX

Returns a value that indicates whether the value of the specified argument is a member of an array

Category: Variable Information

Syntax

VINARRAYX (*expression*)

Arguments

expression
specifies any SAS character expression that evaluates to a variable name.

Restriction: The value of the specified expression cannot denote an array reference.

Details

VINARRAYX returns 1 if the value of the given argument is a member of an array; it returns 0 if the value of the given argument is not a member of an array.

Comparisons

- VINARRAY returns a value that indicates whether the specified variable is a member of an array. VINARRAYX, however, evaluates the argument to determine the variable name. The function then returns a value that indicates whether the variable name is a member of an array.
- VINARRAY does not accept an expression as an argument. VINARRAYX accepts expressions, but the value of the specified expression cannot denote an array reference.
- Related functions return the value of other variable attributes, such as the variable name, informat, and format, among others. For a list, see the "Variable Information" functions in “Functions and CALL Routines by Category” on page 213 .

Examples

SAS Statements	Results
`array x(3) x1-x3;`	
`array vx(4) $6 vx1 vx2 vx3 vx4`	
`   ('x' 'x1' 'x2' 'x3');`	
`y=vinarrayx(vx(1));`	
`z=vinarrayx(vx(2));`	
`put y=;`	`y=0`
`put z=;`	`z=1`

See Also

Functions:
"Variable Information" functions in “Functions and CALL Routines by Category” on page 213

VINFORMAT

Returns the informat that is associated with the specified variable

Category: Variable Information

Syntax

VINFORMAT (*var*)

Arguments

var

specifies a variable, expressed as a scalar or as an array reference.

Restriction: You cannot use an expression as an argument.

Details

VINFORMAT returns the complete informat name, which includes the width and the period (for example, $CHAR20.).

Comparisons

- VINFORMAT returns the informat that is associated with the specified variable. VINFORMATX, however, evaluates the argument to determine the variable name. The function then returns the informat that is associated with that variable name.
- VINFORMAT does not accept an expression as an argument. VINFORMATX accepts expressions, but the value of the specified expression cannot denote an array reference.
- Related functions return the value of other variable attributes, such as the variable name, type, and length, among others. For a list, see the "Variable Information" functions in “Functions and CALL Routines by Category” on page 213 .

Examples

SAS Statements	Results

`informat x $char6.;`
`input x;`
`y=vinformat(x);`
`put y=;`
	`y=$CHAR6.`

See Also

Functions:

"Variable Information" functions in “Functions and CALL Routines by Category” on page 213

VINFORMATD

Returns the informat decimal value that is associated with the specified variable

Category: Variable Information

Syntax

VINFORMATD (*var*)

Arguments

var
specifies a variable, expressed as a scalar or as an array reference.

Restriction: You cannot use an expression as an argument.

Comparisons

- VINFORMATD returns the informat decimal value that is associated with the specified variable. VINFORMATDX, however, evaluates the argument to determine the variable name. The function then returns the informat decimal value that is associated with that variable name.
- VINFORMATD does not accept an expression as an argument. VINFORMATDX accepts expressions, but the value of the specified expression cannot denote an array reference.
- Related functions return the value of other variable attributes, such as the variable name, type, and length, among others. For a list, see the "Variable Information" functions in “Functions and CALL Routines by Category” on page 213 .

Examples

SAS Statements	Results

`informat x comma8.2;`
`input x;`
`y=vinformatd(x);`
`put y=;`
	`y=2`

See Also

Functions:

"Variable Information" functions in “Functions and CALL Routines by Category” on page 213

VINFORMATDX

Returns the informat decimal value that is associated with the value of the specified argument

Category: Variable Information

Syntax

VINFORMATDX (*expression*)

Arguments

expression
specifies any SAS character expression that evaluates to a variable name.

Restriction: The value of the specified variable cannot denote an array reference.

Comparisons

- □ VINFORMATD returns the informat decimal value that is associated with the specified variable. VINFORMATDX, however, evaluates the argument to determine the variable name. The function then returns the informat decimal value that is associated with that variable name.
- □ VINFORMATD does not accept an expression as an argument. VINFORMATDX accepts expressions, but the value of the specified expression cannot denote an array reference.
- □ Related functions return the value of other variable attributes, such as the variable name, length, and type, among others. For a list, see the "Variable Information" functions in "Functions and CALL Routines by Category" on page 213 .

Examples

SAS Statements	Results

`informat x1 x2 x3 comma9.3;`
`input x1 x2 x3;`
`array vx(3) $6 vx1 vx2 vx3`
`   ('x1' 'x2' 'x3');`
`y=vinformatdx(vx(1));`
`put y=;`
	`y=3`

See Also

Functions:

"Variable Information" functions in "Functions and CALL Routines by Category" on page 213

VINFORMATN

Returns the informat name that is associated with the specified variable

Category: Variable Information

Syntax

VINFORMATN (*var*)

Arguments

var
specifies a variable, expressed as a scalar or as an array reference.

Restriction: You cannot use an expression as an argument.

Details

VINFORMATN returns only the informat name, which does not include the width or the period (for example, $CHAR).

Comparisons

- □ VINFORMATN returns the informat name that is associated with the specified variable. VINFORMATNX, however, evaluates the argument to determine the variable name. The function then returns the informat name that is associated with that variable name.
- □ VINFORMATN does not accept an expression as an argument. VINFORMATNX accepts expressions, but the value of the specified expression cannot denote an array reference.
- □ Related functions return the value of other variable attributes, such as the variable name, type, and length, among others. For a list, see the "Variable Information" functions in “Functions and CALL Routines by Category” on page 213 .

Examples

SAS Statements	Results
`informat x $char6.;`	
`input x;`	
`y=vinformatn(x);`	
`put y=;`	`y=$CHAR`

See Also

Functions:

"Variable Information" functions in “Functions and CALL Routines by Category” on page 213

VINFORMATNX

Returns the informat name that is associated with the value of the specified argument

Category: Variable Information

Syntax

VINFORMATNX (*expression*)

Arguments

expression

specifies any SAS character expression that evaluates to a variable name.

Restriction: The value of the specified expression cannot denote an array reference.

Details

VINFORMATNX returns only the informat name, which does not include the width or the period (for example, $CHAR).

Comparisons

- VINFORMATN returns the informat name that is associated with the specified variable. VINFORMATNX, however, evaluates the argument to determine the variable name. The function then returns the informat name that is associated with that variable name.
- VINFORMATN does not accept an expression as an argument. VINFORMATNX accepts expressions, but the value of the specified expression cannot denote an array reference.
- Related functions return the value of other variable attributes, such as the variable name, length, and type, among others. For a list, see the "Variable Information" functions in “Functions and CALL Routines by Category” on page 213 .

Examples

SAS Statements	Results

`informat x1 x2 x3 $char6.;`
`input x1 x2 x3;`
`array vx(3) $6 vx1 vx2 vx3`
`   ('x1' 'x2' 'x3');`
`y=vinformatnx(vx(1));`
`put y=;`
	`y=$CHAR`

See Also

Functions:

"Variable Information" functions in “Functions and CALL Routines by Category” on page 213

VINFORMATW

Returns the informat width that is associated with the specified variable

Category: Variable Information

Syntax

VINFORMATW (*var*)

Arguments

var
specifies a variable, expressed as a scalar or as an array reference.

Restriction: You cannot use an expression as an argument.

Comparisons

- VINFORMATW returns the informat width that is associated with the specified variable. VINFORMATWX, however, evaluates the argument to determine the variable name. The function then returns the informat width that is associated with that variable name.
- VINFORMATW does not accept an expression as an argument. VINFORMATWX accepts expressions, but the value of the specified expression cannot denote an array reference.
- Related functions return the value of other variable attributes, such as the variable name, type, and length, among others. For a list, see the "Variable Information" functions in “Functions and CALL Routines by Category” on page 213 .

Examples

SAS Statements	Results
`informat x $char6.;`	
`input x;`	
`y=vinformatw(x);`	
`put y=;`	`y=6`

See Also

Functions:

"Variable Information" functions in “Functions and CALL Routines by Category” on page 213

VINFORMATWX

Returns the informat width that is associated with the value of the specified argument

Category: Variable Information

Syntax

VINFORMATWX (*expression*)

Arguments

expression
specifies any SAS character expression that evaluates to a variable name.

Restriction: The value of the specified expression cannot denote an array reference.

Comparisons

- VINFORMATW returns the informat width that is associated with the specified variable. VINFORMATWX, however, evaluates the argument to determine the variable name. The function then returns the informat width that is associated with that variable name.
- VINFORMATW does not accept an expression as an argument. VINFORMATWX accepts expressions, but the value of the specified expression cannot denote an array reference.
- Related functions return the value of other variable attributes, such as the variable name, length, and type, among others. For a list, see the "Variable Information" functions in “Functions and CALL Routines by Category” on page 213 .

Examples

SAS Statements	Results

`informat x1 x2 x3 $char6.;`
`input x1 x2 x3;`
`array vx(3) $6 vx1 vx2 vx3`
`   ('x1' 'x2' 'x3');`
`y=vinformatwx(vx(1));`
`put y=;`
	`y=6`

See Also

Functions:

"Variable Information" functions in “Functions and CALL Routines by Category” on page 213

VINFORMATX

Returns the informat that is associated with the value of the specified argument

Category: Variable Information

Syntax

VINFORMATX (*expression*)

Arguments

expression
specifies any SAS character expression that evaluates to a variable name.

Restriction: The value of the specified expression cannot denote an array reference.

Details

VINFORMATX returns the complete informat name, which includes the width and the period (for example, $CHAR20.).

Comparisons

- VINFORMAT returns the informat that is associated with the specified variable. VINFORMATX, however, evaluates the argument to determine the variable name. The function then returns the informat that is associated with that variable name.
- VINFORMAT does not accept an expression as an argument. VINFORMATX accepts expressions, but the value of the specified expression cannot denote an array reference.
- Related functions return the value of other variable attributes, such as the variable name, length, and type, among others. For a list, see the "Variable Information" functions in "Functions and CALL Routines by Category" on page 213 .

Examples

SAS Statements	Results

`informat x1 x2 x3 $char6.;`
`input x1 x2 x3;`
`array vx(3) $6 vx1 vx2 vx3`
`   ('x1' 'x2' 'x3');`
`y=vinformatx(vx(1));`
`put y=;`
	`y=$CHAR6.`

See Also

Functions:
"Variable Information" functions in "Functions and CALL Routines by Category" on page 213

VLABEL

Returns the label that is associated with the specified variable

Category: Variable Information

Syntax

VLABEL (*var*)

Arguments

var
specifies a variable, expressed as a scalar or as an array reference.
Restriction: You cannot use an expression as an argument.

Details

If there is no label, VLABEL returns the variable name.

Comparisons

- VLABEL returns the label of the specified variable or the name of the specified variable, if no label exists. VLABELX, however, evaluates the argument to determine the variable name. The function then returns the label that is associated with that variable name, or the variable name if no label exists.
- VLABEL does not accept an expression as an argument. VLABELX accepts expressions, but the value of the specified expression cannot denote an array reference.
- VLABEL has the same functionality as CALL LABEL.
- Related functions return the value of other variable attributes, such as the variable name, informat, and format, among others. For a list, see the "Variable Information" functions in “Functions and CALL Routines by Category” on page 213 .

Examples

SAS Statements	Results
`array x(3) x1-x3;`	
`label x1='Test1';`	
`y=vlabel(x(1));`	
`put y=;`	`y=Test1`

See Also

Functions:
"Variable Information" functions in “Functions and CALL Routines by Category” on page 213

VLABELX

Returns the variable label for the value of a specified argument

Category: Variable Information

Syntax

VLABELX (*expression*)

Arguments

expression
specifies any SAS character expression that evaluates to a variable name.

Restriction: The value of the specified expression cannot denote an array reference.

Details

If there is no label, VLABELX returns the variable name.

Comparisons

- VLABEL returns the label of the specified variable, or the name of the specified variable if no label exists. VLABELX, however, evaluates the argument to determine the variable name. The function then returns the label that is associated with that variable name, or the variable name if no label exists.
- VLABEL does not accept an expression as an argument. VLABELX accepts expressions, but the value of the specified expression cannot denote an array reference.
- Related functions return the value of other variable attributes, such as the variable name, informat, and format, among others. For a list, see the "Variable Information" functions in “Functions and CALL Routines by Category” on page 213 .

Examples

SAS Statements	Results
`array x(3) x1-x3;`	
`array vx(3) $6 vx1 vx2 vx3`	
`   ('x1' 'x2' 'x3');`	
`label x1='Test1';`	
`y=vlabelx(vx(1));`	
`put y=;`	`y=Test1`

See Also

Functions:
"Variable Information" functions in "Functions and CALL Routines by Category" on page 213

VLENGTH

Returns the compile-time (allocated) size of the specified variable

Category: Variable Information

Syntax

VLENGTH (*var*)

Arguments

var
specifies a variable, expressed as a scalar or as an array reference.

Restriction: You cannot use an expression as an argument.

Comparisons

- □ LENGTH examines the variable at run-time, trimming trailing blanks to determine the length. VLENGTH returns a compile-time constant value, which reflects the maximum length.
- □ VLENGTH returns the length of the specified variable. VLENGTHX, however, evaluates the argument to determine the variable name. The function then returns the compile-time size that is associated with that variable name.
- □ VLENGTH does not accept an expression as an argument. VLENGTHX accepts expressions, but the value of the specified expression cannot denote an array reference.
- □ Related functions return the value of other variable attributes, such as the variable name, informat, and format, among others. For a list, see the "Variable Information" functions in "Functions and CALL Routines by Category" on page 213 .

Examples

SAS Statements	Results
`length x $8;`	
`x='abc';`	
`y=vlength(x);`	
`z=length(x);`	
`put y=;`	`y=8`
`put z=;`	`z=3`

See Also

Functions:

"Variable Information" functions in "Functions and CALL Routines by Category" on page 213

VLENGTHX

Returns the compile-time (allocated) size for the value of the specified argument

Category: Variable Information

Syntax

VLENGTHX (*expression*)

Arguments

expression

specifies any SAS character expression that evaluates to a variable name.

Restriction: The value of the specified expression cannot denote an array reference.

Comparisons

- LENGTH examines the variable at run-time, trimming trailing blanks to determine the length. VLENGTHX, however, evaluates the argument to determine the variable name. The function then returns the compile-time size that is associated with that variable name.
- VLENGTH returns the length of the specified variable. VLENGTHX returns the length for the value of the specified expression.
- VLENGTH does not accept an expression as an argument. VLENGTHX accepts expressions, but the value of the specified expression cannot denote an array reference.
- Related functions return the value of other variable attributes, such as the variable name, informat, format, among others. For a list, see the "Variable Information" functions in "Functions and CALL Routines by Category" on page 213 .

Examples

SAS Statements	Results
`length x1 $8;`	
`x1='abc';`	
`array vx(3) $6 vx1 vx2 vx3`	
`   ('x1' 'x2' 'x3');`	
`y=vlengthx(vx(1));`	
`z=length(x1);`	
`put y=;`	`y=8`
`put z=;`	`z=3`

See Also

Functions:
"Variable Information" functions in "Functions and CALL Routines by Category" on page 213

VNAME

Returns the name of the specified variable

Category: Variable Information

Syntax

VNAME (*var*)

Arguments

var
specifies a variable, expressed as a scalar or as an array reference.

Restriction: You cannot use an expression as an argument.

Comparisons

- VNAME returns the name of the specified variable. VNAMEX, however, evaluates the argument to determine a variable name. If the name is a known variable name, the function returns that name. Otherwise, the function returns a blank.
- VNAME does not accept an expression as an argument. VNAMEX accepts expressions, but the value of the specified expression cannot denote an array reference.
- VNAME has the same functionality as CALL VNAME.
- Related functions return the value of other variable attributes, such as the variable label, informat, and format, among others. For a list, see the "Variable Information" functions in "Functions and CALL Routines by Category" on page 213 .

Examples

SAS Statements	Results

`array x(3) x1-x3;`
`y=vname(x(1));`
`put y=;`
	`y=x1`

See Also

Functions:

"Variable Information" functions in "Functions and CALL Routines by Category" on page 213

VNAMEX

Validates the value of the specified argument as a variable name

Category: Variable Information

Syntax

VNAMEX (*expression*)

Arguments

expression
specifies any SAS character expression.

Restriction: The value of the specified expression cannot denote an array reference.

Comparisons

- □ VNAME returns the name of the specified variable. VNAMEX, however, evaluates the argument to determine a variable name. If the name is a known variable name, the function returns that name. Otherwise, the function returns a blank.
- □ VNAME does not accept an expression as an argument. VNAMEX accepts expressions, but the value of the specified variable cannot denote an array reference.
- □ Related functions return the value of other variable attributes, such as the variable label, informat, and format, among others. For a list, see the "Variable Information" functions in "Functions and CALL Routines by Category" on page 213.

Examples

SAS Statements	Results
`array x(3) x1-x3;`	
`array vx(3) $6 vx1 vx2 vx3`	
`      ('x1' 'x2' 'x3');`	
`y=vnamex(vx(1));`	
`z=vnamex('x'||'1');`	
`put y=;`	`y=x1`
`put z=;`	`z=x1`

See Also

Functions:
"Variable Information" functions in “Functions and CALL Routines by Category” on page 213

VTYPE

Returns the type (character or numeric) of the specified variable

Category: Variable Information

Syntax

VTYPE (*var*)

Arguments

var
specifies a variable, expressed as a scalar or as an array reference.

Restriction: You cannot use an expression as an argument.

Details

VTYPE returns N for numeric variables and C for character variables.

Comparisons

- VTYPE returns the type of the specified variable. VTYPEX, hovever, evaluates the argument to determine the variable name. The function then returns the type (character or numeric) that is associated with that variable name.
- VTYPE does not accept an expression as an argument. VTYPEX accepts expressions, but the value of the specified expression cannot denote an array reference.
- Related functions return the value of other variable attributes, such as the variable name, informat, and format, among others. For a list, see the "Variable

Information" functions in "Functions and CALL Routines by Category" on page 213 .

Examples

SAS Statements	Results

`array x(3) x1-x3;`
`y=vtype(x(1));`
`put y=;`
	`y=N`

See Also

Functions:

"Variable Information" functions in "Functions and CALL Routines by Category" on page 213

VTYPEX

Returns the type (character or numeric) for the value of the specified argument

Category: Variable Information

Syntax

VTYPEX (*expression*)

Arguments

expression
specifies any SAS character expression that evaluates to a variable name.

Restriction: The value of the specified expression cannot denote an array reference.

Details

VTYPEX returns N for numeric variables and C for character variables.

Comparisons

- VTYPE returns the type of the specified variable. VTYPEX, hovever, evaluates the argument to determine the variable name. The function then returns the type (character or numeric) that is associated with that variable name.
- VTYPE does not accept an expression as an argument. VTYPEX accepts expressions, but the value of the specified expression cannot denote an array reference.

- Related functions return the value of other variable attributes, such as the variable name, informat, and format, among others. For a list, see the "Variable Information" functions in “Functions and CALL Routines by Category” on page 213 .

Examples

SAS Statements	Results

`array x(3) x1-x3;`
`array vx(3) $6 vx1 vx2 vx3`
`   ('x1' 'x2' 'x3');`
`y=vtypex(vx(1));`
`put y=;`
	`y=N`

See Also

Functions:

"Variable Information" functions in “Functions and CALL Routines by Category” on page 213

WEEKDAY

Returns the day of the week from a SAS date value

Category: Date and Time

Syntax

WEEKDAY(*date*)

Arguments

date
specifies a SAS expression that represents a SAS date value.

Details

The WEEKDAY function produces an integer that represents the day of the week, where 1=Sunday, 2=Monday, . . . , 7=Saturday.

Examples

SAS Statements	Results

`x=weekday('16mar97'd);`
`put x;`
	`1`

YEAR

Returns the year from a SAS date value

Category: Date and Time

Syntax

YEAR(*date*)

Arguments

date
specifies a SAS expression that represents a SAS date value.

Details

The YEAR function produces a four-digit numeric value that represents the year.

Examples

SAS Statements	Results

`date='25dec97'd;`
`y=year(date);`
`put y;`
	`1997`

See Also

Functions:
“DAY” on page 315
“MONTH” on page 451

YIELDP

Returns the yield-to-maturity for a periodic cashflow stream, such as a bond

Category: Financial

Syntax

YIELDP(*A*,*c*,*n*,*K*,k_0,*p*)

Arguments

A
: the par value.

Range: $A > 0$

c
: the nominal per-period coupon rate, expressed as a fraction.

Range: $0 < c < 1$

n
: the number of coupons per period.

Range: $n > 0$ and is an integer

K
: the number of remaining coupons.

Range: $K > 0$ and is an integer

$\boldsymbol{k_0}$
: the time from present to the first coupon date, expressed in terms of the number of periods.

Range: $0 < k_0 < 1/n$

p
: the present value of the periodic cashflow stream.

Range: $p > 0$

Details

The YIELDP function is based on the relationship

$$P = \sum_{k=1}^{K} c(k) \frac{1}{\left(1 + \frac{y}{n}\right)^{t_k}}$$

where

$t_k = k - (1 - nk_0)$

$c(k) = \frac{c}{n}A$ for k=1, ..., K-1

$c(K) = \left(1 + \frac{c}{n}\right) A$

The YIELDP function solves for *y*.

Examples

```
p=yieldp(1000,1/100,4,14,.33/2,800);
```

The value returned is 0.077503.

YRDIF

Returns the difference in years between two dates

Category: Date and Time

Syntax

YRDIF(*sdate,edate,basis*)

Arguments

sdate
specifies a SAS date value that identifies the starting date.

edate
specifies a SAS date value that identifies the ending date.

basis
identifies a character constant or variable that describes how SAS calculates the date difference. The following character strings are valid:

’30/360’
specifies a 30-day month and a 360-day year in calculating the number of years. Each month is considered to have 30 days, and each year 360 days, regardless of the actual number of days in each month or year.

Alias: ’360’

Tip: If either date falls at the end of a month, it is treated as if it were the last day of a 30-day month.

’ACT/ACT’
uses the actual number of days between dates in calculating the number of years. SAS calculates this value as the number of days that fall in 365-day years divided by 365 plus the number of days that fall in 366-day years divided by 366.

Alias: ’Actual’

’ACT/360’
uses the actual number of days between dates in calculating the number of years. SAS calculates this value as the number of days divided by 360, regardless of the actual number of days in each year.

’ACT/365’
uses the actual number of days between dates in calculating the number of years. SAS calculates this value as the number of days divided by 365, regardless of the actual number of days in each year.

Examples

In the following example, YRDIF returns the difference in years between two dates based on each of the options for *basis*.

```
data _null_;
   sdate='16oct1998'd;
   edate='16feb2003'd;
   y30360=yrdif(sdate, edate, '30/360');
   yactact=yrdif(sdate, edate, 'ACT/ACT');
   yact360=yrdif(sdate, edate, 'ACT/360');
   yact365=yrdif(sdate, edate, 'ACT/365');
   put y30360= yactact= yact360= yact365=;
run;
```

SAS Statements	Results
put y30360=;	**4.333333333**
put yactact=;	**4.3369863014**
put yact360=;	**4.4**
put yact365=	**4.3397260274**

See Also

Functions:

“DATDIF” on page 311

YYQ

Returns a SAS date value from the year and quarter

Category: Date and Time

Syntax

YYQ(*year,quarter*)

Arguments

year
specifies a two- or four-digit integer that represents the year. The YEARCUTOFF= system option defines the year value for two-digit dates.

quarter
specifies the quarter of the year (1, 2, 3, or 4).

Details

The YYQ function returns a SAS date value that corresponds to the first day of the specified quarter. If either *year* or *quarter* is missing, or if the quarter value is not valid, the result is missing.

Examples

SAS Statements	Result
`dv=yyq(2001,3);`	
`put dv /`	`15157`
`dv date7. /`	`01JUL01`
`dv date9.;`	`01JUL2001`

See Also

Functions:

“QTR” on page 509

“YEAR” on page 618

System Option:

“YEARCUTOFF=” on page 1177

ZIPFIPS

Converts ZIP codes to FIPS state codes

Category: State and ZIP Code

Syntax

ZIPFIPS(*zip-code*)

Arguments

zip-code

specifies any SAS character expression containing a five-digit ZIP code.

Requirement: The character expressions you use must have a length of five, or the ZIPFIPS function generates an error.

Details

The ZIPFIPS function returns the two-digit numeric U.S. Federal Information Processing Standards (FIPS) code corresponding to its five-character ZIP code argument.

Examples

SAS Statements	Results

`fips=zipfips('27511');`
`put fips;`
	`37`

`a='27511';`
`fips=zipfips(a);`
`put fips;`
	`37`

See Also

Functions:
 "ZIPNAME" on page 623
 "ZIPNAMEL" on page 624
 "ZIPSTATE" on page 625

ZIPNAME

Converts ZIP codes to uppercase state names

Category: State and ZIP Code

Syntax

ZIPNAME(*zip-code*)

Arguments

zip-code
specifies any SAS character expression that contains a five-digit ZIP code.

Requirement: The character expressions you use must have a length of five, or the function generates an error.

Details

ZIPNAME returns the name of the state or U.S. territory that corresponds to its five-character ZIP code argument. ZIPNAME returns character values up to 20 characters long, all in uppercase.

Comparisons

The ZIPNAME, ZIPNAMEL, and ZIPSTATE functions take the same argument but return different values. ZIPNAME returns the uppercase name of the state or U.S. territory that corresponds to its five-character ZIP code argument. ZIPNAMEL returns the mixed case name of the state or U.S. territory that corresponds to its five-character ZIP code argument. ZIPSTATE returns the uppercase two-character state postal code (or world-wide GSA geographic code for U.S. territories) that corresponds to its five-character ZIP code argument.

Examples

The examples below show the differences when using ZIPNAME, ZIPNAMEL, and ZIPSTATE.

SAS Statements	Results

`state=zipname('27511');`
`put state;`
	`NORTH CAROLINA`

`state=zipnamel('27511');`
`put state;`
	`North Carolina`

`st=zipstate('27511');`
`put st;`
	`NC`

See Also

Functions:

“ZIPFIPS” on page 622

“ZIPNAMEL” on page 624

“ZIPSTATE” on page 625

ZIPNAMEL

Converts ZIP codes to mixed case state names

Category: State and ZIP Code

Syntax

ZIPNAMEL(*zip-code*)

Arguments

zip-code
specifies any SAS character expression that contains a five-digit ZIP code.

Requirement: The character expressions you use must have a length of five, or the function generates an error.

Details

ZIPNAMEL returns the name of the state or U.S. territory that corresponds to its five-character ZIP code argument. ZIPNAMEL returns mixed case character values up to 20 characters long.

Comparisons

The ZIPNAME, ZIPNAMEL, and ZIPSTATE functions take the same argument but return different values. ZIPNAME returns the uppercase name of the state or U.S. territory that corresponds to its five-character ZIP code argument. ZIPNAMEL returns the mixed case name of the state or U.S. territory that corresponds to its five-character ZIP code argument. ZIPSTATE returns the uppercase two-character state postal code (or world-wide GSA geographic code for U.S. territories) that corresponds to its five-character ZIP code argument.

Examples

SAS Statements	Results

state=zipname('27511');
put state;
	NORTH CAROLINA

state=zipnamel('27511');
put state;
	North Carolina

st=zipstate('27511');
put st;
	NC

See Also

Functions:

"ZIPFIPS" on page 622

"ZIPNAME" on page 623

"ZIPSTATE" on page 625

ZIPSTATE

Converts ZIP codes to state postal codes

Category: State and ZIP Code

Syntax

ZIPSTATE(*zip-code*)

Arguments

zip-code
specifies any SAS character expression that contains a five-digit ZIP code.

Requirement: The specified character expression must have a length of 5.

Details

ZIPSTATE returns the two-character state postal code (or world-wide GSA geographic code for U.S. territories) that corresponds to its five-character ZIP code argument. ZIPSTATE returns character values in uppercase.

Comparisons

The ZIPNAME, ZIPNAMEL, and ZIPSTATE functions take the same argument but return different values. ZIPNAME returns the uppercase name of the state or U.S. territory that corresponds to its five-character ZIP code argument. ZIPNAMEL returns the mixed case name of the state or U.S. territory that corresponds to its five-character ZIP code argument. ZIPSTATE returns the uppercase two-character state postal code (or world-wide GSA geographic code for U.S. territories) that corresponds to its five-character ZIP code argument.

Examples

The examples show the differences when using ZIPNAME, ZIPNAMEL, and ZIPSTATE.

SAS Statements	Results

`state=zipname('27511');`
`put state;`
	`NORTH CAROLINA`

`state=zipnamel('27511');`
`put state;`
	`North Carolina`

`st=zipstate('27511');`
`put st;`
	`NC`

See Also

Functions:
- "ZIPFIPS" on page 622
- "ZIPNAME" on page 623
- "ZIPNAMEL" on page 624

References

Abramowitz, M. and Stegun, I. (1964), *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables — National Bureau of Standards Applied Mathematics Series #55,* Washington, DC: U.S. Government Printing Office.

Amos, D.E., Daniel, S.L., and Weston, K. (1977), "CDC 6600 Subroutines IBESS and JBESS for Bessel Functions $I(v,x)$ and $J(v,x)$, $x \geq 0$, $v \geq 0$," *ACM Transactions on Mathematical Software*, 3, 76–95.

Aho, A.V., Hopcroft, J.E., and Ullman, J.D., (1974), The Design and Analysis of Computer Algorithms, Reading, MA: Addison-Wesley Publishing Co.

Cheng, R.C.H. (1977), "The Generation of Gamma Variables," *Applied Statistics*, 26, 71–75.

Duncan, D.B. (1955), “Multiple Range and Multiple F Tests,” *Biometrics*, 11, 1–42.

Dunnett, C.W. (1955), “A Multiple Comparisons Procedure for Comparing Several Treatments with a Control,” *Journal of the American Statistical Association*, 50, 1096–1121.

Fishman, G.S. (1976), “Sampling from the Poisson Distribution on a Computer,” *Computing*, 17, 145–156.

Fishman, G.S. (1978), *Principles of Discrete Event Simulation*, New York: John Wiley & Sons, Inc.

Fishman, G.S. and Moore, L.R. (1982), “A Statistical Evaluation of Multiplicative Congruential Generators with Modulus (2^{31} – 1),” *Journal of the American Statistical Association*, 77, 1 29–136.

Knuth, D.E. (1973), *The Art of Computer Programming, Volume 3. Sorting and Searching*, Reading, MA: Addison-Wesley.

Hochberg, Y. and Tamhane, A.C. (1987), *Multiple Comparison Procedures*, New York: John Wiley & Sons, Inc.

Williams, D.A. (1971), “A Test for Differences Between Treatment Means when Several Dose Levels are Compared with a Zero Dose Control,” *Biometrics*, 27, 103–117.

Williams, D.A. (1972), “The Comparison of Several Dose Levels with a Zero Dose Control,” *Biometrics*, 28, 519–531.

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS® Language Reference, Version 8*, Cary, NC: SAS Institute Inc., 1999.

SAS® Language Reference, Version 8

Copyright © 1999 by SAS Institute Inc., Cary, NC, USA.

ISBN 1–58025–369–5

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

1st printing, October 1999