Paper 57

Introduction to SAS Functions

Neil Howard, Independent Consultant, Charlottesville, VA

Abstract

A FUNCTION returns a value from a computation or system manipulation that requires zero or more arguments. And, like most programming languages, the SAS System provides an extensive library of “built-in” functions. SAS has more than 190 functions for a variety of programming tasks. This tutorial will cover the syntax for invoking functions, an overview of the functions available, examples of commonly used functions, selected character handling and numeric functions, and some tricks and applications of functions that will surprise you.

Breaking Down a FUNCTION - Syntax

Given the above definition of a function, the syntax and components should be examined. A function recognized in a SAS statement by the use of a function name, followed immediately by function argument(s), separated by commas, and enclosed in parentheses.

For a function requiring two arguments the syntax is as follows:

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

where the arguments are:
     constants
     variables
     expressions
     other functions
```

Several functions take no arguments, in which case a null set of parentheses is required. For instance, the TODAY function returns today’s date from the system clock, requiring no arguments:

```
DATA NEW;
  X = TODAY() ;
  PUT ‘X=  ‘   X   MMDDYY8. ;
RUN;

X= 01/18/99;
```

The variable X is returned as the numeric SAS date representation and should be displayed with a format.

Breaking Down a FUNCTION - Arguments

The arguments to any given function can be variables, constants, expressions, and/or other functions.

constant	X = SQRT(9562) ;
variable	Y = ABS(BAL) ;
expression	Z = MAX((BAL-DEBIT),(NEWCAR+GAS)) ;
function	Q = ABS(SQRT(ABC-64)) ;

In these examples, X is the square root of 9562, Y would be the absolute value of the variable BAL, Y contains the value of the larger of the two expressions, and Q is the absolute value of the square root of ABC minus 64. A similar numeric function, MIN, returns the value of the lowest of its nonmissing arguments. The SUM function requires at least two arguments and returns the sum of the nonmissing arguments.

```
TOTAL = SUM( X, Y, Z ) ;
```

Use of the keyword OF gives the user the flexibility to include variable lists, array elements and other shortcuits for referencing variable names.

```
A = SUM( OF TEMP1 - TEMP24 );
B = SUM( OF TEMP1 TEMP2 TEMP3 );
C = SUM( OF TMPARRAY {*} );
D = SUM( OF _NUMERIC_ );
E = SUM( OF TEMP1 -- TEMP24 );
```

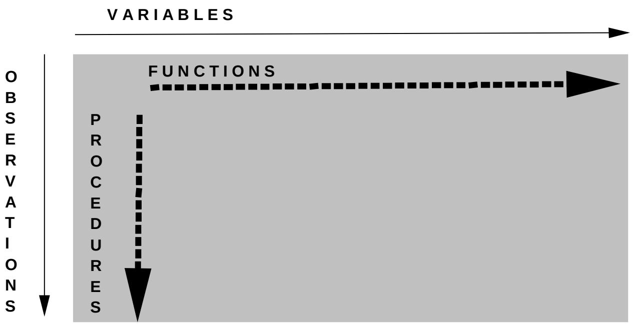
In the examples above, A gives you the total of 24 consecutive temperature readings where numbered variable names were used. Using no commas, you can sum three temperature value to calculate B. When an array named TMPARRAY has been defined, you can pass the elements to the SUM function to get C. All numeric variables in the program data vector (PDV) are added to produce D, and E is derived by adding all variables in placement order in the PDV between and including TEMP1 and TEMP24.

Categories of FUNCTIONS

The library of functions contains several categories of functions, including: arithmetic (like ABS, MIN, MAX, SQRT), array (e.g., DIM), character handling (e.g., LEFT, RIGHT, SUBSTR, REVERSE, LENGTH), date and time (e.g., TODAY, JULDATE, MDY, INTCK, TIMEPART), financial (e.g., MORT, NPV, SAVING), mathematical (e.g., LOG, EXP), probability (e.g., POISSON, PROBCHI), quantile, random number (e.g., NORMAL, UNIFORM), sample statistics (e.g., MEAN, MIN, MAX, STD, NMISS), special functions (e.g., LAG, PUT INPUT), state and zip code, trigonometric and hyberbolic, and truncation (ROUND, CEIL, FLOOR). Chapter 11 of the SAS Language manual for the complete list and details.

FUNCTIONS vs. PROCEDURES

Some functions that are commonly used compute the sum (SUM), arithmetic mean (MEAN), variance (VAR), standard deviation (STD), minimum value (MIN), and maximum value (MAX). These functions compute the same simple descriptive statistics available in PROC MEANS, however. The fundamental difference between functions and procedures is that a function expects the argument values to supplied across an observation in a SAS data set. Procedures expects one variable value per observation.



SAS Data Set

The following code calculates the average temperature per day using the MEAN function executed for each observation. The resulting new data set and new variable AVGTEMP are passed to PROC MEANS to calculate the average temperature per month. Note that, not only if the variable list notation used as a shortcut for specifying the function arguments, the OF keyword prevents the argument specification from being misinterpreted as the expression T1 minus T24.

```
data average ;
   set temp ;
   avgtemp = mean( of T1 - T24 ) ;
run ;

proc sort ;
   by month ;
run ;

proc means ;
   by month ;
   var avgtemp ;
run ;
```

Missing Values

Remembering that functions must be used in SAS statements and that missing values propogate, be aware of how each function handles missing values. Rely on the SAS Language manual specs and your own programmatic testing code to validate your intended results.

For example, the MEAN function will return the arithmetic average for the nonmissing arguments, using the number of nonmissings as the denominator. Likewise, SUM totals all the nonmissing arguments. However, if all the arguments are missing, the total will be missing, NOT zero, which could effect later calculations in your program. To force a zero total, include the constant on your calculation:

```
X = SUM( A,B,C,D,E,F,0 );
Y = SUM( OF T1 - T24, 0 );
```

The functions NMISS and N allow you to determine the number of missing values and nonmissings, respectively, in the argument list.

```
A = NMISS( A,B,C,D,E,F ); * # of missings;
B = N( L,M,N,O,P );        * # of nonmissings;
```

Length of Target Variables

Target refers to the receiving variable on the left side of the equal sign in the SAS statement where a function is used on the right to calculate or otherwise produce a result. The default length for a numeric target is 8; however, for some character functions, the default length of the target variable is 200 or the length of the source variable (argument). The SCAN function returns a given word from a character string using default and specified delimiters.

```
DATA NEW ;
   X = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ' ;
   Y = SCAN(X, 1, 'K' ) ;
   PUT  Y=  ;
RUN ;

Y = ABCDEFGHIJ
```

In the example above, the variable X has a length of 26, and the SCAN function is searching X for the first ‘word’ using K as the delimiter. A PROC CONTENTS run on the data set NEW will show the length for Y in the descriptor as 200.

SUBSTR is a commonly used character handling function that extracts a string from an argument or replaces character value contents. The function takes three arguments: the source (or argument), the starting position in constant or variable form, and the length of the substring expressed as a constant or variable.

```
DATA NEW ;
   SET OLD ;  *** CONTAINS A B C;
   X = SUBSTR( A, 23, 4 ) ;
   Y = SUBSTR( A, B, 3 ) ;
   Z = SUBSTR( A, 9, C ) ;
   PUT A=  B=  C=  X=  Y=  Z=  ;
RUN ;

A = ABCDEFGHIJKLMNOPQRSTUVWXYZ
B = 2
C = 9
X  = WXYZ
Z = IJKLMNOPQ
```

A PROC CONTENTS of data set NEW would show that variable A is character with a length of 26, B and C are character with length of 8, and X, Y, and Z are character with lengths of 26. In the case of X and Y, a LENGTH statement coded before the assignment statements; it will reserve only the number of bytes needed in the PDV. Since the value of the variable C is unknown at compile time, the length of the source variable is used by default.

Consider a case where data is received in 200-byte character chunks defined as variables. Hopefully, extensive documentation describes the 'layout' for extracting the individual variable values.

```
DATA NEW ;
   SET OLD ;  **contains VAR_200;
   IDNUM = SUBSTR(VAR_200, 1, 10 ) ;
   NAME  = SUBSTR(VAR_200, 11, 25) ;
   AGE     = SUBSTR(VAR_200, 36, 2) ;
RUN ;
```

For efficiency, be sure to DROP VAR_200 and code a length statement for the derived variables. Otherwise, IDNUM, NAME, and AGE will all have a default length of 200. Depending on the size of your data set, you may stress your DASD or disk space unnecessarily. Your data would also be difficult to work with in default condition.

```
DATA NEW (DROP=VAR_200) ;
   SET OLD ;  **contains VAR_200 ;
   LENGTH IDNUM $ 10
                NAME  $ 25
                AGE    $   2 ;
   IDNUM = SUBSTR(VAR_200, 1, 10 ) ;
   NAME  = SUBSTR(VAR_200, 11, 25) ;
   AGE     = SUBSTR(VAR_200, 36, 2) ;
RUN ;
```

The Version 5 story held that there were two types of functions: supervisor and library. Supervisor functions were part of the SAS supervisor (now the DATA step processor in Version 6). Library functions were external routines, linked to and invoked by the supervisor. Parameters were passed to them and results returned. Supervisor functions could 'know' information about the argument at compile time and take action regarding the length of the target variable. Library functions do not interact with the calling environment so maximum character lengths were used as defaults. SCAN is the only function were the maximum is still return. This may not be an issue in future releases.

More on SUBSTR(???)

As mentioned, SUBSTR is an example of character string handling function. You cannot substring numerics, unless values were first converted to character, then passed to SUBSTR. Another less obvious solution is to use the MOD and INT numeric functions.

The first argument to the MOD function is a numeric, the second is a non-zero numeric; the result is the remainder when the integer quotient of argument-1 is divided by argument-2. The INT function takes only one argument and returns the integer portion of an argument, truncating the decimal portion. Note that the argument can be an expression.

```
DATA NEW ;
   A = 123456 ;
   X = INT( A/1000 ) ;
   Y = MOD( A, 1000 ) ;
   Z = MOD( INT( A/100 ), 100 ) ;
   PUT A= X= Y= Z=  ;
RUN ;

A=123456
X=123
Y=456
Z=34
```

SUBSTR as a Pseudo-Variable

In an old discussion on SAS-L, participants were intrigues by the use of SUBSTR as a pseudo-variable. If the function is used on the left side of the equal sign in an assignment statement, the SAS System places the value of the expression on the right into the argument of the SUBSTRed expression, beginning with the position you indicate with the second argument, for the length specified in the third argument. The following example is lifted from that SAS-L discussion, with apologies to the original author.

```
DATA FIXIT;
   SET OLD;
   PUT NAME=  '(Before)' ;
   WHERE = INDEX(NAME, 'Ratface' ) ;
   IF (WHERE = 0)
        THEN NAME = TRIM(NAME) || ', Evil Genius' ;
   ELSE SUBSTR(NAME, WHERE, 13) = 'The Powerful' ;
   PUT WHERE= NAME= '(After)' ;
RUN ;

(first execution)
NAME = Duke Owen (Before)
WHERE = 0  NAME = Duke Owen, Evil Genius (After)

(second execution)
NAME = Duke Ratface Owen (Before)
WHERE = 6  NAME = Duke The Powerful (After)
```

Data Conversions

The SAS log message stating that “character values have been converted to numeric: or vice versa, politely advises you that you gave mixed your data types in a SAS statement, and, where possible, the DATA step processor has performed automatic conversions to make your transaction possible. The PUT and INPUT functions allow you to take control of these types of conversions, for clarity, accuracy, efficiency, and data integrity:

```
PUT(source, format)
INPUT(source, informat)
```

The PUT function returns the value of source written with a specified format. The format type must match the source type (character or numeric). Note that the result of the PUT function al always a character string. The INPUT function allows you to read the source with a specified informat. The result can be character or numeric, depending on the informat.

```
DATA NEW ;
  A = 1234 ;
  B = '789,321' ;
  X = PUT( A, 4. ) ;
  Y = INPUT( B, COMMA7. ) ;
  PUT X=  '   CHARACTER' ;
  PUT Y=  '   NUMERIC' ;
RUN ;

X = 1234    CHARACTER
Y = 789321    NUMERIC
```

Table Lookup and More

The PUT and INPUT functions are being cleverly used for a variety of data transformation applications. One such requirement is table lookup, made easy with the PUT function.

```
PROC FORMAT ;
   VALUE REGFMT 1,5,6,9,11-15 = 'NW'
                2,7,10        = 'SW'
                3,4,8,16      = 'NE'
                17-20         = 'SE';
RUN;

DATA RECODE ;
  INFILE DATA ;
  INPUT COUNTY POP ;
  REGION = PUT(COUNTY, REGFMT. ) ;
RUN ;

PROC SORT ;
  BY REGION ;
RUN ;

PROC PRINT ;
  BY REGION ;
  VAR POP ;
  SUM POP ;
RUN ;
```

The PROC FORMAT in this example generates a lookup table, and the PUT function searches the table and returns the label that maps to the variable value for COUNRTY and stores the label in a new character variable REGION. The data step creates a new variable and the PROC PRINT will show the population now calculated at the region level.

```
DATA SUB_NW ;
  INFILE DATA ;
  INPUT COUNTY POP ;
  IF PUT( COUNTY, REGFMT. ) = 'NW' ;
RUN ;
```

Used in this way, the PUT function performs the table lookup for subsetting purposes without creating a new variable.

Character Handling Functions

Both an UPCASE and LOWCASE are now available.

```
DATA NEW ;
  X = 'text' ;
  Y = UPCASE(X) ;
  Z = LOWCASE (Y) ;
  PUT X= Y= Z= ;
RUN ;

X = text
Y = TEXT
Z = text
```

The LEFT and RIGHT functions justify the variable values accordingly. The length of the argument variable does not change.

Consider the use of these functions where the value of an existing character variable is to be selection criteria for a subset. In the example below, the subset only contains observations where DRUG is equal to “Placebo”. Note that the data may have been keyed and stored inconsistently with respect to case. To ensure accurate selection, temporarily modify the value to compare with a constant.

```
DATA CONTROLS ;
  SET OLD ;
  IF UPCASE(LEFT(DRUG)) = 'PLACEBO' ;
RUN ;
```

The COMPRESS function removes every occurrence of specific characters (blanks or other) in a character string. COMPBL compresses multiple blanks between words in a text string to a single blank, but not affect single blanks.

```
DATA NEW ;
  STRING = 'TEXT    IS      A  MESS' ;
  X = COMPRESS(STRING) ;
  Y = COMPBL(STRING) ;
  PUT X= Y= ;
RUN ;
```

```
X = TEXTISAMESS
Y = TEXT IS A MESS
```

The TRANSLATE function replaces specific characters in a string with individual characters you specify, returning an altered string. TRANWRD replaces or removes all occurrences of a word in a string. Note the syntax:

```
TRANSLATE (source, to-1, from-1,.....)
  replaces specific characters in a string with
  Individual characters that you specify

TRANWRD ( source, target, replacement)
  replaces or removes all occurrences of a word or
string

DATA _NULL_ ;
  *** write original value of variable DATA to log ;
  DATA = '1234,ABC,X,Y,Z' ;
  PUT DATA=   'PROGRAM OUTPUT' ;

  *** translate all commas in DATA to blanks ;
  *** write new value to log ;
  DATA = TRANSLATE(DATA, ' ', ',' ) ;
  PUT DATA=   'AFTER' ;

  *** declare original values of OLD and TEXT ;
  OLD = 'BAD' ;
  TEXT = ' NO BAD NEWS' ;
  PUT OLD= TEXT=  '(BEFORE)' ;

  *** text substitution;
  *** substitute new for old ;
  TEXT = TRANWRD (TEXT, OLD, 'NEW' ) ;
  PUT TEXT=   '(AFTER)'  ;
RUN ;

DATA = 1234,ABC,X,Y,Z    PROGRAM OUTPUT
DATA = 1234 ABC X Y Z    (AFTER)
OLD = BAD  TEXT = NO BAD NEWS  (BEFORE)
TEXT = NO NEW NEWS  (AFTER)
```

TRIM removes trailing blanks from a character string. TRIMN does the same thing, except TRIM returns one blank for a blank string and TRIMN returns a null string. These functions are especially useful in concatenation operations.

```
DATA _NULL_ ;
  STRING1 = 'TRIMMED     ' ;
  STRING2 = '?' ;
  STRING3 = '!' ;
  STRING4 = '                    ' ;
  W = STRING1 || STRING2 ;
  X = TRIM(STRING1) || STRING3 ;
  Y = STRING2 || TRIM(STRING4) || STRING3 ;
  Z = STRING2 || TRIMN(STRING4) || STRING3 ;
RUN ;

W = TRIMMED      ?
X = TRIMMED!
Y = ? !
Z = ?!
```

INDEX requires two arguments, the first identifies the character string to be searched, the second identifies the character string to search for in the first argument. INDEX searches argument-1 from left to right for the first occurrence of the string specified in argument-2 and returns the position of the found string's first character. If the string is not found, INDEX returns 0.

The INDEXC function searches argument-1 from left to right for the first occurrence of any character present in the other arguments and returns an integer representing the position in argument-1 of the first character found. INDEXC also returns a value of 0 if none of the characters is found.

```
TEXT = 'SUGI 24, MIAMI, FL' ;
X = '4,M' ;

INDEX_X = INDEX( TEXT, X ) ;
PUT INDEX_X=  ;

INDEXC_Y = INDEXC( TEXT, '0123456789', ';()=. ' ) ;
PUT INDEXC_Y=  ;

INDEX_X = 7
INDEXC_Y = 6
```

Numeric Functions and Sample Selection

As seen before, the MOD function takes two numeric arguments and returns the remainder of argument-1 divided by argument-2. Consider the use of the MOD function in generating a random sample where the input data set is sorted by the numeric variable IDNUM and the user intends to select every 100th patient.

```
DATA TEST ;
  SET PATIENTS ;
  IF MOD(IDNUM, 100) = 0 ;
RUN ;
```

Similarly, a random number generating function can be used for approximated sized samples.

```
DATA TESTDATA ;
  SET PATIENTS ;
  IF UNIFORM( 0 ) < .10 ;  ** approximately 10% ;
RUN;
```

Random number generating functions require a seed (see Chapter 3 of the SAS Language manual for a complete discussion of seed. The are several functions that generate numbers based on specific algorithms. UNIFORM returns a number from the uniform distribution in the interval (0, 1).

A more efficient use of these types of functions in sample select involves the use of the POINT= option on the SET statement to read the input SAS data set using direct access by observation number. When the random number generated by UNIFORM is multiplied by the number of observations in the input data set (available to the SET statement at compile time), you select random observations.

```
DATA SAMPLE ;
  DO J = 1 TO 10 ;
      PTR = (INT(UNIFORM(0) * NOBS )) + 1 ;
```

```
        SET OLD POINT = PTR  NOBS = NOBS ;
        SELPTR = PTR;
        OUTPUT ;
     END ;
     STOP ;  * required to stop data step processing ;
   RUN ;

   PROC PRINT NOOBS LABEL ;
   VAR J SELPTR ;
   LABEL SELPTR = 'Observation Number Selected';
   RUN ;

                     Observation
                       Number
              J       Selected

              1         124
              2         938
              3         289
              4         174
              5         923
              6         499
              7         246
              8         222
              9         335
             10          90
```

More Numeric Functions

The RANGE function takes at least 2 numeric arguments and returns the difference between the largest and the smallest of the nonmissing arguments:

```
   DATA _NULL_  ;
     A=10  ;
     B=27  ;
     C=3 ;

     ARRAY QUESTION Q1-Q5 ;


     DO J=1 to DIM(QUESTION) ;
       QUESTION{J} = INT(UNIFORM(50) *100) ;
       PUT QUESTION{J}= ;
     END ;

     X=RANGE(A, B, C) ;
     Y=RANGE(-1,  . ,  1 ) ;
     Z=RANGE( OF Q1-Q5 ) ;

     PUT X=  ;
     PUT Y=  ;
     PUT Z= ;
   RUN ;

   Q1=24
   Q2=50
   Q3=99
   Q4=96
   Q5=8
   X=24
   Y=2
   Z=91
```

The FLOOR function takes one numeric argument and returns the largest integer that is less than or equal to the argument. The CEIL function returns the smallest integer that is greater than or equal to the argument.

```
   A = FLOOR( 2.1 ) ;          ** A=2 ;
   B = FLOOR( 3 ) ;            ** B=3 ;
   C = CEIL( 2.1 ) ;           ** C=3 ;
   D = CEIL( 3 ) ;             ** D=3 ;
```

Certain manipulations may require that you know the length of particular variable for the current observation. The LENGTH function takes one argument and returns its length.

```
   DATA _NULL_;
      PAT = 'E. Pluribus Turner';
      PAT2 = '  ';

      Y=LENGTH(PAT);
      Z=LENGTH(PAT2);
      PUT Y=;
      PUT Z=;
   RUN;

   Y=18
   Z=1
```

Note that when the argument is missing, LENGTH returns a value of 1.

Applications with Embedded Functions

(from recent SAS-L discussions)

Problem 1: The variable NAME in data set A was coded in all caps. Rewrite the value such that the first character of each word is upper case and the remainder of the word is in lower case.

```
DATA  B ( KEEP = NAME  NEWNAME) ;
   SET A ;  ** contains variable NAME in all caps ** ;

   A1 = SCAN( NAME, 1) ;
   A2 = SCAN( NAME, 2) ;

   NEWNAME = SUBSTR(A1 ,1 ,1) ||
             LOWCASE( SUBSTR(A1, 2, LENGTH( A1 ) - 1 )) ||
             ' ' ||
            SUBSTR( A2, 1, 1 ) ||
             LOWCASE( SUBSTR( A2, 2, LENGTH( A2) - 1 )) ;

   PUT NAME=  NEWNAME= ;
RUN ;

NAME=RESPIRATORY SYSTEM   NEWNAME=Respiratory System
NAME=CARDIOVASCULAR UNIT  NEWNAME=Cardiovascular Unit
NAME=PEDIATRIC WING       NEWNAME=Pediatric Wing
```

Problem 2: Parse a filename dump for the date. The original solution stopped working when a numeric became part of the prefix.

```
data new;
  record="scu3lly981101.xfiles"; output;
  record="mul7dar981102.xfiles"; output;
  record="how9ard981103.xfiles"; output;
run;

data substr;
```

```
set new;
newdate=put(input(substr(record, index(record,'.')-
6,6), yymmdd6.), mmddyy8.);
run;

proc print noobs;
run;
```

```
           RECORD         NEWDATE
scu3lly981101.xfiles      11/01/98
mul7dar981102.xfiles      11/02/98
how9ard981103.xfiles      11/03/98
```

DATE and TIME Functions

The tutorial has not dealt with the many DATE and TIME functions. This group of functions is a tutorial unto itself, and I highly recommend the following presentation scheduled today: beginning tutorial presented by **Andy Karp Monday at 2:30 entitled "Working With SAS Date and Time Functions with Reference to Year 2000 Issues"** will include an impressive tutorial on DATE and TIME functions that will also touch on Y2K.

The following publications are also excellent: **"Working With SAS DATE and TIME Functions"** by Andy Karp, published in the Proceedings of the Twenty First Annual SUGI Conference in Chicago, IL, 1996. **"DATE Intervals - The INTCK Function"** by Debbie Hoppe published in the Proceedings of the Third Annual SESUG Conference (SouthEast SAS Users Group) in Raleigh, NC, 1995.

Conclusion

The intent of this paper was not to illustrate the syntax and invocation of every SAS function, but to tantalize. The breadth of 'functionality' provided in invaluable to the SAS programmer. Chances are you've hardcoded a problem for which there exists a function solution. Give Chapter 11 a long, hard look.

RECOMMENDATIONS

In his book **In the Know: SAS Tips and Techniques from Around the Globe**, Phil Mason has an interesting chapter that deals with some of the defaults and peculiarities of some SAS functions. The behavior of functions is not always intuitive, and it pays to examine your results and test your code thoroughly.

BONUS

From the Valley of the Sun Users Group (VALSUG), Phoenix, AZ, July 1998 Newsletter: (with apologies to Andy Rooney)

"Why is there a DIM function and not a BRIGHTEN one?"

"And a MEAN function and not a NICE function?"

"The TRANSLATE function - that sounds like one I'd have to use a lot."

"And with a FLOOR function and a CEIL function, wouldn't you think they'd need a WALL function?"

Reference

SAS Institute Inc. SAS Language: Reference, Version 6, First Edition, Cary, NC: SAS Institute Inc., 1990.