

Discovering the FUN in Functions!

Neil Howard

MCI Telecommunications Corporation

Abstract and Definition

A FUNCTION returns a value from a computation or system manipulation. Like most programming languages, SAS Software provides a library of built-in functions. Including the new ones available in Version 6.07, SAS has over 190 functions that provide slick solutions to everyday programming needs. This tutorial will cover the syntax for invoking functions, a walk-through of types available, examples of commonly used functions, a sampling of 6.07 additions we've been waiting for, 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 is 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.

Several functions take no arguments, in which case a set of null parentheses is used. 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= 10/08/92
```

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, and/or expressions.

```
X=ABS(BAL);
Y=SQRT(9562);
Z=MAX((BAL-DEBITS),(NEWCAR+GAS));
```

In these examples, X would contain the absolute value of the variable

balance, Y would be the square root of the constant 9562, and Z would return the greater (maximum) of the result of two expressions, the first yielding balance minus debits, the second the cost of a new car and gas to run it.

Using the keyword OF, the user has the flexibility to include variable lists, array elements, and other shortcuts to 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);
```

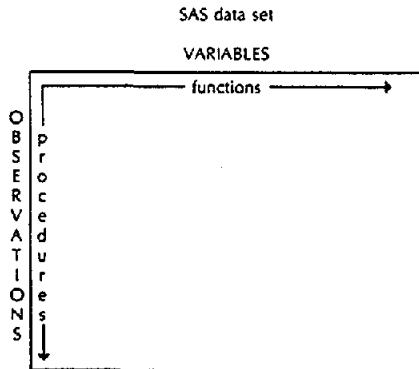
In the above examples, A gives you the total of 24 consecutive temperature values, where the variables were coded with numbered variable names. Using no commas, you can sum three temperature values to get B. If an array named TMPARRAY has been defined, you can pass the elements to the SUM to get C. All 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 fall into variety of categories, including: arithmetic (like ABS, MAX, MIN, SQRT), array (DIM...), character handling (with LEFT, RIGHT, SUBSTR, REVERSE, LENGTH...), date and time (TODAY, JULDATE, TIMEPART, INTCK, MDY....), financial (MORT, NPV, SAVING...), mathematical (LOG, EXP), probability (POISSON, PROBCHI...), quantile, random number (NORMAL, UNIFORM...), sample statistic (MEAN, MIN, MAX, STD, NMISS, VAR...), special functions (LAG, PUT, INPUT), state and zip code, trigonometric and hyperbolic, and truncation (ROUND, INT, FLOOR...). The SAS Language manual lists each with examples.

FUNCTIONS vs. PROCEDURES

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



The following code calculates the average temperature per day using the MEAN function executed for each observation. The resulting new data set and variable AVGTEMP is passed to PROC MEANS to calculate the average temperature per month. Note that, not only is the variable list notation used as a shortcut for specifying the function arguments, the OF keyword prevents the 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 MEAN;
  BY MONTH;
  VAR AVGTEMP;
RUN;
```

Missing Values

Remembering that functions must be used in SAS statements and that missing values propagate, be aware of how each function treats missing values. Rely on the SAS Language manual and testing routines to validate your intended results.

For example, the MEAN function will return the arithmetic average of 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 can effect calculations down the road. To force a zero total, include the constant in your statement:

```
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 that went into a calculation.

Length of Target Variables

Target refers to the variable on the left of the equals sign in the SAS statement where a function is used on the right to calculate or 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 or specified delimiters.

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

Y=ABCDEFGHIJ
```

In this example, the variable X has a length of 26, and the SCAN function is searching for the first word using K as the delimiter. A PROC CONTENTS will show the length of Y in the descriptor is 200.

SUBSTR is a commonly used character handling function that extracts a substring from an argument or replaces character value contents. The function uses 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=ABCEDEFGHIJKLMNOPQRSTUVWXYZ
B=2
C=9
X=WXYZ
Y=BCD
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 numeric 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 can be used to reserve only the number of bytes needed in the PDV; however, since the length indicated by 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 variables with extensive documentation describing the "layout" for extracting the individual meaningful variables.

```
DATA NEW;
  SET OLD; *** contains
    VAR1_200,VAR2_200;
  IDNUM = SUBSTR(VAR1_200,1,10);
  NAME = SUBSTR(VAR1_200,11,25);
  AGE = SUBSTR(VAR1_200,36,2);
RUN;
```

In this example, be sure to DROP VAR1_200 and VAR2_200, and code a LENGTH statement from your documentation for IDNUM, NAME, AGE, etc. Otherwise each variable in the resulting data set will have a length of 200 by default. Depending on the number of observations in your data set, you may stress your DASD or disk space unnecessarily. Your data would also be difficult to work with in this condition.

The Version 5 story held that there were two types of functions: supervisor and library. Supervisor functions were part of the 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. This may not be an issue in future releases.

More on SUBSTR (?)

As mentioned is an example of a character handling function. A similar operation is often required on numerics, however. One possible solution is to convert the numerics to character so that SUBSTR will be effective. Another less obvious solution is to use the MOD and INT numeric functions.

The first argument to the MOD functions is a numeric, the second in 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 the 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
```

Note also that you can code a function within a function within a function.....

Data Conversions

The SAS log message stating that "character values have been converted to numeric", or vice versa, politely advises you that you have mixed your data types on a SAS statement, and where possible, the data step processor has performed the conversion to make your transaction possible. The PUT and INPUT functions allow you take control of such conversion, for clarity, accuracy, 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 be the source type (character or numeric). Note that the result of the PUT is always a character string. The INPUT function allows you to read the source with a specified format. The result will 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 to the variable REGION. The data step creates a new variable and the report will show the population 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.

Old and New

Among the Version 6 new offerings are four in particular that complement existing Version 5 functions. The UPCASE function has been available; LOWCASE is now here:

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

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

The COMPRESS function removes every occurrence of specific characters (blanks or other) from a character string. In Version 6, COMBBL compresses multiple blanks between words in the text string to a single blank, but does not affect single blanks.

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

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

The TRANSLATE functions replace specific characters in a string with individual characters you specify, returning an altered string. Version 6 expands the concept with TRANWRD, which replaces or removes all occurrences of a word in a string. Note the syntax:

```
TRANSLATE(source,to-1,from-1,.....)
TRANSWRD(source,target,replacement)
```

TRIM removes trailing blanks from a character string. The Version 6 TRIMN does the same thing. However, TRIM one blank for a blank string; TRIMN returns a null string. These functions are especially useful in concatenation operations.

```

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

X = TRIMMED    ?
Y = TRIMMED!
Z = ?!

```

Conclusion

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

Bibliography

SAS Institute Inc., SAS Views: SAS Basics. Cary, NC: SAS Institute Inc., 1986.

SAS Institute Inc., SAS Views: SAS Processing. Cary, NC: SAS Institute Inc., 1987.

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

SAS Institute Inc., SAS Technical Report P-222, Changes and Enhancements to Base SAS Software, Release 6.07, Cary, NC: SAS Institute Inc., 1991.