

Functions are Your Friends, or

-- "Honey, Could You Get Me the SQRT of DATE()?";

Kathe Richards, Technical Support Analyst
Texas Comptroller of Public Accounts
Austin, Texas

The first time I saw a SAS Function I thought "What's going on here?". I knew some programming languages, but COBOL never FUNCTIONed like this! I've come a long way since then. Now, anytime I run up against a problem that looks common but tedious, I look for one of my function friends. They're nifty, pre-compiled, and save a lot of work. There are 148 of them documented in Base SAS. I'd like to introduce you to some of my favorites.

A function is a little clump of stored code that performs a specific -- function. It's used mostly in DATA step programming, as part of a SAS statement. To use a function, you must call it by name and usually pass it something to work on. It returns a result -- a number, a character string, a Boolean value -- in the spot you called it from (or sometimes a message complaining about the substandard quality of your input.) The items you pass it to work with are called arguments. They're always packaged in parentheses. That's how you know it's a function -- It's always followed by a set of parentheses (that usually have something between them.)

So, the general form of a function with two arguments looks like this:

function-name (argument-1,argument-2)

Function-name is a keyword that identifies the function you want performed. The *arguments* are the items you want the operation performed on. Arguments can be variable names, constants, or expressions. An argument can even be the result of another function.

Some Samples:

```
BUCKS = INT (CASH);
```

This assignment statement calls the function INT and passes it a variable named CASH for the argument. Assuming the variable CASH is numeric, since INT requires a numeric argument, the integer value of CASH is passed back. If one value for the variable CASH is 10.66, when the statement is executed:

```
BUCKS = INT (10.66);  
turns into  
BUCKS = 10;
```

Similarly,

```
IF SUM(CASH,CREDIT) > 0 THEN PUT 'Doing OK!';
```

When this statement executes, the SUM function adds the values of CASH and CREDIT in the current observation:

```
IF SUM(10.66,2009.22) > 0  
becomes  
IF 2019.88 > 0
```

Or, you could pass a constant as an argument:

```
MYSTATE = ZIPSTATE ('78731');  
The result will be:  
MYSTATE = 'TX' ;
```

You can use an expression as an argument:

```
TOTAL = SUM(PRICE, PRICE * .08);
```

The expression: PRICE * .08

is evaluated first. Then the SUM function adds the two values and returns the sum.

And you can nest functions as arguments to other functions:

```
X = MAX (SUM (A, B, C), ABS (-20)) ;
```

SUM (A, B, C) and ABS (-20) return their values to MAX. Then MAX returns the maximum value of the two to the expression.

Valid arguments to be passed to a function depend upon what the function needs to work with. Some want numbers, some want character strings, some want SAS dates, and others want very special goodies. DIM(x) wants x to be an array since it's going to return the number of elements in the array. TAN(x) wants x to be numeric, specified in radians, and not an odd multiple of $\pi/2$.

If your function needs more than one argument, the arguments must be either separated by commas or preceded by the keyword OF. Usually, they are separated by commas but the OF syntax allows you to specify arrays and variable lists as arguments without having to code each variable.

```
X = SUM(x, y, z);    *<-----a SUM function with commas;
```

```
X = SUM(OF X--Z);    *<-----SUM function with variable list;
```

```
X = SUM(OF _NUMERIC_);    *<----SUM function with all defined  
                           numeric variables;
```

To find out what your function wants, check the "SAS Language Reference" manual. The functions are listed in Chapter 11, in alphabetical order, with samples of how to use them.

Functions can be grouped according to the type of work they do. The functions grouped by categories are listed starting on page 53 of the "SAS Language Reference". My favorites are in the Arithmetic, Character, Date and Time, Truncation, and Special Functions categories. Every now and then, I find a handy helper in the Financial, Sample Statistics, and ZIP code categories, too.

Arithmetic and Truncation functions all like their arguments to be numeric.

Arithmetic:

ABS	returns the absolute value.
MAX	returns the largest value.
MIN	returns the smallest value.
MOD	returns the remainder value.
SIGN	returns the sign of a value.
SQRT	returns the square root of a value.
SUM	returns the sum of the nonmissing arguments.

Truncation:

CEIL	returns the smallest integer greater than or equal to the argument
FLOOR	returns the largest integer less than or equal to the argument
INT	returns the integer value.
ROUND	rounds to the nearest round-off unit.
TRUNC	truncates a numeric value to a specified length.

Now you could probably do all this with program statements if you've got some time on your hands, but it could get pretty messy:

```
IF A > B then SMALL = B;  
  else SMALL = A;  
IF SMALL > C then SMALL = C;
```

is a little more tedious than

```
SMALL = MIN (A, B, C);
```

Some of the Character manipulation functions can save a lot of time and energy, too.

LEFT, RIGHT, TRIM and COMPRESS shift around the positioning of character strings.

```
NAME = ' Mc Donald ';
```

```
LNAME = LEFT (NAME);      ---> 'Mc Donald  '  
RNAME = RIGHT (NAME);     ---> '   Mc Donald'  
TNAME = TRIM (NAME);      ---> '  Mc Donald'
```

```
CNAME = COMPRESS (NAME); ---> 'McDonald'
```

COMPRESS removes blanks by default, but you can specify other characters to remove by indicating them in the second argument.

TRIM is handy for removing extra spaces at the end of a string before concatenating them with other variables:

```
data names;  
  input @1 FIRST $10. @11 LAST $10.;
```

```
  BLANKS = FIRST||LAST;  
  NOBLANKS = TRIM(FIRST)||LAST;  
  put BLANKS= / NOBLANKS= ;
```

```
cards;  
WILL  GEORGE  
;
```

```
BLANKS=WILL  GEORGE  
NOBLANKS=WILLGEORGE
```

REPEAT, REVERSE, SCAN, SUBSTR, TRANSLATE and UPCASE allow us to tinker with the contents of a string:

REPEAT(argument,n) Returns a character string consisting of the first argument repeated n times.

```
X = REPEAT('+----',5);  
PUT X;
```

Will give you:

+----+----+----+----+----+

REVERSE(argument) Returns a character string in reverse order.

SCAN and SUBSTR Allows you to select and/or replace pieces of a character string.

TRANSLATE and
UPCASE Substitutes some characters for others and translates lower case to upper case.

When using character functions, it always helps to remember that the length of the result may default to a larger size than you expect. COMPRESS will return a variable that's the same length as the first argument. You'll have to TRIM or use a LENGTH statement to overcome the default. Some character functions produce results with default lengths of 200. This can be a big surprise the first time you see it.

Some of my favorite functions are in the Date and Time group. The SAS strategy of storing dates as the number of days since January 1, 1960 has puzzled more than one beginning SAS user after feeding an unformatted SAS date to PROC PRINT. Fortunately, in addition to the many FORMATs and INFORMATs that SAS supplies us to work with SAS dates, there are also some good little functions.

DATEJUL and JULDATE convert Julian dates to SAS dates and back again.

DATE() and TODAY() return the current system date as a SAS date value.

There are about 17 other functions, such as DAY, HOUR, YEAR, WEEKDAY, and TIMEPART that extract selected pieces of SAS dates and times.

If you get some of your data through a SAS ACCESS engine from a database, the data will come to you in SAS date form. You can mince a system date or a database time stamp in a flash with these guys.

```
DATA DATES; SET KATHE.TRANSFILE(KEEP=STAMP);
```

```
    DATE=PUT DATEPART(STAMP),MMDDYY8.);  
    TIME=TIMEPART(STAMP);  
    QUARTER=QTR DATEPART(STAMP));  
    YEAR=YEAR DATEPART(STAMP);  
    MINUTE=MINUTE(STAMP);
```

My favorite two functions fall in the "Special" category. These are PUT and INPUT. The terrific thing about these functions is that they let you write or read a value using the whole supply of SAS formats and informats. The PUT function and the PUT statement are similar. The PUT function returns a value in the requested format for you to use as you please. The PUT statement writes the value to an external destination (the SAS Log, a PRINT file, an external file) in the format you specify.

Function (used in an expression):	X=PUT(Y,6.);
-----------------------------------	--------------

Statement:	PUT Y 6.;
------------	-----------

INPUT works in reverse. The INPUT function uses the value and INFORMAT you specify in the arguments to return a result to your statement. The INPUT statement uses the INFORMAT you supply in the statement to read the data you want from an external source.

Think about how powerful this is, especially since you can nest your functions:

```
INPUT @1 BIGNUM PD6.;
```

```
TEXTNUM = PUT(INPUT(BIGNUM,11.),$11.);
```

Since you can build your own FORMATS and INFORMATS using PROC FORMAT, you can use a combination of a special FORMAT and the PUT function to build and assign a classification variable instead of using lots of IF..THEN..ELSE logic to evaluate the data. It's simpler, shorter, more efficient, and easier to maintain!

```
PROC FORMAT;
  VALUE $MONEY
    'US'      = 'DOLLAR'
    'MEXICO'  = 'PESO'
    'JAPAN'   = 'YEN'
    etc. ;
DATA NEW;
  SET FINANCE;
  MONEY=PUT(COUNTRY,$MONEY.);
```

instead of

```
DATA NEW;
  SET FINANCE;
  IF COUNTRY = 'US' THEN
    MONEY = 'DOLLAR';
  ELSE IF COUNTRY = 'MEXICO' THEN
    MONEY = 'PESO';
  ELSE IF COUNTRY = 'JAPAN' THEN
    MONEY = 'YEN';
  etc.
```

If you get to know your functions, they can do a lot for you. Take a function to lunch. (LUNCH=TRANSLATE('BRUNCH',' L','BR');) Or maybe, take lunch to a function! (TIME=LENGTH('lunch');)