

Are You FUNCTIONally Literate?

**Allan M. Rominek
Centerior Energy
Perry, Ohio**

Abstract

This tutorial will consider a number of SAS® functions and the many ways that they can be used to make the life of your friendly neighborhood SAS programmer a little easier.

Introduction

Take a look at your SAS manuals. I don't mean look inside, I mean look at the outside. See where the wear and tear shows up? Those dog-eared pages. The ragged edges. I think the condition of your SAS manuals tells a lot about you as a SAS programmer or support person. I took a look the other day and if you discount the big dent from throwing it against the wall and the burn marks from the science experiment gone awry, there are several well worn sections in my Base SAS manual.

One of these sections is the chapter that covers functions. I'm always amazed at how many experienced users are not aware of all of the functions within the SAS System. Someone will launch into a fairly long winded explanation of what it is that they want to accomplish with some SAS code and I totally floor them by saying, "Oh, yeah, there's a function that can handle that." By no means will I cover all of the functions in this tutorial. Those covered will be those that I personally have found to be most useful.

You're Dating Yourself

Once I got married, I figured that I wouldn't have to worry about dates any more. Now it's 1996 and the year 2000 is looming over the horizon. Everybody is worried about dates. SAS as you all know handles dates very nicely, and there will be no problem with the year 2000, if you use the CUTOFF OPTION.

For handling less cataclysmic dates, there is a whole range of SAS functions. I'd like to discuss a few of these with you now.

One of the more common problems that I encounter in helping people with their SAS programs is that the files that are being read have dates stored in a "foreign" format, like packed decimal. The date has to be read using one of SAS's packed decimal informats, but what people often fail to realize is that SAS stores all numeric values using floating-point representation. Basically, the number is read just as it is stored. For example, if the date 060196, June 1, 1996, were stored as a packed decimal, SAS would store it internally as the floating-point representation for 60,196. As any programmer worth his weight in SAS manuals will tell you SAS stores dates internally as the number of days since January 1, 1960. Somehow, 60,196 has to be converted to the number of days from January 1, 1960 to June 1, 1996. This is where SAS functions come into play. The function to use is always determined by the output of the function. When working with dates, the desired output is always a numeric value. The numeric value 60,196 has to be

reformatted to yield another numeric value which is the now infamous number of days value. Two handy functions for formatting data are the PUT and INPUT functions. The result of the PUT function is always character. The result of the INPUT function is always numeric. How many votes for the INPUT function? There is a catch though, the original data that the INPUT function is applying the new format to has to be character. 60,196 is a numeric value. The solution to this minor dilemma is to first convert the numeric value to a character value and then apply a date format to the character value. Assuming that the date value is contained in a variable called PDDATE, for packed decimal date, the new variable SASDATE can be created as follows:

```
SASDATE=INPUT(PUT(PDDATE,Z6.),MMDDYY6.);
```

The Z6. format is used to ensure that leading zeroes are supplied for months up to and including September. The character representation of the date is really '060196'.

Once a date is stored as a SAS date, there are a number of functions that can be used to find out all kinds of neat stuff about the date.

The DAY function returns the day of the month.

```
MODAY=DAY(SASDATE);
```

Where MODAY can range from 1 to 31.

The MONTH function returns the month of the year.

```
MONOFYR=MONTH(SASDATE);
```

Where MONOFYR can range from 1 to 12.

The QTR function returns the quarter of the year in which the SAS date falls.

```
QUARTER=QTR(SASDATE);
```

Where QUARTER is either 1, 2, 3, or 4.

The WEEKDAY function returns the day of the week for the SAS date value.

```
WKDAY=WEEKDAY(SASDATE);
```

Where WKDAY is an integer where 1 = Sunday, 2 = Monday, ..., 7 = Saturday.

The YEAR function returns a four-digit numeric value representing the year from a SAS date value.

```
YR=YEAR(SASDATE);
```

There is also a whole series of functions that allow SAS dates to be created within a program.

The DATE and TODAY functions return today's date as a SAS date value.

```
NOW = DATE( );  
or  
NOW = TODAY( );
```

Note that there is no argument for either of these functions.

A date other than today's date can be constructed using another set of functions.

The MDY function can create a date based on a month, day, and year value that are supplied as arguments.

```
NOW = MDY(6 , 1, 96);
```

The YYQ function will return a SAS date corresponding to the first day of the specified quarter and year.

```
QTR3 = YYQ(96 , 3);
```

QTR3 will correspond to the SAS date for JULY 1, 1996.

For those Julian date aficionados, there are two functions that deal with Julian dates. The DATEJUL function takes Julian dates and converts them to the equivalent SAS date. The JULDATE function will take a SAS date and convert it to its Julian date equivalent.

There are also functions for dealing with times and datetimes that are of a similar nature to the date functions. The particulars of these functions can be reviewed in the SAS Language manual.

A Function for All Seasons

I've often received phone calls from frantic users who claim that they're trying to do the impossible. One example sticks in my mind. One of the planning functions within the company is done by specifying the week of the quarter. The problem that seemed so impossible was that the first week of the quarter began on the Sunday of the week in which the first day of the quarter fell. In 1996, the first day of the second quarter, April 1, fell on a Saturday. The first week of the quarter began on Sunday, March 26. The task is given a particular date to determine the quarter and week of that quarter into which that date falls. The quarter can be determined by using the QTR function, the WEEKDAY function can be used to determine the day of the week that the date and the Saturday of that week falls on. If the Saturday is in the next

quarter, the date is in the first week of the next quarter. The YYQ function can be used to determine the date of the first day of the quarter. The WEEKDAY function is then used to determine the day of the week. The date is then determined for the Sunday of that week by subtracting the day of the week less one from the date of the first day of the quarter. April 1 fell on a Saturday, the 7th day of the week according to the WEEKDAY function. The date for the Sunday beginning this week is determined by subtracting 6 (7-1) from the Saturday date. Now comes the tough part, determining the week in which the original date fell. There were a number of approaches that were tried. One of them involved the MOD function. The MOD function returns the remainder when one number is divided by another.

```
X = MOD( 65, 7);
```

When 65 is divided by 7 the remainder is 2. The number of days between the date in question and the Sunday of the first week of the quarter is divided by 7 and any decimal places are truncated using the INT function. The INT function returns the integer portion of the argument.

```
WEEK = INT( NUMDAYS/7 );
```

If the result of the MOD function is equal to 0 then WEEK is indeed the week of the quarter in which the date falls. If MOD is greater than 0, then WEEK + 1 is the correct week.

```
IF MOD ( NUMDAYS/7 ) > 0 THEN  
    WEEK = INT( NUMDAYS/7 ) + 1;  
ELSE WEEK = INT( NUMDAYS/7 );
```

Another solution was to use the CEIL function which returns the smallest integer

that is greater than or equal to the argument. In other words, it always rounds up.

```
WEEK = CEIL (NUMDAYS/7);
```

To finish up the numeric functions, there are three more functions that need to be mentioned. The FLOOR function which returns the greatest integer that is less than or equal to the argument.

The ROUND function rounds the argument to the nearest round off unit that is specified as an additional argument. To round a numeric variable, X, to the tens:

```
Y = Round (X , 10);
```

If a second argument is not specified, the function will round to the nearest units position.

The ABS function returns the absolute value of the argument.

```
Y = ABS ( X );
```

The most interesting characters that I've ever met.

Why do people get so hyper when it comes to manipulating character strings? It's a snap if you know your way around a few SAS functions.

The people that I have dealt with are often interested in doing word searches on lengthy text variables. For these I recommend the INDEX function. The INDEX function searches a character string for the presence of another character string and returns the position of the first character of the second character string.

```
POS = INDEX (TEXT, STRING );
```

POS will contain a numeric value. It will either be 0, indicating that STRING is not contained in TEXT, or some positive value indicating the position of the first character of STRING in TEXT.

Another interesting request that I had was to read data from two separate data bases and consolidate them in one SAS data set.

No big deal. Well, the one file contained free form text fields that were 35 characters long and the other data base contained free form text fields that were 70 characters long.

No big deal! Read the one as two 35 character fields, right? Well, the users of the first data base took great pains not to split words between lines and could I please do the same when I read the second file. Ah, therein lies the big deal. The big deal was overcome with a little thought and a lot of experimentation with SAS functions.

The SUBSTR function was used to determine if the 35th character of the string was a blank.

```
CHECK = SUBSTR ( TEXT, 35,1 );
```

The arguments of the SUBSTR function tell SAS to look a 1 character, starting in the 35th position of TEXT. If CHECK is equal to a blank, the string can be split into two strings of 35 characters without splitting a word.

```
TEXT1 = SUBSTR ( TEXT, 1, 35 );
```

```
TEXT2 = SUBSTR ( TEXT, 36, 35 );
```

If the 35th character is not a blank, a DO loop is used to step backwards from position 35 until a blank is discovered.

```
I = 35;  
DO UNTIL ( CHECK = ' ' );  
I = I - 1;
```

```
CHECK = SUBSTR ( TEXT, 1, 1 );
END;
TEXT1 = SUBSTR ( TEXT, 1, 1 );
```

The search for the next chunk of 35 characters begins at 1+35 and so on. Not such a big deal.

There are a few other character functions that have proved useful over the years. These include the following.

The SCAN function separates a character string and returns the nth word.

```
WORD = SCAN ( TEXT, n );
```

The way that SAS separates a character string into words is pretty much at the control of the user. As a default, SAS uses the following delimiters to separate words

```
blank . < ( + | & ! $ * ) ; → - / , % ¢
```

Other delimiters or combinations of the above delimiters can be specified by including a third argument to the function.

```
WORD = SCAN ( TEXT, 2, ' ' );
```

WORD will contain the second word within TEXT, where a word is defined as a character string separated from another character string by blanks.

The REVERSE function takes a text string and returns it in reverse order.

```
BACKWARD = REVERSE ( 'ABCDEF' );
```

BACKWARD will contain FEDCBA.

The TRIM function removes trailing blanks.

This can be very useful when creating new variables by concatenating two or more variables.

```
NAME = FIRST || ' ' || MI || ' ' || LAST;
```

FIRST and LAST can be of any length and concatenating them without first using the TRIM function could result in some peculiar looking names.

```
TFIRST = TRIM ( FIRST );
TLAST = TRIM ( LAST );
```

To obtain a more "normal looking" name, use the following:

```
NAME = TFIRST || ' ' || MI || ' ' || TLAST;
```

The LENGTH function returns the length of a text string, where length is the position of the rightmost nonblank character. I have often used the LENGTH function in conjunction with the INDEX and SUBSTR function. I have often wanted to locate a particular character or string of characters within a text string and extract everything to the right of that position. The INDEX function tells me the starting position of the string of characters in question. The LENGTH function tells me the entire length of the text string. By subtracting the starting position less one from the entire length, I know the length of the text that I want to extract or in other words the arguments of the SUBSTR function.

```
P = INDEX ( TEXT, STRING );
```

```
Q = P - 1;
```

```
L = LENGTH ( TEXT );
```

```
NEW = SUBSTR ( TEXT, P, L - Q );
```

The last functions that I wish to mention are the LEFT and RIGHT functions. These functions can be used to left and right justify text strings, respectively.

JLEFT = LEFT (TEXT);

JRIGHT = RIGHT (TEXT);

SAS is a registered trademark or trademark of SAS Institute Inc. in the USA and other countries.

® indicates USA registration

Conclusion

SAS functions are extremely useful and can be put to a number of uses to get even the most impossible sounding tasks done. For a complete listing and description of all the SAS functions see Chapter 11 of SAS Language: Reference.