Paper 268-2011

Build a Good Foundation with Functions

Deb Cassidy, PPD
Morrisville, NC

ABSTRACT

A major advantage of SAS® over some other languages is the vast number of built-in functions that make it very easy to do a task. SAS provides over 450 such functions. This presentation will include an overview of the structure of functions, some examples of commonly used functions and some pitfalls that often trip a new user. You'll also learn where to go to learn even more about the available functions and what they do. So, if you want to learn a quick way to sum fields that might have missing values, extract part of a text field or just get an idea of what else you can do with functions, then this presentation is for you.

INTRODUCTION

A SAS function is a built-in method to perform a computation or system manipulation and return a value. A CALL routine alters variable values or performs other system functions. CALL routines are similar to functions but differ in that you cannot use them in assignment statements. There are now over 500 functions and call routines in 29 categories. This paper will focus on a sample of functions to give you an overview of how to use them and what they can do. Some of the functions are probably more complex than a new SAS user might use but I wanted to show them because you just might inherent code written by a more experienced user.

The general format of a function is simply:

```
myvalue = FUNCTION_NAME (required_argument1,… required_argumentN, optional_argument1,…
optionalN);
```

However, the number of required and optional arguments varies with each function. Some functions simply have one required argument. Others have one required and one or more optional arguments. More complex functions may have several required arguments. In some cases, the order of the arguments is important. Some people may think that if the code runs without errors or warnings, then they did everything right but that isn't always the case. Some functions will give different results if you accidentally swap the order of the arguments.

If the argument you specify is a literal value, you will need to have quotes. If you specify a numeric value that needs to be treated as a literal, you will also need to have quotes. If you specify a variable name, you will not have the quotes. However, there are a few functions that contradict these rules.

CATEGORIES

The categories of functions were modified in SAS 9.1 and have been updated again in SAS 9.2. Enhancements to SAS 9.2 show 51 new functions, 45 enhanced functions and 7 functions that were replaced or removed.

Category	# of Functions	# of Call Routines
Arithmetical*	1	0
Array	3	
Bitwise Logical Operations	6	
Character**	86	6
Character String Matching**	5	6
Combinatorial*	12	10
Date and Time**	37	1
Descriptive statistics**	32	
Distance*	2	0

External Routines**	2	1
External Files*	36	
Financial**	32	
Hyperbolic**	6	
Macro**	5	3
Mathematical**	32	4
Numeric*	1	
Probability	18	
Quantile	7	
Random Number**	12	10
SAS File I/O**	31	
Search*	2	
Sort*	0	2
Special**	24	4
State and ZIP Code**	12	
Trigonometric**	7	
Truncation**	11	
Variable Control**		3
Variable Information	30	1
Web Tools	4	
TOTAL	**456**	**51**

* New category since SAS 9.1
** Additions/enhancements to category since SAS 9.1
Categories deleted since SAS 9.1: Application Response Measurement, Currency Conversion, DBCS (double-byte character set)

ARITHMETICAL
This category is new to SAS 9.2 and has one function. Many people have written a simple statement like

```
y=a/b;
```

only to discover lots of messages in the SAS log because there were 0's in the B variable and you can not divide by 0. For many years, the recommendation was to use:

```
if b ne 0 then y=a/b;
```

Now there is a function that provides even more information.

```
y=divide(a,b);
```

not only eliminates the log message but returns special missing values to indicate the result when it can not be calculated. For example, you can tell if the missing result was due to a 0 in the denominator or missing value in either variable. You can use this additional information to improve your subsequent processing as needed.

ARRAY
An array is a "group" of variables that can be referenced by a single name instead of each individual variable name. These functions tell you things about the elements and bounds of the array. The DIM function means you don't need to know how many elements are in the array. This is a benefit in writing flexible code. For example, in the pharmaceutical industry you might have an array that refers to all the variables starting with TRT. When you use the same code on another study with a different number of treatments, that part of the code won't change if you have used DIM(array_name) instead of a specific number of elements.

BITWISE LOGICAL OPERATIONS
These functions deal with the bits in each byte of data. Many programmers will never have a reason to use these functions but they are very beneficial for some.

CHARACTER
This is a category with many functions that a beginner will find useful. Of course, given the large number of functions, some are more complex and specialized.

Let's start with something simple. You have a field with names but the data entry person was a poor typist. The names look like:

JOE
John
pETE
RobeRt

You can make them all consistent with

```
first_name=UPCASE(first_name);
```

Perhaps you really want the first letter capital and the rest lower case. In that case, you want.

```
first_name=PROPCASE(first_name);
```

However, this is also a great example of why you need to check your results. Let's say you have the name O'Malley. PROPCASE will give you O'malley. But St. John does come out as expected. You may need to use IF conditions to check for certain types of values and apply different functions to them to get the desired results.

CATQ is a new function that lets you concatenate two or more variables. There are a variety of values for the argument to indicate if the values should be single-quoted or double-quoted, separated by a comma or period or any of several other combinations. For example

```
A='abc        ';
B=' defgh';

combined_1=('ac',a,b);
combined_2=('acs',a,b);
```

results in
```
combined_1:   "abc        "," defgh"           combined_2:   "abc","defgh"
```

One of the other issues that you should always check is the length of the newly created variable. In this example, combined_1 and combined_2 have a length of 200. Under more circumstances, you should use a length statement to specify a shorter length. The excess length can greatly increase the size of a dataset. You just need to make sure it is long enough to hold the longest possible result or else your result will be cut off. Not all functions default to a length of 200 for the new variable. For some functions, it defaults to the length of the original variable. Other functions follow other rules.

The two new variables created above differ due to the treatment of the trailing blanks. This is another issue that you need to watch when using character functions. In the above case, you needed to specify an argument to handle the leading/trailing blanks correctly. In other cases, you actually need to use a related function.

CHAR is another new function that does the same thing as a function that has been around for a long time but has one minor difference. In this example, you want to use the 12th character of the order_code for some additional processing. In the past, you would use code like

```
letter12=substr(order_code,12,1);
```

Now you can just use

```
letter12=char(order_code,12);
```

At first glance, you may not see why a new function was added. However, if you look at the variable information using PROC CONTENTS or other methods, you will see the new function creates a variable with a length of 1. The SUBSTR method created a variable with the same length as the original variable. As noted above, you would want to specify a length. The new function eliminates the need to do so.

TRANSLATE is a function that is an excellent example of why you need to ensure you have your arguments specified in the right order.

```
new_var1=translate(myvar,'$',' ');
new_var2=translate(myvar,' ','$');
```

Both statements will run without errors. However, which one really changes blanks to $ and which one changes $ to blank? I've been using SAS for over 25 years and I still get this one wrong more times than I like to admit. I just follow the principal of always spot-checking the results.

CHARACTER STRING MATCHING
These functions and call routines are for comparing strings. They are based on PERL regular expressions which are used in conjunction with many other languages. These are just one set of functions that have been added to provide functionality in a style similar to other languages.

COMBINATORIAL
These functions provide several ways to find permutations and combinations of values.

DATE AND TIME
There have been many papers presented over the years solely on this category of functions. A SAS date is the number of days since January 1, 1960. A SAS time is the number of seconds since midnight. A SAS datetime is the number of seconds since midnight on January 1, 1960. These are always numeric values.

Some people get confused between functions and formats when it comes to dates and times. Formats control the look of the value. For example, a format of mmddyy8. will print the SAS date 18722 as 04/05/11 while the format weekdate. writes out Tuesday, April 5, 2011. However, the value is still stored as 18722 and that value is used in many of the functions.

If your date comes from a non-SAS source or is stored as a character field rather than a SAS date, you will usually need to convert it to a SAS date before using it. How you do this will depend on the structure of your data. If you have separate numeric values for month, day and year, you simply need to use the MDY function so

```
sas_date=mdy(month,day,year);
```

However, if you have a character field you will need to look at how it is stored and consider several functions to determine which one really fits your situation.

How do you find the numbers of days between a start and end date? If you just need the number of days, you simply subtract the numeric values. You need to know if the case where the start date and end date are the same should be considered as a difference of 0 or 1. If it should be 1, you will need to add 1 to your result. If it is possible for the start date to be greater than the end date, you would not add 1.

The TODAY ( ) function can be used to write a flexible report where you want your number of days calculated based on the run date. For example, you might run a weekly report showing past due information and use

```
days_past_due=TODAY ( ) – invoice_date;
```

This is a function that does not need any arguments but it does need the ( ).

INTCK and INTNX are two useful date functions that are often confused. INTCK determines the number of boundaries crossed between the two dates while INTNX adjusts a value a specific amount. For example, if you need to know the number of months between Jan 15, 2011 and April 5, 2011, you would use

```
num_months=INTCK('months',first_date,last_date);
```

This would result in num_months=3 – February, March and April boundaries were crossed.

If you wanted to advance the date of Jan 15, 2011 by 3 months, you would use

```
date_plus_3=INTNX('month',first_date,3);
```

You might think this will result in April 15, 2011 but it really results in April 1, 2011. The default is the first day of the month. However, you have many options to get the result that is applicable for your needs. You could specify the additional argument to the middle of the month which would be April 15, 2011 or the end of the month which would be April 30, 2011. Notice the result accounts for the fact that there are only 30 days in April.

One of the additions to the DATE and TIME category was to include ISO 8601 compliance for functions like INTCK and INTNX.

Have you ever needed to account for holidays when determining something like due dates for an invoice? The new HOLIDAY function provides the dates of many major holidays.

```
memday=holiday('MEMORIAL',2020);
```

will tell you that Memorial Day will be celebrated in the US in 2020 on Monday, May 25 whereas this year it is on May 30. Not all holidays are included such as the one that changed the deadline for filing US taxes this year to be April 18 instead of the traditional April 15.

DESCRIPTIVE STATISTICS

Many of the descriptive statistic functions have the same name as values you can obtain in procedures such as MEAN or MIN. The difference is that procedures work across observations while functions work across variables for each observation. A major factor to consider with descriptive statistics is how you want missing values handled. For example,

```
total1=var1+ var2 +var3;
```

and

```
total2=sum(var1,var2,var3);
```

will give you the same results in many cases. However, if one of the variables has a missing value, then you will not get the same result. The SUM function is the sum of the non-missing values while the use of the plus sign will result in a missing value for the total.

DISTANCE

Distance functions are new functions that return the geodetic distance between two zip codes or two longitude/latitude points.

EXTERNAL ROUTINES

These functions can be used when working with things such as custom DLL's or in IML. They are not typically for beginners.

EXTERNAL FILES
There are a number of functions for working with external files. One that many people find useful is FILEEXIST. This lets you know if an external file already exists. You can write your code so a file that does exist does not get overwritten. You get a result of 1 if it exists or 0 if it does not. A simple example is:

```
check=fileexist('U:\SGF2011\Abstract.doc');
```

Note that the proper way to write the file name will vary by operating system.

A new function is RENAME. It allows you to rename several types of files including SAS files. The following example changes the name of a Word document.

```
data _null_;
  rc=rename('u:\sgf2011\DebCassidyAbstract.doc', 'u:\sgf2011\test.doc', 'file');
run;
```

RC is commonly used as the variable name for a return code. With this function you get a 0 if the rename was successful and another value if it was not.

FINANCIAL
There are several functions to do financial calculations such as depreciation or net present value. One that might be useful to everyone is MORT. If you bought a $200,000 house and wanted to pay for it over 30 years with an interest rate of 5%, you would pay $1,073.64;

```
A=200000;  * initial amount;
P=.;       * payment amount;
I=.05/12;  * monthly interest rate expressed as a fraction;
N=360;     * number of months;

payment=MORT(a,p,i,n);
```

This function is an example where all arguments must be specified so the field you want to calculate must exist as a variable with a missing value.

HYPERBOLIC
There are several functions for you to computer hyperbolic values such as hyperbolic sine, cosine and tangent.

MACRO
There are functions to create macro variables and to return values of macro variables. The have been many conference papers dealing with macros that cover these functions.

MATHEMATICAL
The are many mathematical functions ranging from common functions such as absolute value to lesser known calculations like the DAIRY function that calculates the derivative of the AIRY function.

It is easy to get a remainder by using

```
remainder=MOD(dividend_variable,divisor_variable);
```

It is very easy to accidentally type a / rather than a comma since you use the / when you write a division equation. However, you'll quickly discover the mistake because you'll get an ERROR when you try to run the function.

MODZ is a related function but with different adjustments for floating-point results. There are several functions that have a corresponding 'Z' function. The "Z" comes from the "fuzz" value. If your result will end up with several decimal places, you will want to investigate which functions you want to use. The results can also differ by operating system.

NUMERIC
This category has one function that was added in SAS 9.2 The functionality of IFN was already easy to do in a DATA step with multiple IF/THEN statements. Like some other additions over the years, it was added for programmers who have used it in other languages and requested the function to be available in SAS as well. It returns a value based on whether the result is true, false or missing.

```
donation=(income > 100000,income * .10, income* .05, 25);
```

The above code will calculate the recommended donation amount. The first argument, income > 100000, is evaluated. If that evaluation is true, then the second argument is the result placed in the variable. If the result is false, then the third argument is placed in the variable. In this example, if the income is missing, then the fourth argument of 25 is returned.

PROBABILITY
Several functions are available to return the probability from a variety of distributions.

QUANTILE
You can also return the quantile for several distributions.

RANDOM NUMBER
Did you think there was only one way to get a random number? There are actually many ways depending upon the distribution that you assume your data has. A commonly used function is RANUNI that returns a random variate from a uniform distribution.

```
random-RANUNI(seed);
```

But what should you use for your seed? Some people recommend using the computer time when the function executes. You just need to ensure your seed is an integer. It is recommended that you write your seed to your log if you use this method so you know what the seed was and can replicate the output if needed. Do you always get the same result? If you use the same seed, you will get the same result as long as you use a separate DATA step. If you create two variables in the same dataset with the same seed, you end up with two different values.

SAS FILE I/O
The FILEEXIST function was an example in the External Files category. There are similar functions for catalogs and data library members. CEXIST will check for a catalog or catalog entry while EXIST will check for a data library member. You need to understand the difference so you use the correct function. Examples include:

```
format_exists=CEXIST("work.formats");
data_exists=EXIST("work.my_datasets");
program_exists=CEXIST("mylib.mycat.somecode.program");
```

There are also functions that correspond to statements such as the LIBNAME and LIBREF functions. You will use the function to check if something exists or to return the value that has been set so you can use it later. You want to use the statement to set the value in the first place.

SEARCH
This is a new category with two functions. WHICHC looks for a character value in the second and subsequent arguments and returns where it is found. The second and subsequent arguments can be individual variables or an array. WHICHN does the same thing for numeric values.

Here's an easy example. Assume you have data for responders' top 5 choices and you want to know how many times people ranked choice 'c' first, second, etc.

```
array choices(*) choice1 – choice5;
ranked_c=whichc('c','of choices[*]);
```

Ranked_c will have a value of 1 for every observation with choice1='c'. Likewise, it will have a value of 2, 3,4 or 5 for every observation where choice2='c', choice3='c', choice4='c' or choice5='c', respectively. You would then do a frequency on RANKED_C to determine how many time people ranked 'c' in each choice.

SORT

This is also a new category and it has two call routines. CALL SORTC sorts the values of character variables and CALL SORTN sorts the values of numeric variables. It is required that all variables have be same length for CALL SORTC. Since this is a CALL routine, it does not return a value directly but the result of the CALL can be used in other statements. For example, you have several variables and you need the value that is first alphabetically for each observation. This example shows only one observation.

```
length var1-var3 $5;
var1='ay';
var2='aaa';
var3='br';

array vars var1-var3;
call sortc(of var1-var3);
put vars(*);
first_alpha=vars(1);
```

The put statement will write "aaa ay br" to the log since that is the sorted order. The variable first_alpha will be created with the value of the first position in the array after it has been sorted. This is a multi-step way to get the minimum value of character variables. There is a MIN function that returns the minimum value for numeric variables but one hasn't been added for character values yet.

SPECIAL

This category has all the functions that do not fit in any other category. They cover a wide variety of things such as putting the system to "sleep" or changing memory values. The latter is a function that should only be used by programmers who also have a good understanding of the operating system and how it handles memory. SLEEP is useful for "stopping" your SAS program while operating system actions are taking place. I use it often to force SAS to open a second interactive session by making the first session "busy" so it can not automatically open the second program in a different window. I want the sessions to be separate so datasets are not overwritten.

A special function that is used by many is the LAG function. However, it is a contender for the most misunderstood function. Many people think LAG is the way to get a value from the previous record. In many cases, that is how it appears to work. However, it is really returning a value from a queue and there are actions that can affect what values are in the queue. This is a function that has been covered in detail by many conference papers over the years.

STATE AND ZIP CODE

Can you quickly tell someone what state has the abbreviation AL? It's Alaska or is it Alabama? SAS provides several functions for changing your state and zip code values into the value you really need.

```
mystate1=STNAME("AL");
mystate2=STNAMEL("AL");
```

The first returns a value of ALABAMA while the second returns Alabama.

There is also a function to return the state name from a zip code. However, there are areas of the country where new zip codes have been added so you may receive an invalid argument note in your log.

TRIGONOMETRIC

This category is related to the HYPERBOLIC category and has functions for computing values such as sine and tangent.

TRUNCATION
The most common form of truncation is probably rounding. Most people learn rounding in our early school years so you would think this would be simple. If you have

```
x=ROUND(myvar,2);
```

and myvar=145.334, the result is 145.33, right? No, the result is 146. The code actually says to round to the "nearest 2" rather than the nearest 2 decimal places. To round to 2 decimal places, you specify

```
x=ROUND(myvar,.01);
```

What happens if your value is 145.345 and you round to 2 decimal places? Is the answer 145.35 or 145.34? It depends on what country and what industry rules you want to follow when you have a "5". SAS provides the ROUND function to give you 145.35 and the ROUNDE function to give you 145.34. ROUNDE may be referred to as the European method by some.

There are also many other function that will return integer values based on certain rules. INT returns the integer portion of your value. FLOOR returns the largest integer equal to or less than your value. CEIL returns the smallest integer equal to or greater than your value. If your value is 123.45, then INT and FLOOR both return 123 while CEIL returns 124. At first glance, you might wonder why you need both INT and FLOOR if they return the same value. If you check additional numbers, you'll discover this only holds true for positive numbers. If your value is -123.45 then INT and CEIL return -123 and FLOOR returns -124.

There are also related functions that use different fuzz values.

VARIABLE CONTROL
This category had 3 CALL routines that deal with assigning variable labels/names or linking variables. They are typically used in macros or array where you are creating or using variable names or labels based on other logic in your code.

VARIABLE INFORMATION
This is a large group of functions that return information about variables such as whether a given variable is character or numeric, whether it has a format assigned or if it had an informat. This allows you to write more flexible code. Some industries such as pharmaceuticals have a need to use the same code for different projects. However, the data may not be set up the same way for each project. For example, gender may have values of 1 and 2 in two projects but one project has it as a numeric variable and the other has it as character. Here is a very simple example for you to try that also uses a macro CALL routine.

```
data myexample;
 myvar=1;
 label myvar='This is an example';
run;

data _null_;
 set myexample;
 call symput("mylabel",vlabel(myvar));
 run;
 title "Report for: &mylabel";
 proc print;
 run;
```

WEB TOOLS
This last category provides functions for encoding or decoding strings for use in HTML. They are fairly easy to use if you have an understanding of HTML.

LEARNING MORE

There are many ways to learn more about functions.

- HELP provides details about the arguments and examples. How you access help depends on how your site is set up. You may just need to open an interactive session and select "HELP" from the menu. Other sites have a PDF version stored for access by everyone. Still other sites may prefer you go to support.sas.com. Once you are in HELP, you will want to go to the Base SAS section and look for the language reference dictionary. You should then be able to find the "Functions and CALL Routines" section. You'll then be able to look up a function alphabetically or by category.
- You can search past conference proceedings for SAS Global Forum (formerly SUGI) as well as regional/special interest conference proceedings. The most comprehensive search engine is maintained by SAS programmer Lex Jansen at www.lexjansen.com. As of this writing, his site searched 11,641 conference papers. A search for "functions" returned far more than 1,000 entries.
- You can also get help from your fellow SAS programmers. A couple examples in this paper were taken from questions asked on SAS-L, an electronic newsgroup. There are several ways to post questions and search the archives. Since there have been conference presentations on how to use SAS-L, I suggest you use Lex's site to do a search on SAS-L. You can also use the SAS Forums on support.sas.com. www.SASCommunity.org will also have information about functions in articles written by fellow programmers.

SUMMARY

There are many functions included with the SAS System that eliminate the need to write lengthy or complex code. Some are very easy and others are more complicated and may require operating system or industry knowledge. The key things to understand in using any function are:

-required versus optional arguments
-order of the arguments
-handling of missing values
-when you need to quote arguments
-whether arguments are case-sensitive
-differences between similar functions and statements
-defaults for the results (length defaults or value defaults)
-combining functions versus separate statements
-exceptions to anything that I've stated in this paper!

When you are learning a new function, you should test the function on a small sample of data to make sure you understand how it works before you use it with a large amount of data. You should have you sample include as many "odd" cases as possible. You should ALWAYS read the log to see if something unexpected happened. This is true even when you have lots of experience with a function because there may be unknown data issues. If you are an experienced user, you will also want to review functions in a new version of SAS because there may be a simpler way to accomplish the same task.

TRADEMARKS

CONTACT INFORMATION

You comments and questions are valued and encouraged.

Deb Cassidy
deborah.cassidy@ppdi.com