

Do We Know How to Function?

Janet E. Stuelpner, ASG, Inc., Cary, North Carolina

ABSTRACT

How do we function in SAS®? There are many functions that can make programming easy and efficient. This paper will take a small section of the functions that are provided and explore their uses, benefits and pitfalls. There are examples of regular code as compared to a more efficient use with functions. There are also instances where a function may not be the best choice. These will be discussed as well.

INTRODUCTION

There are many different components of the SAS system. The function is one of them. A function is a built-in routine that performs common operations on variables and/or constants. They cover a wide-range of tasks. Their purpose is to make coding easier and faster. These routines return a value from a computation or data manipulation. Usually, the user will supply the arguments for the function. However, there are functions that will obtain the value of the argument from the host operating system. We begin by defining the function in general and the various categories in which functions fall. We will then explore several basic functions in detail. These are functions that will be of use to every programmer in data manipulation and reporting.

SYNTAX

A function must be part of a SAS statement. We usually see functions in the DATA step, but they can also be used in some statistical procedures. They can be used anywhere a SAS expression is used. They have two parts; a function name and zero or more arguments. The name of the function tells you what task you want to accomplish: add some numbers, calculate a payment for a mortgage, or find the integer from a floating-point number. The second part is the argument. These specify what variables you want to manipulate. The argument can be any valid SAS expression (variable names, constants or expressions) and must be enclosed in parenthesis. Even if there are no arguments for a function, the parentheses must be there. In other words, the function name is followed by any number of arguments enclosed in parenthesis. For example, the syntax for a general function with one argument is: *function-name(argument)*

When you use a function, you are said to be calling that function. The value that is manipulated and/or computed is said to be returned.

Function arguments must be preceded by the word OF or separated by a comma. The general form of functions is:

```
function-name(argument1,argument2)
function-name(OF argument1 argument2)
function-name(OF argument1-argument-n)
function-name(argument-x -- argument-z)
```

Note the double minus in the last expression. This is not an error. There are many ways to write a variable list. Of course, you can list every variable. However, there are some shortcuts. Abbreviated variable lists are useful after you have defined all of the variables in your DATA step. There are three kinds of abbreviated lists. The first is a numbered range. You must have the variables with the same name except for the last character or characters that are consecutive numbers: var1, var2, var3, var4, var5. The shortcut to write this is var1-var5. The next is the name range. This relies on the position of the variable in the program data vector. It will include all variables in the order that they appear in the data vector: firstvar--lastvar. The last is special SAS names. These include `_NUMERIC_` to include all numeric variables, `_CHARACTER_` to include all character variables and `_ALL_` to include all variables defined in the DATA step.

Let's look at some examples of the different ways to call a function. The easiest way to use a function is demonstrated here. In each case, the result of the function call is the placement of a value into a variable.

```
AGE=ROUND(AGE,.01);
GRADE=PUT(SCORE,LETTER.);
SALARY=SUM(OF WEEK1-WEEK52);
LOANAMT=MORT(.,1500,8/12,30*12);
TODAY=DATE();
STATE=ZIPNAME('06840');
```

There are other more complicated ways to use functions. The argument of a function can be a function call. In some cases, the return value of a function call can be used in conditional statements. However, this is beyond the focus of this paper.

RULES AND REGULATIONS

There are some guidelines to which you must adhere when working with functions. There are certain restrictions with some functions and order requirements for others. Some functions require all of the parameters to be present while others allow for them to be missing. Some functions, like the logarithmic functions, require that the arguments are within a certain range. In the LOG function, for example, the argument must be greater than zero. Other functions require that the arguments are numeric. For example, the ROUND function requires both arguments to be numeric. There are functions in which the order of the arguments is critical. Any deviations produce a syntax error. The last rule involves missing values. Most functions do not permit missing values as arguments. Functions which calculate descriptive statistics such as sums and standard deviations, use only non-missing numeric data. The one important thing to note is that you don't always get an error when you pass inappropriate values for function arguments. Be aware of the result you expect when using functions.

A question often arises as to why there is a SUM function when there is a PROC SUMMARY or a MEAN function when there is a PROC MEANS. There is a difference between using a PROC and using a function. The purpose of a function is to work within an observation. The average that is a result of the MEAN function is an average of the value of several variables in one observation. A PROC, on the other hand, works across many observations. The average that is calculated for a PROC MEANS is the sum of all of the values of a variable for many observations divided by the number of observations. In other words, if you think of a SAS data set as columns (number of variables) and rows (number of observations). The function will SUM across the row and a procedure will SUM down a column. For example, look at the data set EXAM defined as three variables each containing the score on an exam. Below it are the results of the MEAN function versus a PROC MEANS.

	TEST1	TEST2	TEST3
OBS 1	95	94	96
OBS 2	80	82	84
OBS 3	88	84	78
OBS 4	75	85	95

```
DATA AVG;
  SET EXAM;
  AVERAGE=MEAN(OF TEST1-TEST3);
RUN;
```

```
PROC PRINT DATA=AVG NOOBS;
RUN;
```

TEST1	TEST2	TEST3	AVERAGE
95	94	96	95

80	82	84	82
88	84	74	82
75	85	95	85

As you can see from the output above, the result of using the MEAN function is an average for each student taking all three tests. Their final grade may be based on this average. However, the situation is very different when using the MEANS procedure (as can be seen in the results below).

```
PROC MEANS DATA=EXAM;
OUTPUT MEAN=ATEST1-ATEST3;
RUN;
```

Variable	N	Mean	Std Dev	Minimum	Maximum
TEST1	4	84.5	8.8128694	75.0	95.0
TEST2	4	86.25	5.3150729	82.0	94.0
TEST3	4	88.25	8.7321246	78.0	96.0

The output from the PROC MEANS shows an average for each exam that was taken for all of the students. The scores for the students taking the first exam are added and then divided by the number of people taking the test to get the mean. The resulting average may be used to see how the whole class is doing. An instructor can use this information to curve the scores to equalize the scores.

FUNCTION CATEGORIES

There are many categories in which to divide functions. No matter how the functions are grouped, there are some functions that fit into several different categories. The categories below are the ones used by SAS Institute, Inc. The functions that are listed are just a sample of some of the functions in that category. For a more complete listing of the all of the functions in each category, see the SAS Language manual.

Category	Functions
Arithmetic	abs, max,min,mod,sqrt,sign
Array	dim,hbound,lbound
Character	compress,index,scan,substr
Date & Time	date,intck,mdy,qtr,timepart
Financial	compound,mort,saving
Mathematical	log,ordinal
Probability	poisson,probchi,probnorm
Quantile	cinva,finv,probit
Random Number	normal,rannor,ranuni,uniform
Sample Statistic	max,min,mean,n,std,sum
Special Function	dif,input,lag,put,symget
State & Zip Code	Sname,zipname,zipstate
Trig & Hyperbolic	Cos,sin,tan
Truncation	ceil,floor,int,round,trunc

BASIC FUNCTIONS

Below are a sample of functions that are extremely useful in everyday programming. They are divided into categories the categories above. Along with the definitions and syntax are some examples. There are many more functions. These are some of the more basic ones.

Arithmetic

ABS(argument) is the absolute value function. It returns a non-negative number equal in magnitude to that of the argument. The argument is always numeric.

Statement	Value
a=abs(25);	25
b=abs(-25);	25
c=abs(25.5);	25.5
d=abs(-25.5);	25.5

MIN(argument,argument,argument...) is the minimum function. It returns the smallest value of all of the arguments. At least 2 arguments are needed. All arguments must be numeric. This function is very useful when you need to find the lowest value.

Statement	Value
e=min(95,85,99,75,84);	75
f=min(-2,0,-1,1,2);	-2

MAX(argument,argument,argument...) is the maximum function. It returns the largest value of all of the arguments. At least 2 arguments are needed and they must be numeric. This is useful when you need the highest value.

Statement	Value
g=max(95,85,99,75,84);	99
h=max(-2,0,-1,1,2);	2

MOD(argument-1,argument-2) is the modulus function. It returns the remainder when the integer quotient of argument-1 divided by argument-2 is calculated. Both arguments are numeric and argument-2 cannot be zero. In other words, this function finds the remainder when the first argument is divided by the second. This is used most often when you need to find a certain repeating observation.

Statement	Value
i=mod(5,3);	2
j=mod(10,3);	1

In this example, the user needs to delete every 4th record. When the result of the function call is zero, the record is deleted. If there are 4 records in data set Y, then the last record is deleted.

Statement	Value of _N_	Value
-----------	--------------	-------

MOD(_N_,4)	1	1
MOD(_N_,4)	2	2
MOD(_N_,4)	3	3
MOD(_N_,4)	4	0

```
DATA X;
  SET Y;
  IF MOD(_N_,4)=0 THEN DELETE;
RUN;
```

Character

LEFT(argument) The left function returns an argument with leading blanks moved to the end of the value. The SAS character expression is left aligned and its length does not change. The argument is character.

Statement	LEFT(K)
k='test'	'test'
k=' test'	'test'
k=' test'	'test'

RIGHT(argument) The right function returns an argument with trailing blanks moved to the beginning of the value. The SAS character expression is right aligned and its length does not change. The argument is character.

Statement	RIGHT(L)
L='test'	' test'
L=' test'	' test'
L=' test'	' test'

COMPRESS(source,<characters-to-remove>) The compress function removes blanks or characters you specify from a character expression you also specify and returns the result. The source is the character string from which the characters will be removed. The second argument contains the characters that will be forced out. You do not need to specify the second argument. If you do not specify anything for the second argument, all of the blanks will be removed. In the example below, we have a data set called PRES that has two variables, FIRST, the first name (8 characters in length) and LAST, the last name (10 characters in length) of several presidents.

FIRST	LAST
Bill	Clinton
Ronald	Reagan
Lyndon	Johnson
George	Bush

```
DATA COMP;
  SET PRES;
  LENGTH NAME NAMEC $35
  NAME=LAST||' '||FIRST;
  NAMEC=COMPRESS(LAST||' '||FIRST);
RUN;
```

The results of running the above code with the presidential data is below. Notice that without the compress function, all of the first names are spaced away from the last name. With the compress, the blanks at the end of the last name are removed.

NAME	NAMEC
Clinton ,Bill	Clinton,Bill
Reagan ,Ronald	Reagan,Ronald
Johnson ,Lyndon	Johnson,Lyndon
Bush ,George	Bush,George

TRIM(argument) The trim function copies a character argument, removing any trailing blanks, and returns the trimmed argument as a result. This can be useful when concatenating because concatenation does not remove trailing blanks. The argument must be a character string. We will use the same example above. You can see how much easier it is to concatenate variables with this function.

FIRST	LAST	TRIM(FIRST) ' ' LAST
Bill	Clinton	Bill Clinton
Ronald	Reagan	Ronald Reagan
Lyndon	Johnson	Lyndon Johnson
George	Bush	George Bush

SUBSTR(argument,position<,n>) is a function which can extract all or part of a string. It performs two distinct operations, depending on where you use it. On the right side of an assignment statement, it returns the portion of the expression you specify in the argument. It begins from the position designated in the function call for the number of characters defined by n. If n is not specified, it returns from the position to the end of the value of the variable. On the left side of an assignment statement, the SAS system places the value of the expression on the right side into the argument, replacing n characters starting with the character you specify in position. Let's look at some examples. In the first example, the substr is on the right side of an assignment statement. In this case, the data set contains a character variable with a date in the form ddmmyy. We need to extract the month. We know that the month will always start in the third position and is three bytes in length.

EXPRESSION	DATE	VALUE
MONTH=SUBSTR(	01JAN60	JAN
DATE,3,3);	04JUL76	JUL
	05OCT9	OCT
	7	

When the substr function appears on the left side of an assignment statement it is sometime called a substring pseudo-variable. When a string needs to be replaced, this is a good way to do it. In this

example, a database has the city listed as a character variable of length 30. The data was entered incorrectly for one city. Instead of entering NEW YORK CITY, it was entered as NEW YORK TOWN. There are many ways in which this can be corrected. The following example shows how the change can be made using the substr function.

```
DATA CITY;
    SET TOWN;
    IF CITY='NEW YORK' THEN DO;
        SUBSTR(CITY,10,4)='CITY';
    END;
RUN;
```

CITY - before	CITY - after
NEW YORK TOWN	NEW YORK CITY

UPCASE(argument) copies a character argument, converting all lowercase characters to uppercase characters and returns the altered value as a result. The argument must be a character string. This function is useful when trying to find or compare character strings that may be entered in lowercase or mixed upper and lower case.

NAME	UPCASE(NAME)
Abe	ABE
Bob	BOB
Clancy	CLANCY
David	DAVID

Date and Time

TODAY() is a function that produces the current date as a SAS date value representing the number of days since January 1, 1960 and the current date.

DATE	TODAY()
July 14, 1997	13708

MDY(month,day,year) is the function that produces a SAS date value from numeric expressions that represent the month, day and year. All of the arguments for this function must be numeric. Month specifies an integer from 1 through 12. Day an integer from 1 through 31. Year an integer identifying a specific year. The **YEARCUTOFF=** system option will define the year range. Many times the data is input as three separate fields. This function puts them all together into a SAS date value which can then be more easily manipulated and reported using SAS formats.

MONTH	DAY	YEAR	DATE	SAS date
1	1	1960	Jan. 1, 1960	0
7	4	1976	Jul. 4, 1976	6029
10	5	1997	Oct. 5, 1997	13792

HMS(hour,minute,second) is a function that produces a SAS time value from hour, minute and second values. All of the arguments for this function

must be numeric. Hour an integer from 0 thorough 23. Minute is an integer from 0 through 59. Second is an integer from 0 through 59. As in the MDY function, this function can put three separate input fields together into a SAS time value.

HOUR	MINUTE	SECOND	TIME	SAS time
0	0	0	midnight	0
12	0	0	noon	43200
5	30	0	wakeup	19800
17	30	0	dinner	63000

MONTH(date)/DAY(date)/YEAR(date) are functions that return a numeric value representing that part of the date. Date is a SAS expression representing a SAS date value that is the number of days since January 1, 1960.

VDATE	DAY	YEAR	Value
Jul 4, 1976	6029	MONTH(VDATE)	7
Jul 4, 1976	6029	DAY(VDATE)	4
Oct 5, 1997	13792	MONTH(VDATE)	10
Oct 5, 1997	13792	YEAR(VDATE)	1997

Financial

MORT(a,p,r,n) is a function which returns the missing argument in the list of four arguments from an amortization calculation with a fixed interest rate compounded each period. One missing argument must be provided and that is calculated from the other three. All of the arguments are numeric. A is the initial amount to be borrowed. P is the amount of each periodic payment. R is the periodic interest rate expressed as a fraction. N is an integer, the number of compounding periods and must be greater than zero. Suppose you need to know how much the payment each month will be on a 30 year fixed mortgage at an interest rate of 8% for \$100,000.

```
PAYMENT=MORT(100000,...,08/12,30*12);
```

After doing this function, you find out that the payment every month will be \$766.77.

Statistical

SUM(argument, argument,) is a function that returns the sum of the non missing arguments. All arguments must be numeric. This is a very special function. Let's take a moment and look at it carefully.

Statement	Value
m=sum(1,2,3,4);	10
m=sum(2,4,8,16,32);	62

How is this different from an assignment statement that adds the values of the variables? When the variables contain missing values, the assignment statement acts in a different manner from the SUM function. When three variables are added together and the value of one is missing, the value assigned is missing. If the SUM function is used to add these

three variables together, the result is the sum of the non missing values. Suppose a group of people always go shopping together. Not everyone buys something each trip. We need to know how much was spent. Let's use the data in data set SHOP below to illustrate our point.

PERSON	PRICE1	PRICE2	PRICE3
John	5	15	45
Jane	10	.	20
Tom	5	20	.
Sally	10	20	30

```
DATA BUY;
  SET SHOP;
  SPENT=PRICE1 + PRICE2 + PRICE3;
  SUMBUY=SUM(PRICE1-PRICE3);
RUN;
PROC PRINT DATA=BUY;
  VAR PERSON SPENT SUMBUY;
RUN;
```

PERSON	SPENT	SUMBUY
John	65	65
Jane	.	30
Tom	.	25
Sally	60	60

MEAN(argument, argument,) is the function that returns the average of the non-missing arguments. All arguments must be numeric.

Statement	Value
m=mean(1,2,3,4);	2.5
m=mean(2,4,8,16,32);	12.4

Special functions

PUT(source,<?|??> format) enables you to write the value of source with a specified format. The format must be the same type as the value of source. The result of a PUT function is always a character variable. The optional question mark(s) format modifiers suppress the printing of the error messages and the input lines when invalid data values are read. One of the biggest uses of this function is to convert numeric values to character values. Let's look at an example. We are going to take a student's average and convert this into a grade. The first thing that must be done is to create a format.

```
PROC FORMAT;
  VALUE GRADE 90-100 = 'A'
               80-89  = 'B'
               70-79  = 'C'
               65-69  = 'D'
```

```

;
LOW-64   = 'F'
RUN;

```

The data that we will use is the output from a previous example. We used the mean function to calculate each student's average after three exams.

NAME	TEST1	TEST2	TEST3	AVERAGE
ABBY	95	94	96	95
BART	80	82	84	82
JOHN	88	84	74	82
JANE	75	85	95	85

Next the average is converted to a grade using the GRADE format defined above. Then the results are printed so that the instructor can post the grades.

```

DATA GRADES;
  SET EXAM;
  FINAL=PUT(AVERAGE,GRADE.);
RUN;

```

```

PROC PRINT; RUN;

```

NAME	AVERAGE	GRADE
ABBY	94	A
BART	82	B
JOHN	84	B
JANE	85	A

The PUT function and the PUT statement are very similar. The PUT function can be used to store a value in a specified format if it is used in an assignment statement. The PUT statement is used to write a value in a specified format to an external destination.

Truncation

CEIL(argument) is a function which returns the smallest integer that is greater than or equal to the argument. In other words, non-integers are rounded up. There is one argument that must be numeric.

Statement	Value
n=ceil(5);	5
o=ceil(7.3);	8
p=ceil(7.6);	8
q=ceil(-3);	-3
r=ceil(-9.2);	-9
s=ceil(-9.8);	-9

FLOOR(argument) is a function which returns the largest integer that is less than or equal to the argument. In other words, non-integers are rounded down. There is one argument that must be numeric.

Statement	Value
t=floor(5);	5
u=floor(7.3);	7
v=floor(7.6);	7

w=floor(-3);	-3
x=floor(-9.2);	-10
y=floor(-9.8);	-10

INT(argument) is a function that returns the integer portion of the argument. It truncates the decimal portion. If the argument is positive, it has the same effect as the FLOOR function. If the value is negative, it has the same effect as the CEIL function. There is one argument that must be numeric.

Statement	Value
a=int(5);	5
b=int(7.3);	7
c=int(7.6);	7
d=int(-3);	-3
e=int(-9.2);	-9
f=int(-9.8);	-9

ROUND(argument,round-off unit) is a function that returns a value rounded to the nearest round-off unit. Only the first argument is required. If the second argument is missing, a default value of 1 is used and the argument is rounded to the nearest integer. Both arguments must be numeric and the round-off unit must be a positive number. The first set of examples shows the result when the default value of 1 is used.

Statement	Value
g=round(5);	5
h=round(7.3);	7
i=round(7.6);	8
j=round(-9.2);	-9
k=round(-9.8);	-10

The next set of examples shows the effect of the round function with a second argument defined.

Statement	Value
l=round(2222,10);	2220
m=round(2222,100);	2200
n=round(2222,1000);	2000
o=round(15.125,.1);	15.1
p=round(15.125,.01);	15.13
q=round(15.125,.001);	15.125

CONCLUSION

Many functions have been explored. As we have seen, there are many different categories of functions. There is some overlapping with the arithmetic and statistical functions. Overall, functions make coding easier because a great deal of work is done in the background. If used effectively, they enhance programs and make them more efficient.

REFERENCES

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

Dilorio, Frank C.(1991), *SAS Applications Programming: A Gentle Introduction*, Duxbury Press

Aster, Rick and Seidman,Rhena(1991), *Professional SAS Programming Secrets*, Windcrest/McGraw-Hill

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

ACKNOWLEDGMENTS

I would like to take this opportunity to acknowledge all of the help and support given to me by my husband, Bob Stuelpner. He diligently read and reread this paper in an effort to correct the obvious errors and keep me on the right track. His criticisms were constructive and his support never ending.

Author Contact:

Janet Stuelpner
ASG, Inc.
326 Old Norwalk Road
New Canaan, CT 06840
(203) 966-7520 (voice)
(203) 966-8027 (fax)
jstuelpner@att.net