Fully "Function"-al: A Brief Tour of Selected SAS® Numeric Functions

Brenda Beaty, University of Colorado Denver, Denver, CO

ABSTRACT

From the almost 450 functions available in SAS 9.2, I take you on a brief tour of functions I have found extremely useful, limiting myself to non-character functions.

INTRODUCTION

Base SAS offers users literally hundreds of functions (448 at last count, and there were 40 new functions added in SAS 9.2 alone!), as well as the ability to create user-defined functions. But what is a SAS function? A SAS function performs an operation or manipulation on user-supplied arguments, and returns a value. In other words, a function is an "operator" you apply to your data, which gives a useful result. Functions can be used in DATA step programming statements, in a WHERE expression, in macro language statements, in PROC REPORT and in SQL.

TYPES OF FUNCTIONS

SAS lists the following as function types: Arithmetic, Array, Bitwise Logical Operations, Character String Matching, Character, Combinatorial, Date and Time, Descriptive Statistics, Distance, External Files, External Routines, Financial, Hyperbolic, Macro, Mathematical, Numeric, Probability, Quantile, Random Number, SAS File I/O, Search, Special, State and Zip Code, Trigonometric, Truncation, Variable Information, and Web Tools.

This paper will be cover selected functions from the Date and Time, Descriptive Statistics, Mathematical, Random Number, Truncation, and State and Zip Code categories of functions.

SYNTAX OF FUNCTIONS

The most basic syntax of a function is as follows:

function-name (argument 1<,argument n>)

function-name (OF variable list)

Examples:

```
x=mean(a,b,c);
y=mean(of d1-d5);
```

DATE AND TIME FUNCTIONS

Recall that SAS stores date values as the number of days since January 1, 1960. This makes it easy to manipulate date values, but occasionally difficult to interpret them. For example, SAS stores (and by default displays the value representing June 1, 2008 as 17684). In order to make it more understandable, I usually format date values in the mmddyy8. format. There are many other formats that can be applied to date and time values (worddate., for example).

MDY FUNCTION:

"Translates" a date in familiar format to a SAS date.

Syntax:
mdy (month, day, year)
where *month* specifies a numeric expression that represents an integer from 1-12,
day specifies a numeric expression that represents an integer from 1 through 31, and
year specifies a numeric expression that represents an integer identifying a specific year. You can use either a two-digit or four-digit year. If you use a two-digit year, you can use the YEARCUTOFF= system option to define the year range, if necessary.

Example:

```
m=8;
d=27;
y=90;
birthday=mdy(m,d,y);
```

```
put birthday= worddate.;
```

Result:
```
birthday=August 27, 1990
```

DATE FUNCTION (AKA TODAY FUNCTION):
These functions return the current date, and are equivalent functions. They have only parentheses as their argument.

Syntax:
today()
date()

Example:
```
DaysSinceJun1= today() - mdy(6,1,2008);
```

DATEPART FUNCTION:
Extracts the date part from a date/time value. I have used this often when importing data from Microsoft Access, as the default format is DATETIME. However, in SAS 9.2, you can now use the USEDATE=YES option to change the format to DATE.

Syntax:
newdate = datepart (olddatetime);

YRDIF FUNCTION:
Returns the difference in years between two dates. This is an alternate way to calculate age in years, rather than using arithmetic.

Syntax:
yrdif (startdate, enddate, basis)
'basis' is an argument that determines how SAS will calculate the difference. The choices are 'ACT/ACT', '30/360', 'ACT/365', and 'ACT/360.' 'ACT/ACT' will probably be the best choice in most circumstances, as it calculates the actual number of days between the dates, and takes leap years into account.

Example:
In the following example, YRDIF returns the difference in years between two dates based on each of the options for basis.
```
data d;
   sdate='16oct1998'd;
   edate='16feb2003'd;
   y30360=yrdif(sdate, edate, '30/360');
   yactact=yrdif(sdate, edate, 'ACT/ACT');
   yact360=yrdif(sdate, edate, 'ACT/360');
   yact365=yrdif(sdate, edate, 'ACT/365');
run;
```

Results:
```
y30360=4.333333333
yactact=4.3369863014
yact360=4.4
yact365=4.3397260274
```

DESCRIPTIVE STATISTICS FUNCTIONS
Another important category of functions are those that calculate simple statistics. These are useful for almost all levels of analysis.

SUM FUNCTION:
Returns the sum of all non-missing values of the arguments. The arguments can be numeric constants, variables, or expressions. If all the arguments have missing values, the result is a missing value.

Syntax:
sum (argument1, argument2, <argument3…..>)
sum (OF argument-list)
where '*argument'* specifies a numeric constant, variable, or expression. At least two arguments are required. The argument list can consist of a variable list, which is preceded by OF.

The *sum* function return a missing value (.) only if all arguments are missing.
Note that the *sum* function does NOT 'propagate' missing values, as using a simple arithmetic operator would:

```
x1=1;
x2=2;
x3=.;
y1=(x1 + x2 + x3);
result: y1=.
y2=sum(x1, x2, x3);
result: y2=3
```

Examples:
Note that in the examples below, the missing value makes no difference in the result.

```
x1=sum(4,9,3,8);
result: x1 = 24

x2=sum(4,9,3,8,.);
result: x1=24
```

Note the use of 'of' before the list of variables below.

```
x1=9;
x2=39;
x3=sum(of x1-x2);
result: x3=48

x1=5; x2=6;
y1=3; y2=1; y3=7;
z=sum(of x1-x2, of y1-y3);
result: z=28
```

Careful, this one could be tricky!

```
x1=7;
x2=7;
x3=sum(x1-x2);
result: x3=0
```

(Because there is no 'of' before the arguments, SAS will subtract 7 from 7 and get 0!)

MIN, MAX, MEAN, MEDIAN, STD, STDERR FUNCTIONS:
These descriptive statistics functions all 'operate' in the same way, so I've combined them for brevity.

Syntax:
Using 'min' as an example; for other functions, simply substitute *function-name* for 'min.'
min (argument1, argument2, <argument3,…..>)
min (OF argument list)
where '*argument'* specifies a numeric constant, variable, or expression. At least two arguments are required. The argument list can consist of a variable list, which is preceded by OF.

These functions return a missing value (.) only if all arguments are missing.

Example:
```
x=min(.,1);
```
Result:
```
x=1
```

Example:
```
y=min(x,2,8);
```
Result:
```
y=1
```

NMISS FUNCTION:
Returns the number of missing numeric values.

Syntax:
nmiss (argument1 <, argument2,….>.)
nmiss (OF argument list)
where '*argument'* specifies a numeric constant, variable, or expression. At least one argument is required.

Example:
I have found this useful when constructing scales from survey data to determine if enough elements of a scale are present to allow construction of the scale variable, as shown below:
```
* 5 ALLOWED MISSING;
if nmiss(of x1-x10)<=5 then scale=mean(of x1-x10);
  else scale=.;
```

N FUNCTION:
Returns the number of non-missing numeric values.

Syntax:
n (argument1 <, argument2,….>.)
n (OF argument list)
where '*argument'* specifies a numeric constant, variable, or expression. At least one argument is required.

Example:
Similar to example for nmiss, you could use the n function to determine whether enough elements of a scale are non-missing:
```
if n (of x1-x10)>5 then scale=mean (of x1-x10);
  else scale=. ;
```

CMISS FUNCTION:
Counts the number of missing arguments. The *cmiss* function is similar to *nmiss*, but can also evaluate character values.

Syntax:
cmiss (argument1 <,argument2,…>)
Here *argument1* can be either a character or numeric value. A character expression is counted as missing if it evaluates to a string that contains all blanks or has a length of zero. At least one argument is required.

MISSING FUNCTION:
Returns a numeric result that indicates whether the argument contains a missing value. If the argument does not contain a missing value, SAS returns a value of 0. If the argument contains a missing value, SAS returns a value of 1. Note that unlike *nmiss* and *cmiss*, *missing* can only have one argument.

Syntax:
missing (numeric-expression | *character-expression)*

The *expression* can be either character or numeric, and can be a constant, variable or expression. A character expression is counted as missing if it evaluates to a string that contains all blanks for has a length of zero. At least one argument is required.

Example: uses the MISSING function to check whether the input variables contain missing values.

```
data values;
input @1 var1 3. @5 var2 3.;
if missing(var1) then do;
  put 'Variable 1 is Missing.';
end;
else if missing(var2) then do;
  put 'Variable 2 is Missing.';
end;
datalines;
127
988 195
;
run;
```

SAS writes the following output to the log:

```
Variable 2 is Missing.
```

MATHEMATICAL FUNCTIONS

ABS FUNCTION:

Returns the absolute value of a numeric argument.

Syntax:
abs (*argument*)

Example:

```
x=abs(-3);
```

Result:

```
x=3
```

Example:

```
x=abs(2.1);
```

Result:

```
x=2.1
```

EXP FUNCTION:

Returns the value of the exponential function. The *exp* function raises the constant *e*, which is approximately 2.718, to the power that is supplied by the argument.

Syntax:
exp (*argument*)

Example:

```
x=exp(1);
```

Result:

```
x=2.718
```

LOG FUNCTION:

Returns the natural (base e) logarithm. The argument must evaluate to be a positive number.

Syntax:
log (*argument*)

Example:

```
x=log(1);
```

Result:

```
x=0
```

MOD FUNCTION:

Returns the remainder value.

Syntax:
mod (*argument1,argument2*)
where *argument1* is numeric, and *argument2* is numeric and cannot be 0.

This function can be used to select a (not necessarily random!) sample from a data set. The following example selects every 4th observation from a larger data set.

Example:

```
data sample;
  set big;
  if mod (_n_,4)=0;
run;
```

Note that _n_ is an automatic variable is created by every DATA step. _n_ is initially set to 1. Each time the DATA step loops past the DATA statement, the variable _n_ increments by 1. The value of _n_ represents the number of times the DATA step has iterated.

RANDOM NUMBER FUNCTIONS

RANUNI FUNCTION (AKA UNIFORM FUNCTION):

Returns a random number from a uniform distribution, given a starting (seed) value. The seed must be a numeric constant, variable, or expression with an integer value less than 2,147,483,647. If you use a positive seed, you can always replicate the stream of random numbers by using the same DATA step. If a seed is not given or if you use 0 as the seed, the computer clock initializes the stream and the random numbers cannot be replicated!!

Syntax:
ranuni (*seed*)
uniform (*seed*)

Example: An approximate 10% sub-sample can be generated using the following code:

```
data subset ;
  set big ;
  where ranuni ( 12745892 ) le .10 ;
run ;
```

You can also use the *rannor* / *normal* functions to generate a random number from a normal distribution.

For more exact sub-sample selections, check out PROC SURVEYSELECT.

TRUNCATION FUNCTIONS

CEIL, FLOOR, AND INT FUNCTIONS:

These three functions are somewhat related and have the same syntax, so I've grouped them together.

int function: Returns the integer portion of the argument (truncates the decimal portion).
floor function: Returns the largest integer that is less than or equal to the argument
ceil function: Returns the smallest integer that is greater than or equal to the argument

Syntax:
int (*argument*)

floor (*argument*)
ceil (*argument*)
where '*argument'* is numeric.

If the value of *argument* is positive, INT(*argument*) has the same result as FLOOR(*argument*). If the value of *argument* is negative, INT(*argument*) has the same result as CEIL(*argument*).

ROUND FUNCTION:
Rounds the first argument to the nearest multiple of the second argument or to the nearest integer when the second argument is omitted.

Syntax:
round (*argument <,rounding-unit>)*
where *argument* is a numeric constant, variable or expression to be rounded, and *rounding-unit* is a positive, numeric constant, variable or expression that specifies the rounding unit.

Example:

```
data r;
  x1=round(2.5);
  x2=round(2.5, 0.1);
  x3=round(2.55, 0.1);
  x4=round(2.55936, 0.01);
run;

proc print data=r; run;
```

Result:

```
 Obs    x1     x2     x3     x4
  1      3    2.5    2.6    2.56
```

STATE AND ZIP CODE FUNCTIONS

STFIPS, STNAME, AND STNAMEL FUNCTIONS:
All three of these functions take the same argument (a two-letter state postal code) but return different values. STFIPS returns a numeric U.S. Federal Information Processing Standards (FIPS) code. STNAME returns an uppercase state name. STNAMEL returns a mixed case state name.

Syntax:
stfips ('*state postal code*')
stname ('*state postal code*')
stnamel ('*state postal code*')

Examples:

```
data s;
  x1=stfips ('CO');
  x2=stname ('co');
  x3=stnamel ('Co');
run;

proc print data=s; run;
```

Result:

```
Obs    x1       x2          x3
 1      8    COLORADO    Colorado
```

Note that these functions are case-insensitive in their arguments. Also, the functions *fipstate, fipname,* and

fipnamel perform the reciprocal functions; that is, transform a fips code into a state name output in various ways.

ZIPCITY, ZIPNAME, ZIPNAMEL, AND ZIPSTATE FUNCTIONS:
These four functions take the same argument (a five-digit zip code) but return different values:
ZIPCITY returns the uppercase name of the city and the two-character state postal code.
ZIPNAME returns the uppercase name of the state or U.S. territory.
ZIPNAMEL returns the mixed case name of the state or U.S. territory.
ZIPSTATE returns the uppercase two-character state postal code.

Syntax:
zipcity (*five-digit zip code*)
Other functions in this section have the same syntax.

Example:

```
data z;
  x1=zipcity (80045);
  x2=zipname (80045);
  x3=zipnamel (80045);
  x4=zipstate (80045);
run;

proc print data=z; run;
```

Result:

Obs	x1	x2	x3	x4
1	Aurora, CO	COLORADO	Colorado	CO

CONCLUSION
If there is something you want to do to a data value in SAS, most likely there's a function to help you do it!

REFERENCES
Cody, Ron. 2004. *SAS® Functions by Example.* Cary, NC: SAS Institute Inc.
SAS Institute. SAS OnlineDoc: http://support.sas.com/documentation/index.html.

RECOMMENDED READING
Cody, Ron. 2004. *SAS® Functions by Example.* Cary, NC: SAS Institute Inc.

CONTACT INFORMATION
Your comments and questions are valued and encouraged. Contact the author at:

Brenda Beaty, MSPH
Colorado Health Outcomes Program
University of Colorado Denver
Mail Stop F443
PO Box 6508
Aurora, CO 80045-0508
Work Phone: (303) 724-1076
Fax: (303) 724-1839
E-mail: Brenda.Beaty@ucdenver.edu