Functioning With Data Step Functions

Arthur L. Carpenter
California Occidental Consultants

ABSTRACT

Functions provide access to powerful tools and routines that make our everyday programming life easier. However since there are so many functions, some programmers are unaware of potentially valuable tools. To make matters worse many functions have been added to the SAS System since the Version 6 manuals were first published, and consequently many of these functions are under utilized. Additionally most of the functions that were originally only available in Screen Control Language are now, with the latest releases of Version 6, available in the DATA step.

Selected functions from the categories of new, under utilized, neglected, and formerly SCL are discussed through the use of examples.

KEYWORDS

Functions, DATA step, P-222, SCL, Screen Control Language

INTRODUCTION

SAS provides a large number of pre-written subroutines for common operations. These functions return a value that can be a part of an expression or an argument to another function.

Chapter 3 of the SAS Language Reference Manual lists those functions that were a part of the base SAS System at the release of V6.06. Since that time numerous functions have been added. In addition a number of functions have behaviors that do not seem to be intuitive.

The DATA step can now utilize many of the functions that were originally introduced in Screen Control Language. New macro functions have also been added as has the ability to access DATA step functions through the macro language.

Functions have always been a powerful tool of the base SAS System, now they are even stronger and more useful.

Features of SAS functions include:

- returns a value from a computation or manipulation
- requires zero or more arguments
- function name is always followed by parentheses
- a function is used as part of a SAS expression

The syntax for function calls can be expressed as:

`functionname(`*`argumentlist`*`)`

The number of function arguments and how they are separated varies from

function to function. In general function arguments are:

- separated by commas, unless a list of variables
- preceded with the OF operator for lists of variables
- positional for some functions

FUNCTION BASICS

The SUM function sums the numeric arguments. The arguments are separated by commas. Notice that an expression can be a function argument.

```
total =
sum(jancost,5,febcost-febadj);
```

The OF operator can be used to eliminate the need for commas.

```
total =
sum(of jancost 5 febcost-febadj);
```

Variable name shorthand can be used when the OF operator is used, and the list of variables have a common prefix and a numeric suffix.

```
total = sum(of month1-month12);
```

Function arguments can include calls to other functions. Below MAXTOT will contain the maximum of either the budget or the average of the twelve monthly values.

```
maxtot =
max(mean(of month1-month12),
       budget);
```

CHARACTER FUNCTIONS

The following example converts names that are of the form "LASTNAME, FIRSTNAME" into separate variables for the first and last name. Three different ways of determining the first name are shown. The variable NAME in the data set NAMES contains values such as:

```
Jones, Clint
Doe, Jane
Adams, Scott

* Split the name into first
* and last names;
data flnames;
set names;

len = length(name);
col = index(name,',');
lname = substr(name,1,col-1);
fname =
  substr(name,col+1,len-col);
fname1 = substr(name,col+2);
fname2 =
  substr(name, col+2,len-col-1);
run;
```

The index function has been used to 'find' the comma and note its location in the variable COL. This information is used in the SUBSTR function to extract the first and last names.

NUMERIC FUNCTIONS

In the following example a single observation from a weight gain study is shown. The PDV for the data set WTGAIN is:

DOB	AGE	FNAME	LNAME	WT1	WT2	SEX
04jun75	18	Clint	Jones	159	170	M

The following DATA step and will create the values that are in the PROC PRINT listing shown below:

```
data funct;
set wtgain;

* statistics functions;
maxwt = max(wt1, wt2);
meanwt = mean(of wt1 wt2);

*date functions;
today = date();
day = day(dob);
year = year(dob);
run;
```

Use of selected functions

	F	L			M	M		T		
	N	N			A	E		D		Y
D	A	A	W	W	X	N		D	D	E
O	M	N	T	T	W	W		O	A	A
B	E	E	1	2	T	T		B	Y	R
5633	Clint	Jones	159	170	170	164.5	06/04/75	12452	4	1975

UNDER UTILIZED FUNCTIONS

Because there have been almost continuous updates to the software without updates to the primary written documentation, a number of functions are not fully utilized. Also some functions are less easily understood or their purpose is not immediately obvious, and they are therefore not as often incorporated into programs. This section contains some functions that are useful, but fall into this category.

DATE FUNCTIONS

Most date functions are well known, however the following two seem to be somewhat neglected.

➤INTCK

Returns the number of time intervals in a given time span. The FROM and TO values are adjusted to the start of the stated INTERVAL. This means that '22mar1999'd will be seen as '01mar1999'd if the interval is 'MONTH' and as '01jan1999'd if the interval is 'YEAR'.

SYNTAX

```
intck('interval', from, to)
```

Where interval can be day, month, week, or qtr.

EXAMPLE

```
data a;
dob = '05apr78'd;
today = today();
days  = intck('day' ,dob,today);
weeks = intck('week',dob,today);
years = intck('year',dob,today);
run;

proc print data=a;
title 'Using the INTCK Function';
run;
```

Using the INTCK Function

OBS	DOB	TODAY	DAYS	WEEKS	YEARS
1	6669	12676	6007	858	16

It may be necessary to calculate age in years given two dates. Assume that today's date is stored in TODAY and the date of birth is stored in DOB.

```
age1 = (today-dob)/365.25;

age2 =
int(intck('month',dob,today)/12);
if month(dob) = month(today) then
      age2 = age2 -
      (day(dob)>day(today));
```

AGE1 tends to be the more mathematically accurate, but does not represent how we normally express age *i.e.* the age is not advanced until the day of the birthday and fractional years are not used. For AGE2 INTCK is used to calculate complete months which is used to determine full years. An adjustment is then made if the birthday has transpired

for the current year.

➤INTNX

Advances a date, time or datetime value by a given number of intervals.

SYNTAX

```
intnx('interval', from, number)
```

EXAMPLE

```
data a;
cdate = '12dec1995'd;
* Determine the start and endpoints for
the
* previous month;
mbeg = intnx('month', cdate, -1);
mend = intnx('month', cdate, 0) -1;
* Date 3 months in the future;
mon3 = intnx('month', cdate, 3)+
day(cdate) - 1;
format cdate mbeg mend mon3 date9.;
run;

proc print data=a;
title 'Using the INTNX Function';
title2
'Function returns the interval start';
run;
```

```
Using the INTNX Function
Function returns the interval start

   CDATE      MBEG       MEND      MON3

12DEC1995 01NOV1995 30NOV1995 12MAR1996
```

CHARACTER FUNCTIONS

These functions manipulate and work with character strings.

➤COMPBL

Removes multiple blanks between words in a character string.

SYNTAX

```
compbl(string)
```

EXAMPLE

```
city = 'Los      Angeles';
short = compbl(city);
```

The variable SHORT takes on the value of 'Los Angeles'.

➤INDEXC

Searches a character string for any of a series of characters contained in one or more excerpts. The column number of the first character (from any in the excerpts) is returned.

SYNTAX

```
indexc(string,excerpt1,excerpt2,
...)
```

EXAMPLE

```
data a;
namelist =
'Billie-Bob aBob Boba Bob';
dashcol =
indexc(namelist,'12','.,-','o');
bcol    =
indexc(namelist,'12','.,-','B');
put dashcol= bcol=;
run;
```

The LOG shows:

```
DASHCOL=7 BCOL=1
```

➤INDEXW

Searches a character string for a sub-string pattern that begins and ends on a word boundary. P-222 the original documentation for this function only specified that it searched for strings 'starting' on word boundaries.

SYNTAX

```
indexw(string,excerpt)
```

EXAMPLE

```
data a;
namelist =
'Billie-Bob aBob Boba Bob';
namecol =
indexw(namelist,'Bob');
put _all_;
run;
```

The numeric variable NAMECOL will have a value of 22.

➤GETOPTION

Retrieves current option settings. These can be used to reconstruct the current settings if your program might alter option settings during execution.

SYNTAX

```
getoption(optionname,<keyform>)
```

The first argument is the selected option, which can include graphics options (goptions). If the KEYFORM argument is included and the option is normally set using an equal sign (*e.g.* it is not a on/off type option), the returned value will include the option name followed by an equal sign.

EXAMPLE

```
data _null_;
value   = getoption('ps');
keyform =
  getoption('ps','keyword');
put 'value is ' value;
put 'using keyword 'keyform;
run;
```

The LOG shows:

```
value is 54
using keyword PS=54
```

➤LOWCASE

Converts a character string to all lower case characters. This is the functional opposite of the UPCASE function.

SYNTAX

```
lowcase(string)
```

EXAMPLE

```
x = 'ABCD1234';
low = lowcase(x);
```

The character variable LOW will have a value of 'abcd1234'.

➤TRIM

TRIM is used to remove trailing blanks from a character string.

SYNTAX

```
trim(string)
```

EXAMPLE

```
part1 = 'Los     ';
part2 = 'Angeles ';
untrimd = part1 || part2;
trimmed =
  trim(part1)||' '||part2;
```

UNTRIMD is 'Los Angeles '
TRIMMED is 'Los Angeles '.

➤TRANSLATE

Replaces specific characters in a text string with other characters.

SYNTAX

```
translate(string, to_list,
from_list)
```

EXAMPLE

```
letters = 'ABCDEFG';
numbers =
translate(letters,'137','ACG');
```

The variable NUMBERS will contain '1B3DEF7'.

➤TRANWRD

Replaces a set of characters in a text string with another set of characters.

SYNTAX

```
tranwrd(string, target,new)
```

EXAMPLE

```
name = 'Mrs. Johnson';
name =
   tranwrd(name,'Mrs', 'Ms');
```

The variable NAME will be:
'Ms. Johnson'.

➤SOUNDEX

Encodes a character string to a numeric value to facilitate character searches. The first letter is maintained, vowels are dropped, double letters are compressed, and the remaining letters are assigned numeric values.

SYNTAX

```
soundex(string)
```

EXAMPLE

```
* find all variations of the
* last name JOHNSON;
if soundex(name) =
soundex('Johnson');
```

The value of `soundex('Johnson')` is `'J525'` and each of the following variations would be selected:

```
'Johnnson'
'Jonson'
'Johnsen'.
```

➤SPEDIS

Another function that can be used to compare strings with dissimilar spelling is the spelling distance function (SPEDIS). This function calculates a numeric 'cost' that effectively allows you to determine, using a relative measure, how similar two strings are in spelling.

SYNTAX

```
spedis('query','keyword')
```

EXAMPLE

```
data _null_;
dist0 =
  spedis('Johnson','Johnson');
dist1 =
  spedis('Johnnson','Johnson');
dist2 =
  spedis('Jonson','Johnson');
dist3 =
  spedis('Johnsen','Johnson');
dist4 = spedis('Fred','Johnson');
put dist0= dist1= dist2=;
put dist3= dist4=;
run;
```

The LOG shows:

```
DIST0=0 DIST1=6 DIST2=8
DIST3=14 DIST4=162
```

NUMERIC FUNCTIONS

Numeric functions operate on numeric values and return numeric results.

➤ROUND, CEIL, FLOOR, INT

These functions are used to adjust the value of a number.

SYNTAX

```
round(number,nearest_value)
ceil(value)
floor(value)
int(value)
```

EXAMPLE

```
data a;
x=-6.63; output;
x=-2.28; output;
x= 6.63; output;
x= 2.28; output;
run;

data b; set a;
rnd = round(x,.2);
int = int(x);
ceil = ceil(x);
floor = floor(x);
run;

proc print data=b;
title 'Changing a Value';
run;
```

The PROC PRINT produces the following table.

Changing a Value

OBS	X	RND	INT	CEIL	FLOOR
1	-6.63	-6.6	-6	-6	-7
2	-2.28	-2.2	-2	-2	-3
3	6.63	6.6	6	7	6

USING WHAT WERE SCL FUNCTIONS

Starting with V6.11 many of what had been exclusively SCL functions, became available in the DATA step. These functions are now indistinguishable from the traditional DATA step functions, except that they are documented in the SCL Reference Manual (Second Edition) Chapter 20, and are summarized in Chapter 19 pp. 201-220.

➤LIBNAME

The LIBNAME function allows you to conditionally create a libref in a DATA step.

SYNTAX

```
libname('libref','path',
            <engine>,<options>);
```

EXAMPLE

This example clears a libref prior to reestablishing it.

```
data _null_;
* Clear the libref JUNK prior
* to reestablishing it;
sysrc = libname('junk','');
* Create the libref JUNK;
sysrc =
libname('junk','d:\junk');
run;
```

➤SYSMSG

The SYSMSG function will return a text description of the results of the previous function. It returns a blank if the last result was successful.

SYNTAX

```
sysmsg()
```

A LIBREF example which follows, includes the use of the SYSMSG function.

➤LIBREF

This function returns a 0 if the specified libref has been defined.

SYNTAX

```
libref('libref')
```

EXAMPLE

```
data _null_;
x = libref('sasuser');
put x=;
if libref('dbmsout') then msg=
sysmsg();
put msg=;
run;
```

The LOG shows:

```
X=0
MSG=ERROR: Libname DBMSOUT is
not assigned.
```

➤PATHNAME

The PATHNAME function returns the physical path associated with an established libref.

SYNTAX

```
pathname(libref)
```

EXAMPLE

This DATA _NULL_ step is used to determine the path associated with the libref MYDATA. This path is then used to create a new libref with a different engine.

```
data _null_;
* libref for location of new
* file;
aa = pathname("mydata");

* If the DBMSOUT libref exists
* then clear it;
if libref('dbmsout') then
libname('dbmsout','');
* Create a libref for the stated
* Engine;
sysrc =
libname('dbmsout',aa,'v604');
```

```
run;
```

A more complete example of this and the following function can be found in Carpenter, 1998.

➢EXIST

The EXIST function can be used to determine if a data set exists or if it is known to the system.

SYNTAX

exist(*datasetname*)

EXAMPLE

This DATA _NULL_ step is used to determine if a data set exists and sets a macro variable for use later.

```
data _null_;
* Specify the data set name;
dsn = 'sasuser.feeder';
if exist(dsn) then
   call symput('exist','Y');
else call symput('exist','N');
run;
%put &exist;
```

When the specified data set exists, the macro variable &EXIST takes on the value of 'Y'.

SUMMARY

DATA step functions provide powerful tools for use by the SAS programmer. Because there is such a wide variety of functions and because so many have been added since the last major update to the documentation, it is not unusual for some of these functions to be under utilized. SAS programmers must be constantly alert to new functions and updates to old ones.

Just to make things more interesting for those who are trying to keep up with the advances, a large number of functions were recently added to the DATA step. Many of the functions that were originally implemented in Screen Control Language have now been made available in the DATA step.

ABOUT THE AUTHOR

Art Carpenter's publications list includes two chapters in *Reporting from the Field*, the two books *Quick Results with SAS/GRAPH® Software*, and *Carpenter's Complete Guide to the SAS® Macro Language*, and over three dozen papers and posters presented at SUGI, PharmaSUG, and WUSS. Art has been using SAS since 1976 and has served as a steering committee chairperson of both the Southern California SAS User's Group, SoCalSUG, and the San Diego SAS Users Group, SANDS; a conference cochair of the Western Users of SAS Software regional conference, WUSS; and Section Chair at the SAS User's Group International conference, SUGI.

Art is a SAS Quality Partner™ and through California Occidental Consultants he teaches SAS courses and provides contract SAS programming support nationwide.

Art Carpenter
California Occidental Consultants
P.O. Box 6199
Oceanside, CA 92058-6199

(760) 945-0613
art@caloxy.com
www.caloxy.com

REFERENCES

Carpenter, Arthur L., *Carpenter's Complete Guide to the SAS® Macro Language*, Cary, NC: SAS Institute Inc., 1998, 242 pp.

SAS Institute Inc., SAS® Technical Report P-222, *Changes and Enhancements to the Base SAS® Software, Release 6.07*, Cary, NC: SAS Institute Inc., 1991, 344 pp.

TRADEMARK INFORMATION

SAS and SAS Quality Partner are registered trademarks of SAS Institute, Inc. in the USA and other countries. ® indicates USA registration.