

A Survey of Some of the Most Useful SAS® Functions

Ron Cody, Ed.D.

SAS Functions

- There are so many useful SAS functions it is very difficult to choose which ones to discuss
- Several of the functions in this talk are relatively new and amazing. They can save enormous amounts of time
- This talk covers some of my favorite functions

LENGTHN and LENGTHC

Lengthn(*Char_value*) returns the length of a string, not counting trailing blanks. Note that if *Char_value* is a missing value, Lengthn returns a 0, the old Length function returns a 1 in this case.

Lengthc(*Char_value*) returns the storage length of a string.

Character Storage Lengths

```
data chars1;  
  length String $ 7;  
  String = 'abc';  
  Storage_length = lengthc(String);  
  Length = lengthn(String);  
  Display = ":" || String || ":";  
  put Storage_length= /  
      Length= /  
      Display=;  
run;
```

SAS Log

```
11 data chars1;
12     length String $ 7;
13     String = 'abc';
14     Storage_length = lengthc(String);
15     Length = lengthn(String);
16     Display = ":" || String || ":";
17     put Storage_length= /
18         Length= /
19         Display=;
20 run;
```

Storage_length=7

Length=3

Display=:abc :

Moving the LENGTH Statement

```
data chars2;  
    String = 'abc';  
    length String $ 7;  
    Storage_length = lengthc(String);  
    Length = lengthn(String);  
    Display = ":" || String || ":";  
    put Storage_length= /  
        Length= /  
        Display=;  
run;
```

SAS Log

```
1 data chars2;  
2     String = 'abc';  
3     length String $ 7;
```

WARNING: Length of character variable string has already been set. Use the LENGTH statement as the very first statement in the DATA STEP to declare the length of a character variable.

```
4     Storage_length = lengthc(String);  
5     Length = lengthn(String);  
6     Display = ":" || String || ":";  
7     put Storage_length= /  
8         Length= /  
9         Display=;  
10 run;
```

Storage_length=3

Length=3

Display=:abc:

MISSING

Missing(*character or numeric value*)

- Returns a value of “true” if the value is missing, false otherwise

***Old way;**

if Age = . then . . .

If Char = ' ' then . . .

***New way;**

if missing(Age) then . . .

if missing(Char) then . . .

CALL MISSING Routine

Sets all of its arguments to missing
(character or numeric)

Old way

```
array x[10] x1-x10;  
array chars[5] a b c;  
do i = 1 to 10;  
    x[i] = .;  
end;  
do i = 1 to 3;  
    chars[i] = ' ';  
end;  
drop i;
```

Using call

```
call missing(of x1-x10,a,b,c);
```

INPUT

Input(*Char_value*, *informat*)

- "Reads" *Char_value* using the supplied *informat*
- Often used for character-to-numeric conversion

INPUT Example

```
data _null_;  
  c_date = "9/15/2004";  
  c_num = "123";  
  Sas_Date = input(c_date,mmddy10.);  
  Number = input(c_num,10.);  
  put SAS_Date= Number=;  
run;
```

SAS_Date = 16329
Number = 123

PUT

Put(*Value,format*)

- Formats *Value* using the *format* specified as the second argument. The result is a character value.
- Often used for numeric to character conversion. Also useful to create new variables using a user-written format

PUT Example

```
data _null_;  
    SAS_Date = 1;  
    Number = 1234;  
    SS_num = 123456789;  
    Char_Date = put(SAS_Date, mddy10.);  
    Money = put(Number, dollar8.2);  
    SS_char = put(ss, ssn.);  
    put Char_date= Money= SS_char=;  
run;
```

```
Char_Date = "1/2/1960"  
Money = "$1,234.00"  
SS_Char = "123-45-6789"
```

Another PUT Example

```
proc format;  
    value agegrp 0-20='0 to 20'  
                21-40='21 to 40'  
                41-high='41+' ;  
  
run;  
data PutEx;  
    input Age @@;  
    AgeGroup = put(Age, agegrp.);  
datalines;  
15 25 60  
;
```

Age	AgeGroup
15	0 to 20
25	21 to 40
60	41+

FIND and FINDC

find or findc(*String*, *Substring*, *modifiers*, *start*)

- Find searches for a substring in a string
- Findc searches for individual characters
- Both functions return a 0 if the search is unsuccessful.
- Modifiers and starting positions are optional and may be placed in any order.
- Replaces the older INDEX and INDEXC functions

FIND and FINDC Example

```
data locate;  
  input String $10.;  
  First = find(String, 'xyz', 'i');  
  First_c = findc(String, 'xyz', 'i');  
  /* i means ignore case */
```

```
datalines;  
abczyx1xyz  
1234567890  
abcz1y2x39  
XYZabcxyz  
;
```

String	First	First_c
abczyx1xyz	8	4
1234567890	0	0
abcx1y2z39	0	4
XYZabcxyz	1	1

COMPRESS

`compress(string, 'list', 'modifiers')`

- If only one argument is used, blanks are removed from *string*.
- *List* (optional) is a list of characters to remove from *string* (unless a 'k' modifier is used.)
- *Modifiers* (optional) allow you to remove classes of characters such as all digits, or letters (see list on next slide)
- The 'k' modifier says to keep all the characters you specify and remove all the rest (very useful)

COMPRESS Function Modifiers

Selected list of COMPRESS modifiers (upper- or lowercase)

- a upper- and lowercase letters
- d numerals (digits)
- i ignores case
- k keeps listed characters instead of removing them
- s space (blank, tabs, lf, cr) to the list
- p punctuation

COMPRESS Examples

For these examples, `char = "A C123XYZ"`,
`phone = "(908) 777-1234"`

Function	Value Returned
<code>COMPRESS ("A C XYZ")</code>	<code>ACXYZ</code>
<code>COMPRESS (phone, " (-) ")</code>	<code>9087771234</code>
<code>COMPRESS (CHAR, "0123456789")</code>	<code>A CXYZ</code>
<code>COMPRESS (CHAR, , "ds")</code>	<code>ACXYZ</code>
<code>COMPRESS (CHAR, "12345", "k")</code>	<code>123</code>
<code>COMPRESS (PHONE, , "kd")</code>	<code>9087771234</code>

COMPRESS Example

```
data phone;  
    input Phone $15.;  
    Phone1 = compress(Phone);  
    Phone2 = compress(Phone, '(-)');  
    Phone3 = compress(Phone, , 'kd');  
datalines;  
(908)235-4490  
(201) 555-77 99  
;
```

Phone	Phone1	Phone2	Phone3
(908)235-4490	(908)235-4490	9082354490	9082354490
(201) 555-77 99	(201)555-7799	2015557799	2015557799

Another COMPRESS Example

```
data Units;  
  input @1 Wt $10.;  
  Wt_Lbs =  
    input(compress(Wt, , 'kd'), 8.);  
  if findc(Wt, 'K', 'i') then  
    Wt_Lbs = 2.2*Wt_Lbs;  
datalines;  
155lbs  
90Kgs.  
;
```

Listing of Data Set Units

Wt	Wt_Lbs
155lbs	155
90Kgs.	198

SUBSTR

`substr(string,start,length)`

- Takes a substring from *string*, starting a position specified by *start* for a length specified by *length*. If *length* is not specified, the substring ends at the last non-blank character in *string*.
- If not specified prior to using the function, the length of the resulting variable will be the same as the length of *string*.

SUBSTR Example

```
data pieces_parts;  
  input Id $9.;  
  length State $ 2;  
  State = substr(Id,3,2);  
  Num = input(substr(Id,5),4.);
```

```
datalines;
```

```
XYNY123
```

```
XYNJ1234
```

```
;
```

Listing of Data Set PIECES_PARTS

Id	State	Num
XYNY123	NY	123
XYNJ1234	NJ	1234

SUBSTR Function on the Left-Hand Side of the Equal Sign

```
data bank;  
  input Id Account : $9. @@;  
  Account2 = Account;  
  substr(Account2,1,5) = '*****';  
datalines;  
001 123456789 002 049384756 003 119384757  
;
```

Id	Account2
1	*****6789
2	*****4756
3	*****4757

SCAN

Scan(String, which word, delimiters)

- The Scan function returns the value of the n'th "word" in a string. Negative values mean scan from right to left.
- There is a long list of default delimiters. We recommend you supply your own.
- If a length has not been defined for the result, a default value of 200 will be used.

SCAN Example

```
data first_last;  
  length Last_Name $ 15;  
  input @1 Name $20.;  
  Last_Name = scan(Name, -1, ' ');
```

```
datalines;
```

```
Jeff W. Snoker  
Raymond Albert  
Alfred E. Newman  
Steven J. Foster  
Jose Romeroz  
;
```

Name	Last_ Name
Jeff W. Snoker	Snoker
Raymond Albert	Albert
Alfred E. Newman	Newman
Steven J. Foster	Foster
Jose Romeroz	Romeroz

UPCASE, LOWCASE, and PROPCASE Examples

```
data case;  
  input Name $15.;  
  Upper = upcase(Name);  
  Lower = lowercase(Name);  
  Proper = propcase(Name, " ' ");  
datalines;  
gEOrge SMITH  
D'Angelo  
;
```

Name	Upper	Lower	Proper
gEOrge SMITH	GEORGE SMITH	george smith	George Smith
D'Angelo	D'ANGELO	d'angelo	D'Angelo

TRANWRD

Tranwrd(*String, Find, Replace*)

- This function performs a "find" and "replace" operation on String.
- If the length of the result has not be previously defined, it will default to 200

TRANWRD Example

```
data convert;  
  input @1 Address $20. ;  
  *** Convert Street, Avenue and  
  Boulevard to their abbreviations;  
  Address = tranwrd(Address, 'Street', 'St. ');  
  Address = tranwrd(Address, 'Avenue', 'Ave. ');  
  Address = tranwrd(Address, 'Road', 'Rd. ');  
datalines;  
89 Lazy Brook Road  
123 River Rd.  
12 Main Street  
;
```

Listing of Data Set CONVERT	
Obs	Address
1	89 Lazy Brook Rd.
2	123 River Rd.
3	12 Main St.

SPEDIS

Spedis(*string1*, *string2*)

- Returns the "spelling distance" between *string1* and *string2*.
- If the strings match exactly, the result is 0
- For each spelling mistake, you are assigned penalty points
- The spelling distance is the sum of the penalty points divided by the length of *string1*.

SPEDIS Example

```
data compare;  
  length String1 String2 $ 15;  
  input String1 String2;  
  Points = spedis(String1,String2);
```

```
datalines;
```

```
same same
```

```
same sam
```

```
first xirst
```

```
last lasx
```

```
receipt reciept
```

```
;
```

String1	String2	Points
same	same	0
same	sam	8
first	xirst	40
last	lasx	25
receipt	reciept	7

TRIMN and STRIP

Trimn(*String*) removes trailing blanks from a string. If the string contains a missing value, the length of the result is 0. The older Trim function returns a 1 in that case.

Strip(*String*) removes leading and trailing blanks from a string.

TRIMN and STRIP Example

```
data _null_;
  length Concat $ 8;
  One = '   ABC   ';
  Two = 'XYZ';
  One_two = ':' || One || Two || ':';
  Trim = ':' || trimn(One) || Two || ':';
  Strip = ':' || strip(One) || strip(Two) || ':';
  Concat = cats(':',One,Two,');
  put one_two= / Trim= / Strip= /
      Concat=;
run;
```

```
One_two=:   ABC   XYZ:
Trim=:     ABCXYZ:
Strip=:ABCXYZ:
Concat=:ABCXYZ:
```

NOTDIGIT, NOTALPHA, and NOTALNUM

Notdigit(*String*, *start*)

- Notdigit searches a string and returns the first position of a character that is NOT a digit.
- If *string* only contains digits, the function returns a 0.
- A starting position is optional. If negative the search is from right to left.
- Notalpha does the same for letters (upper- or lowercase) and Notalnum does the same for any letter or number.

NOTALPHA, NOTDIGIT, and NOTALNUM Example

```
data data_cleaning;  
  input String $20.;  
  Not_alpha = notalpha(strip(String));  
  Not_digit = notdigit(strip(String));  
  Not_alnum = notalnum(strip(String));
```

```
datalines;
```

```
abcdefg
```

```
1234567
```

```
abc123
```

```
1234abcd
```

```
;
```

String	Not_ alpha	Not_ digit	Not_ alnum
abcdefg	0	1	0
1234567	1	0	0
abc123	4	1	0
1234abcd	1	5	0

CATS and CATX

Cats(list of values)

Catx(delimiter, list of values)

- Cats first strips leading and trailing blanks from the strings and concatenates the results
- Catx also strips leading and trailing blanks and inserts a *delimiter* between each of the strings
- Values may be character or numeric. If numeric, there are no conversion messages in the log.

CATS and CATX Examples

```
data join_up;  
  length Cats $ 6 Catx $ 13;  
  String1 = 'ABC   ' ;  
  String2 = '   XYZ   ' ;  
  String3 = '12345' ;  
  Cats = cats(String1,String2);  
  Catx = catx('-',of String1-String3);  
run;
```

```
Cats = 'ABCXYZ'  
Catx = 'ABC-XYZ-12345'
```

Without the length statement, Cats and Catx would have a length of 200

COUNT and COUNTC

- Count(*String*, *substring*) counts the number of times *substring* appears in *String*.
- Countc(*String*, *list of characters*) counts the total number of characters found in *String*.

COUNT and COUNTC Example

```
data Dracula; /* Get it Count Dracula */
  input String $20.;
  Count_abc = count(String, 'abc');
  Countc_abc = countc(String, 'abc');
  count_abc_i = count(String, 'abc', 'i');
datalines;
xxabcxABCxxbbbb
cbacba
;
```

String	Count_ abc	Countc_ abc	Count_ abc_i
xxabcxABCxxbbbb	1	7	2
cbacba	0	6	0

Interesting use of the COUNTC and CATS Functions

```
data Survey;  
  input (Q1-Q5) ($1.);  
  Num = countc(cats(of Q1-Q5), 'y', 'i');  
datalines;  
yynnY  
nnnnn  
;
```

Listing of Survey

Q1	Q2	Q3	Q4	Q5	Num
y	y	n	n	Y	3
n	n	n	n	n	0

Some Date Functions

- MDY(Month,Day,Year) - returns a SAS date
- Weekday(Date) - returns the day of the week (1-Sunday, 2-Monday, etc.)
- Day(Date) - returns the day of the month (1-31)
- Month(Date) - returns the month (1-12)
- Year(Date) - returns the year
- Yrdif(Date1,Date2) - returns the number of years from Date1 to Date2*

*If you are using a version of SAS prior to 9.3, use:
Yrdif(Date1,Date2,'actual')

Date Function Example

```
data DateExamples;  
  input (Date1 Date2)(:mmddy10.) M D Y;  
  SAS_Date = MDY(M,D,Y);  
  WeekDay = weekday(Date1);  
  MonthDay = day(Date1);  
  Year = year(Date1);  
  Age = yrdif(Date1,Date2);  
  format Date: mmddy10.;  
datalines;  
10/21/1955 10/21/2012 6 15 2011  
;
```

Date1	Date2	SAS_Date	Week Day	Month Day	Year	Age
10/21/1955	10/21/2012	18793	6	21	1955	57

DIM

Dim(*Array_Name*)

Returns the number of elements in an array

DIM Example

```
data convert;  
  input (A B C) ($) x1-x3 y z;  
  array nums[*] _numeric_;  
  array chars[*] _character_;  
  do i = 1 to dim(nums);  
    if nums[i]=999 then nums[i]=.;  
  end;  
  do i = 1 to dim(chars);  
    chars[i] = propcase(chars[i], " '");  
  end;  
  drop i;  
  datalines;
```

```
RON jOhN mary 1 2 999 3 999
```

```
;
```

A	B	C	x1	x2	x3	y	z
Ron	John	Mary	1	2	.	3	.

N, NMISS, SUM and MEAN

- $N(\textit{list of values})$ - returns the number of non-missing values in the list
- $N\textit{miss}(\textit{list of values})$ - returns the number of missing values in the list
- $\textit{Sum}(\textit{list of values})$ - returns the sum of the values (ignoring missing values)
- $\textit{Mean}(\textit{list of values})$ - returns the mean of the values (ignoring missing values)

Note: If a list in the form Base1-Basen is used, precede the list with the keyword OF.

N, NMISS, SUM, and MEAN Example

```
data descriptive;  
  input x1-x5;  
  Sum = sum(of x1-x5);  
  if n(of x1-x5) ge 4 then  
    Mean1 = mean(of x1-x5);  
  if nmiss(of x1-x5) le 3 then  
    Mean2 = mean(of x1-x5);
```

```
datalines;  
1 2 . 3 4  
. . . 8 9  
;
```

Sum	Mean1	Mean2
10	2.5	2.5
17	.	8.5

SMALLEST and LARGEST

- `smallest(n, list of values)` - returns the *n*'th smallest value. When *n*=1, the function returns the same value as the `min` function.
- `Largest(n, list of values)` - returns the *n*'th largest value. when *n*=1, the function returns the same value as the `Max` function.
- Both functions ignore missing values

SMALLEST and LARGEST Example

```
data descriptive;  
  input x1-x5;  
  S1 = smallest(1, of x1-x5);  
  S2 = smallest(2, of x1-x5);  
  L1 = largest(1, of x1-x5);  
  L2 = largest(2, of x1-x5);
```

```
datalines;
```

```
7 2 . 6 4
```

```
10 . 2 8 9
```

```
;
```

x1	x2	x3	x4	x5	S1	S2	L1	L2
7	2	.	6	4	2	4	7	6
10	.	2	8	9	2	8	10	9

LAG

- The Lag function returns the value of its argument the last time the function executed
- In addition to Lag, there is Lag2, Lag3, etc. the n'th previous value
- When using LAG to access data from a previous *observation*, ensure the LAG function is never conditionally executed (in a SELECT group or IF-THEN logic).

LAG Example

```
data Moving;  
  input X @@;  
  Moving = mean(X, lag(x), lag2(x));  
datalines;  
50 40 55 20 70 50  
;
```

X	Moving
50	50.0000
40	45.0000
55	48.3333
20	38.3333
70	48.3333
50	46.6667

CALL SORTN and CALL SORTC Routines

- call `sortn(list of numeric variables)` replaces the original values with values in ascending sort order
- call `sortc(list of character variables)` replaces the original values with values in ascending order.*

* All the character variables must be the same length

CALL SORTN Example

```
data Scores;  
  input Score1-Score5;  
  call sortn(of Score1-Score5);  
  Top3 = mean(of Score3-Score5);  
datalines;  
80 70 90 10 80  
;
```

Score1	Score2	Score3	Score4	Score5	Top3
10	70	80	80	90	83.3333

Reference

- If you would like to learn more about these functions or other SAS functions, please take a look at my book:



SAS© Functions by Example, 2nd Edition
available from SAS Press.

support.sas.com/cody

Contact Information

Author: Ron Cody

email: ron.cody@gmail.com

Visit my author site at: support.sas.com/cody

