Paper FF-05

Fun with Functions

Author, Yogini Thakkar, Morrisville, NC

ABSTRACT

How often do we sit back and take the time to improve our SAS® programming skills by learning to use something better than what we have routinely used in the past? I invite you to join me as we come out of our comfort zone to examine what is new in SAS 9 and find out how we can use these new tools to improve our code. In this paper we will revisit some commonly used functions and see how they have been improved in SAS9, and find out how this will help us simplify our code while reducing the chance of errors

Everything relevant to this paper is in BASE SAS®. Although the examples have been executed on a PC under Windows, the examples are independent of any particular operating system with the exception of file definitions. .

INTRODUCTION

In this paper we will go through a few functions which we commonly use , some that are new to SAS 9 and a few which have enhanced features when used with SAS 9 .

Functions that will be covered in this paper include but not limited to:
COMPBL
COMPRESS
INDEX
FIND
COUNT
ANY
NOT
CAT

FUNCTIONS THAT REMOVE OR KEEP CHARACTERS OR NUMERALS:

These functions are character functions and work only on character variables.

1. COMPBL FUNCTION:

“Compress Blank” or COMPBL function is used to convert multiple blanks to a single blank. It is really useful when we have a long String of words which may be separated by multiple blanks and we need to convert those to a single blank.

Syntax: Newvar = COMPBL(String);

Examples:

String	Function	Output
"AB C D RON"	COMPBL(String)	“AB C D RON” *
"IT RAINED"	COMPBL(String)	“IT RAINED” **

* Multiple Blanks are converted to single blank.
** Multiple Blanks are converted to single blank.

2. COMPRESS FUNCTION:

COMPRESS function Compresses/removes blanks or specified characters from a given string.

Syntax: Newvar = COMPRESS(string,<compress chars>);

Examples:

String	Function	Output
"AB C D RON"	COMPRESS(string)	“ABCDRON” *
"(919)- 998- 1032"	COMPRESS(string,”(-)”)	“919 998 1032” **
"AB CD 12 34 EF GH"	COMPRESS(string,”0123456789)	“AB CD EF GH” ***

* Simple Compress without any compress chars just compresses all the blanks in the string.
** Compress char “(-)” are compressed. Note: Since blank is not a compress char blanks are left in the string.
***Compress char “0123456789” compresses all the numbers in the text leaving all the blanks as is.

Compress function is a great function to compress blanks and other characters but what if ….

a. We want to remove Capital A and lowercase 'a' ? We would have to do the following:
   e.g. char4 = "AB aC AD RON";
   newchar4 = COMPRESS(char4,'Aa ');

b. What if we need to remove all the characters and keep only numbers ? We would have to do the following:
   e.g. char5 = "A(919)- B998- 1032zaghH";
   newchar5 = COMPRESS(char5,'AaBbCcDd....GgHh..Zz ');

c. What if we need to Compress all the Punctuations in the string? We would have to do the following:
   e.g. char6 = "A(919)- B998- 1032zaghH.'-*";
   newchar6 = COMPRESS(char6,"(-)..'-*");

But what if we had to remove all the alphabets and punctuations from a string? It can be tedious. So SAS introduced the third argument to the COMPRESS function in SAS 9.

3. COMPRESS Function with the third Argument:

This Function Compresses/removes blanks or specified characters from a given string. It also has an additional Modifier that could be passed to give more instructions.

Syntax: Newvar = COMPRESS(string, <compress char> , <modifier>);

Modifiers that can be used are:

K = Keep
I = Ignore case
D = Digits
A = Alpha (upper and lower)
S = Space
P = Punctuations

Examples:

String	Function	Output
"AB aC AD RON"	COMPRESS(String,'A','i')	“B C D RON” *
"A(919)- B998- 1032zaghH"	COMPRESS(String, ,'a')	“(919)- 998- 1032” **
"A(919)- B998- 1032zaghH.'-*"	COMPRESS(String, ,'p')	“A919 B998 1032zaghH” ***
"A(919)- B998- 1032zaghH.'-*"	COMPRESS(String, ,'kds')	“919 998 1032” ****
"A(919)- B998- 1032zaghH.'-*"	COMPRESS(String,'0','kds')	“919 998 1032” *****

* Compress (String,’A’,’I’) would compress the letter ‘A’ and with the modifier ‘I’ case would be ignored . So this function will compress ‘A’ and ‘a’.

** Compress (String, ,'a') does not have the second argument but has the modifier 'a' which says drop Alpha (upper and Lower). Hence all the alphabets from the string are compressed.
*** Compress (string, , 'p') does not have the second argument but has the modifier 'p' which says drop punctuations. Hence all the punctuations from the string are compressed. Note that the blank spaces are not part of punctuations and hence are not compressed.
**** Compress(String, ,'kds') does not have the second argument but has the modifiers 'kds' which say 'Keep', 'Digits' and 'Space'. Hence only digits and spaces from the string are kept.
***** Compress (String,'0','kds') has a second argument and also additional modifiers. Second argument '0' instructs to compress all Zero's in the string. However the third argument 'kds' instructs to keep all digits and space. As observed in the output the third argument takes precedence over the second argument.

Conclusion:
COMPBL function can be very useful for removing multiple blanks in a sting. The COMPRESS function with the third argument (new modifiers) can be very handy to simplify our code and reduce chances of errors.

FUNCTIONS THAT SEARCH FOR STRINGS:

These functions are character functions and work only on character variables.

1. INDEX, INDEXC AND INDEXW FUNCTIONS:

INDEX and INDEXC functions return the position where the char or string of characters first occur in the string. INDEXW function searches for entire word. It searches for position of text separated by word separators like spaces, start of line or End of line.

Syntax: INDEX(String, Search String);
INDEXC(String, Search String);
INDEXW(String, Search Word);

Examples:

String	Function	Output
"ABCDRADFE"	INDEX(String,'A')	1 *
"BCDaHgij"	INDEX(upcase(String),'A')	4 **
"yz XY uv aB"	INDEXC(upcase(String),'AB')	10 ***
"This is a Presentation"	INDEXW(string,'IS')	0 ****
"This is a Presentation"	INDEXW(string,'is')	6 *****

* INDEX (String,'A') searches for letter 'A' in the string. It is found at position 1.
** INDEX(upcase(String),'A') upcases the string and then searches for 'A'. This helps search for 'a' or 'A' in the String and it returns the position of 4.
*** INDEXC(upcase(String),'AB') upcases the string and then searches for 'A' or 'B'. Hence it will return Position of 10 where it first encounters 'A".
**** INDEXW(string,'IS') searches the string for whole word 'IS' in the string separated from other words either by space, start of line or End of line. Since there is no 'IS' in the String the function returns a 0.
***** INDEXW(string,'is') searches the string for whole word 'is' in the string separated from other words either by space, start of line or End of line. The function returns 6.

Let us look at the different INDEX functions with same example:
String = 'this is where her bag was'

Function	Output
INDEX (string,'her')	10 *
INDEXC (string,'her')	2 **
INDEXW (string,'her')	15 ***

* INDEX (string,'her') searches for the substring ‘her’ in the string and returns the position of 10.
** INDEXC (string,'her') searches the string for either ‘h’,’e’ or ‘r’ and returns the position of 2 which is when it first encounters letter ‘h’.
*** INDEXW (string,'her') searches the string for the word ‘her’ and returns the position of 15 where if first finds ‘her’.

But it is not so much fun to remember UPCASE option to search for upper and Lower cases. So SAS introduced more arguments to the FIND function in SAS 9.

2. FIND, FINDC AND FINDW FUNCTIONS:

FIND and FINDC functions return the position where the char or string of characters first occur in the string.

Syntax : FIND (String, find,<modifier>, startpos)
FINDC (String, find,<modifier>, startpos)

Modifiers:
I = Ignore Case
T = Ignore Trailing Blanks
V = Search of any other string than required

Startpos:
Indicates the Position from where to start the search.

Examples:

String	Function	Output
"ABCDRADFE”	FIND(String,‘A')	1 *
“BCDaHgij”	FIND(String,'A‘,'i')	4 **
“yz XY uv aB”	FINDC(String,‘AB','i')	10 ***
“This is a Presentation”	FIND(String,'is','i',4)	6 ****
“This is a Presentation”	FIND(String,'is','i',-99)	6 *****
“This is a Presentation”	FINDC(String,'is','iv')	1 ******

* FIND(String,‘A') looks for ‘A’ and return and values of 1.
** FIND(String,'A‘,'i') uses modifier ‘i’ which ignores case . The function now searches for ‘A’ as well as ‘a’.
*** FINDC(String,‘AB','i') searches for string ‘AB’ by ignoring case. The function thus returns 10.
**** FIND(String,'is','i',4) searches from position 4 for string ‘is’ by ignoring case returning value of 6.
***** FIND(String,'is','i',-99) searches backward for the string ‘is’ by ignoring case returning value of 6. Note: Search is backward but position returned is still from the start of string.
****** FINDC(String,'is','iv') searches for any string other than ‘is’ by ignoring case and hence returns value of 1.

FUNCTIONS THAT ARE NEW TO SAS 9:

These functions are character functions and work only on character variables.

1. COUNT AND COUNTC FUNCTIONS:

These functions count the occurrences of a string in a bigger character string.

Syntax: COUNT(String, search string,<Modifier>)
COUNTC (String, search string,<Modifier>)

Modifiers :
I = Ignore Case
T = Remove Leading and Trailing Blanks

Examples:
String = "Ab ab de ab AB gh ab"

Function	Output
COUNT(string,'ab')	3 *
COUNT(string,'AB')	1 **
COUNT(string,'ab','I')	5 ***
COUNTC(string,'ab')	7 ****
COUNTC(string,'ab','I')	10 *****

* COUNT(string,'ab') will count all the occurrences of 'ab' in the string and hence returns a value of 3.
** COUNT(string,'AB') will count all the occurrences of 'AB' in the string and hence returns a value of 1.
*** COUNT(string,'ab','I') will count all the occurrences of 'ab' and 'AB' due to 'I' modifier in the string and hence returns value of 5.
**** COUNTC(string,'ab') will count all the occurrences of 'a' as well as 'b' in the string and hence return the value of 7.
***** COUNTC(string,'ab') will count all the occurrences of 'a','A','b' and 'B' in the string due to the 'I' modifier and hence returns the value of 10.

2. ANY FUNCTIONS:

These functions return the first position of ANY DIGIT, ALPHA, ALNUM, PUNCT or SPACE in the string of characters.

Syntax: ANY*****(String, Startpos);
***** :
DIGIT = Any digit
ALPHA = Any character
ALNUM = Any character or digit
PUNCT = Any punctuation
SPACE = Any space

Examples:
String = "Abc123, yog376"

Function	Output
ANYDIGIT(string)	4 *
ANYDIGIT(string,7)	12 **
ANYDIGIT(string,-99)	14 ***
ANYALPHA(string)	1 ****
ANYALNUM(string)	1 *****
ANYPUNCT(string)	7 ******
ANYSPACE(string)	8 *******

* ANYDIGIT(string) searches for any digit and returns the position of 4 which is for number '1'

** ANYDIGIT(string,7) searches for any digit in the string starting from position 7 and returns the position of 12 which is for number ‘3’.
*** ANYDIGIT(string,-99) searches for any digit in the string starting from -99 or backward and returns the position of 14 which is for number ‘6’. Note: the search is backward but position returned is relative to the starting position.
**** ANYALPHA(string) searches for any character in the string and returns the position of 1 which is for character ‘A’
***** ANYALNUM(string) searches for any character or digit in the string and returns the position of 1 which is for character ‘A’.
****** ANYPUNCT(string) searches for any punctuation in the string and returns the position of 7 which is for ‘,’.

3. NOT FUNCTIONS:

These functions return the first position in the string which is NOT DIGIT, ALPHA, ALNUM, PUNCT or SPACE.

Syntax: Syntax: NOT*****(String, Startpos);
***** :
DIGIT = Not digit
ALPHA = Not character
ALNUM = Not character or digit
PUNCT = Not punctuation
SPACE = Not space

Examples:
String = "Abc123, yog376"

Function	Output
NOTDIGIT(string)	1 *
NOTDIGIT(string,7)	7 **
NOTDIGIT(string,-99)	11 ***
NOTALPHA(string)	4 ****
NOTALNUM(string)	7 *****
NOTPUNCT(string)	1 ******
NOTSPACE(string)	1 *******

* NOTDIGIT(string) searches for first non digit value in the string and returns the position 1 which is for character ‘A’.
** NOTDIGIT(string,7) searches for the first non digit value in the string starting from position 7 and returns the position 7 which is for punctuation ‘,’.
*** NOTDIGIT(string,-99) searches for the first non digit value in the string starting from position -99 or backward and returns the position of 11 which is for character ‘g’.
**** NOTALPHA(string) searches for the first non character value in the string and returns the position of 4 which is for the number ‘1’.
***** NOTALNUM(string) searches for the first non character non digit value in the string and returns the position of 7 which is for the punctuation ‘,’ .
****** NOTALNUM(string) searches for the first value in the string which is not a punctuation and returns the position of 1 which is for character ‘A”.
******* NOTSPACE(string) searches for the first value in the string which is not a space and returns the position of 1 which is for character ‘A”.

4. COMPARE FUNCTION:

This function compares one character string with another. if the strings match the function returns 0 else it returns the position of first non matching value in string.

Syntax: COMPARE(String1, String2, <Modifier>)

Modifiers:
I = Ignore Case
L = Remove Leading Blanks. Trailing blanks are ignored by default.
: = Truncate the Longer string to the length of the Shorter String .

Example 1:
string1 = "v430.210";
string2 = " V430.210";

COMPARE(string1,string2);

Since string1 and string2 are not the same the function returns the position of first non matching value in the string 1 which is a blank in string 2 and ‘v’ in string 1.

Example 2:
string1 = "v430.210";
string2 = " V430.210";

COMPARE(string1,string2,’L’);
This example is similar to example1 but the addition of ‘L’ modifier removes the leading blanks and compares the two strings. Since string1 and string2 are not the same the function returns the position of first non matching value in the string 1 which is a ‘V’ in string 2 and ‘v’ in string 1.

Example 3:
string1 = "v430.210";
string2 = " V430.210";

COMPARE(string1,string2,’IL’);
This example is similar to example2 but the addition of ‘I’ modifier ignores case and compares the two strings. Since string1 and string2 are now same the function returns the value of 0.

Example 4:
string1 = "v430.210";
string2 = "V430";

COMPARE(string1,string2,'I:') ;
Since the modifier ‘:’ is used the function will truncate string1 to the length of string2 and also ignore case due to ‘I’ modifier. Now string1 and string2 are same and hence the function returns value of 0.

Example 5:
string1 = "v430.210";
string2 = ".210";

COMPARE(string1,string2,'I:') ;
Since the modifier ‘:’ is used the function will truncate string1 to the length of string2 and also ignore case due to ‘I’ modifier. However String1 and String2 are still not same and hence the function returns the value of 1.

FUNCTIONS THAT CONCATENATE STRINGS:

1. CAT FUNCTION:

This function simply concatenates strings.

Syntax: CAT(String1,…Stringn);

Example1:
string1 = " ABC “
string2 = "def"
string3 = "*"

string4 = CAT(string1, string2, string3) .
CAT function simply concatenates strings hence string4 = “ ABC def*” . Note: The leading and trailing blanks are maintained in the output.

Example 2:
string1 = " ABC “

string2 = CAT(":",string1,":")
CAT function will concatenate all the strings hence string2 = ““: ABC :” Note: The leading and trailing blanks are maintained in the output.

2. CATS FUNCTION:

This function concatenates strings by removing leading and trailing blanks.

Syntax: CATS(String1, …Stringn);

Example 1:
string1 = " ABC “
string2 = "def"
string3 = "*"

string4 = CATS(string1, string2,string3)
CATS function removes the leading and trailing blanks before concatenation and hence string4 = ““ABCdef*”

Example 2:
string1 = " ABC “
string2 = "def"
string3 = put(5,5.)

string4 = CATS(string1,string4, string2)
CATS function remove the leading and trailing blanks from each string before concatenation. Hence string4 = “ABC5def” . Note: use of a function in string4.

3. CATX FUNCTION:

This function concatenates strings and puts a Separator in between after removing leading and trailing blanks.

Syntax: CATS(Separator, String1, …Stringn);

Example 1:
string1 = " ABC “
string2 = "def"
string3 = "*"

string4 = CATX('-',string1, string2, string3) ;
CATX will remove the leading and trailing blanks from the strings and then concatenate them with a ‘-‘ in between. Hence string4 = ““ABC-def-*” .

String5 = CATX(' and ',string1, string2, string3)
CATX will remove the leading and trailing blanks from the strings and then concatenate them with a ‘ and ‘ in between. Hence string4 = “ABC and def and *” . Note: the leading and trailing blanks around the ‘ and ‘ (separator) are not removed.

NUMERIC FUNTIONS:

There are several numeric functions available already a few of them with their functions are listed below.
N - Counts the Number of non missing values in a list of values

NMISS - Counts the Number of missing values in a list of values
SUM – Returns the sum of values
MEAN – Returns the mean of list of values
MEDIAN – Returns the median of list of values
MIN,MAX - Returns the largest or smallest non missing value from the list

New to SAS 9.
There were two new numeric functions introduced in SAS 9. They are LARGEST and the SMALLEST.

1. LARGEST FUNCTION:
This function returns the largest value from a list of values after ignoring the missing values.

Syntax ; LARGEST(n, list of values)
'n' = nth value. If n =2 then the function returns the second largest number.

Example:
x1 = 3 x2 = 4 x3 = . x4 = 8
x5 = 10 x6 = 5 x7 = 6 x8 = .

newnum1 = LARGEST(1,of x1-x7)
Since n = 1 the function will return the largest number in the list ignoring the missing. Hence newnum1 = 10.

newnum2 = LARGEST(3,of x1-x7)
Since n = 3 the function will return the third largest number in the list ignoring the missing. Hence newnum2 = 6.

2. SMALLEST FUNCTION:
This function returns the smallest value from a list of values after ignoring the missing values.

Syntax ; SMALLEST(n, list of values)
'n' = nth value. If n =2 then the function returns the second smallest number.

Example:
x1 = 3 x2 = 4 x3 = . x4 = 8
x5 = 10 x6 = 5 x7 = 6 x8 = .

newnum1 = SMALLEST(1,of x1-x7)
Since n = 1 the function will return the smallest number in the list ignoring the missing. Hence newnum1 = 3.

newnum2 = SMALLEST(2,of x1-x7)
Since n = 2 the function will return the second smallest number in the list ignoring the missing. Hence newnum2 = 4.

Note: MIN and SMALLEST function ignore the missing values while looking at the list of values. However there is one function which considers missing as the smallest or min value which is the ORDINAL function.

Syntax: ORDINAL(n, list of values)
'n' = nth value. If n =2 then the function returns the second smallest number.

Example:
x1 = 3 x2 = 4 x3 = . x4 = 8
x5 = 10 x6 = 5 x7 = 6 x8 = .

newnum1 = ORDINAL (1,of x1-x7)
Since n = 1 the function will return the smallest number in the list including the missing. Hence newnum1 = missing.

newnum2 = ORDINAL (3,of x1-x7)
Since n = 3 the function will return the third smallest number in the list including the missing. Hence considering there are two missing values in the list the third smallest is 3. Hence newnum2 = 3.

CONCLUSION

Functions can be really fun to use. They help make your code short, precise and easy to follow, at the same time reducing the chances of error. With the new functions introduced with SAS 9 code can be simplified to a great extent.

The COMPRESS function with the new modifiers can come in really handy to manipulate character variables. The FIND function with its modifiers has a lot of advantage over the INDEX function and would be a better choice to use when possible. The COUNT, COUNTC, NOT and ANY functions are really helpful new functions for character variables. The COMPARE function is definitely my favorite as it reduces chances of error drastically with its modifiers. The new SMALLEST and the LARGEST function can come in real handy if we don't want to do too much coding yet find the nth largest/smallest value in the list.

I encourage everyone to come out of their comfort zones and try out these functions.

CONTACT INFORMATION

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

Name : Yogini Thakkar
Enterprise : Quintiles
City, State ZIP : Morrisville, 27560
Work Phone: 919-998-1032
E-mail: yogini.thakkar@quintiles.com