PharmaSUG 2016 - Paper QT24

Remember to always check your simple SAS function code!

Yingqiu Yvette Liu, Merck & Co. Inc., North Wales, PA

ABSTRACT

In our daily programming work we may not get expected results when using seemingly clear logic and simple SAS functions. When we dig into the problem, we may discover the issue: either a SAS function was utilized incorrectly or the programming logic wasn't applied properly. In this paper, the author will use examples such as SCAN function and LAG function to demonstrate these points in an effort to share potential pitfalls and challenges when using SAS functions. Reviewing others' mistakes is often an excellent way to learn and improve our programming skills!

INTRODUCTION

Some SAS functions may seem very simple to use. However, if not used properly, they may bring unexpected and incorrect results. Examples of SCAN function and LAG function will be used in the body of the paper to support this point.

I. SCAN FUNCTION

SCAN function is a very handy tool to extract a certain part of (i.e. it returns the n^{th} word from) a character string. How does it differentiate each word in the character string? Use delimiters.

Example 1:

```
data one (keep=p1 p2 p5 p6);
    string = "part1 part2 part3 part4    part5      part6";
    p1 = SCAN(string,1);
    p2 = SCAN(string,2);
    p5 = SCAN(string,-2);
    p6 = SCAN(string,-1);
run;
```

Content of dataset ONE

p1	p2	p5	p6
part1	part2	part5	part6

Table 1. Content of dataset ONE

We can see from this example that

1. The 2^{nd} argument of SCAN function can be either a positive integer (meaning searching the n^{th} word from left to right of the character string) or a negative integer (meaning searching from right to the left).
2. The argument of delimiters (the 3^{rd} argument) is omitted in example 1, resulting in the default delimiters being used – the only delimiter displayed in the character string, BLANK. Both single blank character and multiple consecutive blank characters are treated the same by SCAN function.

Example 2:

Content of input dataset PK

SUBJID	ANALYTE	PERIOD	Day	TPT_DESC
1	Drug1	1	1	hour 0 predose
1	Drug1	1	1	hour 0.5
1	Drug1	1	1	hour 1
1	Drug1	1	1	hour 2
1	Drug1	1	1	hour 3

Table 1. Content of input dataset PK

```
data two;
    set data pk;
    hour = input(scan(tpt_desc,2),best.);
run;
```

Content of dataset TWO

SUBJID	ANALYTE	PERIOD	Day	TPT_DESC	HOUR
1	Drug1	1	1	hour 0 predose	0
1	Drug1	1	1	hour 0.5	**0**
1	Drug1	1	1	hour 1	1
1	Drug1	1	1	hour 2	2
1	Drug1	1	1	hour 3	3

Table 3. Content of dataset TWO

It's a big surprise that when processing the 2nd record, it seems SCAN function didn't behave as expected. Let's have a close look at the 2nd record of dataset TWO:

- Why the HOUR value is 0 but not the expected value 0.5?
- What has the SAS code done on this record when extracting the 2nd word from time point description string TPT_DESC and then converted it to numeric?
- Why we get expected and correct HOUR value on rest of records but not on 2nd record?

The problem existed in the omitted third argument of SCAN function, argument of delimiter. SCAN function uses default delimiter(s) when the 3rd argument of delimiters is omitted. In ASCII environment, the default delimiters are:

Blank ! $ % ( ) * + , - . / ; < ^ ¦

Now, it's clear - the 2nd record TPT_DESC value "hour 0.5" has two default delimiters displayed, BLANK and DECIMAL POINT (it is also a punctuation character PERIOD). So, the 2nd "word" parsed and returned by SCAN function is "0", but not "0.5". In order to get value "0.5" returned, we have to specify 3rd delimiter argument specifically, in this example, it should be BLANK. Have a look at the code below and pay attention to the 3rd argument of SCAN function.

```
data three;
    set data pk;
    hour = input(scan(tpt_desc,2," "),best.);
run;
```

Content of dataset THREE

SUBJID	ANALYTE	PERIOD	Day	TPT_DESC	HOUR
1	Drug1	1	1	hour 0 predose	0
1	Drug1	1	1	hour 0.5	**0.5**
1	Drug1	1	1	hour 1	1
1	Drug1	1	1	hour 2	2
1	Drug1	1	1	hour 3	3

Table 4. Content of dataset THREE

Now, everything works fine.

Let's play SCAN function with more fun in example 3.

Example 3:

```
data four (drop=string);
    string = "word1, word2; word9610/ (word4511) word289";
    w1 = scan(string,1);
    w2 = scan(string,2,"d");
    w3 = scan(string,3,"1");
```

```
        w4 = scan(string,4,"d2");
        output;
        w1 = scan(string,1,"w,;1");
        w2 = scan(string,2,"w,;1");
        w3 = scan(string,3,"w,;1");
        w4 = scan(string,4,"w,;1");
        output;
        w1 = scan(string,5,"w,;1");
        w2 = scan(string,6,"w,;1");
        w3 = scan(string,7,"w,;1");
        w4 = scan(string,8,"w,;1");
        output;
    run;
```

word1, word2; word9610/ (word4511) word289

nth= 1 2 3 4 5 6 7 8 9

delimiter="**w,;1"**

Content of dataset FOUR

w1	w2	W3	W4
word1	1, wor	0/ (word45	9610/ (wor
ord		ord2	
ord96	0/ (	ord45	)

Table 5. Content of dataset FOUR

Let's summarize the lesson learned for SCAN function – as a good practice, do not omit 3rd argument, though theoretically the 3rd argument is optional; instead, always specify 3rd argument of delimiters explicitly.

Syntax of SCAN function:

SCAN(string, count <, charlist <, modifier>>)

Argument 1: *string*.

It can be a character constant, variable or expression.

Argument 2: *count*.

It can be a non-zero numeric constant, variable or expression that has integer value telling which word in character string that SCAN function returns. Positive integer value means searching word in character string from left to right, negative integer value means searching from right to left.

Argument 3: *charlist*.

- It is an optional argument specifies a list of delimiters to separate words. By default, all characters specified for this argument will be used as delimiters.
- When this argument is omitted, default list of delimiters will be used. Default list of delimiters in ASCII environment are (will be a bit different in ABCDIC system):

  Blank ! $ % ( ) * + , - . / ; < ^ ¦
- If the "K" or "k" value is specified in the 4th argument of modifier, all characters NOT in 2nd argument *charlist* will be used as delimiters.

Argument 4: *modifier*.

It is an optional argument that can be a character constant, a variable or an expression on which each non-blank character modifies the action of SCAN function. This paper will not cover details of this argument, please reference SAS Help or SAS Language Reference for details.

II. LAG FUNCTION

LAG function is one of a few SAS functions or statements that can perform computations across observations in DATA step.

The syntax of the LAG function is: **LAGn(argument)**

where ***n*** is a positive integer representing the nth previous execution of the function, and ***argument*** can be any numeric or character constant, variables or expression.

LAG1(argument) can also be written as LAG(argument).

LAGn function returns value from the top of a queue (this queue stores number of values of n consecutive previous records) .

The table below is a display of dataset FIVE (which only has one variable X).

Content of dataset FIVE

X
1
2
3
4
5
6
7
8
9
10

Table 5. Content of dataset FIVE

Example 4:

```
data six;
   set five;
   lag1x = lag1(x);
   lag2x = lag2(x);
   lag3x = lag3(x);
   lag4x = lag4(x);
   lagcont1 = lag(10);   *LAG function argument is numeric constant 10;
   lagcont2 = lag(x+10); *LAG function argument is numeric expression;
   lagcont3 = lag("A");  *LAG function argument is character constant "A";
   lagcont4 = lag("A"||"B"); *LAG function argument is character expression;
run;
```

Content of dataset SIX

X	LAG1X	LAG2X	LAG3X	LAG4X	LAGCONT1	LAGCONT2	LAGCONT3	LAGCONT4
1	.	.	.	.	.	.		
2	1	.	.	.	10	11	A	AB
3	2	1	.	.	10	12	A	AB
4	3	2	1	.	10	13	A	AB
5	4	3	2	1	10	14	A	AB
6	5	4	3	2	10	15	A	AB

7	6	5	4	3	10	16	A	AB
8	7	6	5	4	10	17	A	AB
9	8	7	6	5	10	18	A	AB
10	9	8	7	6	10	19	A	AB

Table 6. Content of dataset SIX

Example 5:

```
data seven;
   set five;
   lag1x = lag(x);

   if x in (1,3,5,7,9) then do;
          xlagc1 = lag(x); *LAG function called in IF conditions;
          xlagc2 = lag1x;
   end;
run;
```

Content of dataset SEVEN

X	LAG1X	XLAGC1	XLAGC2
1	.	.	.
2	1	.	.
3	2	1	2
4	3	.	.
5	4	3	4
6	5	.	.
7	6	5	6
8	7	.	.
9	8	7	8
10	9	.	.

Table 6. Content of dataset SEVEN

Quite interestingly, the two variables XLAGC1 and XLAGC2 calculated inside the IF condition have different values on the same record 1,3,5,7,9 where the IF condition was satisfied.

Out of 10 records of input dataset FIVE, in the DATA SEVEN code block, the IF condition has been satisfied 5 times – on record 1, 3, 5, 7, 9. It also tells that value assignment for XLAGC1 and XLAGC2 have been implemented 5 times.

Let's look at the code of DATA SEVEN again.

```
data seven;
   set five;
   lag1x = lag(x);    *LINE1: unconditionally calling LAG function;

   if x in (1,3,5,7,9) then do;
     xlagc1 = lag(x);*LINE2: conditionally calling LAG function;
     xlagc2 = lag1x; *LINE3: conditionally assign value returned from ;
                      *unconditional LAG function call which performed outside of;
                      *IF condition;
   end;
run;
```

Remember to always check your simple SAS function code!, continued

Let's revise the data step a little bit by adding more log messages so that we may see the data flow better.

Example 6:

```
data eight;
   set five;

   putlog "RECORD " _n_;

   lag1x = lag(x);
   putlog "line1 before IF: " x= "lag(x)=" xlag1=;

   if x in (1,3,5,7,9) then do;
          xlagc1 = lag(x);
          putlog "line2 inside IF: " x= "lag(x)=" lagc1=;

          xlagc2 = lag1x;
          putlog "line3 inside IF: " x= "xlag1=" xlagc2=;
   end;

   putlog "-----------------------------------";
run;
```

Once the data step excited, let's have a look at log messages the above data step wrote explicitly:

```
RECORD 1
line1 before IF: x=1 lag(x)=xlag1=.
line2 inside IF: x=1 lag(x)=xlagc1=.
line3 inside IF: x=1 xlag1=xlagc2=.
----------------------------------------
RECORD 2
line1 before IF: x=2 xlag1=1
----------------------------------------
RECORD 3
line1 before IF: x=3 lag(x)=xlag1=2     AUTHOR's NOTE:
line2 inside IF: x=3 lag(x)=xlagc1=1    ←Reference value x=1 in RECORD 1
line3 inside IF: x=3 xlag1=xlagc2=2     ←Reference value x=2 in RECORD 2
----------------------------------------
RECORD 4
line1 before IF: x=4 xlag1=3
----------------------------------------
RECORD 5
line1 before IF: x=5 lag(x)=xlag1=4     AUTHOR's NOTE:
line2 inside IF: x=5 lag(x)=xlagc1=3    ←Reference value x=3 in RECORD 3
line3 inside IF: x=5 xlag1=xlagc2=4     ←Reference value x=4 in RECORD 4
----------------------------------------
RECORD 6
line1 before IF: x=6 xlag1=5
----------------------------------------
RECORD 7
line1 before IF: x=7 lag(x)=xlag1=6     AUTHOR's NOTE:
line2 inside IF: x=7 lag(x)=xlagc1=5    ←Reference value x=5 in RECORD 5
line3 inside IF: x=7 xlag1=xlagc2=6     ←Reference value x=6 in RECORD 6
----------------------------------------
RECORD 8
line1 before IF: x=8 xlag1=7
----------------------------------------
RECORD 9
line1 before IF: x=9 lag(x)=xlag1=8     AUTHOR's NOTE:
line2 inside IF: x=9 lag(x)=xlagc1=7    ←Reference value x=7 in RECORD 7
```

```
line3 inside IF: x=9 xlag1=xlagc2=8      ←Reference value x=8 in RECORD 8
----------------------------------------
RECORD 10
line1 before IF: x=10 xlag1=9
----------------------------------------
```

So now it's clear why the two assignments inside IF condition have difference values – because they reference different previous records when LAG function called or using returned value from LAG function executed before block of IF statement.

It is a common trap that executing LAG function conditionally, such as in IF…THEN… block. In this situation, the queue that LAG function stores only includes values from records when the condition satisfied.

As a lesson, please follow good programming practice - do not call LAG conditionally.

Example 6: good practice - calling LAG function unconditionally

It's a good practice that:

1. Always call LAG function outside of any condition, and assign returned value from LAG function to a new variable.
2. May reference and make assignment of the LAG function returned value (stored in the new variable in the above step) inside any conditional block of code.
3. Doing this way, LAG should return value as what we expected.

```
data nine;
   set five;

   lag1x = lag(x);   *unconditionally calling LAG function;

   if x in (1,3,5,7,9) then do;
     xlagc2 = lag1x; *conditionally assign value returned by unconditional;
                     *LAG function call which executed outside of IF condition;
   end;
run;
```

Example 7: bad practice - calling LAG function conditionally

```
data ten;
   set five;

   if x in (1,3,5,7,9) then do;
     xlagc1 = lag(x);*conditional calling LAG function;
   end;
run;
```

CONCLUSION

When using SAS functions, please always remember to check SAS function code even it seems very simple - have you utilized the function correctly? Is the programming logic applied properly? Reviewing others' mistakes is often an excellent way to learn and improve our programming skills!

REFERENCES

SAS® 9.2 Language Reference: Dictionary, Fourth Edition
SCAN Function
http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/viewer.htm#a000214639.htm

SAS® 9.2 Language Reference: Dictionary, Fourth Edition

LAG Function
http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/viewer.htm#a000212547.htm

ACKNOWLEDGMENTS

The author would like to thank Amy Gillespie for her effort in helping revising the abstract!

CONTACT INFORMATION

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

Yingqiu Yvette Liu
Merck & Co., Inc.
351 N Sumneytown Pike
North Wales, PA 19454
yvliu99@gmail.com