SESUG Paper 194-2018

DO the Loop De Loop!

Jennifer H. Lindquist, Durham Veterans Affairs Health Care System

ABSTRACT

Like the break neck speed of a rollercoaster careening through loops, barreling through programs using SAS® DO loops and arrays instead of repetitive statements can shorten your programming time and lines of code. This introduction to arrays and loop processing covers the basics - starting with the syntax of each and preceding to examples. By combining the two concepts you can construct a simple iterative DO loop to traverse an array with ease.

Make going around in circles work for you!

INTRODUCTION

Arrays and loops are techniques that allow your code to fly through repetitive tasks. They can reduce lines of code and make programs more efficient. When you find yourself cutting and pasting the same snippet of code over and over, consider using arrays and loops. This paper presents the syntax for loops and arrays. At the end, this paper shows the power of pulling together the two constructs.

.

LOOPS

SAS, like most computer languages, includes several looping options. One of the most common is the iterative DO loop.

The general syntax for an iterative loop is:

DO *index* = ***start*** <TO ***stop***> <BY ***increment***>;
. . . more statements . . .

END;

The key word **DO** indicates the start of the loop.

The i*ndex* is a variable. SAS assigns a new value to the index for each iteration of the loop,

Start is value of the index for the first iteration of the loop.

Stop is the stopping value or stopping condition of the loop. When the index exceeds the stop value, the program control exits the loop and proceeds to the next statement in the DATA step after the END statement

Increment is the amount the index increases (or decreases, if negative) for the next pass through the loop.

The key word **END** indicates the end of the loop.

Optional attributes designated by < >.

Here is an example of a simple iterative DO loop:

```
DATA Sample;

DO i = 1 TO 5;
      x=2*i;
      y=RANUNI(0);
      OUTPUT;
 END;

RUN;
```

The loop begins with the keyword DO; the last statement of the loop is the key word END. The variable i is the index variable. In this example, i takes on the values 1, 2, 3, 4, and 5. SAS repeatedly executes the lines of code between the DO and END statements for each value of the index variable. In the example above, SAS assigns the variable x a value two times the value of i, assigns the variable y a randomly generated number, and then outputs the observation to the data set SAMPLE. The resulting data set is:

Obs	i	x	y
1	1	2	0.92252
2	2	4	0.29188
3	3	6	0.03757
4	4	8	0.92823
5	5	10	0.46620

Output 1. Output from Simple Loop

Loops are a painless way to generate test data. First, write statements to create one observation. Then place the looping structure around those statements along with an output statement to produce the number of observations you want in the test data set.

Although the simple example above is an often-used format of the iterative DO Loop, using the DO Loop's optional syntax, allow the DO Loop to take on many additional forms.

One variation is to list the values the index is to assume:

```
DO i = 2, 3, 5, 8;
```

The index can also be a character variable with the values listed enclosed in quotes.

```
DO Mon = 'Sept', 'Oct', 'Nov';
```

The increment value (italicized in the example below) indicates how much to increase/decrease the index (j) for each trip through the loop. Specifying the increment is optional and it has a default value of one.

```
DO j=3 TO 11 BY 2;
```

In the above example SAS executes the loop five times with the value of the index variable j starting at 3 and increasing by 2 each time. The values of j are 3, 5, 7, 9, and 11.

When the increment is negative, the start value needs to be larger than the stop value for the loop to be executed. Likewise, if the increment is positive, the start value needs to be smaller than the stop value.

SAS evaluates the value of the index, compares it with the stop value and the direction of increment (positive or negative) prior to entering the loop.

```
DO k=10 TO 0 BY -1.5;
```

SAS executes the loop seven times with k taking on the values 10, 8.5, 7, 5.5, 4, 2.5, and 1. Notice the last value of k is not the stop value because 1-1.5= -0.5 which is beyond the stop value.

NOTE: ***A restriction of loops is that the index cannot take on both numeric and character values.

ARRAY STATEMENT

The array statement defines the members of an array.

The general syntax is:

ARRAY *array-name* {*dimension*} <$><*length*> <*array-elements*> <(*initial-values*)>;

The *array-name* is a term for a group of variables. It can be any user designated SAS name not used elsewhere in your program.

The *dimension* shows the number and arrangement of items in the array with either integer(s) or a wildcard asterisk inside ( ), { } or [ ].

An array statement consisting of character variables must include $.

Length allows you to specify the length of variables not previously assigned a length.

Array-elements lists variables that make up the array

The *initial-values* is a list of starting values for the array elements.

Optional attributes designated by < >.

Here is an example of a typical array statement:

```
ARRAY  Aka{4}  age gender race ethnicity;
```

In the above example the array Aka consists of 4 components, namely Aka(1), Aka(2), Aka(3) and Aka(4). Since array-elements are specified, Aka(1), Aka(2), Aka(3) and Aka(4) are not new variables but are shorthand nicknames or aliases for the itemized variables indexed by the position the variable appears in the list

Aka(1) corresponds to / is also known as age

Aka(2) corresponds to / is also known as gender

Aka(3) corresponds to / is also known as race

Aka(4) corresponds to / is also known as ethnicity

In the CONTENTS procedure, the variables age, gender, race and ethnicity appear but not the aliases. Both age and Aka(1) retrieve values from the same memory location. So, when you change the value of Aka(1), the value of age changes.

If the array elements are not specified, then SAS does create new variables;

```
ARRAY b(4);
```

The above statement creates 4 new numeric variables named b1, b2, b3, and b4. These will show up in a proc contents.

Array elements are variables and an array can consist of either character or numeric variables, but not both in the same array. They must be the same type; either all numeric or all character.

The examples with the Aka array and the b array above are arrays consisting of numeric variables. Below is an example of an array consisting of character variables.

```
ARRAY x(3)  $5.  textvar1 textvar2 textvar3 ('aa', 'bbbbb', 'ccc');
```

You must use $ when the collection consists of character values.

SAS allows you to set starting values for the first traversing of the loop, regardless of whether the variables had existing values prior to the array statement.

The array statement allows a length to be set for previously unreferenced variables. In this example all three variables textvar1, textvar2 and textvar3 have a length of 5. If the length of 5 had not been specified, and none of the variables been referenced earlier, textvar1 would have a length of 2, textvar2 would have a length of 5 and textvar3 would have a length of 3; i.e. the would have the length of their initial assignment.

The special reserve word _NUMERIC_ creates an array consisting of all the numeric variables in the dataset.

```
ARRAY numlist (*)  _NUMERIC_;
```

Similarly, _CHARACTER_ creates an array of all the dataset's character variables.

```
ARRAY charlist {*}  _CHARACTER_;
```

```
ARRAY ADL [*]  Phone Drive PrepMeals Shop Dress Bathe;
```

The asterisk instructs SAS to count the number of specified array elements to decide the dimension. This is particularly useful in long lists and with the _NUMERIC_/_CHARACTER_ options. You may use (), {} or [] to reference the dimension of the array.

```
ARRAY x(2,3) var1-var6;
```

SAS allows multidimensional arrays; simply declare the dimensions separated by commas. In this case, SAS creates a matrix array with 2 rows and 3 columns for the six numeric variables var1-var6.

ARRAYS AND LOOPS TOGETHER

Array and loops, (like peanut butter and chocolate!), were made to be together. Since the array is a numbered list, SAS can process the whole collection in a DO loop.

Survey software enables the capture of respondent's answers, immediately producing an electronic dataset. However, you still need to check and clean your data before running any statistical descriptives or analyses. Questions that are refused, skipped, or otherwise missing can be coded in a variety of ways (depending on the software package) which may or may not be compatible with SAS.

Many data entry forms use special values like 999 as a placeholder for missing values. However, SAS uses . (dot) as an indicator for missing numeric values. For SAS to correctly perform calculations, you need to update the 999 values to . (dot). The inclusion of 999 instead of missing can influence even simple statistics like means.

You could write out the following code:

```
If age=999 then age=.;
If gender=999 then gender=.;
If race=999 then race=.;
If ethnicity=999 then ethnicity=.;
```

The problem with this approach is it requires a line of code for each variable in the dataset. For four variables this is very do-able, for forty variables copy and paste is your best friend, for four hundred it is downright tedious. Arrays and loops can make the reassignment much easier.

First recognize the pattern

```
If age            =999 then age           =.;
If gender         =999 then gender        =.;
If race           =999 then race          =.;
If ethnicity      =999 then ethnicity     =.;
```

The pattern is If _____=999 then _____ =.;

For each line, you plug in the next variable.

The first line plugs in the variable age.

The second line plugs in the variable gender.

The third plugs in race and the fourth plugs in ethnicity.

This pattern suggests the use of an array in a DO loop.

```
ARRAY  Aka{4}  age gender race ethnicity;
DO i=1 TO 4;
   IF Aka(i)=999 THEN Aka(i)=.;
END;
```

SAS executes the loop four times, evaluating the line in the body of the loop:

Loop Value of i	Plug in for i IF a(i)=999 THEN a(i)=.;	Equivalent statement replacing the alias/nickname with the array element/variable
1	`IF Aka(1) = 999 THEN Aka(1) = .;`	`IF age = 999 THEN age = .;`
2	`IF Aka(2) = 999 THEN Aka(2) = .;`	`IF gender = 999 THEN gender = .;`
3	`IF Aka(3) = 999 THEN Aka(3) = .;`	`IF race = 999 THEN race = .;`
4	`IF Aka(4) = 999 THEN Aka(4) =.;`	`IF ethnicity = 999 THEN ethnicity = .;`

Table 1 Steps SAS uses when processing an array in a loop.

In this simple example, you are replacing four if-then statements with four lines of code using an array and an iterative DO loop – not very exciting. So, what is the hoopla for the loop? This basic structure could be changed to 40 or 400 variables by simply adding the variable names to the ARRAY statement and changing the stopping value of the DO loop. Or if you want to change all the numeric variables, you could use the _NUMERIC_ keyword. It would still only take four lines of code!

```
ARRAY numlist (*)   _NUMERIC_;
DO j=1 TO DIM(numlist);
   IF numlist(j)=999 THEN numlist(j)=.;
END;
```

DIM ( ) is a function which returns the dimension of the specified array so the loop will process through all the numeric variables.

Another problem is often data imported from various data entry packages have ‘hidden’ characters like tabs, leading and trailing blanks. A common feature in Microsoft® products is the non-breaking space, designated in hexadecimal notation as 'A0'. Arrays can allow you to quickly clean up both problems.

```
ARRAY charlist (*) $  _CHARACTER_;
DO j=1 TO DIM(charlist);
     IF index(charlist(i), 'A0'x)>0 THEN
              charlist(i)=TRANWRD(charlist(i),'A0'x, '');
```

```
    Charlist(i)=strip(charlist(i));
END;
```

The loop processes each character variable in the dataset. It checks to see if the value contains the hexadecimal notation 'A0'. If SAS finds any 'A0'x, SAS replaces all occurrences in that variable with ''. In the second statement in the loop, the strip function removes all leading and trailing blanks.

*Careful here – index(charlist(i),'A0'x)>x is the **function** index, not the index of the DO loop! The index function returns the position of the first occurrence of the target string 'A0' x.

OTHER LOOPING CONSTRUCTS

In SAS there is rarely only one way to accomplish a task. The next example shows a DO OVER construct which is a variation of the DO loop. It drops the explicit index and corresponding dimension. SAS processes the loop statements for all the array elements in the array. Changing all the character variables to proper case is a snap.

```
ARRAY charlist $ _CHARACTER_;
DO OVER charlist;
   Charlist=PropCase(charlist);
END;
```

Loop constructs found in other programming languages are also available in SAS include the DO… WHILE and the DO... UNTIL statements. In the DO... WHILE loop SAS evaluates the stopping condition prior to the start of each iteration. So, it is possible the loop is never executed. In the DO... UNTIL loop, SAS always executes the loop once since SAS evaluates the stopping condition at the end of the loop. Use both constructs with extreme care since programming can inadvertently result in endless loops where the stopping condition is never reached.

```
ARRAY Aka(4) age gender race ethnicity;
i=1;
DO WHILE (i<5); *The stopping condition is checked prior to start of each
                iteration;
   IF Aka(i)=999 then Aka(i)=.;
   i=i+1;  *If this line was left out, the program would get stuck in an
            endless loop;

END;


ARRAY Aka(4) age gender race ethnicity;
I=1;
DO UNTIL (i=5);
   IF Aka(i)=999 then Aka(i)=.;
   i=i+1;  *Vital for the stopping condition to be met;
END;
```

CONCLUSION

Arrays and loops are an efficient way to process repetitive statements. Using arrays and loops, you can process many variables quickly and efficiently with a few lines of code. When you find yourself cutting

and pasting the same snippet of code over and over with only small modifications, consider barreling through doing the Loop de Loop with arrays and loops instead.

.

REFERENCES

1. SAS OnlineDoc® documentation DO Statement, Iterative Accessed August 16,2018 http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/viewer.htm#a000201276.htm

2. SAS OnlineDoc® documentation ARRAY Statement Accessed August 16,2018

   http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/viewer.htm#a000201956.htm

CONTACT INFORMATION

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

Jennifer Lindquist
Durham Veteran Affairs Health Care System
Jennifer.Lindquist@va.gov