Paper 6406-2016

An Introduction to SAS® Arrays

Andrew T. Kuligowski, HSN
Lisa Mendez, Ph.D., IMS Government Solutions

ABSTRACT

So, you've heard about SAS® arrays, but are not sure when – or why – you would use them. This presentation will provide the attendee / reader with a background in SAS arrays, from an explanation as to what occurs during compile time through to their programmatic use, and will include a discussion regarding how DO-loops and macro variables can enhance array usability. Specific examples, including Fahrenheit to Celsius temperature conversion, salary adjustments, and data transposition / counting will assist the user with effective use of SAS arrays in their own work, and provide a few caveats as to their usage, as well.

INTRODUCTION

What is an array? An array refers to a set of numbers, characters, or objects that are grouped together for ease of reference. For example, the months of the year form an array (Jan, Feb, Mar, etc.). The days contained in each month can also be considered to be an array (1, 2, 3, 4, 5, etc.). An array is an orderly arrangement of variables; collected in rows, columns, or in a matrix (multi-dimensional). Knowing when to use an array can help your SAS code be more efficient and less prone to errors. Arrays can be used to cut down the number of statements.

OVERVIEW OF SAS ARRAYS

In SAS, an array is a temporary group of variables arranged in a specific order which only exists in the current data step. There are some things to remember about arrays: 1) An array is not a new data structure, 2) the array name is not a variable, and 3) with limited exception, arrays do not define additional variables. Instead, an array provides a different name to collectively reference a group of variables.

SAS arrays contain variables that share certain characteristics; for example, they are of the same type (character or numeric). They are referenced as either character arrays or numeric arrays. In the event that variables associated with an array have not been explicitly defined elsewhere in the DATA step; SAS will implicitly create them. As usual, variables will default to numeric unless specifically defined as character in the array statement.

One of the main reasons arrays are utilized is for efficiency. They are great when performing repetitive calculations, to create variables with the same attributes, to restructure data, and to perform table lookups. Arrays are optional. You can program a Data Step to create output without using arrays, but arrays can cut down on the amount of lines it takes to get to the same result. It should be noted that arrays are not permanent data structures – arrays only exist for the duration of the DATA step in which they are defined. If an array is to be used in a subsequent DATA step, it must be redefined in that DATA step.

THE ARRAY STATEMENT

In order to use an array, it must be defined in the Data Step. The array statement looks like this:

```
array array-name {n} <$> <length> array-elements, (initial values) ;
```

Let's break down the statement:

1. **ARRAY** – SAS keyword to identify that the statements following the keyword define the array
2. array-name – any valid SAS name (but cannot be the same name as any variable on the SAS data set)
3. {n} – number of elements within the array (brackets, square brackets, or parenthesis can be used). An asterisk can be used when a specific number of variables are unknown. SAS will count the variables for you. The asterisk is used with one of the special variables to define the elements <$> - indicates that the elements are character type variables. If not used, the default is numeric.
4. <length> – optional, specifies a common length for the array elements. The default of array variables or other elements in the array is a length of 8 bytes.

5. array-elements – list of SAS variables to be included in the array (the group). All variables within the group must be either character or numeric.
6. (initial values) – provides the initial values for each of the array elements. This is optional.

Here is a sample of the array statement in action:

```
array cities {4} $10 ('New York' 'Los Angeles' 'Dallas' 'Chicago');
```

The array statement above declares an array named "cities" that has four elements. The value of each element is 10 characters in length. Since no variables are specified in the ARRAY statement, SAS will generate a series of 4, named cities1, cities2, cities3, and cities4, with the initial values of New York, Los Angele, Dallas, and Chicago. (Note that the letter "s" has been cut off from the second city's name "Los Angeles" – including the space, there are a total of 11 characters in the value, which exceeds the specified 10 character length.)

Alternatively, an array can be defined using specified variables, such as:

```
array states {4} state01-state4;
```

Note also that it is not necessary to include the "$" to indicate character values when the variables have been pre-defined as character.

THE ARRAY REFERENCE

In order to reference an element within an array, use `array-name {n}`. The value of n is the element's position within the array. It looks something like this: `cities {3}`. The reference would point to the element with the value "Dallas" in the sample above.

THE ARRAY BOUNDS

By default, SAS arrays begin with subscript 1. The lower bound defaults to 1 and the upper bound is the number of array elements. In the "cities" array, the lower bound would be 1, and the upper bound would be 4. Understanding the bounds is important when other programming methods, such as do loops, are used in conjunction with arrays.

THE ARRAY STATEMENT AT COMPILE TIME

What happens at compile time when an array is declared? An array statement is a compile-time only statement. That means that the statement provides information to the Program Data Vector (PDV) and cannot (except with the macro language) be conditionally executed. Placement of the array statement is critical because the attributes of the variables within the array are determined by the first reference to the compiler (Howard, pg.1).

DO LOOP PROCESSING

In order to use an array efficiently, many times you will use them in conjunction with a Do Loop. An iteration Do Loop will use the array and assign the value to the array variable.

The standard syntax for iteration Do Loop is

DO <index variable> **=** <start variable> **TO** <stop variable> <BY <increment value>>;
    SAS commands;
END;

Where

DO	is a keyword that causes the SAS commands to be repeated
<index variable>	must be a SAS variable
=	sets the index variable to the start value
<start value>	can be a numeric constant or a numerical variable
TO	compares the index variable to the stop value. If the index variable does not exceed the stop value, SAS performs the statements in the loop. If the index

	variable exceeds the stop value, SAS exits the Do Loop by going to the next statement after END;
<stop value>	can be a numeric constant or a numeric variable
<BY <increment value>>	changes the index variable by the amount of the increment value (or by 1 if no increment value is specified.) The increment value can be negative.

After SAS performs the *SAS commands* it returns to the top of the loop, the *index variable* is changed by the amount of the *increment value* (by 1 is the default), and the loop continues until the *index variable* is past the *stop value*.

WHEN TO USE AN ARRAY

There are many scenarios where an array can help you be more efficient. If you find yourself applying the same programming logic to many of the same type of variables, you may be able to use an array. Sometimes it might be difficult to notice when an array could be beneficial. Let's look at a few scenarios to detail how an array can be implemented easily and efficiently.

SCENARIO ONE – CONVERT PATIENT TEMPERATURES FROM FAHRENHEIT TO CELSIUS

You have a data set that has numerous temperatures that were recorded in Fahrenheit degrees; however the person who will be using this data set wants the temperatures in Celsius. There are 24 temperature recordings. You can either code 24 lines of code that will convert each temperature variable, or you can use an array in conjunction with a Do Loop to efficiently convert them.

The first thing we need to do is declare the array. Then, we will write the Do Loop to handle the variables within the array. Third, we will do a little clean up of the data set to drop a few variables that we will not need going forward.

SCENARIO ONE SAS CODE

```
/*-------------------------------------------------------------------------------*/
/*     Convert Fahrenheit Temps to Celsius Temps                                 */
/*-------------------------------------------------------------------------------*/
❶ Data Patient_Temps_Converted (Drop = i);
❷      Set Patient_Temps;

       /* Declare arrays here */
❸      array temps {24} temp1-temp24;
❹      array temps_C {24} temp_C1-temp_C24;

       /* Do Loop to convert the Fahrenheit temps to Celsius */
❺      do i = 1 to 24;
❻            temps_C(i) = 5/9*(temps(i) - 32) ;
❼      end;

❽ run;
```

SCENARIO ONE SAS CODE EXPLAINED

1. Create a data set named Patient_Temps_Converted and drop the variable i
2. Read in the data set named Patient_Temps;
3. Declare the array that will hold the original Fahrenheit temperatures. The array is named temps and will have 24 data elements. Each data element will be named temp1, temp2, temp3…etc. Since there is not a dollar sign ($) the values of the data elements are numeric.
4. Declare the array that will hold the new Celcius temperatures. The array is named temps_C and will have 24 data elements. Each data element will be named temp_C1, temp_C2, temp_C3…etc. Since there is not a dollar sign ($) the values of the data elements are numeric.
5. Begin the Do Loop statement.

6. Convert the Fahrenheit value to Celsius value by referencing the data elements within both arrays. Basically, the line is assigning the value of the data element in the array temps_C by using the value of the data element in the temps array in a calculation.
7. Keyword to end the Do Loop
8. Keyword run; to execute the Data Step.

SCENARIO TWO – CONVERT PATIENT TEMPERATURES FROM FAHRENHEIT TO CELSIUS FOR ONLY THOSE TEMPERATURES TAKEN IN THE DAYTIME

After you changed the temperatures from Fahrenheit to Celsius you are told that only the daytime temperatures need to be converted. No worries, you just use a *range* within your array declaration. Once you determine what constitutes "daytime" you can then determine which range of variables you need to convert. This will allow you to only change those temperatures within a given range.

SCENARIO TWO SAS CODE

```
/*-------------------------------------------------------------------------------------*/
/*     CONVERT FAHRENHEIT TEMPS TO CELSIUS TEMPS FOR DAYTIME                           */
/*-------------------------------------------------------------------------------------*/
Data Patient_Temps_Daytime_Converted (Drop = i);
       Set Patient_Temps;

       /* Declare arrays with range */
❶      array temps {6:18} temp6-temp18;
❷      array temps_C {6:18} temp_C6-temp_C18;

       /* Write code for do loop */
❸      do i = 6 to 18;
              temps_C(i) = 5/9*(temps(i) - 32) ;
       end;

 run;
```

SCENARIO TWO SAS CODE EXPLAINED

1. Declare the array that will hold the original Fahrenheit temperatures. The array is named temps and will have 13 data elements. Each data element will be named temp6, temp7, temp8…etc up to temp18. Using {6:18} we declare a range beginning at the sixth element through the 18th element. Since there is not a dollar sign ($) the values of the data elements are numeric.
2. Declare the array that will hold the new Celcius temperatures. The array is named temps_C and will have 13 data elements. Each data element will be named temp_C6, temp_C7, temp_C8…etc up to temp_C18. Using {6:18} we declare a range beginning at the sixth element through the 18th element. Since there is not a dollar sign ($) the values of the data elements are numeric.
3. Begin the Do Loop statement. The start value is 6 and the end value is 18. This will ensure only those elements will be converted.

SCENARIO TWO SAS CODE – NOT RECOMMENDED, BUT WILL WORK

```
/*-------------------------------------------------------------------------------------*/
/*     CONVERT FAHRENHEIT TEMPS TO CELSIUS TEMPS FOR DAYTIME                           */
/*-------------------------------------------------------------------------------------*/
Data Patient_Temps_Daytime_Converted (Drop = i);
       Set Patient_Temps;

       /* Declare arrays with range */
❶      array temps {13} temp6-temp18;
❷      array temps_C {13} temp_C6-temp_C18;

       /* Write code for do loop */
❸      do i = 1 to 18;
              temps_C(i) = 5/9*(temps(i) - 32) ;
       end;
```

```
run;
```

SCENARIO TWO NOT RECOMMENDED SAS CODE EXPLAINED

1. Declare the array that will hold the original Fahrenheit temperatures. The array is named temps and will have 13 data elements. Each data element will be named temp6, temp7, temp8…etc up to temp18. Using {13} we declare 13 data elements beginning at the sixth element through the 18th element. Since there is not a dollar sign ($) the values of the data elements are numeric.
2. Declare the array that will hold the new Celcius temperatures. The array is named temps_C and will have 13 data elements. Each data element will be named temp_C6, temp_C7, temp_C8…etc up to temp_C18. Using {13} we declare 13 data elements beginning at the sixth element through the 18th element. Begin the Do Loop statement. The start value is 6 and the end value is 18. This will ensure only those elements will be converted.
3. Begin the Do Loop statement. The start value is 1 and the end value is 18. This will ensure only 13 elements will be converted.

This approach will work; however, it may make future maintenance unnecessarily challenging. We show it here as an example that it CAN be done, but coding best practices should be followed to ensure easy maintenance in case of future modifications.

SCENARIO THREE – INCREASE SALARY VALUES BY 20 PERCENT

You have a data set that has salary information for the company's sales force. Due to a steady growth in revenue, the company has decided to give all current sales force personnel a 20 percent increase. You could write out a program that will take each salary variable and add 20 percent (which is fine if you have a handful of employees), or you can declare an array to take each of the salary variables, use a do loop, and easily increase the values. Using an array helps streamline the programming process if you have many employees. This also allows you to keep the original salary values before the increase was applied.

SCENARIO THREE SAS CODE

```
/*-------------------------------------------------------------------------------*/
/*     INCREASE SALARY VALUES BY 20 PERCENT                                      */
/*-------------------------------------------------------------------------------*/
Data salary_increased;
   Set salary;
       /* Declare array(s) here */
❶      array salaries {5} admin clinical direct_care_prof direct_care_paraprof rn  ;

       /* Do Loop to calculate increase */
       do i = 1 to 5;
              if salaries(i) gt 0 then
❷                    salaries(i) = salaries(i) + (salaries(i) * .2) ;
       end;

 run;
```

SCENARIO THREE SAS CODE EXPLAINED

1. Declare the array that will hold the original salary values. The array is named salaries and will have 4 data elements. Each data element will be named admin, clinical, direct_care_prof, direct_care_paraprof, and rn. Since there is not a dollar sign ($) the values of the data elements are numeric.
2. If the salary value is greater than zero, then increase the salary by 20%.

SCENARIO FOUR – COUNT THE NUMBER OF VISITS FOR EACH CLINIC FOR EACH FISCAL MONTH/YEAR

You have a data set that has total visit values for each fiscal month and year for multiple clinics. Each record indicates how many visits were complete for a given fiscal month and fiscal year at each clinic. You can think of this as a vertical view. The problem is you want to see a summary of results. You want each clinic and the clinic's total visits for each month. You want to see each clinic on one row with each fiscal month in columns. Think of the results as a horizontal view.

First, let's take a look at the data set. It has four variables (Fiscal_Year, Fiscal_Month, Parent_DMIS, and Total_Visits). The Parent_DMIS_ID indicates a specific clinic. There are three different clinics.

This is how your data looks:

	Fiscal_Year	Fiscal_Month	Parent_DMIS_ID	Total_Visits
1	2011	4	117	55875
2	2011	1	28	11154
3	2011	8	28	10578
4	2011	6	37	72344
5	2011	9	37	59292
6	2011	1	117	55820
7	2011	7	117	58124
8	2011	3	28	9378
9	2011	5	28	9665
10	2011	4	37	60583
11	2011	9	117	57490
12	2011	10	117	47023
13	2011	3	37	56505
14	2011	7	37	63431
15	2011	8	37	59233
16	2011	3	117	52231
17	2011	2	28	11253
18	2011	6	28	12198
19	2011	2	117	50616
20	2011	8	117	57813
21	2011	12	117	49041
22	2011	11	28	11644
23	2011	5	37	61118
24	2011	5	117	53643
25	2011	11	117	53955
26	2011	4	28	10406
27	2011	10	28	9410
28	2011	1	37	64072
29	2011	7	28	10720
30	2011	9	28	10681
31	2011	12	28	12098
32	2011	2	37	60974
33	2011	6	117	65948

This is how you want your data to look:

	Fiscal_Year	Parent_DMIS_ID	fm1	fm2	fm3	fm4	fm5	fm6	fm7	fm8	fm9	fm10	fm11	fm12
1	2011	28	11154	11253	9378	10406	9665	12198	10720	10578	10681	9410	11644	12098
2	2011	37	64072	60974	56505	60583	61118	72344	63431	59233	59292	.	.	.
3	2011	117	55820	50616	52231	55875	53643	65948	58124	57813	57490	47023	53955	49041

By using an array, you can transpose your data. The key to the following code is using an array with a Do Loop and a Proc Means, which will consolidate and summarize the data.

SCENARIO FOUR SAS CODE

```
/*-------------------------------------------------------------------------------------*/
/*     COUNT THE NUMBER VISITS FOR EACH MONTH FOR EACH CLINIC                          */
/*-------------------------------------------------------------------------------------*/
Data Visits_Transpose (drop = i) ;
       Set Total_Visits;

       /* Declare array(s) here */
       ❶ array FMonths {12} fm1-fm12;

       /* Do Loop to create a count */
       do i = 1 to 12;
              ❷ if Fiscal_Month = i then
                     FMonths(i) = total_visits;
       end;

run;

/*-------------------------------------------------------------------------------------*/
/*     CONDENSE THE OBSERVATIONS                                                       */
/*-------------------------------------------------------------------------------------*/
❸ PROC MEANS DATA = VISITS_TRANSPOSE NWAY NOPRINT;
       CLASS  FISCAL_YEAR PARENT_DMIS_ID;
       OUTPUT OUT = VISITS_TRANSPOSE_2 (DROP = _TYPE_ _FREQ_ FISCAL_MONTH
          TOTAL_VISITS) SUM=;
```

```
RUN;
```

SCENARIO FOUR SAS CODE EXPLAINED

1. Declare the array that will hold the fiscal month values. The array is named FMonths and will have12 data elements. Each data element will be named fm1, fm2, fm3, etc. Since there is not a dollar sign ($) the values of the data elements are numeric.
2. If the Fiscal_Month variable's value equals the do loop value (i) then the first element in the FMonths array equals the value of the total_visits variable.
3. The Proc Means procedure is used to condense the variables for each fiscal month. In brief, the Class statement indicates the variables to condense. The Output Out= statement defines the output data set. The Sum= statement instructs the procedure to sum up all of the numeric values by Fiscal_Year and Parent_DMIS_ID. (As Class variables, Fiscal_Year and Parent_DMIS_ID are exempt from SUM=.)

SCENARIO FOUR(A1) – ALTERNATIVE SAS CODE

```
/*-------------------------------------------------------------------------------*/
/*     COUNT THE NUMBER VISITS FOR EACH MONTH FOR EACH CLINIC                    */
/*-------------------------------------------------------------------------------*/
Data Visits_Transpose (drop = i) ;
       Set Total_Visits;

       /* Declare array(s) here */
         array FMonths {12} fm1-fm12;

       /* Do Loop to create a count */
       do i = 1 to 12;
              ❶  if Fiscal_Month = i then do;
                   FMonths(i) = total_visits;
              ❷     Leave;
              ❷     i = 13;
              ❶  end;
       end;
run;
       <subsequent SAS code removed for clarity>
```

Select *either* of these two statements – *not both.*

SCENARIO FOUR(A1) SAS CODE EXPLAINED

1. Create an IF/THEN DO loop rather than a single IF/THEN statement, in order to execute multiple conditional statements.
2. Once the value of i corresponding to the value of Fiscal_Month is identified, there is no need to process the rest of the values of i. The LEAVE statement will cause SAS to transfer processing out of existing loops, while the i = 13 assignment will cause the DO i loop to reach its upper bound and terminate. Either statement will trigger this action; they would be redundant if used together.

SCENARIO FOUR(A2) – ALTERNATIVE SAS CODE

```
/*-------------------------------------------------------------------------------*/
/*     COUNT THE NUMBER VISITS FOR EACH MONTH FOR EACH CLINIC                    */
/*-------------------------------------------------------------------------------*/
Data Visits_Transpose ;
       Set Total_Visits;

       /* Declare array(s) here */
         array FMonths {12} fm1-fm12;

     ❶  FMonths(Fiscal_Month) = total_visits;
run;
       <subsequent SAS code removed for clarity>
```

SCENARIO FOUR(A2) SAS CODE EXPLAINED

❶ Skip the DO loop altogether and simply use the value of Fiscal_Month directly as the index to the array. This is the most efficient way to handle the problem. HOWEVER, the DO Loop provides some data validation – any value for Fiscal_Month other than the range of whole numbers between 1 and 12 will be ignored. In this "efficient" alternative, a value that does not correspond to one of the 12 valid values will cause the Data step to abort.

SCENARIO FIVE – LETTING SAS DETERMINE SUBSCRIPT

All of the examples that have been discussed to this point have involved the coder having to specify the array dimension to SAS. The astute programmer may wonder why this is necessary when a list of elements is specified; after all, it is inherently obvious how large an array is based on the number of elements that were provided. Assuming that there are no complications in the request, such as requesting that SAS create an array of Temporary variables or requesting a multi-dimensional array (both topics will also be covered in this paper), it is indeed possible to let SAS determine the size of the array. By using the asterisk (*) within the array statement, SAS will count the variables to set the size of the array.

SCENARIO FIVE – SAS CODE

```
/*---------------------------------------*/
/* COUNT # MONTHS A PROCEDURE WAS PERFORMED */
/*---------------------------------------*/
Data Month_Count (drop = i) ;
   Set Visits_Transpose_2;
    /* INITILIZE CPT_COUNT */
   FM_count = 0;
    /* DECLARE ARRAY(S) HERE */
❹   array FMvalues {*} fm1-fm12;
    /* DO LOOP TO CREATE A COUNT */
   do i = 1 to 12;
❻     if FMvalues(i) ^= . then
         FM_count  = FM_count + 1 ;
   end;
run;
```

SCENARIO FIVE - SAS CODE EXPLAINED

❹ SAS will count the variables provided in the variable list, and will set the array size – in this instance, to 12.

❻ If array element is not null, increment the counter.

SCENARIO SIX – MULTIDIMENSIONAL ARRAY

So far, all of the examples have only involved a one-dimensional array. Sometimes, there is a need to examine things in two (or more) dimensions, creating a multi-dimensional array. (Picture a simple cross-tab in your mind as you read this, and the numerous reasons why you may want to process data stored in one.) SAS will allow you to generate a multidimensional array.

SCENARIO SIX – SAS CODE

```
/*---------------------------------*/
/* ACCUMULATE DAY-BY-DAY RESULTS   */
/*---------------------------------*/
DATA MONTH_COUNT (DROP = I) ;
   SET VISITS_TRANSPOSE_2;

/* DECLARE ARRAY(S) HERE */
❸   ARRAY DAYRESULTS {12,31} DAY001-DAY365;
❹   ARRAY DAYPAYMENT {12,31} PAY001-PAY365;

/* DO LOOP TO CREATE A COUNT */
❺   DAYRESULTS( MONTH(VISITDT), DAY(VISITDT) ) + 1;
❻   DAYPAYMENT( MONTH(VISITDT), DAY(VISITDT) ) =
```

```
      SUM(DAYPAYMENT( MONTH(VISITDT), DAY(VISITDT) ),
          PATIENTBILL );
   END;
RUN;
```

SCENARIO SIX - SAS CODE EXPLAINED

❸ and ❹ Declare 12 by 31 arrays (corresponding to 12 months, with 31 days in longest months).
To include Leap Year, use `DAY001-DAY366;`
12 x 31 does not allow for flexible days per month.

It is important to point out here that the example above ***WILL NOT WORK AS WRITTEN***. SAS will error out, producing the following message in the SASLOG for each affected array:

ERROR: Too few variables defined for the dimension(s) specified for the array <arrayname>.

The issue is simple: A table dimensioned at 12 by 31 contains a total of 372 elements, but the example only provides 365 contributing variables. We could expand the request to include "DAY001-DAY372", or we could add extra variables after DAY365, such as DUMMY1-DUMMY7; either solution would work.

However the issue is resolved, it should be noted that the array defined above will not line up to Julian Dates, as one might expect on close reflection. Elements (1,1) to (1,31) will correspond to January's Julian dates 001 to 031, and, (2,1) to (2,28) will correspond to February's dates 032 to 059. However, the next date, 060, will be stored in array element (2,29) – NOT (3,1) as one might expect when mentally correlating the array with the calendar (at least in a non-leap year). This means that the variable suffix will not correspond to Julian Date on/after March 1st, which is not programmatically incorrect BUT may be confusing to people attempting to follow the logic and/or modify the code in the future.

❺ and ❻ Use functions to determine array element. By definition it will be 1-12 and 1-31, respectively.

Note: IMPORTANT SAFETY TIP: Do not give arrays the same name as SAS functions!

SCENARIO SEVEN – TRANSPOSE YEAR/DAY TO MONTH/DAY

Having reviewed the previous example, where a simple attempt to map a Julian date into a multidimensional array which is built to resemble a Gregorian calendar, it might be wondered how the code might be structured to actually create the illusion of a Gregorian calendar. There are several possible solutions, as there are with virtually any SAS coding problem. We will take the opportunity to introduce the concept of temporary arrays, where temporary variables are created to store a specific set of values that are only available during the execution of the DATA step in question.

SCENARIO SEVEN – SAS CODE

```
/*-----------------------------------*/
/* TRANSPOSE YEAR/DAY TO MONTH/DAY   */
/*-----------------------------------*/
DATA DAY_RESULTS (DROP = I) ;
❷   ARRAY DAYRSLTS   {12,31} DAYKEEP001-DAYKEEP372;
❸   ARRAY MONTHDAYS  [12]    _TEMPORARY_
            ( 31 29 31 30 31 30 31 31 30 31 30 31 );
    RETAIN  MONTH  DAY  1;
    DO I = 1 TO 366;
      DAYRSLTS{MONTH,DAY} = FLOOR(RANUNI(0)*100) +1;
❼      IF  DAY = MONTHDAYS( MONTH )  THEN DO;
          MONTH = MONTH + 1;
          DAY   = 1;
       END;
     ELSE  DAY = DAY + 1;
   END;
 RUN;
```

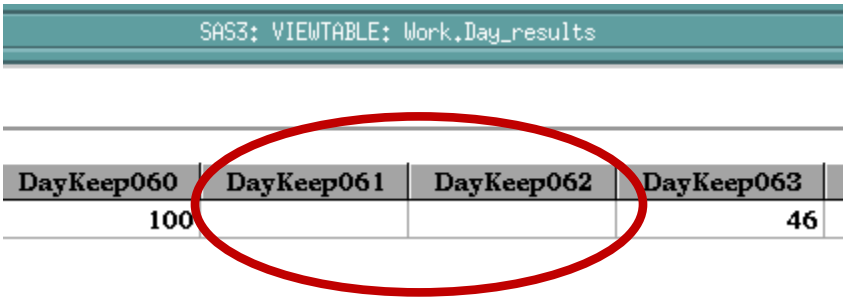
SCENARIO SEVEN - SAS CODE EXPLAINED

❷ Similar Year array to previous sample – no need to re-explain.

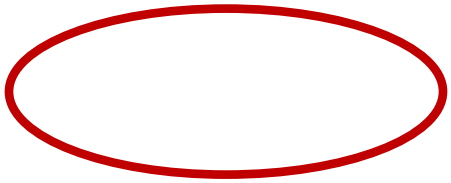
❸ Temporary array. Elements not specified nor written to permanent dataset.

❼ Note that different brackets may be interchanged when referring to the same array.

Note: The data set work.day results has 1 observation and 374 variables.



In the figure above you will see a gap – undefined values between values #60 and #63, which correspond to Julian Day 60 (Feb 29th) and 63 (Mar 1st).



Notice there are 37**4** variables – We did not keep the 12 elements of MONTHDAYS, *BUT* we forgot to drop indexes Month and Day, explaining the two extra variables.

CONCLUSION

SAS arrays, at their root, are a very basic concept – gather a bunch of things together under a common umbrella so that they can be dealt with in a similar and structured manner. Various options can be included to enhance the process based on particular needs.

It is hoped that this short presentation can provide the basis of the definition and use of arrays. Readers can then take these concepts and apply them to their own real-world situations. It is never possible to discuss every possible option, nuance, and flavor of definition and usage for any option; if the reader wishes to perform a task not defined in this presentation, they hopefully will have the foundation to perform additional research and experimentation in order to achieve their own desired results.

REFERENCES / RECOMMENDED READING

Howard, Neil (2004). "How SAS® Thinks or Why the DATA Step Does What It Does". *Proceedings of the Twenty-Ninth Annual SAS Users Group International Conference.* Cary, NC: SAS Institute, Inc.
http://www2.sas.com/proceedings/sugi29/252-29.pdf

Li, Arthur (2004). "The Many Ways to Effectively Utilize Array Processing". *Proceedings of the SAS Global Forum 2011 Conference.* Cary, NC: SAS Institute, Inc.
http://support.sas.com/resources/papers/proceedings11/244-2011.pdf

Mendez, Lisa and Alonzo, Lizette (2011). "SAS® WOW! How to Streamline Your SAS Programs by Shedding Lines and Adding Substance". *Proceedings of the SAS Global Forum 2011 Conference.* Cary, NC: SAS Institute, Inc.

http://support.sas.com/resources/papers/proceedings11/060-2011.pdf

SAS Institute, Inc. (2015), *SAS 9.4 Language Reference: Concepts, Fifth Edition.* Cary, NC: SAS Institute, Inc. https://support.sas.com/documentation/cdl/en/lrcon/68089/HTML/default/viewer.htm#titlepage.htm

SAS Institute, Inc. (2015), *SAS 9.4 Statements: Reference, Fourth Edition.* Cary, NC: SAS Institute, Inc. https://support.sas.com/documentation/cdl/en/lestmtsref/68024/HTML/default/viewer.htm#titlepage.htm

SAS Institute, Inc. (2011), *"Using Arrays in SAS® Programming".* Cary, NC: SAS Institute, Inc. https://support.sas.com/resources/papers/97529_Using_Arrays_in_SAS_Programming.pdf

Waller, Jennifer (2010). "How to Use ARRAYs and DO Loops: Do I DO OVER or Do I DO i?". *Proceedings of the SAS Global Forum 2011 Conference.* Cary, NC: SAS Institute, Inc. http://support.sas.com/resources/papers/proceedings10/158-2010.pdf

ACKNOWLEDGMENTS

Special thanks go out to the individuals who believed in the concept of this paper and asked that it be fleshed out and presented. This includes Misty Johnson and Deanna Schreiber-Gregory of the Midwest SAS Users Group, Joni Shreve at Louisiana State University and the South Central SAS Users Group, Lizette Alonzo the Co-Chair of the 2016 South Central SAS Users Group, and Pete Lund at SAS Global Forum.

CONTACT INFORMATION

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

Andrew Kuligowski
HSN
St. Petersburg, FL
E-mail: KuligowskiConference@gmail.com

Lisa Mendez
IMS Government Solutions
San Antonio, TX
E-mail: lmendez@us.imshealth.com