Paper 3259-2015

You deserve ARRAYs; How to be more efficient using SAS®!

Kate Burnett-Isaacs, Statistics Canada

ABSTRACT

Everyone likes getting a raise, and using arrays in SAS® can help you do just that! Using arrays simplifies processing, allowing for reading and analyzing of repetitive data with minimum coding. Improving the efficiency of your coding and in turn, your SAS productivity, is easier than you think! Arrays simplify coding by identifying a group of related variables that can then be referred to later in a DATA step. In this quick tip, you learn how to define an array using an ARRAY statement that establishes the array name, length, and elements. You also learn the criteria and limitations for the ARRAY name, the requirements for array elements to be processed as a group, and how to call an array and specific array elements within a DATA step. This quick tip reveals useful functions and operators, such as the DIM function and using the OF operator within existing SAS functions, that make using arrays an efficient and productive way to process data. This paper takes you through an example of how to do the same task with and without using arrays in order to illustrate the ease and benefits of using them. Your coding will be more efficient and your data analysis will be more productive, meaning you will deserve arrays.

INTRODUCTION

When faced with repetitively processing a large set of variables in SAS® we seek a way to reduce our effort, our chance of errors and the time to carry out our task. We can use arrays to make such tasks less daunting and, more importantly, achievable with minimal coding. Arrays can replace repetitive coding by identifying a group of related variables that can then be referred to and processed later in a DATA step, and so are particularly useful when applying a repetitive process to a large set of variables. Arrays are easy to establish and can save time and reduce errors, making the output more reliable and make the programmer more deserving of a raise. In this Quick Tip paper, you will learn how to establish arrays, when and how to use them and tips on getting more comfortable with their use.

WHAT ARE ARRAYS AND WHY USE THEM?

An array is a group of related variables defined in a DATA step (First and Schudrowitz, 2005). Arrays are extremely useful when a repetitive process, such as a calculation, transformation or function, needs to be applied to a large set of variables. This is because an array enables the programmer to set a list of variables once and then refer this list, or elements within the list, any number of times within a single DATA step.

ESTABLISHING THE ARRAY

We define an array in an ARRAY statement within the DATA step, and this array only exists for the DATA step in which it is defined. It unfortunately cannot be used in any subsequent DATA step. Within the DATA step, the array must be established after the SET statement and before it is referenced. The ARRAY statement is established by the keyword 'array', proceeded by defining an array name (array_name), the length of the array ({N}) and variables or array elements that it will contain (variable1... variableN), such as:

```
array array_name {N} variable1 variable2… variableN;
```

For simplicity, if the array elements are named with a numeric suffix, we can establish our array elements by hyphenating the first and last elements:

```
array array_name {N} variable1-variableN;
```

If no array elements are established in the ARRAY statement, SAS will create new array elements using the array name appended by a numerical suffix, beginning with 1 and ending with the length of the array (Keelan, 2002), such as:

```
array_name1 array_name2 … array_nameN
```

This is useful when we want to create new variables instead of referring to variables that already exist in our dataset.

Parentheses or square brackets around the length value, N, can also be used, however, braces, { }, are more commonly applied because they are not used in any other SAS statement (First and Schudrowitz, 2005).

The array name must follow SAS naming conventions and the array name cannot be the same as any name of the array elements or other variables in the SAS dataset from which the array elements are taken. Although we can use a function name as an array name, this is not recommended. It can give unpredictable results because SAS will not recognize the function name anymore, rather deem it an array name. In this situation, a warning message will be written to the log, but SAS will proceed to process the function name as an array name (First and Schudrowitz, 2005). It is best to avoid using function names as array names all together.

When setting the length of an array in the ARRAY statement, we can either set the length by putting the corresponding length value in the braces, or we can use an asterisk, *, in order for SAS to count the number of array elements for us. Using the asterisk to determine the array length is particularly useful when there is a large number of array elements that might be too large for us to effectively count.

Because all array variables are processed as a group, the defined variables must be either ALL numeric or ALL character. Unfortunately, we cannot mix types, however useful that may be! By default, each array element is set as a numeric variable (First and Schudrowitz, 2005). If we want an array of character variables, we include a dollar sign, $, or the term _CHARACTER_ in our ARRAY statement, like so:

```
array array_name {N} $ variable1 variable2 … variableN;
```

USING THE ARRAY

Now that we have established our array, we are going to want to use it. Otherwise, how will we get our raise? We can use an array to either refer to one particular element or to use the entire series of elements. For the first case, we will want to refer to a specific variable by using the array name and the position in the list of array elements of the variable in question. The number in the braces indicates the corresponding element in the array. For example, to call *variable2*, we want to use the following:

```
array_name {2};
```

To apply a repetitive process to a large set of variables, we will use the array in a DO loop in order to call and process the array elements in sequential order as they appear in the ARRAY statement:

```
do i=1 to N;
      array_name {i}=i;
end;
```

A helpful tool in the DO loop, especially in the case where we may not know the number of elements in our array until runtime, is to get SAS to count the total number of array elements by using the DIM function (SAS Institute Inc. 2011). The DIM function returns the number of elements in an array, allowing SAS to create an end value for the DO loop:

```
do i=1 to dim(array_name);
      array_name {i}=i;
end;
```

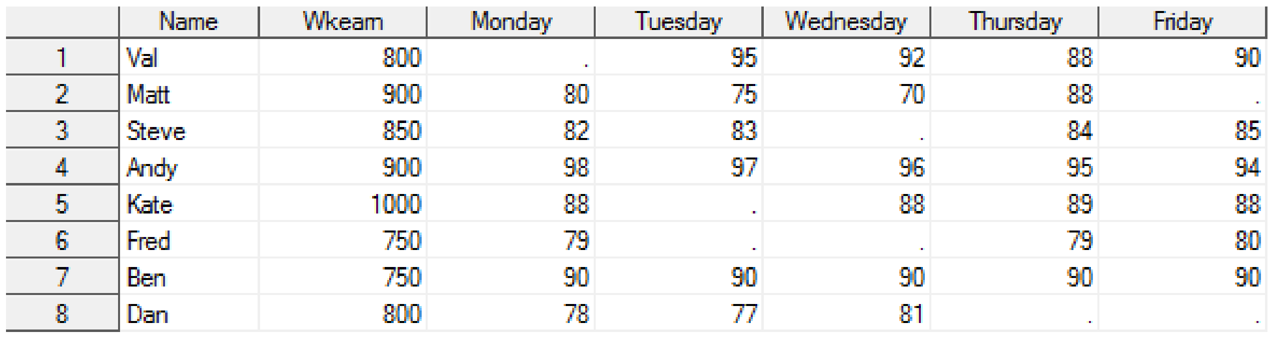
An array can also be useful when we want to use all of the array elements together, for example, in the calculation of the sum or mean of all the array elements. To do this, we need to use OF operator in front of the array name, and use the asterisk to signify we want to use all array elements; we can simplify our code (SAS Institute Inc. 2011):

```
      Total=sum(of array_name {*});
```

There are some limitations as to what we can do with arrays. Unfortunately, an array cannot be referenced in a FORMAT, LENGTH, DROP or KEEP statement in a DATA step (Keelan, 2002). If we want to use the array elements in these statements, we have to write them out individually.

AN EXAMPLE

The best way to understand how to use arrays is to go through an example. Let's consider the following scenario: you are an employer, whose employees took a week long SAS course. The dataset in Figure 1 shows you the percentage grade given to each employee on a given day of their SAS course. If they were absent, no grade is present.



	Name	Wkearn	Monday	Tuesday	Wednesday	Thursday	Friday
1	Val	800	.	95	92	88	90
2	Matt	900	80	75	70	88	.
3	Steve	850	82	83	.	84	85
4	Andy	900	98	97	96	95	94
5	Kate	1000	88	.	88	89	88
6	Fred	750	79	.	.	79	80
7	Ben	750	90	90	90	90	90
8	Dan	800	78	77	81	.	.

Figure 1: SAS course grades for each employee

To reward participation, each employee gets 2% of their weekly earnings for every day they attend the course. You want to show your employees how much they made each day of the course and what their total weekly salary will be at the end of the week long course.

We can do this in two ways. Using an IF-ELSE statement is perhaps the method that first comes to mind when addressing this problem. In this case, we would use an IF-ELSE statement for each day of the week in order to create a daily bonus variable (*bnsMon-bnsFri)* that reflects the amount earned by each employee for each day of the week they attended the SAS course. In order to determine the final weekly earnings (*fnlearn*), any daily bonuses attained will be added to the weekly earnings of each employee. The code to construct this dataset, using IF-ELSE statements, is as follows:

```
      data No_Arrays;
            set SAS_scores;
                  if NOT MISSING(Monday) then BnsMon=(Wkearn*0.02);
                        else BnsMon=0;
                  if NOT MISSING(Tuesday) then BnsTues=(Wkearn*0.02);
                        else BnsTues=0;
                  if NOT MISSING(Wednesday) then BnsWed=(Wkearn*0.02);
                        else BnsWed=0;
                  if NOT MISSING(Thursday) then BnsThurs=(Wkearn*0.02);
                        else BnsThurs=0;
                  if NOT MISSING(Friday) then BnsFri=(Wkearn*0.02);
                        else BnsFri=0;
                  fnlearn=wkearn+sum(BnsMon, BnsTues, BnsWed, BnsThurs,
      BnsFri);
            keep Name BnsMon BnsTues abWed BnsThurs BnsFri fnlearn;
      run;
```

This method is fairly tedious to code and error-prone. Imagine the number of IF-ELSE statements needed to conduct these calculations over several months or even a year, instead of a week! Whether we are processing five or hundreds of variables, it can be more efficiently coded using arrays. First we must define the arrays within the DATA step before referencing it. We will want to create two different arrays: one to refer to the set of day of the week variables (array_day) from our dataset and another array to define the daily bonus variables (bonus) that we will be creating. We will be using both arrays for our calculations, so we'll want all elements to be numeric:

```
array array_day {*} Monday Tuesday Wednesday Thursday Friday;
array bonus {*} BnsMon BnsTues BnsWed BnsThurs BnsFri;
```

Then, using a DO loop, we can easily create bonus variables with less code than previously used:

```
do i=1 to dim(array_day);
      if NOT MISSING (array_day {i}) then  bonus {i}=Wkearn*(0.02);
            else bonus {i}=0;
end;
```

Creating the final weekly earnings variable is also made simpler by an array, through the use of the OF operator within the summation function. The asterisk conveys that all of the variables in the *bonus* array will be included in the summation function:

```
fnlearn=(wkearn+sum(of bonus {*}));
```

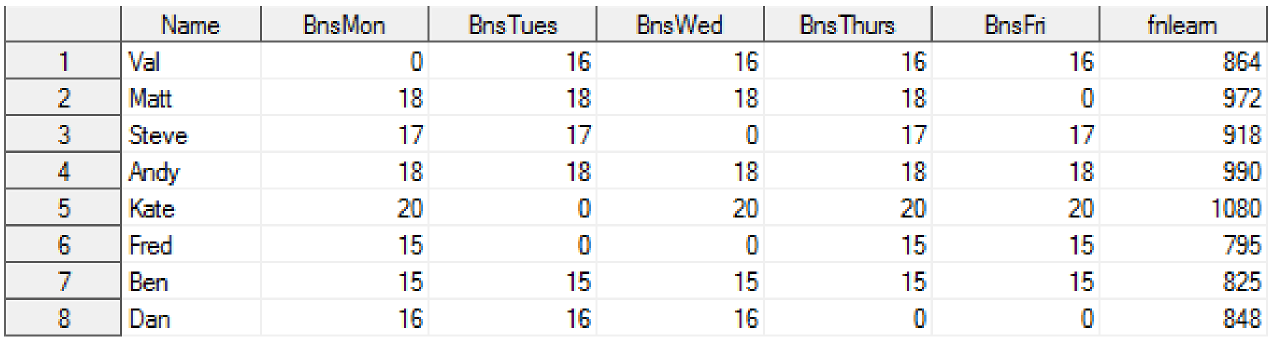
If we were to use a number instead of an asterisk in the OF operator, only that array element will be used in the calculation. Since this is not what we want, we must use the asterisk with the OF operator.

Putting the code together we get a much more compact and efficient DATA step:

```
data Using_Arrays;
      set SAS_scores;
      array array_day {*} Monday Tuesday Wednesday Thursday Friday;
      array bonus {*} BnsMon BnsTues BnsWed BnsThurs BnsFri;
      do i=1 to dim(array_day);
            if NOT MISSING(array_day {i}) then bonus{i}=Wkearn*(0.02);
                  else bonus {i}=0;
      end;
      fnlearn=(wkearn+sum(of bonus {*}));
keep Name BnsMon BnsTues abWed BnsThurs BnsFri fnlearn;

run;
```

Both methods result in the creation of the same variables, as shown in Figure 2.



	Name	BnsMon	BnsTues	BnsWed	BnsThurs	BnsFri	fnlearn
1	Val	0	16	16	16	16	864
2	Matt	18	18	18	18	0	972
3	Steve	17	17	0	17	17	918
4	Andy	18	18	18	18	18	990
5	Kate	20	0	20	20	20	1080
6	Fred	15	0	0	15	15	795
7	Ben	15	15	15	15	15	825
8	Dan	16	16	16	0	0	848

Figure 2: Output of IF-ELSE statement and array methods

In this example we are dealing with only a handful of variables, so the improvements in coding efficiency may seem minor. However, because arrays can easily scale with the size of the program and the data, the time and effort saved will also increase accordingly.

CONCLUSION

An array is a very useful tool when repetitively processing large sets of variables. Arrays allow repetitive code to be more efficiently executed and less prone to coding mistakes. With these helpful tips, processing large sets of variables will no longer seem daunting and can be accomplished efficiently and effectively. Use arrays and in no time you will receive your raise!

REFERENCES

First, Steve and Schudrowitz, Teresa. 2005. “Arrays Made Easy: An Introduction to Arrays and Array Processing”, *Proceedings of the Thirtieth Annual SAS® Users Group International Conference*, Cary, NC: SAS Institute Inc.

Gilsen, Bruce. 1995. “SAS® Arrays: A Basic Tutorial”, *Proceedings of the Twentieth Annual SAS® Users Group International Conference*, Cary, NC: SAS Institute Inc.

Keelan, Stephen. 2002. “Off and Running with Arrays in SAS®”, *Proceedings of the Twenty-Seventh Annual SAS® Users Group International Conference*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. 2011. *SAS® Language Reference: Dictionary, Fourth Edition*. Cary, NC: SAS Institute Inc.

ACKNOWLEDGMENTS

Special thanks to Andrew Clapson, Data Mining Analyst from MD Financial Management for his advice and review of this paper.

CONTACT INFORMATION

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

Kate Burnett-Isaacs
Statistics Canada
Kate.Burnett-Isaacs@statcan.gc.ca