Paper 1780-2007

# Adventures in Arrays: A Beginning Tutorial

Jane Stroupe, SAS Institute, Chicago, IL

## ABSTRACT

Do you need to

- perform repetitive calculations on a group of variables
- create many variables with the same attributes
- restructure your data
- perform a table lookup with one or more factors?

If your answer is “YES” to any of these questions, you should consider using an array in your DATA step.

## INTRODUCTION

In SAS®, an array is simply a way to refer to a group of variables in one observation with a single name.

Think of an array as being an alias for the names of SAS variables on which you need to perform the same task. The variables can be existing variables, new variables or temporary data elements as long as they are the same type.

This tutorial explores the many uses of an array by using examples involving vacation data, collected from ten fictitious vacation resorts. Let's explore the costs and benefits of a vacation at each of these locations.

**DATA:**

SASDATA.EXPENSES stores off-season rates from ten resorts for room, food, and various activities.

| ResortName | Resort | Expense1 | Expense2 | Expense3 | Expense4 | Expense5 | Expense6 |
|---|---|---:|---:|---:|---:|---:|---:|
| Mouse House | HOTEL1 | 165.89 | 45.50 | 78.00 | 45.00 | 12.00 | 35.00 |
| The Princess Palace | HOTEL2 | 215.32 | 64.00 | 54.00 | 50.00 | 18.00 | 30.00 |
| Minnie Mansion | HOTEL3 | 178.90 | 76.00 | 32.00 | 49.00 | 25.00 | 29.99 |
| Sleeping Beauty's Reststop | HOTEL4 | 210.78 | 54.00 | 50.00 | 40.00 | 17.00 | 24.00 |
| Chip and Dale's Nuthouse | HOTEL5 | 189.87 | 49.00 | 67.00 | 63.00 | 38.00 | 42.00 |
| Cinderella's Castle Inn | HOTEL6 | 312.15 | 78.00 | 86.00 | 67.00 | 14.00 | 19.40 |
| Dalmation Doghouse | HOTEL7 | 197.12 | 45.00 | 95.00 | 32.00 | 26.00 | 18.95 |
| Peter Pan's Playhouse | HOTEL8 | 240.98 | 43.00 | 65.13 | 45.00 | 12.00 | 32.00 |
| Bambi Bed & Breakfast | HOTEL9 | 312.10 | 29.98 | 32.87 | 24.00 | 15.00 | 26.00 |
| Fantasia Funhouse | HOTEL10 | 187.98 | 36.00 | 21.54 | 43.00 | 24.00 | 29.65 |

## EXAMPLE 1: USING A ONE-DIMENSIONAL ARRAY TO PERFORM REPETITIVE CALCULATIONS.

**TASK**:

You want a SAS data set with both the off-season and seasonal rates.  The seasonal rates are 25% higher than the off-season rates.

**SOLUTIONS:**

- Use six assignment statements
- Use an array.

**PROGRAM – VERSION 1:**

The first solution requires considerable typing**:**

```
data work.SeasonRates;
   set sasdata.expenses(drop=ResortName);
   seasonal1 = expense1*1.25;
   seasonal2 = expense2*1.25;
   seasonal3 = expense3*1.25;
   seasonal4 = expense4*1.25;
   seasonal5 = expense5*1.25;
   seasonal6 = expense6*1.25;
run;
```

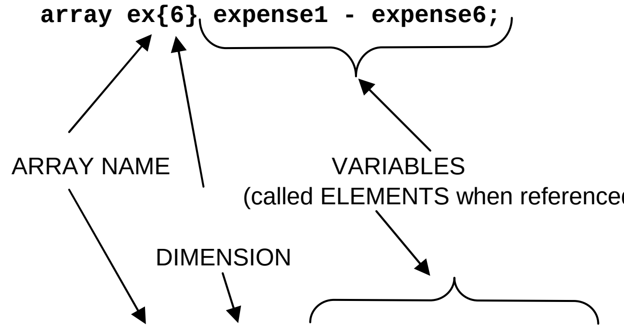
**PROGRAM – VERSION 2:**

To solve this problem using an array, you must first establish the correspondence between

- one array name and the existing variables, expense1 - expense6
- a second array name and the new variables, seasonal1 - seasonal6.

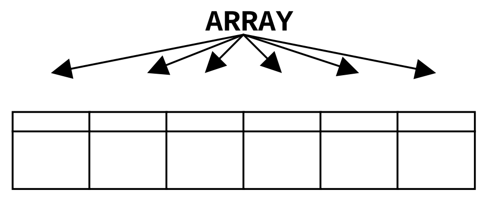
Use array statements to set up the correspondences:

```
array ex{6} expense1 - expense6;
```



```
array seasonal{6} seasonal1 - seasonal6;
```

The ARRAY statement is a compile time statement that creates the correspondence between the array name and, in this example, the variables in the Program Data Vector.



Program Data Vector Variables

The general form of a simple ARRAY statement

| | |
|---|---|
| **ARRAY** *array-name*{*n*} $ *length elements*; | |
| *array-name* | any valid SAS name (avoid function names) |
| *n* | number of elements to which the array refers (often referred to as the dimension(s) of the array) |
| *$* | indication that the array refers to new character data variables |
| *length* | the length for new character variables. The default is 8. |
| *elements* | the variables in the data to which the array refers. If the variables do not exist, SAS will create them. |

Once you establish this correspondence, you can use a DO loop to execute six assignment statements that calculate the weekly total.

```
do i=1 to 6;
   seasonal{i} = ex{i}*1.25;
end;
```

Putting this all together in a second version of the DATA step:

```
data work.SeasonRates(drop=i);
    set sasdata.expenses(drop=ResortName);
①  array ex{6} expense1 - expense6;
②  array seasonal{6} seasonal1 - seasonal6;
③   do i=1 to 6;
       seasonal{i} = ex{i}*1.25;
     end;
④##format expense1 - expense6 seasonal1 - seasonal6 dollar9.2;
run;
```

① To use an array, you must first declare it in an ARRAY statement. At compile time, the ARRAY statement establishes a correspondence between the name of the array, EX, and the numeric DATA step variables to which it refers, EXPENSE1 - EXPENSE6.

The association of an array reference with a variable is only for the duration of the DATA step. If you need to refer to the array in another DATA step, you must repeat the array reference in that DATA step.

② The array, SEASONAL, references variables that do not exist in the Program Data Vector. SAS creates the variables SEASONAL1 - SEASONAL6 from the ARRAY statement.

③ The DO loop processes each element of the arrays sequentially. The assignment statement using the array reference calculates seasonal rates for each of the six expenses.

④ Any FORMAT, LABEL, DROP, KEEP, or LENGTH statements must use DATA step variable names, not array references.

**Data Set SeasonRates (8 of 14 variables)**

| ResortName | Resort | seasonal1 | seasonal2 | seasonal3 | seasonal4 | seasonal5 | seasonal6 |
|---|---|---:|---:|---:|---:|---:|---:|
| Mouse House | HOTEL1 | $207.36 | $56.88 | $97.50 | $56.25 | $15.00 | $43.75 |
| The Princess Palace | HOTEL2 | $269.15 | $80.00 | $67.50 | $62.50 | $22.50 | $37.50 |
| Minnie Mansion | HOTEL3 | $223.63 | $95.00 | $40.00 | $61.25 | $31.25 | $37.49 |
| Sleeping Beauty's Reststop | HOTEL4 | $263.48 | $67.50 | $62.50 | $50.00 | $21.25 | $30.00 |
| Chip and Dale's Nuthouse | HOTEL5 | $237.34 | $61.25 | $83.75 | $78.75 | $47.50 | $52.50 |
| Cinderella's Castle Inn | HOTEL6 | $390.19 | $97.50 | $107.50 | $83.75 | $17.50 | $24.25 |
| Dalmation Doghouse | HOTEL7 | $246.40 | $56.25 | $118.75 | $40.00 | $32.50 | $23.69 |
| Peter Pan's Playhouse | HOTEL8 | $301.23 | $53.75 | $81.41 | $56.25 | $15.00 | $40.00 |
| Bambi Bed & Breakfast | HOTEL9 | $390.13 | $37.48 | $41.09 | $30.00 | $18.75 | $32.50 |
| Fantasia Funhouse | HOTEL10 | $234.98 | $45.00 | $26.93 | $53.75 | $30.00 | $37.06 |

**Valid Syntax for the ARRAY statement:**

1. The array name can be the same as a variable list root word as long as there is no variable named the root word. You should, however, avoid function names since array names that are the same as a SAS function turn off the availability of that function for the duration of the DATA step.

   For example, these ARRAY statements are valid because there is not a variable named EXPENSE in the data set.
   ```
   array expense{6} expense1 - expense6;
   ```
   or
   ```
   array expense{6};
   ```

2. The variable names do not have to be a variable list.

   If the data set variables were named ROOMRATE, DAILYFOOD, SPAVISITS, ROUNDOFGOLF, HORSEBACKRIDING, and PARKADMISSION, you could use the following ARRAY statement.
   ```
   array expense{6} RoomRate DailyFood SpaVisits RoundOfGolf HorseBackRiding
                    ParkAdmission;
   ```

   You can use the double dash to refer to this list of variables that are contiguous in the SAS data set.
   ```
   array expense{6} RoomRate -- ParkAdmission;
   ```

   You could refer to the variable list, just not in ascending suffix order.
   ```
   array expense{6} expense6 expense3 expense2 expense4 expense5 expense1;
   ```

   Since EXPENSE1 - EXPENSE6 are all numeric variables, you can use the key word _NUMERIC_ to refer to them. You can use the _NUMERIC_ keyword only for variables currently defined in the PDV.
   ```
   array expense{6} _numeric_;
   ```

   For the new variables, you can name the variables with a list that has a different root word.

   For example:
   ```
   array seasonalexpense{6} seasonal1 - seasonal6;
   ```
   or
   ```
   array seasonal{6} sRoomRate sDailyFood sSpaVisits sRoundOfGolf
                      sHorseBackRiding sParkAdmission;
   ```
   or
   ```
   array seasonalexpense{6} seasonal6 seasonal4 seasonal2 seasonal1
                            seasonal5 seasonal3;
   ```

   You can not use the double dash to refer to new variable names.

3. The ARRAY statement can automatically name new variables.
   For example, this ARRAY statement automatically creates variables SEASONAL1 - SEASONAL6 since they do not exist in the Program Data Vector.
   ```
   array seasonal{6};
   ```

4. You do not have to specify the dimensions with a number as we did in the array statements.

   However, during compile time, SAS has to know what the dimensions are. You can use a macro variable to specify the dimensions.
   For example:
```
%let number=6;
array ex{&number} expense1 – expense6;
```

   If appropriate, you can use a range of values for the dimensions.
   For example, use this ARRAY statement and DO loop to process the range of values.
```
array ex{94:99}  expense94 – expense99;
do i=94 to 99;
   SAS statements
end;
```

   Sometimes, you can use the * to specify the dimensions and the DIM function in your DO loop.
```
array ex{*} expense1 - expense6;
do i=1 to dim(ex);
   SAS statements
end;
```
   If you use the *, you must specify the elements or variables. The DIM function takes an array name as its argument and returns the dimension of the array.

## EXAMPLE 2: USING A ONE-DIMENSIONAL ARRAY TO PERFORM A TABLE LOOKUP.

**TASK:**
You have budgeted the following amounts for each of the daily expenses: $175 for expense1 ( room), $75 for expense2 (food), $25 for expense3 (spa treatments), $35 for expense4 (a round of golf), $25 for expense5 (horse back riding), and $30 for expense6 (theme park admission). Use an array to assign these budget amounts as initial values and determine the difference between the budgeted amounts and the actual rates.

**SOLUTIONS:**
- Use six assignment statements to calculate the differences.
- Use an array.

**PROGRAM**
```
data work.diffs;
     drop i;
     set sasdata.expenses
              (drop=ResortName);
①    array budget{6} _temporary_ (175,75,25,35,25,30);
     array expense{*} expense1 - expense6;
     array diff{6};
②    do i=1 to dim(expense);
        diff{i} = budget{i} - expense{i};
     end;
run;
```

① You can assign initial values for the corresponding elements in the array by putting these initial values within parenthesis separated by either a comma or a blank. Initial values can be assigned to both variables and temporary data elements.

In the ARRAY statement, the keyword _TEMPORARY_ creates a list of temporary data elements that can be either numeric or character. The temporary elements must be used in a DATA step and are automatically retained for each iteration of the DATA step. You must refer to these elements by the array name and dimension since they do not have variable names, and you cannot use the  asterisk (*) to refer to all the elements.

Arrays of temporary elements are useful when the only purpose for creating an array is to perform a calculation (often referred to as performing a table lookup). You can improve performance CPU time by using temporary array references.

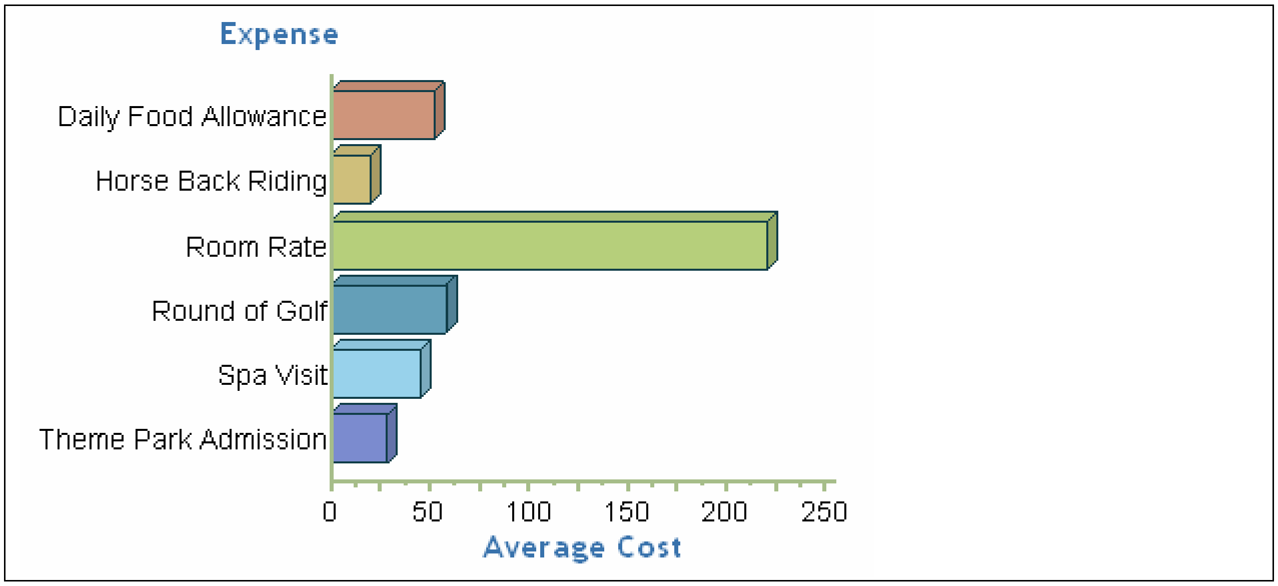
② You can use the DIM function to return the dimensions of the array EXPENSE.

**Data Set Diffs (7 of 13 variables)**

| Resort | diff1 | diff2 | diff3 | diff4 | diff5 | diff6 |
|---|---|---|---|---|---|---|
| HOTEL1 | 9.11 | 29.50 | -53.00 | -10 | 13 | -5.00 |
| HOTEL2 | -40.32 | 11.00 | -29.00 | -15 | 7 | 0.00 |
| HOTEL3 | -3.90 | -1.00 | -7.00 | -14 | 0 | 0.01 |
| HOTEL4 | -35.78 | 21.00 | -25.00 | -5 | 8 | 6.00 |
| HOTEL5 | -14.87 | 26.00 | -42.00 | -28 | -13 | -12.00 |
| HOTEL6 | -137.15 | -3.00 | -61.00 | -32 | 11 | 10.60 |
| HOTEL7 | -22.12 | 30.00 | -70.00 | 3 | -1 | 11.05 |
| HOTEL8 | -65.98 | 32.00 | -40.13 | -10 | 13 | -2.00 |
| HOTEL9 | -137.10 | 45.02 | -7.87 | 11 | 10 | 4.00 |
| HOTEL10 | -12.98 | 39.00 | 3.46 | -8 | 1 | 0.35 |

## EXAMPLE 3: USING A ONE-DIMENSIONAL ARRAY TO RESTRUCTURE YOUR DATA AND PERFORM A TABLE LOOKUP WITH CHARACTER DATA.

**TASK:**
You need to create the following graph.



In order to create the graph, the data must be in this form.

**Data Set Rotate**

| Resort | Expense | Amount | TypeOfExpense |
|---|---|---|---|
| HOTEL1 | 1 | 165.89 | Room Rate |
| HOTEL1 | 2 | 45.50 | Daily Food Allowance |
| HOTEL1 | 3 | 78.00 | Round of Golf |
| HOTEL1 | 4 | 45.00 | Spa Visit |
| HOTEL1 | 5 | 12.00 | Horse Back Riding |
| HOTEL1 | 6 | 35.00 | Theme Park Admission |
| HOTEL2 | 1 | 215.32 | Room Rate |
| HOTEL2 | 2 | 64.00 | Daily Food Allowance |
| HOTEL2 | 3 | 54.00 | Round of Golf |
| ⋮ | ⋮ | ⋮ | ⋮ |

**SOLUTIONS:**

- Use complicated logic to rotate the data, then use 6 assignment statements to create a variable that is the name of the expense.
- Use PROC TRANSPOSE to rotate the data, then a DATA step with assignment statements to create a variable that is the name of the expense.
- Use an array.

**PROGRAM**

```
   data work.rotate;
        keep resort expense amount TypeOfExpense;
        set sasdata.expenses;
①       array ex{*} _numeric_;
②       array name{6} $ 32
                    ('Room Rate'
                     'Daily Food Allowance'
                     'Round of Golf'
                     'Spa Visit'
                     'Horse Back Riding'
                     'Theme Park Admission');
③       do Expense = lbound(ex) to hbound(ex);
④          Amount = ex{Expense};
⑤          TypeOfExpense = name{Expense};
⑥          output;
      end;
   run;
```

① You can use the following SAS variables to reference variables that have been previously defined in the same DATA step:

| | |
|---|---|
| **_NUMERIC_** | indicates all numeric variables. |
| **_CHARACTER_** | indicates all character variables. |
| **_ALL_** | indicates all variables. For arrays, all the previously defined variables must be of the same type. |

In this case _NUMERIC_ refers to the variables EXPENSE1 – EXPENSE6. The SET statement must come before the ARRAY statement in order to refer to these variables with the _NUMERIC_ keyword.

② Since the array NAME refers to character values, the ARRAY statement uses the $ to indicate character data and a length (32). The length needs to be established in the array statement; otherwise, SAS uses the default length of 8. You can use a length statement prior to the array statement to assign the length.

③ The index variable, EXPENSE, becomes a data set variable that, for this example, contains the number of the expense, 1 - 6.

You can use the HBOUND function to return the upper bound of a one-dimensional array and the LBOUND function to return the lower bound. By using these functions you can avoid changing the bounds of an iterative DO loop each time you change the bounds of the array. The HBOUND and LBOUND functions are especially useful if you are using a range for the dimensions.
For example:

```
array expense{100:105} expense1 - expense6;
do i = lbound(expense) to hbound(expense);
```

④ Assign the array value in EX{expense} to the variable AMOUNT.

⑤ Assign the array value in NAME{expense} to the variable TYPEOFEXPENSE.

⑥ Use an OUTPUT statement in the DO loop in order to write out an observation for each iteration of the loop.

**EXAMPLE 4: USING A TWO-DIMENSIONAL ARRAY**

You can think of a two-dimensional array as having 'rows' and 'columns', but to SAS, a two-dimensional array is no different than a one-dimensional array. It is still an alias for data set variables in the PDV or for temporary data elements.

**TASK:**
You know the following information about daily rates for taxes and gratuity charged by the resorts.

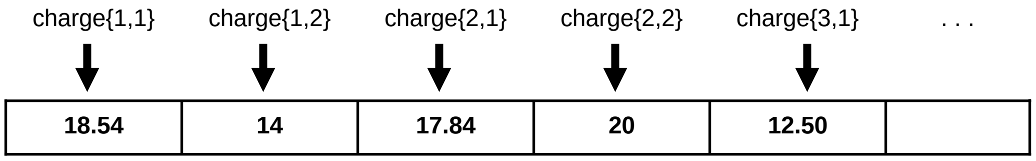
| Resort | Taxes | Gratuity |
|---|---|---|
| 1 | 18.54 | 14 |
| 2 | 17.84 | 20 |
| 3 | 12.50 | 15 |
| 4 | 14.25 | 18 |
| 5 | 16.33 | 12 |
| 6 | 19.00 | 16 |
| : | : | : |

You want to combine this information with the data in SASDATA.EXPENSES to calculate the total cost for each resort.

**SOLUTIONS:**
- Create a SAS data set containing these constants and merge it with SASDATA.EXPENSES.
- Use IF..THEN logic and assignment statements to create the new variables.
- Since this information is in rows and columns, use a temporary two-dimensional array to hold the tax and gratuity information.

**PROGRAM**
```
data work.total;
     drop num;
①    array charge{10,2} _temporary_
                 (18.54      14
                  17.84      20
                  12.50      15
                  14.25      18
                  16.33      12
                  19.00      16
                  12.75      19
                  14.98      16
                  15.76      20
                  13.75      17);
     set sasdata.expenses;
②    num = input(substr(resort,6),2.);
③    tax = charge{num,1};
     gratuity = charge{num,2};
     total = sum(of expense1-expense6,tax,gratuity);
run;
```

① The array, CHARGE, refers to 10 rows and 2 columns. It is used to hold constants. Even though you can type them into rows and columns and that helps you visualize what the program needs to do, SAS is assigning these values to contiguous slots in memory. You can refer to these slots by row and column number.



If you had created variables instead of using _TEMPORARY_, the variables would have been named

CHARGE1 referring to CHARGE{1,1}
CHARGE2 referring to CHARGE{1,2}
CHARGE3 referring to CHARGE{2,1}
CHARGE4 referring to CHARGE{2,2},
and so forth.

② The variable RESORT in SASDATA.EXPENSES has values HOTEL1, HOTEL2, etc. You want to use this as an index when you reference the array, but an array index must be numeric. Therefore, you must extract the digit from the end of this value by using the SUBSTR(*'text',start,end*) function. The SUBSTR function returns a character value. Using the INPUT function then converts the character value to numeric.

③ Create a variable TAX and a variable GRATUITY using the appropriate resort number as the row dimension, and the appropriate number as the column dimension, where column 1 is the value for TAX and column 2 is the value for GRATUITY.

**Data Set Total**

| ResortName | Resort | Expense1 | Expense2 | Expense3 | Expense4 | Expense5 | Expense6 | tax | gratuity | total |
|---|---|---|---|---|---|---|---|---|---|---|
| Mouse House | HOTEL1 | 165.89 | 45.50 | 78.00 | 45.00 | 12.00 | 35.00 | 18.54 | 14 | 413.93 |
| The Princess Palace | HOTEL2 | 215.32 | 64.00 | 54.00 | 50.00 | 18.00 | 30.00 | 17.84 | 20 | 469.16 |
| Minnie Mansion | HOTEL3 | 178.90 | 76.00 | 32.00 | 49.00 | 25.00 | 29.99 | 12.50 | 15 | 418.39 |
| Sleeping Beauty's Reststop | HOTEL4 | 210.78 | 54.00 | 50.00 | 40.00 | 17.00 | 24.00 | 14.25 | 18 | 428.03 |
| Chip and Dale's Nuthouse | HOTEL5 | 189.87 | 49.00 | 67.00 | 63.00 | 38.00 | 42.00 | 16.33 | 12 | 477.20 |
| Cinderella's Castle Inn | HOTEL6 | 312.15 | 78.00 | 86.00 | 67.00 | 14.00 | 19.40 | 19.00 | 16 | 611.55 |
| Dalmation Doghouse | HOTEL7 | 197.12 | 45.00 | 95.00 | 32.00 | 26.00 | 18.95 | 12.75 | 19 | 445.82 |
| Peter Pan's Playhouse | HOTEL8 | 240.98 | 43.00 | 65.13 | 45.00 | 12.00 | 32.00 | 14.98 | 16 | 469.09 |
| Bambi Bed & Breakfast | HOTEL9 | 312.10 | 29.98 | 32.87 | 24.00 | 15.00 | 26.00 | 15.76 | 20 | 475.71 |
| Fantasia Funhouse | HOTEL10 | 187.98 | 36.00 | 21.54 | 43.00 | 24.00 | 29.65 | 13.75 | 17 | 372.92 |

## EXAMPLE 5: CREATING A MULTI-DIMENSIONAL ARRAY FROM A SAS DATA SET.

**TASK:**

You need to determine the overall ratings for your choice of resorts. The SAS data set, SASDATA.HTLRATE, contains ratings for each hotel in each of the categories. room, food, and the activities offered. Room is rated on a scale from 1 to 5, food is rated on a scale from 1 to 4, and activities is rated on a scale from 1 to 3. The higher the rating, the better the hotel.

| Hotel | RoomRate | FoodRate | ActRate |
|---|---|---|---|
| HOTEL1 | 4 | 3 | 1 |
| HOTEL2 | 2 | 4 | 2 |
| HOTEL3 | 5 | 2 | 1 |
| HOTEL4 | 2 | 4 | 2 |
| HOTEL5 | 4 | 1 | 3 |
| HOTEL6 | 2 | 2 | 3 |
| HOTEL7 | 5 | 3 | 2 |
| HOTEL8 | 2 | 4 | 3 |
| HOTEL9 | 3 | 2 | 3 |
| HOTEL10 | 5 | 3 | 1 |

The SAS data set, SASDATA.RATINGS, has descriptions of the overall ratings from a guidebook. You need to combine these in order to determine the best place to stay.

| ROOM | FOOD | ACTIVITY | RATING |
|---|---|---|---|
| 1 | 1 | 1 | Awful |
| 1 | 1 | 2 | Awful |
| 1 | 1 | 3 | Maybe |
| : | : | : | : |
| 5 | 3 | 2 | Good |
| 5 | 3 | 3 | Great |
| 5 | 4 | 1 | Good |
| 5 | 4 | 2 | Great |
| 5 | 4 | 3 | Fantastic |

**PROGRAM:**

```
data compare(keep=hotel roomrate foodrate actrate HotelRating);
① array rate{5,4,3} $ _temporary_;
② if _n_ = 1 then do i=1 to all;
      set sasdata.ratings nobs=all;
③     rate{room,food,activity}=rating;
    end;
④ set sasdata.htlrate;
⑤ HotelRating=rate{roomrate,foodrate,actrate};
run;
```

① The dimensions {5,4,3} indicate that the array, RATE, refers to 5 “pages”, 4 “rows”, and 3 “columns”. In this example, RATE refers to 5 room rates, 4 food rates, and 3 activity rates. If the ARRAY statement had not used the keyword, _TEMPORARY_, SAS would have created 60 variables, RATE1 - RATE60. With the _TEMPORARY_ keyword, RATE refers to 60 contiguous “slots” in memory in which the values of SASDATA.RATINGS will be stored.

② The first time through the DATA step (when _n_ = 1), the value of RATING from SASDATA.RATINGS is loaded into the array. During compile time, the NOBS=ALL SET statement option assigns the number of observations from SASDATA.RATINGS (60) to the temporary numeric variable ALL. Therefore, the DO loop will execute 60 times.

③ ROOM, FOOD, and ACTIVITY are used as the dimensions to load RATING into the correct slot in memory.

④ After the array is loaded, SASDATA.HTLRATE is read.

⑤ The variable, HOTELRATING, is created by assigning the appropriate value of the RATE array.

**Data Set Compare**

| Obs | Hotel | RoomRate | FoodRate | ActRate | HotelRating |
|---|---|---|---|---|---|
| 1 | HOTEL1 | 4 | 3 | 1 | OK |
| 2 | HOTEL2 | 2 | 4 | 2 | OK |
| 3 | HOTEL3 | 5 | 2 | 1 | OK |
| 4 | HOTEL4 | 2 | 4 | 2 | OK |
| 5 | HOTEL5 | 4 | 1 | 3 | OK |
| 6 | HOTEL6 | 2 | 2 | 3 | Maybe |
| 7 | HOTEL7 | 5 | 3 | 2 | Good |
| 8 | HOTEL8 | 2 | 4 | 3 | OK |
| 9 | HOTEL9 | 3 | 2 | 3 | OK |
| 10 | HOTEL10 | 5 | 3 | 1 | OK |

## CONCLUSION

If your data processing needs repetitive code, creation of many variables with the same attributes, converting data from a horizontal structure to a vertical structure, assigning many constants, then think ARRAY. Arrays are powerful tools because they allow you to group similar variables under one umbrella alias for similar processing. Though array references are assigned at compile time, the syntax allows for flexibility when declaring variable names and dimensions.

So enjoy yourself as your adventure in arrays takes you further in your data step programming.

## ACKNOWLEDGMENTS

The author would like to thank Linda Jolley, Cynthia Zender, Rick Bell, and Kay Alden who helped the author by reviewing this paper.

## CONTACT INFORMATION

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

Jane Stroupe
Phone: 847.367.6216
Email: Jane.Stroupe@sas.com