Doing More with Arrays and Do Loops: When One Dimension Starts to Wear Thin

Created 3/2008

By Afton-Royal Training & Consulting

What to Expect

We have 45 minutes to learn more advanced techniques for arrays and do loops. We'll assume you got the basic syntax of the array statement and using arrays in do loops.

- First, we'll learn some functions and special keywords often used with arrays.
- Second, we'll look at two-dimensional arrays "live".

Part 1

Special Array Functions and Keywords

More Detail on Defining an Array

- Although we often put only one number in the parentheses of an array statement, this assumes that the lower bound or starting value is 1.

```
array eggs(1:3) blue green yellow;
```

- To make your code more robust to changes, calculate the dimension and bounds of the array, rather than typing them in.

```
array eggs(*) blue green yellow;
```

Functions for Arrays

- The HBOUND() function returns the upper or High bound of the array.

```
array eggs(1:3) blue green yellow;
*so hbound(eggs)=3*;
```

- The LBOUND() function returns the lower bound.

```
*so lbound(eggs)=1*;
```

- The DIM() function counts the dimension or number of elements in the array. As long as your array starts at 1 and counts by 1, this will equal the upper bound.

```
*so dim(eggs)=3*;
```

Example: Still One Dimension – But More Flexible

```
*define the array first*;
array eggs(*) blue green yellow;

*SAS counts the elements for you*;
do i=lbound(eggs) to hbound(eggs);
*or do i=1 to dim(eggs)*;
   if eggs(i) ne . then
      AllEggs + eggs(i);
end;
```

Useful Array Keywords

- To array all character or numeric variables in the data set, use _CHARACTER_ or _NUMERIC_.
- _ALL_ uses all the variables, but only works if all the variables are the same type.

```
array numbers(*) _numeric_;
array characters(*) $ _character_;
array all(*) _all_;
```

Example: DO Over (Undocumented)

- Do over uses no index variable.
- It is left from Version 6, and SAS recommends we not use it, but it's good to recognize it.

```
ARRAY able a1-a5;
*Define the array first*;

*Use the array in the DO over loop*;
do over able;
   if able = . then able = 5;
end;
```

Part 2

Two-Dimensional Arrays: Nothing to Be Afraid of?

Why Two Dimensions?

Sometimes you have data that is more like a spreadsheet (rows and columns), and you need access to the entire data set at once.

- A different price applies to an account based on the combination of values in two variables.
- Based on the day of the week and the time of day, different associates are “on call”.
- You want to use a two-dimensional “spreadsheet” to look up how many eggs of a particular color were made each Easter!

What Does the Array Look Like?

- A two dimensional array is arranged much like a spreadsheet, or a small data set.
- Remember that all arrays are temporary, and don’t exist outside the data step.
- Think of a two dimensional array reading like a piece of paper, first the top row from left to right, then the second row from left to right, etc.
- You will need two subscripts, one for each dimension.

The “Spreadsheet”

	Blue	Green	Yellow
2004	1	2	3
2005	2	1	3
2006	2	2	2
2007	3	4	2

The Code

```
*Slide 12*;
*the spreadsheet data is in file='myeggdata.txt'*;

*look at the data we'll use*;
proc print data=searchfor; run;

data full ;

*Create the array*;
array Easter(2004:2007,3);
array Size(2004:2007) $; *we added this*;

*don't let it disappear*;
retain Easter1-Easter12 size1-size4;
drop Easter1-Easter12 size1-size4;

*read the raw data into the array*;
infile 'myeggdata.txt';

if _n_=1 then do;
   do year=2004 to 2007;
          do color=1 to 3;
                      input Easter(year,color)@;
                   end;
                   input size(year) $; *we added this*;
        end;
end;

*read in the SAS data*;
set searchfor;

*use the array to look up the value*;
NumEggs=Easter(year,color);
TheSize=size(year); *we added this*;
run;

proc print; run;
```