

Arrays and DO Loops in SAS

1) ARRAYS

- in computer science, an ARRAY is a structure for holding multiple, related items that allows the user to reference the items using matrix notation; like matrices, ARRAYs can be uni-dimensional or multi-dimensional
 - the array itself has a name
 - the elements of the array are referenced through “subscripts”
 - the number of subscripts corresponds to the number of dimensions
- example, suppose that we had survey information regarding a respondent’s monthly participation in the Food Stamp Program over the preceding calendar year
 - suppose the name of the array was *fspart*
 - the array would have 12 elements—a series of binary indicators corresponding to participation in each of the months of the year
 - the structure for this *uni-dimensional* ARRAY could be depicted

| | | | | | | | | | | | | |
|-----------------|------|------|------|------|-----|------|------|------|------|------|------|------|
| Month: | Jan. | Feb. | Mar. | Apr. | May | Jun. | Jul. | Aug. | Sep. | Oct. | Nov. | Dec. |
| subscriber: | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| <i>fspart</i> : | | | | | | | | | | | | |

- so the January element of the array would be *fspart*{1}; the February element would be *fspart*{2}, and so on
- example, suppose that we had similar survey information regarding monthly Food Stamp Participation but that it covered two years instead of one
 - the structure for this *multi-dimensional* ARRAY could be depicted

| | | | | | | | | | | | | | |
|------------|------------|------|------|------|------|-----|------|------|------|------|------|------|------|
| | Month: | Jan. | Feb. | Mar. | Apr. | May | Jun. | Jul. | Aug. | Sep. | Oct. | Nov. | Dec. |
| Year: | subscripts | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| <i>t-2</i> | 1 | | | | | | | | | | | | |
| <i>t-1</i> | 2 | | | | | | | | | | | | |

- the array has 24 elements—binary indicators for every month of two years
- the element corresponding to participation in May two years ago would be *fspart*{1,5}; the element for May a one year ago would be *fspart*{2,5}
- note that the user usually supplies the interpretation of the subscripts
- as should be clear, there are many potential ways to arrange the data

2) ARRAYS in SAS

- SAS permits the “construction” and use of ARRAYS in DATA steps
- the first step in using an ARRAY in SAS is to declare one; this is done through an ARRAY statement¹

¹ SAS allows “explicit” and “implicit” definitions of ARRAYS. The implicit definition is an older and less used approach. This course will only cover the explicit definition.

- syntax:

```
ARRAY <array-name{subscript(s)}> <$> <length>  
    <<array-elements> <(initial values)>;
```

- at a minimum the ARRAY definition must contain the name of the ARRAY and either an indication of the subscripts or a description of the elements themselves
- the array-name must be a valid SAS name, similar to a variable name
- the subscript or subscripts, which must appear in braces {}, is either a numerical constant, an asterisk, or a range of numbers
 - ◇ a numerical constant is the most common specification and would simply give the number of elements in the ARRAY
 - ◇ if multiple dimensions are desired, the numbers specifying each dimension would be separated by commas
 - ◇ the asterisk is a wild card character and indicates that the ARRAY has as many elements as are defined subsequently in the ARRAY statement; if the asterisk is used elements must be defined AND the ARRAY can only have one dimension
 - ◇ ranges take the form *lower:upper* where *lower* is the value of the first subscript in the range and *upper* is the value of the last; the number of elements would be (*upper* - *lower* + 1); ranges are useful when the subscripts are tied to other values such as years
- the array elements are usually SAS variables
 - ◇ the elements must all be of the same type
 - ◇ they may include existing SAS variables; in which case, the ARRAY serves mainly as an alternative way to reference the variables
 - ◇ however, they can also be new variables
 - ◇ if you do not define the elements (or if there are fewer elements than are defined by the subscript), SAS creates new variables that have the array-name as the prefix and the subscript number as the suffix
- other ways to define elements
 - ◇ `_NUMERIC_` will define the elements as all of the previously defined numeric variables from the DATA step
 - ◇ `_CHARACTER_` does the same thing with character variables
 - ◇ `_ALL_` will define the elements as all of the previously defined variables from the DATA step (all of the previously defined variables would have to be of the same type)
 - ◇ `_TEMPORARY_` will define a list of temporary variables; these do not have names, are automatically `RETAINED`, and are not transferred to the SAS data set that is created by the DATA step
- you can fill the initial values of the elements by listing the values in parentheses and separated by blanks or commas at the end of the ARRAY statement
- you can also specify whether the elements are character or numeric by using the \$ and specify the LENGTH; if these are not specified, the ARRAY statement will use the existing types of the variables
- some examples

- ◇ for the uni-dimensional Food Stamp Participation ARRAY that we had discussed earlier

```
ARRAY fspart{12};
```

would create variables *fspart1* – *fspart12*

- ◇ suppose that we already had variables called *fsp1* – *fsp12*; we could also specify the ARRAY as

```
ARRAY fspart{12} fsp1 fsp2 fsp3 fsp4 fsp5 fsp6
                fsp7 fsp8 fsp9 fsp10 fsp11 fsp12;
```

or even more compactly as

```
ARRAY fspart{*} fsp1-fsp12;
```

- ◇ suppose that we wanted to initialize an ARRAY for *fspart1* – *fspart12* with zeroes, we would issue the command

```
ARRAY fspart{12} (0 0 0 0 0 0 0 0 0 0 0 0);
```

- now consider the two dimensional example
 - ◇ suppose that you had a calendar with 24 months of data starting with January from two years ago (*fsp1*) and continuing to December from last year (*fsp24*)
 - ◇ you could specify a two dimensional ARRAY as

```
ARRAY fspart{2,12} fsp1-fsp24;
```

- ◇ note that if you had issued the statement

```
ARRAY fspart{2,12};
```

SAS would have constructed variables *fspart1* – *fspart24*

- referencing ARRAY elements
 - after an ARRAY is defined, you can reference an element by typing

```
array-name{subscript}
```

where *array-name* is the name that was defined earlier by the ARRAY statement and *subscript* is a numerical constant or numerical value with the desired subscript index

- this specification works just like any other variable specification in SAS and can be used in the same ways that variables are (e.g., in assignment statements, in conditioning statements, etc.)
- in assignment statements, ARRAY elements can appear on the left or right side (or both sides)
- using our previous uni-dimensional example, the code

```
ARRAY fspart{12} fsp1-fsp12;
```

```
x = fspart{7};
```

creates a 12-element uni-dimensional ARRAY of Food Stamp Participation indicators and assigns the July value to the SAS variable *x*

- this code does the same thing

```
ARRAY fspart{12} fsp1-fsp12;
```

```
i = 7;
```

```
x = fspart{i};
```

- referencing all of the elements in an ARRAY as a list
 - if you use the asterisk as the subscript reference, SAS treats the ARRAY as a list of variables
 - for example, to INPUT the elements of *fspart*, you could type

```
ARRAY fspart{12} fsp1-fsp12;
```

```
INPUT fspart{*};
```

- to determine whether someone had participated at all in the Food Stamp Program during the preceding year, you could type

```
anyfsprt = MAX(of fspart{*});
```

- DIM function
 - the function DIM(*array-name*) returns the number of elements in *array-name*;
 - ◊ if the ARRAY is multi-dimensional, DIM() returns the number of elements in the first dimension
 - ◊ to obtain the number of elements in higher dimensions, use DIM n (), where n is the dimension that you are interested in
 - this function can be useful for writing general code that does not have to be updated if your ARRAY specifications change
- out-of-range references
 - each time that you try to reference an element in an ARRAY, SAS checks the subscripts against the definition for the ARRAY
 - if the subscript is below 1 or the *lower* value (if a range is specified) or above the maximum possible subscript value, SAS will issue a run-time error message that an “array subscript is out of range” and stop processing the DATA step

3) DO loops

- the use of variables as subscript references is a big advantage of ARRAY processing

- variable references mean that you can pick out an element of an ARRAY relative to some characteristic of the current observation or relative to some condition that you are processing
- an additional tool that extends variable subscript references is the iterative DO loop
- the standard syntax for a DO loop is

```
DO <index_variable> = <start_value> TO <stop_value> <BY <increment_value>>;
   SAS commands
END;
```

- the DO loop causes the SAS commands to be performed repeatedly
- the DO loop first sets the *index_variable* to the *start_value*
 - ◇ the *index_variable* must be a SAS variable
 - ◇ the *start_value* can be either a numerical constant or a numerical variable
- the DO loop compares the *index_variable* to the *stop_value* (which itself can be a numerical constant or a numerical variable)
 - ◇ if the *index_variable* is not yet “past” the *stop_value* (greater than the value if positive increments are used or less than the value if negative increments are used), SAS performs the statements in the loop
 - ◇ if the *index_variable* is “past” the *stop_value*, SAS “exits” the DO loop by going to the next statement following the END
- the DO loop is iterative
 - ◇ after SAS performs the statements, it returns to the top of the loop (returns to the DO statement)
 - ◇ the *index_variable* is then changed by the amount of the *increment_value* or by 1 if no *increment_value* is specified; note that the *increment_value* can be negative
 - ◇ the loop continues until the *index_variable* is past than the *stop_value*
- DO loops are useful in lots of contexts, but they are especially useful for traversing (accessing all the elements of) ARRAYS
- example #3.1:
 - suppose that we have a uni-dimensional ARRAY *fspart* that describes Food Stamp Participation over the preceding year
 - suppose also that we want to count the number of months out of the year that someone participated
 - we could use the code

```
ARRAY fspart{12} fsp1-fsp12;

fsmnths = 0;
DO month = 1 TO 12;
   fsmnths = fsmnths + fspart{month};
END;
```

- examples 3.2 and 3.3:

- consider the same ARRAY, but now assume that we want to measure the number of quarters out of the last year that someone was receiving Food Stamps
- we could use code with nested DO loops (example #3.2)

```

ARRAY fspart{12} fsp1-fsp12;

fsqtrs = 0;
month = 0;
DO qtr = 1 TO 4;                                /* loop over quarters */
  fsthistrqtr = 0;
  DO qtrmonth = 1 TO 3;                          /* loop over months in qtr */
    month = month + 1;
    fsthistrqtr = MAX(fsthistrqtr, fspart{month});
  END;                                           /* end of qtrmonth loop */
  fsqtrs = fsqtrs + fsthistrqtr;
END;                                           /* end of qtr loop */

```

- or code with incremental indexing (example #3.3)

```

ARRAY fspart{12} fsp1-fsp12;

fsqtrs = 0;
DO month = 3 TO 12 BY 3;
  fsqtrs = fsqtrs + MAX(fspart{month}, fspart{month-1}, fspart{month-2});
END;

```

- example #3.4:
 - consider a two-dimensional ARRAY, and assume that we want to measure the number of quarters out of the last two years that someone was receiving Food Stamps
 - we could use code with nested DO loops

```

ARRAY fspart{2,12};

fsqtrs = 0;
DO year_ndx = 1 TO 2;                            /* loop over years */
  DO month = 3 TO 12 BY 3;
    fsqtrs = fsqtrs + MAX(fspart{month}, fspart{month-1}, fspart{month-2});
  END;                                           /* end of month loop */
END;                                           /* end of year loop */

```

4) Other DO loops

- SAS supports two other types of DO loops
- DO WHILE loop
 - syntax

```

DO WHILE (<logical_expression>);
  SAS commands

```

END;

- this loop will execute these commands while the logical expression is true
- if the logical expression is false when the loop is entered, SAS skips the commands (effectively the condition is checked at the top of the loop)
- for example, we could redo example #3.1 from above with the following code

```
ARRAY fspart{12} fsp1-fsp12;

fsmnths = 0;
month = 1;
DO WHILE (month LE 12);
    fsmnths = fsmnths + fspart{month};
    month = month + 1;
END;
```

- a similar construct is the DO UNTIL loop
 - syntax

```
DO UNTIL (<logical_expression>);
    SAS commands
END;
```

- this loop will execute these commands until the logical expression is true
- if the logical expression is false when the loop is entered, SAS executes the commands (effectively the condition is checked at the bottom of the loop); this means that DO UNTIL loops are always executed at least once
- for example, we could redo the previous example with the following code

```
ARRAY fspart{12} fsp1-fsp12;

fsmnths = 0;
month = 1;
DO UNTIL (month GT 12);
    fsmnths = fsmnths + fspart{month};
    month = month + 1;
END;
```

- WHILE and UNTIL conditions can also be added to incremental DO loops
- the syntax is

```
DO <index_variable> = <start_value> TO <stop_value> <BY <increment_value>>
    <WHILE (<logical_expression>)> <UNTIL(<logical_expression>)>;
    SAS commands
END;
```

- the DO loop would check
 - ◊ the iterative condition
 - ◊ the WHILE condition, if it is included, and
 - ◊ the UNTIL condition, if it is included
- **WARNING:** loops can be both tricky and dangerous to program
 - you need to consider the conditions very carefully; as the foregoing discussion indicates, the distinctions between different types of loops can be very subtle
 - as importantly, you need to verify that the conditions that terminate a loop will be met at some point
 - if the conditions are not met, you could create an “endless loop”
 - consider the following code

```

ARRAY fspart{12} fsp1-fsp12;

fsmnths = 0;
month = 1;
DO WHILE (month LE 12);
    fsmnths = fsmnths + fspart{month};
END;

```

- we’ve made a mistake; the variable *month* is never incremented
- because of this the WHILE condition always remains true
- SAS will continue to process this statement until you either
 - ◊ issue a “break” (the red exclamation point in the black circle in the SAS tool bar across the top of the SAS window) or
 - ◊ shut down SAS