CHAPTER

25

Array Processing

Definitions

array
is a temporary grouping of SAS variables that are arranged in a particular order and identified by an *array-name*. The array exists only for the duration of the current DATA step. The array-name distinguishes it from any other arrays in the same DATA step; it is not a variable.

Note: Arrays in SAS are different from those in many other programming languages. In SAS, an array is not a data structure but is just a convenient way of temporarily identifying a group of variables. △

array processing
is a method that enables you to perform the same tasks for a series of related variables.

array reference
is a method to reference the elements of an array.

one-dimensional array
is a simple grouping of variables that, when processed, results in output that can be represented in simple row format.

multidimensional array
is a more complex grouping of variables that, when processed, results in output that could have two or more dimensions, such as columns and rows.

Basic array processing involves the following steps:

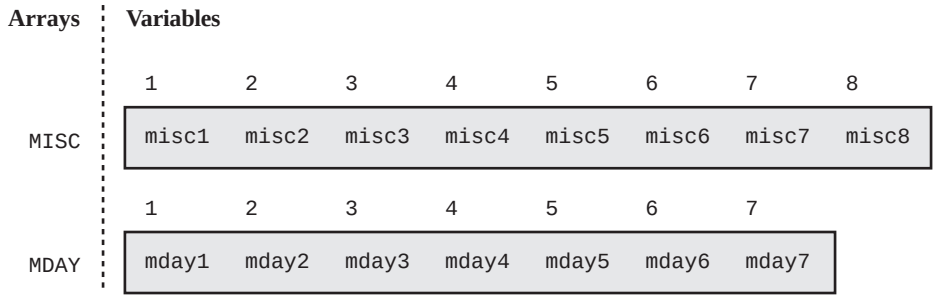
- grouping variables into arrays
- selecting a current variable for an action
- repeating an action.

A Conceptual View of Arrays

One-Dimensional Array

The following figure is a conceptual representation of two one-dimensional arrays, MISC and MDAY.

Figure 25.1 One-Dimensional Array



MISC contains eight elements, the variables MISC1 through MISC8. To reference the data in these variables, use the form MISC{*n*}, where *n* is the element number in the array. For example, MISC{6} is the sixth element in the array.

MDAY contains seven elements, the variables MDAY1 through MDAY7. MDAY{3} is the third element in the array.

Two-Dimensional Array

The following figure is a conceptual representation of the two-dimensional array EXPENSES.

Figure 25.2 Example of a Two-Dimensional Array

First Dimension		**Second Dimension**							
Expense Categories		Days of the Week							Total
		1	2	3	4	5	6	7	8
Hotel	1	hotel1	hotel2	hotel3	hotel4	hotel5	hotel6	hotel7	hotel8
Phone	2	phone1	phone2	phone3	phone4	phone5	phone6	phone7	phone8
Pers. Auto	3	peraut1	peraut2	peraut3	peraut4	peraut5	peraut6	peraut7	peraut8
Rental Car	4	carrnt1	carrnt2	carrnt3	carrnt4	carrnt5	carrnt6	carrnt7	carrnt8
Airfare	5	airlin1	airlin2	airlin3	airlin4	airlin5	airlin6	airlin7	airlin8
Dues	6	dues1	dues2	dues3	dues4	dues5	dues6	dues7	dues8
Registration Fees	7	regfee1	regfee2	regfee3	regfee4	regfee5	regfee6	regfee7	regfee8
Other	8	other1	other2	other3	other4	other5	other6	other7	other8
Tips (non-meal)	9	tips1	tips2	tips3	tips4	tips5	tips6	tips7	tips8
Meals	10	meals1	meals2	meals3	meals4	meals5	meals6	meals7	meals8

The EXPENSES array contains ten groups of eight variables each. The ten groups (expense categories) comprise the first dimension of the array, and the eight variables (days of the week) comprise the second dimension. To reference the data in the array variables, use the form EXPENSES{*m*,*n*}, where *m* is the element number in the first dimension of the array, and *n* is the element number in the second dimension of the array. EXPENSES{6,4} references the value of dues for the fourth day (the variable is DUES4).

Syntax for Defining and Referencing an Array

To define a simple or a multidimensional array, use the ARRAY statement. The ARRAY statement has the following form:

ARRAY *array-name* {*number-of-elements*} <*list-of-variables*>;

where

array-name
is a SAS name that identifies the group of variables.

number-of-elements
is the number of variables in the group. You must enclose this value in parentheses, braces, or brackets.

list-of-variables
is a list of the names of the variables in the group. All variables that are defined in a given array must be of the same type-either all character or all numeric.

For complete information about the ARRAY statement, see *SAS Language Reference: Dictionary*.

To reference an array that was previously defined in the same DATA step, use an Array Reference statement. An array reference has the following form:

array-name {*subscript*}

where

array-name
is the name of an array that was previously defined with an ARRAY statement in the same DATA step.

subscript
specifies the subscript, which can be a numeric constant, the name of a variable whose value is the number, a SAS numeric expression, or an asterisk (*).

Note: Subscripts in SAS are 1-based by default, and not 0-based as they are in some other programming languages. △

For complete information about the Array Reference statement, see "Statements" in *SAS Language Reference: Dictionary*.

Processing Simple Arrays

Grouping Variables in a Simple Array

The following ARRAY statement creates an array named BOOKS that contains the three variables Reference, Usage, and Introduction:

```
array books{3} Reference Usage Introduction;
```

When you define an array, SAS assigns each array element an *array reference* with the form *array-name*{*subscript*}, where *subscript* is the position of the variable in the list. The following table lists the array reference assignments for the previous ARRAY statement:

Variable	Array reference
Reference	books{1}
Usage	books{2}
Introduction	books{3}

Later in the DATA step, when you want to process the variables in the array, you can refer to a variable by either its name or its array reference. For example, the names Reference and books{1} are equivalent.

Using a DO Loop to Repeat an Action

To perform the same action several times, use an iterative DO loop. A simple iterative DO loop that processes an array has the following form:

DO *index-variable*=1 **TO** *number-of-elements-in-array*;
... more SAS statements ...

END;

The loop is processed repeatedly (iterates) according to the instructions in the iterative DO statement. The iterative DO statement contains an *index-variable* whose name you specify and whose value changes at each iteration of the loop.

To execute the loop as many times as there are variables in the array, specify that the values of *index-variable* are 1 TO *number-of-elements-in-array*. SAS increases the value of *index-variable* by 1 before each new iteration of the loop. When the value exceeds the *number-of-elements-in-array*, SAS stops processing the loop. By default, SAS automatically includes *index-variable* in the output data set. Use a DROP statement or the DROP= data set option to prevent the index variable from being written to your output data set.

An iterative DO loop that executes three times and has an index variable named count has the following form:

```
do count=1 to 3;
    … more SAS statements …
end;
```

The first time the loop processes, the value of count is 1; the second time, 2; and the third time, 3. At the beginning of the fourth iteration, the value of count is 4, which exceeds the specified range and causes SAS to stop processing the loop.

Using a DO Loop to Process Selected Elements in an Array

To process particular elements of an array, specify those elements as the range of the iterative DO statement. For example, the following statement creates an array DAYS that contains seven elements:

```
array days{7} D1-D7;
```

The following DO statements process selected elements of the array DAYS:

do i=2 to 4;	processes elements 2 through 4
do i=1 to 7 by 2;	processes elements 1, 3, 5, and 7
do i=3,5;	processes elements 3 and 5

Selecting the Current Variable

You must tell SAS which variable in the array to use in each iteration of the loop. Recall that you identify variables in an array by their array references and that you use a variable name, a number, or an expression as the subscript of the reference. Therefore, you can write programming statements so that the index variable of the DO loop is the subscript of the array reference (for example, *array-name*{*index-variable*}). When the value of the index variable changes, the subscript of the array reference (and therefore the variable that is referenced) also changes.

The following example uses the index variable count as the subscript of array references inside a DO loop:

```
array books{3} Reference Usage Introduction;
do count=1 to 3;
   if books{count}=. then books{count}=0;
end;
```

When the value of count is 1, SAS reads the array reference as books{1} and processes the IF-THEN statement on books{1}, which is the variable Reference. When count is 2, SAS processes the statement on books{2}, which is the variable Usage. When count is 3, SAS processes the statement on books{3}, which is the variable Introduction.

The statements in the example tell SAS to

- perform the actions in the loop three times
- replace the array subscript count with the current value of count for each iteration of the IF-THEN statement
- locate the variable with that array reference and process the IF-THEN statement on it
- replace missing values with zero if the condition is true.

The following DATA step defines the array BOOK and processes it with a DO loop.

```
options nodate pageno=1 linesize=80 pagesize=60;

data changed(drop=count);
   input Reference Usage Introduction;
   array book{3} Reference Usage Introduction;
   do count=1 to 3;
      if book{count}=. then book{count}=0;
   end;
   datalines;
45 63 113
.  75 150
62 .   98
;

proc print data=changed;
   title 'Number of Books Sold';
run;
```

The following output shows the CHANGED data set.

Output 25.1 Using an Array Statement to Process Missing Data Values

```
                          Number of Books Sold                          1

                Obs    Reference    Usage    Introduction

                 1         45         63          113
                 2          0         75          150
                 3         62          0           98
```

Defining the Number of Elements in an Array

When you define the number of elements in an array, you can either use an asterisk enclosed by braces ({*}), brackets ([*]), or parentheses ((*)) to count the number of elements or to specify the number of elements. You must list each array element if you use the asterisk to designate the number of elements. In the following example, the array C1TEMP references five variables with temperature measures.

```
array c1temp{*} c1t1 c1t2 c1t3 c1t4 c1t5;
```

If you specify the number of elements explicitly, you can omit the names of the variables or array elements in the ARRAY statement. SAS then creates variable names by concatenating the array name with the numbers 1, 2, 3, and so on. If a variable name in the series already exists, SAS uses that variable instead of creating a new one. In the following example, the array c1t references five variables: c1t1, c1t2, c1t3, c1t4, and c1t5.

```
array c1t{5};
```

Rules for Referencing Arrays

Before you make any references to an array, an ARRAY statement must appear in the same DATA step that you used to create the array. Once you have created the array, you can

- use an array reference anywhere that you can write a SAS expression
- use an array reference as the arguments of some SAS functions
- use a subscript enclosed in braces, brackets, or parentheses to reference an array
- use the special array subscript asterisk (*) to refer to all variables in an array in an INPUT or PUT statement or in the argument of a function.

  Note: You cannot use the asterisk with _TEMPORARY_ arrays. △

An array definition is in effect only for the duration of the DATA step. If you want to use the same array in several DATA steps, you must redefine the array in each step. You can, however, redefine the array with the same variables in a later DATA step by using a macro variable. A macro variable is useful for storing the variable names you need, as shown in this example:

```
%let list=NC SC GA VA;

data one;
   array state(*) &list;
   … more SAS statements …
run;

data two;
   array state(*) &list;
   … more SAS statements …
run;
```

Variations on Basic Array Processing

Determining the Number of Elements in an Array Efficiently

The DIM function in the iterative DO statement returns the number of elements in a one-dimensional array or the number of elements in a specified dimension of a multidimensional array, when the lower bound of the dimension is 1. Use the DIM function to avoid changing the upper bound of an iterative DO group each time you change the number of elements in the array.

The form of the DIM function is as follows:

DIM*n*(*array-name*)

where *n* is the specified dimension that has a default value of 1.

You can also use the DIM function when you specify the number of elements in the array with an asterisk. Here are some examples of the DIM function:

- **do i=1 to dim(days);**
- **do i=1 to dim4(days) by 2;**

DO WHILE and DO UNTIL Expressions

Arrays are often processed in iterative DO loops that use the array reference in a DO WHILE or DO UNTIL expression. In this example, the iterative DO loop processes the elements of the array named TREND.

```
data test;
   array trend{5} x1-x5;
   input x1-x5 y;
   do i=1 to 5 while(trend{i}<y);
   … more SAS statements …
   end;
   datalines;
… data lines …
;
```

Using Variable Lists to Define an Array Quickly

SAS reserves the following three names for use as variable list names:

CHARACTER

NUMERIC

ALL

You can use these variable list names to reference variables that have been previously defined in the same DATA step. The _CHARACTER_ variable lists character values only. The _NUMERIC_ variable lists numeric values only. The _ALL_ variable lists either all character or all numeric values, depending on how you previously defined the variables.

For example, the following INPUT statement reads in variables X1 through X3 as character values using the $8. informat, and variables X4 through X5 as numeric variables. The following ARRAY statement uses the variable list _CHARACTER_ to include only the character variables in the array. The asterisk indicates that SAS will determine the subscript by counting the variables in the array.

```
input (X1-X3) ($8.) X4-X5;
array item {*} _character_;
```

You can use the _NUMERIC_ variable in your program if, for example, you need to convert currency. In this application, you do not need to know the variable names. You need only to convert all values to the new currency.

For more information about variable lists, see the ARRAY statement in *SAS Language Reference: Dictionary*.

Multidimensional Arrays

Grouping Variables in a Multidimensional Array

To create a multidimensional array, place the number of elements in each dimension after the array name in the form {*n*, ... } where *n* is required for each dimension of a multidimensional array.

From right to left, the rightmost dimension represents columns; the next dimension represents rows. Each position farther left represents a higher dimension. The following ARRAY statement defines a two-dimensional array with two rows and five columns. The array contains ten variables: five temperature measures (t1 through t5) from two cities (c1 and c2):

```
array temprg{2,5} c1t1-c1t5 c2t1-c2t5;
```

SAS places variables into a multidimensional array by filling all rows in order, beginning at the upper-left corner of the array (known as row-major order). You can think of the variables as having the following arrangement:

```
c1t1 c1t2 c1t3 c1t4 c1t5
c2t1 c2t2 c2t3 c2t4 c2t5
```

To refer to the elements of the array later with an array reference, you can use the array name and subscripts. The following table lists some of the array references for the previous example:

Variable	Array reference
c1t1	temprg{1,1}
c1t2	temprg{1,2}
c2t2	temprg{2,2}
c2t5	temprg{2,5}

Using Nested DO Loops

Multidimensional arrays are usually processed inside nested DO loops. As an example, the following is one form that processes a two-dimensional array:

DO *index-variable-1*=1 **TO** *number-of-rows*;
  DO *index-variable-2*=1 **TO** *number-of-columns*;
    ... *more SAS statements* ...
  END;
END;

An array reference can use two or more index variables as the subscript to refer to two or more dimensions of an array. Use the following form:

array-name {*index-variable-1, ...,index-variable-n*}

The following example creates an array that contains ten variables- five temperature measures (t1 through t5) from two cities (c1 and c2). The DATA step contains two DO loops.

- The outer DO loop (DO I=1 TO 2) processes the inner DO loop twice.
- The inner DO loop (DO J=1 TO 5) applies the ROUND function to all the variables in one row.

For each iteration of the DO loops, SAS substitutes the value of the array element corresponding to the current values of I and J.

```
options nodate pageno=1 linesize=80 pagesize=60;

data temps;
   array temprg{2,5} c1t1-c1t5 c2t1-c2t5;
   input c1t1-c1t5 /
         c2t1-c2t5;
   do i=1 to 2;
     do j=1 to 5;
       temprg{i,j}=round(temprg{i,j});
     end;
   end;
   datalines;
89.5 65.4 75.3 77.7 89.3
73.7 87.3 89.9 98.2 35.6
75.8 82.1 98.2 93.5 67.7
101.3 86.5 59.2 35.6 75.7
;

proc print data=temps;
   title 'Temperature Measures for Two Cities';
run;
```

The following data set TEMPS contains the values of the variables rounded to the nearest whole number.

Output 25.2 Using a Multidimensional Array

Temperature Measures for Two Cities 1

Obs	c1t1	c1t2	c1t3	c1t4	c1t5	c2t1	c2t2	c2t3	c2t4	c2t5	i	j
1	90	65	75	78	89	74	87	90	98	36	3	6
2	76	82	98	94	68	101	87	59	36	76	3	6

The previous example can also use the DIM function to produce the same result:

```
do i=1 to dim1(temprg);
   do j=1 to dim2(temprg);
      temprg{i,j}=round(temprg{i,j});
   end;
end;
```

The value of DIM1(TEMPRG) is 2; the value of DIM2(TEMPRG) is 5.

Specifying Array Bounds

Identifying Upper and Lower Bounds

Typically in an ARRAY statement, the subscript in each dimension of the array ranges from 1 to *n*, where *n* is the number of elements in that dimension. Thus, 1 is the lower bound and *n* is the upper bound of that dimension of the array. For example, in the following array, the lower bound is 1 and the upper bound is 4:

```
array new{4} Jackson Poulenc Andrew Parson;
```

In the following ARRAY statement, the bounds of the first dimension are 1 and 2 and those of the second dimension are 1 and 5:

```
array test{2,5} test1-test10;
```

Bounded array dimensions have the following form:

{<*lower-1*:>*upper-1*<,...<*lower-n*:>*upper-n*>}

Therefore, you can also write the previous ARRAY statements as follows:

```
array new{1:4} Jackson Poulenc Andrew Parson;
array test{1:2,1:5} test1-test10;
```

For most arrays, 1 is a convenient lower bound, so you do not need to specify the lower bound. However, specifying both the lower and the upper bounds is useful when the array dimensions have beginning points other than 1.

In the following example, ten variables are named Year76 through Year85. The following ARRAY statements place the variables into two arrays named FIRST and SECOND:

```
array first{10} Year76-Year85;
array second{76:85} Year76-Year85;
```

In the first ARRAY statement, the element first{4} is variable Year79, first{7} is Year82, and so on. In the second ARRAY statement, element second{79} is Year79 and second{82} is Year82.

To process the array names SECOND in a DO group, be sure that the range of the DO loop matches the range of the array as follows:

```
do i=76 to 85;
   if second{i}=9 then second{i}=.;
end;
```

Determining Array Bounds: LBOUND and HBOUND Functions

You can use the LBOUND and HBOUND functions to determine array bounds. The LBOUND function returns the lower bound of a one-dimensional array or the lower bound of a specified dimension of a multidimensional array. The HBOUND function returns the upper bound of a one-dimensional array or the upper bound of a specified dimension of a multidimensional array.

The form of the LBOUND and HBOUND functions is as follows:

LBOUND*n*(*array-name*)

HBOUND*n*(*array-name*)

where

n
is the specified dimension and has a default value of 1.

You can use the LBOUND and HBOUND functions to specify the starting and ending values of the iterative DO loop to process the elements of the array named SECOND:

```
do i=lbound{second} to hbound{second};
   if second{i}=9 then second{i}=.;
end;
```

In this example, the index variable in the iterative DO statement ranges from 76 to 85.

When to Use the HBOUND Function instead of the DIM Function

The following ARRAY statement defines an array containing a total of five elements, a lower bound of 72, and an upper bound of 76. It represents the calendar years 1972 through 1976:

```
array years{72:76} first second third fourth fifth;
```

To process the array named YEARS in an iterative DO loop, be sure that the range of the DO loop matches the range of the array as follows:

```
do i=lbound(years) to hbound(years);
   if years{i}=99 then years{i}=.;
end;
```

The value of LBOUND(YEARS) is 72; the value of HBOUND(YEARS) is 76.

For this example, the DIM function would return a value of 5, the total count of elements in the array YEARS. Therefore, if you used the DIM function instead of the HBOUND function for the upper bound of the array, the statements inside the DO loop would not have executed.

Specifying Bounds in a Two-Dimensional Array

The following list contains 40 variables named X60 through X99. They represent the years 1960 through 1999.

```
X60   X61   X62   X63   X64   X65   X66   X67   X68   X69
X70   X71   X72   X73   X74   X75   X76   X77   X78   X79
X80   X81   X82   X83   X84   X85   X86   X87   X88   X89
X90   X91   X92   X93   X94   X95   X96   X97   X98   X99
```

The following ARRAY statement arranges the variables in an array by decades. The rows range from 6 through 9, and the columns range from 0 through 9.

```
array X{6:9,0:9} X60-X99;
```

In array X, variable X63 is element X{6,3} and variable X89 is element X{8,9}. To process array X with iterative DO loops, use one of these methods:

Method 1:

```
do i=6 to 9;
   do j=0 to 9;
      if X{i,j}=0 then X{i,j}=.;
   end;
end;
```

Method 2:

```
do i=lbound1(X) to hbound1(X);
   do j=lbound2(X) to hbound2(X);
      if X{i,j}=0 then X{i,j}=.;
   end;
end;
```

Both examples change all values of 0 in variables X60 through X99 to missing. The first example sets the range of the DO groups explicitly, and the second example uses the LBOUND and HBOUND functions to return the bounds of each dimension of the array.

Examples

Example 1: Using Character Variables in an Array

You can specify character variables and their lengths in ARRAY statements. The following example groups variables into two arrays, NAMES and CAPITALS. The dollar sign ($) tells SAS to create the elements as character variables. If the variables have already been declared as character variables, a dollar sign in the array is not necessary. The INPUT statement reads all the variables in array NAMES.

The statement inside the DO loop uses the UPCASE function to change the values of the variables in array NAMES to uppercase and then store the uppercase values in the variables in the CAPITALS array.

```
options nodate pageno=1 linesize=80 pagesize=60;

data text;
   array names{*} $ n1-n10;
   array capitals{*} $ c1-c10;
   input names{*};
      do i=1 to 10;
         capitals{i}=upcase(names{i});
      end;
   datalines;
smithers michaels gonzalez hurth frank bleigh
rounder joseph peters sam
;

proc print data=text;
   title 'Names Changed from Lowercase to Uppercase';
run;
```

The following output shows the TEXT data set.

Output 25.3 Using Character Variables in an Array

```
                  Names Changed from Lowercase to Uppercase                  1

Obs    n1       n2       n3     n4    n5     n6     n7      n8     n9   n10

 1  smithers michaels gonzalez hurth frank bleigh rounder joseph peters sam

Obs    c1       c2       c3     c4    c5     c6     c7      c8     c9   c10  i

 1  SMITHERS MICHAELS GONZALEZ HURTH FRANK BLEIGH ROUNDER JOSEPH PETERS SAM 11
```

Example 2: Assigning Initial Values to the Elements of an Array

This example creates variables in the array TEST and assigns them the initial values 90, 80, and 70. It reads values into another array named SCORE and compares each element of SCORE to the corresponding element of TEST. If the value of the element in SCORE is greater than or equal to the value of the element in TEST, the variable NewScore is assigned the value in the element SCORE, and the OUTPUT statement writes the observation to the SAS data set.

The INPUT statement reads a value for the variable named ID and then reads values for all the variables in the SCORE array.

```
options nodate pageno=1 linesize=80 pagesize=60;

data score1(drop=i);
   array test{3} t1-t3 (90 80 70);
   array score{3} s1-s3;
   input id score{*};
   do i=1 to 3;
      if score{i}>=test{i} then
         do;
            NewScore=score{i};
            output;
         end;
   end;
   datalines;
1234  99 60 82
5678  80 85 75
;

proc print noobs data=score1;
   title 'Data Set SCORE1';
run;
```

The following output shows the SCORE1 data set.

Output 25.4 Assigning Initial Values to the Elements of an Array

Data Set SCORE1 1

t1	t2	t3	s1	s2	s3	id	New Score
90	80	70	99	60	82	1234	99
90	80	70	99	60	82	1234	82
90	80	70	80	85	75	5678	85
90	80	70	80	85	75	5678	75

Example 3: Creating an Array for Temporary Use in the Current DATA Step

When elements of an array are constants that are needed only for the duration of the DATA step, you can omit variables from an array group and instead use temporary array elements. You refer to temporary data elements by the array name and dimension. Although they behave like variables, temporary array elements do not have names, and they do not appear in the output data set. Temporary array elements are automatically retained, instead of being reset to missing at the beginning of the next iteration of the DATA step.

To create a temporary array, use the _TEMPORARY_ argument. The following example creates a temporary array named TEST:

```
options nodate pageno=1 linesize=80 pagesize=60;

data score2(drop=i);
      array test{3} _temporary_ (90 80 70);
      array score{3} s1-s3;
      input id score{*};
         do i=1 to 3;
            if score{i}>=test{i} then
               do;
                  NewScore=score{i};
                  output;
               end;
         end;
      datalines;
   1234  99 60 82
   5678  80 85 75
   ;

   proc print noobs data=score2;
      title 'Data Set SCORE2';
   run;
```

The following output shows the SCORE2 data set.

Output 25.5 Using _TEMPORARY_ Arrays

```
                              Data Set SCORE2                                1

                                                    New
                        s1    s2    s3     id     Score

                        99    60    82    1234     99
                        99    60    82    1234     82
                        80    85    75    5678     85
                        80    85    75    5678     75
```

Example 4: Performing an Action on All Numeric Variables

This example multiplies all the numeric variables in array TEST by 3.

```
options nodate pageno=1 linesize=80 pagesize=60;

data sales;
   infile datalines;
   input Value1 Value2 Value3 Value4;
   datalines;
11 56 58 61
22 51 57 61
22 49 53 58
;

data convert(drop=i);
   set sales;
   array test{*} _numeric_;
   do i=1 to dim(test);
      test{i} = (test{i}*3);
   end;
run;

proc print data=convert;
   title 'Data Set CONVERT';
run;
```

The following output shows the CONVERT data set.

Output 25.6 Output From Using a _NUMERIC_ Variable List

```
                              Data Set CONVERT                               1

                  Obs    Value1    Value2    Value3    Value4

                   1        33        168       174       183
                   2        66        153       171       183
                   3        66        147       159       174
```

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS Language Reference: Concepts*, Cary, NC: SAS Institute Inc., 1999. 554 pages.

SAS Language Reference: Concepts

ISBN 1–58025–441–1

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st printing, November 1999