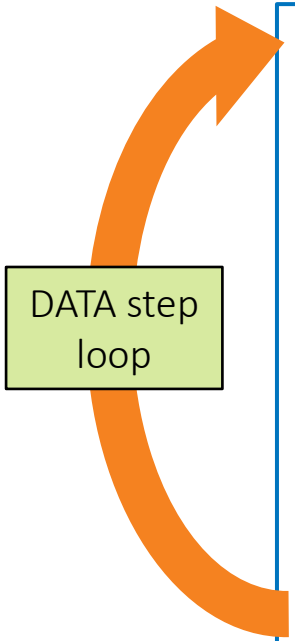
STAT604 SAS Lesson 14

Processing Repetitive Code

Using Iterative DO Loops – Prep Guide Chapter 11

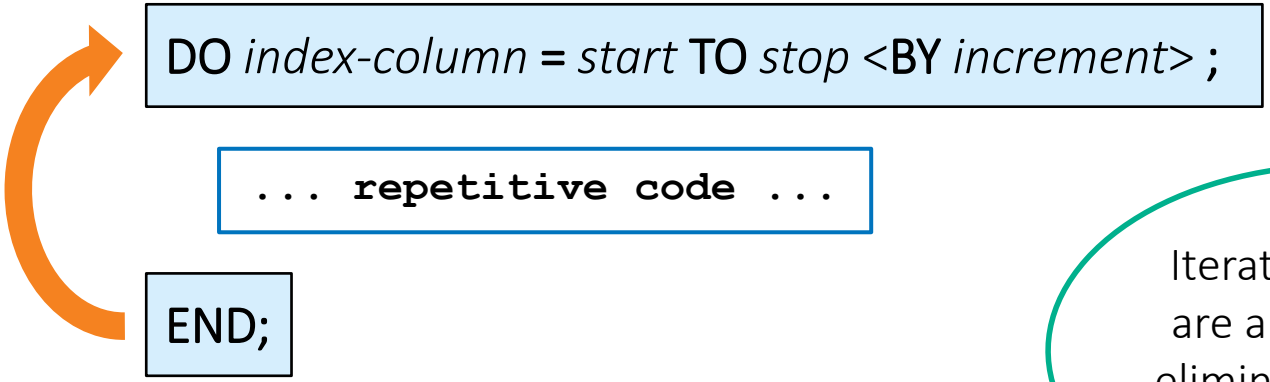
Processing Repetitive Code

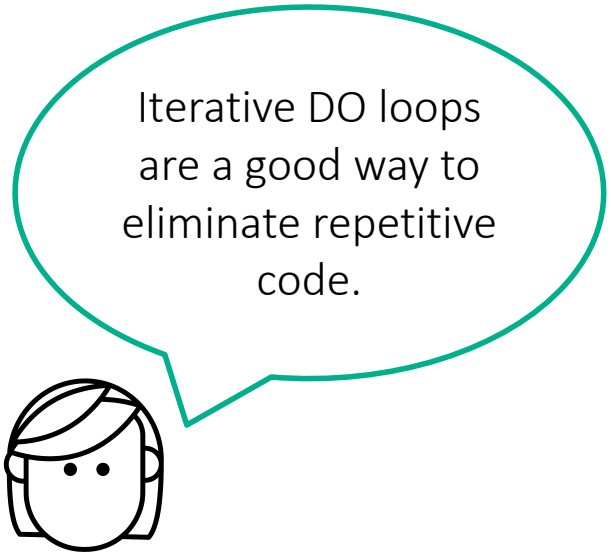



```
data forecast;
     set sashelp.shoes(rename=(Sales=ProjectedSales));
     Year=1;
     ProjectedSales=ProjectedSales*1.05;
     output;
     Year=2;
     ProjectedSales=ProjectedSales*1.05;
     output;
     Year=3;
     ProjectedSales=ProjectedSales*1.05;
     output;
     keep Region Product Subsidiary Year ProjectedSales;
     format ProjectedSales dollar10.;
run;
```

```
NOTE: There were 395 observations read from the data set SASHELP.SHOES.
NOTE: The data set WORK.FORECAST has 1185 observations and 5 variables
```

Iterative DO Loop





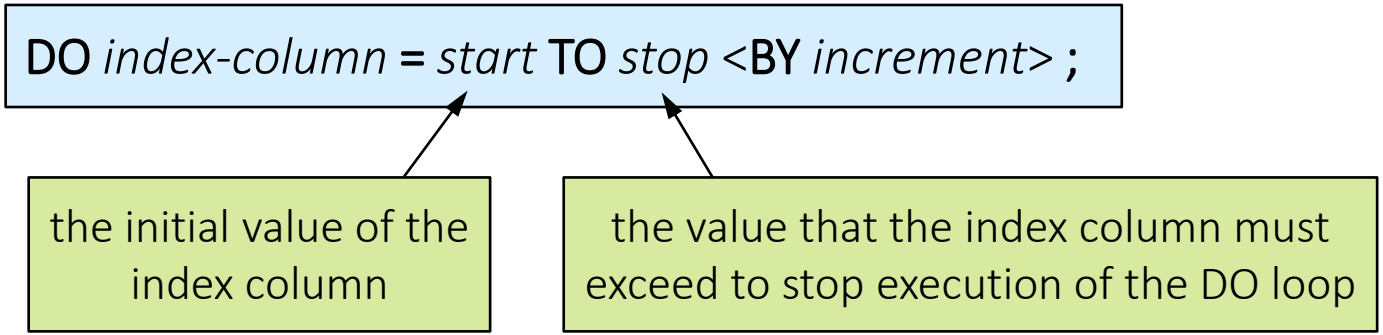
Iterative DO Statement

DO *index-column* = *start* TO *stop* <BY *increment*> ;

column whose value controls DO loop execution

```
do Year =
```

Iterative DO Statement




```
do Year = 1 to 3;
```

Iterative DO Statement

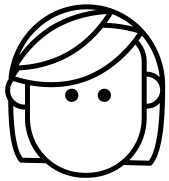
DO *index-column* = *start* TO *stop* <BY *increment*> ;

a positive or negative number for incrementing the value of the index column

```
do Year = 1 to 3;
```

```
do Year = 1 to 3 by 1;
```

If you don't specify an increment, the default is 1.



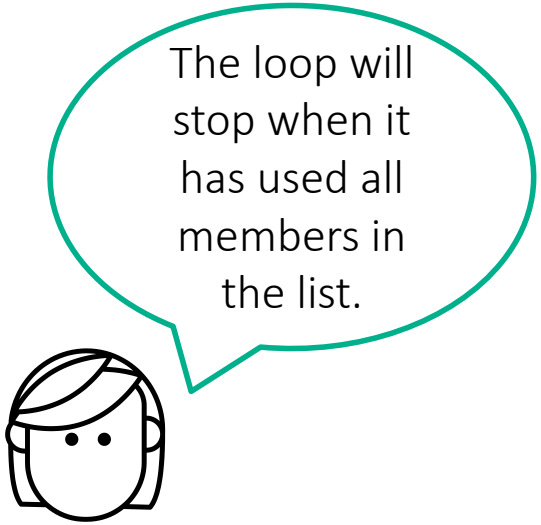
Iterative DO Statement

DO *index-column* = *list*;

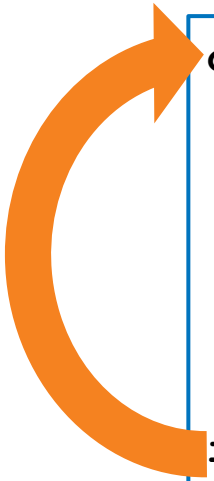
A list of numeric or character values can also be used as index values

```
do Year = 1, 2, 3;
```

```
do Mon = 'Jan', 'Feb';
```



Executing an Iterative DO Loop



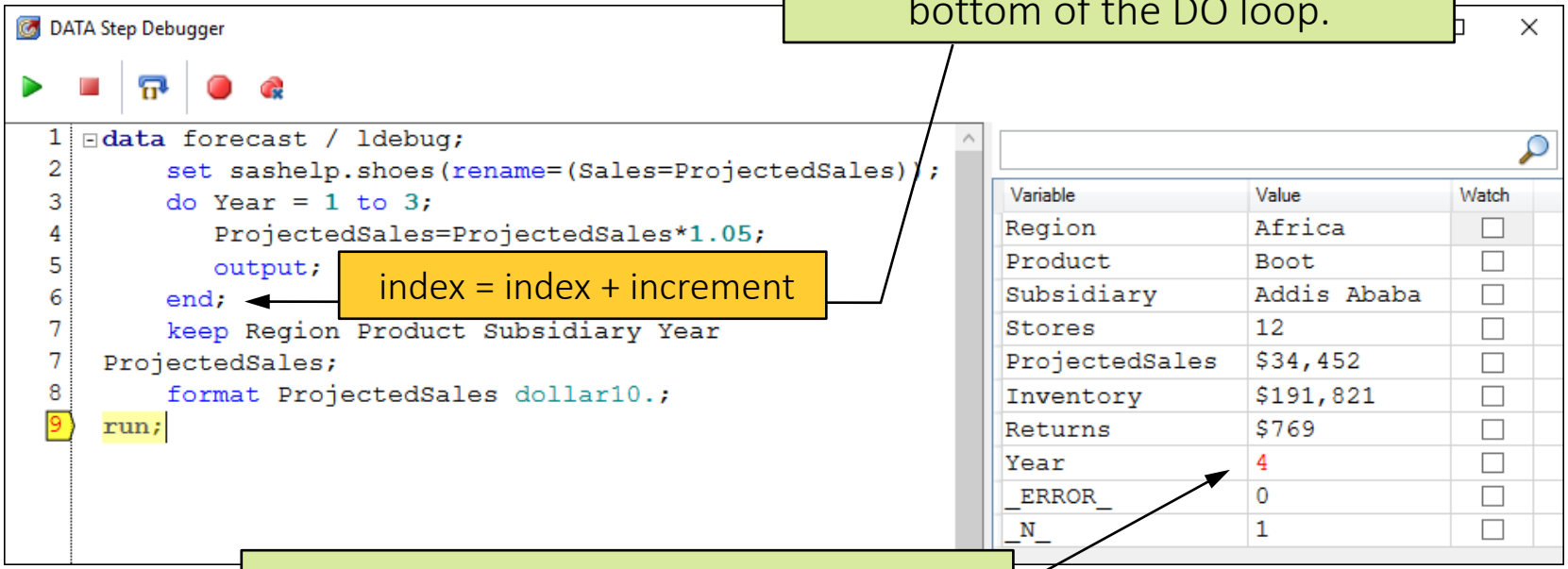
```
data forecast;
    set sashelp.shoes(rename=(Sales=ProjectedSales));
    do Year = 1 to 3;
       ProjectedSales=ProjectedSales*1.05;
       output;
    end;
   keep Region Product Subsidiary Year ProjectedSales;
   format ProjectedSales dollar10.;
run;
```

Executing an Iterative DO Loop

This demonstration illustrates using the DATA step debugger in PC SAS to see the execution of an iterative DO loop.

Executing an Iterative DO Loop

Year is incremented by 1 at the bottom of the DO loop.



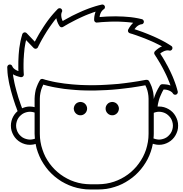
The DO loop terminates when **Year** exceeds the stop value of 3.

Scenario

```
data YearlySavings;
     Amount=200;
     do Month=1 to 12;
        Savings+Amount;
        output;
     end;
     format Savings 12.2;
run;
```

Amount	Month	Savings
200	1	200.00
200	2	400.00
200	3	600.00
200	4	800.00
200	5	1000.00
200	6	1200.00
200	7	1400.00
200	8	1600.00
200	9	1800.00
200	10	2000.00
200	11	2200.00
200	12	2400.00

How much money is in savings each month if we save $200 per month for a year?



Activity

1. Run the program on the previous slide. How much is in savings at month 12?
2. Delete the OUTPUT statement and run the program again.
3. How many rows are created?
4. What is the value of **Month**?
5. What is the value of **Savings**?

Activity – Correct Answer

1. How much is in savings at month 12? **2426.16**

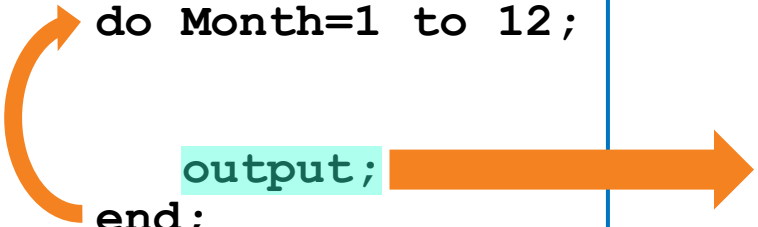
	Amount	Month	Savings
8	200	8	1612.05
9	200	9	1815.07
10	200	10	2018.43
11	200	11	2222.12
12	200	12	2426.16

3. How many rows are created? **one**
4. What is the value of **Month**? **13**
5. What is the value of **Savings**? **2426.16**

Amount	Month	Savings
200	13	2426.16

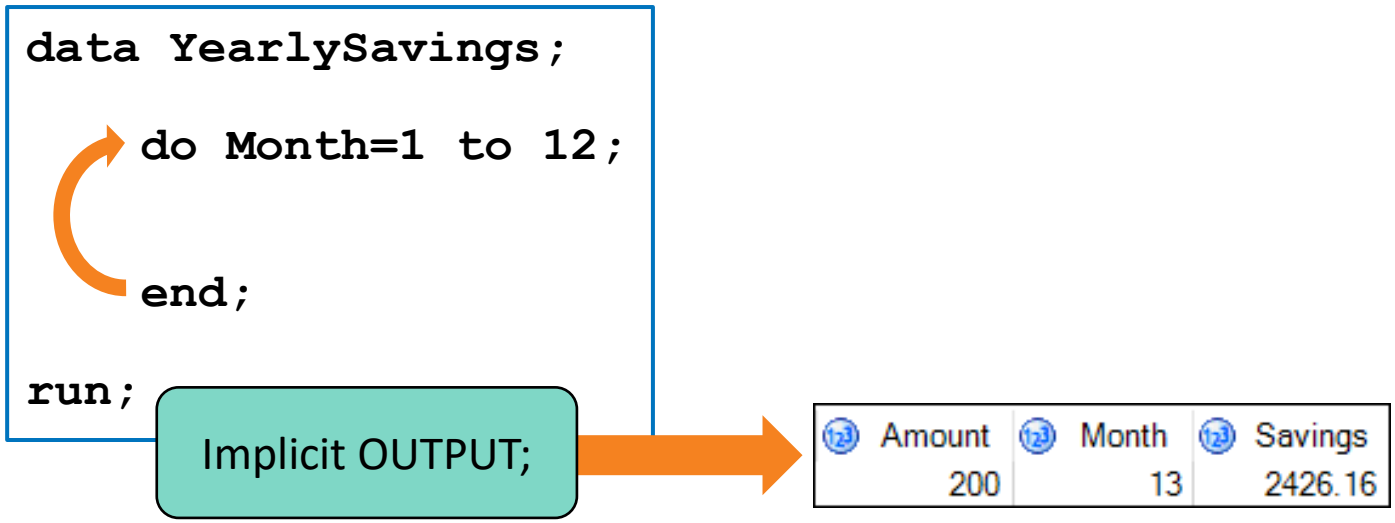
Output inside the DO Loop

```
data YearlySavings;

   do Month=1 to 12;


      output;
   end;

run;
```

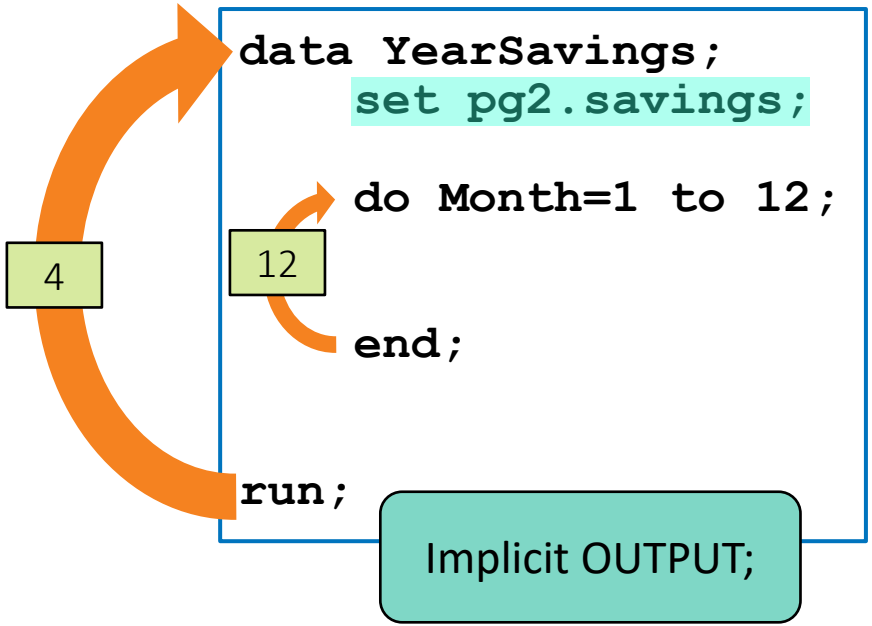


Amount	Month	Savings
200	1	200.33
200	2	401.00
200	3	602.00
200	4	803.34
200	5	1005.01
200	6	1207.02
200	7	1409.36
200	8	1612.05
200	9	1815.07
200	10	2018.43
200	11	2222.12
200	12	2426.16

Output outside the DO Loop



DO Loop with an Input Table



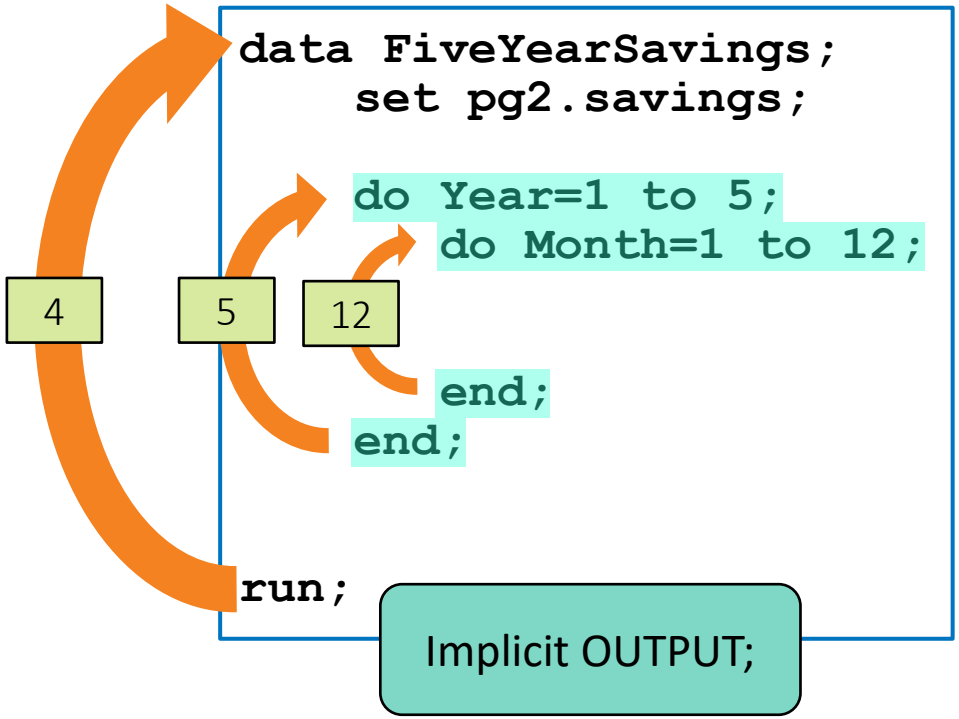
pg2.savings

Name	Amount
James	250
Linda	300
Mary	275
Robert	350

work.YearSavings

Name	Amount	Savings
James	250	3,032.70
Linda	300	3,639.24
Mary	275	3,335.97
Robert	350	4,245.78

Nested DO Loops



pg2.savings

Name	Amount
James	250
Linda	300
Mary	275
Robert	350

work.FiveYearSavings

Name	Amount	Savings
James	250	15,788.11
Linda	300	18,945.73
Mary	275	17,366.92
Robert	350	22,103.35

Processing Repetitive Code

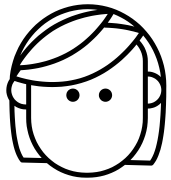
Using Conditional DO Loops

Iterative DO Loop

```
data YearSavings;
     set pg2.savings;
     Savings=0;
     do Month=1 to 12;
        Savings+Amount;
        Savings+(Savings*0.02/12);
     end;
     drop Month;
     format Savings comma12.2;
run;
```

12

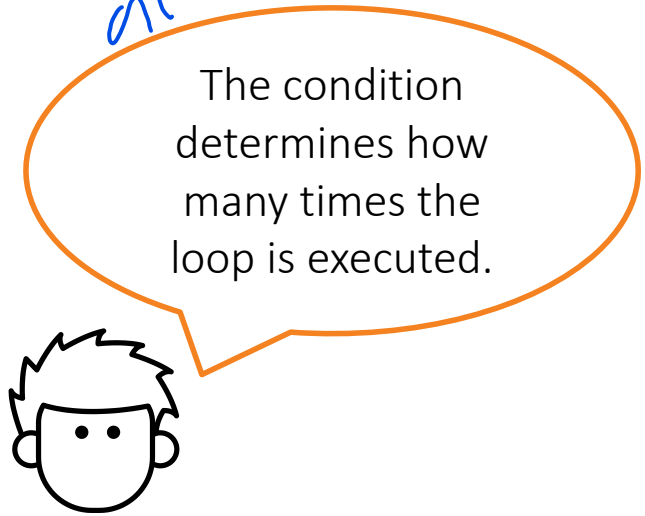
An iterative DO loop works great when you know exactly how many iterations to perform.



Conditional DO Loop

```
data Savings3K;
     set pg2.savings;
     Month=0;
     Savings=0;
     do until (Savings>3000);
        Month+1;
        Savings+Amount;
        Savings+(Savings*0.02/12);
     end;
     format Savings comma12.2;
run;
```

?



Conditional DO Loops

executes repetitively *until* a condition is true

```
DO UNTIL (expression) ;
   ... repetitive code ...
END;
```

executes repetitively *while* a condition is true

```
DO WHILE (expression) ;
   ... repetitive code ...
END;
```

Conditional DO Loops

executes repetitively *until* a condition is true

```
do until (Savings>3000);
    Month+1;
    Savings+Amount;
    Savings+(Savings*0.02/12);
end;
```

executes repetitively *while* a condition is true

```
do while (Savings<=3000);
    Month+1;
    Savings+Amount;
    Savings+(Savings*0.02/12);
end;
```

Checking the Condition

checks the condition at the top of the loop

```
DO WHILE (expression) ;
   ... repetitive code ...
END;
```

```
DO UNTIL (expression) ;
   ... repetitive code ...
END;
```

checks the condition at the bottom of the loop

Checking the Condition

pg2.savings2

Name	Amount	Savings
James	250	1250
Linda	300	3600
Mary	275	2200
Robert	350	1750

```
do until (Savings>3000);
```

Name	Amount	Savings	Month
James	250	3,026.36	7
Linda	300	3,906.50	1
Mary	275	3,038.77	3
Robert	350	3,167.54	4

DO UNTIL always executes once.

```
do while (Savings<=3000);
```

Name	Amount	Savings	Month
James	250	3,026.36	7
Linda	300	3,600.00	0
Mary	275	3,038.77	3
Robert	350	3,167.54	4

DO WHILE executes only if the condition is true.

Using Conditional DO Loops

This demonstration compares the operation of the UNTIL and WHILE loops.

Combining Iterative and Conditional DO Loops

DO *index-column* **=** *start* **TO** *stop* <**BY** *increment*> **UNTIL** | **WHILE** *(expression)* **;**

iterative

conditional

The DO loop stops executing when the stop value is exceeded or the condition is met, whichever is first.

Iterative and Conditional DO Loop

```
do Month=1 to 12 until (Savings>5000);
    Savings+Amount;
    Savings+(Savings*0.02/12);
end;
```

At the bottom of loop, the condition is checked *before* the index column is incremented.

```
do Month=1 to 12 while (Savings<=5000);
    Savings+Amount;
    Savings+(Savings*0.02/12);
end;
```

At the top of loop, the condition is checked.

At the bottom of loop, the index column is incremented.

Combining Iterative and Conditional DO Loops

This demonstration illustrates modifying an existing DATA step with variations of the iterative and conditional DO loop.

Processing Repetitive Code

SAS Array Processing

Array Processing

You can use arrays to simplify programs that do the following:

- perform repetitive calculations
- create many variables with the same attributes
- read data
- compare variables
- perform a table lookup

Business Scenario

The **orion.employee_donations** data set contains quarterly contribution data for each employee. Orion management is considering a 25 percent matching program. Calculate each employee's quarterly contribution, including the proposed company supplement.

Employee_ID	Qtr1	Qtr2	Qtr3	Qtr4
120265	.	.	.	25
120267	15	15	15	15
120269	20	20	20	20
120270	20	10	5	.
120271	20	20	20	20
120272	10	10	10	10

Performing Repetitive Calculations

```
data charity;
   set orion.employee_donations;
   keep employee_id qtr1-qtr4;
   Qtr1=Qtr1*1.25;
   Qtr2=Qtr2*1.25;
   Qtr3=Qtr3*1.25;
   Qtr4=Qtr4*1.25;
run;
proc print data=charity noobs;
run;
```

Partial PROC PRINT Output

Employee_ID	Qtr1	Qtr2	Qtr3	Qtr4
120265	.	.	.	31.25
120267	18.75	18.75	18.75	18.75
120269	25.00	25.00	25.00	25.00
120270	25.00	12.50	6.25	.

Performing Repetitive Calculations

The four calculations cannot be replaced by a single calculation inside a DO loop because they are not identical.

```
data charity;
   set orion.employee_donations;
   keep employee_id qtr1-qtr4;
   Qtr1=Qtr1*1.25;
   Qtr2=Qtr2*1.25;
   Qtr3=Qtr3*1.25;
   Qtr4=Qtr4*1.25;
run;
proc print data=charity noobs;
run;
```

```
do i=1 to 4;
      ?
end;
```

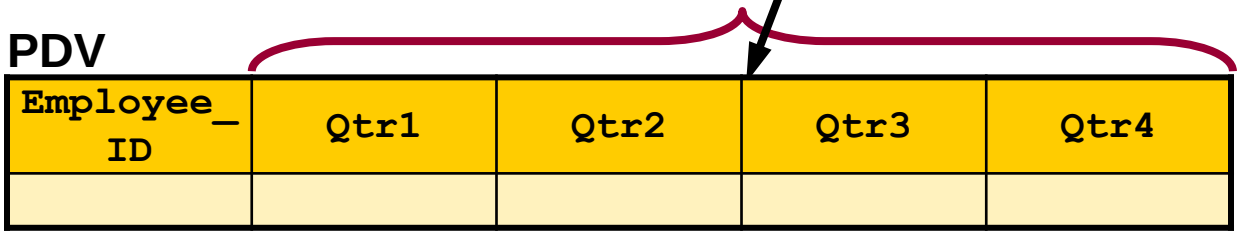
A SAS array can be used to simplify this code.

Use Arrays to Simplify Repetitive Calculations

An array provides an alternate way to access values in the PDV, which simplifies repetitive calculations.

```
data charity;
   set orion.employee_donations;
   keep employee_id qtr1-qtr4;
   Qtr1=Qtr1*1.25;
   Qtr2=Qtr2*1.25;
   Qtr3=Qtr3*1.25;
   Qtr4=Qtr4*1.25;
run;
```

An array can be used to access Qtr1-Qtr4.



What Is a SAS Array?

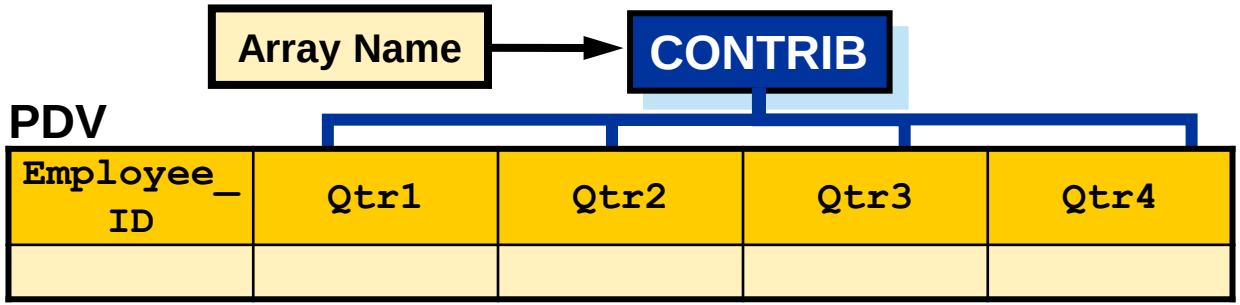
A *SAS array*

- is a temporary grouping of SAS variables that are arranged in a particular order
- is identified by an *array name*
- must contain all numeric or all character variables
- exists only for the duration of the current DATA step
- is **not** a variable.

must all be of the same type.

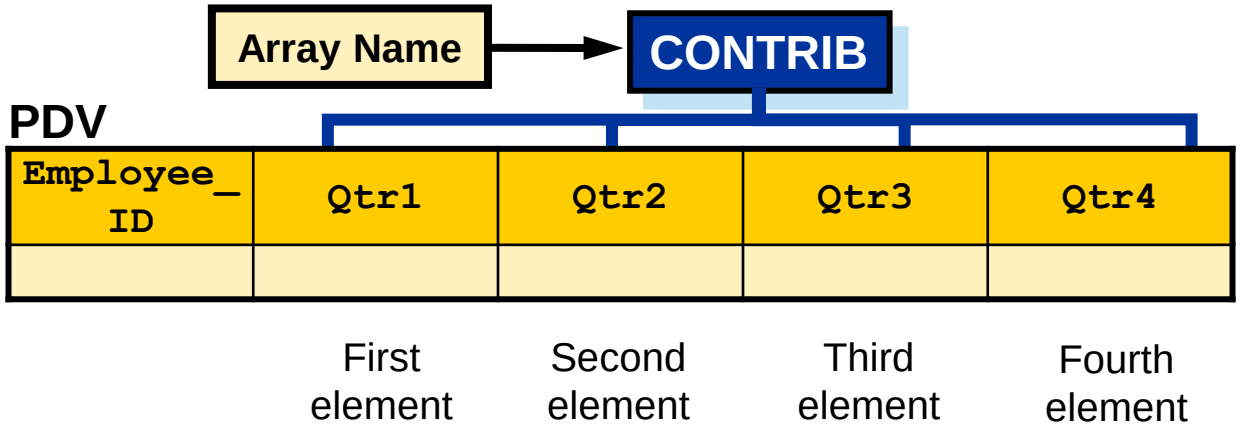
Why Use a SAS Array?

Create an array named **Contrib** and use it to access the four numeric variables, **Qtr1** – **Qtr4**.



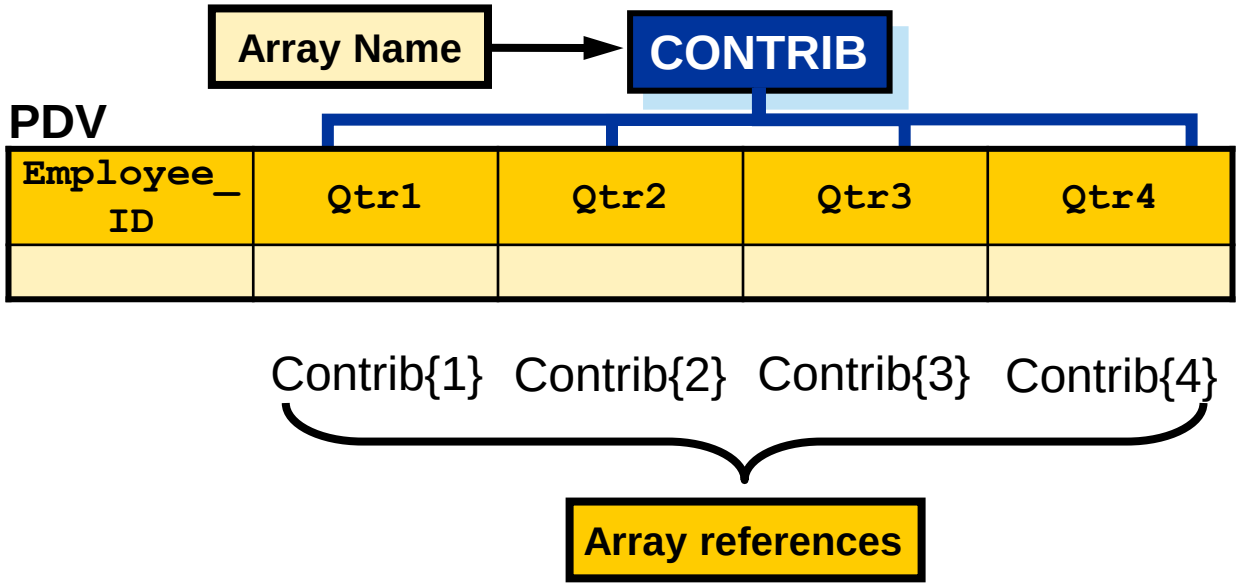
Array Elements

Each value in an array is called an *element.*



Referencing Array Elements

Each element is identified by a *subscript* that represents its position in the array. When you use an *array reference*, the corresponding value is substituted for the reference.



The ARRAY Statement

The ARRAY statement is a compile-time statement that defines the elements in an array. The elements are created if they do not already exist in the PDV.

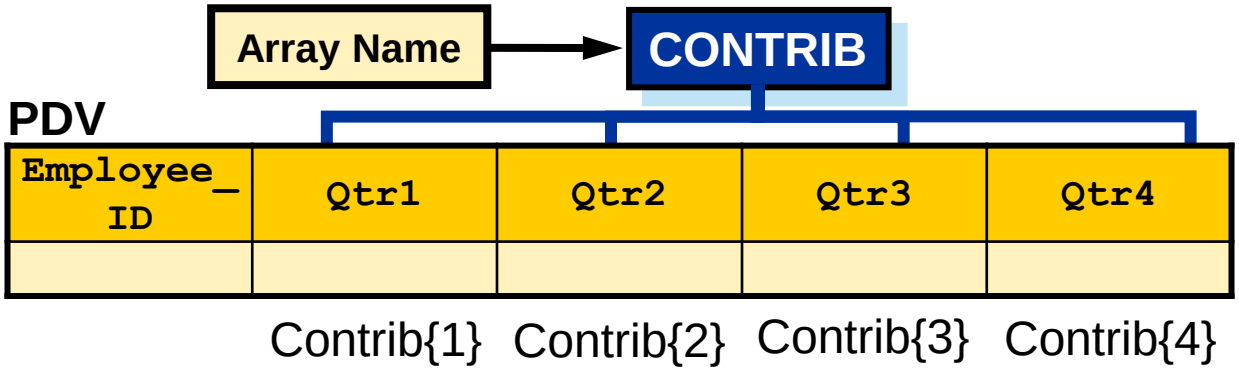
ARRAY *array-name* {*subscript*} <$> <*length*> <*array-elements*>**;**

{subscript}	the number of elements
$	indicates character elements
length	the length of elements
array-elements	the names of elements

Defining an Array

The following ARRAY statement defines an array, **Contrib**, to access the four quarterly contribution variables.

```
array Contrib{4} qtr1 qtr2 qtr3 qtr4;
```



Defining an Array

An alternate syntax uses an asterisk instead of a subscript. SAS determines the subscript by counting the variables in the element-list. The element-list must be included.

```
array Contrib{*} qtr1 qtr2 qtr3 qtr4;
```

The alternate syntax is often used when the array elements are defined with a SAS variable list.

```
array Contrib{*} qtr:;
```

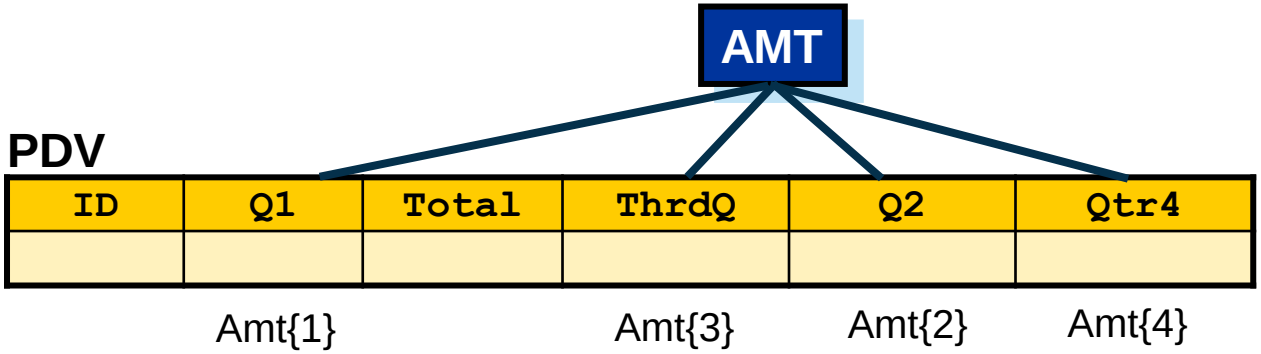
very flexible

Defining an Array

Variables that are elements of an array do not need the following:

- to have similar, related, or numbered names
- to be stored sequentially
- to be adjacent

```
array Amt{*} Q1 Q2 ThrdQ Qtr4;
```



Quiz

What do you think would cause an error if you executed the code below?

```
data charity(keep=employee_id qtr1-qtr4);
   set orion.employee_donations;
   array Contrib1{3} qtr1-qtr4;
   array Contrib2{5} qtr:;
   /* additional SAS statements */
run;
```

Quiz – Correct Answer

The subscript and the number of elements in the list do not agree.

```
data charity(keep=employee_id qtr1-qtr4);
   set orion.employee_donations;
   array Contrib1{3} qtr1-qtr4;
   array Contrib2{5} qtr:;
   /* additional SAS statements */
run;
```

The subscript and element-list must agree.

Partial SAS Log

```
177     array Contrib1{3} qtr1-qtr4;
ERROR: Too many variables defined for the dimension(s) specified
for the array Contrib1.
178     array Contrib2{5} qtr:;
ERROR: Too few variables defined for the dimension(s) specified
for the array Contrib2.
```

Using a DO Loop to Process an Array

Array processing often occurs within an iterative DO loop in the following form:

```
DO index-variable=1 TO number-of-elements-in-array;
    <additional SAS statements>
END;
```

To reference an element, the *index-variable* is often used as a subscript:

```
array-name{index-variable}
```

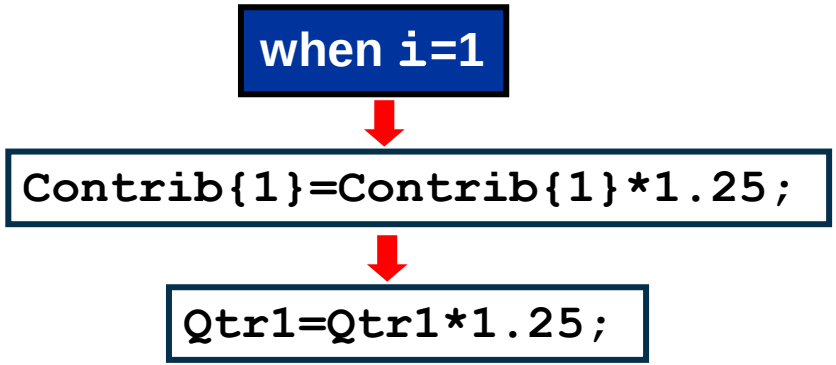
Using a DO Loop to Process an Array

```
data charity;
   set orion.employee_donations;
   keep employee_id qtr1-qtr4;
   array Contrib{4} qtr1-qtr4;
   do i=1 to 4;
      Contrib{i}=Contrib{i}*1.25;
   end;
run;
```

The index variable, **i**, is not written to the output data set because it is not listed in the KEEP statement.

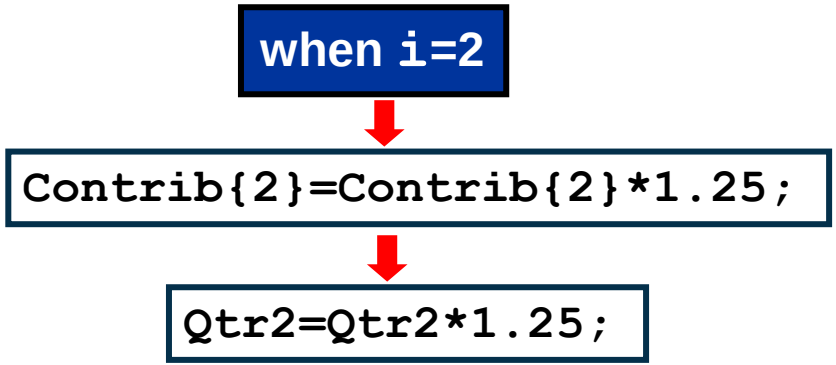
First Iteration of the DO Loop

```
data charity;
   set orion.employee_donations;
   keep employee_id qtr1-qtr4;
   array Contrib{4} qtr1-qtr4;
   do i=1 to 4;
      Contrib{i}=Contrib{i}*1.25;
   end;
run;
```



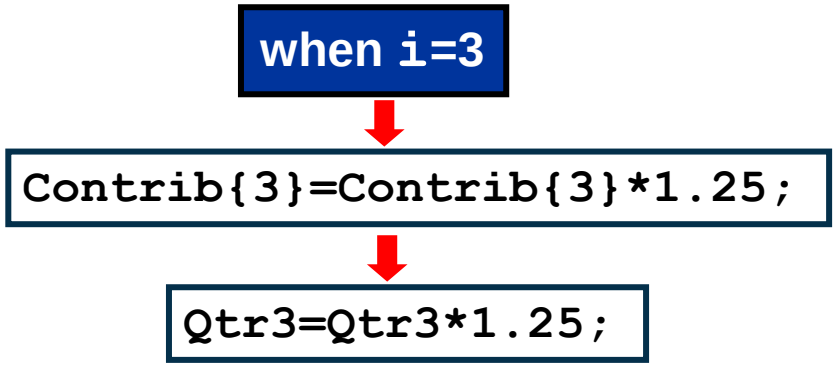
Second Iteration of the DO Loop

```
data charity;
   set orion.employee_donations;
   keep employee_id qtr1-qtr4;
   array Contrib{4} qtr1-qtr4;
   do i=1 to 4;
      Contrib{i}=Contrib{i}*1.25;
   end;
run;
```



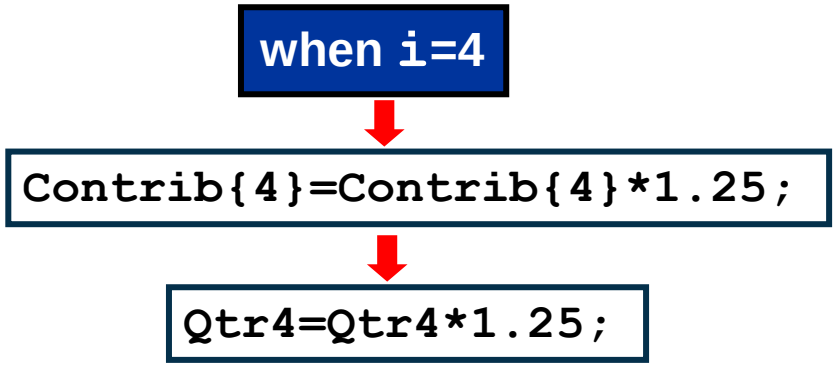
Third Iteration of the DO Loop

```
data charity;
   set orion.employee_donations;
   keep employee_id qtr1-qtr4;
   array Contrib{4} qtr1-qtr4;
   do i=1 to 4;
      Contrib{i}=Contrib{i}*1.25;
   end;
run;
```



Fourth Iteration of the DO Loop

```
data charity;
   set orion.employee_donations;
   keep employee_id qtr1-qtr4;
   array Contrib{4} qtr1-qtr4;
   do i=1 to 4;
      Contrib{i}=Contrib{i}*1.25;
   end;
run;
```



Output: Using a Do Loop to Process an Array

```
proc print data=charity noobs;
run;
```

Partial PROC PRINT Output

Employee_ID	Qtr1	Qtr2	Qtr3	Qtr4
120265	.	.	.	31.25
120267	18.75	18.75	18.75	18.75
120269	25.00	25.00	25.00	25.00
120270	25.00	12.50	6.25	.
120271	25.00	25.00	25.00	25.00
120272	12.50	12.50	12.50	12.50
120275	18.75	18.75	18.75	18.75
120660	31.25	31.25	31.25	31.25
120662	12.50	.	6.25	6.25

Processing Repetitive Code

Using SAS Arrays

Using an Array as a Function Argument

The program below passes an array to the SUM function.

```
data test;
   set orion.employee_donations;
   array val{4} qtr1-qtr4;
   Tot1=sum(of qtr1-qtr4);
   Tot2=sum(of val{*});
run;
proc print data=test;
   var employee_id tot1 tot2;
run;
```

The array is passed as if it were a variable list.

Partial PROC PRINT Output

Obs	Employee_ID	Tot1	Tot2
1	120265	25	25
2	120267	60	60
3	120269	80	80

The DIM Function

The DIM function returns the number of elements in an array. This value is often used as the stop value in a DO loop.

General form of the DIM function:

DIM(*array_name*)

```
array Contrib{*} qtr:;
num_elements=dim(Contrib);

do i=1 to num_elements;
   Contrib{i}=Contrib{i}*1.25;
end;
run;
```

The DIM Function

A call to the DIM function can be used in place of the stop value in the DO loop.

```
data charity;
   set orion.employee_donations;
   keep employee_id qtr1-qtr4;
   array Contrib{*} qtr:;
   do i=1 to dim(Contrib);
      Contrib{i}=Contrib{i}*1.25;
   end;
run;
```

Using an Array to Create Numeric Variables

An ARRAY statement can be used to create new variables in the program data vector.

```
array discount{4} discount1-discount4;
```

If **discount1-discount4** do not exist in the PDV, they are created.

```
array Pct{4};
```

Four new variables are created:

PDV

Pct1 N 8	Pct2 N 8	Pct3 N 8	Pct4 N 8

Using an Array to Create Character Variables

Define an array named **Month** to create six variables to hold character values with a length of 10.

```
array Month{6} $ 10;
```

PDV

Month1 $ 10	Month2 $ 10	Month3 $ 10	Month4 $ 10	Month5 $ 10	Month6 $ 10

Business Scenario

Using **orion.employee_donations** as input, calculate the percentage that each quarterly contribution represents of the employee's total annual contribution. Create four new variables to hold the percentages.

Partial Listing of **orion.employee_donations**

Employee_ID	Qtr1	Qtr2	Qtr3	Qtr4
120265	.	.	.	25
120267	15	15	15	15
120269	20	20	20	20
120270	20	10	5	.
120271	20	20	20	20
120272	10	10	10	10

Creating Variables with Arrays

```
data percent(drop=i);
   set orion.employee_donations;
   array Contrib{4} qtr1-qtr4;
   array Percent{4};
   Total=sum(of contrib{*});
   do i=1 to 4;
      percent{i}=contrib{i}/total;
   end;
run;
```

The second ARRAY statement creates four numeric variables: **Percent1**, **Percent2**, **Percent3**, and **Percent4**.

Output: Creating Variables with Arrays

```
proc print data=percent noobs;
   var Employee_ID percent1-percent4;
   format percent1-percent4 percent6.;
run;
```

Partial PROC PRINT Output

Employee_ID	Percent1	Percent2	Percent3	Percent4
120265	.	.	.	100%
120267	25%	25%	25%	25%
120269	25%	25%	25%	25%
120270	57%	29%	14%	.
120271	25%	25%	25%	25%
120272	25%	25%	25%	25%
120275	25%	25%	25%	25%
120660	25%	25%	25%	25%
120662	50%	.	25%	25%
120663	.	.	100%	.
120668	25%	25%	25%	25%

Business Scenario

Using **orion.employee_donations** as input, calculate the difference in each employee's contribution from one quarter to the next.

Partial Listing of **orion.employee_donations**

Employee_ID	Qtr1	Qtr2	Qtr3	Qtr4
120265	.	.	.	25
120267	15	15	15	15
120269	20	20	20	20
120270	20	10	5	.
120271	20	20	20	20
120272	10	10	10	10

First difference: Qtr2 – Qtr1
Second difference: Qtr3 – Qtr2
Third difference: Qtr4 – Qtr3

Quiz

How many ARRAY statements would you use to calculate the difference in each employee's contribution from one quarter to the next?

Partial Listing of **orion.employee_donations**

Employee_ID	Qtr1	Qtr2	Qtr3	Qtr4
120265	.	.	.	25
120267	15	15	15	15
120269	20	20		

First difference: Qtr2 – Qtr1
Second difference: Qtr3 – Qtr2
Third difference: Qtr4 – Qtr3

Quiz – Correct Answer

How many ARRAY statements would you use to calculate the difference in each employee's contribution from one quarter to the next? **Answers can vary, but one solution is to use two arrays.**

Partial Listing of **`orion.employee_donations`**

Employee_ID	Qtr1	Qtr2	Qtr3	Qtr4
120265	.	.	.	25
120267	15	15	15	15
120269	20	20		

First difference: Qtr2 – Qtr1
Second difference: Qtr3 – Qtr2
Third difference: Qtr4 – Qtr3

Use one array to refer to the existing variables and a second array to create the three **`Difference`** variables.

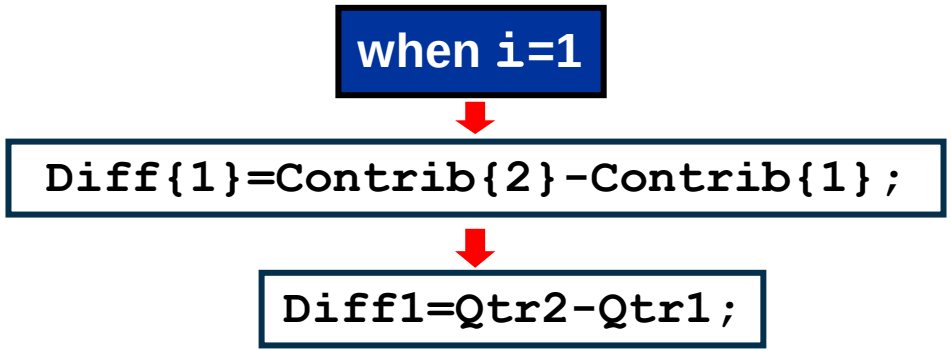
Creating Variables with Arrays

```
data change;
   set orion.employee_donations;
   drop i;
   array Contrib{4} Qtr1-Qtr4;
   array Diff{3};
   do i=1 to 3;
      Diff{i}=Contrib{i+1}-Contrib{i};
   end;
run;
```

The **Contrib** array refers to existing variables. The **Diff** array creates three variables: **Diff1**, **Diff2**, and **Diff3**.

Creating Variables with Arrays

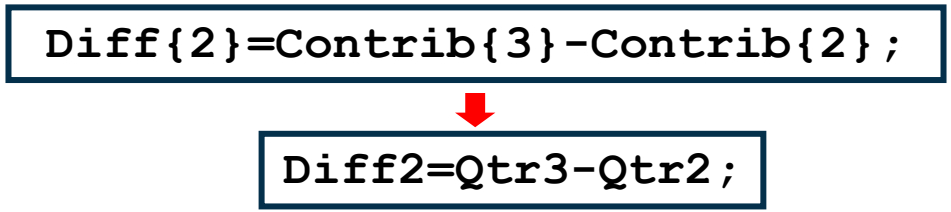
```
data change;
   set orion.employee_donations;
   drop i;
   array Contrib{4} Qtr1-Qtr4;
   array Diff{3};
   do i=1 to 3;
      Diff{i}=Contrib{i+1}-Contrib{i};
   end;
run;
```



Creating Variables with Arrays

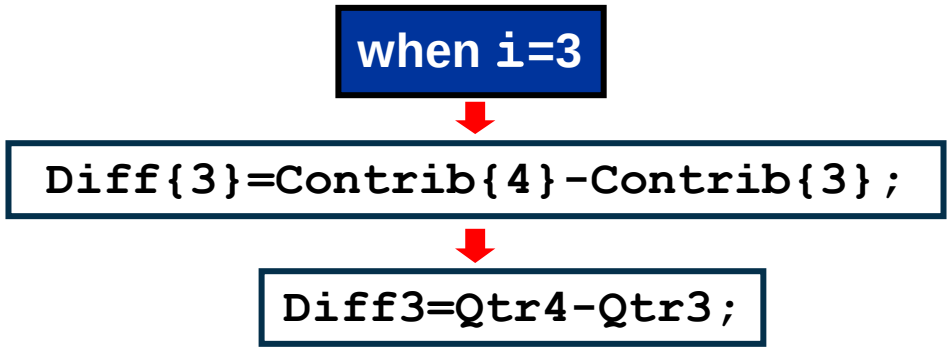
```
data change;
   set orion.employee_donations;
   drop i;
   array Contrib{4} Qtr1-Qtr4;
   array Diff{3};
   do i=1 to 3;
      Diff{i}=Contrib{i+1}-Contrib{i};
   end;
run;
```

when i=2



Creating Variables with Arrays

```
data change;
   set orion.employee_donations;
   drop i;
   array Contrib{4} Qtr1-Qtr4;
   array Diff{3};
   do i=1 to 3;
      Diff{i}=Contrib{i+1}-Contrib{i};
   end;
run;
```



Creating Variables with Arrays

```
proc print data=change noobs;
   var Employee_ID Diff1-Diff3;
run;
```

Partial PROC PRINT Output

Employee_ID	Diff1	Diff2	Diff3
120265	.	.	.
120267	0	0	0
120269	0	0	0
120270	-10	-5	.
120271	0	0	0
120272	0	0	0
120275	0	0	0
120660	0	0	0
120662	.	.	0

Restructuring Tables

Restructuring Data with the DATA Step

Restructuring Tables

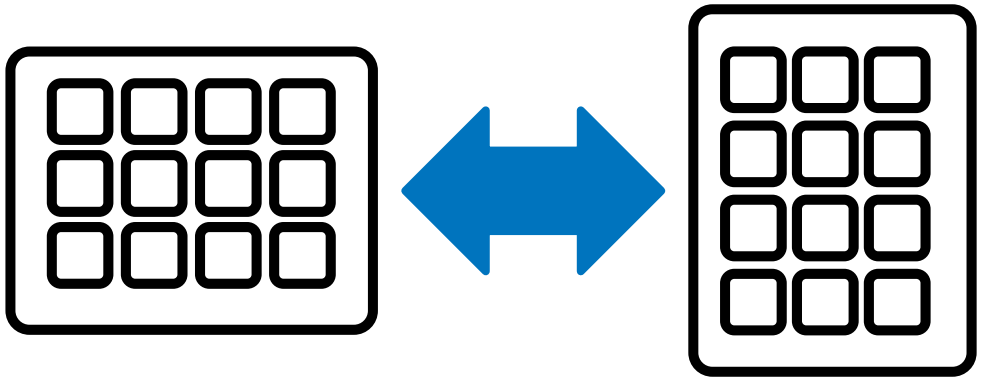


Table Structure

class_test_wide

	Name	Math	Reading
1	Alfred	82	79
2	Alice	71	67
3	Barbara	96	86

class_test_narrow

	Name	TestSubject	TestScore
1	Alfred	Math	82
2	Alfred	Reading	79
3	Alice	Math	71
4	Alice	Reading	67
5	Barbara	Math	96
6	Barbara	Reading	86

Both tables include the same information, but they are structured differently.

Multiple Choice Question

Which table and column (or columns) could you use with PROC MEANS to calculate an average for all test scores combined?

a. class_test_wide, Math and Reading

b. class_test_narrow, TestScore

```
proc means data=???;
    var ???;
run;
```

	Name	Math	Reading
1	Alfred	82	79
2	Alice	71	67
3	Barbara	96	86

class_test_wide

	Name	TestSubject	TestScore
1	Alfred	Math	82
2	Alfred	Reading	79
3	Alice	Math	71
4	Alice	Reading	67
5	Barbara	Math	96
6	Barb		86

class_test_narrow

Multiple Choice Question – Correct Answer

Which table and column (or columns) could you use with PROC MEANS to calculate an average for all test scores combined?

a. class_test_wide, Math and Reading

b. class_test_narrow, TestScore

```
proc means data=pg2.class_test_narrow
           maxdec=1;
    var TestScore;
run;
```

The MEANS Procedure

Analysis Variable : TestScore

N	Mean	Std Dev	Minimum	Maximum
38	77.7	11.3	55.0	99.0

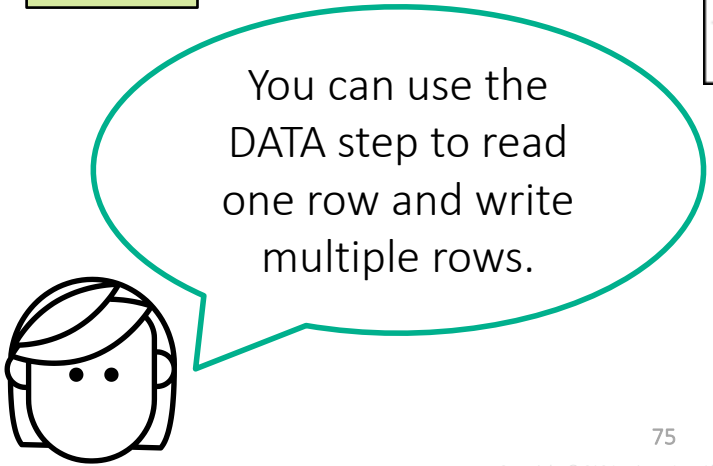
Restructuring Data with the DATA Step

	Name	Math	Reading
1	Alfred	82	79
2	Alice	71	67
3	Barbara	96	86

wide

	Name	TestSubject	TestScore
1	Alfred	Math	82
2	Alfred	Reading	79
3	Alice	Math	71
4	Alice	Reading	67
5	Barbara	Math	96
6	Barbara	Reading	86

narrow



Creating a Narrow Table with the DATA Step

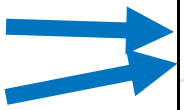
```
data class_test_narrow(keep=Name Subject Score);
       set pg2.class_test_wide;
       length Subject $ 7;
       Subject="Math";
       Score=Math;
       output;
       Subject="Reading";
       Score=Reading;
       output;
run;
```

How could this be more efficient or programmer friendly?

Restructuring Data with the DATA Step

	Name	TestSubject	TestScore
1	Alfred	Math	82
2	Alfred	Reading	79
3	Alice	Math	71
4	Alice	Reading	67
5	Barbara	Math	96
6	Barbara	Reading	86

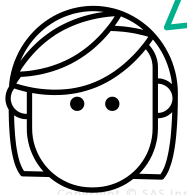
narrow



	Name	Math	Reading
1	Alfred	82	79
2	Alice	71	67
3	Barbara	96	86

wide

You can use the DATA step to read multiple rows before writing one row to the output table.



Restructuring Data with the DATA Step

	Name	TestSubject	TestScore
1	Alfred	Math	82
2	Alfred	Reading	79
3	Alice	Math	71
4	Alice	Reading	67
5	Barbara	Math	96
6	Barbara	Reading	86

narrow

	Name	Math	Reading
1	Alfred	82	79
2	Alice	71	67
3	Barbara	96	86

wide

```
if TestSubject="Math" then Math=TestScore;
else if TestSubject="Reading" then Reading=TestScore;
```

Activity

1. Examine the last DATA step code in **19-loops.sas** and run the program. What statement is necessary to carry the data from the first iteration over to the second?
2. Add a statement to include only the last row per student in the output table. Run the program.
3. What must be true of the input table for the DATA step to work?

continued...

Activity – Correct Answer

1. Examine the DATA step code and run the program. Add the RETAIN statement and run the program again. Why is the RETAIN statement necessary?

The RETAIN statement hold values in the PDV across multiple iterations of the DATA step. The last row for each student includes both test scores.

Name	Math	Reading
Alfred	82	.
Alfred	.	79
Alice	71	.
Alice	.	67

without RETAIN

Name	Math	Reading
Alfred	82	.
Alfred	82	79
Alice	71	79
Alice	71	67

with RETAIN

Activity – Correct Answer

2. Add a subsetting IF statement to include only the last row per student in the output table.

```
data class_wide;
     set pg2.class_test_narrow;
     by name;
     retain Name Math Reading;
     keep Name Math Reading;
     if TestSubject="Reading" then Reading=TestScore;
     else if TestSubject="Math" then Math=TestScore;
     if last.name=1 then output;
run;
```

3. What must be true of the input table for the DATA step to work?

   The data must be sorted by Name.

Iterative Processing - Lesson Quiz

1. Which output table is produced from the following step?

```
data Earnings(keep=Qtr Earned);
     Amount=1000; Rate=.075/4;
     do Qtr=1 to 4;
        Earned+(Amount+Earned)*Rate;
     end;
run;
```

a.

Qtr	Earned
4	77.135865784

b.

Qtr	Earned
5	77.135865784

c.

Qtr	Earned
1	18.75
2	37.8515625
3	57.311279297
4	77.135865784

d.

Qtr	Earned
1	18.75
2	37.8515625
3	57.311279297
4	77.135865784
5	77.135865784

1. Which output table is produced from the following step?

```
data Earnings(keep=Qtr Earned);
     Amount=1000; Rate=.075/4;
     do Qtr=1 to 4;
        Earned+(Amount+Earned)*Rate;
     end;
run;
```

a.

Qtr	Earned
4	77.135865784

b.

Qtr	Earned
5	77.135865784

c.

Qtr	Earned
1	18.75
2	37.8515625
3	57.311279297
4	77.135865784

d.

Qtr	Earned
1	18.75
2	37.8515625
3	57.311279297
4	77.135865784
5	77.135865784

2. Which statement is true regarding the iterative DO loop?

DO *index-column* **=** *start* **TO** *stop* <**BY** *increment*> **;**

a. The start and stop values can be character or numeric values.
b. If an increment value is not specified, the default increment is 0.
c. The index column is incremented at the bottom of each DO loop.
d. The index column is not in the final table unless specifically kept.

2. Which statement is true regarding the iterative DO loop?

DO *index-column* **=** *start* **TO** *stop* <**BY** *increment*> **;**

a. The start and stop values can be character or numeric values.
b. If an increment value is not specified, the default increment is 0.
c. The index column is incremented at the bottom of each DO loop.
d. The index column is not in the final table unless specifically kept.

3. How many rows are in the output table given the following?

pg2.savings

Name	Amount
James	250
Linda	300
Mary	275
Robert	350

```
data work.savings;
    set pg2.savings;
    Savings=0;
    do Year=1 to 5;
       do qtr=1 to 4;
          Savings+Amount;
          Savings+(Savings*0.02/12);
       end;
    end;
run;
```

a. 1

b. 4

c. 5

d. 20

3. How many rows are in the output table given the following?

pg2.savings

Name	Amount
James	250
Linda	300
Mary	275
Robert	350

```
data work.savings;
    set pg2.savings;
    Savings=0;
    do Year=1 to 5;
       do qtr=1 to 4;
          Savings+Amount;
          Savings+(Savings*0.02/12);
       end;
    end;
run;
```

a. 1

b. 4

c. 5

d. 20

4. What is the final value of **Year** given the following step?

```
data invest;
    do Year=2010 to 2019;
       Capital+5000;
       Capital+(Capital*.03);
    end;
run;
```

a. . (missing)

b. 2010

c. 2019

d. 2020

4. What is the final value of **Year** given the following step?

```
data invest;
    do Year=2010 to 2019;
       Capital+5000;
       Capital+(Capital*.03);
    end;
run;
```

a. . (missing)

b. 2010

c. 2019

d. 2020

5. Which of the following statements contains valid syntax?

a. `do 1 to 10 by 2;`

b. `do while (Year>2025);`

c. `do until Earnings<=100000;`

d. `do date='01JAN2019' to '31JAN2019';`

5. Which of the following statements contains valid syntax?

a. `do 1 to 10 by 2;`

b. `do while (Year>2025);`

c. `do until Earnings<=100000;`

d. `do date='01JAN2019' to '31JAN2019';`

6. How many rows are in the output table given the following?

work.bikeinfo

name	bike
Marco	12
Angela	10

```
data bikeinfo2;
     set bikeinfo;
     do month=1 to 3;
        do week=1 to 4;
           bike=bike+2;
        end;
     output;
     end;
run;
```

a. 2

b. 3

c. 6

d. 12

e. 24

6. How many rows are in the output table given the following?

work.bikeinfo

name	bike
Marco	12
Angela	10

```
data bikeinfo2;
     set bikeinfo;
     do month=1 to 3;
        do week=1 to 4;
           bike=bike+2;
        end;
     output;
     end;
run;
```

a. 2

b. 3

c. 6

d. 12

e. 24

7. What is the value of **x** at the completion of the DATA step?

```
data test;
     x=15;
     do until(x>12);
        x+1;
     end;
run;
```

a. . (missing)

b. 13

c. 15

d. 16

7. What is the value of **x** at the completion of the DATA step?

```
data test;
     x=15;
     do until(x>12);
        x+1;
     end;
run;
```

a. . (missing)

b. 13

c. 15

d. 16

8. Which statement is false?

a. The DO UNTIL loop executes until a condition is true.
b. The DO WHILE loop always executes at least one time.
c. The DO WHILE loop checks the condition at the top of the loop.
d. The DO UNTIL loop checks the condition at the bottom of the loop.

8. Which statement is false?

a. The DO UNTIL loop executes until a condition is true.
b. The DO WHILE loop always executes at least one time.
c. The DO WHILE loop checks the condition at the top of the loop.
d. The DO UNTIL loop checks the condition at the bottom of the loop.

9. Which of the following statements contains valid syntax?

a. `do Age=10 to 14 and while (Weight<150);`
b. `do week=1 to 52 do until (Mileage ge 2750);`
c. `do Increase=5 to 10 while (temperature lt 102);`
d. `do Year=2018 to 2028 or until (Earnings<=100000);`

9. Which of the following statements contains valid syntax?

a. `do Age=10 to 14 and while (Weight<150);`

b. `do week=1 to 52 do until (Mileage ge 2750);`

c. `do Increase=5 to 10 while (temperature lt 102);`

d. `do Year=2018 to 2028 or until (Earnings<=100000);`

10. Which output table is produced from the following step?

```
data test;
     bike=10;
     do day=1 to 7 while (bike lt 13);
        bike=bike+2;
     end;
run;
```

a.

bike	day
14	2

b.

bike	day
14	3

c.

bike	day
24	7

d.

bike	day
24	8

10. Which output table is produced from the following step?

```
data test;
     bike=10;
     do day=1 to 7 while (bike lt 13);
        bike=bike+2;
     end;
run;
```

a.

bike	day
14	2

b.

bike	day
14	3

c.

bike	day
24	7

d.

bike	day
24	8

Restructuring Tables - Lesson Quiz

1. Which is the better description for the following table?

Year	Jan	Feb	Mar	Apr	May	Jun
Yr1956	284	277	317	313	318	374
Yr1957	315	301	356	348	355	422
Yr1958	340	318	362	348	363	435

a. wide table

b. narrow table

1. Which is the better description for the following table?

Year	Jan	Feb	Mar	Apr	May	Jun
Yr1956	284	277	317	313	318	374
Yr1957	315	301	356	348	355	422
Yr1958	340	318	362	348	363	435

a. wide table

b. narrow table

2. Which statement is needed for creating multiple rows from a single row when using the DATA step to go from a wide to a narrow table?

a. WIDE
b. NARROW
c. RETAIN
d. OUTPUT

2. Which statement is needed for creating multiple rows from a single row when using the DATA step to go from a wide to a narrow table?

a. WIDE
b. NARROW
c. RETAIN
d. OUTPUT

3. How many rows will be in the final table if **work.airwide** contains three rows?

a. 3

b. 6

c. 9

d. 12

```
data work.airnarrow;
     set work.airwide;
     Month='Jan';
     Air=Jan;
     output;
     Month='Feb';
     Air=Feb;
     output;
     Month='Mar';
     Air=Mar;
     output;
     keep Year Month Air;
run;
```

3. How many rows will be in the final table if **work.airwide** contains three rows?

```
data work.airnarrow;
     set work.airwide;
     Month='Jan';
     Air=Jan;
     output;
     Month='Feb';
     Air=Feb;
     output;
     Month='Mar';
     Air=Mar;
     output;
     keep Year Month Air;
run;
```

a. 3

b. 6

c. 9

d. 12

4. When using the DATA step to go from a narrow table to a wide table, the KEEP statement is needed to hold values in the PDV across multiple iterations of the DATA step.

a. True
b. False

4. When using the DATA step to go from a narrow table to a wide table, the KEEP statement is needed to hold values in the PDV across multiple iterations of the DATA step.

a. True

b. False

5. Which statement needs to be added to the DATA step to include only the last row per **Year** in the output table?

```
data work.airwide2(keep=Year Jan Feb Mar);
    set work.airnarrow;
    by Year;
    retain Jan Feb Mar;
    if Month='Jan' then Jan=Air;
    else if Month='Feb' then Feb=Air;
    else if Month='Mar' then Mar=Air;
    ... insert statement here ...
run;
```

a. `output;`

b. `if Last then output;`

c. `if Last.Year=1 then output;`

d. `if Last.Year=0 then output;`

5. Which statement needs to be added to the DATA step to include only the last row per **Year** in the output table?

```
data work.airwide2(keep=Year Jan Feb Mar);
    set work.airnarrow;
    by Year;
    retain Jan Feb Mar;
    if Month='Jan' then Jan=Air;
    else if Month='Feb' then Feb=Air;
    else if Month='Mar' then Mar=Air;
    ... insert statement here ...
run;
```

a. `output;`

b. `if Last then output;`

c. `if Last.Year=1 then output;`

d. `if Last.Year=0 then output;`