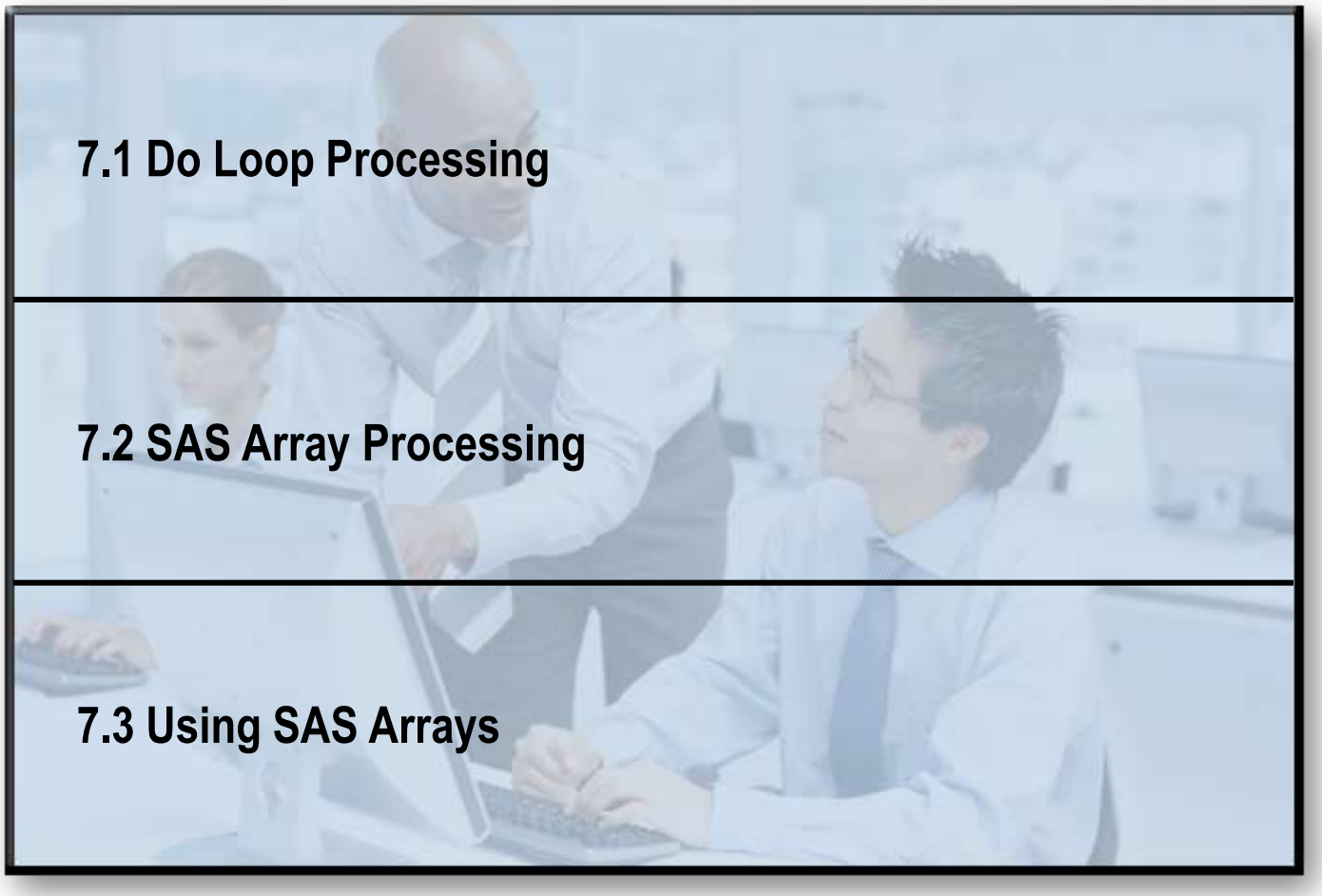
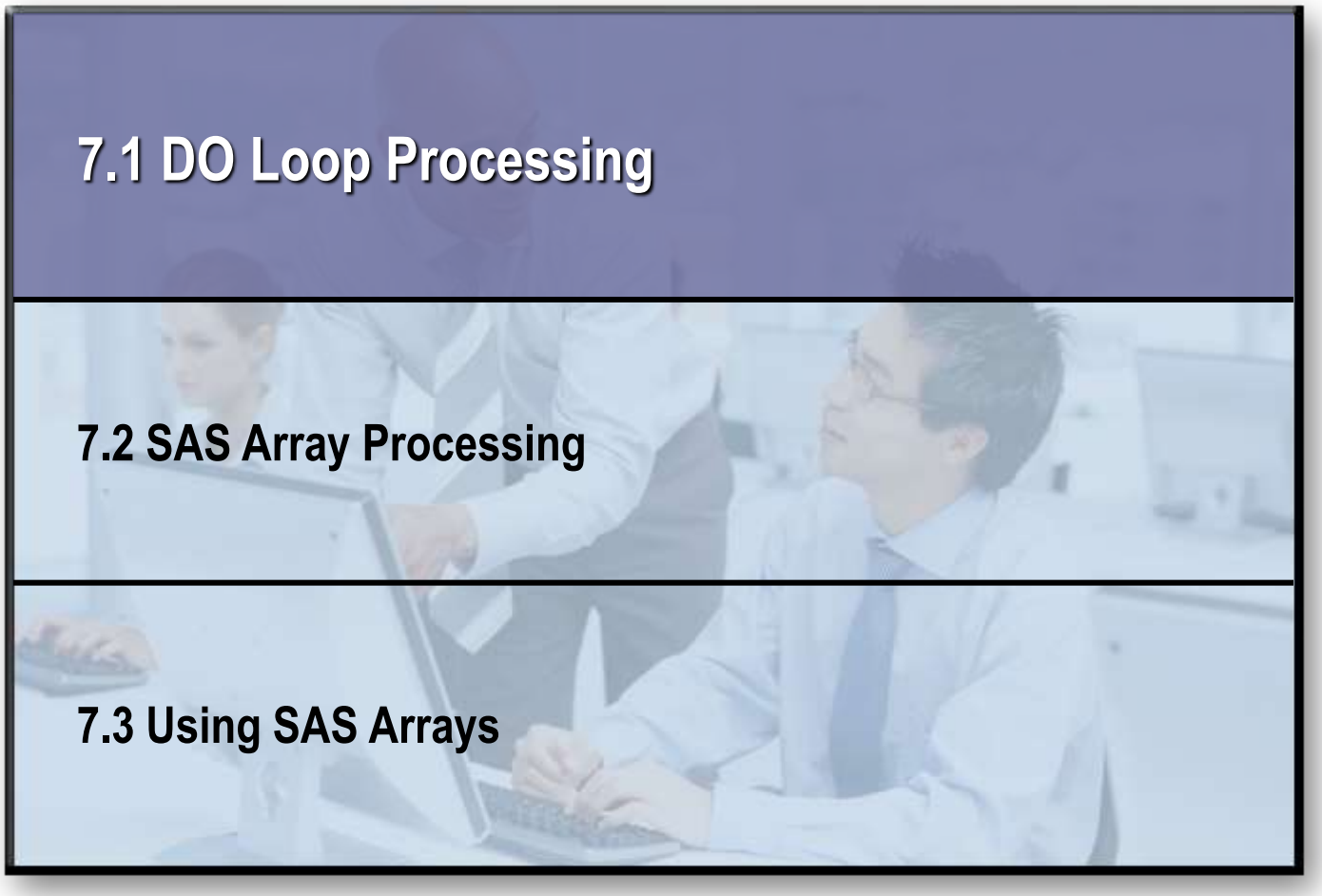
STAT604

Lesson SAS 09

Chapter 7: Processing Data Iteratively



Chapter 7: Processing Data Iteratively



Objectives

- Explain iterative DO loops.
- Use DO loops to eliminate redundant code and repetitive calculations.
- Use conditional DO loops.
- Use nested DO loops.

Business Scenario

An Orion Star employee wants to compare the interest for yearly versus quarterly compounding on a $50,000 investment made for one year at 4.5 percent interest.

How much money will the employee accrue in each situation?

Repetitive Coding

```
data compound;
   Amount=50000;
   Rate=.045;
   Yearly=Amount*Rate;
   Quarterly+((Quarterly+Amount)*Rate/4);
   Quarterly+((Quarterly+Amount)*Rate/4);
   Quarterly+((Quarterly+Amount)*Rate/4);
   Quarterly+((Quarterly+Amount)*Rate/4);
run;
proc print data=compound noobs;
run;
```

PROC PRINT Output

Amount	Rate	Yearly	Quarterly
50000	0.045	2250	2288.25

Repetitive Coding

What if the employee wants to determine annual and quarterly compounded interest for a period of 20 years (80 quarters)?

```
data compound;
   Amount=50000;
   Rate=.045;
   Yearly +(Yearly+Amount)*Rate;
   .
   .
   .
   Yearly +(Yearly+Amount)*Rate;
   Quarterly+((Quarterly+Amount)*Rate/4);
   .
   .
   .
   Quarterly+((Quarterly+Amount)*Rate/4);
run;
```

20x

80x

DO Loop Processing

Use DO loops to perform the repetitive calculations.

```
data compound(drop=i);
   Amount=50000;
   Rate=.045;
   do i=1 to 20;
      Yearly +(Yearly+Amount)*Rate;
   end;
   do i=1 to 80;
      Quarterly+((Quarterly+Amount)*Rate/4);
   end;
run;
```


p207d02

Various Forms of Iterative DO Loops

There are several forms of iterative DO loops that execute the statements between the DO and END statements repetitively.

```
DO index-variable=start TO stop <BY increment>;
    iterated SAS statements…
END;
```

```
DO index-variable=item-1 <,…item-n>;
    iterated SAS statements…
END;
```

The Iterative DO Statement

General form of an iterative DO statement:

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

The values of *start*, *stop*, and *increment*

- must be numbers or expressions that yield numbers
- are established before executing the loop
- if omitted, *increment* defaults to 1.

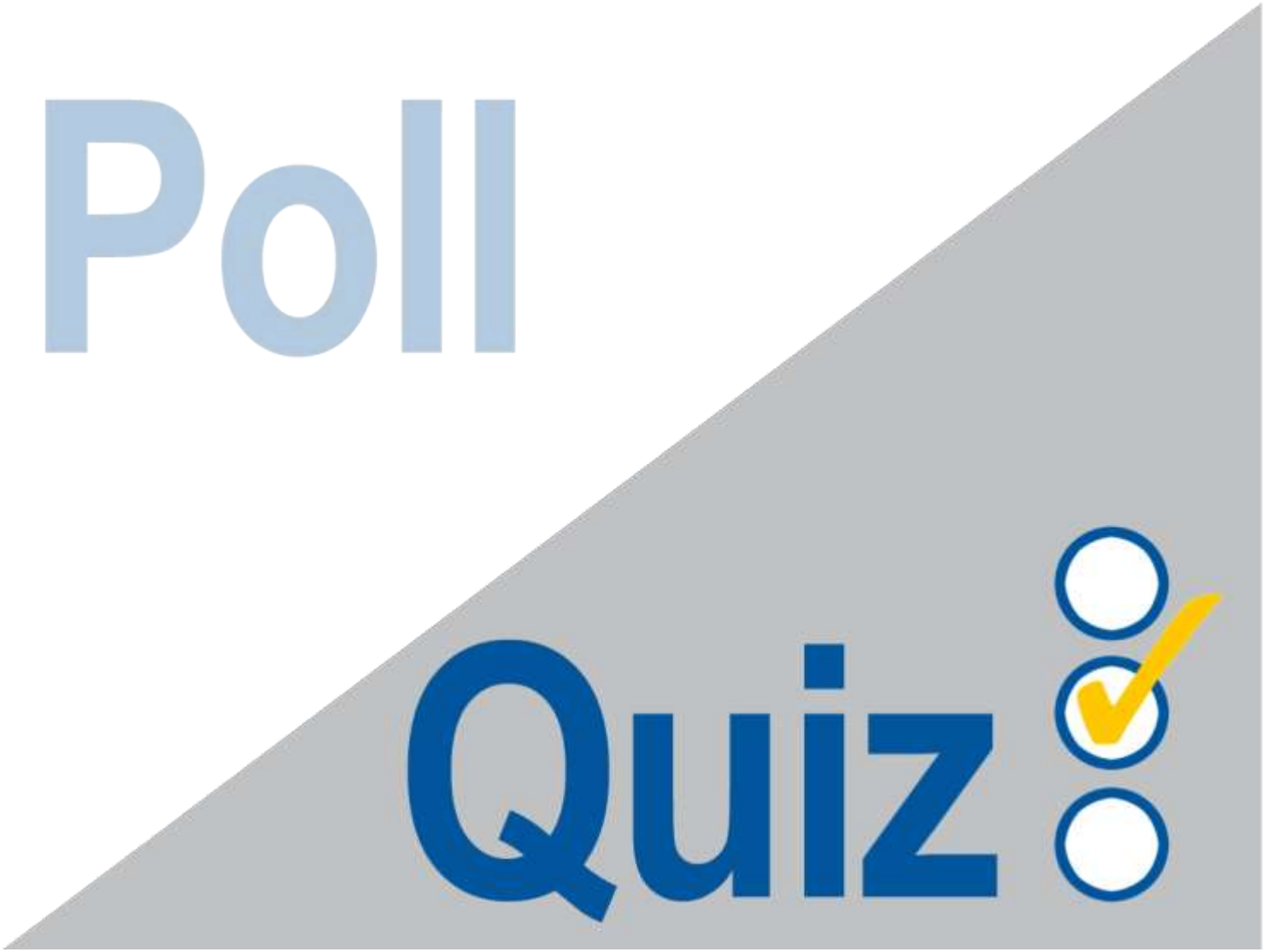
The Iterative DO Statement

Index-variable details:

- The *index-variable* is written to the output data set by default.
- At the termination of the loop, the value of *index-variable* is one *increment* beyond the *stop* value.

Modifying the value of *index-variable* affects the number of iterations, and might cause infinite looping or early loop termination.

Poll



7.01 Quiz

What are the final values of the index variables after the following DO loops execute?

```
do i=1 to 5;
   …
end;                          1 2 3 4 5 6

do j=2 to 8 by 2;
   …
end;

do k=10 to 2 by -2;
   …
end;
```

The final values are highlighted.

7.01 Quiz – Correct Answer

What are the final values of the index variables after the following DO statements execute?

The final values are highlighted.

```
do i=1 to 5;                          1 2 3 4 5 6
   …
end;

do j=2 to 8 by 2;                     2 4 6 8 10
   …
end;

do k=10 to 2 by -2;                   10 8 6 4 2 0
   …
end;
```

The Iterative DO Statement

General form of an iterative DO statement with an *item-list*:

DO *index-variable=item-1* <,*…item-n*>;

- The DO loop is executed once for each item in the list.
- The list must be comma separated.

Sample DO Loops with Item Lists

Items in the list can be all numeric or all character constants, or they can be variables.

```
do Month='JAN','FEB','MAR';
   …
end;
```

Character constants

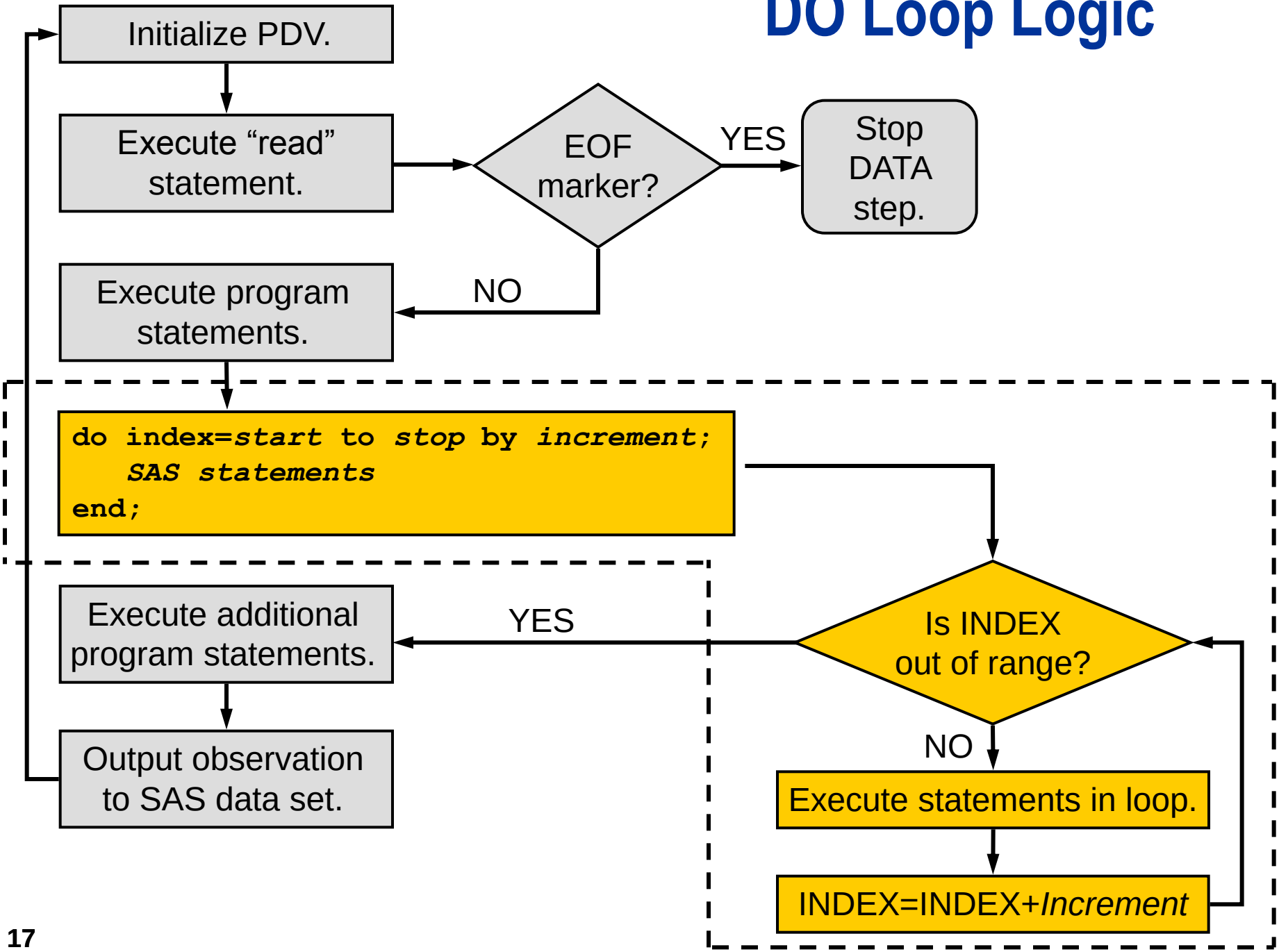
```
do odd=1,3,5,7,9;
   …
end;
```

Numeric constants

```
do i=Var1,Var2,Var3;
   …
end;
```

Variables

DO Loop Logic



Business Scenario

On January 1 of each year, an Orion Star employee invests $5,000 in an account. Determine the value of the account after three years based on a constant annual interest rate of 4.5 percent, starting in 2008.

```
data invest;
   do Year=2008 to 2010;
      Capital+5000;
      Capital+(Capital*.045);
   end;
run;
```


p207d03

Execution: Performing Repetitive Calculations

```
data invest;
   do Year=2008 to 2010;
      Capital+5000;
      Capital+(Capital*.045);
   end;
run;
```

Initialize PDV

PDV

Year	R Capital	D _N_
.	0	1

- **Capital** is used in a sum statement, so it is automatically initialized to zero and retained.

...

Execution: Performing Repetitive Calculations

```
data invest;
   do Year=2008 to 2010;
      Capital+5000;
      Capital+(Capital*.045);
   end;
run;
```

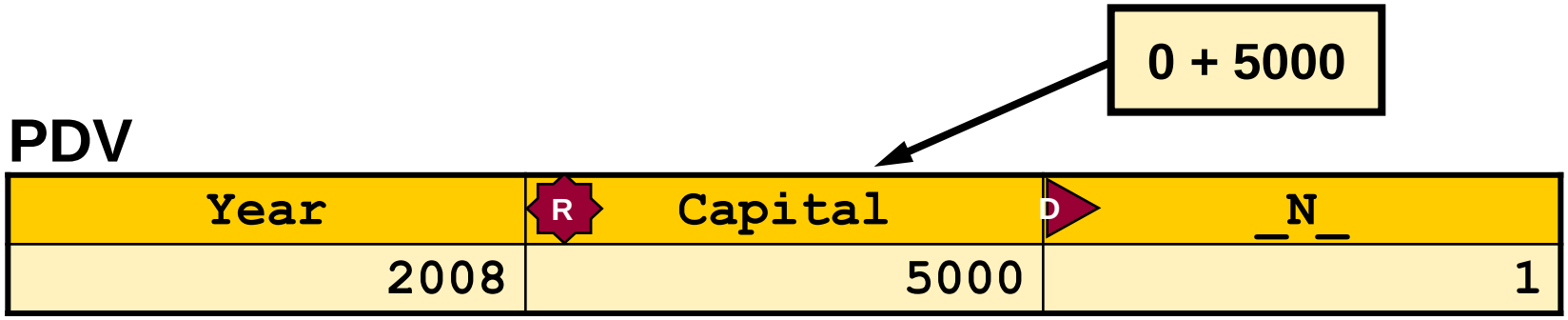
PDV

Year	R	Capital	D	_N_
2008		0		1

...

Execution: Performing Repetitive Calculations

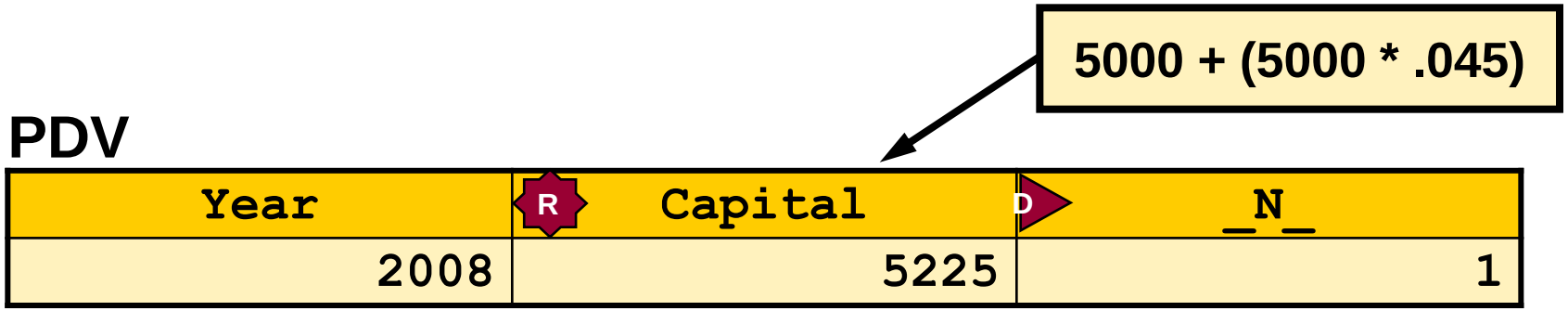
```
data invest;
   do Year=2008 to 2010;
      Capital+5000;
      Capital+(Capital*.045);
   end;
run;
```



...

Execution: Performing Repetitive Calculations

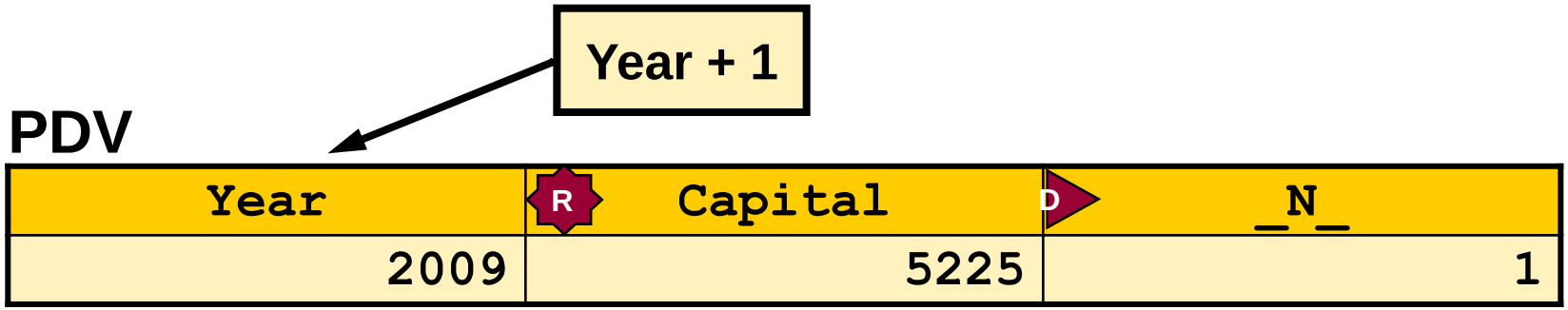
```
data invest;
   do Year=2008 to 2010;
      Capital+5000;
      Capital+(Capital*.045);
   end;
run;
```



...

Execution: Performing Repetitive Calculations

```
data invest;
   do Year=2008 to 2010;
      Capital+5000;
      Capital+(Capital*.045);
   end;
run;
```



...

Execution: Performing Repetitive Calculations

```
data invest;
   do Year=2008 to 2010;
      Capital+5000;
      Capital+(Capital*.045);
   end;
run;
```

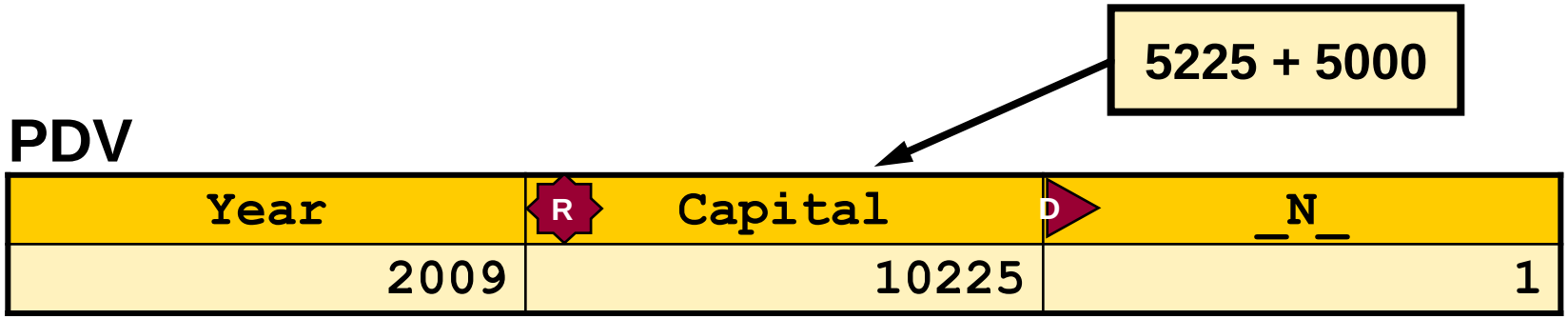
PDV

Year	R	Capital	D	_N_
2009		5225		1

...

Execution: Performing Repetitive Calculations

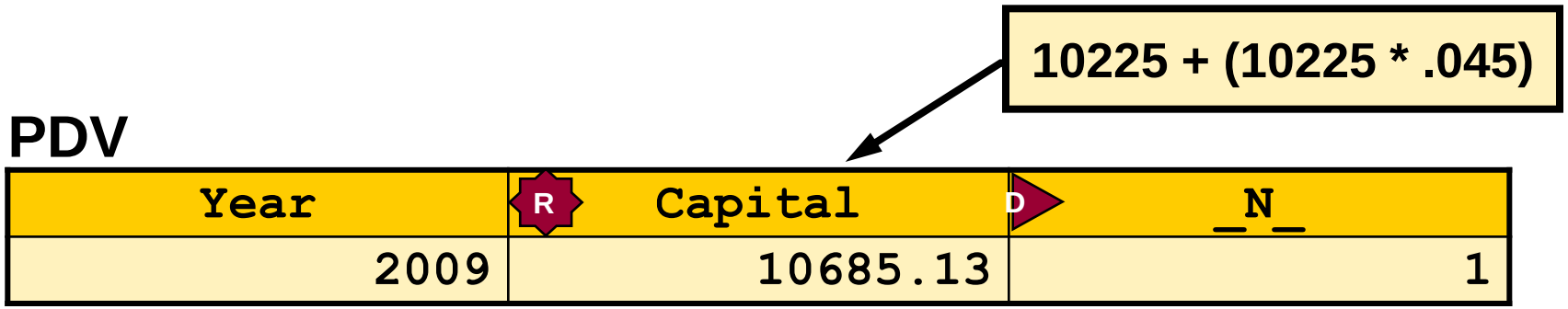
```
data invest;
   do Year=2008 to 2010;
      Capital+5000;
      Capital+(Capital*.045);
   end;
run;
```



...

Execution: Performing Repetitive Calculations

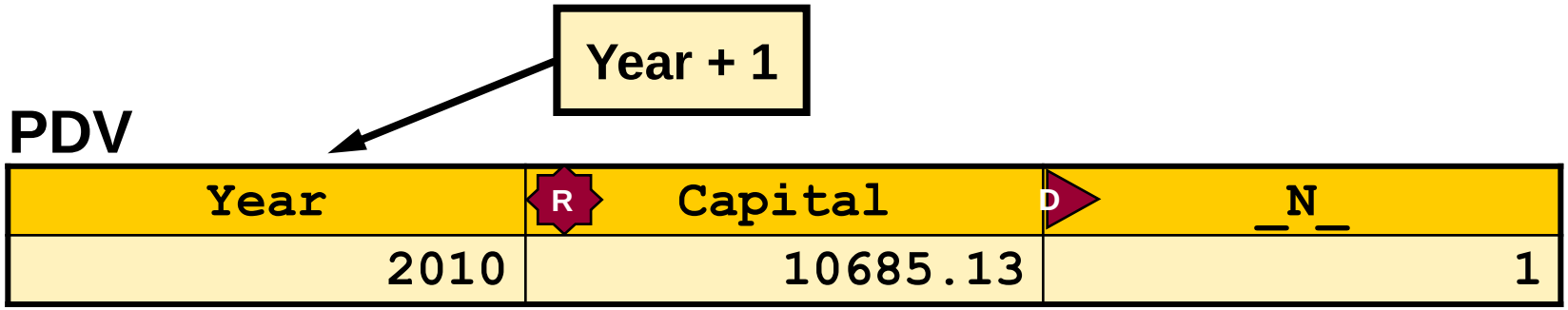
```
data invest;
   do Year=2008 to 2010;
      Capital+5000;
      Capital+(Capital*.045);
   end;
run;
```



...

Execution: Performing Repetitive Calculations

```
data invest;
   do Year=2008 to 2010;
      Capital+5000;
      Capital+(Capital*.045);
   end;
run;
```



...

Execution: Performing Repetitive Calculations

```
data invest;
   do Year=2008 to 2010;
      Capital+5000;
      Capital+(Capital*.045);
   end;
run;
```

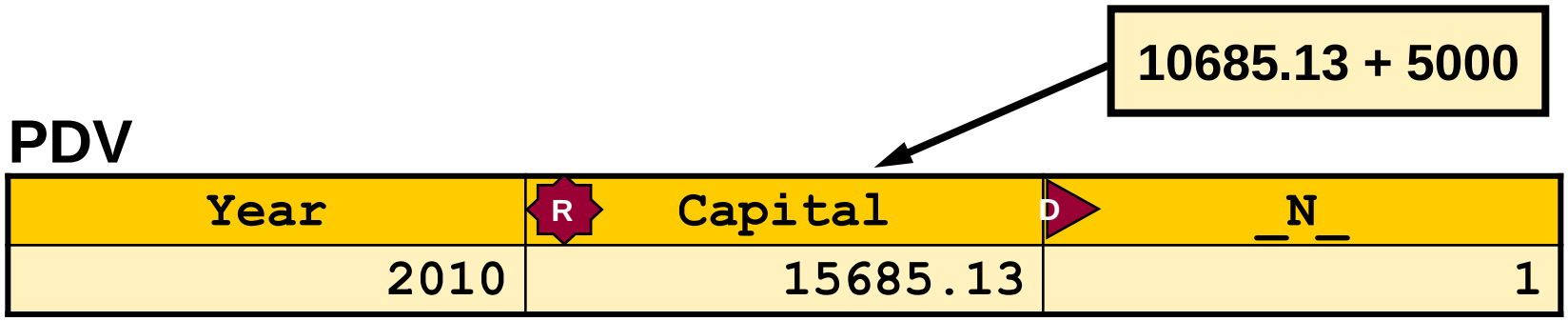
PDV

Year	R	Capital	D	_N_
2010		10685.13		1

...

Execution: Performing Repetitive Calculations

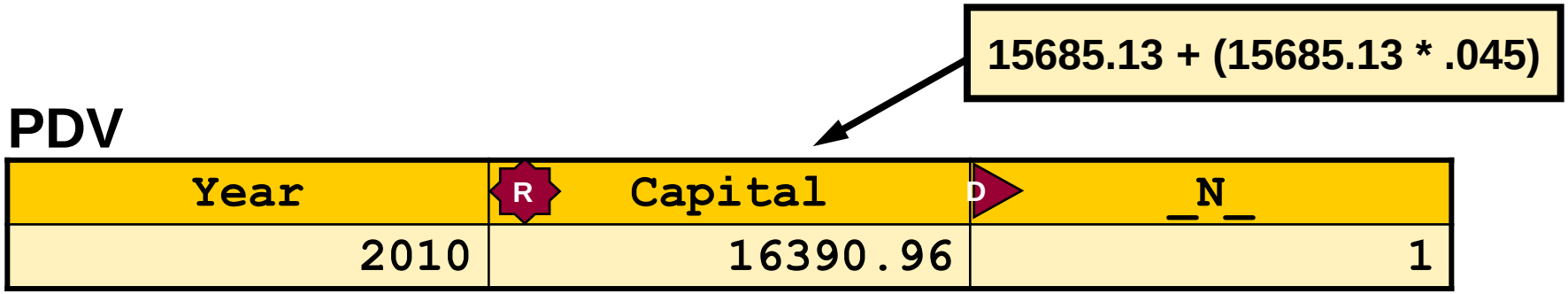
```
data invest;
   do Year=2008 to 2010;
      Capital+5000;
      Capital+(Capital*.045);
   end;
run;
```



...

Execution: Performing Repetitive Calculations

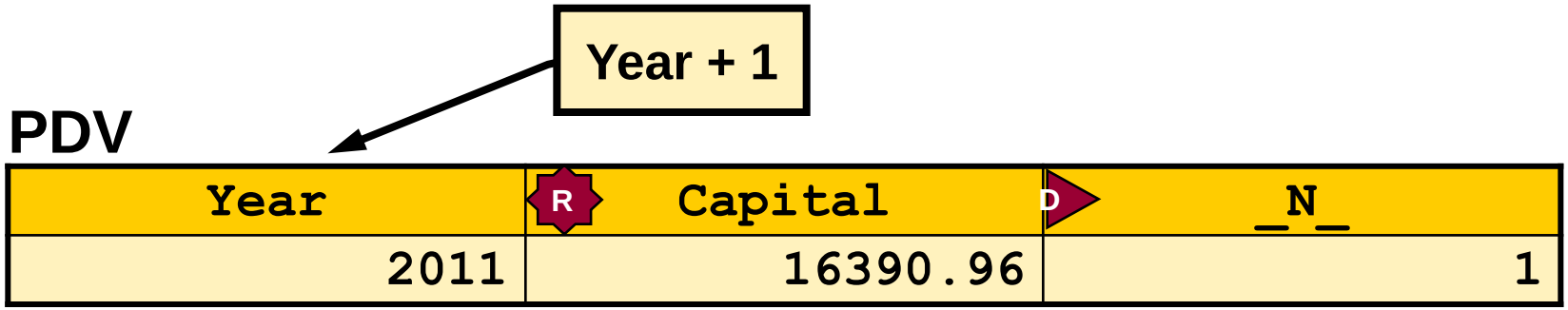
```
data invest;
   do Year=2008 to 2010;
      Capital+5000;
      Capital+(Capital*.045);
   end;
run;
```



...

Execution: Performing Repetitive Calculations

```
data invest;
   do Year=2008 to 2010;
      Capital+5000;
      Capital+(Capital*.045);
   end;
run;
```



...

Execution: Performing Repetitive Calculations

```
data invest;
   do Year=2008 to 2010;
      Capital+5000;
      Capital+(Capital*.045);
   end;
run;
```

PDV

Year	R	Capital	D	_N_
2011		16390.96		1

...

Execution: Performing Repetitive Calculations

```
data invest;
   do Year=2008 to 2010;
      Capital+5000;
      Capital+(Capital*.045);
   end;
run;
```

Implicit OUTPUT;
No Implicit RETURN;

PDV

Year	R Capital	D _N_
2011	16390.96	1

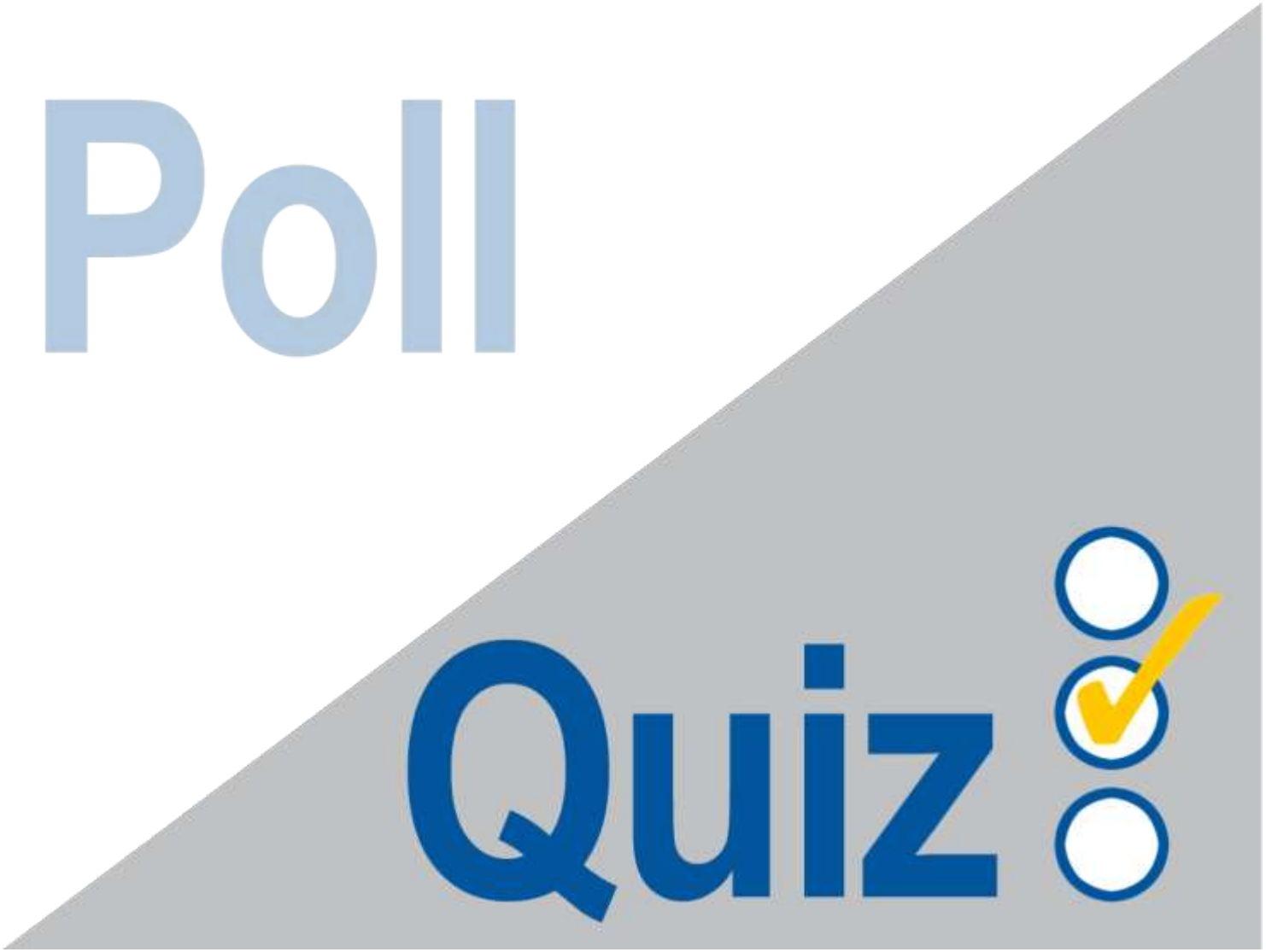
Output: Performing Repetitive Calculations

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

PROC PRINT Output

```
        Year      Capital

        2011     16390.96
```

Poll



7.02 Quiz

How can you generate a separate observation for each year?

```
data invest;
   do Year=2008 to 2010;
      Capital+5000;
      Capital+(Capital*.045);
   end;
run;
proc print data=invest noobs;
run;
```


p207a01

7.02 Quiz – Correct Answer

How can you generate a separate observation for each year? **Place an explicit OUTPUT statement inside the DO loop.**

```
data invest;
   do Year=2008 to 2010;
      Capital+5000;
      Capital+(Capital*.045);
      output;
   end;
run;
proc print data=invest noobs;
run;
```

PROC PRINT Output

Year	Capital
2008	5225.00
2009	10685.13
2010	16390.96

There is no observation for 2011.

p207a01s

Business Scenario

Recall the example that forecasts the growth of several departments at Orion Star. Modify the forecasting application to use a DO loop to eliminate redundant code.

Listing of **`orion.growth`**

Department	Total_ Employees	Increase
Administration	34	0.25
Engineering	9	0.30
IS	25	0.10
Marketing	20	0.20
Sales	201	0.30
Sales Management	11	0.10

A Forecasting Application (Review)

```
data forecast;
   set orion.growth;
   Year=1;
   Total_Employees=Total_Employees*(1+Increase);
   output;
   Year=2;
   Total_Employees=Total_Employees*(1+Increase);
   output;
run;
proc print data=forecast noobs;
run;
```

What if you want to forecast growth over the next six years?

Use a DO Loop to Reduce Redundant Code

```
data forecast;
   set orion.growth;
   do Year=1 to 6;
      Total_Employees=
         Total_Employees*(1+Increase);
      output;
   end;
run;

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

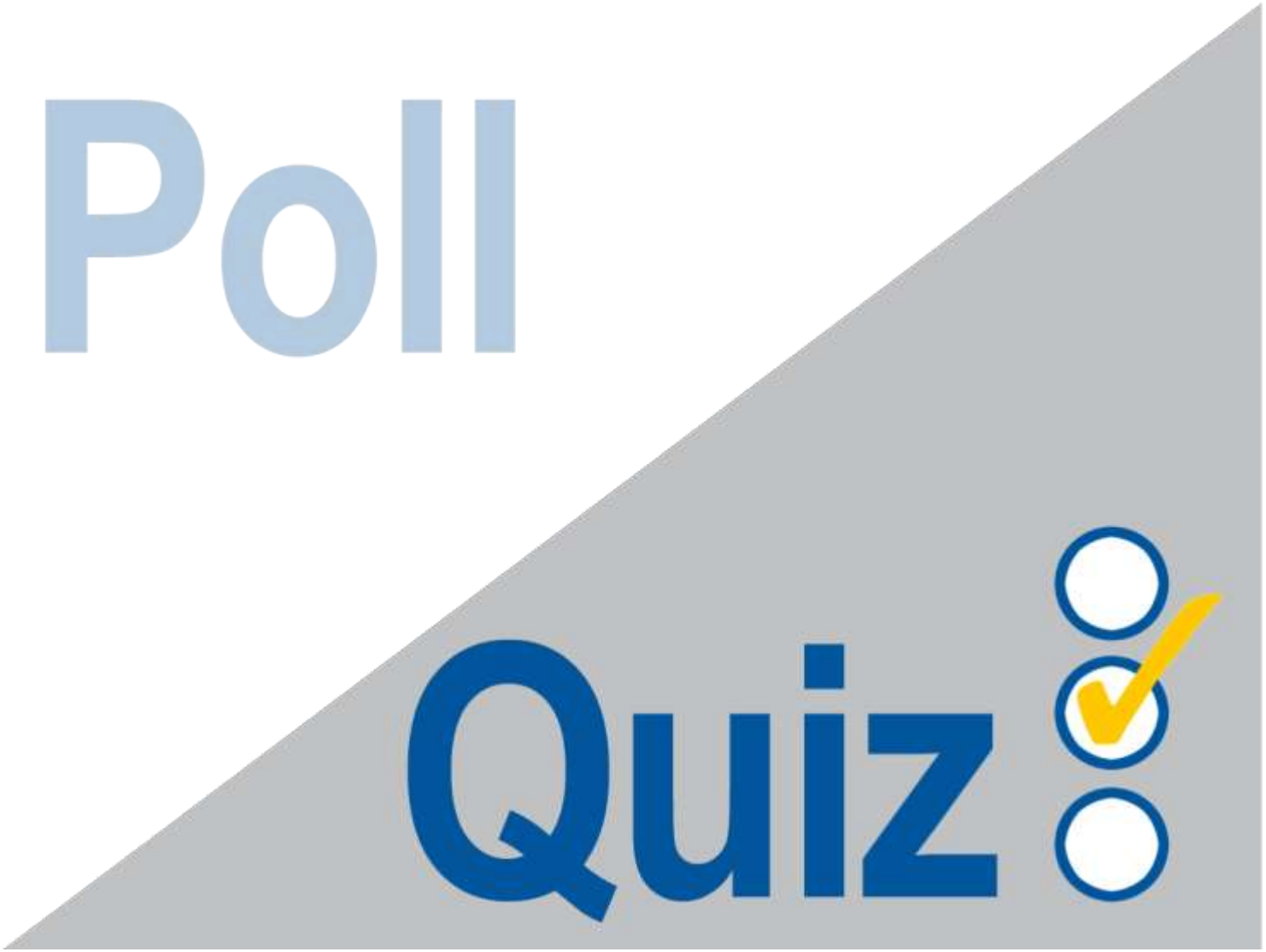

p207d05

Output

Partial PROC PRINT Output

Department	Total_ Employees	Increase	Year
Administration	42.500	0.25	1
Administration	53.125	0.25	2
Administration	66.406	0.25	3
Administration	83.008	0.25	4
Administration	103.760	0.25	5
Administration	129.700	0.25	6
Engineering	11.700	0.30	1
Engineering	15.210	0.30	2
Engineering	19.773	0.30	3
Engineering	25.705	0.30	4
Engineering	33.416	0.30	5
Engineering	43.441	0.30	6
IS	27.500	0.10	1

Poll



7.03 Quiz

What stop value would you use in the DO loop to determine the number of years that it would take for the Engineering department to exceed 75 people?

```
data forecast;
   set orion.growth;
   do Year=1 to 6;
      Total_Employees=
         Total_Employees*(1+Increase);
      output;
   end;
run;
proc print data=forecast noobs;
run;
```

p207d05

7.03 Quiz – Correct Answer

What stop value would you use in the DO loop to determine the number of years it would take for the Engineering department to exceed 75 people?
Unknown.

```
data forecast;
   set orion.growth;
   do Year=1 to 6;
      Total_Employees=
         Total_Employees*(1+Increase);
      output;
   end;
run;
proc print data=forecast noobs;
run;
```

Use *conditional iterative processing* to stop a loop when a condition is met.

Conditional Iterative Processing

You can use DO WHILE and DO UNTIL statements to stop the loop when a condition is met rather than when the loop executed a specific number of times.

To avoid infinite loops, be sure that the specified condition will be met.

The DO WHILE Statement

The DO WHILE statement executes statements in a DO loop repetitively while a condition is true.

General form of the DO WHILE loop:

```
DO WHILE (expression);
      <additional SAS statements>
END;
```

The value of *expression* is evaluated at the **top** of the loop.

The statements in the loop never execute if *expression* is initially false.

The DO UNTIL Statement

The DO UNTIL statement executes statements in a DO loop repetitively until a condition is true.

General form of the DO UNTIL loop:

```
DO UNTIL (expression);
      <additional SAS statements>
END;
```

The value of *expression* is evaluated at the **bottom** of the loop.

The statements in the loop are executed at least once.

Although the condition is placed at the top of the loop, it is evaluated at the bottom of the loop.

Business Scenario

Determine the number of years that it would take for an account to exceed $1,000,000 if $5,000 is invested annually at 4.5 percent.

Using the DO UNTIL Statement

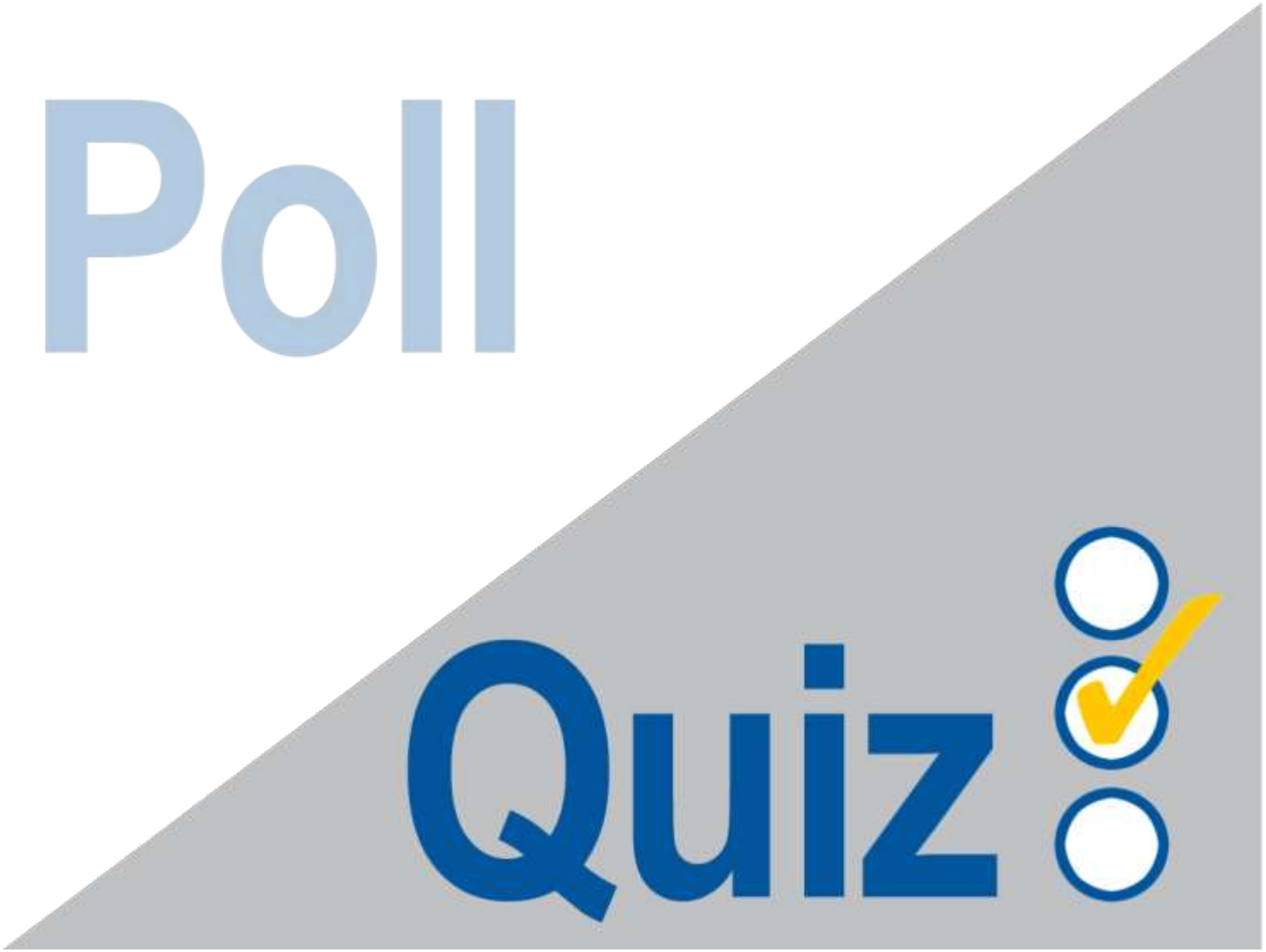
```
data invest;
   do until(Capital>1000000);
      Year+1;
      Capital+5000;
      Capital+(Capital*.045);
   end;
run;

proc print data=invest noobs;
   format Capital dollar14.2;
run;
```

PROC PRINT Output

Capital	Year
$1,029,193.17	52

p207d06

Poll



7.04 Quiz

How can you generate the same result with a DO WHILE statement?

```
data invest;
   do until(Capital>1000000);
      Year+1;
      Capital+5000;
      Capital+(Capital*.045);
   end;
run;

proc print data=invest noobs;
   format capital dollar14.2;
run;
```

p207a02

7.04 Quiz – Correct Answer

How could you generate the same result with a DO WHILE statement? **Change the DO UNTIL statement to a DO WHILE statement and modify the condition.**

```
data invest;
   do while(Capital<=1000000);
      Year+1;
      Capital+5000;
      Capital+(Capital*.045);
   end;
run;

proc print data=invest noobs;
   format capital dollar14.2;
run;
```

p207a02s

Iterative DO Loop with a Conditional Clause

You can combine DO WHILE and DO UNTIL statements with the iterative DO statement.

General form of the iterative DO loop with a conditional clause:

```
DO index-variable=start TO stop <BY increment>
     WHILE | UNTIL (expression);
     <additional SAS statements>
END;
```

- This is one method of avoiding an infinite loop in a DO WHILE or DO UNTIL statements.

Using DO UNTIL with an Iterative DO Loop

Determine the value of the account again. Stop the loop if 30 years is reached or more than $250,000 is accumulated.

```
data invest;
   do Year=1 to 30 until(Capital>250000);
      Capital+5000;
      Capital+(Capital*.045);
   end;
run;
proc print data=invest noobs;
   format capital dollar14.2;
run;
```

PROC PRINT Output

Year	Capital
27	$264,966.67

In a DO UNTIL loop, the condition is checked **before** the index variable is incremented.

p207d07

Using DO WHILE with an Iterative DO Loop

Determine the value of the account again, but this time use a DO WHILE statement.

```
data invest;
   do Year=1 to 30 while(Capital<=250000);
      Capital+5000;
      Capital+(Capital*.045);
   end;
run;
proc print data=invest noobs;
   format capital dollar14.2;
run;
```

In a DO WHILE loop, the condition is checked **after** the index variable is incremented.

PROC PRINT Output

Year	Capital
28	$264,966.67

p207d07

Nested DO Loops

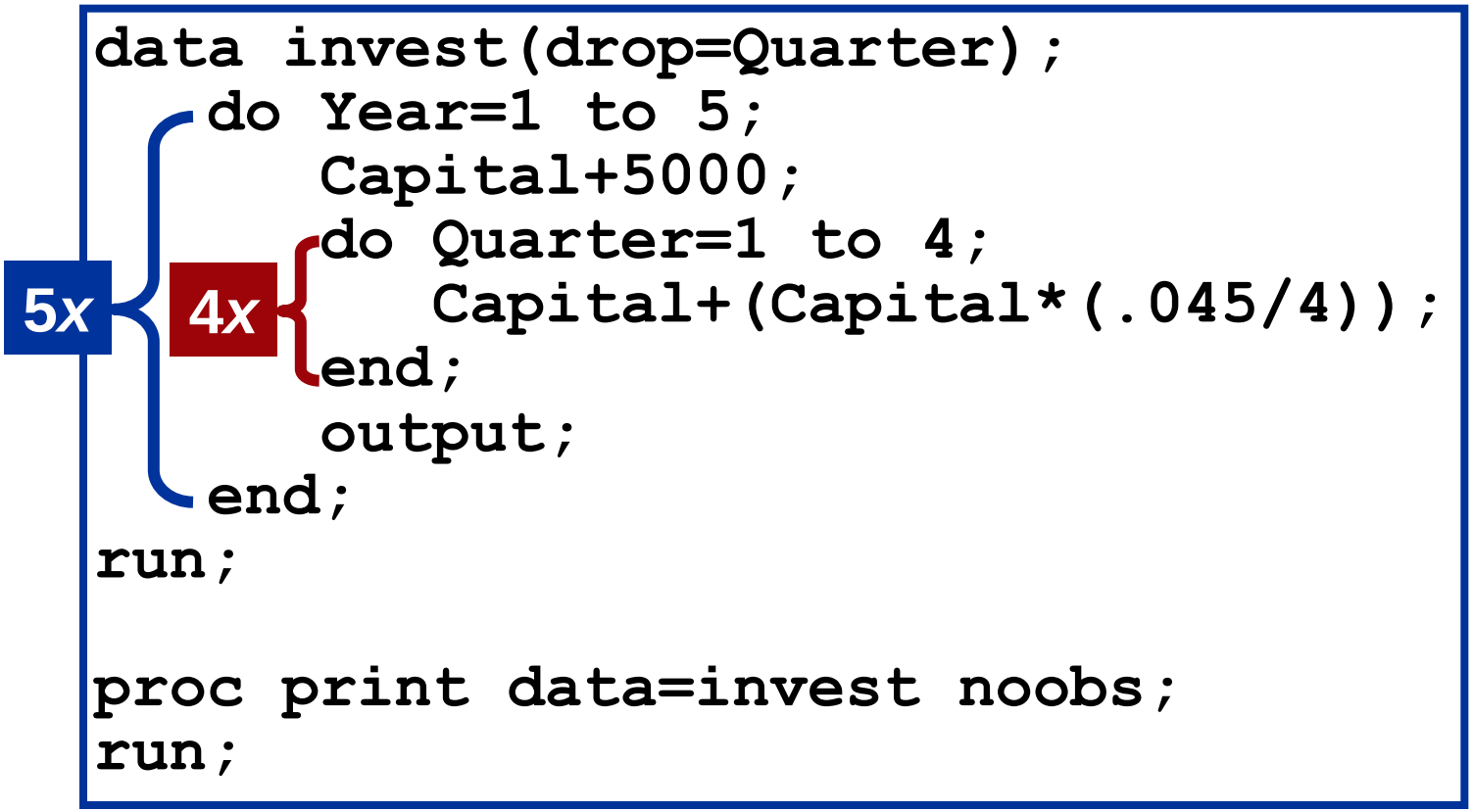
Nested DO loops are loops within loops.

- Be sure to use different index variables for each loop.
- Each DO statement must have a corresponding END statement.
- The inner loop executes completely for each iteration of the outer loop.

```
DO index-variable-1=start TO stop <BY increment>;
    DO index-variable-2=start TO stop  <BY increment>;
        <additional SAS statements>
    END;
END;
```

Business Scenario

Create one observation per year for five years, and show the earnings if you invest $5,000 per year with 4.5 percent annual interest compounded quarterly.




```
data invest(drop=Quarter);
   do Year=1 to 5;
      Capital+5000;
      do Quarter=1 to 4;
         Capital+(Capital*(.045/4));
      end;
      output;
   end;
run;

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

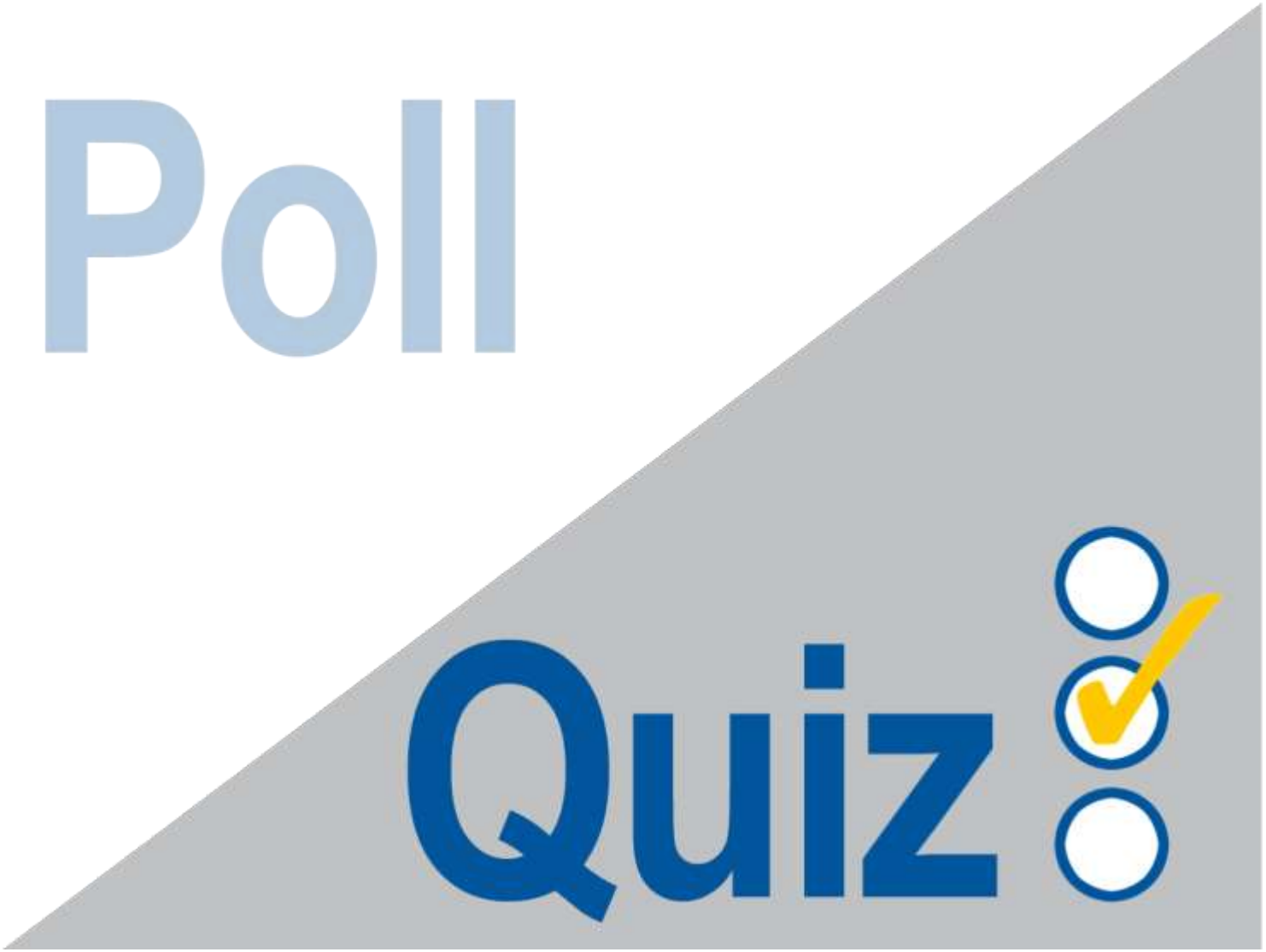

p207d08

Output: Nested DO Loops

PROC PRINT Output

Year	Capital
1	5228.83
2	10696.95
3	16415.32
4	22395.39
5	28649.15

Poll



7.05 Quiz

How can you generate one observation for each quarterly amount?

```
data invest(drop=Quarter);
   do Year=1 to 5;
      Capital+5000;
      do Quarter=1 to 4;
         Capital+(Capital*(.045/4));
      end;
      output;
   end;
run;

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

p207a03

7.05 Quiz – Correct Answer

How can you generate one observation for each quarterly amount? **Move the OUTPUT statement to the inner loop and do not drop `Quarter`.**

```
data invest;
   do Year=1 to 5;
      Capital+5000;
      do Quarter=1 to 4;
         Capital+(Capital*(.045/4));
         output;
      end;
   end;
run;
proc print data=invest
run;
```

Partial PROC PRINT Output

Year	Capital	Quarter
1	5056.25	1
1	5113.13	2
1	5170.66	3
1	5228.83	4
2	10343.90	1
2	10460.27	2

p207a03s

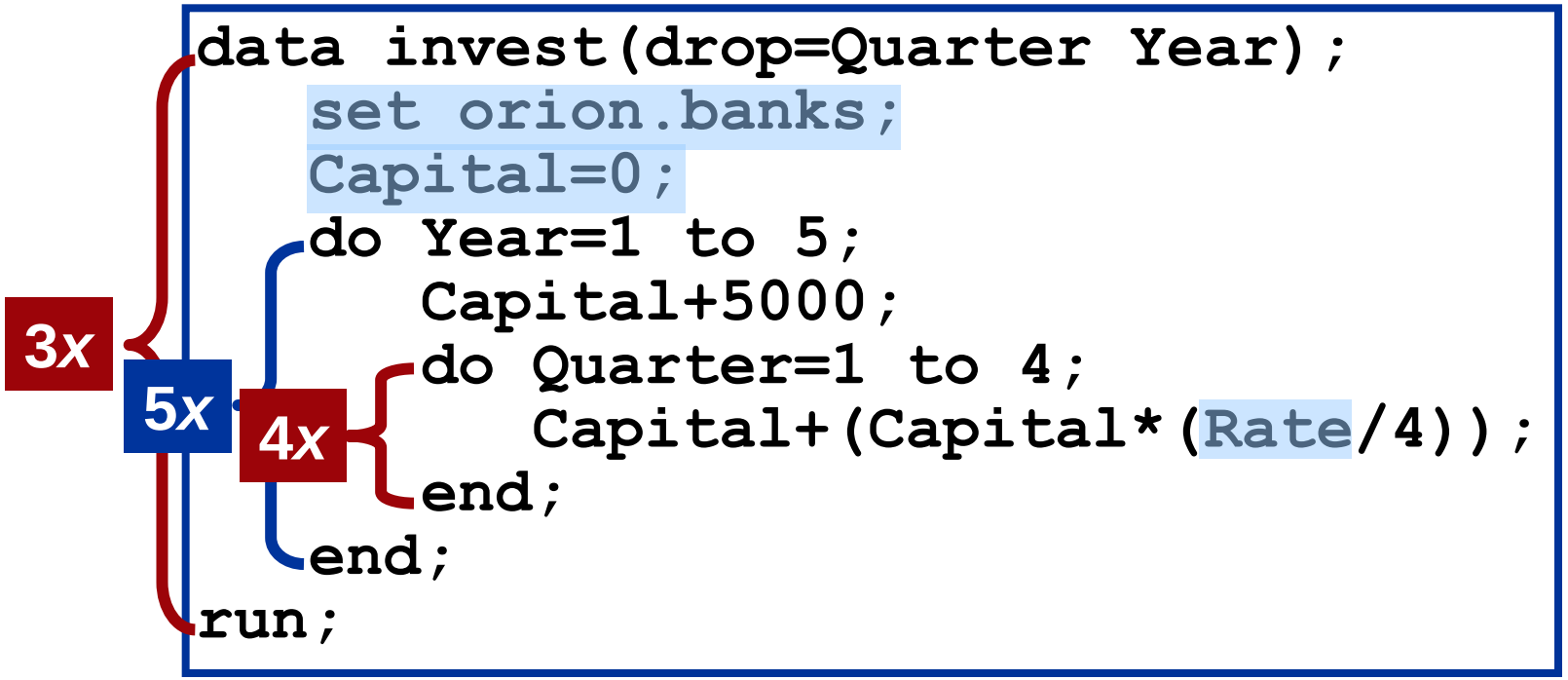
Business Scenario

Compare the final results of investing $5,000 a year for five years in three different banks that compound interest quarterly. Assume that each bank has a fixed interest rate, stored in the **orion.banks** data set.

Listing of **orion.banks**

Name	Rate
Carolina Bank and Trust	0.0318
State Savings Bank	0.0321
National Savings and Trust	0.0328

Using Nested DO Loops with a SET Statement



There are three observations in **orion.banks**. Therefore, there will be three iterations of the DATA step. **Capital** must be set to zero on each iteration of the DATA step.

p207d09

Execution: Nested DO Loops

```
data invest(drop=Quarter Year);
   set orion.banks;
   Capital=0;
   do Year=1 to 5;
      Capital+5000;
      do Quarter=1 to 4;
         Capital+(Capital*(Rate/4));
      end;
   end;
run;
```

First Iteration

0.0318

Partial PDV

Name	Rate	_N_
Carolina Bank and Trust	0.0318	1

...

Execution: Nested DO Loops

```
data invest(drop=Quarter Year);
   set orion.banks;
   Capital=0;
   do Year=1 to 5;
      Capital+5000;
      do Quarter=1 to 4;
         Capital+(Capital*(Rate/4));
      end;
   end;
run;
```

Second Iteration

0.0321

Partial PDV

Name	Rate	_N_
State Savings Bank	0.0321	2

...

Execution: Nested DO Loops

```
data invest(drop=Quarter Year);
   set orion.banks;
   Capital=0;
   do Year=1 to 5;
      Capital+5000;
      do Quarter=1 to 4;
         Capital+(Capital*(Rate/4));
      end;
   end;
run;
```

Third Iteration

0.0328

Partial PDV

Name	Rate	_N_
National Savings and Trust	0.0328	3

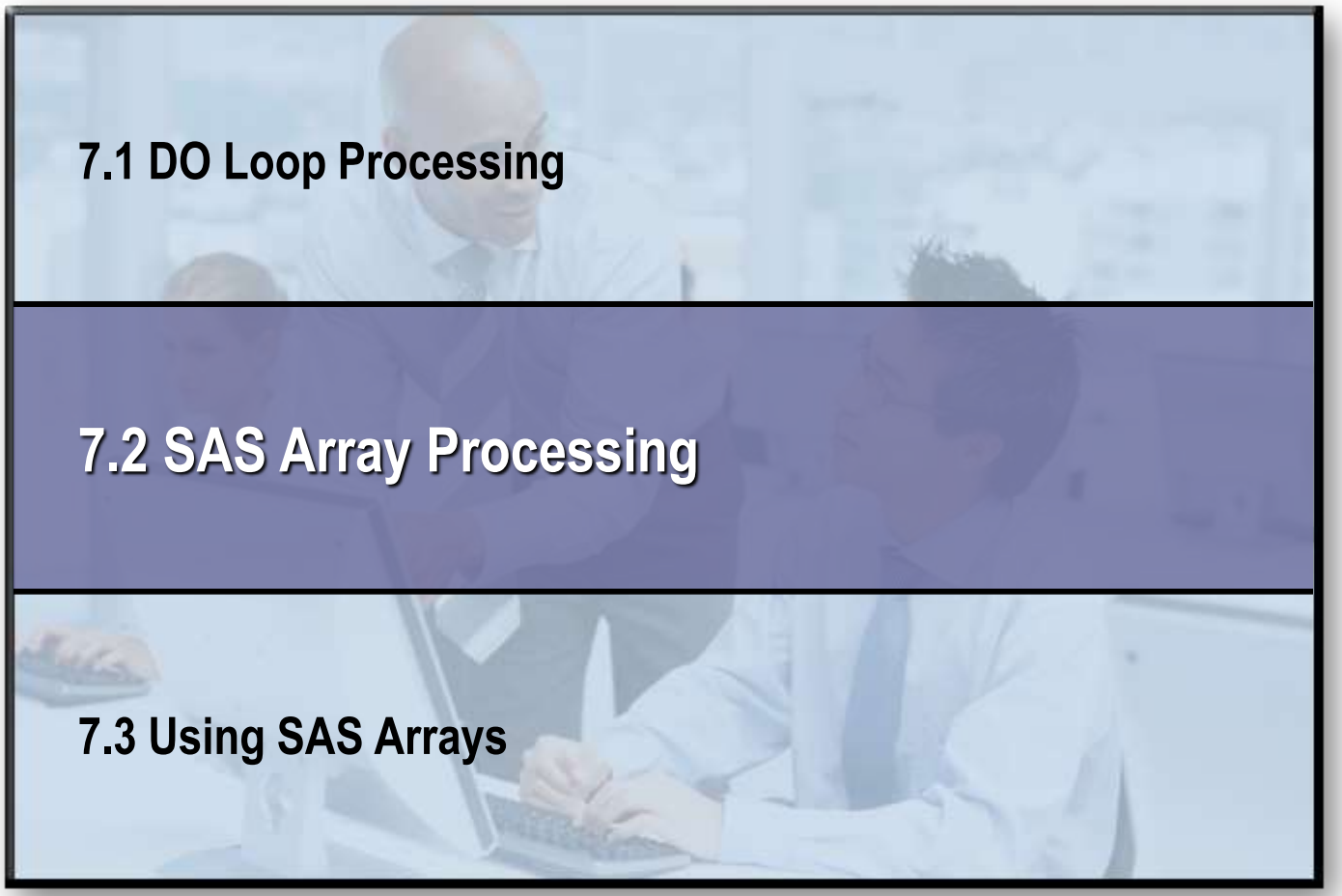
Output: Nested DO Loops

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

PROC PRINT Output

Name	Rate	Capital
Carolina Bank and Trust	0.0318	27519.69
State Savings Bank	0.0321	27544.79
National Savings and Trust	0.0328	27603.47

Chapter 7: Processing Data Iteratively



Objectives

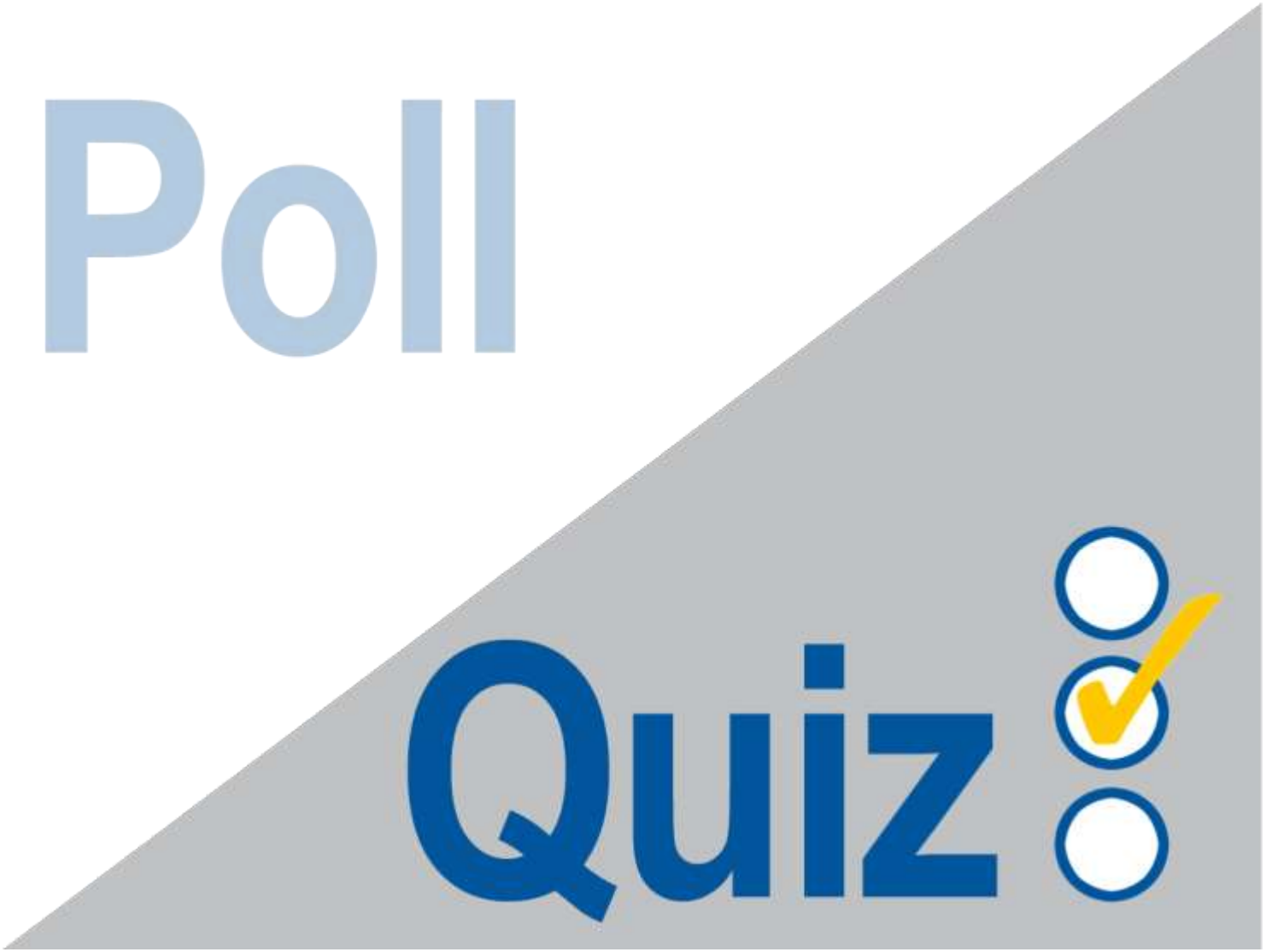
- Explain the concepts of SAS arrays.
- Use SAS arrays to perform repetitive calculations.

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

Poll



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.

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

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

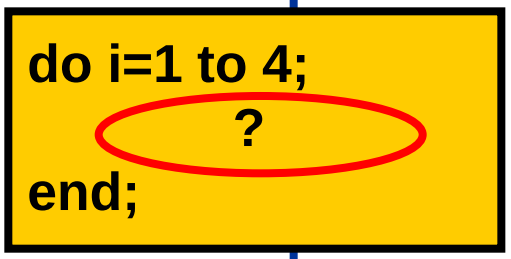
Employee_ID	Qtr1	Qtr2	Qtr3	Qtr4
120265	.	.	.	31.25
120267	18.75	18.75	18.75	18.75
120269	25.00	25.00	25.00	25.00
120270	25.00	12.50	6.25	.

p207d10

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;
```



A SAS array can be used to simplify this code.

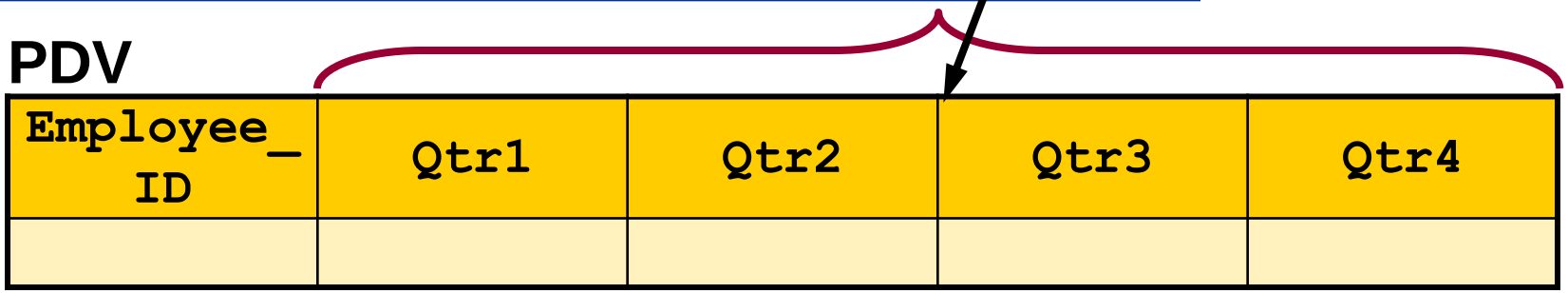
Use Arrays to Simplify Repetitive Calculations

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

An array can be used to access Qtr1-Qtr4.

PDV



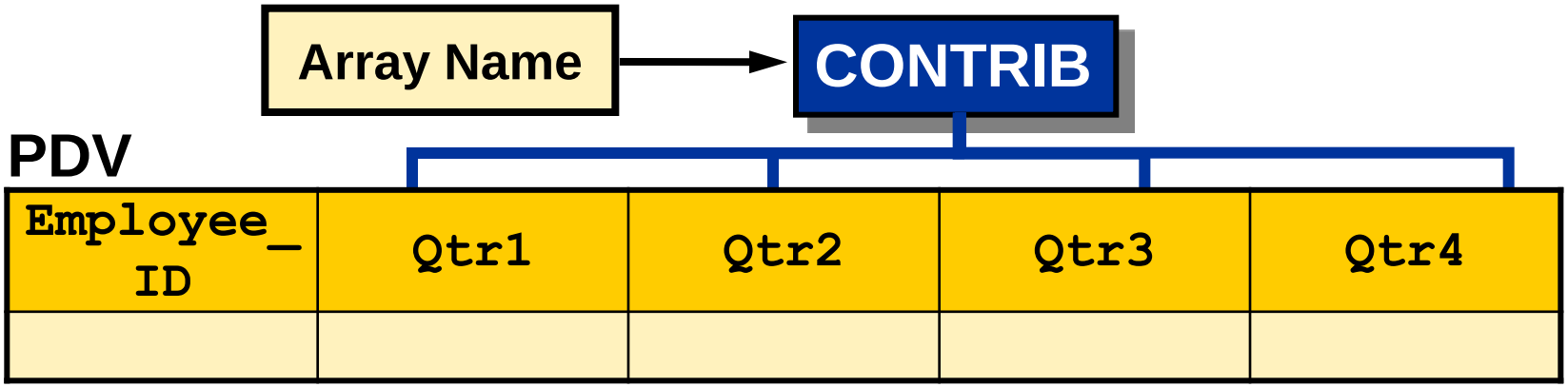
What Is a SAS Array?

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.

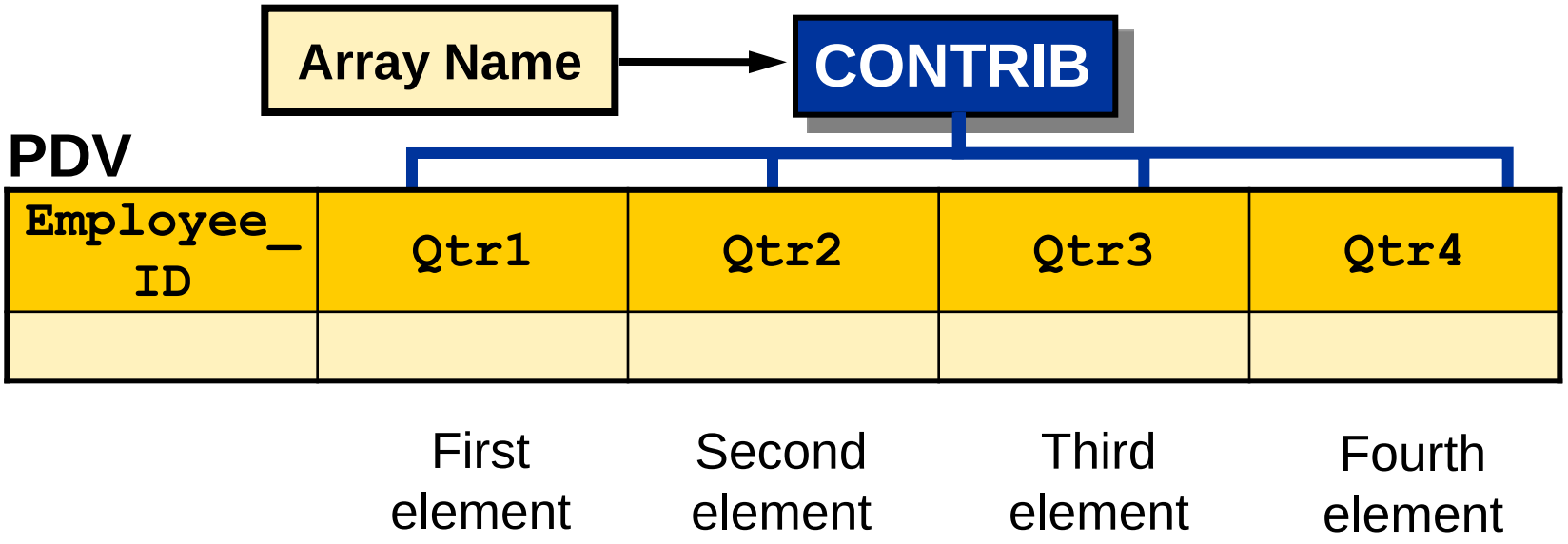
Why Use a SAS Array?

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



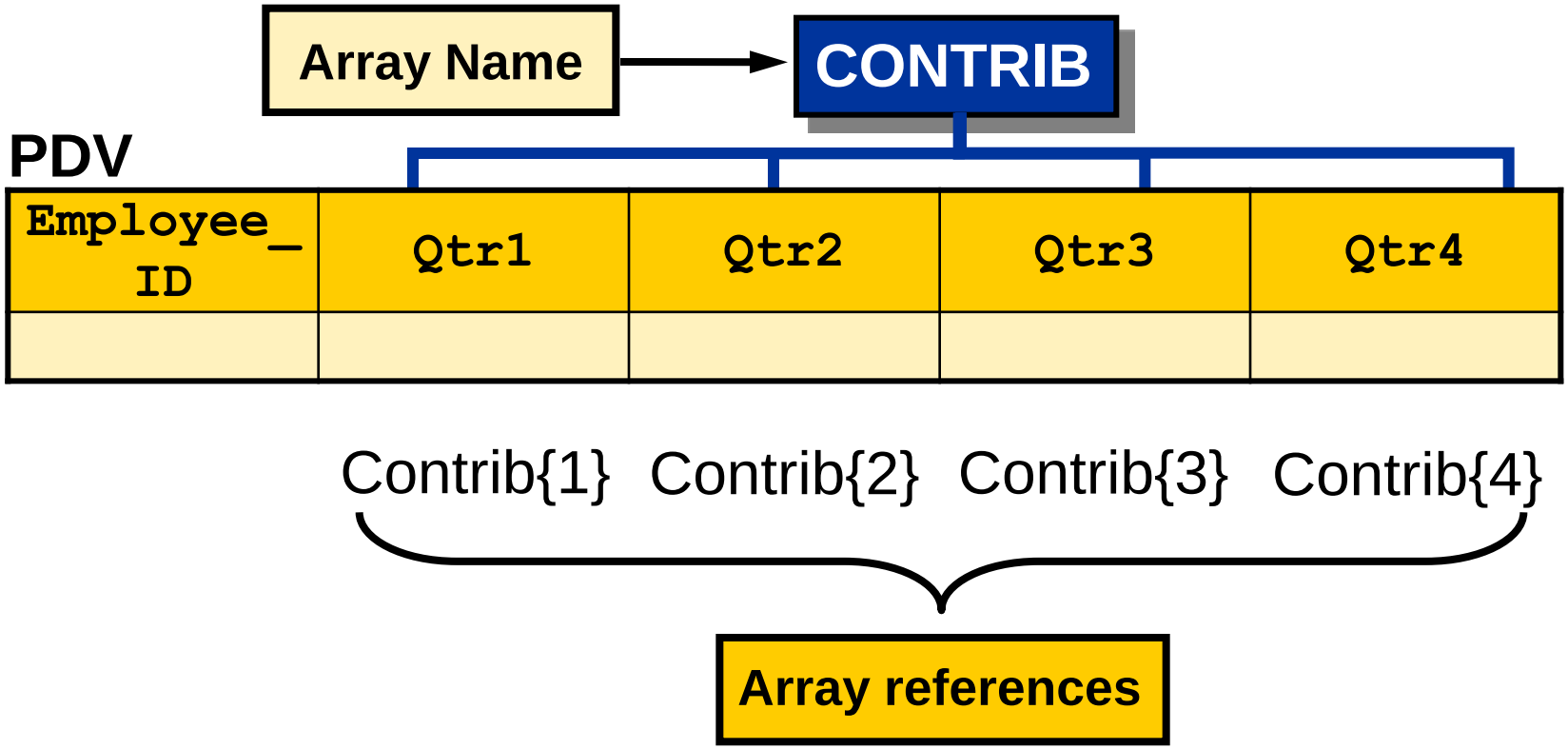
Array Elements

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.

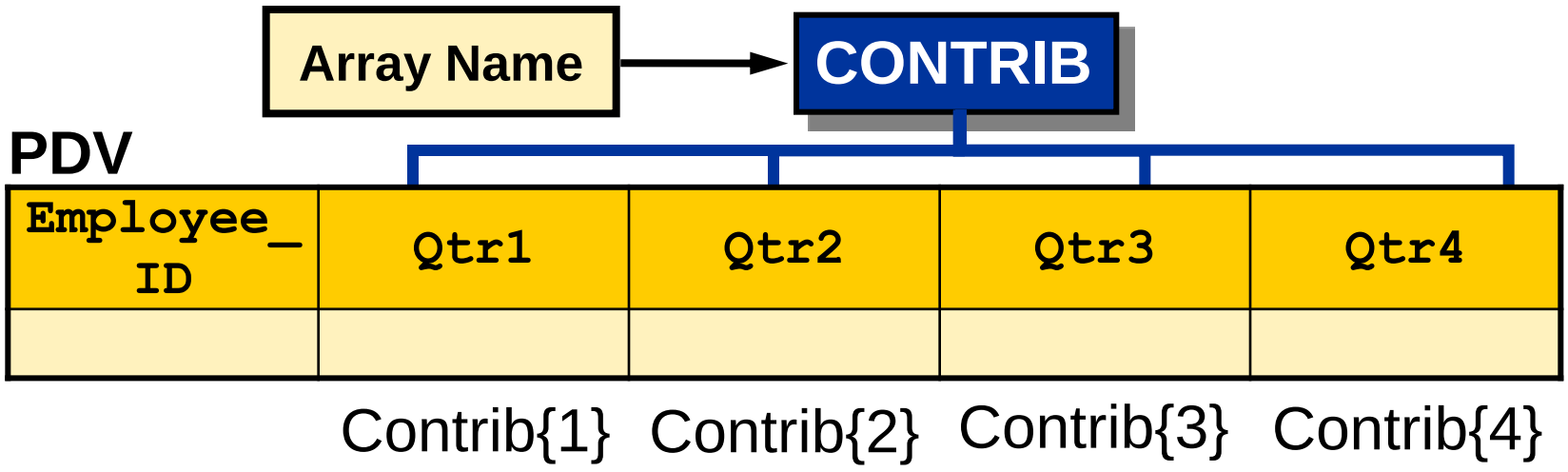
ARRAY *array-name* {*subscript*} <$> <*length*> <*array-elements*>**;**

{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.

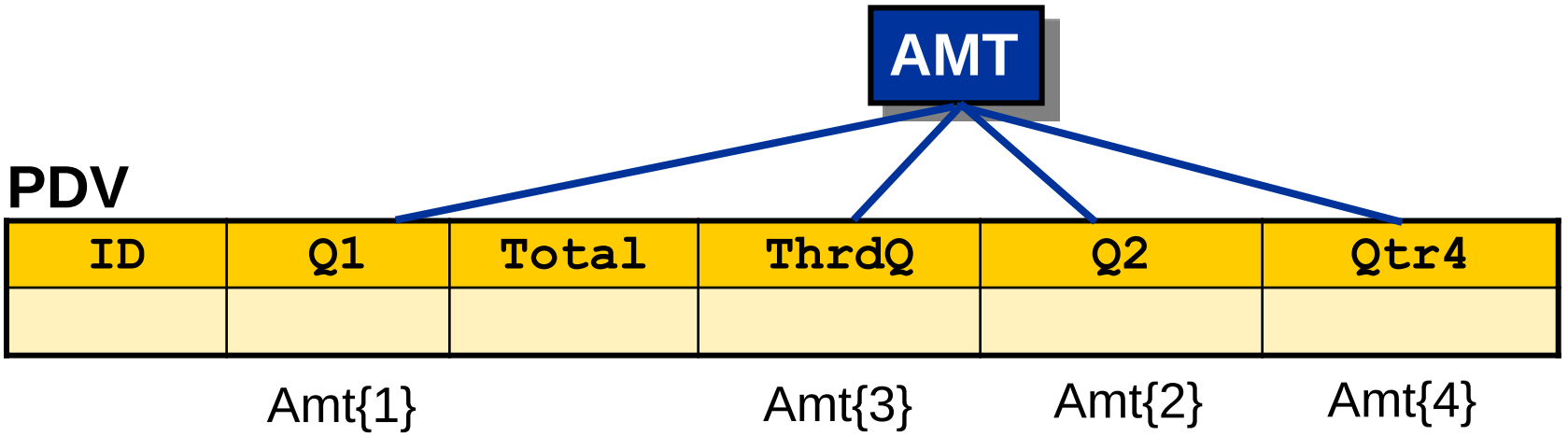
```
array Contrib{*} qtr:;
```

Defining an Array

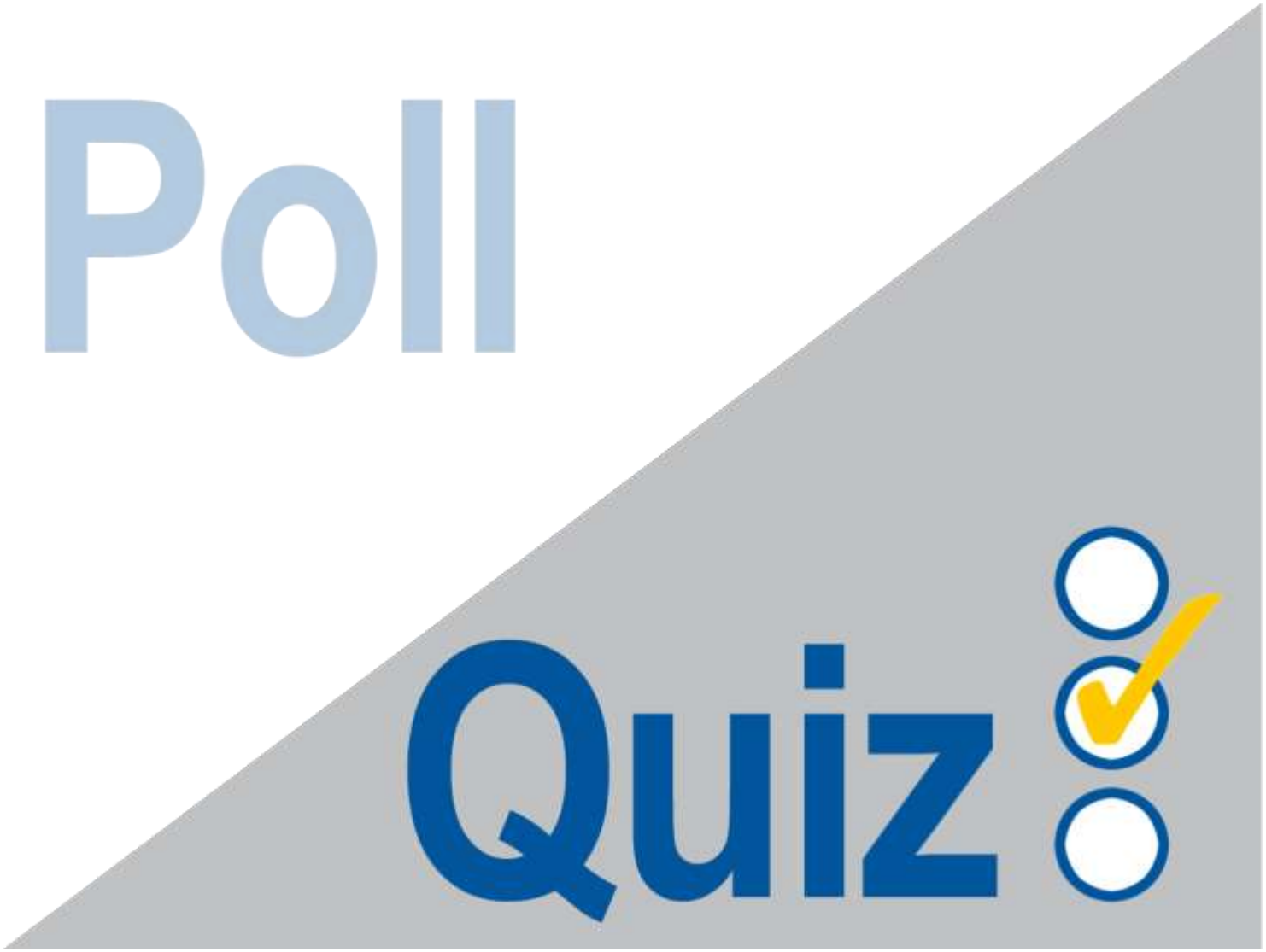
Variables that are elements of an array do not need the following:

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

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



Poll



7.07 Quiz

Open and submit **p207a04**. View the log to determine the cause of the error.

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

p207a04

7.07 Quiz – Correct Answer

Open and submit **p207a04**. View the log to determine the cause of the error. **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}
```

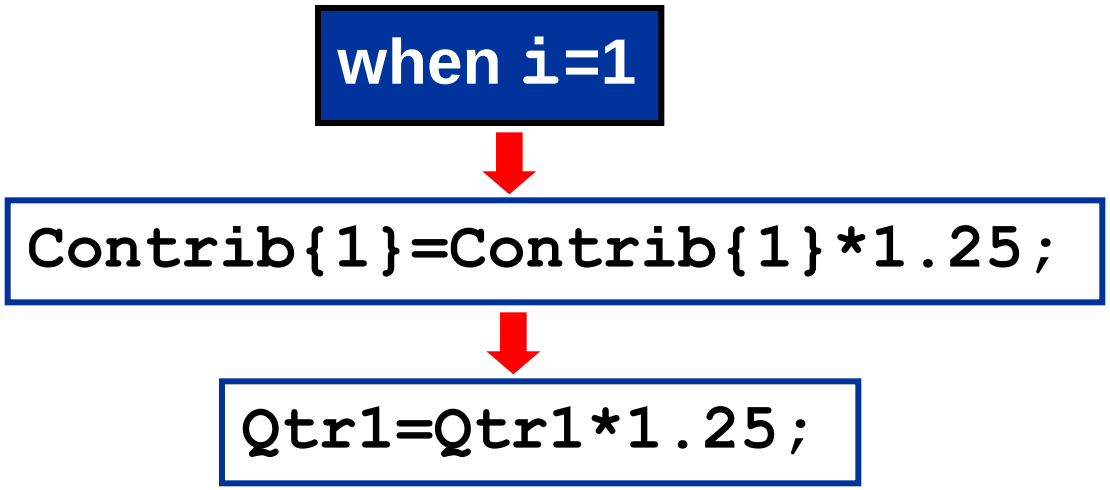
Using a DO Loop to Process an Array

```
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;
```

when i=2

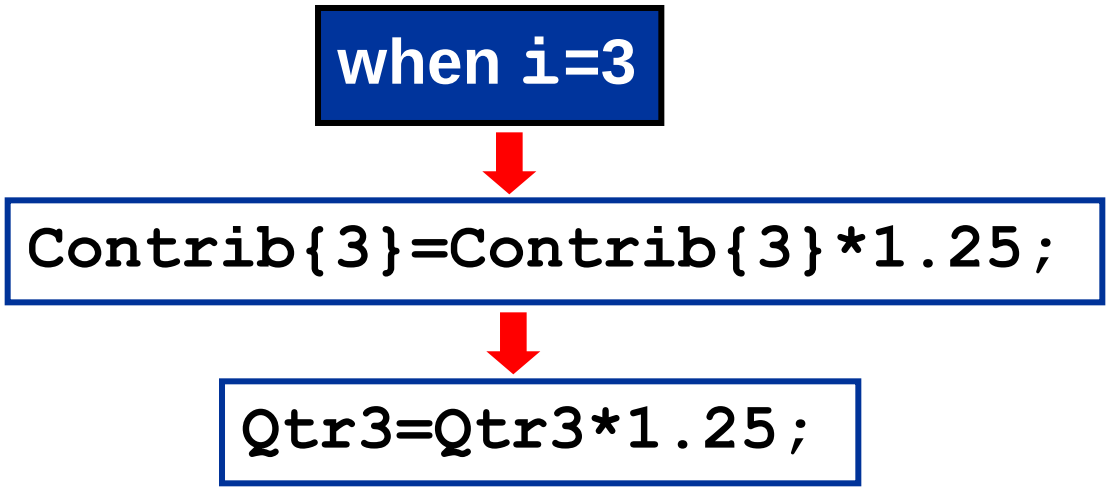
```
Contrib{2}=Contrib{2}*1.25;
```

```
Qtr2=Qtr2*1.25;
```


...

Third Iteration of the DO Loop

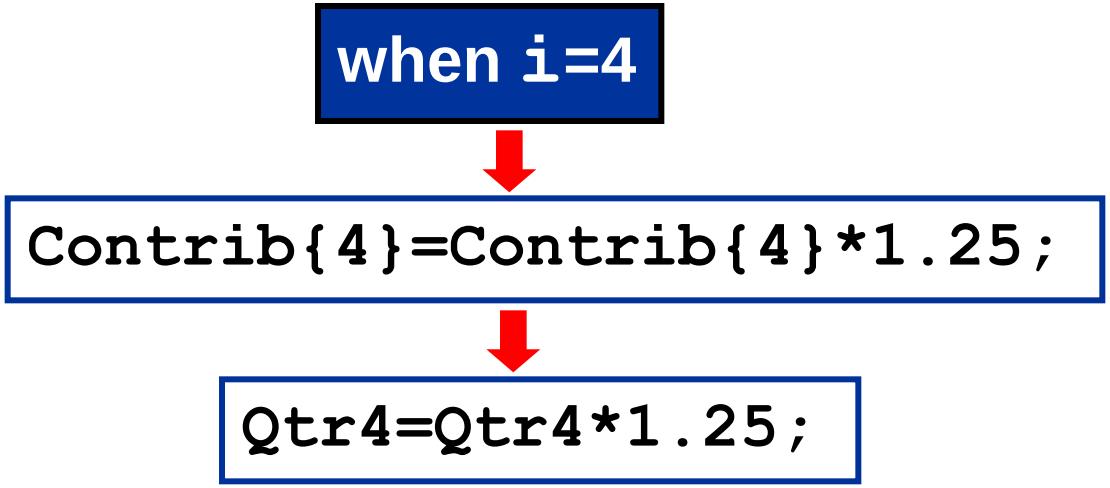
```
data charity;
   set orion.employee_donations;
   keep employee_id qtr1-qtr4;
   array Contrib{4} qtr1-qtr4;
   do i=1 to 4;
      Contrib{i}=Contrib{i}*1.25;
   end;
run;
```



...

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;
```



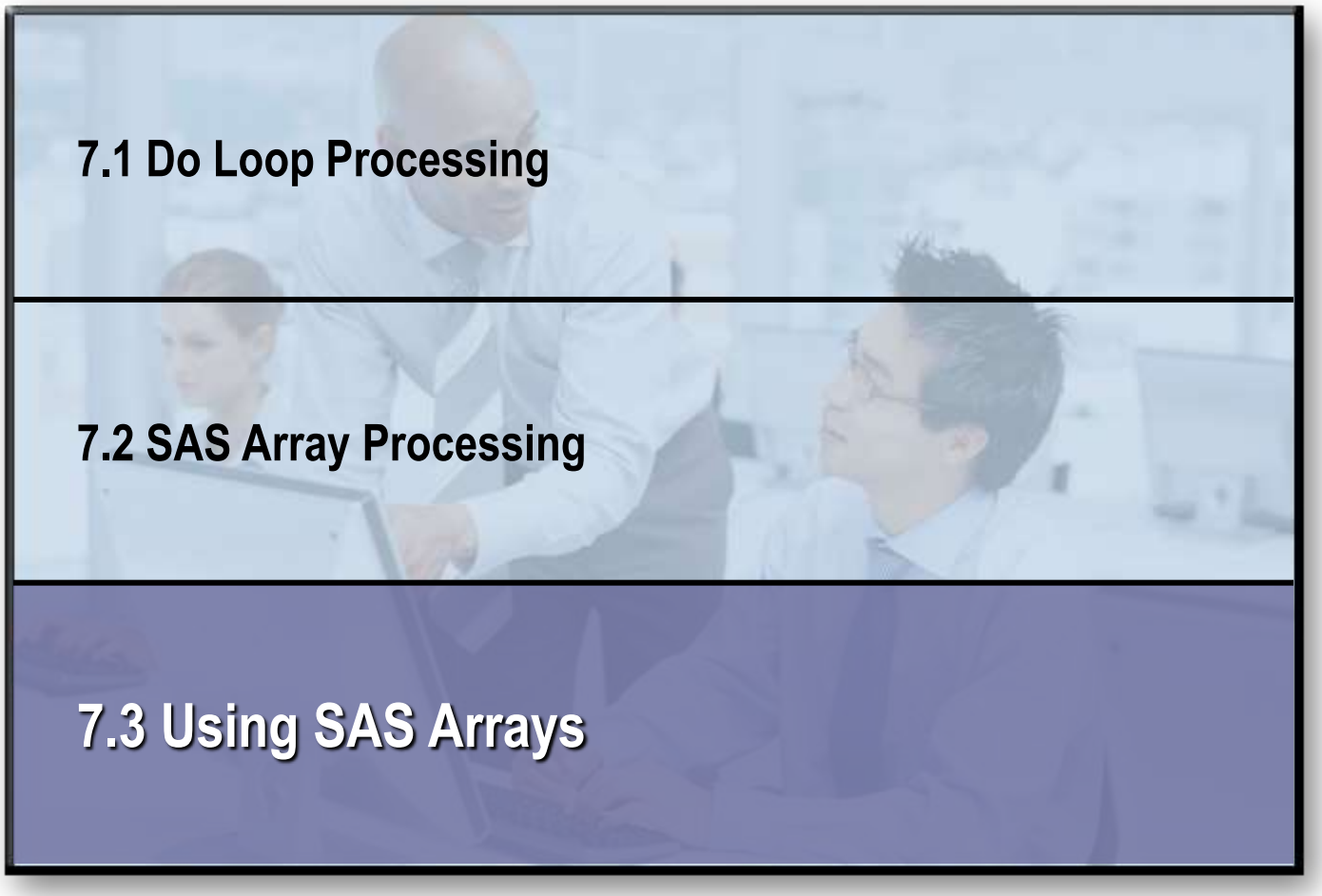
Output: Using a Do Loop to Process an Array

```
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

Chapter 7: Processing Data Iteratively



Objectives

- Use arrays as arguments to SAS functions.
- Explain array functions.
- Use arrays to create new variables.
- Use arrays to perform a table lookup.

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

p207d13

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

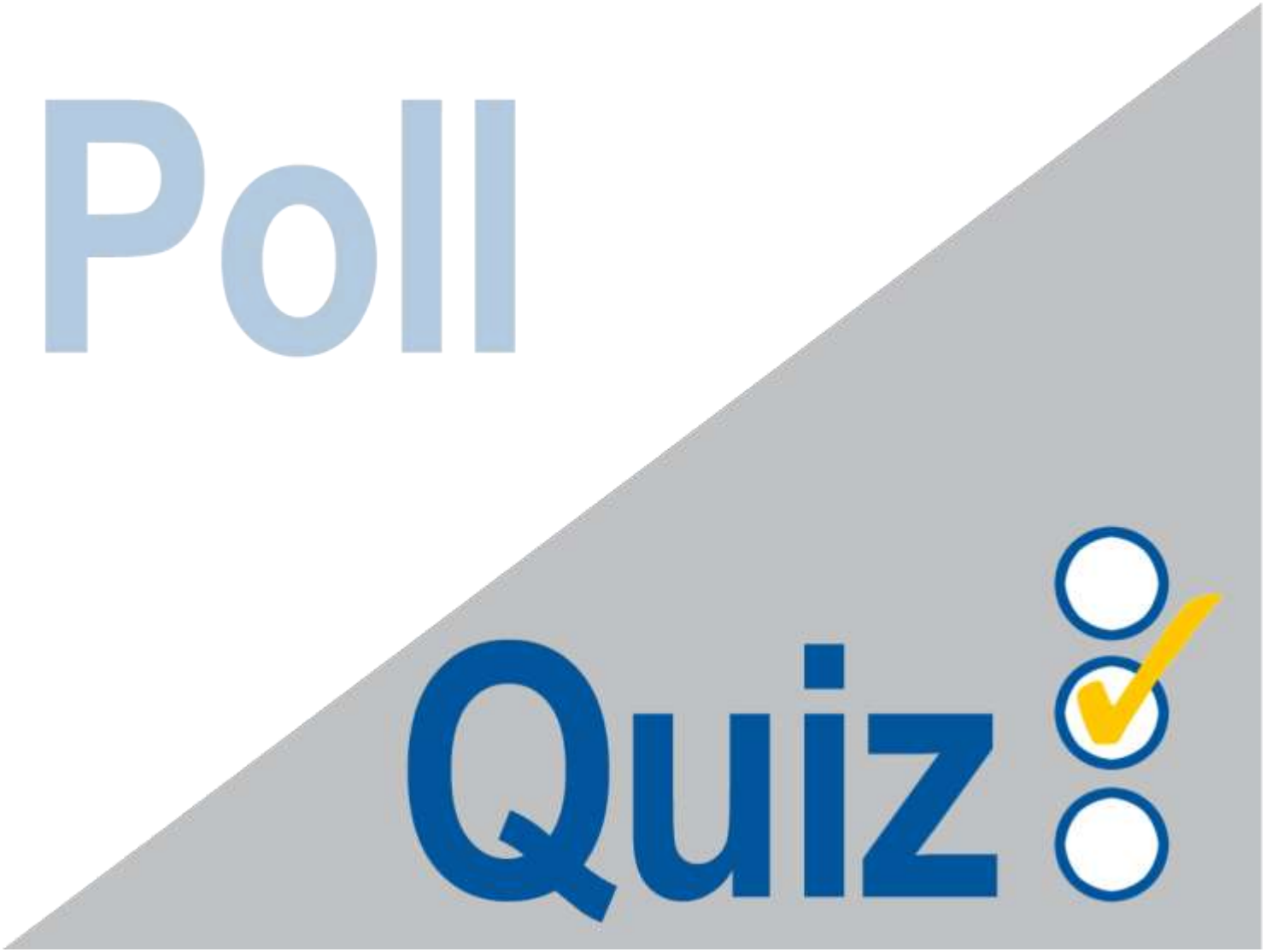
Using **orion.employee_donations** as input, calculate the difference in each employee's contribution from one quarter to the next.

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

Poll



7.08 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

7.08 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.

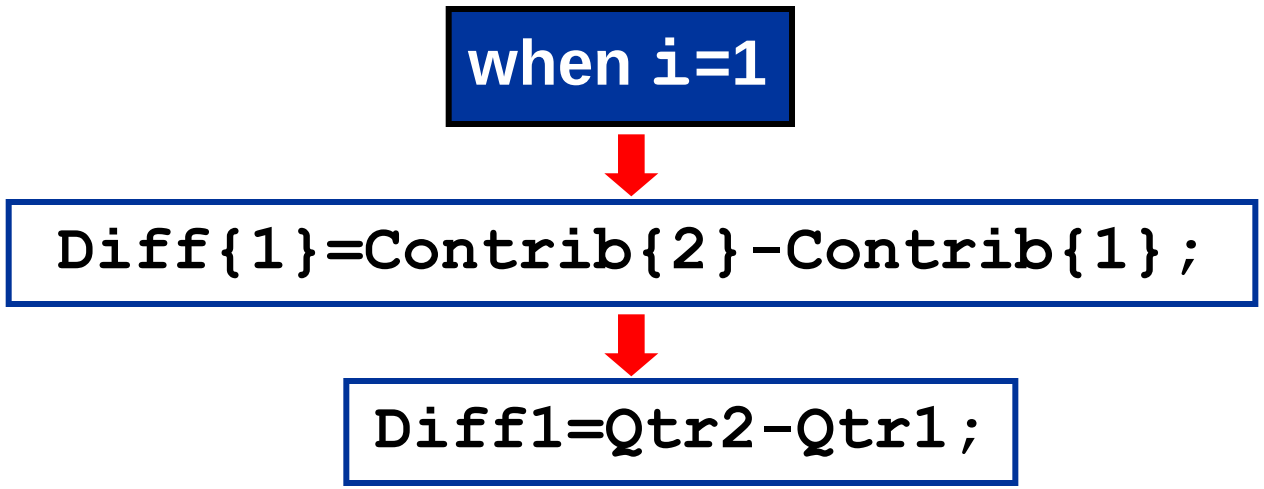
Creating Variables with Arrays

```
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

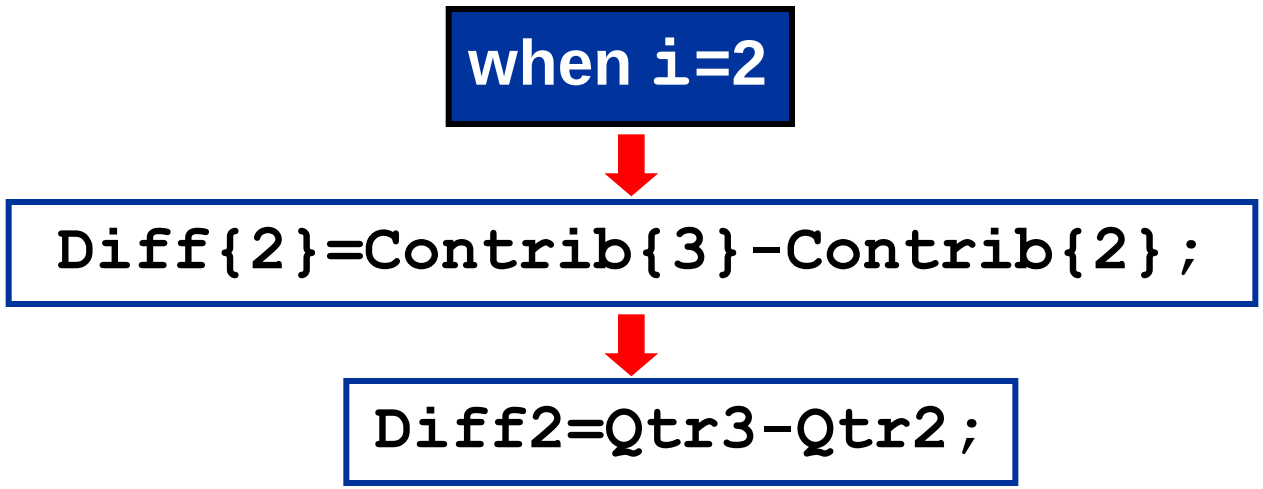
```
data change;
   set orion.employee_donations;
   drop i;
   array Contrib{4} Qtr1-Qtr4;
   array Diff{3};
   do i=1 to 3;
      Diff{i}=Contrib{i+1}-Contrib{i};
   end;
run;
```



...

Creating Variables with Arrays

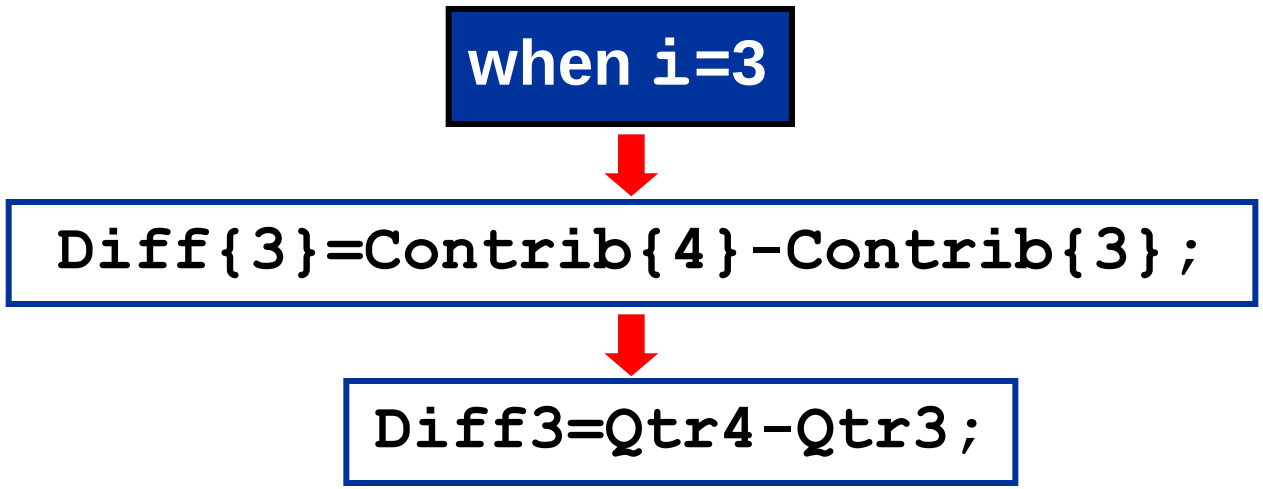
```
data change;
   set orion.employee_donations;
   drop i;
   array Contrib{4} Qtr1-Qtr4;
   array Diff{3};
   do i=1 to 3;
      Diff{i}=Contrib{i+1}-Contrib{i};
   end;
run;
```



...

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