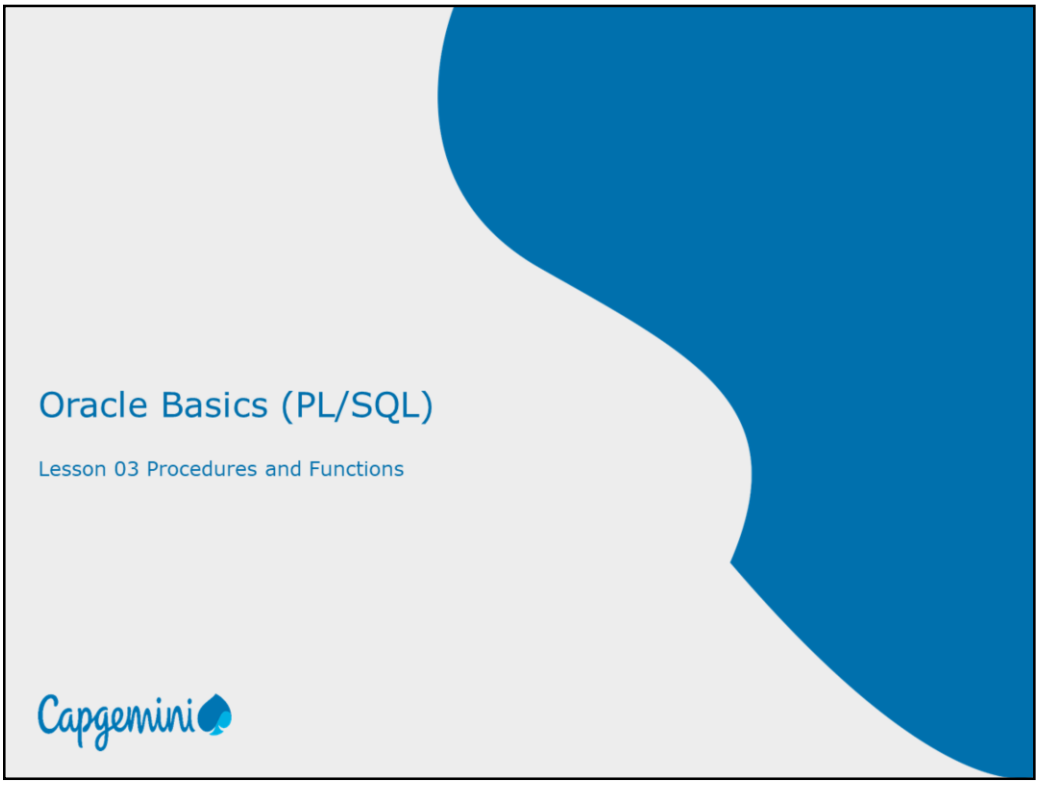
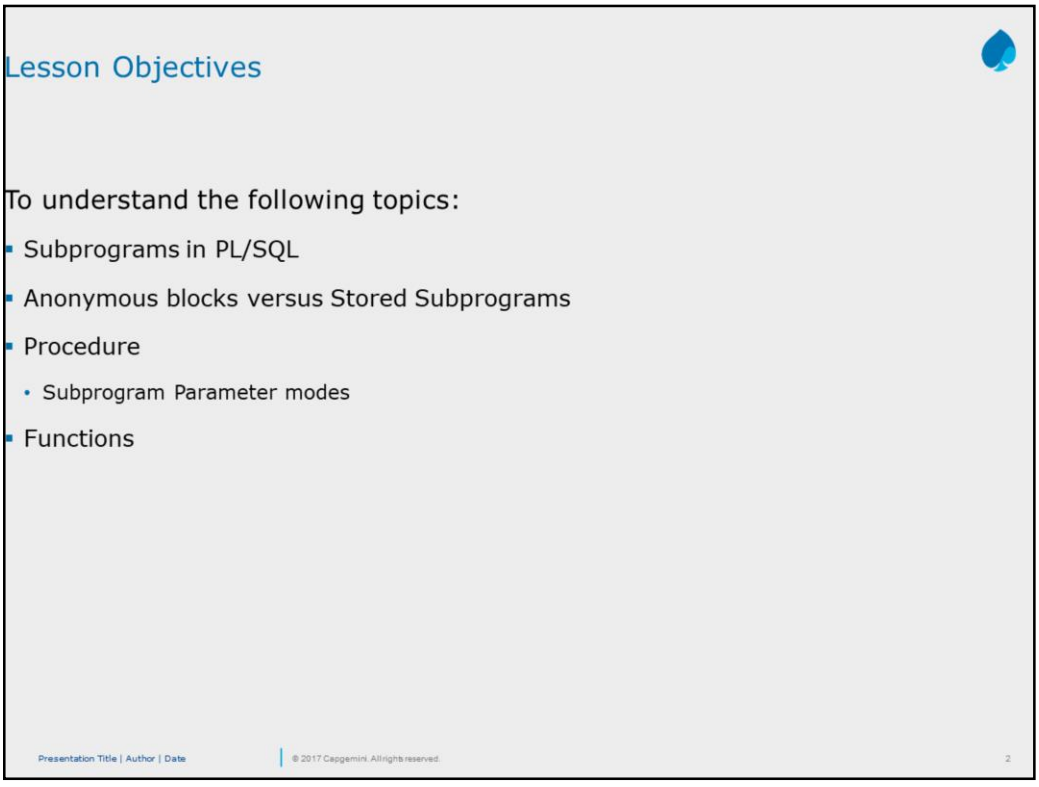
Instructor Notes:



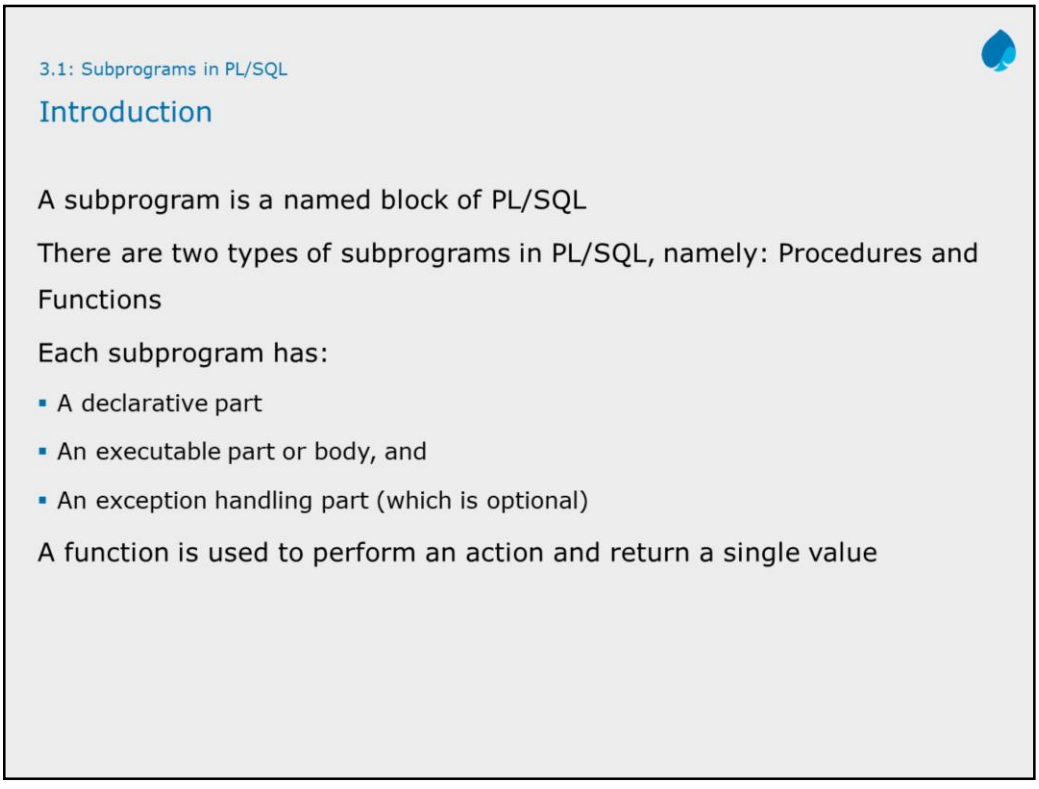
Instructor Notes:

None



Instructor Notes:

None

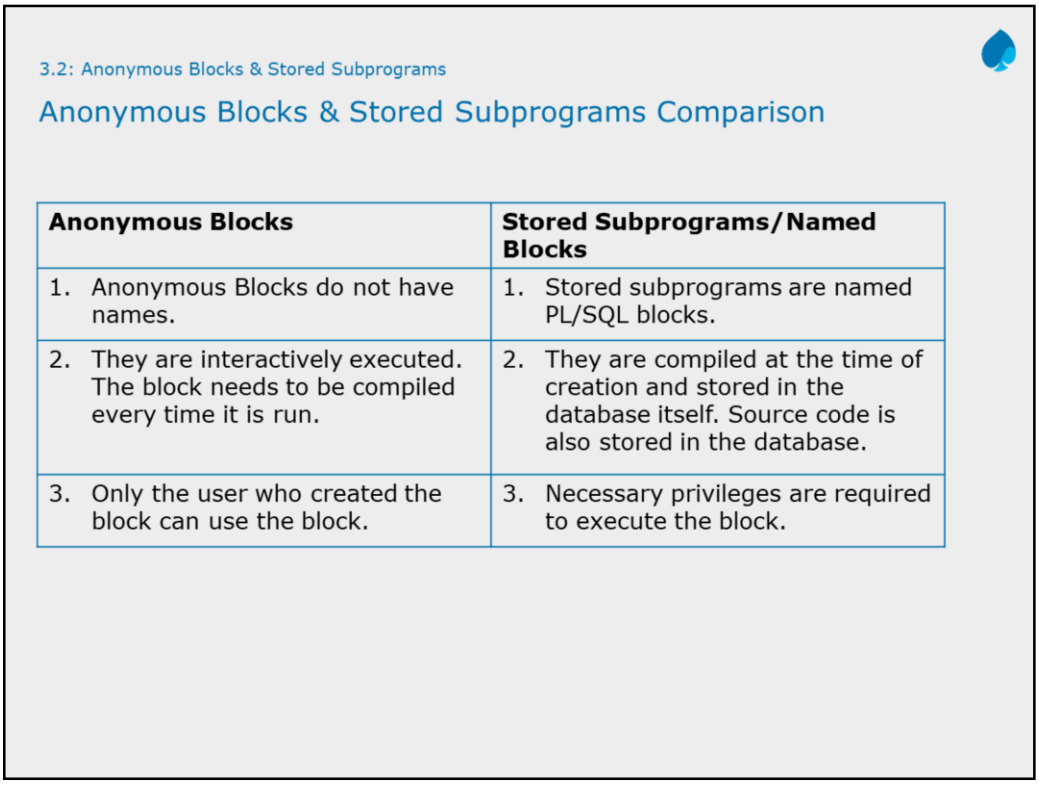


Subprograms in PL/SQL:

- The subprograms are compiled and stored in the Oracle database as “stored programs”, and can be invoked whenever required. As the subprograms are stored in a compiled form, when called they only need to be executed. Hence this arrangement saves time needed for compilation.
- When a client executes a procedure or function, the processing is done in the server. This reduces the network traffic.
- Subprograms provide the following advantages:
  - They allow you to write a PL/SQL program that meets our need.
  - They allow you to break the program into manageable modules.
  - They provide reusability and maintainability for the code.

Instructor Notes:

None



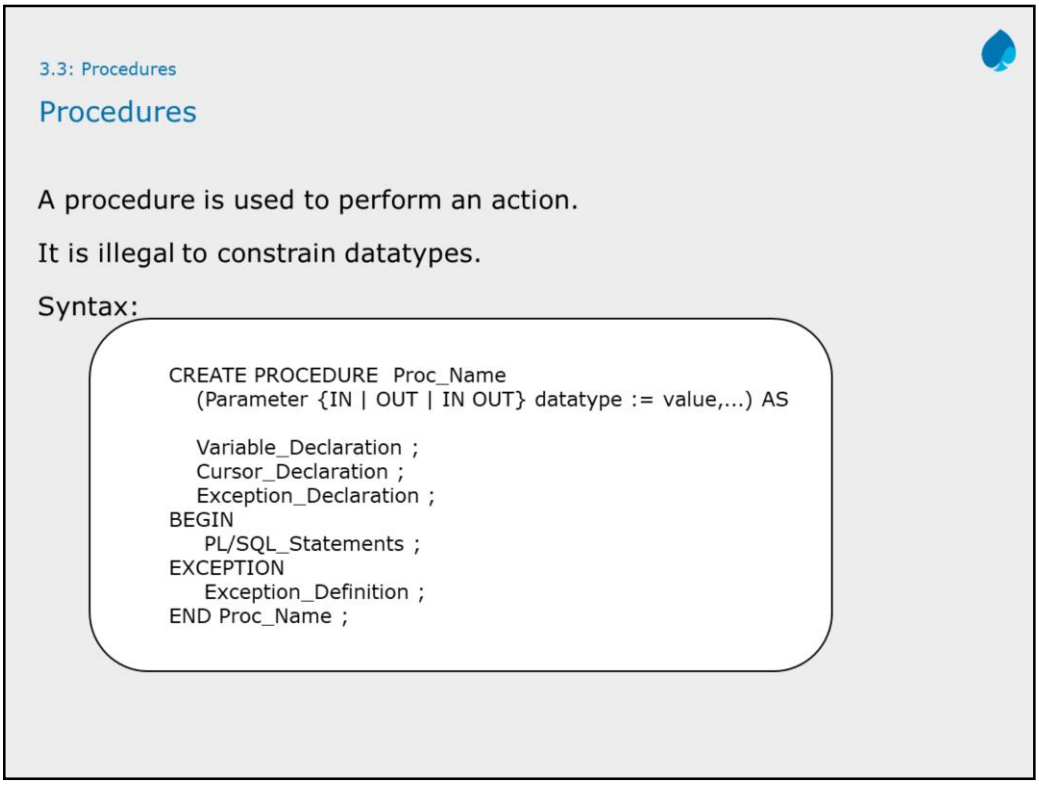
3.2: Anonymous Blocks & Stored Subprograms

Anonymous Blocks & Stored Subprograms Comparison

Anonymous Blocks	Stored Subprograms/Named Blocks
1. Anonymous Blocks do not have names.	1. Stored subprograms are named PL/SQL blocks.
2. They are interactively executed. The block needs to be compiled every time it is run.	2. They are compiled at the time of creation and stored in the database itself. Source code is also stored in the database.
3. Only the user who created the block can use the block.	3. Necessary privileges are required to execute the block.

Instructor Notes:

None



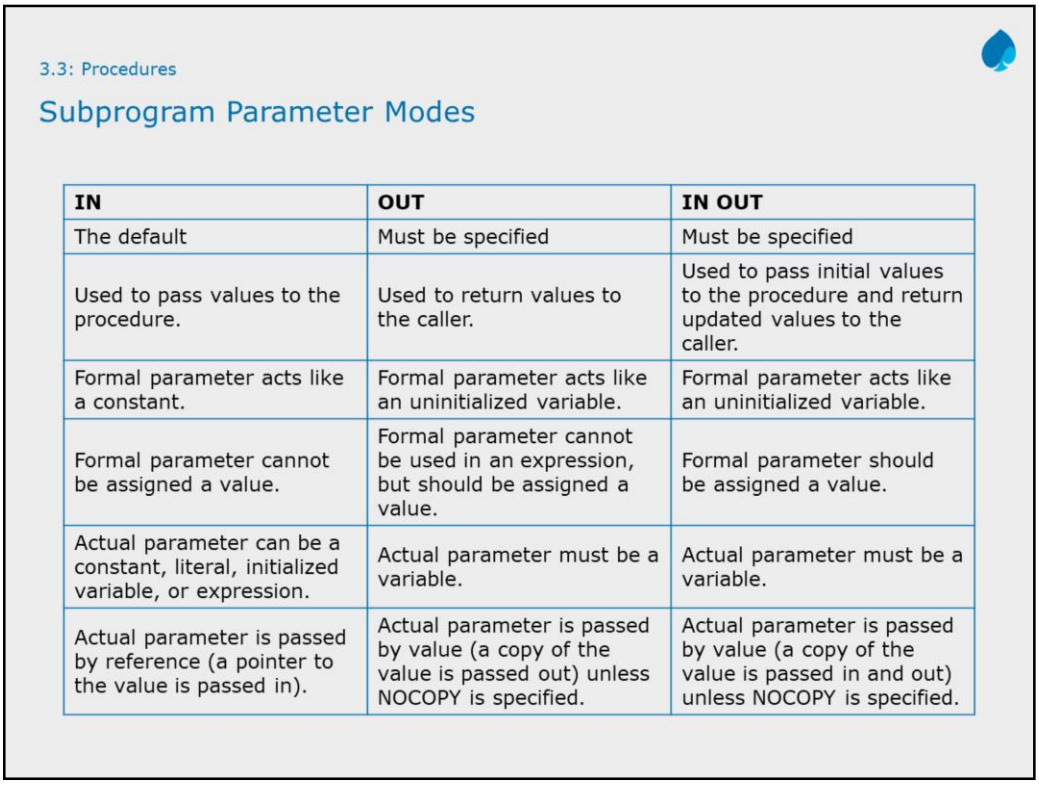
Procedures:

- A procedure is a subprogram used to perform a specific action.
- A procedure contains two parts:
  - the specification, and
  - the body
- The procedure specification begins with CREATE and ends with procedure name or parameters list. Procedures that do not take parameters are written without a parenthesis.
- The procedure body starts after the keyword IS or AS and ends with keyword END.

contd.

Instructor Notes:

None



IN	OUT	IN OUT
The default	Must be specified	Must be specified
Used to pass values to the procedure.	Used to return values to the caller.	Used to pass initial values to the procedure and return updated values to the caller.
Formal parameter acts like a constant.	Formal parameter acts like an uninitialized variable.	Formal parameter acts like an uninitialized variable.
Formal parameter cannot be assigned a value.	Formal parameter cannot be used in an expression, but should be assigned a value.	Formal parameter should be assigned a value.
Actual parameter can be a constant, literal, initialized variable, or expression.	Actual parameter must be a variable.	Actual parameter must be a variable.
Actual parameter is passed by reference (a pointer to the value is passed in).	Actual parameter is passed by value (a copy of the value is passed out) unless NOCOPY is specified.	Actual parameter is passed by value (a copy of the value is passed in and out) unless NOCOPY is specified.

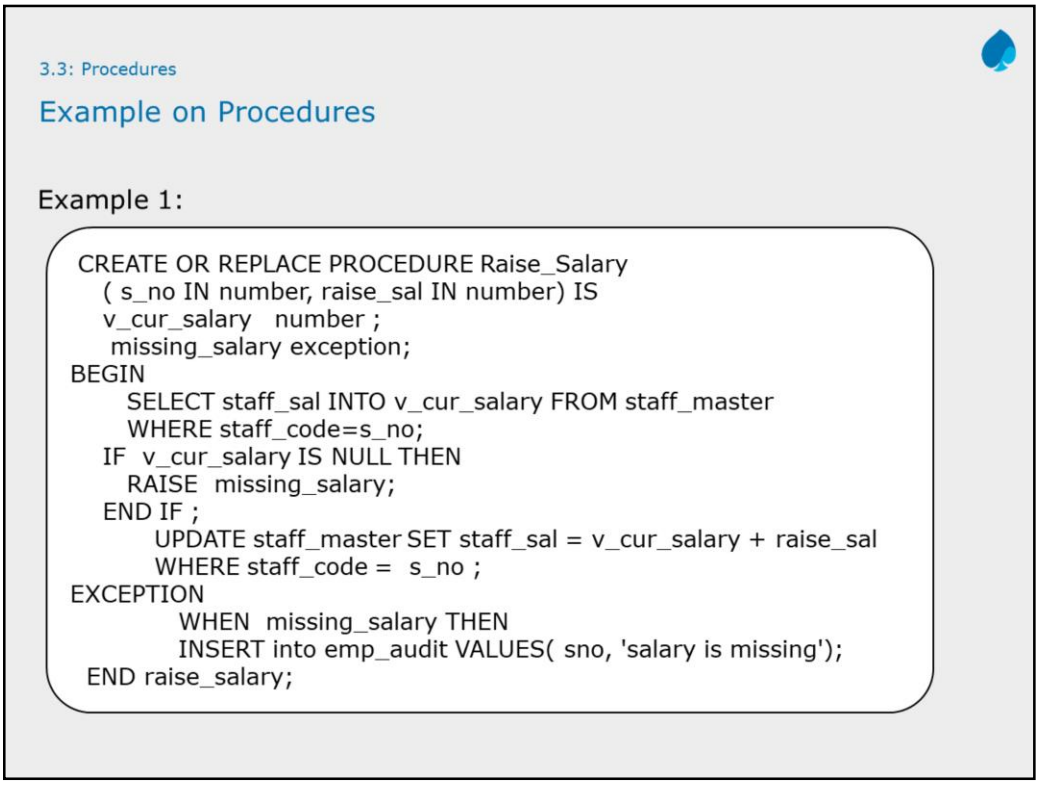
Subprogram Parameter Modes:

- You use “parameter modes” to define the behavior of “formal parameters”. The three parameter modes are IN (the default), OUT, and INOUT. The characteristics of the three modes are shown in the slide.
- Any parameter mode can be used with any subprogram.
- Avoid using the OUT and INOUT modes with functions.
- To have a function return multiple values is a poor programming practice. Besides functions should be free from side effects, which change the values of variables that are not local to the subprogram.
- Example1:

```
CREATE PROCEDURE split_name
(
  phrase IN VARCHAR2, first OUT VARCHAR2, last
OUT VARCHAR2
)
IS
  first := SUBSTR(phrase, 1, INSTR(phrase, ' ')-1);
  last := SUBSTR(phrase, INSTR(phrase, ' ')+1);
  IF first = 'John' THEN
    DBMS_OUTPUT.PUT_LINE('That is a common first
name.');
  END IF;
END;
```

Instructor Notes:

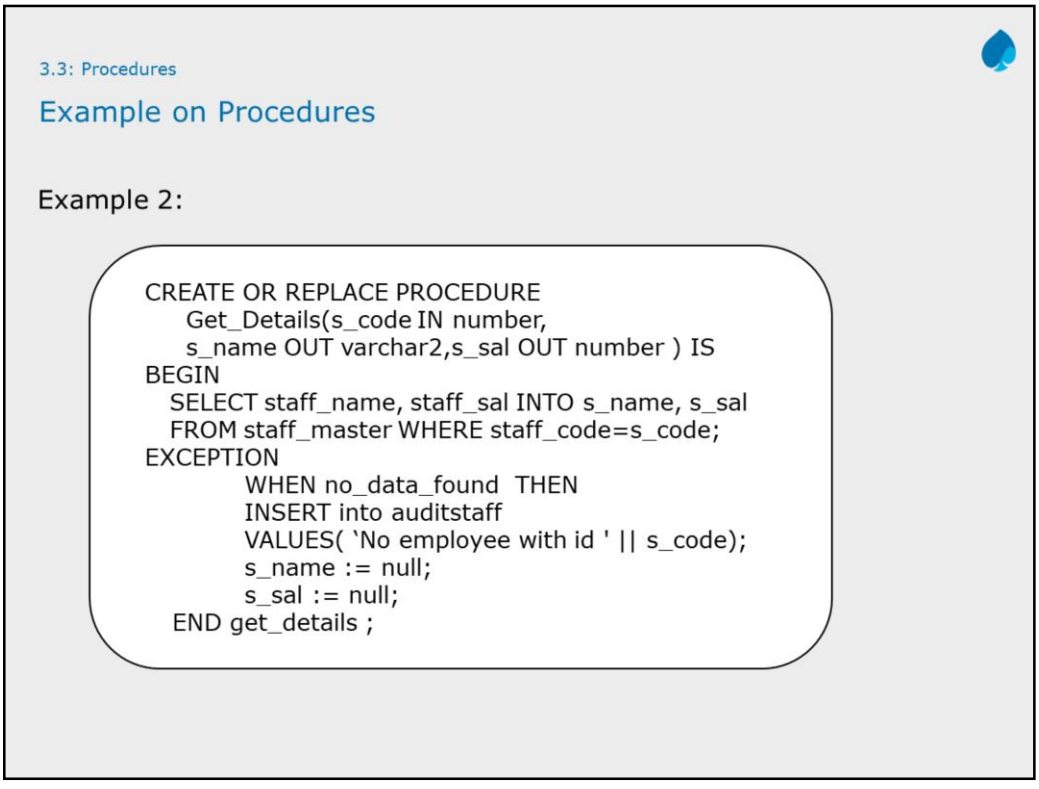
None



The procedure example on the slide modifies the salary of staff member. It also handles exceptions appropriately. In addition to the above shown exception you can also handle “NO_DATA_FOUND” exception. The procedure accepts two parameters which is the staff_code and amount that has to be given as raise to the staff member.

Instructor Notes:

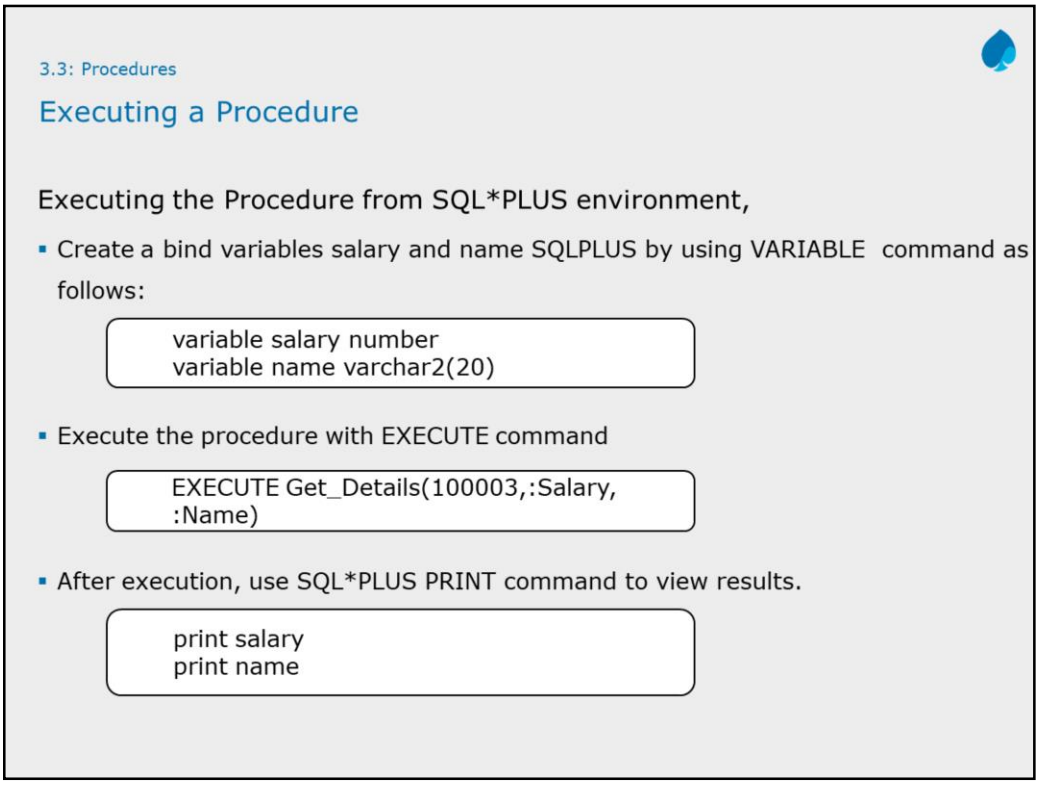
None



The procedure on the slide accept three parameters, one is IN mode and other two are OUT mode. The procedure retrieves the name and salary of the staff member based on the staff_code passed to the procedure. The S_NAME and S_SAL are the OUT parameters that will return the values to the calling environment

Instructor Notes:

This is executing procedure on the command line. The commands are issued at the SQL prompt. Show how the parameters correspond to formal parameters defined in the procedure.



Procedures can be executed through command line as shown on the slide or can be called from other procedures/functions/Anonymous PL/SQL blocks.

On the slide the first snippet declares two variables viz. salary and name. The second snippet calls the procedure and passes the actual parameters. The first is a literal string and the next two parameters are empty variables which will be assigned with values within the procedure.

Calling the procedure from an anonymous PL/SQL block

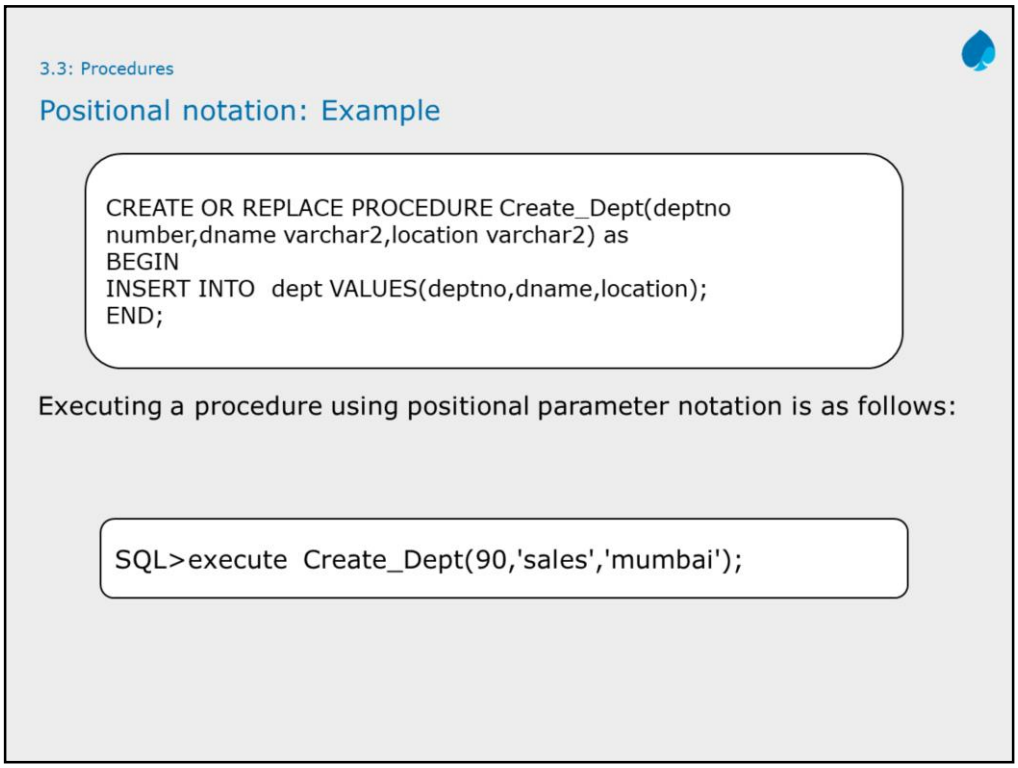
```
DECLARE
      s_no  number(10):=&sno;
      sname varchar2(10);
      sal  number(10,2);
BEGIN
      Get_Details(s_no,sname,sal);
      dbms_output.put_line('Name:'||sname||'Salary'||sal);
END;
```

Instructor Notes:

A bind variable is a variable that you declare in a host environment and then use to pass runtime values.
These values can be character or numeric. You can pass these values either in or out of one or more PL/SQL programs, such as packages, procedures, or functions.
To declare a bind variable in the SQL*Plus environment, you use the command VARIABLE.
For example,
VARIABLE salary NUMBER

Upon declaration, the **bind variables** now become **host** to that environment, and you can now use these variables within your PL/SQL programs, such as packages, procedures, or functions.
To reference host variables, you must add a prefix to the reference with a colon (:) to distinguish the host variables from declared PL/SQL variables.
example
EXECUTE Get_Details(100003,:salary, :name)

Instructor Notes:



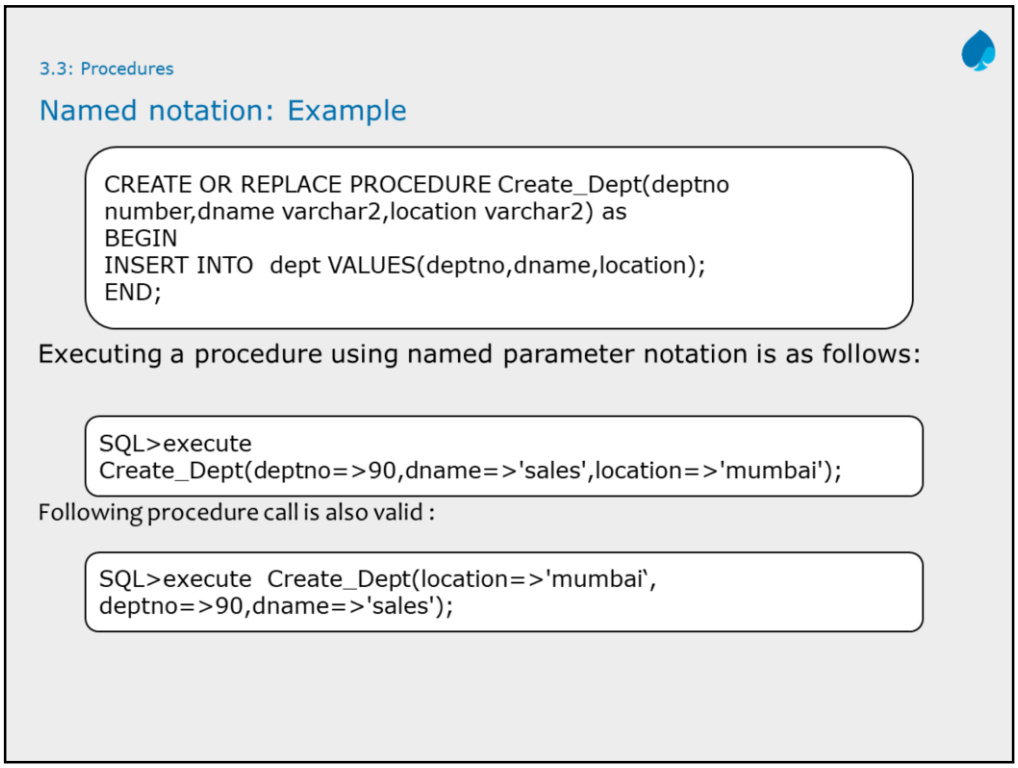
Positional Notation :

You specify the parameters in the same order as they are declared in the procedure.

This notation is compact, but if you specify the parameters (especially literals) in the wrong order, the bug can be hard to detect.

You must change your code if the procedure's parameter list changes

Instructor Notes:

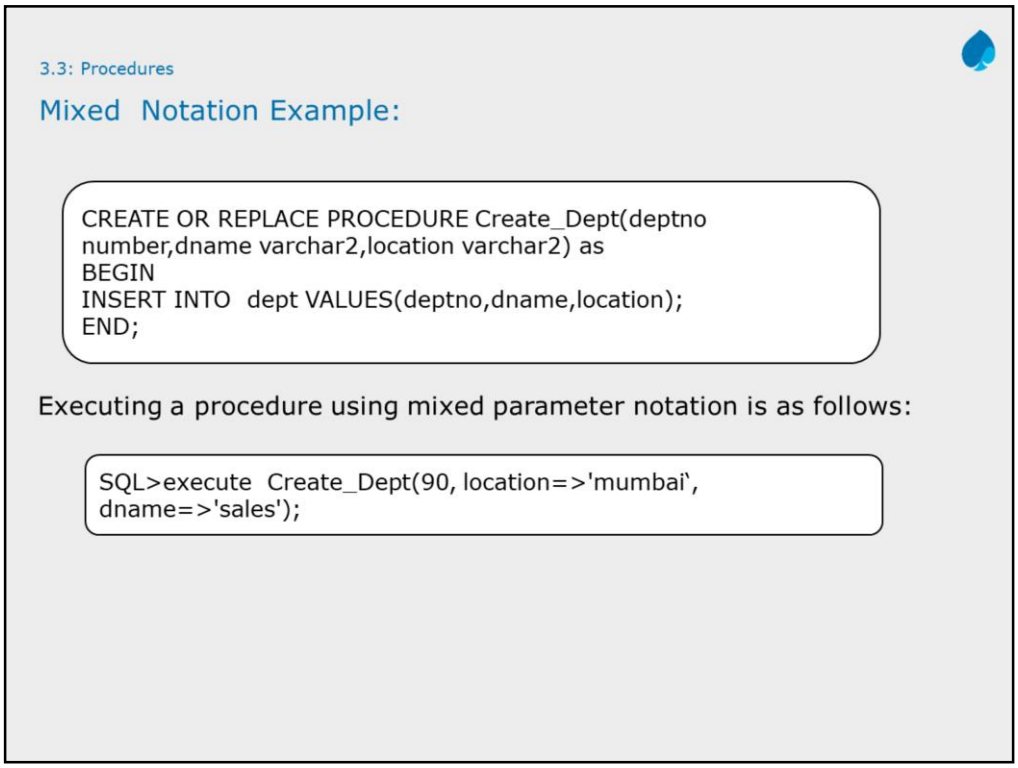


Named notation:

You specify the name of each parameter along with its value. An arrow (=>) serves as the “association operator”. The order of the parameters is not significant.

While executing the procedure, the names of the parameters must be the same as those in the procedure declaration.

Instructor Notes:



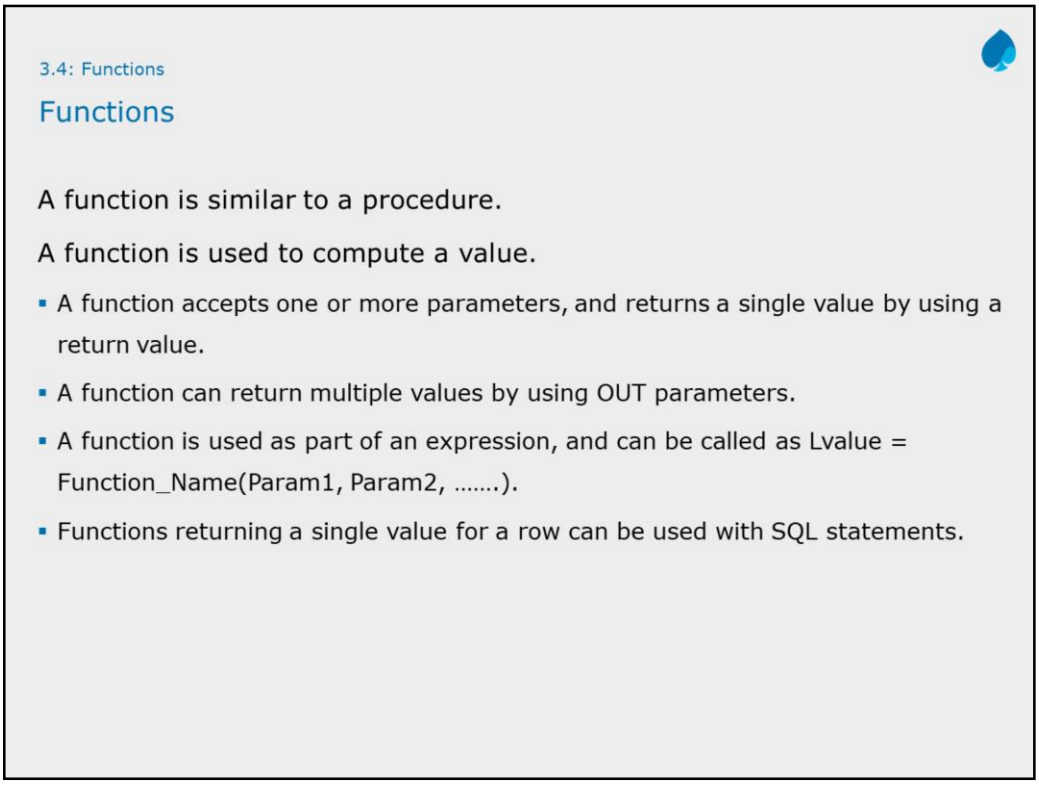
Mixed notation:

You specify the first parameters with “Positional notation”, and then switch to “Named notation” for the last parameters.

You can use this notation to call procedures that have some “required parameters”, followed by some “optional parameters”.

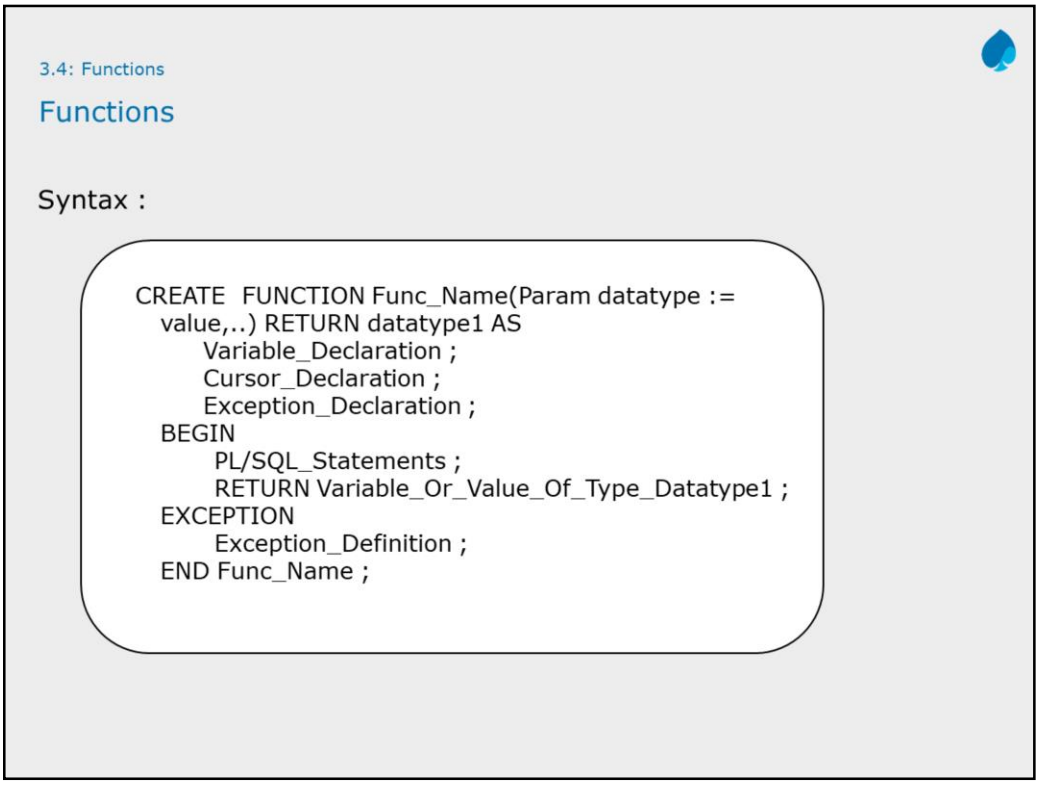
Instructor Notes:

None



Instructor Notes:

None



Instructor Notes:

None

3.4: Functions

Examples on Functions

Example 1:

```
CREATE FUNCTION Crt_Dept(dno number,
  dname varchar2)  RETURN number AS
BEGIN
     INSERT into department_master
     VALUES (dno,dname) ;
              return 1 ;
EXCEPTION
     WHEN others THEN

              return 0 ;

END crt_dept ;
```

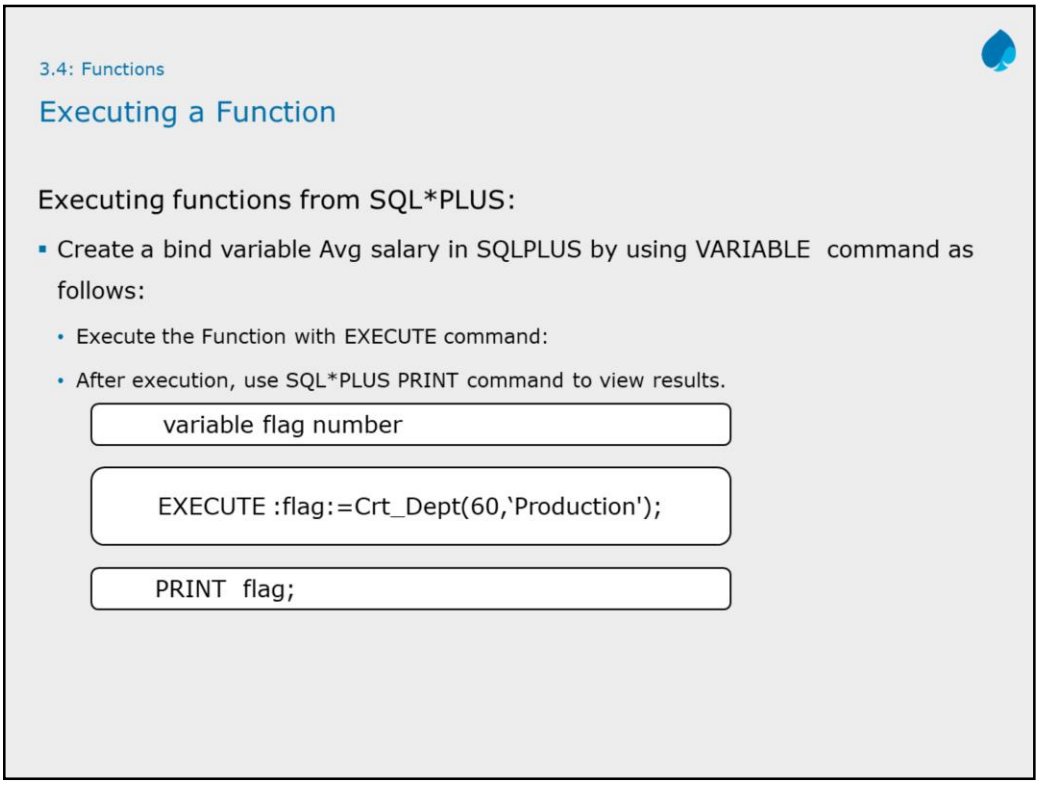
Example 2:

- Function to calculate average salary of a department:
  - Function returns average salary of the department
  - Function returns -1, in case no employees are there in the department.
  - Function returns -2, in case of any other error.

```
CREATE OR REPLACE FUNCTION Get_Avg_Sal(p_deptno
in number) RETURN number AS
   V_Sal number;
BEGIN
         SELECT Trunc(Avg(staff_sal)) INTO V_Sal
         FROM staff_master
         WHERE deptno=P_Deptno;
         IF v_sal is null THEN
                  v_sal := -1 ;
          END IF;
                  return v_sal;
EXCEPTION
        WHEN others THEN
                return -2;  --signifies any other errors
END get_avg_sal;
```

Instructor Notes:

None



Functions can also be executed through command line as shown on the slide or can be called from other procedures/functions/Anonymous PL/SQL blocks.
The second snippet calls the function and passes the actual parameters. The variable declared earlier is used for collecting the return value from the function
Calling the function from an anonymous PL/SQL block

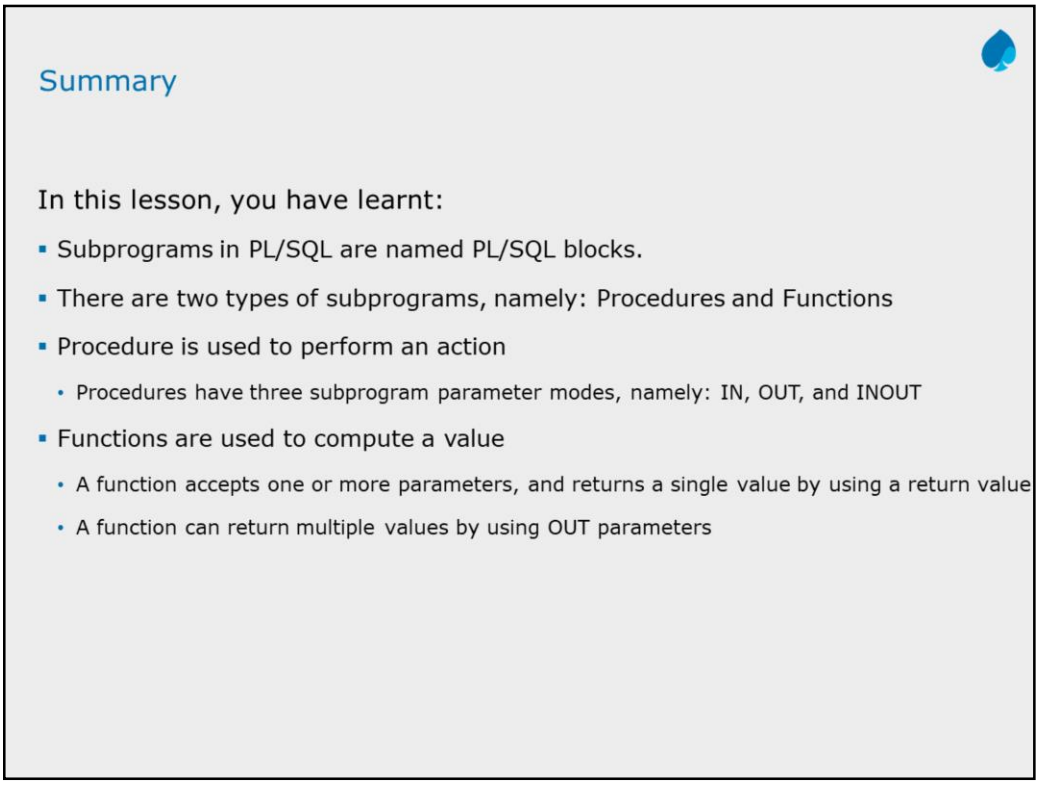
```
DECLARE
  avgsalary  number;
BEGIN
     avgsalary:=Get_Avg_Sal(20);
     dbms_output.put_line('The average salary of Dept 20 is'||
     avgsalary);
END;
```

Calling function using a Select statement

```
SELECT  Get_Avg_Sal(30) FROM staff_master;
```

Instructor Notes:

None



Instructor Notes:

Answers for the Review Questions:

Question 1:
Answer: True

Question 2:
Answer: True

Review – Questions

Question 1: Anonymous Blocks do not have names.

- True / False

Question 2: A function can return multiple values by using OUT parameters

- True / False

Instructor Notes:

Answers for the Review Questions:

Question 3:
Answer: OUT

Question 4:
Answer: specification; body

Question 5:
Answer: Named

Review – Questions

Question 3: An ___ parameter returns a value to the caller of a subprogram.

Question 4: A procedure contains two parts: _____ and _____.

Question 5: In ___ notation, the order of the parameters is not significant.