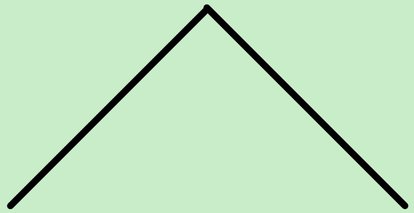
PL/ SQL

FUNCTIONS AND PROCEDURES

BY: 20PW17, 20PW18, 20PW19

PL/ SQL SUBPROGRAM



FUNCTIONS

Return a single value; mainly used to compute & return a value.

PROCEDURES

Do not return a value directly; mainly used to perform an action.

PROCEDURE

CREATING A PROCEDURE

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
{IS | AS}
BEGIN
  < procedure_body >
END procedure_name;
```

```
CREATE OR REPLACE PROCEDURE greetings
AS
BEGIN
   dbms_output.put_line('Hello World!');
END;
/
```

EXECUTING A PROCEDURE

A standalone procedure can be called in two ways:

- Using EXECUTE keyword
- Calling the name of the procedure from PL/SQL block

```
EXECUTE greetings;
```

```
Hello World

PL/SQL procedure successfully completed.
```

```
BEGIN
   greetings;
END;
/
```

```
Hello World

PL/SQL procedure successfully completed.
```

DELETING

A standalone procedure is deleted with the **DROP PROCEDURE** statement.

```
DROP PROCEDURE procedure-name;
```

```
DROP PROCEDURE greetings;
```

PARAMETERS

You will have to define parameters to create a procedure.
There are three ways to pass a parameter in a procedure.

- **IN** Parameter:
  - Used for giving input to the subprograms.
  - Read - only variable.
  - Value cannot be overwritten by the procedure.
  - By default, the parameters are of IN type.

- **OUT** Parameter:
  - Used for getting output from the subprograms
  - Read - write variable.
  - Value can be overwritten by the procedure.

- **INOUT** Parameter:
  - Used for both, giving input and getting output from subprograms.
  - Read - write variable.
  - Value can be overwritten by the procedure.

EXAMPLE 1

```
DECLARE
   a number;
   b number;
   c number;
PROCEDURE findMin(x IN number, y IN number, z OUT number) IS
BEGIN
   IF x < y THEN
      z:= x;
   ELSE
      z:= y;
   END IF;
END;
BEGIN
   a:= 23;
   b:= 45;
   findMin(a, b, c);
   dbms_output.put_line(' Minimum of (23, 45) : ' || c);
END;
/
```

```
Minimum of (23, 45) : 23

PL/SQL procedure successfully completed.
```

EXAMPLE 2

```
DECLARE
   a number;
PROCEDURE squareNum(x IN OUT number) IS
BEGIN
  x := x * x;
END;
BEGIN
   a:= 23;
   squareNum(a);
   dbms_output.put_line(' Square of (23): ' || a);
END;
/
```

```
Square of (23): 529

PL/SQL procedure successfully completed.
```

Methods of Passing Parameter

There are 3 methods:

- Positional Notation:
  - The actual parameter is substituted for the formal parameter.
- Named Notation:
  - The actual parameter is associated with the formal parameter using the arrow symbol (=>) .
- Mixed Notation:
  - We can mix both the notations in the procedure call. However, the positional should precede the name notation

POSITIONAL NOTATION

```
findMin(a, b, c, d);
```

NAMED NOTATION

```
findMin(x => a, y => b, z => c, m => d);
```

MIXED NOTATION

```
findMin(a, b, c, m => d);
```

```
findMin(x => a, b, c, d);
```

CREATING FUNCTIONS

CREATING A FUNCTION

```
CREATE [OR REPLACE] FUNCTION function_name
[(parameter_name [IN [, ...])]
RETURN return_datatype
{IS | AS}
BEGIN
   < function_body >
END [function_name];
```

```
CREATE OR REPLACE FUNCTION totalCustomers
RETURN number IS
   total number(2) := 0;
BEGIN
   SELECT count(*) into total
   FROM customers;

   RETURN total;
END;
/
```

CALLING A FUNCTION

```
DECLARE
   c number(2);
BEGIN
   c := totalCustomers();
   dbms_output.put_line('Total no. of Customers: ' || c);
END;
/
```

```
Total no. of Customers: 6
```

RECURSIVE FUNCTIONS

When a subprogram calls itself, it is referred to as a recursive call and the process is known as recursion.

```
DECLARE
   num number;
   factorial number;

FUNCTION fact(x number)
RETURN number
IS
   f number;
BEGIN
   IF x=0 THEN
      f := 1;
   ELSE
      f := x * fact(x-1);
   END IF;
RETURN f;
END;

BEGIN
   num:= 6;
   factorial := fact(num);
   dbms_output.put_line(' Factorial '|| num || ' is ' || factorial);
END;
/
```

```
Factorial 6 is 720
```

DELETING

```
DROP FUNCTION function_name;
```

ADVANTAGES

- Improves Database Performance
- Provides reusability and avoids redundancy
- Maintains integrity
- Maintains security
- Saves memory

PROCEDURES vs FUNCTIONS

PROCEDURES	FUNCTIONS
May or may no return a value.	Always returns a value.
Uses IN, OUT, INOUT parameters.	Uses IN parameter only.
Returns value using ‘OUT’.	Returns value using ‘RETURN’.
Cannot be called from the function block of the program.	Can be called from the procedure block of the program.

THANK YOU