Example

The following example creates a simple procedure that displays the string 'Hello World!' on the screen when executed.

CREATE OR REPLACE PROCEDURE greetings

AS

BEGIN

dbms\_output.put\_line('Hello World!');

END;

/

When the above code is executed using the SQL prompt, it will produce the following result −

Procedure created.

Executing a Standalone Procedure

A standalone procedure can be called in two ways −

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

The above procedure named **'greetings'** can be called with the EXECUTE keyword as −

EXECUTE greetings;

The above call will display −

Hello World

PL/SQL procedure successfully completed.

The procedure can also be called from another PL/SQL block −

BEGIN

greetings;

END;

/

The above call will display −

Hello World

PL/SQL procedure successfully completed.

Deleting a Standalone Procedure

A standalone procedure is deleted with the **DROP PROCEDURE**statement. Syntax for deleting a procedure is −

DROP PROCEDURE procedure-name;

You can drop the greetings procedure by using the following statement −

DROP PROCEDURE greetings;

Parameter Modes in PL/SQL Subprograms

The following table lists out the parameter modes in PL/SQL subprograms −

|  |  |
| --- | --- |
| **S.No** | **Parameter Mode & Description** |
| 1 | **IN**  An IN parameter lets you pass a value to the subprogram. **It is a read-only parameter**. Inside the subprogram, an IN parameter acts like a constant. It cannot be assigned a value. You can pass a constant, literal, initialized variable, or expression as an IN parameter. You can also initialize it to a default value; however, in that case, it is omitted from the subprogram call. **It is the default mode of parameter passing. Parameters are passed by reference**. |
| 2 | **OUT**  An OUT parameter returns a value to the calling program. Inside the subprogram, an OUT parameter acts like a variable. You can change its value and reference the value after assigning it. **The actual parameter must be variable and it is passed by value**. |
| 3 | **IN OUT**  An **IN OUT** parameter passes an initial value to a subprogram and returns an updated value to the caller. It can be assigned a value and the value can be read.  The actual parameter corresponding to an IN OUT formal parameter must be a variable, not a constant or an expression. Formal parameter must be assigned a value. **Actual parameter is passed by value.** |

IN & OUT Mode Example 1

This program finds the minimum of two values. Here, the procedure takes two numbers using the IN mode and returns their minimum using the OUT parameters.

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;

/

When the above code is executed at the SQL prompt, it produces the following result −

Minimum of (23, 45) : 23

PL/SQL procedure successfully completed.

IN & OUT Mode Example 2

This procedure computes the square of value of a passed value. This example shows how we can use the same parameter to accept a value and then return another result.

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;

/

When the above code is executed at the SQL prompt, it produces the following result −

Square of (23): 529

PL/SQL procedure successfully completed.

Methods for Passing Parameters

Actual parameters can be passed in three ways −

* Positional notation
* Named notation
* Mixed notation

Positional Notation

In positional notation, you can call the procedure as −

findMin(a, b, c, d);

In positional notation, the first actual parameter is substituted for the first formal parameter; the second actual parameter is substituted for the second formal parameter, and so on. So, **a** is substituted for **x, b** is substituted for **y, c** is substituted for **z** and **d** is substituted for **m**.

Named Notation

In named notation, the actual parameter is associated with the formal parameter using the **arrow symbol ( => )**. The procedure call will be like the following −

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

Mixed Notation

In mixed notation, you can mix both notations in procedure call; however, the positional notation should precede the named notation.

The following call is legal −

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

However, this is not legal:

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