

# PL/SQL - FUNCTIONS

[http://www.tutorialspoint.com/plsql/plsql\\_functions.htm](http://www.tutorialspoint.com/plsql/plsql_functions.htm)

Copyright © tutorialspoint.com

A PL/SQL function is same as a procedure except that it returns a value. Therefore, all the discussions of the previous chapter are true for functions too.

## Creating a Function

A standalone function is created using the CREATE FUNCTION statement. The simplified syntax for the CREATE OR REPLACE PROCEDURE statement is as follows:

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

Where,

- *function-name* specifies the name of the function.
- [OR REPLACE] option allows modifying an existing function.
- The optional parameter list contains name, mode and types of the parameters. IN represents that value will be passed from outside and OUT represents that this parameter will be used to return a value outside of the procedure.
- The function must contain a **return** statement.
- RETURN clause specifies that data type you are going to return from the function.
- *function-body* contains the executable part.
- The AS keyword is used instead of the IS keyword for creating a standalone function.

## Example:

The following example illustrates creating and calling a standalone function. This function returns the total number of CUSTOMERS in the customers table. We will use the CUSTOMERS table, which we had created in [PL/SQL Variables](#) chapter:

```
Select * from customers;
```

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00

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

    RETURN total;
END;
```

```
/
```

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

```
Function created.
```

## Calling a Function

While creating a function, you give a definition of what the function has to do. To use a function, you will have to call that function to perform the defined task. When a program calls a function, program control is transferred to the called function.

A called function performs defined task and when its return statement is executed or when its last end statement is reached, it returns program control back to the main program.

To call a function you simply need to pass the required parameters along with function name and if function returns a value then you can store returned value. Following program calls the function `totalCustomers` from an anonymous block:

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

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

```
Total no. of Customers: 6

PL/SQL procedure successfully completed.
```

## Example:

The following is one more example which demonstrates Declaring, Defining, and Invoking a Simple PL/SQL Function that computes and returns the maximum of two values.

```
DECLARE
    a number;
    b number;
    c number;
FUNCTION findMax(x IN number, y IN number)
RETURN number
IS
    z number;
BEGIN
    IF x > y THEN
        z := x;
    ELSE
        z := y;
    END IF;

    RETURN z;
END;
BEGIN
    a := 23;
    b := 45;

    c := findMax(a, b);
    dbms_output.put_line(' Maximum of (23,45): ' || c);
END;
/
```

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

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

```
PL/SQL procedure successfully completed.
```

## PL/SQL Recursive Functions

We have seen that a program or subprogram may call another subprogram. When a subprogram calls itself, it is referred to as a recursive call and the process is known as recursion.

To illustrate the concept, let us calculate the factorial of a number. Factorial of a number  $n$  is defined as:

```
n! = n*(n-1)!
    = n*(n-1)*(n-2)!
    ...
    = n*(n-1)*(n-2)*(n-3)... 1
```

The following program calculates the factorial of a given number by calling itself recursively:

```
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;
/
```

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

```
Factorial 6 is 720
```

```
PL/SQL procedure successfully completed.
```