**What is PL/SQL Subprograms?**

Subprograms are named PL/SQL blocks that can be called with a set of parameters. PL/SQL has two types of subprograms, procedures, and functions.  
**Subprograms have:**  
■ A declarative part, with declarations of types, cursors, constants, variables, exceptions, and nested subprograms. These items are local and cease to exist when the subprogram ends.  
■ An executable part, with statements that assign values, control execution and manipulate Oracle data.  
■ An optional exception-handling part, which deals with runtime error conditions.

**Advantages of PL/SQL Subprograms**

* Subprograms let you extend the [PL/SQL language](https://intellipaat.com/blog/tutorial/oracle-plsql-tutorial/plsql-language-elements/). Procedures act like new statements. Functions act like new expressions and operators.
* Subprograms break a program down into manageable, well-defined modules.
* Subprograms promote reusability.
* Subprograms promote maintainability.
* Dummy subprograms (stubs) let you defer the definition of procedures and functions until after testing the main program.

***Want to know more about SQL? Read this extensive***[***SQL Tutorial***](https://intellipaat.com/blog/tutorial/sql-tutorial/)***!***

**PL/SQL Procedures**  
A procedure is a subprogram that performs a specific action. You specify the name of the procedure, its parameters, its local variables, and the BEGIN-END block that contains its code and handles any exceptions.  
For each parameter, you specify:  
■ It is the name.  
■ Its parameter mode (IN, OUT, or IN OUT). If you omit the mode, the default is IN. The optional NOCOPY keyword speeds up the processing of large OUT or IN OUT parameters.  
■ It is a datatype. You specify only the type, not any length or precision constraints.  
■ Optionally, it is the default value.

**PL/SQL Functions**  
A function is a subprogram that computes a value. Functions and procedures are structured alike, except that functions have a RETURN clause. Functions have a number of optional keywords, used to declare a special class of functions known as table functions. They are typically used for transforming large amounts of data in data warehousing applications.

* The AUTHID clause determines whether a stored function executes with the privileges of its owner (the default) or current user and whether its unqualified references to schema objects are resolved in the schema of the owner or current user.
* The PARALLEL\_ENABLE option declares that a stored function can be used safely in the slave sessions of parallel DML evaluations.
* The DETERMINISTIC option helps the optimizer avoid redundant function calls.
* The pragma AUTONOMOUS\_TRANSACTION instructs the PL/SQL compiler to mark a function as autonomous (independent).

Like a procedure, a function has two parts: the spec and the body. The function spec begins with the keyword FUNCTION and ends with the RETURN clause, which specifies the data type of the return value. Parameter declarations are optional. Functions that take no parameters are written without parentheses.  
The function body begins with the keyword IS (or AS) and ends with the keyword END followed by an optional function name. The function body has three parts: a declarative part, an executable part, and an optional exception-handling part.  
The declarative part contains local declarations, which are placed between the keywords IS and BEGIN. The keyword DECLARE is not used. The executable part contains statements, which are placed between the keywords BEGIN and EXCEPTION (or END). One or more RETURN statements must appear in the executable part of a function. The exception-handling part contains exception handlers, which are placed between the keywords EXCEPTION and END.

A function is called as part of an expression. For example:

IF sal\_ok(new\_sal, new\_title) THEN ...

The RETURN statement immediately ends the execution of a subprogram and returns control to the caller. A subprogram can contain several RETURN statements.  
  
**Declaring Nested PL/SQL Subprograms**  
You can declare subprograms in any PL/SQL block, a subprogram, or package. The subprograms must go at the end of the declarative section, after all, other items.  
**Example: Forward Declaration for a Nested Subprogram**

DECLARE

PROCEDURE proc1(number1 NUMBER); -- forward declaration

PROCEDURE proc2(number2 NUMBER) IS

BEGIN

proc1(number2); -- calls proc1

END;

PROCEDURE proc1(number1 NUMBER) IS

BEGIN

proc2 (number1); -- calls proc2

END;

BEGIN

NULL;

END;

/

**Passing Parameters to PL/SQL Subprograms –**  
**1. Actual Versus Formal Subprogram Parameters**  
Subprograms pass information using parameters:  
■ The variables declared in a subprogram specification and referenced in the subprogram body are formal parameters.  
■ The variables or expressions passed from the calling subprogram are actual parameters.  
A good programming practice is to use different names for actual and formal parameters.  
**Example: Formal Parameters and Actual Parameters**

DECLARE

emp\_num NUMBER(6) := 120;

bonus NUMBER(6) := 100;

merit NUMBER(4) := 50;

PROCEDURE raise\_salary (emp\_id NUMBER, amount NUMBER) IS

BEGIN

UPDATE employees SET salary = salary + amount WHERE employee\_id = emp\_id;

END raise\_salary;

BEGIN

raise\_salary(emp\_num, bonus); -- procedure call specifies actual parameters

raise\_salary(emp\_num, merit + bonus); -- expressions can be used as parameters

END;

/

**2. Using Positional, Named, or Mixed Notation for Subprogram Parameters**  
When calling a subprogram, you can write the actual parameters using either:  
■ **Positional notation.** You specify the same parameters in the same order as they are declared in the procedure.  
■ **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.  
■ **Mixed notation**. You specify the first parameters with positional notation, then switch to named notation for the last parameters.

***Come to Intellipaat’s***[***PL/SQL Community***](https://intellipaat.com/community/tag/plsql)***if you have more queries on PL/SQL!***

**Specifying Subprogram Parameter Modes**  
You use parameter modes to define the behavior of formal parameters. The three-parameter modes are IN (the default), OUT, and IN OUT.

* **Using the IN Mode**

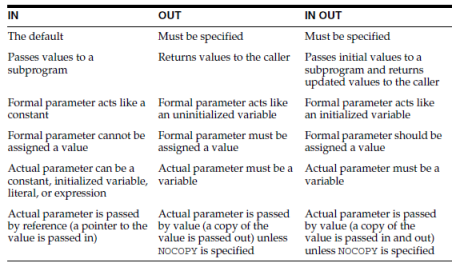
An IN parameter lets you pass values to the subprogram being called. Inside the subprogram, an IN parameter acts like a constant. It cannot be assigned a value. You can pass a constant, literal, initialized variable, or expression as an IN parameter.  
IN parameters can be initialized to default values, which are used if those parameters are omitted from the subprogram call.

* **Using the OUT Mode**

An OUT parameter returns a value to the caller of a subprogram. Inside the subprogram, an OUT parameter acts like a variable. You can change its value and reference the value after assigning it.

* **Using the IN OUT Mode**

An IN OUT parameter passes initial values to a subprogram and returns updated values to the caller. It can be assigned a value and its value can be read. Typically, an IN OUT parameter is a string buffer or numeric accumulator, that is read inside the subprogram and then updated.  
The actual parameter that corresponds to an IN OUT formal parameter must be a variable; it cannot be a constant or an expression.

[](https://cdn.intellipaat.com/wp-content/uploads/2015/09/parameter-nodes1.png)  
**Using Default Values for Subprogram Parameters**  
By initializing IN parameters to default values, you can pass different numbers of actual parameters to a subprogram, accepting the default values for any parameters you omit. You can also add new formal parameters without having to change every call to the subprogram.  
If a parameter is omitted, the default value of its corresponding formal parameter is used. You cannot skip a formal parameter by leaving out its actual parameter. To omit the first parameter and specify the second, use named notation. You cannot assign a null to an uninitialized formal parameter by leaving out its actual parameter. You must pass the null explicitly, or you can specify a default value of NULL in the declaration.

***Go for this in-depth job-oriented***[***PL/SQL Training***](https://intellipaat.com/pl-sql-online-training/)***Course now!***

**Overloading Subprogram Names**  
PL/SQL lets you overload subprogram names and type methods. You can use the same name for several different subprograms as long as their formal parameters differ in number, order, or datatype family.  
The procedures initialize different types of collections. Because the processing in these two procedures is the same, it is logical to give them the same name.  
You can place the two overloaded initialize procedures in the same block, subprogram, package, or object type. PL/SQL determines which procedure to call by checking their formal parameters. The version of initializing that PL/SQL uses depends on whether you call the procedure with a DateTabTyp or NumTabTyp parameter.  
**Example: Overloading a Subprogram Name**

DECLARE

TYPE DateTabTyp IS TABLE OF DATE INDEX BY PLS\_INTEGER;

TYPE NumTabTyp IS TABLE OF NUMBER INDEX BY PLS\_INTEGER;

hiredate\_tab DateTabTyp;

sal\_tab NumTabTyp;

PROCEDURE initialize (tab OUT DateTabTyp, n INTEGER) IS

BEGIN

FOR i IN 1..n LOOP

tab(i) := SYSDATE;

END LOOP;

END initialize;

PROCEDURE initialize (tab OUT NumTabTyp, n INTEGER) IS

BEGIN

FOR i IN 1..n LOOP

tab(i) := 0.0;

END LOOP;

END initialize;

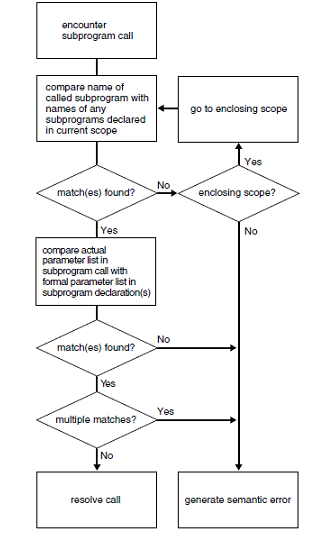
BEGIN

initialize(hiredate\_tab, 50); -- calls first (DateTabTyp) version

initialize(sal\_tab, 100); -- calls second (NumTabTyp) version

END;

/

[](https://cdn.intellipaat.com/wp-content/uploads/2015/09/How-the-PLSQL-Compiler-Resolves-Calls.png)

**Invoker’s Rights versus Definer’s Rights (AUTHID Clause)**  
By default, stored procedures and SQL methods execute with the privileges of their owner, not their current user. Such definer’s rights subprograms are bound to the schema in which they reside, allowing you to refer to objects in the same schema without qualifying their names.  
**Example: Specifying Invoker’s Rights With a Procedure**

CREATE OR REPLACE PROCEDURE create\_dept (

v\_deptno NUMBER,

v\_dname VARCHAR2,

v\_mgr NUMBER,

v\_loc NUMBER)

AUTHID CURRENT\_USER AS

BEGIN

INSERT INTO departments VALUES (v\_deptno, v\_dname, v\_mgr, v\_loc);

END;

/

CALL create\_dept(44, 'Information Technology', 200, 1700);

***Check out the top***[***PL/SQL Interview Questions***](https://intellipaat.com/blog/interview-question/pl-sql-interview-questions/)***to learn what is expected from PL/SQL professionals!***

**Advantages of Invoker’s Rights**  
Invoker’s rights subprograms let you reuse code and centralize application logic. They are especially useful in applications that store data using identical tables in different schemas. All the schemas in one instance can call procedures owned by a central schema. You can even have schemas in different instances call centralized procedures using a database link.  
You can restrict access to sensitive data by calling from an invoker’s rights subprogram to a definer’s rights subprogram that queries or updates the table containing the sensitive data. Although multiple users can call the invoker’s rights subprogram, they do not have direct access to sensitive data.

**Recursion with PL/SQL**  
Recursion is a powerful technique for simplifying the design of algorithms. Basically, recursion means self-reference. In a recursive mathematical sequence, each term is derived by applying a formula to preceding terms. The Fibonacci sequence (0, 1, 1, 2, 3, 5, 8, 13, 21, …), is an example. Each term in the sequence (after the second) is the sum of the two terms that immediately precede it.  
In a recursive definition, something is defined as simpler versions of itself. Consider the definition of n factorial (n!), the product of all integers from 1 to n:

n! = n \* (n - 1)!

**A recursive subprogram** is one that calls itself. Each recursive call creates a new instance of any items declared in the subprogram, including parameters, variables, cursors, and exceptions.