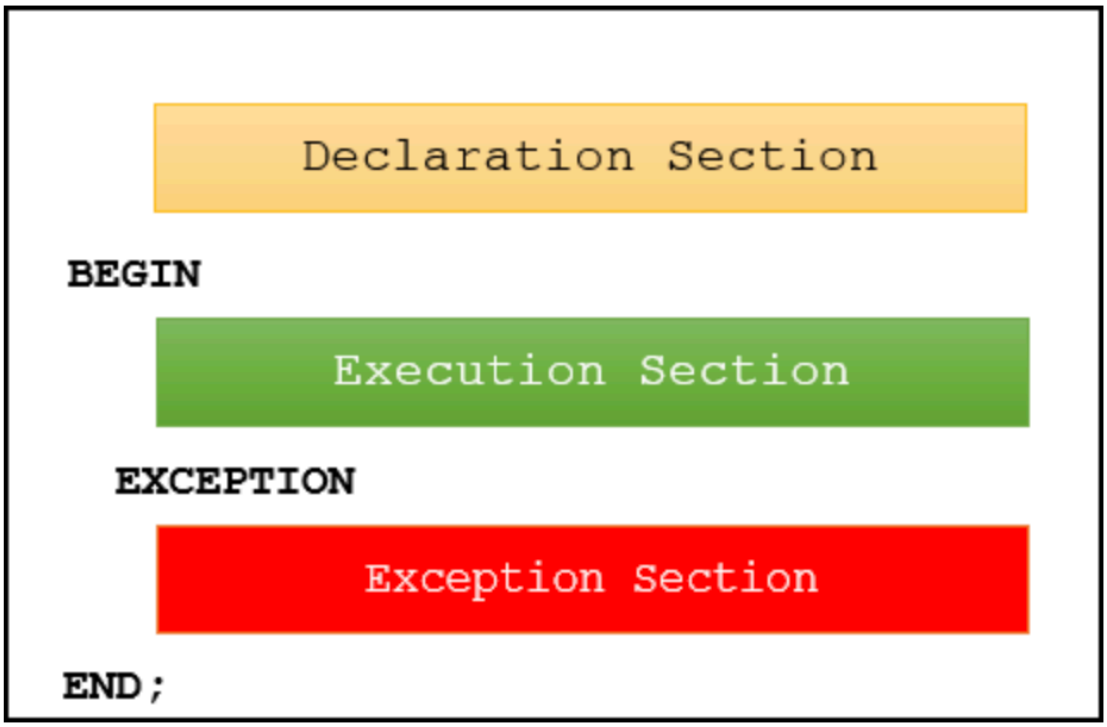
PL/SQL is a block-structured language whose code is organized into blocks. A PL/SQL block consists of three sections: declaration, executable, and exception-handling sections. In a block, the executable section is mandatory while the declaration and exception-handling sections are optional.

A PL/SQL block has a name. Functions or Procedures is an example of a named block. A block without a name is an anonymous block. However, PL/SQL anonymous blocks can be useful for testing purposes.



1) Declaration section

A PL/SQL block has a declaration section where you declare variables, allocate memory for cursors, and define data types.

2) Executable section

A PL/SQL block has an executable section. An executable section starts with the keyword `BEGIN` and ends with the keyword `END`. The executable section must have a least one executable statement, even if it is the `NULL` statement which does nothing.

3) Exception-handling section

A PL/SQL block has an exception-handling section that starts with the keyword `EXCEPTION`. The exception-handling section is where you catch and handle exceptions raised by the code in the execution section.

Note a block itself is an executable statement, therefore you can nest a block within other blocks.

Types of PL/SQL block

PL/SQL blocks are of mainly two types.

1. Anonymous blocks
2. Named Blocks

Anonymous blocks:

Anonymous blocks are PL/SQL blocks which do not have any names assigned to them. They need to be created and used in the same session because they will not be stored in the server as database objects.

Since they need not store in the database, they need no compilation steps. They are written and executed directly, and compilation and execution happen in a single process.

Below are few more characteristics of Anonymous blocks.

- These blocks don't have any reference name specified for them.
- These blocks start with the keyword 'DECLARE' or 'BEGIN'.
- Since these blocks do not have any reference name, these cannot be stored for later purpose. They shall be created and executed in the same session.
- They can call the other named blocks, but call to anonymous block is not possible as it is not having any reference.
- It can have nested block in it which can be named or anonymous. It can also be nested in any blocks.
- These blocks can have all three sections of the block, in which execution section is mandatory, the other two sections are optional.

Named blocks:

Named blocks have a specific and unique name for them. They are stored as the database objects in the server. Since they are available as database objects, they can be referred to or used as long as it is present on the server. The compilation process for named blocks happens separately while creating them as a database objects.

Below are few more characteristics of Named blocks.

- These blocks can be called from other blocks.
- The block structure is same as an anonymous block, except it will never start with the keyword 'DECLARE'. Instead, it will start with the keyword 'CREATE' which instruct the compiler to create it as a database object.
- These blocks can be nested within other blocks. It can also contain nested blocks.
- Named blocks are basically of two types:

1. Procedure
2. Function

If -else

```
DECLARE
a NUMBER :=10;
BEGIN
dbms_output.put_line(‘Program started.' );
IF( a > 100 ) THEN
dbms_output.put_line('a is greater than 100');
END IF;
dbms_output.put_line(‘Program completed.');
END;
/
```

Code Explanation:

- Code line 2: Declaring the variable 'a' as 'NUMBER' data type and initializing it with value '10'.
- Code line 4: Printing the statement "Program started".
- Code line 5: Checking the condition, whether variable 'a' is greater than '100.'
- Code line 6: If 'a' is greater than '100', then "a is greater than 100" will be printed. If 'a' is lesser than or equal to 100, then condition fails, so the above printing statement ignored.
- Code line 8: Printing the statement "Program completed".

Loops

```
DECLARE
a NUMBER:=1;
BEGIN
dbms_output.put_line('Program started.');
LOOP
dbms_output.put_line(a);
```

```
a:=a+1;
EXIT WHEN a>5;
END LOOP;
dbms_output.put_line('Program completed');
END:
/
```

Code Explanation:

- Code line 2: Declaring the variable 'a' as 'NUMBER' data type and initializing it with value '1'.
- Code line 4: Printing the statement "Program started".
- Code line 5: Keyword 'LOOP' marks the beginning of the loop.
- Code line 6: Prints the value of 'a'.
- Code line 7: Increments the value of 'a' by +1.
- Code line 8: Checks whether the value of 'a' is greater than 5.
- Code line 9: Keyword 'END LOOP' marks the end of execution block.
- The code from line 6 to line 8 will continue to execute till 'a' reaches the value 6, as the condition will return TRUE, and the control will EXIT from the loop.
- Code line 10: Printing the statement "Program completed"

While loop

```
DECLARE
a NUMBER :=1;
BEGIN
dbms_output.put_line('Program started');
WHILE (a <= 5)
LOOP
dbms_output.put_line(a);
a:=a+1;
END LOOP;
dbms_output.put_line(‘Program completed' );
END:
/
```

Code Explanation:

- **Code line 2**: Declaring the variable 'a' as 'NUMBER' data type and initializing it with value '1'.
- **Code line 4**: Printing the statement "Program started".
- **Code line 5:** Keyword 'WHILE' marks the beginning of the loop, and it also checks whether the value of 'a' is less than or equal to 5
- **Code line 7:** Prints the value of 'a'.

- **Code line 8:** Increments the value of 'a' by +1.
- **Code line 9:** Keyword 'END LOOP' marks the end of execution block.
- The code from line 7 and line 8 will continue to execute till 'a' reaches the value 6, as the condition will return TRUE, and the control will EXIT from the loop.
- **Code line 10:** Printing the statement "Program completed"

Parameter:

The parameter is variable or placeholder of any valid PL/SQL datatype through which the PL/SQL subprogram exchange the values with the main code. This parameter allows to give input to the subprograms and to extract from these subprograms.

- These parameters should be defined along with the subprograms at the time of creation.
- These parameters are included n the calling statement of these subprograms to interact the values with the subprograms.
- The datatype of the parameter in the subprogram and the calling statement should be same.
- The size of the datatype should not mention at the time of parameter declaration, as the size is dynamic for this type.

Based on their purpose parameters are classified as

1. IN Parameter
2. OUT Parameter
3. IN OUT Parameter

IN Parameter:

- This parameter is used for giving input to the subprograms.
- It is a read-only variable inside the subprograms. Their values cannot be changed inside the subprogram.
- In the calling statement, these parameters can be a variable or a literal value or an expression, for example, it could be the arithmetic expression like '5*8' or 'a/b' where 'a' and 'b' are variables.
- By default, the parameters are of IN type.

OUT Parameter:

- This parameter is used for getting output from the subprograms.
- It is a read-write variable inside the subprograms. Their values can be changed inside the subprograms.
- In the calling statement, these parameters should always be a variable to hold the value from the current subprograms.

IN OUT Parameter:

- This parameter is used for both giving input and for getting output from the subprograms.
- It is a read-write variable inside the subprograms. Their values can be changed inside the subprograms.
- In the calling statement, these parameters should always be a variable to hold the value from the subprograms.

These parameter type should be mentioned at the time of creating the subprograms.

RETURN

RETURN is the keyword that instructs the compiler to switch the control from the subprogram to the calling statement. In subprogram RETURN simply means that the control needs to exit from the subprogram. Once the controller finds RETURN keyword in the subprogram, the code after this will be skipped.

Normally, parent or main block will call the subprograms, and then the control will shift from those parent block to the called subprograms. RETURN in the subprogram will return the control back to their parent block. In the case of functions RETURN statement also returns the value. The datatype of this value is always mentioned at the time of function declaration. The datatype can be of any valid PL/SQL data type.

What is Procedure in PL/SQL?

A **Procedure** in PL/SQL is a subprogram unit that consists of a group of PL/SQL statements that can be called by name. Each procedure in PL/SQL has its own unique name by which it can be referred to and called. This subprogram unit in the Oracle database is stored as a database object.

Note: Subprogram is nothing but a procedure, and it needs to be created manually as per the requirement. Once created they will be stored as database objects.

Below are the characteristics of Procedure subprogram unit in PL/SQL:

- Procedures are standalone blocks of a program that can be stored in the database.
- Call to these PLSQL procedures can be made by referring to their name, to execute the PL/SQL statements.
- It is mainly used to execute a process in PL/SQL.
- It can have nested blocks, or it can be defined and nested inside the other blocks or packages.
- It contains declaration part (optional), execution part, exception handling part (optional).
- The values can be passed into Oracle procedure or fetched from the procedure through parameters.
- These parameters should be included in the calling statement.
- A Procedure in SQL can have a RETURN statement to return the control to the calling block, but it cannot return any values through the RETURN statement.
- Procedures cannot be called directly from SELECT statements. They can be called from another block or through EXEC keyword.

Syntax:

```
CREATE OR REPLACE PROCEDURE
<procedure_name>
     (
```

```
     <parameterl IN/OUT <datatype>
     ..
     .
     )
[ IS | AS ]
     <declaration_part>
BEGIN
     <execution part>
EXCEPTION
     <exception handling part>
END;
```

- CREATE PROCEDURE instructs the compiler to create new procedure in Oracle. Keyword 'OR REPLACE' instructs the compile to replace the existing procedure (if any) with the current one.
- Procedure name should be unique.
- Keyword 'IS' will be used, when the stored procedure in Oracle is nested into some other blocks. If the procedure is standalone then 'AS' will be used. Other than this coding standard, both have the same meaning.

Example1: Creating Procedure and calling it using EXEC

In this example, we are going to create an Oracle procedure that takes the name as input and prints the welcome message as output. We are going to use EXEC command to call procedure.

```
CREATE OR REPLACE PROCEDURE welcome_msg(p_name IN VARCHAR2)
IS
BEGIN
dbms_output.put_line ('Welcome '|| p_name);
END;
/
```

```
EXEC welcome_msg (‘abc’);
```

Code Explanation:

- **Code line 1**: Creating the procedure with name 'welcome_msg' and with one parameter 'p_name' of 'IN' type.
- **Code line 4**: Printing the welcome message by concatenating the input name.
- Procedure is compiled successfully.
- **Code line 7**: Calling the procedure using EXEC command with the parameter 'Guru99'. Procedure is executed, and the message is printed out as "Welcome Guru99".

What is Function?

Functions is a standalone PL/SQL subprogram. Like PL/SQL procedure, functions have a unique name by which it can be referred. These are stored as PL/SQL database objects. Below are some of the characteristics of functions.

- Functions are a standalone block that is mainly used for calculation purpose.
- Function use RETURN keyword to return the value, and the datatype of this is defined at the time of creation.
- A Function should either return a value or raise the exception, i.e. return is mandatory in functions.
- Function with no DML statements can be directly called in SELECT query whereas the function with DML operation can only be called from other PL/SQL blocks.
- It can have nested blocks, or it can be defined and nested inside the other blocks or packages.
- It contains declaration part (optional), execution part, exception handling part (optional).
- The values can be passed into the function or fetched from the procedure through the parameters.
- These parameters should be included in the calling statement.
- A PLSQL function can also return the value through OUT parameters other than using RETURN.
- Since it will always return the value, in calling statement it always accompanies with assignment operator to populate the variables.

Syntax:

```
CREATE OR REPLACE FUNCTION
<procedure_name>
        (
        <parameter1 IN/OUT <datatype>
        ..
        .
        )
        RETURN <datatype>
[ IS | AS ]
        <declaration_part>
BEGIN
        <execution part>
EXCEPTION
        <exception handling part>
END;
```

Syntax

```
CREATE OR REPLACE FUNCTION
<procedure_name>
(
<parameterl IN/OUT <datatype>
```

```
)
RETURN <datatype>
[ IS | AS ]
<declaration_part>
BEGIN
<execution part>
EXCEPTION
<exception handling part>
END;
```

- CREATE FUNCTION instructs the compiler to create a new function. Keyword 'OR REPLACE' instructs the compiler to replace the existing function (if any) with the current one.
- The Function name should be unique.
- RETURN datatype should be mentioned.
- Keyword 'IS' will be used, when the procedure is nested into some other blocks. If the procedure is standalone then 'AS' will be used. Other than this coding standard, both have the same meaning.

Example1: Creating Function and calling it using Anonymous Block

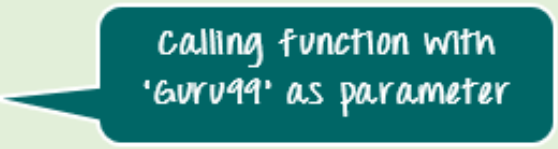
In this program, we are going to create a function that takes the name as input and returns the welcome message as output. We are going to use anonymous block and select statement to call the function.

```
1.  CREATE OR REPLACE FUNCTION welcome_msg_func ( p_name IN VARCHAR2)
2.  RETURN VARCHAR2
3.  IS
4.  BEGIN
5.  RETURN ('Welcome '|| p_name);
6.  END;
7.  /
```

Output:

Function created


```
8.  DECLARE
9.  lv_msg VARCHAR2(250);
10. BEGIN
11. lv_msg := welcome_msg_func ('Guru99');
12. dbms_output.put_line(lv_msg);
13. END;
```



Output:

Welcome Guru99

```
14. SELECT welcome_msg_func('Guru99') FROM DUAL;
```

Output:

Welcome Guru99

```
CREATE OR REPLACE FUNCTION welcome_msg_func ( p_name IN VARCHAR2)
RETURN VARCHAR2
IS
BEGIN
RETURN ('Welcome '|| p_name);
END;
/
```

```
DECLARE
lv_msg VARCHAR2(250);
BEGIN
lv_msg := welcome_msg_func ('ABC');
dbms_output.put_line(lv_msg);
END;
```

```
SELECT welcome_msg_func('abc') FROM DUAL;
```

Code Explanation:

- **Code line 1**: Creating the Oracle function with name 'welcome_msg_func' and with one parameter 'p_name' of 'IN' type.
- **Code line 2**: declaring the return type as VARCHAR2
- **Code line 5**: Returning the concatenated value 'Welcome' and the parameter value.
- **Code line 8**: Anonymous block to call the above function.
- **Code line 9**: Declaring the variable with datatype same as the return datatype of the function.
- **Code line 11**: Calling the function and populating the return value to the variable 'lv_msg'.
- **Code line 12**: Printing the variable value. The output you will get here is "Welcome Guru99"
- **Code line 14**: Calling the same function through SELECT statement. The return value is directed to the standard output directly.

Similarities between Procedure and Function

- Both can be called from other PL/SQL blocks.
- If the exception raised in the subprogram is not handled in the subprogram exception handling section, then it will propagate to the calling block.
- Both can have as many parameters as required.
- Both are treated as database objects in PL/SQL.

Procedure Vs. Function: Key Differences

Procedure	Function
• Used mainly to a execute certain process	• Used mainly to perform some calculation
• Cannot call in SELECT statement	• A Function that contains no DML statements can be called in SELECT statement
• Use OUT parameter to return the value	• Use RETURN to return the value
• It is not mandatory to return the value	• It is mandatory to return the value
• RETURN will simply exit the control from subprogram.	• RETURN will exit the control from subprogram and also returns the value

• Return datatype will not be specified at the time of creation	• Return datatype is mandatory at the time of creation

Built-in Functions in PL/SQL

PL/SQL contains various built-in functions to work with strings and date datatype. Here we are going to see the commonly used functions and their usage.

Conversion Functions

These built-in functions are used to convert one datatype to another datatype.

Function Name	Usage	Example
TO_CHAR	Converts the other datatype to character datatype	TO_CHAR(123);
TO_DATE (string, format)	Converts the given string to date. The string should match with the format.	TO_DATE('2015-JAN-15', 'YYYY-MON-DD'); Output: 1/15/2015

TO_NUMBER (text, format)	Converts the text to number type of the given format. Informat '9' denotes the number of digits	Select TO_NUMBER('1234','9999') from dual; Output: 1234 Select TO_NUMBER('1,234.45','9,999.99') from dual; Output: 1234

String Functions

These are the functions that are used on the character datatype.

Function Name	Usage	Example
INSTR(text, string, start, occurance)	Gives the position of particular text in the given string. • text – Main string • string – text that need to be searched • start – starting position of the search (optional) • accordance – occurrence of the searched string (optional)	Select INSTR('AEROPLANE','E',2,1) from dual **Output**: 2 Select INSTR('AEROPLANE','E',2,2) from dual Output: 9 (2_{nd} occurance of E)
SUBSTR (text, start, length)	Gives the substring value of the main string. • text – main string • start – starting position • length – length to be sub stringed	select substr('aeroplane',1,7) from dual **Output**: aeropla

UPPER (text)	Returns the uppercase of the provided text	Select upper('guru99') from dual; **Output**: GURU99
LOWER (text)	Returns the lowercase of the provided text	Select lower ('AerOpLane') from dual; **Output**: aeroplane
INITCAP (text)	Returns the given text with the starting letter in upper case.	Select ('guru99') from dual **Output**: Guru99 Select ('my story') from dual **Output**: My Story
LENGTH (text)	Returns the length of the given string	Select LENGTH ('guru99') from dual; **Output**: 6
LPAD (text, length, pad_char)	Pads the string in the left side for the given length (total string) with the given character	Select LPAD('guru99', 10, '$') from dual; **Output**: $$$ $guru99
RPAD (text, length, pad_char)	Pads the string in the right side for the given length (total string) with the given character	Select RPAD('guru99',10,'-') from dual **Output**: guru99----

LTRIM (text)	Trims the leading white space from the text	Select LTRIM(' Guru99') from dual; **Output**: Guru99
RTRIM (text)	Trims the trailing white space from the text	Select RTRIM('Guru99 ') from dual; **Output**; Guru99

Date Functions

These are functions that are used for manipulating with dates.

Function Name	Usage	Example
ADD_MONTHS (date, no.of months)	Adds the given months to the date	ADD_MONTH('2015-01-01',5); **Output**: 05/01/2015
SYSDATE	Returns the current date and time of the server	Select SYSDATE from dual; **Output**: 10/4/2015 2:11:43 PM
TRUNC	Round of the date variable to the lower possible value	select sysdate, TRUNC(sysdate) from dual; **Output**: 10/4/2015 2:12:39 PM 10/4/2015

ROUND	Rounds the date to the nearest limit either higher or lower	Select sysdate, ROUND(sysdate) from dual **Output**: 10/4/2015 2:14:34 PM 10/5/2015
MONTHS_BETWEEN	Returns the number of months between two dates	Select MONTHS_BETWEEN (sysdate+60, sysdate) from dual **Output**: 2