PL/SQL (**procedural language extension to Structured Query Language**)

- PL/SQL is a combination of SQL along with the procedural features of programming languages.
- It was developed by Oracle Corporation in the early 90's to enhance the capabilities of SQL.

Features of PL/SQL

- PL/SQL has the following features –

- PL/SQL is tightly integrated with SQL.
- It offers extensive error checking.
- It offers numerous data types.
- It offers a variety of programming structures.
- It supports structured programming through functions and procedures.
- It supports object-oriented programming.
- It supports the development of web applications and server pages.
- Applications written in PL/SQL are fully portable.
- PL/SQL provides high security level.
- PL/SQL provides access to predefined SQL packages.
- PL/SQL provides support for Object-Oriented Programming.
- PL/SQL provides support for developing Web Applications and Server Pages.

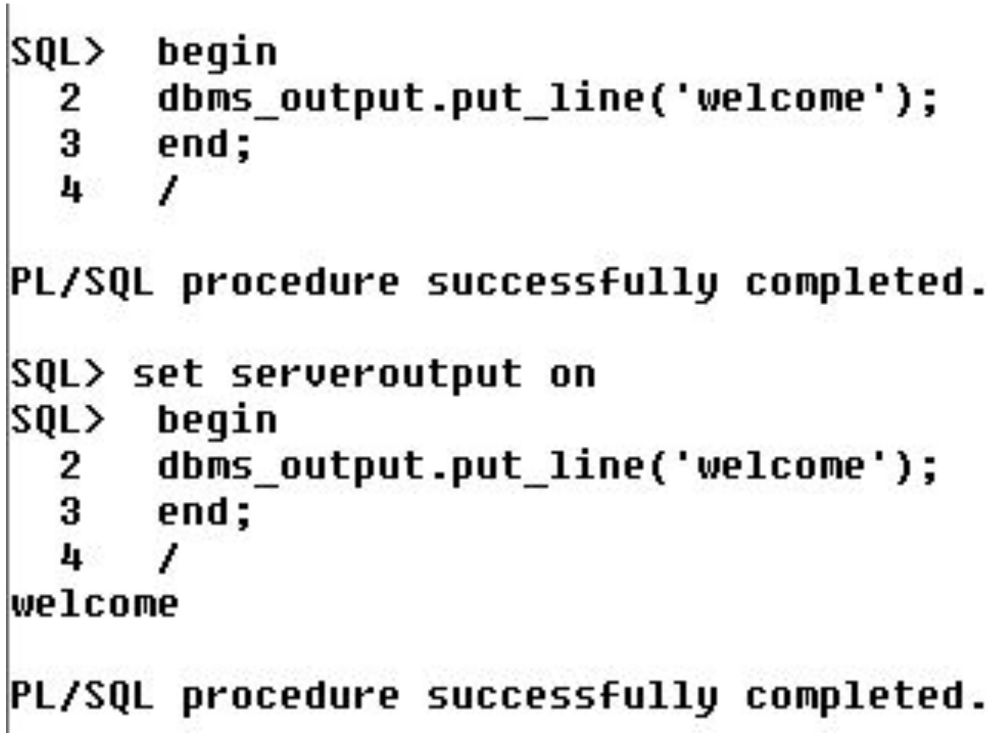
- PL/SQL is not case sensitive so you are free to use lower case letters or upper case letters except within string and character literals.

BASIC SYNTAX

S.No	Sections & Description
1	**Declarations** ● This section starts with the keyword **DECLARE**. ● It is an optional section and defines all variables, cursors, subprograms, and other elements to be used in the program.
2	**Executable Commands** ● This section is enclosed between the keywords **BEGIN** and **END** and it is a mandatory section. ● It consists of the executable PL/SQL statements of the program. ● It should have at least one executable line of code, ● which may be just a **NULL command** to indicate that nothing should be executed.
3	**Exception Handling** This section starts with the keyword **EXCEPTION**. This optional section contains **exception(s)** that handle errors in the program.

- Every PL/SQL statement ends with a semicolon (;).
- PL/SQL blocks can be nested within other PL/SQL blocks using BEGIN and END.

```
DECLARE
   <declarations section>
BEGIN
   <executable command(s)>
EXCEPTION
   <exception handling>
END;
```



```
SQL>  begin
  2   dbms_output.put_line('welcome');
  3   end;
  4   /

PL/SQL procedure successfully completed.

SQL> set serveroutput on
SQL>  begin
  2   dbms_output.put_line('welcome');
  3   end;
  4   /
welcome

PL/SQL procedure successfully completed.
```

Variable

A variable is a meaningful name which facilitates a programmer to store data temporarily during the execution of code.

It helps you to manipulate data in PL/SQL programs.

It is nothing except a name given to a storage area.

Each variable in the PL/SQL has a specific data type which defines the size and layout of the variable's memory.

A variable should not exceed 30 characters. Its letter optionally followed by more letters, dollar signs, numerals, underscore etc.

It needs to declare the variable first in the declaration section of a PL/SQL block before using it.

Variable Declaration in PL/SQL

- PL/SQL variables must be declared in the declaration section or in a package as a global variable.
- PL/SQL allocates memory for the variable's value and the storage location is identified by the variable name.

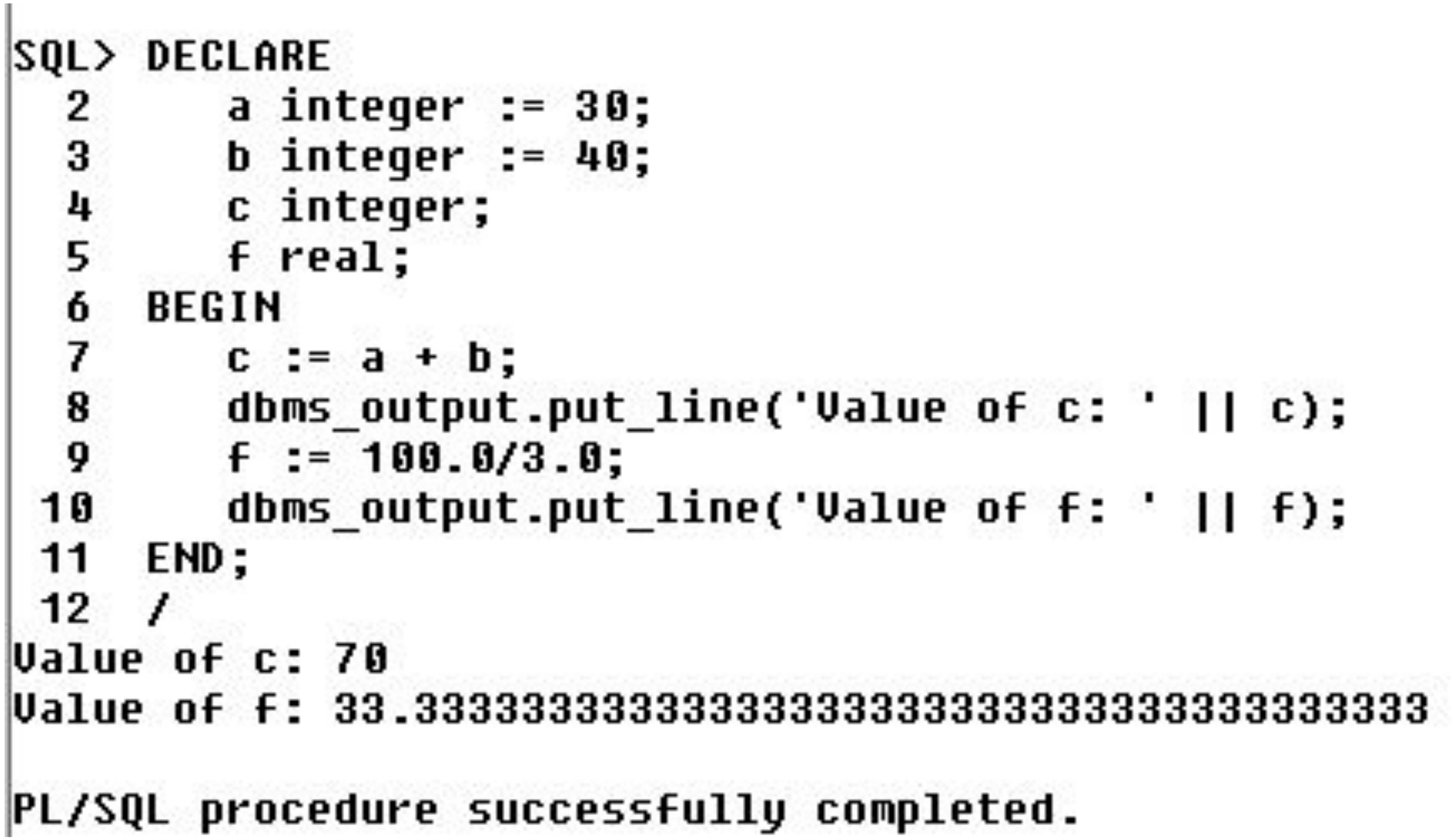
```
variable_name [CONSTANT] datatype [NOT NULL] [:= | DEFAULT
initial_value]
```

Example:

```
sales number(10, 2);
name varchar2(25);
address varchar2(100);
```

Initializing Variables in PL/SQL

- default value NULL.
- initialize a variable with a value other than the NULL value, you can do so during the declaration, using either of the following −
  - The DEFAULT keyword
  - The assignment operator
- For example −
  - counter number:= 0;
  - greetings varchar2(20) DEFAULT 'Have a Good Day';

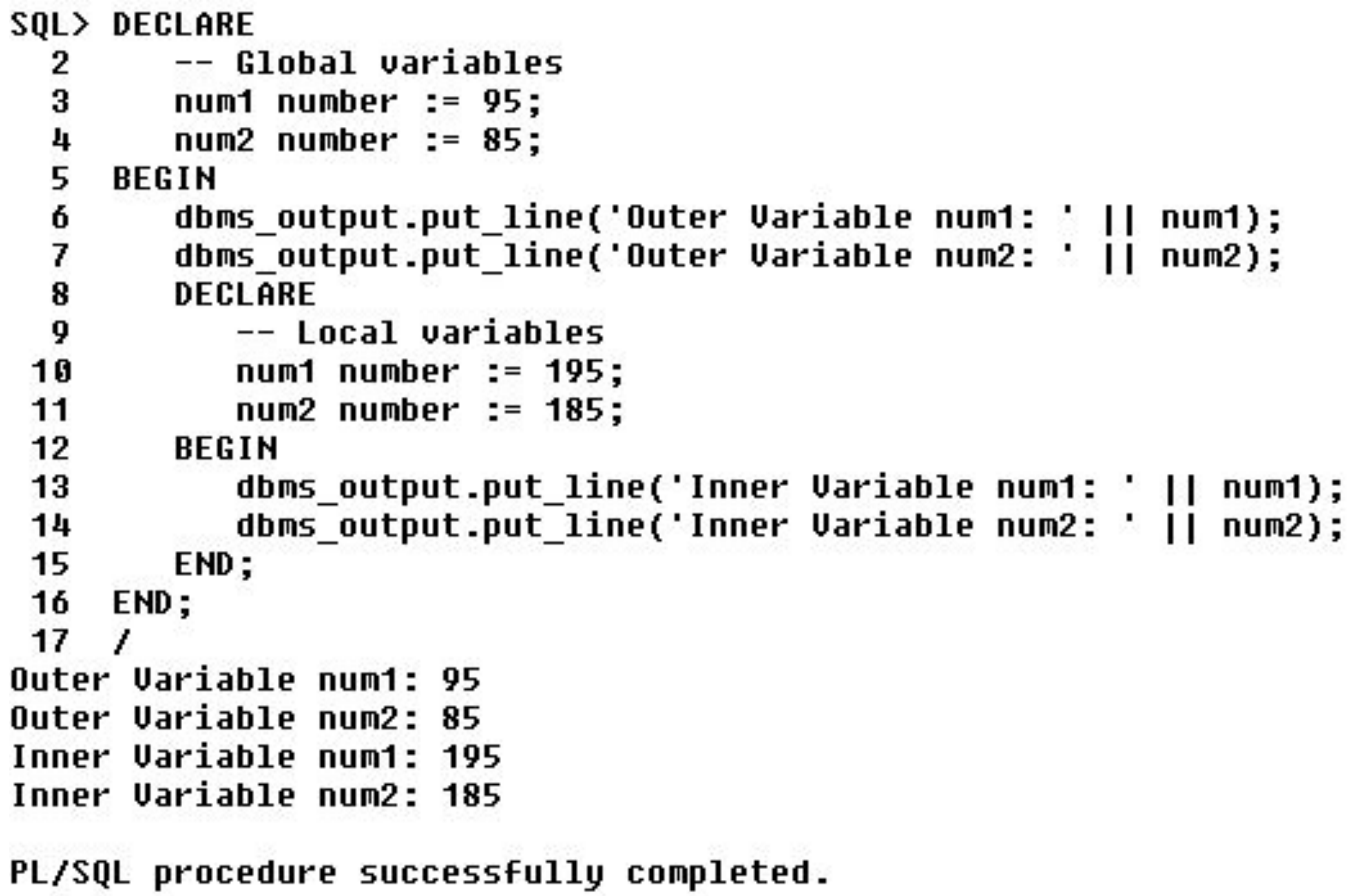


```
SQL> DECLARE
  2      a integer := 30;
  3      b integer := 40;
  4      c integer;
  5      f real;
  6   BEGIN
  7      c := a + b;
  8      dbms_output.put_line('Value of c: ' || c);
  9      f := 100.0/3.0;
 10      dbms_output.put_line('Value of f: ' || f);
 11   END;
 12   /
Value of c: 70
Value of f: 33.33333333333333333333333333333333333333

PL/SQL procedure successfully completed.
```

Variable Scope in PL/SQL

- PL/SQL allows the nesting of blocks,
  - i.e., each program block may contain another inner block.
  - If a variable is declared within an inner block, it is not accessible to the outer block.
- However, if a variable is declared and accessible to an outer block, it is also accessible to all nested inner blocks.
- There are two types of variable scope
  - Local variables − Variables declared in an inner block and NOT accessible to outer blocks.
  - Global variables − Variables declared in the outermost block or a package.



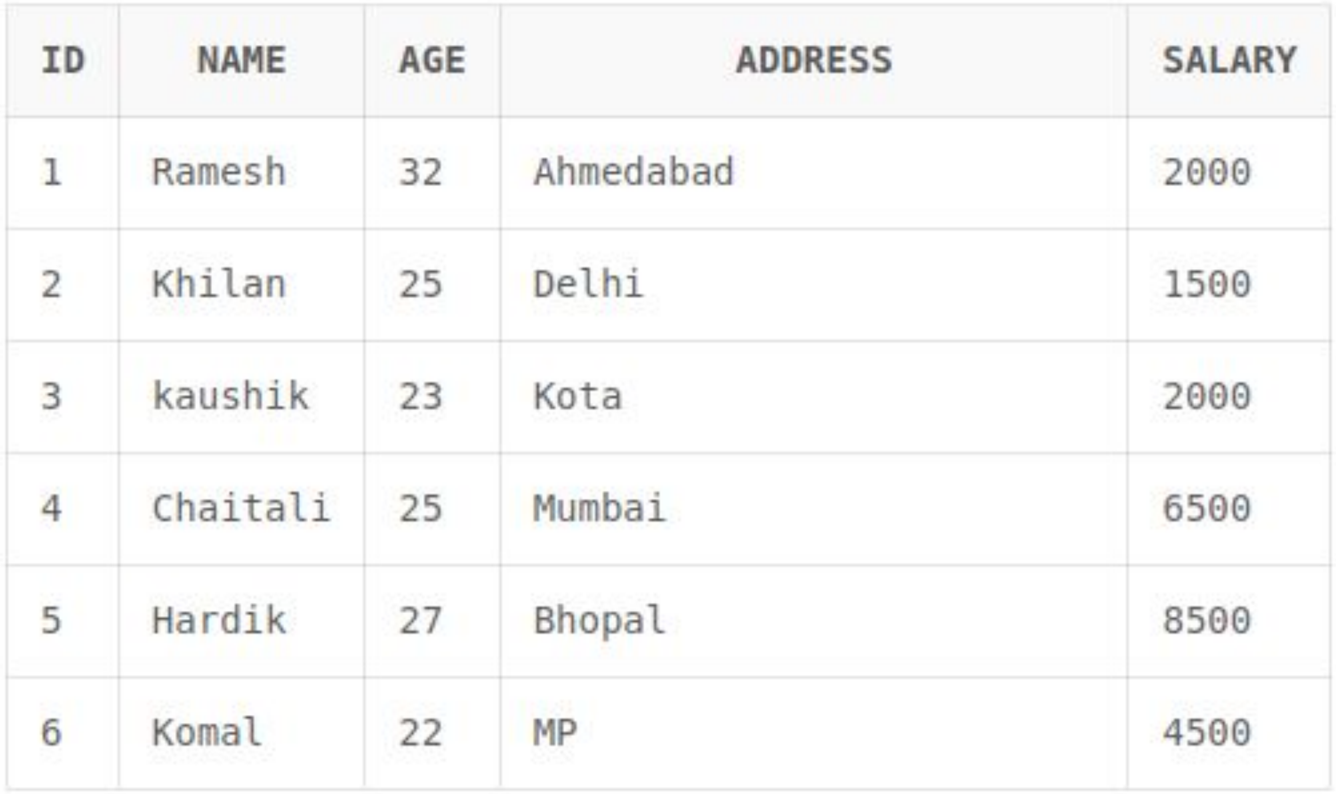
```
SQL> DECLARE
  2     -- Global variables
  3     num1 number := 95;
  4     num2 number := 85;
  5  BEGIN
  6     dbms_output.put_line('Outer Variable num1: ' || num1);
  7     dbms_output.put_line('Outer Variable num2: ' || num2);
  8     DECLARE
  9        -- Local variables
 10        num1 number := 195;
 11        num2 number := 185;
 12     BEGIN
 13        dbms_output.put_line('Inner Variable num1: ' || num1);
 14        dbms_output.put_line('Inner Variable num2: ' || num2);
 15     END;
 16  END;
 17  /
Outer Variable num1: 95
Outer Variable num2: 85
Inner Variable num1: 195
Inner Variable num2: 185

PL/SQL procedure successfully completed.
```

User Input in sqlplus

```
declare
x number;
begin
x:=&x;
dbms_output.put_line(x);
end;
/
```

Value from Table



ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000
2	Khilan	25	Delhi	1500
3	kaushik	23	Kota	2000
4	Chaitali	25	Mumbai	6500
5	Hardik	27	Bhopal	8500
6	Komal	22	MP	4500

- declare a PL/SQL variable to hold the column values,
    - it must be of correct data types and precision, otherwise error will occur on execution.
- Rather than hard coding the data type and precision of a variable.
- PL/SQL provides the facility to declare a variable without having to specify a particular data type using
    - %TYPE and
    - %ROWTYPE attributes.
- These two attributes allow us to specify a variable and have that variable data type be defined by a table/view column or a PL/SQL package variable.

%TYPE

The %TYPE attribute is used to declare variables according to the already declared variable or database column.

It is used when you are declaring an individual variable, not a record.

The data type and precision of the variable declared using %TYPE attribute is the same as that of the column that is referred from a given table.

This is particularly useful when declaring variables that will hold database values.

When using the %TYPE keyword, the name of the columns and the table to which the variable will correspond must be known to the user.

These are then prefixed with the variable name.

If some previously declared variable is referred then prefix that variable name to the %TYPE attribute.

The syntax for declaring a variable with %TYPE is:

```
<var_name> <tab_name>.<column_name>%TYPE;
```

```
c_id customers.id%type := 1;
```

```
DECLARE
   c_id customers.id%type := 1;
   c_name  customers.name%type;
   c_addr customers.address%type;
   c_sal  customers.salary%type;
BEGIN
   SELECT name, address, salary INTO c_name, c_addr, c_sal
   FROM customers  WHERE id = c_id;
   dbms_output.put_line ('Customer ' ||c_name || ' from ' || c_addr || ' earns ' || c_sal);
END;
/
```

```
Customer Ramesh from
Ahmedabad earns 2000

PL/SQL procedure
completed
successfully
```

%ROWTYPE

- **%ROWTYPE:**

The %ROWTYPE attribute is used to declare a record type that represents a row in a table.

The record can store an entire row or some specific data selected from the table.

A column in a row and corresponding fields in a record have the same name and data types.

The syntax for declaring a variable with %ROWTYPE is:

```
<var_name> <tab_name>.ROW%TYPE;
```

CONSTANT

```
PI CONSTANT NUMBER := 3.141592654;
DECLARE
   -- constant declaration
   pi constant number := 3.141592654;
   -- other declarations
   radius number(5,2);
   dia number(5,2);
   circumference number(7, 2);
   area number (10, 2);
BEGIN
   -- processing
   radius := 9.5;
   dia := radius * 2;
   circumference := 2.0 * pi * radius;
   area := pi * radius * radius;
   -- output
   dbms_output.put_line('Radius: ' || radius);
   dbms_output.put_line('Diameter: ' || dia);
   dbms_output.put_line('Circumference: ' ||
circumference);
   dbms_output.put_line('Area: ' || area);
END;
/
```

```
Radius: 9.5
Diameter: 19
Circumference: 59.69
Area: 283.53

  Pl/SQL procedure successfully
  completed.
```

PL/SQL If

1. IF condition
2. **THEN**
3. Statement: {It **is** executed **when** condition **is true**}
4. **END** IF;

1. IF condition
2. **THEN**
3. {...statements **to execute when** condition **is TRUE**...}
4. **ELSE**
5. {...statements **to execute when** condition **is FALSE**...}
6. **END** IF;

```
IF condition1
THEN
   {...statements to execute when condition1 is TRUE...}
ELSIF condition2
THEN
   {...statements to execute when condition2 is TRUE...}
END IF;
```

```
IF condition1
THEN
   {...statements to execute when condition1 is TRUE...}
ELSIF condition2
THEN
   {...statements to execute when condition2 is TRUE...}
ELSE
   {...statements to execute when both condition1 and condition2 are FALSE...}
END IF;
```

```
DECLARE
  a number(3) := 500;
BEGIN
  -- check the boolean condition using if statement
  IF( a < 20 ) THEN
    -- if condition is true then print the following
    dbms_output.put_line('a is less than 20 ' );
  ELSE
    dbms_output.put_line('a is not less than 20 ' );
  END IF;
  dbms_output.put_line('value of a is : ' || a);
END;
```

```
Statement processed.
a is not less than 20
value of a is : 500
```

While Loop

```
WHILE <condition>
 LOOP statements;
END LOOP;
```

```
DECLARE
i number := 1;
BEGIN
WHILE i <= 10 LOOP
DBMS_OUTPUT.PUT_LINE(i);
i := i+1;
END LOOP;
END;
```

```
 Statement processed.
1
2
3
4
5
6
7
8
9
10
```

Note: You must follow these steps while using PL/SQL WHILE Loop.

- Initialize a variable before the loop body.
- Increment the variable in the loop.
- You can use EXIT WHEN statements and EXIT statements in While loop but it is not done often.