**PL/SQL:**

**The 'Hello World' Example:**

DECLARE  
message varchar2 (20):= 'Hello, World!' ;  
BEGIN  
dbms\_output.put\_line(message);

END;

/

**-- The PL/SQL Comments**

DECLARE  
-- variable declaration  
message varchar2(20):= 'Hello, World!' ;  
BEGIN  
/\*  
\* PL/SQL executable statement(s)  
\*/  
dbms\_output. put\_line(message);  
END;  
/

**--Initializing Variables in PL/SQL**

DECLARE  
a integer :=10;  
b integer := 20;  
c integer;  
f real;  
BEGIN  
c := a + b;  
dbms\_output. put\_line('Value of c: ' || c);  
f := 70.0/3.0;  
dbms\_output. put\_line('Value of f: ' || f);  
END;  
/

**--Variable Scope in PL/SQL**

DECLARE  
-- Global variables  
num1 number :=95;  
num2 number := 85;  
BEGIN  
dbms\_output. put\_line('Outer Variable num1: ' || num1);  
dbms\_output. put\_line('Outer Variable num2: ' || num2);  
DECLARE  
-- Local variables  
num1 number := 195;  
num2 number := 185;  
BEGIN  
dbms\_output. put\_line('Inner Variable num1: ' || num1);  
dbms\_output. put\_line('Inner Variable num2: ' || num2);  
END;  
END;  
/

**Constants**

**Declaring a 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;  
/

**--Operators**

**Arithmetic Operators**

BEGIN  
dbms\_output. put\_line(10 + 5);  
dbms\_output. put\_line(10 - 5);  
dbms\_output. put\_line(10 \* 5);  
dbms\_output. put\_line(10 / 5);  
dbms\_output. put\_line(10 \*\* 5);  
END;  
/

**Relational Operators**

DECLARE  
a number (2) := 21;  
b number (2) := 10;  
BEGIN  
IF (a = b) then  
dbms\_output. put\_line('Line 1 - a is equal to b' );  
ELSE  
dbms\_output. put\_line('Line 1 - a is not equal to b' );  
END IF;

IF (a <b) then  
dbms\_output. put\_line('Line 2 - a is less than b' );  
ELSE  
dbms\_output. put\_line('Line 2 - a is not less than b' );  
END IF;  
IF ( a >b ) THEN  
dbms\_output. put\_line('Line 3 - a is greater than b' );  
ELSE  
dbms\_output. put\_line('Line 3 - a is not greater than b' );  
END IF;  
-- Lets change value of a and b  
a := 5;  
b := 20;  
IF ( a <= b ) THEN  
dbms\_output. put\_line('Line 4 - a is either equal or less than b' );  
END IF;  
IF ( b >= a ) THEN  
dbms\_output. put\_line('Line 5 - b is either equal or greater than a' );  
END IF;  
IF ( a <>b ) THEN  
dbms\_output. put\_line('Line 6 - a is not equal to b' );  
ELSE  
dbms\_output. put\_line('Line 6 - a is equal to b' );  
END IF;  
END;  
/

**Comparison Operators**

**1. LIKE Operator:**

DECLARE  
PROCEDURE compare (value varchar2, pattern varchar2 )is  
BEGIN  
IF value LIKE pattern THEN  
dbms\_output. put\_line('True' );  
ELSE  
dbms\_output. put\_line('False' );  
END IF;  
END;  
BEGIN  
compare('Zara Ali' , 'Z%A\_i' );  
compare('Nuha Ali' , 'Z%A\_i' );  
END;  
/

**2. BETWEEN Operator:**

DECLARE  
x number(2) := 10;  
BEGIN  
IF (x between 5 and 20) THEN  
dbms\_output. put\_line('True' );  
ELSE  
dbms\_output. put\_line('False' );  
END IF;  
IF (x BETWEEN 5 AND 10) THEN  
dbms\_output. put\_line('True' );

ELSE  
dbms\_output. put\_line('False' );  
END IF;  
IF (x BETWEEN 11 AND 20) THEN  
dbms\_output. put\_line('True' );  
ELSE  
dbms\_output. put\_line('False' );  
END IF;  
END;  
/

**3. IN and IS NULL Operators:**

DECLARE  
letter varchar2(1) := 'm' ;  
BEGIN  
IF (letter in ('a' , 'b' , 'c' )) THEN  
dbms\_output. put\_line('True' );  
ELSE  
dbms\_output. put\_line('False' );  
END IF;  
IF (letter in ('m' , 'n' , 'o' )) THEN  
dbms\_output. put\_line('True' );  
ELSE  
dbms\_output. put\_line('False' );  
END IF;  
IF (letter is null) THEN  
dbms\_output. put\_line('True' );  
ELSE  
dbms\_output. put\_line('False' );  
END IF;  
END;  
/

**Logical Operators**

DECLARE  
a boolean:=true;  
b boolean:= false;  
BEGIN  
IF (a AND b) THEN  
dbms\_output. put\_line('Line 1 - Condition is true' );  
END IF;  
IF (a OR b) THEN  
dbms\_output. put\_line('Line 2 - Condition is true' );  
END IF;  
IF (NOT a) THEN  
dbms\_output. put\_line('Line 3 - a is not true' );  
ELSE  
dbms\_output. put\_line('Line 3 - a is true' );  
END IF;  
IF (NOT b) THEN  
dbms\_output. put\_line('Line 4 - b is not true' );  
ELSE  
dbms\_output. put\_line('Line 4 - b is true' );  
END IF;  
END;  
/

**--Conditions**

**IF - THEN statement**

DECLARE  
a number(2) := 10;  
BEGIN  
a:= 10;  
-- 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 ' );  
END IF;  
dbms\_output. put\_line('value of a is : ' || a);  
END;  
/

**2.**

SELECT salary  
INTO c\_sal  
FROM customers  
WHERE id = c\_id;  
IF (c\_sal<= 2000) THEN  
UPDATE customers  
SET salary = salary + 1000  
WHERE id = c\_id;  
dbms\_output. put\_line('Salary updated' );  
END IF;  
END;  
/

**IF-THEN-ELSE statement**

DECLARE  
a number(3) := 100;  
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;  
/

**IF-THEN-ELSIF statement**

DECLARE  
a number(3) := 100;  
BEGIN  
IF ( a = 10 ) THEN  
dbms\_output. put\_line('Value of a is 10' );  
ELSIF ( a = 20 ) THEN  
dbms\_output. put\_line('Value of a is 20' );  
ELSIF ( a = 30 ) THEN  
dbms\_output. put\_line('Value of a is 30' );  
ELSE  
dbms\_output. put\_line('None of the values is matching' );  
END IF;  
dbms\_output. put\_line('Exact value of a is: ' || a );  
END;  
/

**Case statement**

DECLARE  
grade char(1) := 'A' ;  
BEGIN  
CASE grade  
when 'A' then dbms\_output. put\_line('Excellent' );  
when 'B' then dbms\_output. put\_line('Very good' );  
when 'C' then dbms\_output. put\_line('Well done' );  
when 'D' then dbms\_output. put\_line('You passed' );  
when 'F' then dbms\_output. put\_line('Better try again' );  
else dbms\_output. put\_line('No such grade' );  
END CASE;  
END;  
/

**Searched CASE statement**

DECLARE  
grade char(1) := 'B' ;  
BEGIN  
case  
when grade = 'A' then dbms\_output. put\_line('Excellent' );  
when grade = 'B' then dbms\_output. put\_line('Very good' );  
when grade = 'C' then dbms\_output. put\_line('Well done' );  
when grade = 'D' then dbms\_output. put\_line('You passed' );  
when grade = 'F' then dbms\_output. put\_line('Better try again' );  
else dbms\_output. put\_line('No such grade' );  
end case;

END;  
/

**Nested IF-THEN-ELSE**

DECLARE  
a number(3) := 100;  
b number(3) := 200;  
BEGIN  
-- check the boolean condition  
IF( a = 100 ) THEN  
-- if condition is true then check the following  
IF( b = 200 ) THEN  
-- if condition is true then print the following  
dbms\_output. put\_line('Value of a is 100 and b is 200' );  
END IF;  
END IF;  
dbms\_output. put\_line('Exact value of a is : ' || a );  
dbms\_output. put\_line('Exact value of b is : ' || b );  
END;  
/

**--Loops**

**PL/SQL Basic LOOP**

DECLARE  
x number :=10;  
BEGIN  
LOOP  
dbms\_output. put\_line(x);  
x := x + 10;  
IF x >50 THEN  
exit;  
END IF;  
END LOOP;  
-- after exit, control resumes here  
dbms\_output. put\_line('After Exit x is: ' || x);  
END;  
/

**PL/SQL WHILE LOOP**

DECLARE  
a number(2) := 10;  
BEGIN  
WHILE a <20 LOOP  
dbms\_output. put\_line('value of a: ' || a);  
a := a + 1;  
END LOOP;  
END;  
/

**PL/SQL FOR LOOP**

DECLARE  
a number(2);  
BEGIN

FOR a in 10 ..20 LOOP  
dbms\_output. put\_line('value of a: ' || a);  
END LOOP;  
END;  
/

**Reverse FOR LOOP Statement**

DECLARE  
a number(2) ;  
BEGIN  
FOR a IN REVERSE 10 .. 20 LOOP  
dbms\_output. put\_line('value of a: ' || a);  
END LOOP;  
END;  
/

**Nested loops in PL/SQL**

The following program uses a nested basic loop to find the prime numbers from 2 to 100:

DECLARE  
inumber(3);  
j number(3);  
BEGIN  
i:= 2;  
LOOP  
j := 2;  
LOOP  
exit WHEN ((mod(i, j ) = 0) or (j = i));  
j := j +1;  
END LOOP;  
IF (j = i) THEN  
dbms\_output. put\_line(i|| ' is prime' );  
END IF;  
i:= i+ 1;  
exit WHEN i= 50;  
END LOOP;  
END;  
/

**Labeling a PL/SQL Loop**

DECLARE  
inumber(1);  
j number(1);  
BEGIN  
<<outer\_loop>>  
FOR i IN 1. . 3 LOOP  
<<inner\_loop>>  
FOR j IN 1. . 3 LOOP  
dbms\_output. put\_line('i is: ' || i|| ' and j is: ' || j );  
END loop inner\_loop;  
END loop outer\_loop;  
END;  
/

**The EXIT WHEN Statement**

DECLARE  
x number :=10;  
BEGIN  
LOOP  
dbms\_output. put\_line(x);  
x := x + 10;  
exit WHEN x >50;  
END LOOP;  
-- after exit, control resumes here  
dbms\_output. put\_line('After Exit x is: ' || x);  
END;  
/

**CONTINUE statement**

DECLARE  
a number(2) := 10;  
BEGIN  
-- while loop execution  
WHILE a <20 LOOP  
dbms\_output. put\_line('value of a: ' || a);  
a := a + 1;  
IF a = 15 THEN  
-- skip the loop using the CONTINUE statement  
a := a + 1;  
CONTINUE;  
END IF;  
END LOOP;  
END;  
/

**GOTO statement**

DECLARE  
a number(2) := 10;  
BEGIN  
<<loopstart>>  
-- while loop execution  
WHILE a <20 LOOP  
dbms\_output. put\_line('value of a: ' || a);  
a := a + 1;  
IF a = 15 THEN  
a := a + 1;  
GOTO loopstart;  
END IF;  
END LOOP;  
END;  
/

**Declaring String Variables**

DECLARE  
name varchar2(20);  
company varchar2(30);

introduction clob;  
choice char(1);  
BEGIN  
name := 'John Smith' ;  
company := 'Infotech' ;  
introduction := ' Hello! I''m John Smith from Infotech.' ;  
choice := 'y' ;  
IF choice = 'y' THEN  
dbms\_output. put\_line(name);  
dbms\_output. put\_line(company);  
dbms\_output. put\_line(introduction);  
END IF;  
END;  
/

**Procedures**

**Creating a Procedure**

CREATE OR REPLACE PROCEDURE greetings  
AS  
BEGIN  
dbms\_output. put\_line('Hello World!' );  
END;  
/

**IN & OUT Mode Example 1**

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;  
/

**2.**

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;  
/

**Functions**

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.

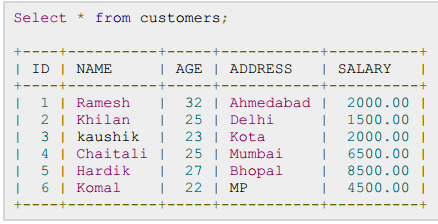
DECLARE  
a number;  
b number;  
c number;  
FUNCTION findMax(x IN number, y IN number)  
RETURN number  
IS  
z number;  
BEGIN  
IF x >y THEN  
z:= x;  
ELSE  
Z:= y;  
END IF;  
RETURN z;  
END;  
BEGIN  
a:= 23;  
b:= 45;  
c := findMax(a, b);  
dbms\_output. put\_line(' Maximum of (23,45): ' || c);  
END;  
/

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

DECLARE  
numnumber;  
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;  
/

**Cursors**

**Implicit Cursors**



The following program would update the table and increase salary of each customer by 500 and  
use the SQL%ROWCOUNT attribute to determine the number of rows affected:

DECLARE  
total\_rows number(2);  
BEGIN  
UPDATE customers  
SET salary = salary + 500;  
IF sql%notfound THEN  
dbms\_output. put\_line('no customers selected' );  
ELSIF sql%found THEN  
total\_rows := sql%rowcount;  
dbms\_output. put\_line(total\_rows || ' customers selected ' );  
END IF;  
END;  
/

**Explicit Cursors**

DECLARE  
c\_id customers. id%type;  
c\_name customers. name%type;  
c\_addr customers. address%type;  
CURSOR c\_customers is  
SELECT id, name, address FROM customers;  
BEGIN  
OPEN c\_customers;  
LOOP  
FETCH c\_customers into c\_id, c\_name, c\_addr;  
dbms\_output. put\_line(c\_id|| ' ' || c\_name|| ' ' || c\_addr);  
EXIT WHEN c\_customers%notfound;  
END LOOP;  
CLOSE c\_customers;  
END;  
/

**Records**

**Table-Based Records**

DECLARE  
customer\_rec customers%rowtype;  
BEGIN  
SELECT \* into customer\_rec  
FROM customers  
WHERE id = 5;  
dbms\_output. put\_line('Customer ID: ' || customer\_rec. id);  
dbms\_output. put\_line('Customer Name: ' || customer\_rec. name);  
dbms\_output. put\_line('Customer Address: ' || customer\_rec. address);  
dbms\_output. put\_line('Customer Salary: ' || customer\_rec. salary);

END;  
/

**Cursor-Based Records**

DECLARE  
CURSOR customer\_cur is  
SELECT id, name, address  
FROM customers;  
customer\_rec customer\_cur%rowtype;  
BEGIN  
OPEN customer\_cur;  
LOOP  
FETCH customer\_cur into customer\_rec;  
EXIT WHEN customer\_cur%notfound;  
DBMS\_OUTPUT. put\_line(customer\_rec. id || ' ' ||  
customer\_rec. name);  
END LOOP;  
END;  
/

**User-Defined Records**

DECLARE  
type books is record  
(title varchar(50),  
author varchar(50),  
subject varchar(100),  
book\_idnumber);  
book1 books;  
book2 books;  
BEGIN  
-- Book 1 specification  
book1. title:= 'C Programming' ;  
book1. author:= 'Nuha Ali ' ;  
book1. subject:= 'C Programming Tutorial' ;  
book1. book\_id:= 6495407;  
-- Book 2 specification  
book2. title:= 'Telecom Billing' ;  
book2. author:= 'Zara Ali' ;  
book2. subject:= 'Telecom Billing Tutorial' ;  
book2. book\_id:= 6495700;

END;  
/

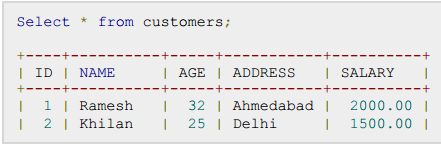
**Exceptions**

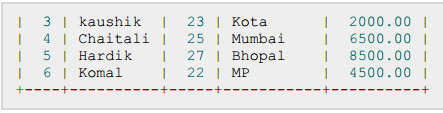
DECLARE  
c\_id customers. id%type:= 8;  
c\_name customers. name%type;  
c\_addr customers. address%type;  
BEGIN  
SELECT name, address INTO c\_name, c\_addr  
FROM customers  
WHERE id = c\_id;  
DBMS\_OUTPUT. PUT\_LINE ('Name: ' || c\_name);  
DBMS\_OUTPUT. PUT\_LINE ('Address: ' || c\_addr);  
EXCEPTION  
WHEN no\_data\_found THEN  
dbms\_output. put\_line('No such customer!' );  
WHEN others THEN  
dbms\_output. put\_line('Error!' );  
END;  
/

**User-defined Exceptions**

DECLARE  
c\_id customers. id%type:= &c\_id;  
c\_name customers. name%type;  
c\_addr customers. address%type;  
-- user defined exception  
ex\_invalid\_id EXCEPTION;  
BEGIN  
IF c\_id<= 0 THEN  
RAISE ex\_invalid\_id;  
ELSE  
SELECT name, address INTO c\_name, c\_addr  
FROM customers  
WHERE id = c\_id;  
DBMS\_OUTPUT. PUT\_LINE ('Name: ' || c\_name);  
DBMS\_OUTPUT. PUT\_LINE ('Address: ' || c\_addr);  
END IF;  
EXCEPTION  
WHEN ex\_invalid\_id THEN  
dbms\_output. put\_line('ID must be greater than zero!' );  
WHEN no\_data\_foundTHEN  
dbms\_output. put\_line('No such customer!' );  
WHEN others THEN  
dbms\_output. put\_line('Error!' );  
END;  
/

**Triggers**





CREATE OR REPLACE TRIGGER display\_salary\_changes  
BEFORE DELETE OR INSERT OR UPDATE ON customers  
FOR EACH ROW  
WHEN (NEW.ID > 0)  
DECLARE  
sal\_diff number;  
BEGIN  
sal\_diff := :NEW.salary - :OLD.salary;  
dbms\_output.put\_line('Old salary: ' || :OLD.salary);  
dbms\_output.put\_line('New salary: ' || :NEW.salary);  
dbms\_output.put\_line('Salary difference: ' || sal\_diff);  
END;  
/

INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)  
VALUES (7, 'Kriti' , 22, 'HP' , 7500.00 );

UPDATE customers  
SET salary = salary + 500  
WHERE id = 2;

**Packages**

CREATE OR REPLACE PACKAGE BODY cust\_salAS  
PROCEDURE find\_sal(c\_id customers. id%TYPE) IS  
c\_sal customers. salary%TYPE;  
BEGIN  
SELECT salary INTO c\_sal  
FROM customers  
WHERE id = c\_id;  
dbms\_output. put\_line('Salary: ' || c\_sal);  
END find\_sal;  
END cust\_sal;  
/