Differentiate between a PL/SQL function and a procedure ?

	Function	Procedure
1)	A function is used to calculate result using given inputs	A procedure is used to perform certain task in order.
2)	A function can be called by a procedure	A procedure cannot be called by a function.
3)	DML statements cannot be executed within a function	DML statements can be executed within a procedure.
4)	A function can be called within a query.	A procedure cannot be called within a query.
5)	Whenever a function is called, it is first compiled before being called.	A procedure is compiled once & can be called multiple times without being compiled.
6)	A function returns a value and control to calling function or code.	A procedure returns the control but not any value to calling function or code.

cannot = DML, try-catch
explicit transaction
handling.

7	A function has no support for try-catch	A procedure has support for try-catch blocks
8	A select statement can have a function call.	A select statement can't have procedure call
9	A function can not have explicit transaction handling.	A procedure can use explicit transaction handling

Q) Explain with syntax how standalone function can be created in PL/SQL. Develop a PL/SQL function that computes & returns maximum of two values

→ A standalone function is created using CREATE FUNCTION statement.

Syntax:

```
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,

1) function_name = name of the function.

2) [OR REPLACE] option allows the modification of an existing function.

3) The optional parameter list contains
   1) name 2) mode & types of parameters.

   IN = value that will be passed from outside

   OUT = parameter that will be used to return a value outside of the procedure.

4) function must contain a return statement

5) RETURN clause specifies the data type you are going to return from the function

6) function-body contains the executable part.

7) AS keyword is used instead of IS keyword for creating a standalone function

ex:

```
declare
    n1  number;
    n2  number;
    sum1  number;
Function  fun (x IN  number, y IN number)
RETURN  number
IS
    z  number;
Begin
    IF  x>y  then
    z1 = x;
    else
    z1 = y;
    end  if;
    return z;
end;

Begin
    n1:=112;
    n2:=15;
    sum1 := fun (n1, n2);
    dbms_output.put_line ('Maximum of Two Numbers'
                          || sum1);

end;
```

3) What are the components of PL/SQL block structure? Explain with syntax & an example.

→ Consists of three sections

1) The Declaration section (optional)
2) The Executional section (mandatory)
3) The Exception Handling (Error) section (optional)

Declarations

Starts with keyword DECLARE
It is optional section
Defines all variables, cursors, subprograms & other elements to be used in the program

Executable Commands

Enclosed between keywords BEGIN & END
It is mandatory section
Consists of executable PL/SQL statements of program
It should have at least one executable line of code, which may be just NULL command to indicate that nothing should be executed

Exception Handling

starts with keyword EXCEPTION
Optional section
Contains exception(s) that handle errors in program

~~Example~~:

Syntax:

```
DECLARE
    <declaration section>

BEGIN
    <executable command(s)>

EXCEPTION
    <expection handling>
END; 4
```

Example:

```
DECLARE
   a number;
   b number;
   c number;
BEGIN
   a := 10;
   b := 100;
   IF a > b THEN
      c := a;
   ELSE
      c := b;
   END IF;
   dbms_output.put_line('Maximum number in 10 & 100: ' ||
END;
```

4) When would you use a PL/SQL loop? With syntax and an example, explain PL/SQL FOR and WHILE loops

→

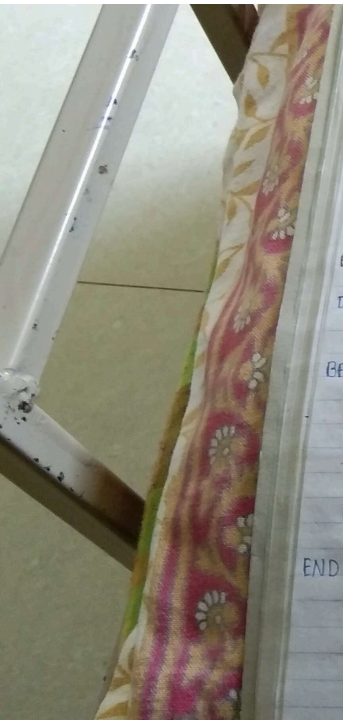
Sequence of statements is enclosed between the LOOP and the END LOOP statements. At each iteration, the sequence of statement is executed and then control resumes at the top of the loop

```
LOOP
   sequence of statements;
END LOOP;
```

Example

```
DECLARE
   x number := 10;
BEGIN
   LOOP
      dbms_output.put_line(x);
      x := x+10;
      IF x > 50 THEN
         exit;
      END IF;
   END LOOP;

   dbms_output.put_line ("After Exit x is: ' || x);
END;
```



PL/SQL WHILE LOOP

Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.

Syntax:

```
WHILE condition LOOP
   sequence_of_statements
END LOOP;
```

Example:

```
DECLARE
   a number(2) := 10;

BEGIN

   WHILE a < 20 LOOP
      dbms_output.putline('value of a: ' ||a);
      a := a+1;

   END LOOP;

END;
```

PL/SQL FOR LOOP

Execute a sequence of statements multiple times & abbreviates the code that manages the loop variable.

Syntax:

```
FOR counter IN initial_value .. final_value LOOP
   sequence_of_statements;
END LOOP;
```

Example:

```
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.putline('value of a: ' ||a);
   END LOOP;
END;
```

5) Explain with syntax how a procedure can be created in PL/SQL.

→ A procedure is created with the CREATE OR REPLACE PROCEDURE statement.

Syntax :

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[(parameter_name [IN/OUT/IN OUT] type [,...])]

{IS AS}
BEGIN
   <procedure_body>

END procedure_name;
```

Where,

1) procedure_name = the name of procedure.

2) [OR REPLACE] = modification of existing procedure

3) optional parameter list contains name, mode & types of the parameters

4) IN value that will be passed from outside

   OUT the parameter that will be used to return a value outside of the procedure

5) procedure_body : executable part.

As keyword is used instead of the IS keyword for creating a standalone procedure

Example

```
CREATE OR REPLACE 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
```

```
declare
a number;
b number;
c number;
begin
a:=23;
b:=45;
findMin(a,b,c);
dbms_output.put_line('MINIMUM' || c);
end;
```

o/p

23

6) Explain PL/SQL variables declarations.

→ 1) The name of a PL/SQL variable consists of a letter optionally followed by more 1) letters 2) dollar signs 3) numerals or underscores 5) number signs 6) should not exceed 30 characters.

2) By default, variable names are not case-sensitive.

Variable Declaration in PL/SQL

1) PL/SQL variables must be declared in the declaration section or
2) in package as a global variable.

3) PL/SQL allocates memory for the variable's value.

4) Storage location is identified by variable name.

Syntax:

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

Example:

```
sales number (10,2);
pi CONSTANT double precision := 3.1415;
name varchar2(25);
address varchar2(100);
```

When you provide a size, scale or precision limit with data type, it is called a constrained declaration.

Constrained declaration require less memory than unconstrained declaration.

Ex

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

Initializing Variables in PL/SQL

1) Default value of is NULL
2) If you want to initialize a variable with a value other than NULL using either of following

1) The DEFAULT keyword
2) The assignment operator

Ex:

```
counter binary_integer := 0;
greetings varchar2(20) DEFAULT 'Have a Good Day';
```

7) Write PL/SQL code to find factrial of a number.

→

```
declare
   fac number := 1;
   n number := &1;
begin
   while n > 0 loop
   fac := n * fac;
   n := n-1;
end loop;
dbms_output.put_line(fac);
end;
```

o/p (if given i/p as 5)

120

8) Write PL/SQL code to swap two numbers.

→

```
declare
   num1 number;
   num2 number;
   temp number;
begin
   num1 := 1000;
   num2 := 2000;
   dbms_output.putlines('before');
   dbms_output.putline('num1 = ' || num1 ||
                       'num2 = ' || num2);
   temp := num1;
   num1 := num2;
   num2 := temp;
   dbms_output.put_line ('after');
   dbms_output.put_line ('num1 = ' || num1 || ' num2 = ' ||
                         num2);
end;
```

o/p

before	after
num1 = 1000	num1 = 2000
num2 = 2000	num2 = 1000

Q: Write PL/SQL code to reverse a given number

```
declare
  n number;
  ? number;
  rev number := 0;
  r number;

begin
  n := &n;
  while n>0
  loop
   r:=mod (n,10);
   rev:= (rev * 10) +r;
    n:= trunc (n/10);
   end loop

  dbms_output.put_line ('reverse is ' ||rev);

end;
```

o/p:

enter value for n : 4578

old 8 : n := &n;

new 8 : n := 4578;

reverse is 8754