```
--------------------------------------------------------------------------------
--------------------------------
------Whate Are The Functionb And Procedur And Why We Use------------
--------------------------------------------------------------------------

-----------------An anonymous block example-------------------------
declare
    cursor c_emps is select * from employees_copy for update;
    v_salary_increase number:= 1.10;
    v_old_salary number;
begin
    for r_emp in c_emps loop
      v_old_salary := r_emp.salary;
      r_emp.salary := r_emp.salary*v_salary_increase + r_emp.salary *
nvl(r_emp.commission_pct,0);
      update employees_copy set row = r_emp where current of c_emps;
      dbms_output.put_line('The salary of : '|| r_emp.employee_id
                           || ' is increased from '||v_old_salary||' to '||
r_emp.salary);
    end loop;
end;
-----------------An anonymous block example
2-----------------------------------------------------
declare
    cursor c_emps is select * from employees_copy for update;
    v_salary_increase number:= 1.10;
    v_old_salary number;
    v_new_salary number;
    v_salary_max_limit pls_integer := 20000;
begin
    for r_emp in c_emps loop
      v_old_salary := r_emp.salary;
      --check salary area
      v_new_salary := r_emp.salary*v_salary_increase + r_emp.salary *
nvl(r_emp.commission_pct,0);
      if v_new_salary > v_salary_max_limit then
       RAISE_APPLICATION_ERROR(-20000, 'The new salary of '||r_emp.first_name|| '
cannot be higher than '|| v_salary_max_limit);
      end if;
      r_emp.salary := r_emp.salary*v_salary_increase + r_emp.salary *
nvl(r_emp.commission_pct,0);
      ----------
      update employees_copy set row = r_emp where current of c_emps;
      dbms_output.put_line('The salary of : '|| r_emp.employee_id
                           || ' is increased from '||v_old_salary||' to '||
r_emp.salary);
    end loop;
end;

--------------------------------------------------------------------------------
--------------------------------
-----------------------------Creating And Using Stored Procedure--------------
--------------------------------------------------------------------------
----------------- Creating a procedure
create procedure increase_salaries as
    cursor c_emps is select * from employees_copy for update;
    v_salary_increase number := 1.10;
    v_old_salary number;
begin
```

```
    for r_emp in c_emps loop
      v_old_salary := r_emp.salary;
      r_emp.salary := r_emp.salary * v_salary_increase + r_emp.salary *
nvl(r_emp.commission_pct,0);
      update employees_copy set row = r_emp where current of c_emps;
      dbms_output.put_line('The salary of : '|| r_emp.employee_id
                           || ' is increased from '||v_old_salary||' to '||
r_emp.salary);
    end loop;
end;
----------------- Multiple procedure
usage-----------------------------------------
begin
  dbms_output.put_line('Increasing the salaries!...');
  INCREASE_SALARIES;
  INCREASE_SALARIES;
  INCREASE_SALARIES;
  INCREASE_SALARIES;
  dbms_output.put_line('All the salaries are successfully increased!...');
end;
----------------- Different procedures in one
block-----------------------------------
begin
  dbms_output.put_line('Increasing the salaries!...');
  INCREASE_SALARIES;
  new_line;
  INCREASE_SALARIES;
  new_line;
  INCREASE_SALARIES;
  new_line;
  INCREASE_SALARIES;
  dbms_output.put_line('All the salaries are successfully increased!...');
end;
-----------------Creating a procedure to ease the dbms_output.put_line
procedure-----------------
create procedure new_line as
begin
  dbms_output.put_line('-----------------------------------------');
end;
-----------------Modifying the procedure with using the OR REPLACE
command.-----------------------
create or replace procedure increase_salaries as
    cursor c_emps is select * from employees_copy for update;
    v_salary_increase number := 1.10;
    v_old_salary number;
begin
    for r_emp in c_emps loop
      v_old_salary := r_emp.salary;
      r_emp.salary := r_emp.salary * v_salary_increase + r_emp.salary *
nvl(r_emp.commission_pct,0);
      update employees_copy set row = r_emp where current of c_emps;
      dbms_output.put_line('The salary of : '|| r_emp.employee_id
                           || ' is increased from '||v_old_salary||' to '||
r_emp.salary);
    end loop;
    dbms_output.put_line('Procedure finished executing!');
end
--------------------------------------------------------------------------------
--------------------------------
```

```
--------------------------------Usin IN OUT
Parameter-----------------------------------------------------
-------------------------------------------------------------------------
-----------------Creating a procedure with the IN parameters
create or replace procedure increase_salaries (v_salary_increase in number,
v_department_id pls_integer) as
    cursor c_emps is select * from employees_copy where department_id =
v_department_id for update;
    v_old_salary number;
begin
    for r_emp in c_emps loop
      v_old_salary := r_emp.salary;
      r_emp.salary := r_emp.salary * v_salary_increase + r_emp.salary *
nvl(r_emp.commission_pct,0);
      update employees_copy set row = r_emp where current of c_emps;
      dbms_output.put_line('The salary of : '|| r_emp.employee_id
                           || ' is increased from '||v_old_salary||' to '||
r_emp.salary);
    end loop;
    dbms_output.put_line('Procedure finished executing!');
end;
----------------- Creating a procedure with the OUT
parameters-----------------------------------------
create or replace procedure increase_salaries
    (v_salary_increase in out number, v_department_id pls_integer,
v_affected_employee_count out number) as
    cursor c_emps is select * from employees_copy where department_id =
v_department_id for update;
    v_old_salary number;
    v_sal_inc number := 0;
begin
    v_affected_employee_count := 0;
    for r_emp in c_emps loop
      v_old_salary := r_emp.salary;
      r_emp.salary := r_emp.salary * v_salary_increase + r_emp.salary *
nvl(r_emp.commission_pct,0);
      update employees_copy set row = r_emp where current of c_emps;
      dbms_output.put_line('The salary of : '|| r_emp.employee_id
                           || ' is increased from '||v_old_salary||' to '||
r_emp.salary);
      v_affected_employee_count := v_affected_employee_count + 1;
      v_sal_inc := v_sal_inc + v_salary_increase + nvl(r_emp.commission_pct,0);
    end loop;
    v_salary_increase := v_sal_inc / v_affected_employee_count;
    dbms_output.put_line('Procedure finished executing!');
end;
-----------------Another example of creating a procedure with the IN parameter
----------------------
CREATE OR REPLACE PROCEDURE PRINT(TEXT IN VARCHAR2) IS
BEGIN
  DBMS_OUTPUT.PUT_LINE(TEXT);
END;
-----------------Using the procedures that has the IN parameters
---------------------------------
begin
 PRINT('SALARY INCREASE STARTED!..');
 INCREASE_SALARIES(1.15,90);
 PRINT('SALARY INCREASE FINISHED!..');
end;
```

```
----------------Using the procedure that has OUT parameters
-----------------------------------
declare
  v_sal_inc number := 1.2;
  v_aff_emp_count number;
begin
 PRINT('SALARY INCREASE STARTED!..');
 INCREASE_SALARIES(v_sal_inc,80,v_aff_emp_count);
 PRINT('The affected employee count is : '|| v_aff_emp_count);
 PRINT('The average salary increase is : '|| v_sal_inc || ' percent!..');
 PRINT('SALARY INCREASE FINISHED!..');
end;
--------------------------------------------------------------------------------
--------------------------------
------------------------------Named & Mixed Notations and Default Option (Code
Samples)--------------------
--------------------------------------------------------------------------
----------------- A standard procedure creation with a default value
create or replace PROCEDURE PRINT(TEXT IN VARCHAR2 := 'This is the print
function!.') IS
BEGIN
  DBMS_OUTPUT.PUT_LINE(TEXT);
END;
-----------------Executing a procedure without any parameter. It runs because it
has a default value.----------
exec print();
-----------------Running a procedure with null value will not use the default value
-------------------------
exec print(null);
-----------------Procedure creation of a default value usage
create or replace procedure add_job(job_id pls_integer, job_title varchar2,
                                    min_salary number default 1000, max_salary
number default null) is
begin
  insert into jobs values (job_id,job_title,min_salary,max_salary);
  print('The job : '|| job_title || ' is inserted!..');
end;
-----------------A standard run of the
procedure-----------------------------------------------

exec ADD_JOB('IT_DIR','IT Director',5000,20000);
-----------------Running a procedure with using the default
values----------------------------

exec ADD_JOB('IT_DIR2','IT Director',5000);
-----------------Running a procedure with the named
notation-----------------------------------

exec ADD_JOB('IT_DIR5','IT Director',max_salary=>10000);
-----------------Running a procedure with the named notation example
2-------------------------

exec ADD_JOB(job_title=>'IT Director',job_id=>'IT_DIR7',max_salary=>10000 ,
min_salary=>500);
--------------------------------------------------------------------------------
--------------------------------
------------------------------Creating and Using PL/SQL Functions (Code
Samples)--------------------------------
--------------------------------------------------------------------------
```

```
CREATE OR REPLACE FUNCTION get_avg_sal (p_dept_id departments.department_id%type)
RETURN number AS
v_avg_sal number;
BEGIN
  select avg(salary) into v_avg_sal from employees where department_id = p_dept_id;
  RETURN v_avg_sal;
END get_avg_sal;
----------------- using a function in begin-end
block-------------------------------------------

declare
  v_avg_salary number;
begin
  v_avg_salary := get_avg_sal(50);
  dbms_output.put_line(v_avg_salary);
end;
----------------- using functions in a select
clause-----------------------------------------------------

select employee_id,first_name,salary,department_id,get_avg_sal(department_id)
avg_sal from employees;
----------------- using functions in group by, order by, where clauses
select get_avg_sal(department_id) from employees
where salary > get_avg_sal(department_id)
group by get_avg_sal(department_id)
order by get_avg_sal(department_id);
----------------- dropping a function---------------------------

drop function get_avg_sal;
-------------------------------------------------------------------------------------------
---------------------------------
-----------------------------Local Subprograms (Code
Samples)-----------------------------------------------------
-------------------------------------------------------------------------
----------------- creating and using subprograms in anonymous blocks - false usage
create table emps_high_paid as select * from employees where 1=2;
/
declare
  procedure insert_high_paid_emp(emp_id employees.employee_id%type) is
    emp employees%rowtype;
    begin
      emp := get_emp(emp_id);
      insert into emps_high_paid values emp;
    end;
  function get_emp(emp_num employees.employee_id%type) return employees%rowtype is
    emp employees%rowtype;
    begin
      select * into emp from employees where employee_id = emp_num;
      return emp;
    end;
begin
  for r_emp in (select * from employees) loop
    if r_emp.salary > 15000 then
      insert_high_paid_emp(r_emp.employee_id);
    end if;
  end loop;
end;
----------------- reating and using subprograms in anonymous blocks - true
usage---------------------
```

```
declare
  function get_emp(emp_num employees.employee_id%type) return employees%rowtype is
    emp employees%rowtype;
    begin
      select * into emp from employees where employee_id = emp_num;
      return emp;
    end;
  procedure insert_high_paid_emp(emp_id employees.employee_id%type) is
    emp employees%rowtype;
    begin
      emp := get_emp(emp_id);
      insert into emps_high_paid values emp;
    end;
begin
  for r_emp in (select * from employees) loop
    if r_emp.salary > 15000 then
      insert_high_paid_emp(r_emp.employee_id);
    end if;
  end loop;
end;
----------------- The scope of emp
record-----------------------------------------------

declare
  procedure insert_high_paid_emp(emp_id employees.employee_id%type) is
    emp employees%rowtype;
    e_id number;
    function get_emp(emp_num employees.employee_id%type) return employees%rowtype
is
      begin
        select * into emp from employees where employee_id = emp_num;
        return emp;
      end;
    begin
      emp := get_emp(emp_id);
      insert into emps_high_paid values emp;
    end;
begin
  for r_emp in (select * from employees) loop
    if r_emp.salary > 15000 then
      insert_high_paid_emp(r_emp.employee_id);
    end if;
  end loop;
end;
--------------------------------------------------------------------------------
---------------------------------
-------------------------------Overloading the Subprograms (Code
Samples)----------------------------------------
--------------------------------------------------------------------------
declare
  procedure insert_high_paid_emp(p_emp employees%rowtype) is
    emp employees%rowtype;
    e_id number;
    function get_emp(emp_num employees.employee_id%type) return employees%rowtype
is
      begin
        select * into emp from employees where employee_id = emp_num;
        return emp;
```

```
      end;
    function get_emp(emp_email employees.email%type) return employees%rowtype is
      begin
        select * into emp from employees where email = emp_email;
        return emp;
      end;
    begin
      emp := get_emp(p_emp.employee_id);
      insert into emps_high_paid values emp;
    end;
begin
  for r_emp in (select * from employees) loop
    if r_emp.salary > 15000 then
      insert_high_paid_emp(r_emp);
    end if;
  end loop;
end;
----------------- overloading with multiple usages----------------------

declare
  procedure insert_high_paid_emp(p_emp employees%rowtype) is
    emp employees%rowtype;
    e_id number;
    function get_emp(emp_num employees.employee_id%type) return employees%rowtype
is
      begin
        select * into emp from employees where employee_id = emp_num;
        return emp;
      end;
    function get_emp(emp_email employees.email%type) return employees%rowtype is
      begin
        select * into emp from employees where email = emp_email;
        return emp;
      end;
    function get_emp(f_name varchar2, l_name varchar2) return employees%rowtype is
      begin
        select * into emp from employees where first_name = f_name and last_name =
l_name;
        return emp;
      end;
    begin
      emp := get_emp(p_emp.employee_id);
      insert into emps_high_paid values emp;
      emp := get_emp(p_emp.email);
      insert into emps_high_paid values emp;
      emp := get_emp(p_emp.first_name,p_emp.last_name);
      insert into emps_high_paid values emp;
    end;
begin
  for r_emp in (select * from employees) loop
    if r_emp.salary > 15000 then
      insert_high_paid_emp(r_emp);
    end if;
  end loop;
end;
--------------------------------------------------------------------------------
---------------------------------
--------------------------Handling the Exceptions in Subprograms (Code
Samples)--------------------------------
```

```
-------------------------------------------------------------------------
----------------- An unhandled exception in function
create or replace function get_emp(emp_num employees.employee_id%type) return
employees%rowtype is
  emp employees%rowtype;
 begin
   select * into emp from employees where employee_id = emp_num;
  return emp;
 end;
----------------- calling that function in an anonymous
block----------------------------

declare
  v_emp employees%rowtype;
begin
  dbms_output.put_line('Fetching the employee data!..');
  v_emp := get_emp(10);
  dbms_output.put_line('Some information of the employee are : ');
  dbms_output.put_line('The name of the employee is : '|| v_emp.first_name);
  dbms_output.put_line('The email of the employee is : '|| v_emp.email);
  dbms_output.put_line('The salary of the employee is : '|| v_emp.salary);
end;
----------------- hanling the exception wihout the return clause - not
working-------------------------


create or replace function get_emp(emp_num employees.employee_id%type) return
employees%rowtype is
  emp employees%rowtype;
 begin
   select * into emp from employees where employee_id = emp_num;
  return emp;
 exception
  when no_data_found then
    dbms_output.put_line('There is no employee with the id '|| emp_num);
 end;
----------------- handling and raising the
exception----------------------------------------------

create or replace function get_emp(emp_num employees.employee_id%type) return
employees%rowtype is
  emp employees%rowtype;
 begin
   select * into emp from employees where employee_id = emp_num;
  return emp;
 exception
  when no_data_found then
    dbms_output.put_line('There is no employee with the id '|| emp_num);
    raise no_data_found;
 end;
----------------- handling all possible exception
cases---------------------------------------

create or replace function get_emp(emp_num employees.employee_id%type) return
employees%rowtype is
  emp employees%rowtype;
 begin
   select * into emp from employees where employee_id = emp_num;
  return emp;
```

```
 exception
  when no_data_found then
    dbms_output.put_line('There is no employee with the id '|| emp_num);
    raise no_data_found;
  when others then
    dbms_output.put_line('Something unexpected happened!.');
 return null;
 end;
--------------------------------------------------------------------------------
--------------------------------
------------------------------Regular & Pipelined Table Functions (Code
Samples)------------------------------
--------------------------------------------------------------------------
CREATE TYPE t_day AS OBJECT (
  v_date DATE,
  v_day_number INT
);
----------------- creating a nested table type---------------------------------

CREATE TYPE t_days_tab IS TABLE OF t_day;

----------------- creating a regular table function-----------------------------

CREATE OR REPLACE FUNCTION f_get_days(p_start_date DATE , p_day_number INT)
              RETURN t_days_tab IS
v_days t_days_tab := t_days_tab();
BEGIN
 FOR i IN 1 .. p_day_number LOOP
  v_days.EXTEND();
  v_days(i) := t_day(p_start_date + i, to_number(to_char(p_start_date + i,
'DDD')));
 END LOOP;
 RETURN v_days;
END;
----------------- querying from the regular table function------------------

select * from table(f_get_days(sysdate,1000000));
----------------- querying from the regular table function without the table
operator---------

select * from f_get_days(sysdate,1000000);
----------------- creating a pipelined table function-----------

create or replace function f_get_days_piped (p_start_date date , p_day_number int)
              return t_days_tab PIPELINED is
begin
 for i in 1 .. p_day_number loop
  PIPE ROW (t_day(p_start_date + i,
                  to_number(to_char(p_start_date + i,'DDD'))));
 end loop;
 RETURN;
end;
----------------- querying from the pipelined table function-------------

select * from f_get_days_piped(sysdate,1000000)
--------------------------------------------------------------------------------
--------------------------------
--------------------------------------------------------------------------------
---
```