Introduction To Exceptions ====>
==========

* Exceptions are Run Time Errors i.e errors which appear during the execution time of our program.

* Whenever an exception occurs , PL-SQL takes 2 steps:
- It immediately terminates the code
- It displays an error message related to the exception

* Both the steps are very unfriendly.

* So to make our code user friendly , Oracle strongly recommends us to use Exception Handling Mechanism in our code.

* Using Exception Handling we can test the code and avoid it from exiting abruptly.

* Syntax :-

```
BEGIN
    -- executable section ...
    -- exception-handling section
EXCEPTION
    WHEN e1 THEN
    -- exception_handler1
    WHEN e2 THEN
    -- exception_handler1
    WHEN OTHERS THEN
    -- other_exception_handler
END;
```

WA PL_SQL script to accept bookname 2 integers from the user and display the result of their division . Handle the `ZERO_DIVIDE` exception that might occur.

```
1  declare
2     a int;
3     b int;
4     c float;
5  begin
6     a := &a;
7     b := &b;
8     c := a / b;
9     dbms_output.put_line('Div is ' || c);
10 exception
11    when zero_divide then
12         dbms_output.put_line('Denominator can not be Zero');
13 end;
```

Exceptions Thrown By Select Into ===>
==========

* `NO_DATA_FOUND` :- When a SELECT...INTO clause does not return any row from a table.

* `TOO_MANY_ROWS` :- When we SELECT or fetch more than one row into a record or variable.

WA PL-SQL script to accept a empno from the user and display the name and sal of that employee . In case the given empno does not exists display the message emp not found.

```
1  declare
2    id emp.empno%type;
3    name emp.ename%type;
4    earn emp.sal%type;
5  begin
6    id := &empno;
7    select ename, sal into name, earn from emp where empno = id;
8    dbms_output.put_line(name || ',' || earn);
9  exception
10   when no_data_found then
11     dbms_output.put_line('No emp id ' || id || ' Found');
12 end;
```

List Of Oracle's Built In Exceptions ===>

Exception Name	Reason
CURSOR_ALREADY_OPEN	When you open a cursor that is already open.
INVALID_NUMBER	Wwhen the conversion of a character string into a number fails
NO_DATA_FOUND	When a SELECT...INTO clause does not return any row from a table.
TOO_MANY_ROWS	When you SELECT or fetch more than one row
ZERO_DIVIDE	When you attempt to divide a number by zero.
VALUE_ERROR	Assign a value gr than permissible limit

Handling Numbered Exceptions ===>

* In Oracle, some of the pre-defined exceptions are numbered in the form of four integers preceded by a hyphen symbol.

* To handle such exceptions we should assign a name to them before using them.

* This can be done by using the Pragma exception technique in which a numbered exception handler is bound to a name.

```
DECLARE
    exception_name EXCEPTION;
    PRAGMA EXCEPTION_INIT (exception_name, Err_code);
BEGIN
    Execution section
EXCEPTION
    WHEN exception_nameTHEN
        handle the exception
END;
```

Example :-

* Below we have a table with Student's data in it.

ROLLNO	SNAME	AGE	COURSE

```
  11       Anu     20     BSC
  12       Asha    21     BCOM
  13       Arpit   18     BCA
  14       Chetan  20     BCA
```

* When we add a record in a table with primary key constraint and the constraint gets violated because of duplicate key then Oracle raises the error with error number -1.

```
DECLARE
    sno student.rollno%type;
    snm student.sname%type;
    s_age student.age%type;
    cr student.course%type;
    already_exist EXCEPTION;
    pragma exception_init(already_exist, -1);
BEGIN
    sno := &rollno;
    snm := '&sname';
    s_age := &age;
    cr := '&course';
    INSERT into student values(sno, snm, s_age, cr);
    dbms_output.put_line('Record inserted');
EXCEPTION
    WHEN already_existTHEN
    dbms_output.put_line('Record already exist');
END;
```

User Defined Exceptions ===>

* Sometimes, it is necessary for programmers to name and trap their own exceptions - ones that aren't defined already by PL/SQL.

* These are called User Defined Exceptions and it is done in three steps:
- Create the exception
- Detect the condition and raise the exception
- Finally handle it in the exception block

Syntax :-

```
DECLARE
    user_define_exception_name EXCEPTION;
BEGIN
    statement(s);
    IF condition THEN
        RAISE user_define_exception_name;
    END IF;
EXCEPTION
    WHEN user_define_exception_nameTHEN
        User defined statement (action) will be taken;
END;
```

Example :-

* Below we have a table with Student's data in it.

```
ROLLNO   SNAME   Total_Courses
```

```
======  ======= ===============
  11      Anu         2
  12      Asha        1
  13      Arpit       3
  14      Chetan      1
```

WAPL-SQL script to insert an new record in the above STUDENTS table and if the total_courses inputted by the user in more than 3 then generate an exception.

```
DECLARE
    sno student.rollno%type;
    snm student.sname%type;
    crno student.total_course%type;
    invalid_total EXCEPTION;
BEGIN
    sno := &rollno;
    snm := '&sname';
    crno:=total_courses;
    IF (crno > 3) THEN
        RAISE invalid_total;
    ELSE
        INSERT into student values(sno, snm, crno);
    END IF;
EXCEPTION
    WHEN invalid_totalTHEN
    dbms_output.put_line('Total number of courses cannot be more than 3');
END;
```

Resuming After Exception ===>

```
BEGIN
    DECLARE
        var_decl;
    BEGIN
        executable_stmts;
    EXCEPTION
        WHEN <exception_name> Then
                stmt
    END;

    DECLARE
        var_decl;
    BEGIN
        executable_stmts;
    EXCEPTION
        WHEN <exception_name> Then
                stmt
    END;
END;
```

Exercise ==>

* Assume we have a Table called CIRCLE , with 2 columns called radius and area.

* Also assume that radius is primary key and the table has some records in it already.

```
    RAD      AREA
---------- ----------
    2        12.56
    5        78.5
    3        28.26
```

```
# WAPL-SQL script to insert records with the radius from 1 to 10 and display
the message which radius numbers are present and which are inserted by your
code.

declare
    already_exists exception;
    pragma exception_init(already_exists, -1);
    ar float;
begin
    for r in 1 .. 10 loop
        begin
        ar := 3.14 * power(r, 2);
            insert into circle values(r, ar);
                dbms_output.put_line('Radius' || r || ' record inserted');
        exception
            when already_exists then
                dbms_output.put_line('Radius' || r || ' already exists');
        end;
    end loop;
end;
/

# Output ==>
Radius1 record inserted
Radius2 already exists
Radius3 already exists
Radius4 record inserted
Radius5 already exists
Radius6 record inserted
Radius7 record inserted
Radius8 record inserted
Radius9 record inserted
Radius10 record inserted


## Using Raise_Application_Error ===>
====================================

* `RAISE_APPLICATION_ERROR` is a stored procedure which comes in-built with
Oracle software.

* Using this procedure we can associate an error number with the custom error
message and terminates the application.

* Combining both the error number as well as the custom error message we can
compose an error string which looks similar to those default error strings
which are displayed by Oracle engine when an error occurs.

* Syntax :-
# raise_application_error (error_number, message)

* Here the error_number is a negative integer in the range of -20000. to. -
20999 and the message is a character string up to 2048 bytes long.
```

* Example :-

```
DECLARE
    sno student.rollno%type;
    snm student.sname%type;
    crno student.total_course%type;
BEGIN
    sno := &rollno;
    snm := '&sname';
    crno:=&total_courses;
    IF (crno > 3) THEN
        raise_application_error (-20001, 'Total course cannot exceed 3'));
    END IF;
    INSERT into student values(sno, snm, crno);
END;
```