LOOPs In PL-SQL ====>
==========================

* PL-SQL provides the following types of looping statements :-
- Basic Loop
- While Loop
- For Loop

Basic Loop ===>
===================

* This loop statement is the simplest loop structure in PL/SQL.

* The execution block starts with keyword 'LOOP' and ends with the keyword 'END LOOP'.

* Syntax :-

```
LOOP
    statements;
    EXIT WHEN condition;
END LOOP;
```

OR

```
LOOP
    statements;
    IF <condition> THEN
        EXIT;
    ENDIF
END LOOP;
```

WAP to display first 10 natural numbers.

```
declare
i int;
begin
    i := 1;
    loop
      dbms_output.put_line(i);
        exit when i = 10;
        i := i + 1;
    end loop;
end;
```

OR

```
declare
i int;
begin
i := 1;
    loop
        dbms_output.put(i);
        exit when i = 10;
        i := i + 1;
    end loop;
    dbms_output.new_line;
end;
```

# WAP to accept an integer and calculate its factorial.

```
declare
    i int;
    f int := 1;
    begin
      i := &i;
      loop
          exit when i <= 1;
          f := f * i;
          i := i - 1;
      end loop;
      dbms_output.put_line('Fact is ' || f);
 end;
```

# The WHILE Loop ==>
====================

* A WHILE LOOP statement in PL/SQL programming language repeatedly executes a target statement as long as a given condition is true.

* Syntax:

```
WHILE condition LOOP
    statements;
END LOOP;
```

# WAP to print first 10 natural numbers using WHILE loop.

```
declare
i int := 1;
begin
   while i <= 10 loop
       dbms_output.put_line(i);
       i := i + 1;
   end loop;
end;
```

# Redesign the factorial program using WHILE loop.

```
declare
a int;
f int := 1;
begin
   a := &a;
   while a > 1 loop
       f := f * a;
       a := a - 1;
   end loop;
   dbms_output.put_line('Fact is ' || f);
end;
```

# The FOR Loop ==>
==================

* PL/SQL FOR LOOP executes a sequence of statements a specified number of times.

* Syntax:

```
FOR index IN lower_bound .. upper_bound LOOP
    statements;
END LOOP;
```

# Important Points About FOR Loop ==>

- The index is an implicit variable.
- It is local to the FOR LOOP statement.
- In other words, we cannot reference it outside the loop.
- Inside the loop, we can reference index but we cannot change its value.
- After the FOR LOOP statement executes, the index becomes undefined.
- The lower_bound and upper_bound are evaluated once when the FOR LOOP statement starts.
- Their values are stored as temporary PLS_INTEGER values.
- If we modify the values of lower_bound or upper_bound insidethe loop, the change will have no effect because they are evaluated once only before the first loop iteration starts.
- If index is less than upper_bound, index is incremented by one, the statements execute, and control again returns to the top of the loop.
- When index is greater than upper_bound, the loop terminates, and control transfers to the statement after the FOR LOOP statement.

* When lower_bound is greater than upper_bound, the statements do not execute at all.

# WAP to print first 10 natural numbers using FOR loop.

```
declare
a int;
begin
   for a in 1 .. 10 loop
      dbms_output.put_line(a);
   end loop;
end;
```

# WAP to calculate sum of first n natural numbers where n is to be taken from the user.

```
declare
a int;
s int := 0;
begin
   a := &a;
   for i in 1 .. a loop
      s := s + i;
   end loop;
   dbms_output.put_line('Sum is ' || s);
end;
```

# WAP to calculate power of n to p , where n and p are to be taken from the user.

```
declare
n int;
p int;
a int := 1;
begin
   n := &n;
   p := &p;
   for i in 1 .. p loop
      a := a * n;
    end loop;
    dbms_output.put_line(a);
end;
```

# Running FOR Loop In Reverse ===>

* The following shows the structure of the FOR LOOP statement with REVERSE keyword:

* Syntax :-
```
FOR index IN REVERSE lower_bound .. upper_bound LOOP
    statements;
END LOOP;
```

* With the `REVERSE` keyword, the index is set to upper_bound and decreased by one in each loop iteration until it reaches lower_bound.

```
begin
   for i in reverse 1 .. 10 loop
       dbms_output.put_line(i);
   end loop;
end;
```