## UNIT III (GE8151 PROBLEM SOLVING AND PYTHON PROGRAMMING)

# 3.3. Iteration: state, while, for, break, continue, pass

### Iteration (or) Looping Statement

An Iterative statement allows us to execute a statement or group of statement multiple times. Repeated execution of a set of statements is called iteration or looping.

Types of Iterative Statement

1. while loop
2. for loop
3. Nested loop

***1. while loop***

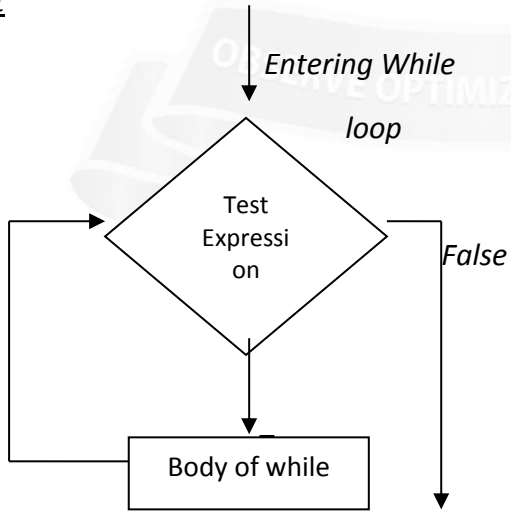
A while loop executes a block of statements again and again until the condition gets false.

The while keyword is followed by test expression and a colon. Following the header is an indented body.

*Syntax*

```
while expression:
        true statements
```

*Flow Chart*



*Example-1:*

1.Write a python program to print the first 100 natural numbers

```
i=1
while (i<=100):
        print(i)
        i=i+1
```

*Result:*

Print numbers from 1 to 100

*Example-2:*

2. Write a python program to find factorial of n numbers.

```
n=int(input("Enter the number:"))
i=1
fact=1
while(i<=n):
        fact=fact*i
        i=i+1
print("The factorial is",fact)
```

*Result:*

Enter the number: 5

The factorial is 120

***2. for loop***

The for loop is used to iterate a sequence of elements (list, tuple, string) for a specified number of times.

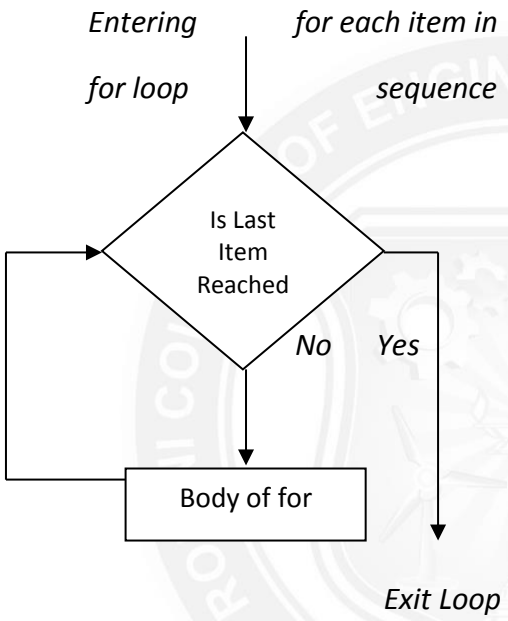
For loop in python starts with the keyword *"for"*followed by an arbitrary variable name,which holds its value in the following sequence objects.

*Syntax*

```
for iterating_variable in sequence:
        statements
```

A sequence represents a list or a tuple or a string. The iterating variable takes the first item in the sequence. Next, the statement block is executed. Each item in the list is assigned to the iterating variable and the statements will get executed until the last item in the sequence get assigned.

*Flow Chart*




*Example-1:*

```
for letter in"python":
    print("The current letter:", letter)
```

*Output:*

```
The current letter: p
The current letter: y
The current letter: t
The current letter: h
The current letter: o
The current letter: n
```

*Example-2:*

```
>>>fruit=[‘apple’, ‘orange’, ‘mango’]
>>>for f in fruit:
        print(“The current fruit”, f)
>>>print(“End of for”)
```

*Output:*

```
The current fruit: apple
The current fruit: orange
The current fruit: mango
End of for
```

***3.Nested loop***

Python Programming allows using one loop inside another loop. For example using a while loop or a for loop inside of another while or for loop.

*Syntax- nested for loop*

```
for iterating_variable in sequence:
        for iterating_variable in sequence:
                Innerloop statements
Outer Loop statements
```

## Unconditional Statement

A situation in which need to exit a loop completely when an external condition is triggered or need to skip a part of the loop. In such situation python provide unconditional statements.

Types of Unconditional looping Statement

1. break statement
2. continue statement
3. pass statement

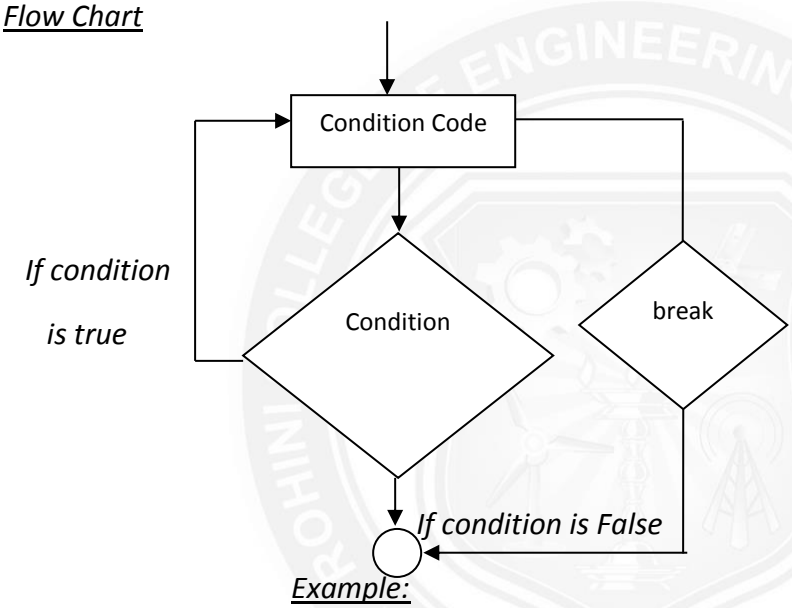
*1.break statement*

A break statement terminates the current loop and transfers the execution to statement immediately following the loop. The break statement is used when some external condition is triggered.

*Syntax*

```
break
```

*Flow Chart*




*Example:*

```
for letter in"python":
    if letter=='h':
        break
        print(letter)
print("bye")
```

*Output:*

```
pyt
bye
```

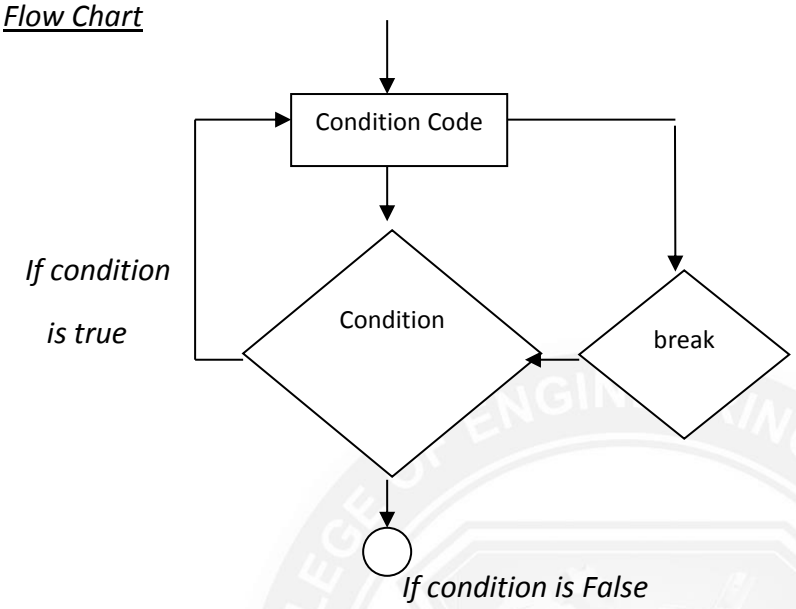
*2. continue statement*

A continue statement returns the control to the beginning of the loop statement. The continue statement rejects all remaining statement and moves back to the top of the loop.

*Syntax*

```
continue
```

*Flow Chart*




*Example:*

```
for letter in"python":
        if letter=='h':
                continue
                print(letter)
    print("bye")
```

*Output:*

```
pyton
bye
```

*3. pass statement*

A pass statement is a null operation, and nothing happens when it executed. It can be used when a statement is required syntactically but the program requires no action.

*Syntax*

```
pass
```

*Example:*

```
for letter in"python":
```

```
        if letter=='h':
                pass
                print(letter)
    print("bye")
```

*Output:*

```
python
bye
```

**The range() function**

If you do need to iterate over a sequence of numbers, the built-in function range() comes in handy. It generates arithmetic progressions:

Eg:

```
# Prints out the numbers
0,1,2,3,4 for x in range(5):
   print(x)
```

This function does not store all the values in memory, it would be inefficient. So it remembers the start, stop, step size and generates the next number on the go.