Unit
3

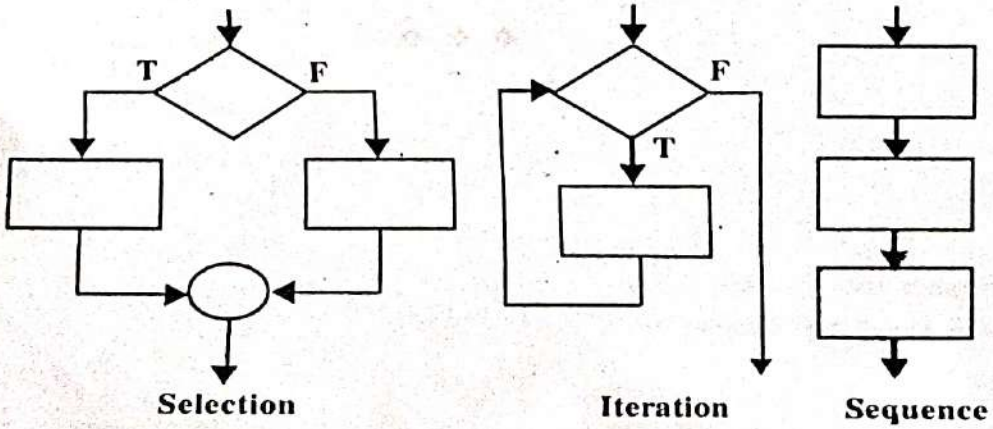
Flow of Control

The control flow of a Python program is regulated by conditional statements, loops, and function cal

3.1 Introduction to Flow of Control

Python has *three* types of control structures:

- **Sequential** - default mode
- **Selection** - used for decisions and branching
- **Repetition** - used for looping, i.e., repeating a piece of code multiple times.



- **Selection**

Selection statements allow programmers to ask questions and then, based on the result, perform different actions. Most programming languages provide two versions of this useful construct: the ifelse and the if

- **Iteration**

For iteration, Python provides a standard while statement and a very powerful for statement. The while statement repeats a body of code as long as a condition is true.

- **Sequential**

Sequential execution is when statements are executed one after another in order. You don't need to do anything more for this to happen.

3.2 Selection

Python provides four conditional statements.

3.2.1 If Statements

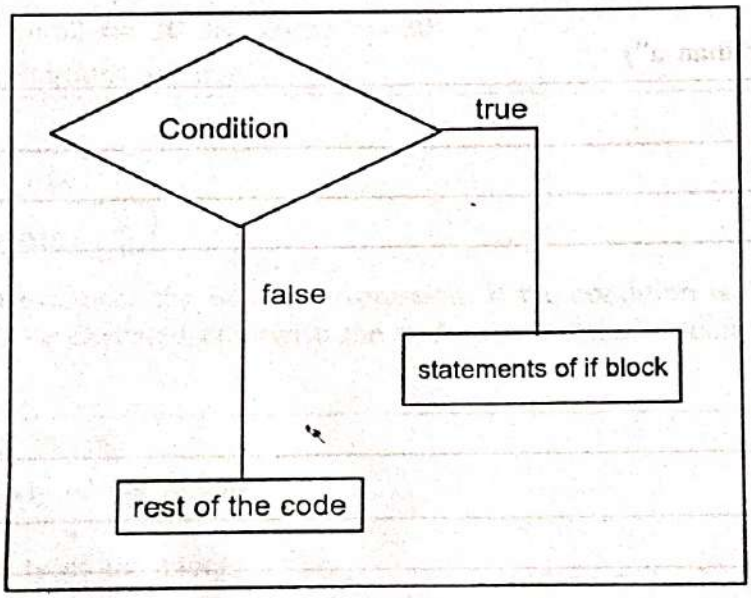
Python if statement decides whether certain statements need to be executed or not. It checks for a given condition, if the condition is true, then the set of code present inside the " if " block will be executed otherwise not.

- **Syntax :**

```
If ( EXPRESSION == TRUE ):
Block of code
else:
Block of code
```

Here, the condition will be evaluated to a Boolean expression (true or false). If the condition is true, then the statement or program present inside the " if " block will be executed and if the condition is false, then the statements or program present inside the "else" block will be executed.

- **Flow chart**



Let's see some examples of " if " statements.

Example : 1

```
num = 5
if (num < 10):
    print("Num is smaller than 10")
```

Output :

```
Num is smaller than 10.
```

In the above example, we declared a variable called 'Num' with the value as 5 and the " if " state is checking whether the number is lesser than 10 or not. If the condition is true then a set of staten inside the if block will be executed.

Example : 2

```
a = 7
b = 0
if (a > b):
    print("a is greater than b")
```

Output :

```
a is greater than b
```

In the above example, we are checking the relationship between a and b using the greater than operator in the if condition. If "a" is greater than "b" then we will get the above output.

Example : 3

```
a = 0
b = 7
if (b > a):
    print("b is greater than a")
```

Output :

```
b is greater than a.
```

Example : 4

```
a = 7
b = 0
if (a):
    print("true")
```

Output :

```
true
```

If you observe, in the above example, we are not using or evaluating any condition in the "if" statement. Always remember that in any programming language, the positive integer will be treated as true value and an integer which is less than 0 or equal to 0 will be treated as false.

Here the value of a is 7 which is positive, hence it prints true in the console output.

Python If Statement In One Line

In Python, it is permissible to write if statement in one line.

Syntax :

```
if (condition): #Set of statements to execute if condition in true
```

There can be multiple statements as well, you just need to separate it by a semicolon (;)

Syntax :

```
if (condition) : statement 1; statement 2; statement 3;...;statement n
```

If the condition is true, then execute statement 1, statement 2 and so on up to statement n.

In case if the condition is false then none of the statements will be executed.

Example : 1

```
num = 7
if (num > 0): print("Number is greater than Zero")
```

Output :

```
Number is greater than Zero
```

Multiple Conditions In If Statements

It's not that you can only write one condition inside an "if" statement, we can also evaluate multiple conditions in an "if" statement like below.

Example : 1

```
num1 = 10
num2 = 20
num3 = 30
if (num1 == 10 and num2 == 20 and num3 == 30):
    print("All the conditions are true")
```

Output :

```
All the conditions are true
```

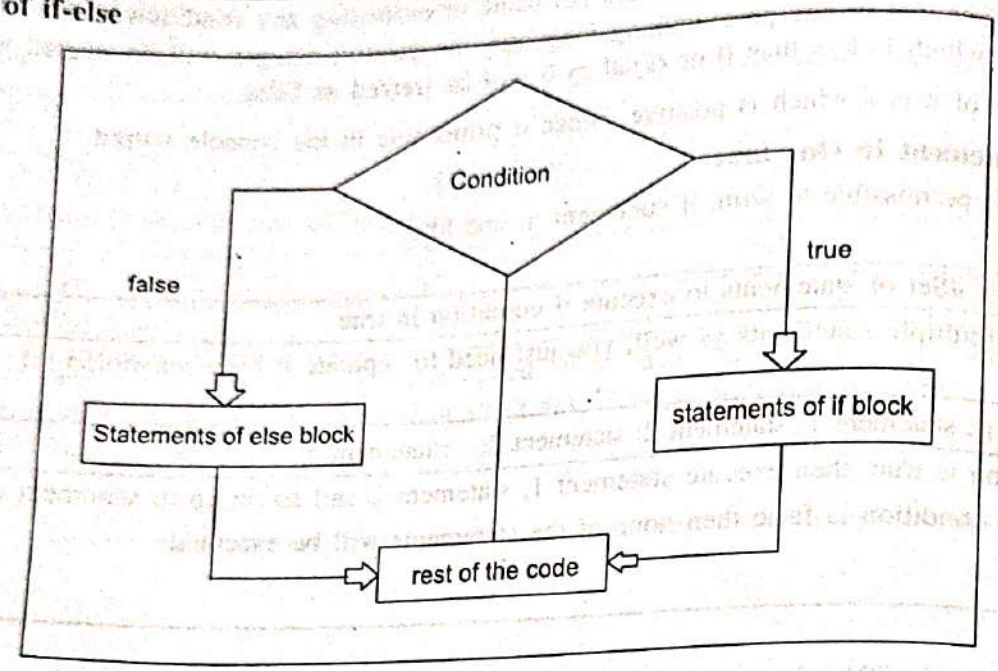
3.2.2 If-Else Statements

if-else statement evaluates the Boolean expression. If the condition is TRUE then, the code present in the " if " block will be executed otherwise the code of the "else" block will be executed

Syntax :

```
If (EXPRESSION == TRUE) :
        Statement (Body of the block)
else:
        Statement (Body of the block)
```

Flowchart of if-else



Example : 1

```
num = 5
if(num > 10):
    print("number is greater than 10")
else:
    print("number is less than 10")
```

Output :

```
number is less than 10.
```

In the above example, we have declared a variable called 'num' with the value as 5 and in the "if" statement we are checking if the number is greater than 5 or not.

If the number is greater than 5 then, the block of code inside the "if" block will be executed and if the condition fails then the block of code present inside the "else" block will be executed.

Example : 2

```
a = 7
b = 0
if (a > b):
    print("a is greater than b")
else:
    print("b is greater than a")
```

Output :

```
a is greater than b
```

the above code if "a" is greater than "b" then the statements present inside the "if" block will be ecuted and the statements present inside the "else" block will be skipped.

ample : 3

```
= 7
= 0
f (a < b):
    print( "a is smaller than b" )
lse:
    print( "b is smaller than a" )
```

tput :

```
is smaller than a
```

the above code, "a" is smaller than "b", hence statements present inside the "else" block will be ecuted and statements present inside the "if" block will be skipped.

-else Statements In One Line

ntax :

```
f (condition): #Set of statement to execute if condition is true
lse: #Set of statement to execute if condition is false
```

here can be multiple statements as well, you just need to separate it by a semicolon (;)

ntax :

```
f (condition): statement 1; statement 2; statement 3;...;statement n
lse: statement 1; statement 2; statement 3;...;statement n
```

xample : 1

```
num = 7
if (num > 0): print("Number is greater than Zero")
else : print("Number is smaller than Zero")
```

utput :

```
Number is smaller than Zero
```

xample : 2

```
if ('a' in 'fruits'): print("Apple"); print("Orange")
else: print("Mango"); print("Grapes")
```

utput :

```
Mango
Grapes
```

3.2.3 Elif Statements

In Python, we have one more conditional statement called "elif" statements. "elif" statement is used to check multiple conditions only if the given condition is false. It's similar to an "if-else" statement and the only difference is that in "else" we will not check the condition but in "elif" we will check the condition.

"elif" statements are similar to "if-else" statements but "elif" statements evaluate multiple conditions.

Syntax :

```
if (condition):
        #Set of statement to execute if condition is true
elif (condition):
        #Set of statements to be executed when if condition is false and elif condition is true
else:
    #Set of statement to be executed when both if and elif conditions are false
```

Example : 1

```
num = 10
if (num == 0):
   print("Number is Zero")
elif (num > 5):
    print("Number is greater than 5")
else:
    print("Number is smaller than 5")
```

Output :

```
Number is greater than 5
```

In the above example we have declared a variable called 'num' with the value as 10, and in the if statement we are checking the condition if the condition becomes true. Then the block of code present inside the "if" condition will be executed.

If the condition becomes false then it will check the "elif" condition if the condition becomes true, then a block of code present inside the "elif" statement will be executed.

If it is false then a block of code present inside the "else" statement will be executed.

Example : 2

```
num = -7
if (num > 0):
        print("Number is positive")
elif (num < 0):
        print("Number is negative")
else:
        print("Number is Zero")
```

Output :

```
Number is negative
```

In the above example, first, we are assigning the value 7 to a variable called num. The controller will come to the "if" statement and evaluate the Boolean expression num > 0 but the number is not greater than zero hence if block will be skipped.

As the if condition is evaluated to false the controller will come to the "elif" statement and evaluate the Boolean expression num < 0, hence in our case number is less than zero hence 'Number is negative' is printed.

In case both the "if" and "elif" condition is evaluated to false then a set of statements present inside the "else" block will be executed.

Elif Statements In One Line

The elif block can also be written as below.

Syntax :

```
if (condition): #Set of statement to execute if condition is true
elif (condition1): #Set of statement to execute if condition1 is true
else: #Set of statement to execute if condition and condition1 is false
```

There can be multiple statements as well; you just need to separate it by a semicolon (:)

Syntax :

```
if (condition): statement 1; statement 2; statement 3;...:statement n
elif (condition): statement 1; statement 2; statement 3;...:statement n
else: statement 1; statement 2; statement 3;...;statement n
```

Example : 1

```
num = 7
if (num < 0): print("Number is smaller than Zero")
elif (num > 0): print("Number is greater than Zero")
else: print("Number is Zero")
```

Output :

```
Number is greater than Zero
```

Example : 2

```
if ('a' in 'fruits'): print("Apple"); print("Orange")
elif ('e' in 'fruits'): print("Mango"); print("Grapes")
else: print("No fruits available")
```

Output :

```
No fruits available
```

3.2.4 Nested if-Else Statements (3)

Nested "if-else" statements mean that an "if" statement or "if-else" statement is present inside another if or if-else block. Python provides this feature as well, this in turn will help us to check multiple conditions in a given program.

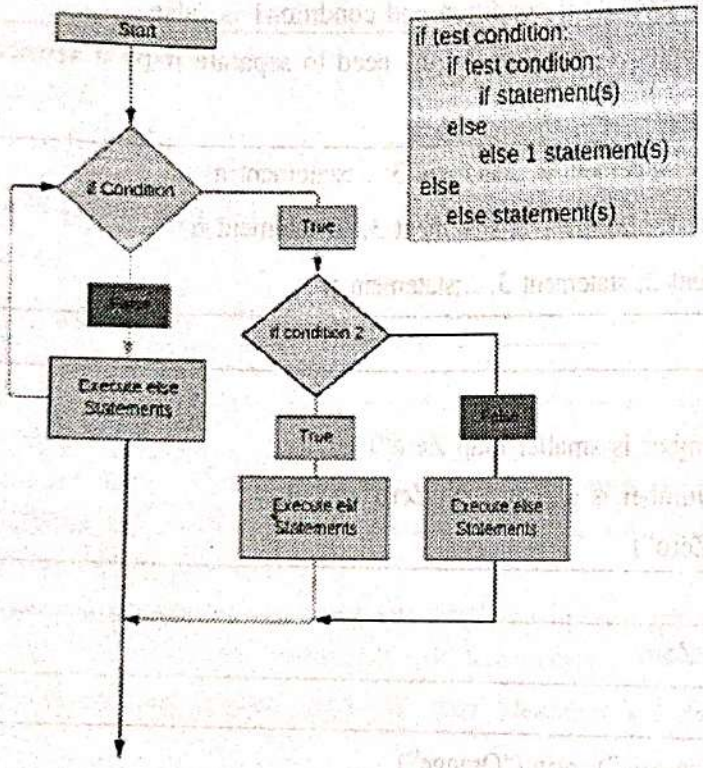
An "if" statement is present inside another "if" statement which is present inside another "if" statements and so on.

Nested if-else Syntax:

```
if(condition):
        #Statements to execute if condition is true
        if(condition):
                #Statements to execute if condition is true
        else:
                #Statements to execute if condition is false
else:
        #Statements to execute if condition is false
```

Here we have included the "if-else" block inside an if block, you can also include an "if-else" block inside "else" block.

Let's look at the nested if-else statement



Example: nested if-else statement

```
score=74;
if score >= 90:
    print('A')
else:
    if score >=80:
```

```
    print('B')
else:
    if score >= 70:
        print('C')
    else:
        if score >= 60:
            print('D')
        else:
            print('F')
```

Output :

```
C
```

3.2.5 Elif Ladder

We have seen about the "elif" statements but what is this elif ladder? As the name itself suggests a program that contains a ladder of "elif" statements or "elif" statements are structured in the form of a ladder.

This statement is used to test multiple expressions.

Syntax :

```
if (condition):
        #Set of statement to execute
elif (condition):
        #Set of statements to be executed
elif (condition):
        #Set of statements to be executed
elif (condition):
        #Set of statements to be executed
else:
    #Set of statement to be executed when all if and elif conditions are false
```

Flowchart of elif ladder

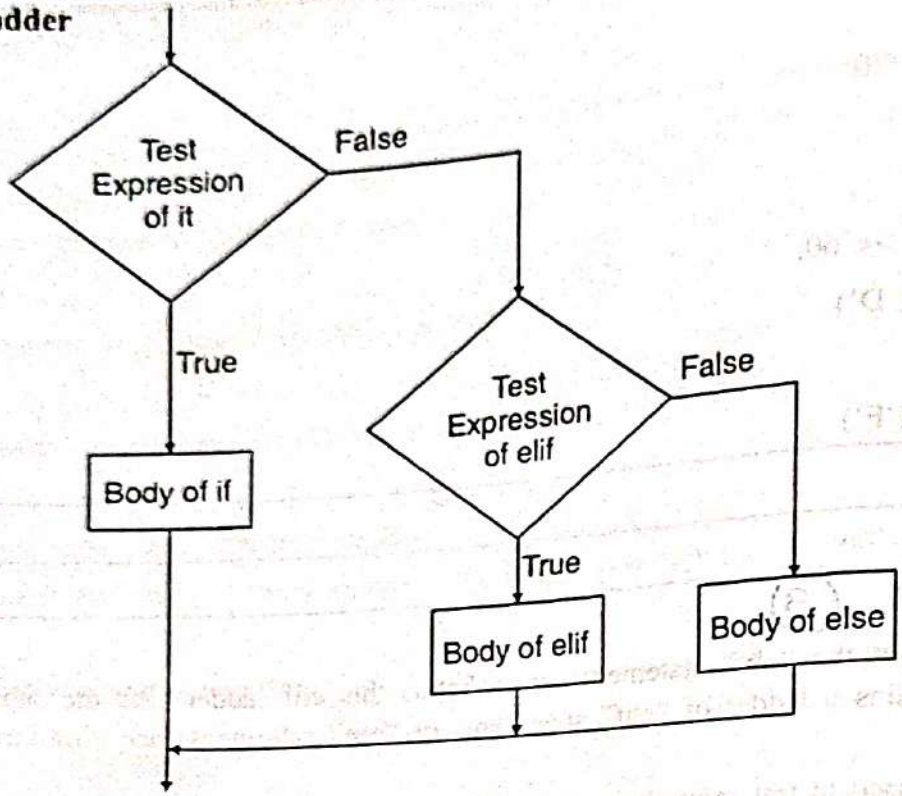


Fig: Operation of if...elif...else statement

Example : 1

```
score=74;
if score >= 90:
    print('A')
elif score >=80:
    print('B')
elif score >= 70:
    print('C')
elif score >= 60:
    print('D')
else:
    print('F')
```

Output :

```
C
```

The above example describes the elif ladder. Firstly, the control enters the "if" statement and evaluates the condition if the condition is true then the set of statements present inside the if block will be executed else it will be skipped and the controller will come to the first elif block and evaluate the condition.

A similar process will continue for all the remaining "elif" statements and in case all if and elif conditions are evaluated to false then the else block will be executed.

In Python, we can write "if" statements, "if-else" statements and "elif" statements in one line without worrying about the indentation.

We know

Example : 2

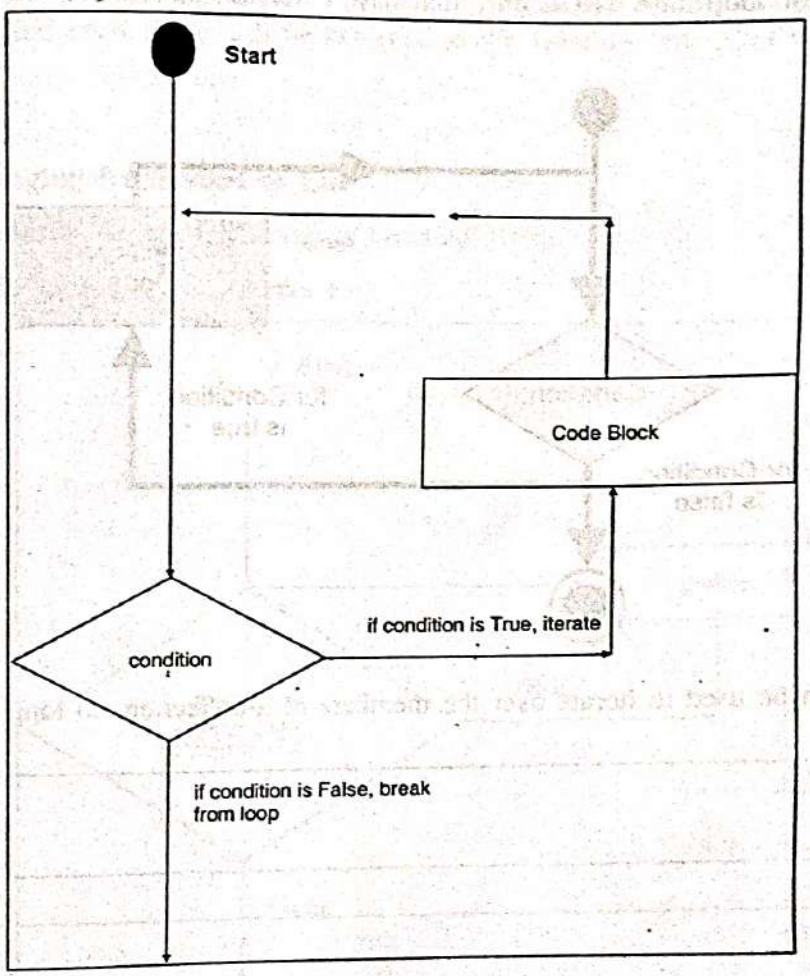
```
a = 10
if (a): print( " The given value of a: " ); print(a)
```

Output :

```
The given value of a: 10
```

3.3 Repetition

In Python, statements are executed in a sequential manner i.e., if our code is made up of several lines of code, then execution will start at the first line, followed by the second, and so on.However, there will be cases where we may want a block of code to execute several times until a condition is met. Thanks to loop statements, we can do just that.Given below is a flowchart that illustrates how a loop statement works.



Flowchart of a Loop Statement

Based on the above diagram, a Python program will start at **Start**[circle], and the execution will proceed to the condition statement [Diamond], if the condition is TRUE, then the program will execute the code block.

Execution will proceed again to the condition statement and the same process continues each time the condition is TRUE. It only breaks out of the loop or stops executing the code block if the condition is FALSE, and in this case, the program will continue execution sequentially.

Python has two types of Loops.

Loop type	Description
for loop	Is an iterator based loop, which steps through the items of iterable objects like lists, tuples, string and executes a piece of code repeatedly for a number of times, based on the number of items in that iterable object.
while loop	Executes a block of statements repeatedly as long as the condition is TRUE.

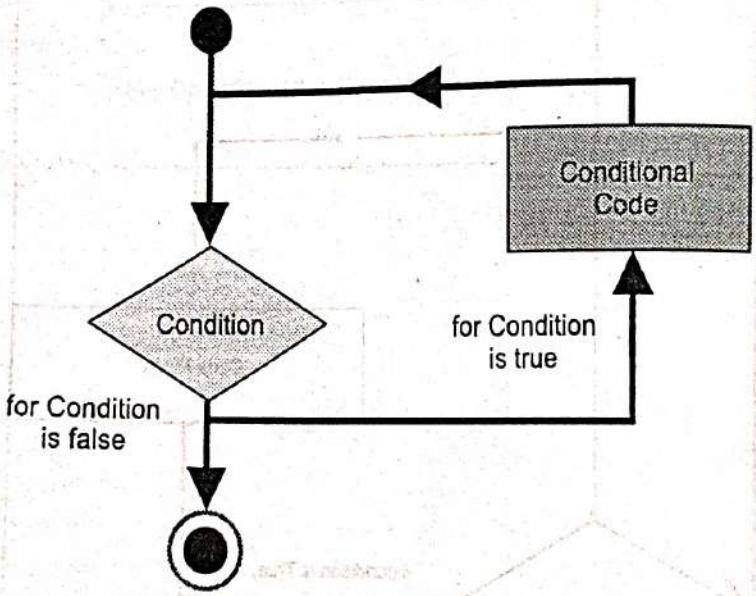
For Loop

the for statement, can be used in conjunction with many of the Python collections. The **for loop** is zero indexed and has the following syntax.

```
for in n :
```

The condition in the **for loop** stays TRUE only if it hasn't iterated through all the items in the iterable object(n).

Flowchart : for loop



The for statement can be used to iterate over the members of a collection, so long as the collection is a sequence.

```
for item in [1,3,6,2,5]:
    print(item)
```

Output :

```
1
3
6
2
5
```

assigns the variable item to be each successive value in the list [1,3,6,2,5]. The body of the iteration is then executed. This works for any collection that is a sequence (lists, tuples, and strings).

A common use of the for statement is to implement definite iteration over a range of values. The statement

```
for item in range(5):
    print(item**2)
```

Output :

```
0
1
4
9
16
```

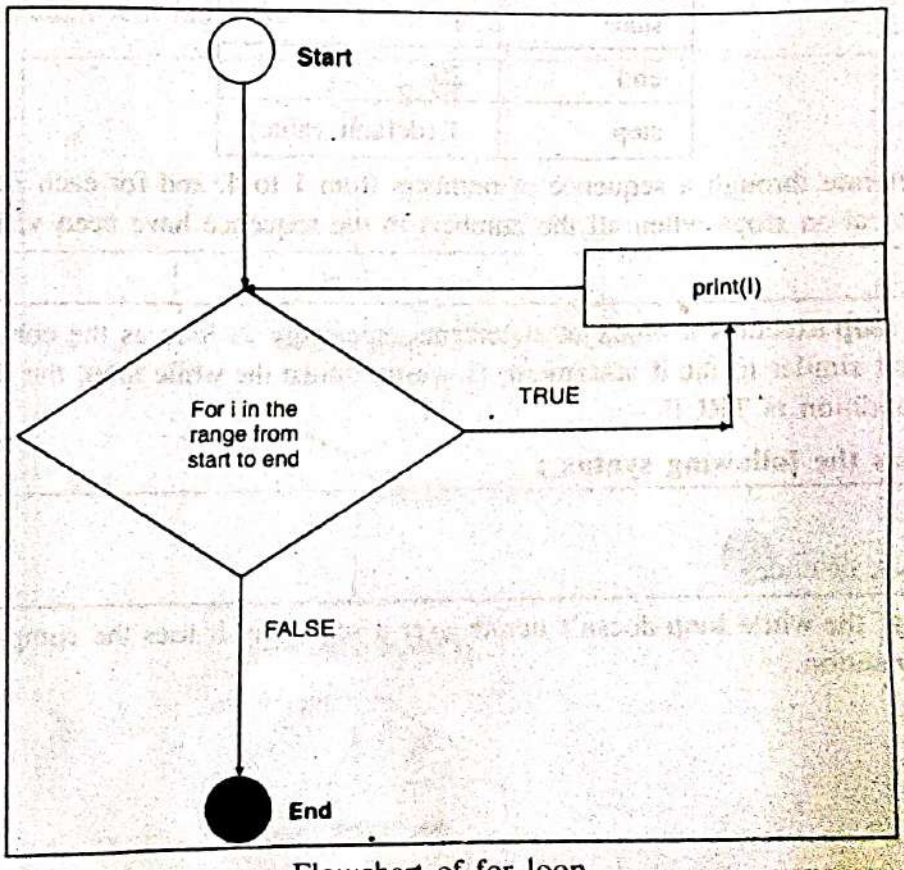
will perform the print function five times. The range function will return a range object representing the sequence 0,1,2,3,4 and each value will be assigned to the variable item. This value is then squared and printed.

Example 1 :

Print Numbers ranging from Start to End

To achieve this, we will use the Python range function.

This is how the flowchart will look like :



Flowchart of for loop

Example: Using for loop in to print range of numbers

```
start = int(input('Enter a start number:'))
end = int(input('Enter an end number: '))

for i in range(start, end+1):
    print(i)
```

Output :

```
Enter a start number:1
Enter an end number: 4
1
2
3
4
```

In the above example, we used Python range, which is a function that returns a sequence of numbers, starting from a **start** number (0 by default), increments by a **step** (1 by default), and stops before an **end** number.

Parameters and Values for the Python range function

Parameters	Value
start	1
end	20
step	1 (default value)

So, **for loop** will iterate through a sequence of numbers from 1 to 4, and for each iteration, it will print the number. The iteration stops when all the numbers in the sequence have been visited.

While Loop

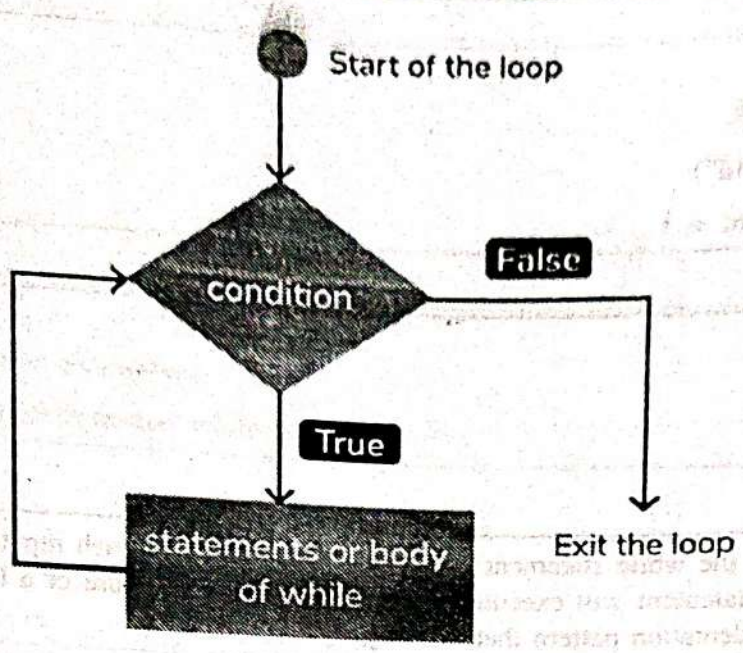
The Python **while loop** executes a block of statements repeatedly as long as the condition is TRUE. We notice that it is a bit similar to the **if statement**. However, unlike the **while loop**, the if statement executes only once if its condition is TRUE.

The while loop has the following syntax :

```
While condition :
    expression(block of code)
```

Unlike the **for loop**, the **while loop** doesn't iterate over a sequence. It uses the comparison operators and booleans for its condition.

Flowchart: While loop

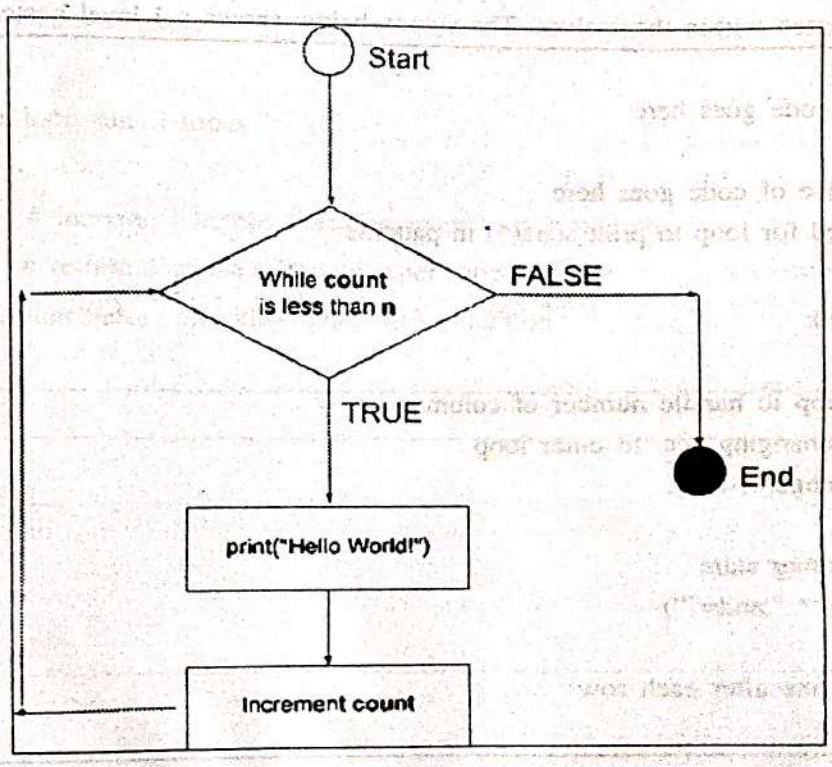


Let's look at some examples to better understand how it is used.

Example 1 : Print "Hello World!" a count number of times

The **while loop** checks the condition(count < n), each time when it's TRUE, it prints our **"Hello world!"** and increments the count.

This is how the flowchart will look like:



Program :

```
count = 1
while count <= 5:
print("Hello, world")
    count = count + 1
```

Output :

```
Hello, world
Hello, world
Hello, world
Hello, world
Hello, world
```

The condition on the while statement is evaluated at the start of each repetition. If the condition is True, the body of the statement will execute. It is easy to see the structure of a Python while statement due to the mandatory indentation pattern that the language enforces.

The while statement is a very general-purpose iterative structure that we will use in a number of different algorithms.

Nested Loop

The cool thing about Python loops is that they can be nested i.e. we can use one or more loops inside another loop. This enables us to solve even more complex problems.

1) Nesting for Loops

for loops can be nested within themselves. The syntax below shows a 1-level nested for loop.

```
for in n:
    # piece of code goes here
    for in n:
        #  piece of code goes here
```

Example: Use nested for loop to print stars(*) in patterns

```
n=5
for i in range(0, n):

    # inner loop to handle number of columns
    # values changing acc. to outer loop
    for j in range(0, i+1):

        # printing stars
        print("* ",end="")

    # ending line after each row
    print("\r")
```

Output

```
*
* *
* * *
* * * *
* * * * *
```

2) Nesting While Loops

While loops can be nested within themselves.

The syntax below shows a 1-level nested while loop.

```
while condition:
    # piece of code goes here
    while condition:
        # piece of code goes here
```

Example 3 : Use nested while loop to print stars(*) in patterns (4)

```
n=5
i = 0 # initialize to zero for outer loop
j = 0 # initialize to zero for inner loop
while i<= n:
        # outer loop runs n times
        while j < i:
            # inner loop runs i times
print("* ",end="")
            j += 1  # increment before checking inner loop condition
        j = 0       # re-initialize after leaving inner loop
i += 1      # increment before checking outer loop condition
print('')
```

Output :

```
*
* *
* * *
* * * *
* * * * *
```

Example : Number Patterns Using nested loop.

```
n=5
# initialising starting number
num = 1

    # outer loop to handle number of rows
for i in range(0, n):

        # re assigning num
        num = 1

        # inner loop to handle number of columns
            # values changing acc. to outer loop
        for j in range(0, i+1):

                # printing number
            print(num, end=" ")

            # incrementing number at each column
            num = num + 1

        # ending line after each row
        print("\r")
```

Output :

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

Example: Character Patterns Using nested loop.

```
n=5
num = 65

# outer loop to handle number of rows
# 5 in this case
for i in range(0, n):

    # inner loop to handle number of columns
    # values changing acc. to outer loop
    for j in range(0, i+1):

        # explicitely converting to char
        ch = chr(num)

        # printing char value
        print(ch, end=" ")

    # incrementing number
    num = num + 1

    # ending line after each row
    print("\r")
```

Output :

```
A
B B
C CC
D DDD
E EEEE
```

3.4 Break and Continue Statements

Control statements in python are used to control the order of execution of the program based on the values and logic.

1) Continue Statement :

When the program encounters a continue statement, it will skip the statements which are present after the continue statement inside the loop and proceed with the next iterations.

Syntax :

```
continue
```

Example :

```
for char in 'Python':
    if (char == 'y'):
        continue
    print("Current character: ", char)
```

Output :

```
Current character: P
Current character: t
Current character: h
Current character: o
Current character: n
```

In the above example during the second iteration if the condition evaluates to true, then it will execute the continue statement. So whatever statements are present below, for loop will be skipped, hence letter 'y' is not printed.

2) Break Statement:

The break statement is used to terminate the loop containing it, the control of the program will come out of that loop.

Syntax :

```
break
```

Example :

```
for char in 'Python':
    if (char == 'h'):
        break
    print("Current character: ", char)
```

Output :

```
Current character: P
Current character: y
Current character: t
```

3) Pass Statement :

Pass statement is python is a null operation, which is used when the statement is required syntactically.

Syntax :

```
pass
```

Example :

```
for char in 'Python':
    if (char == 'h'):
        pass
print("Current character: ", char)
```

Output :

```
Current character: P
Current character: y
Current character: t
Current character: h
Current character: o
Current character: n
```