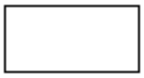
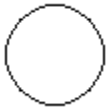
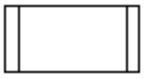
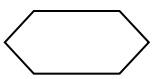
## 1.2 Flow Control:

A programs control flow is the order in which the program's code executes. The control flow of a python program is regulated by conditional statements, loops, and function calls. Real strength of program is not just running or executing instructions sequentially one after the other, but skipping and iterating instructions based on the conditions.Flow control statements can decide which Python instructions to execute under which conditions.

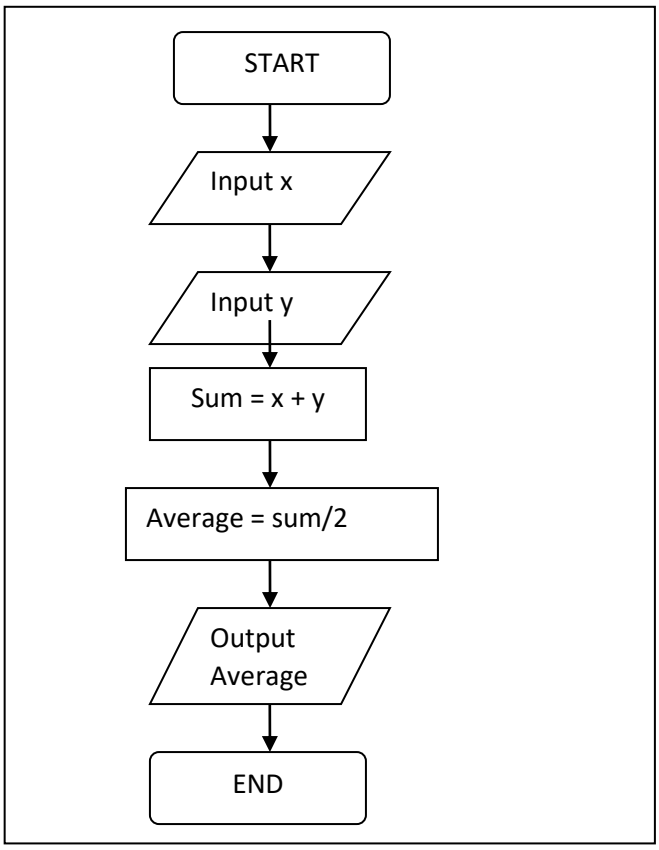
**Flow Chart:**Flowchart is a graphical representation of an algorithm. Flowcharts use special shapes to represent different types of actions or steps in a process. Lines and arrows show the sequence of the steps, and the relationships among them. In flowchart symbols or shapes are used to represent the data flow, operations, process and recourses used to complete a given task. Table1.1 below shows the commonly used flowchart symbols and their purpose.

**Table 1.1 : Flowchart Symbols**

| Flowchart Symbol | Purpose |
|---|---|
|  | **Start/End:** The terminator (Ellipse like) symbol is used to represent the starting or ending point. It usually contains the word "Start" or "End." |
|  | **Action or Process :**A rectangular box is used to represent a process defined in an algorithm |
|  | **Decision:** A diamond symbol is used to represent the decision or branching point. |
|  | **Input/output:** Parallelogram symbol is used to represent an input or output operation. |
|  | **Connector:** Circle symbol is used to connect one point of flow to another point. |
|  | **Flow Line:** Used to show the direction of a control flow and to connect the various flowchart symbols. |
|  | **Subroutine:** The symbol is used to represent the sub functions in a given program. |
|  | **Looping:** The symbol is used to represent the looping or repetition of a given set of statements. |

## **Examples:** Write a flowchart for the following

### 1. To find the average of two numbers

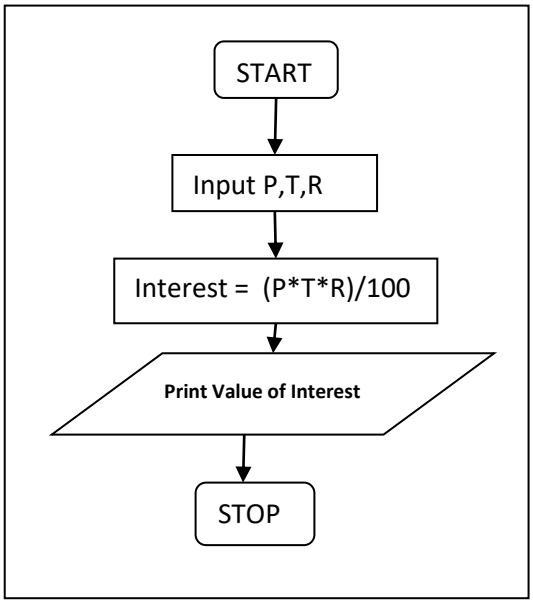



**Python Program:**

```
# Read two numbers from the user
n1,n2 = input("Enter two numbers").split()
# Convert the numbers into integer
n1 = int(n1)
n2 = int(n2)
sum = n1 + n2
avg = sum/2
print("Average : ",avg)
```

```
Enter two numbers3 4
Average :  3.5
```

## 2. To find the simple interest or interest given the value of P,T and R.




## Python Program:

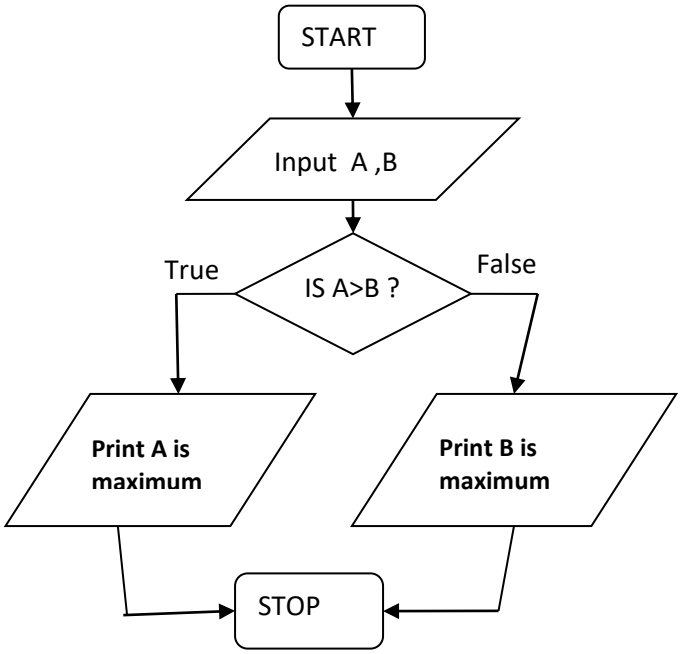
```
# Read the value of P,T and R
P = float(input("Enter the Principle Amount in Rupees"))
T = float(input("Enter the time period in years"))
R = float(input("Enter the Rate of interest per anum"))

SI = (P*T*R)/100

print("The Simple Interest is :",SI)
```

```
Enter the Principle Amount in Rupees25000
Enter the time period in years3.2
Enter the Rate of interest per anum1.49
The Simple Interest is : 1192.0
```

## 3. To find the maximum of two numbers

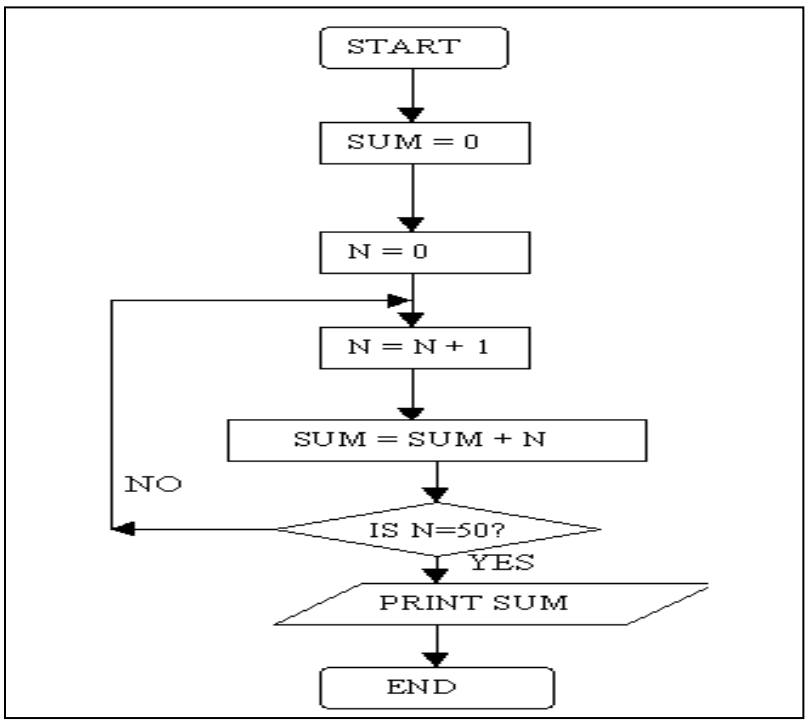



**Python Program:**

```
# Read the two numbers
n1 = int(input("Enter the first number"))
n2 = int(input("Enter the second Number"))
if n1 > n2 :# Condition Checking
    print("The maximum of two numbers is: ",n1)
else :
    print("The maximum of two numbers is: ",n2)
```

```
Enter the first number40
Enter the second Number20
The maximum of two numbers is:  40
```

## 4. To find the sum of first 50 natural numbers.



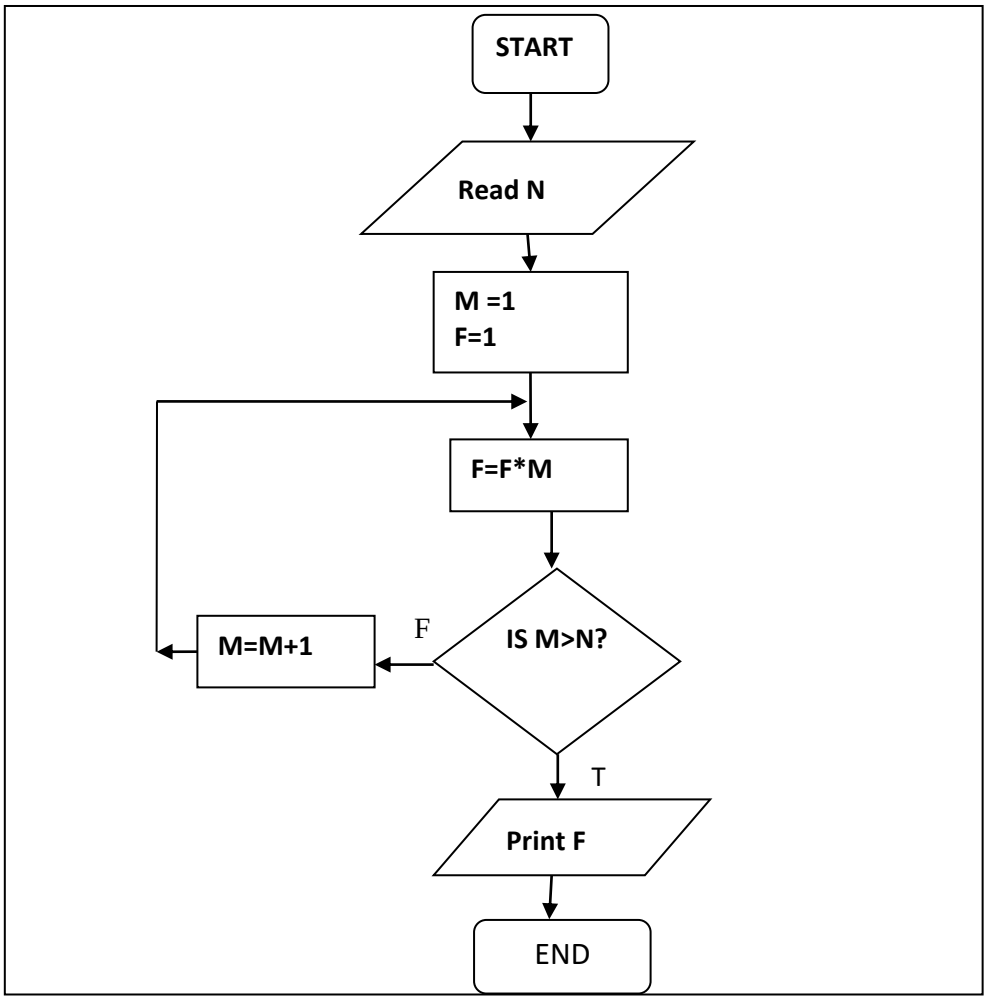

**Python Program :**

```
# Initialisation of sum and n
sum = 0
n = 0
# Loop Starts
while n<50:
    n= n+1
    sum = sum + n
print("The sum of first 50 natural numbers is : ",sum)


```

```
The sum of first 50 natural numbers is :  1275
```

## 5. To find the factorial of a given number N.




**Python Program:**

```python
n = int(input("Enter the value of n: "))
m = 1
f = 1
# Loop starts
while m<=n:
    f= f*m
    m= m+1
print("The factorial of given number :",f)

```

```
Enter the value of n: 5
The factorial of given number : 120
```

**Boolean Values:** A Boolean value is either true or false. It is named after the British mathematician, George Boole, who first formulated Boolean algebra. In Python the two Boolean Values are True and False and the Python type is bool. Enter the following into the Python shell and observe the output.

**Code Snippet1:**

```
1 type(True)
```

```
bool
```

```
1 type(False)
```

```
bool
```

```
1 type(true)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-5-6f4d8242c3d0> in <module>
----> 1 type(true)

NameError: name 'true' is not defined
```

```
1 type(false)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-6-5d8f55c25b5e> in <module>
----> 1 type(false)

NameError: name 'false' is not defined
```

**Code Snippet2:**

```
1 ham = True
```

```
1 ham
```

```
True
```

```
1 true
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-9-724ba28f4a9a> in <module>
----> 1 true

NameError: name 'true' is not defined
```

A Boolean expression is an expression that evaluated to produce a result which is a Boolean value. For example, the operator == tests if two values are equal. It produces (or yields) a Boolean value:

```
1 5 ==(1+4)
```

```
True
```

```
1 5 == 6
```

```
False
```

```
1 j="hel"
```

```
1 j+"lo" == "hello"
```

```
True
```

In the first statement the two operands evaluate to equal values, so the expression evaluates to **True**; in the second statement, 5 is not equal to 6 we get **False.**

Comparison Operators

Comparison operators compare two values and evaluate down to a single Boolean value. The **==** operator is one of six common comparison operators which all produce a bool result. Table lists all the comparison operators:

| **Operator** | **Meaning** |
|---|---|
| == | Equal to |
| != | Not Equal to |
| < | Less than |
| > | Greater than |
| <= | Les than or equal to |
| >= | Greater than or equal to |

Comparison operators evaluate to True or False depending on the values we provide to them .

```
1 55==55
```

```
True
```

```
1 55 ==79
```

```
False
```

```
1 2!=3
```

```
True
```

```
1 4!=4
```

```
False
```

Based on the above observations it is clear that == (equal) evaluates to True when the value on both sides are the same , and != (not equal to ) evaluates to True when the two values are different . T Equal to and Not equal to operators can work with values of any data type.

```
1 "Ceaser"=="Ceaser"
```

```
True
```

```
1 "Ceaser"=="ceaser"
```

```
False
```

```
1 "Dog"!="Cat"
```

```
True
```

```
1 True == True
```

```
True
```

```
1 True != False
```

```
True
```

```
1 15 ==15.0
```

```
True
```

```
1 15 == "15"
```

```
False
```

The other comparison operators like <, >, <= and >= work properly only with integer and floating-point values.

```
1 12<13
```

```
True
```

```
1 55.0>66.789
```

```
False
```

```
1 "tag"<= 2
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-29-582a7fc0e527> in <module>
----> 1 "tag"<= 2

TypeError: '<=' not supported between instances of 'str' and 'int'
```

```
1 25>=9
```

```
True
```

## Difference between == and = Operator

| = | == |
|---|---|
| It is an assignment operator | It is a comparison operator |
| It is used for assigning the value to a variable | It is used for comparing two values.<br>It returns 1 if both the value is equal otherwise returns 0 |
| Constant term cannot be place on left hand side<br>**Example:**<br>1= x; is invalid | Constant term can be placed in the left-hand side.<br>Example: 1 ==1 is valid and return 1 |

## Boolean Operators:

Boolean operators evaluate the expression to Boolean Values True/False. Python supports three Boolean operators and, or & not. Based on the number of operands required they can be classified into Binary Boolean operators and Unary Boolean Operators.

### **Binary Boolean operators** : and & or

Since both **and & or** operators takes two operands, they are considered as binary operators. The **and** operator returns true value if both operands are true and return false otherwise. While or operator returns false when both operands are false and returns **true** otherwise.

```
1 True and True
```

```
True
```

```
1 True and False
```

```
False
```

```
1 False and True
```

```
False
```

```
1 False and False
```

```
False
```

```
1 False or False
```

```
False
```

```
1 True or True
```

```
True
```

```
1 True or False
```

```
True
```

```
1 False or True
```

Following truth tables illustrates all possible logical combinations and values for OR and AND Boolean binary operators.

**Table:** Truth Table for AND

| Op1 | Op2 | Op1 and Op2 |
|---|---|---|
| True | False | False |
| False | True | False |
| False | False | False |
| True | True | True |

**Table :**Truth Table for OR

| Op1 | Op2 | Op1 or Op2 |
|---|---|---|
| True | False | True |
| False | True | True |
| False | False | False |
| True | True | True |

**Not operator**: It is a unary operator and evaluates the expression to opposite value true or false as illustrated below :

```
1 not True
```

```
False
```

```
1 not False
```

```
True
```

```
1 not True
```

```
False
```

```
1 not not not not True
```

```
True
```

Truth Table for **not** operator:

| op | not op |
|---|---|
| True | False |
| False | True |

Examples for Mixing Boolean and Comparison Operators: Boolean operators and comparison operators can be used in combination as illustrated below :

```
x = 10
y = 20
```

```
x<y and x>y
```

```
False
```

```
(x<y) and (x!=y) or (x*2) and (x<20 or y<20)
```

```
True
```

```
2 + 2 == 4 and not 2 + 2 == 5 and 2 * 2 == 2 + 2
```

```
True
```

```
5*7 + 8 == 7 or   not 5+7 == 10 and  5*4 == 20
```

```
True
```

## Elements of Flow Control:

**Program Execution:** Program is a set of statements.The statements in Python are executed sequentially one after the other from the beginning of the program to the end of a program. This kind of program execution where in the control flows sequentially is termed as *sequential execution of program*. In some context it becomes inevitable to skip certain set of statements in order to execute another set of statements. This type of program execution is termed as *selective execution*. In Python language the selective execution is achieved using the statements known as *Branching statements or Conditional control structures or statements.*Also, in certain situations a set of statements has to be executed *repeatedly* for given number of times. This is achieved with the help of *Loop Control constructs or statement*s. The mechanism of repeating one or more statements execution either for a fixed number of times or until certain condition is satisfied is termed as *Looping*.

Flow control statements often starts with condition followed by a block of code called clause.

## Conditions:

Conditions are Boolean expressions with the boolean values True or False. Flow control statements decides what to do based on the condition whether it is true or false.

## Blocks of Code /Clause:

The set of more than one statements grouped with same indentation so that they are syntactically equivalent to a single statement is known as Block or Compound Statement. One can tell when a block begins and ends based on indentation of the statements . Following are three rules for blocks :

1. Blocks begin when the indentation increases
2. Blocks can have nested blocks
3. Blocks end when the indentation decreases to zero

**Example1 :**

```
x = int(input("Enter a number: "))
if x >=10 :
    x = x+25
    y = x
    print(x,y)
print("Next Statement")
```

**Example 2: Nested Blocks**

```
n = int(input("Enter a number of your choice"))
if n>0:
    print("Positive")
    if n%2==0:
        print("Multiple of Two")
print("End")
```

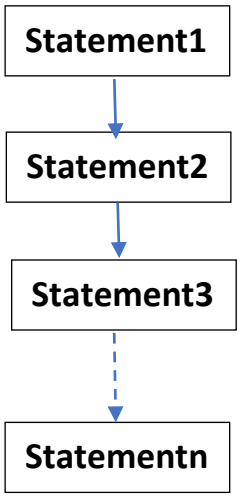
The control and flow of a program in any programming language can be broadly classified into three categories:

1. Sequential
2. Selection or Branching
3. Iteration or looping

In Python program, statements can be executed sequentially , selectively or iteratively . The following section discuses the program control flow constructs in brief .

1. **Sequential:**

   Here the program are executed in a sequential order or in a linear fashion , one after another without skipping or any jump in the program.As illustrated in the figure below Statement1 will be executed first followed by statement 2 , statement3 ,------statementn.




## 2. Selection or Branching Statements:

These are the statements which will transfer control flow from one place to another place, so as **to execute** a set of instructions ,*if some condition is met* or, **to skip** the execution of the statements *if the condition is not me*t. They are also known as Conditional selection statements or decision statements. They specify the order in which computation are performed. The Python programming statement which provides ***two paths to control flow*** to follow one for the true condition and the other for the false condition as shown in figure is called two-way selection statement.

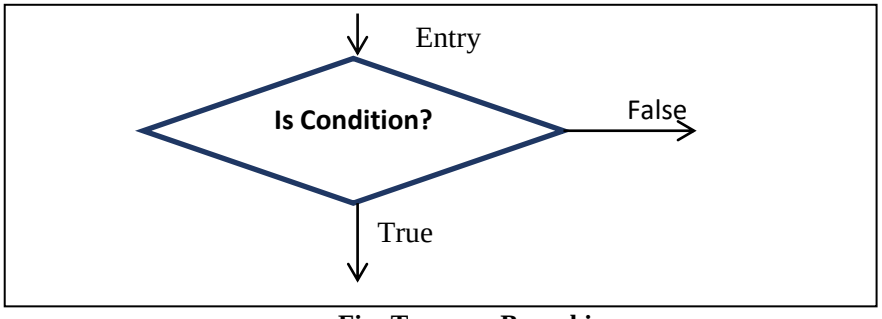



**Fig: Two way Branching**

## Types of Condition Control Statements

The different two-way selection statements supported by Python language are:

1. *if* statement
2. *if- else* statement
3. Nested *if else* statement
4. *if – elif* ladder

### 1. if statement:

It is basically a two way decision statement and it is used in conjunction with an expression. *It is used to **execute** a set of statements if the condition is true. If the condition is false it **skips** executing those set of statements.* The syntax and flow chart of if statement is as illustrated below:

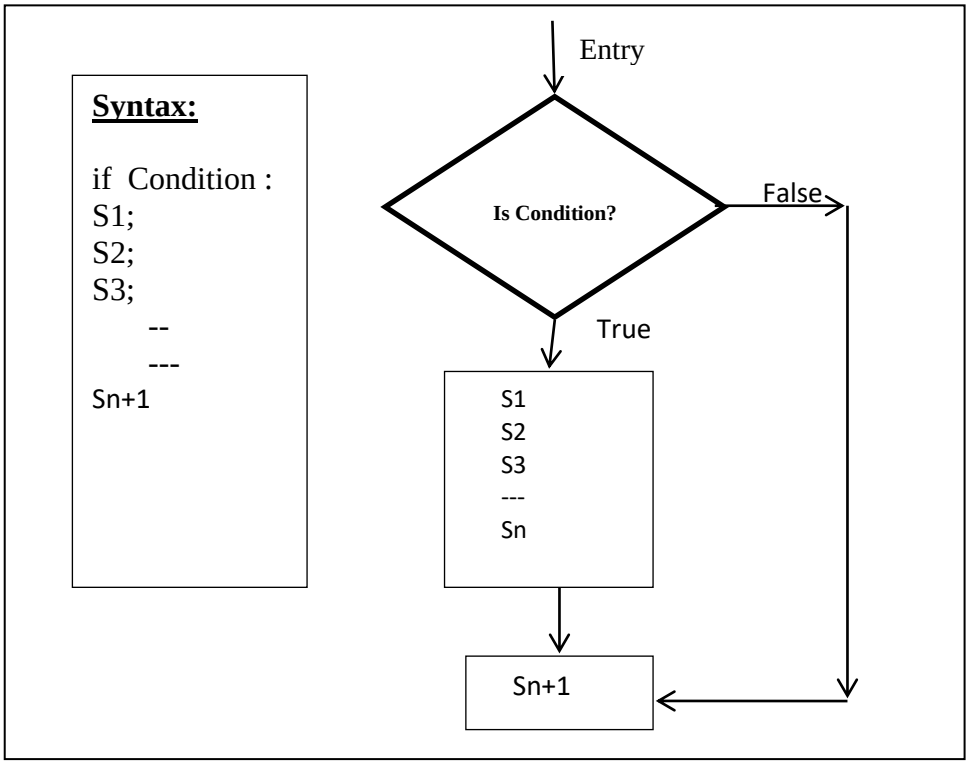


**Fig:Syntax and flow diagram for if statement**

**Example1: Python program to find the largest number using if statement.**

```
x,y = input("Enter two integer numbers: ").split()
n1 = int(x)
n2 = int(y)
large = n1
if n2>large:
    large = n2
print("Largest number = ",large)
```

```
Enter two integer numbers: 3 4
Largest number =  4
```

**Example2: Python Program to determine whether a person is eligible to vote using if.**

```
age = int(input("Enter your age: "))
if age>=18:
    print("You are eligible to vote")
```

```
Enter your age: 25
You are eligible to vote
```

## 2. If else statement:

It is an extension of if statement. It is used to execute any one set of two set of statements at a time. If condition is true it executes one set of statements otherwise it executes another set of statements. The syntax and flow diagram of if else is as shown in the figure below. As illustrated in the figure if the condition is true the set of statements {S11,S12,------S1n} gets executed else if the condition is false the set of statements {S21,S22,S23------S2n} gets executed.

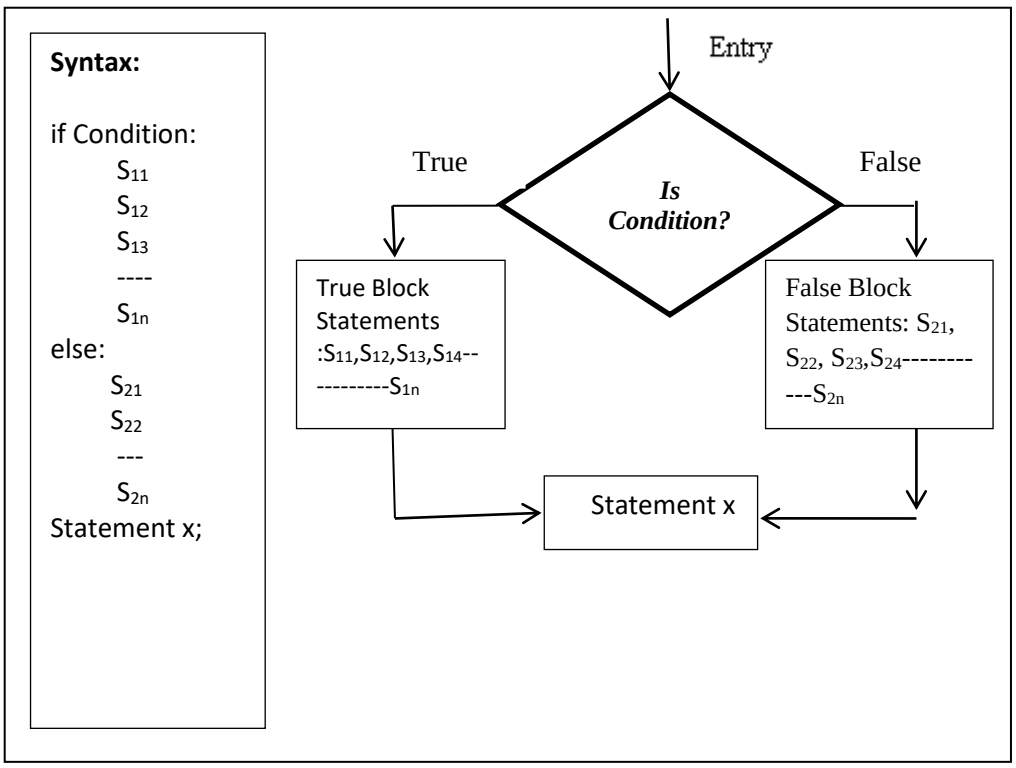



**Fig2: Syntax and flow diagram for if else statement**

**Example:** Program to check whether a given number is even or odd using if else.

```
1  n = int(input("Enter a number: "))
2  if n%2==0:
3      print("Entered number is Even")
4  else:
5      print("Entered number is Odd")
```

```
Enter a number: 2
Entered number is Even
```

3. **Nested if else:** When a series of decisions are involved, we may have to use more than one *if else* statement in nested form. The nested *if else* statements are multidecision statements which consist of *if else* control statement within another *if or else* section. The syntax and flow diagram for nested if else is as shown below:

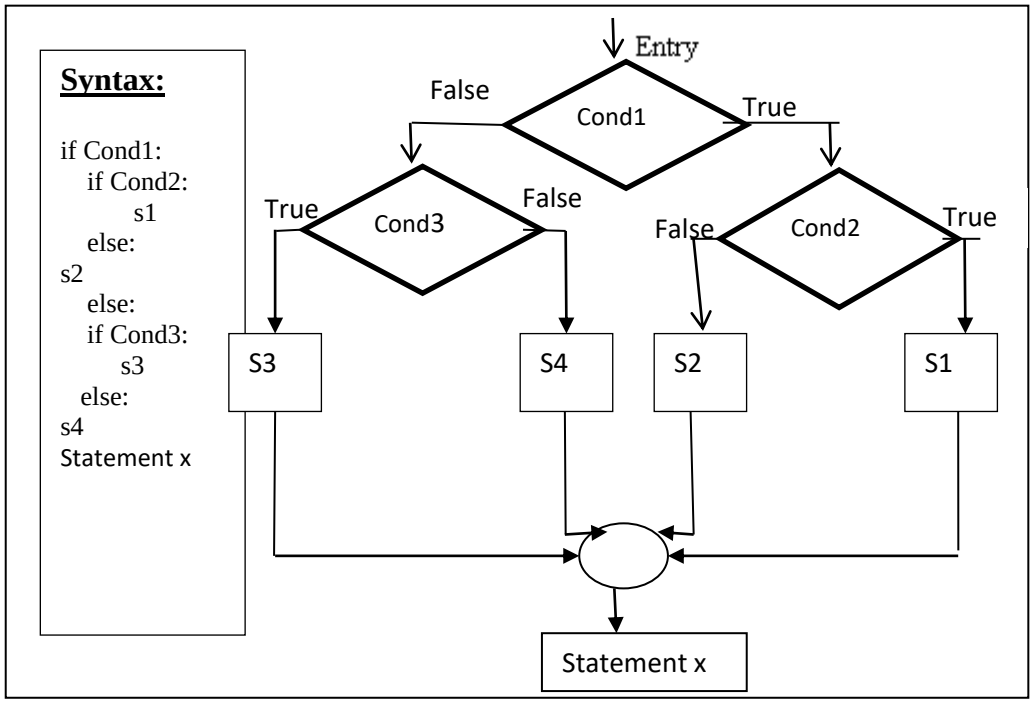



**Fig2.4: Syntax and flow diagram for nested if else statement**

## Example :

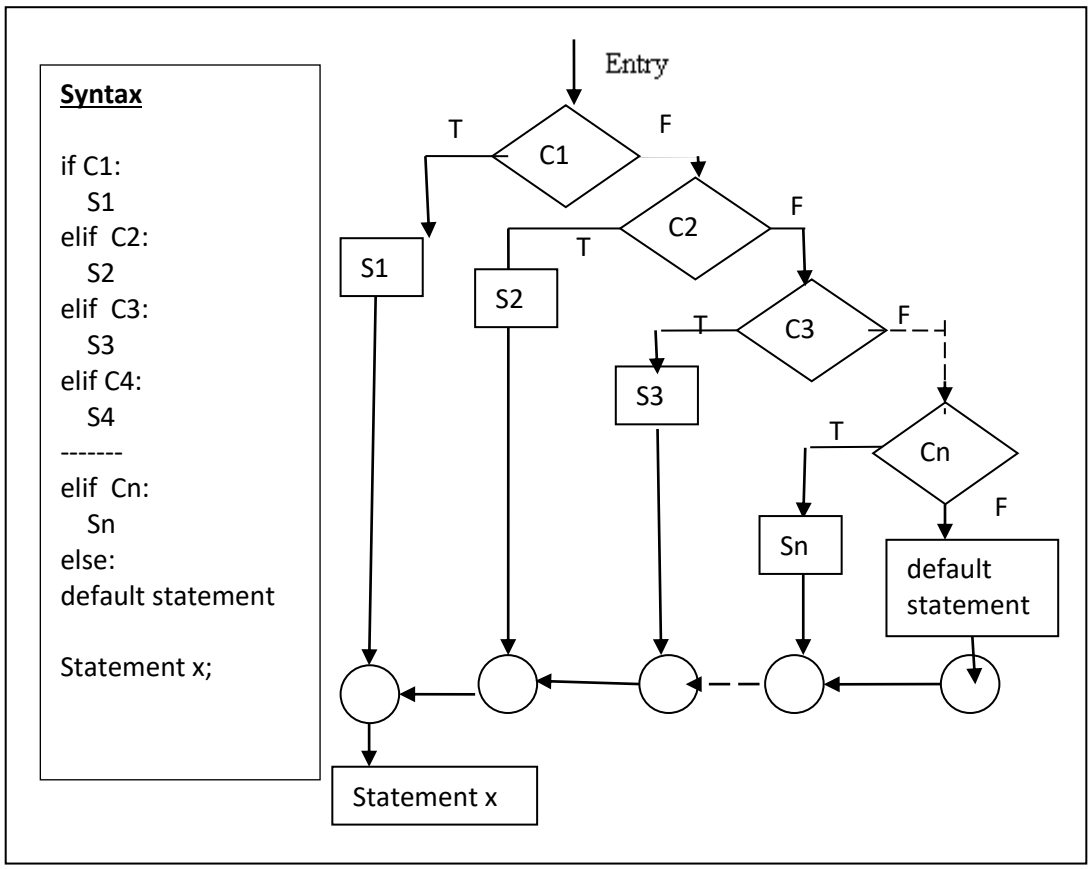
```
1  a,b,c = input("Enter three numbers : ").split()
2  a = int(a)
3  b = int(b)
4  c = int(c)
5  if a>b:
6      if a>c:
7          print("{0} is largest".format(a))
8      else:
9          print("{0} is largest".format(c));
10 else:
11     if b>c:
12         print("{0} is largest".format(b));
13     else:
14         print("{0} is largest".format(c))
```

```
Enter three numbers : -2 3 -7
3 is largest
```

## 4. *if – elif* ladder:

Cascaded *if elif else* is a multipath decision statements which consist of chain of *if elseif* where in the nesting take place only in else block. As soon as one of the conditions controlling the 'if' is true , then statement /statements associated with that 'if' are executed and the rest of the ladder is bypassed. If condition is false then it checks the first*elif* condition , if it is found true then first elif is executed. However, if none of the elif ladder is correct then the final else statement will be executed and control flows from top to down.

The syntax and flow diagram for cascaded *if else* is as shown in figure.



Example:

```
marks = int(input("Enter the marks [0 -100] : "))
if marks>=0 and marks<=39:
    print("Grade F\n");
elif marks>=40 and marks<=49:
    print("Grade E")
elif marks>=50 and marks<=59:
    print("Grade D")
elif marks>=60 and marks<=69:
    print("Grade C")
elif marks>=70 and marks<=79:
    print("Grade B")
elif marks>=80 and marks<=89:
    print("Grade A")
elif marks>=90 and marks<=100:
    print("Outstanding")
else:
    print("Invalid Entry")
```

```
Enter the marks [0 -100] : 90
Outstanding
```

## 3. Iteration or looping:

The statement which are used to repeat a set of statements repeatedly for a given number of times or until a given condition is satisfied is called as *looping constructs or looping statements*. The set or block of statements used for looping is called loop.

## Types of Looping statements:

Depending on the position of the control statement in the loop the looping statements are classified into the following two types:

1. Entry controlled Loop
2. Exit Controlled Loop

**1. Entry Controlled Loop:** In the entry controlled loop the control conditions are tested before the start of the loop execution. If the conditions are not satisfied, then the body of the loop will not be executed. It is also known as ***pretest loop or top test loop***. The flow chart for the entry controlled loop is as illustrated in fig .

**2.Exit Controlled Loop**: In the exit controlled loop the test is performed at the end of the body of the loop and therefore the body is executed unconditionally for the first time. It is also known as ***posttest loop***. Here the body of the loop will get executed at least once before testing.The flow chart for the exit controlled loop is as illustrated in fig .

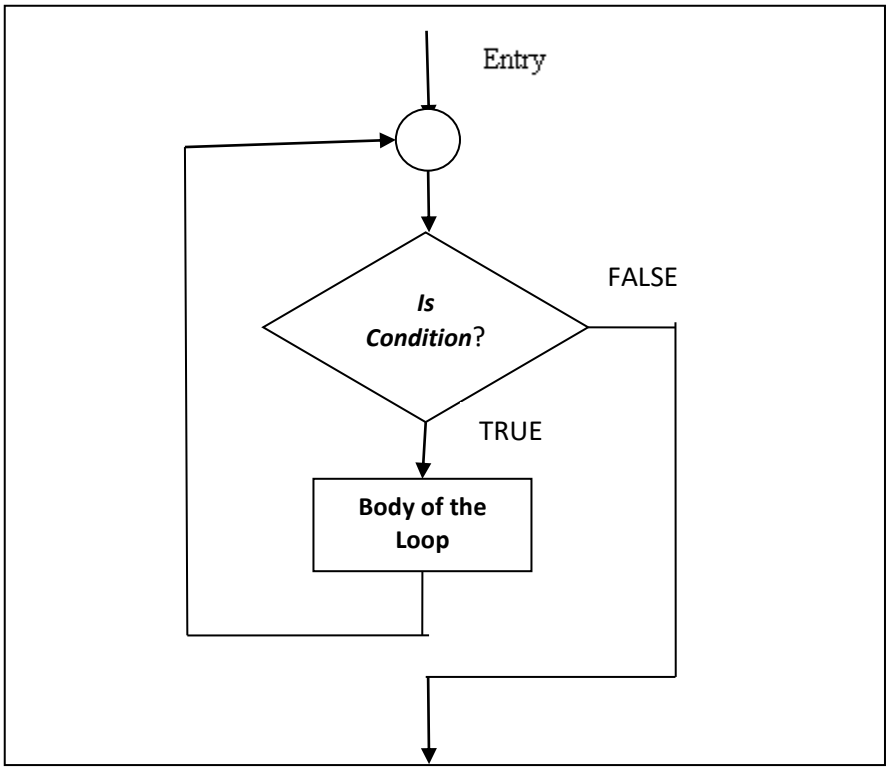



**Fig:Flow diagram for Entry Controlled Loop**

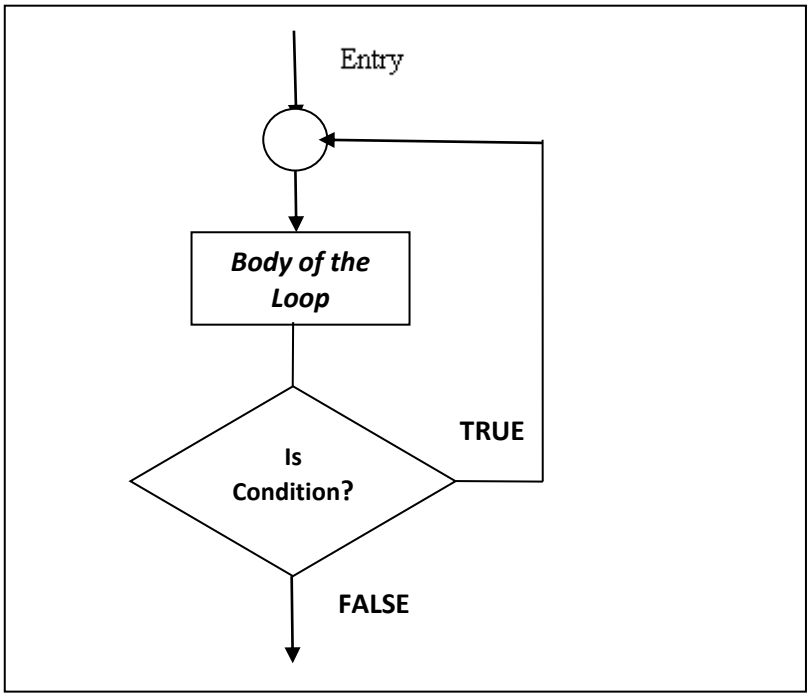



**Fig:Flow diagram for Exit Controlled Loop**

**The looping includes the following four steps:**

1. Initialization of a condition variable
2. Testing for a specified value of the condition variable for execution of the loop.
3. Execution of the statements in the loop
4. Updating (Incrementing or Decrementing) the condition variable

**Types of Loops Supported in Python:** The Python Language supports the following two looping operations:

1. The while statement
2. The for statement

**The while statement:** It is an entry controlled loop statement. In case of while loop the initialization, testing the condition and incrementation or updation is done in separate statements. First initialization of the loop counter is performed.Next the condition is checked. If the condition evaluates to be true then the control enters the body of the loop and executes the statements in the body.

**1.** The syntax or basic format of the while statement is as shown in figure 2.9 below :

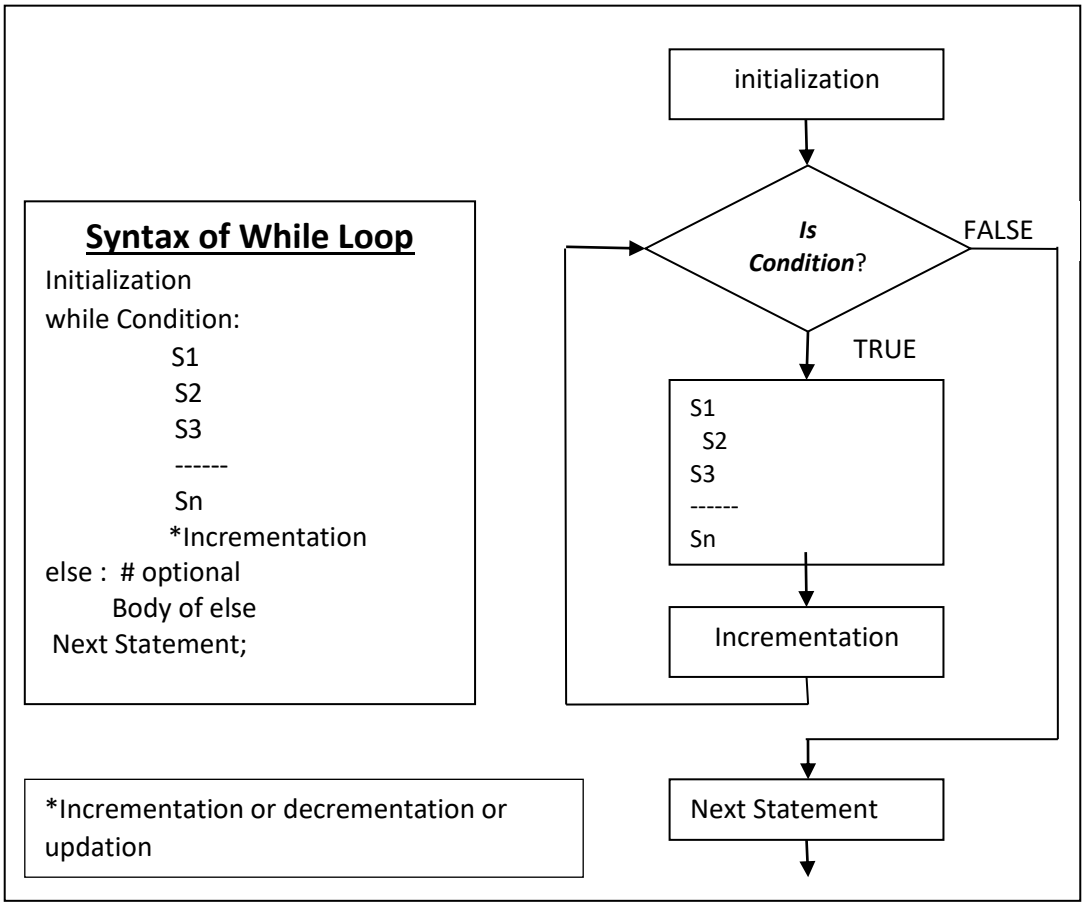



**Fig:Flow diagram for Exit Controlled Loop**

While statement in Python executes a block of code repeatedly as long as the test/control condition of the loop is true. The control condition of the while loop is executed before any statement in the body of the loop is executed . If the condition is true , the body of the loop is executed . Again the control condition of the loop is tested and the loo continue as long as the condition remains true. When the test outcome of this condition remains true. When the test outcome of this condition becomes false , the loop is not entered again and the control is transferred to the statement immediately following the body of the loop as shown in the flow diagram.

**Example:Program to find the sum of n natural numbers using while loop.**

```
1  n = int(input("Enter the number: "))
2  i=1
3  sum=0
4  while i<=n:
5      sum = sum + i
6      i = i+1
7  print("The sum of natural numbers = {}".format(sum))
```

```
Enter the number: 10
The sum of natural numbers = 55
```

## For loop:

The for loop is another entry-controlled loop. It is used to execute the set of statements repeatedly over a range of values or a sequence. With every iteration of the loop, the control variable checks whether each of the values in the range has been traversed or not. When all the items in the range are traversed the control is then transferred to the statement immediately following for loop.

**Syntax of for loop** :

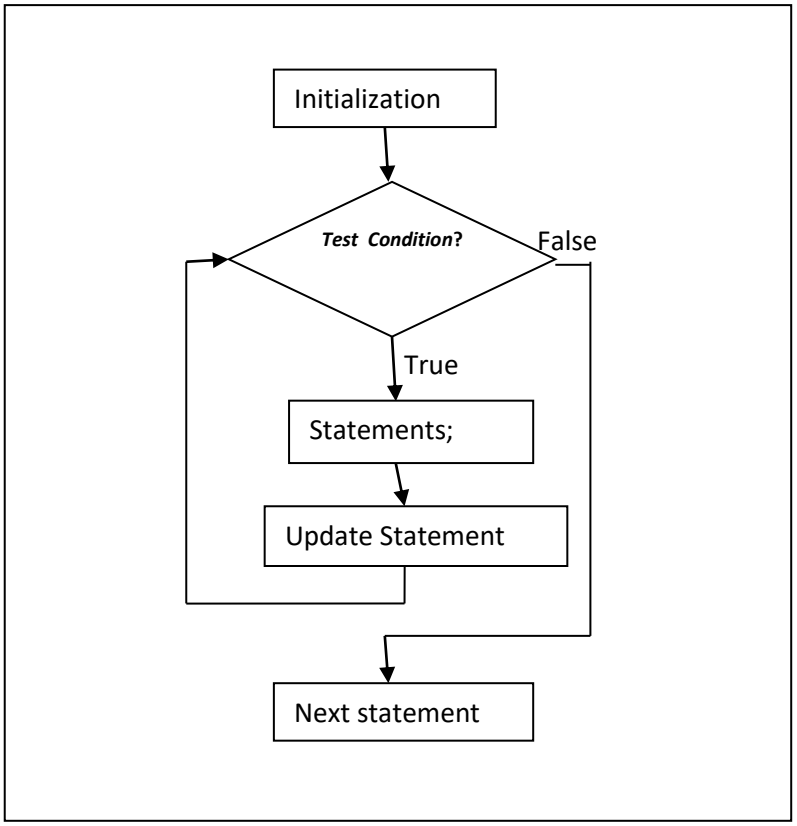
**for**<control_variable> in <sequence/items in range**>:**

<statements in body of the loop>

**else:** # optional

<Statements>

**Flow Chart**:




**Example: Python Program to find the sum of natural numbers upto n.**

```
n = int(input("Enter the value of n: "))
sum = 0
for i in range(1,n+1):
    sum = sum + i
print("Sum of First n natural numbers :",sum)
```

```
Enter the value of n: 10
Sum of First n natural numbers : 55
```

**range ( ) function**: The range() function returns a sequence of numbers , starting from 0 to a specified number ,incrementing each time by 1.

**Syntax** :

range(start,step,stop)

**Example** :

```
x = range(3, 6)
for n in x:
      print(n)
```

```
3
4
5
```

```
x = range(3, 20, 2)
for n in x:
      print(n)
```

```
3
5
7
9
11
13
15
17
19
```

| Command | Output |
|---|---|
| range(10) | [ 0,1,2,3,4,5,6,7,8,9] |
| range(1,11) | [ 1,2,3,4,5,6,7,8,9,10] |
| range(0,30,5) | [0,5,10,15,20,25] |
| range(0,-9,-1) | [0,-1,-2,-3,-4,-5,-6,-7,-8 |

The argument of **range()** function must be integers. The step parameter can be any positive or negative integer other than zero.

**Infinite Loop**: A loop becomes infinite loop if a condition never becomes FALSE. You must use caution when using while loops because of the possibility that this condition never resolves to a FALSE value. This results in a loop that never ends. Such a loop is called an infinite loop.
An infinite loop might be useful in client/server programming where the server needs to run continuously so that client programs can communicate with it as and when required.

```
var = 1
while var == 1 :  # This conditions results in an infinite loop
    num = input("Enter a number  :")
    print("You entered: ", num)
print("Good bye!")
```

```
Enter a number  :20
You entered:  20
Enter a number  :1
You entered:  1
Enter a number  :32
You entered:  32
Enter a number  :12
You entered:  12
Enter a number  :32
You entered:  32

Enter a number  :
```

## Nested Loops:

Python programming language allows to use one loop inside another loop.

### Syntax for nested for loop:

```
for iterating_var in sequence:
   for iterating_var in sequence:
      statements(s)
   statements(s)
```

## Example:

```
for i in range(1,6):
    for j in range(1,i+1):
        print(j,end =' ')
    print()
```

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

## Syntax for nested while loop:

The syntax for a **nested while loop** statement in Python programming language is as follows –

```
while expression:
   while expression:
      statement(s)
   statement(s)
```

A final note on loop nesting is that you can put any type of loop inside of any other type of loop. For example a for loop can be inside a while loop or vice versa. The following program uses a nested for loop to find the prime numbers from 2 to 20−

```
i = 2
while(i < 20):
    j = 2
    while(j <= (i/j)):
        if not(i%j): break
        j = j + 1
    if (j > i/j) : print(i, " is prime")
    i = i + 1
print("Good bye!")
```

```
2  is prime
3  is prime
5  is prime
7  is prime
11  is prime
13  is prime
17  is prime
19  is prime
Good bye!
```

## Jump statements:

You might face a situation in which you need to exit a loop completely when an external condition is triggered or there may also be a situation when you want to skip a part of the loop and start next execution.

Python provides **break** and **continue** statements to handle such situations and to have good control on your loop.

The **break** statement in Python terminates the current loop and resumes execution at the next statement, just like the traditional break found in C.

The most common use for break is when some external condition is triggered requiring a hasty exit from a loop. The **break** statement can be used in both *while* and *for* loops.

```
1  for letter in 'Python':     # First Example
2      if letter == 'h':
3            break
4      print('Current Letter :', letter)
5
6  var = 10                    # Second Example
7  while var > 0:
8      print('Current variable value :', var)
9      var = var -1
10     if var == 5:
11         break
12 print("Good bye!")
```

```
Current Letter : P
Current Letter : y
Current Letter : t
Current variable value : 10
Current variable value : 9
Current variable value : 8
Current variable value : 7
Current variable value : 6
Good bye!
```

The **continue** statement in Python returns the control to the beginning of the while loop. The **continue** statement rejects all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop.

The **continue** statement can be used in both *while* and *for* loops.

**Example:**

```
for letter in 'Python':     # First Example
    if letter == 'h':
        continue
    print('Current Letter :', letter)

var = 5                    # Second Example
while var > 0:
    var = var -1
    if var == 5:
        continue
    print('Current variable value :', var)
print("Good bye!")
```

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : o
Current Letter : n
Current variable value : 4
Current variable value : 3
Current variable value : 2
Current variable value : 1
Current variable value : 0
Good bye!
```

## else statement used with loops :

Python supports to have an **else** statement associated with a loop statements.

- If the **else** statement is used with a **for** loop, the **else** statement is executed when the loop has exhausted iterating the list.
- If the **else** statement is used with a **while** loop, the **else** statement is executed when the condition becomes false.

**Example :**

The following example illustrates the combination of an else statement with a for statement that searches for prime numbers from 10 through 20.

```
for num in range(10,20):  #to iterate between 10 to 20
    for i in range(2,num): #to iterate on the factors of the number
        if num%i == 0:      #to determine the first factor
            j=num/i #to calculate the second factor
            print('%d equals %d * %d' % (num,i,j))
            break #to move to the next number, the #first FOR
    else:         # else part of the loop
        print(num, 'is a prime number')
```

```
10 equals 2 * 5
11 is a prime number
12 equals 2 * 6
13 is a prime number
14 equals 2 * 7
15 equals 3 * 5
16 equals 2 * 8
17 is a prime number
18 equals 2 * 9
19 is a prime number
```

Similar way you can use **else** statement with **while** loop.

**The Pass Statement :**

The **pass** statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.The **pass** statement is a *null* operation; nothing happens when it executes. The **pass** is also useful in places where your code will eventually go, but has not been written yet (e.g., in stubs for example):

```python
for letter in 'Python':
    if letter == 'h':
          pass
          print('This is pass block')
    print('Current Letter :', letter)

print("Good bye!")
```

```
Current Letter : P
Current Letter : y
Current Letter : t
This is pass block
Current Letter : h
Current Letter : o
Current Letter : n
Good bye!
```

# Importing Modules

Python function definition can, usefully, be stored in one or more separate files for easier maintenance and to allow them to beused in several programs without copying the definitions into each one. Each file storing function definitions is called a "module" and the module name is the file name with "**.py**" extension.

**Example:**

Functions stored in the module are made available to a program using the Python import keyword followed by the module name. Although not essential, it is customary to put any import statements at the beginning of the program.

Imported functions can be called using their name dot -suffixed after the module name. For example, a "**f1()**" function from an imported module named "userdefined" can be called with ***userdefined.f1()*** .

Design a new Python module called userdefined by defining all functions as illustrated below. Save the file as userdefined.py and run the file.

## userdefined.py

```
1  def f1():
2      print("I am in f1")
3
4  def f2():
5      print("I am in f2")
6
7  def f3():
8      print("I am in f3")
9
10 def n5(n):
11     return n**5
12
13 def n6(n):
14     return n**6
```

Start a new script with name **importexample.py** (/ipynb). Next call each function with and without arguments as per the requirements. Run the program and get the output as illustrated below:

## Importing Modules:

```
import userdefined
userdefined.f1()
userdefined.f2()
userdefined.f3()
x = userdefined.n5(2)
print(x)
y = userdefined.n6(3)
print(y)
```

```
I am in f1
I am in f2
I am in f3
32
729
```

## Importing the sys and keyword modules :

Python includes "**sys**" and "keyword" modules that are useful for interrogating the Python system itself. The keyword module contains a list of all Python keywords in its ***kwlist*** attribute and provides as ***iskeyword ()*** method if you want to rest a word.

```
import sys,keyword
```

```
print("Python Version:",sys.version)
```

```
Python Version: 3.7.3 (default, Mar 27 2019, 17:13:21) [MSC v.1915 64 bit (AMD6
4)]
```

```
print("Python Interpreter Location:",sys.executable)
```

```
Python Interpreter Location: C:\Users\thyagaraj\Anaconda3\python.exe
```

```
print('Python Module Search Path:')
for dir in sys.path:
    print(dir)
```

```
Python Module Search Path:
C:\Users\thyagaraj\V SEM 2020
C:\Users\thyagaraj\Anaconda3\python37.zip
C:\Users\thyagaraj\Anaconda3\DLLs
C:\Users\thyagaraj\Anaconda3\lib
C:\Users\thyagaraj\Anaconda3
C:\Users\thyagaraj\AppData\Roaming\Python\Python37\site-packages
C:\Users\thyagaraj\Anaconda3\lib\site-packages
C:\Users\thyagaraj\Anaconda3\lib\site-packages\web.py-0.40.dev1-py3.7.egg
C:\Users\thyagaraj\Anaconda3\lib\site-packages\win32
C:\Users\thyagaraj\Anaconda3\lib\site-packages\win32\lib
C:\Users\thyagaraj\Anaconda3\lib\site-packages\Pythonwin
C:\Users\thyagaraj\Anaconda3\lib\site-packages\IPython\extensions
C:\Users\thyagaraj\.ipython
```

```
# To display the list of all the Python Keywords
print('Python Keywords:')
for word in keyword.kwlist:
    print(word ,',',end=' ')
```

```
Python Keywords:
False , None , True , and , as , assert , async , await , break , class , continue , def , del , elif , else , except , finally , for , from , global , if , import , in , is , lambda , nonlocal , not , or , pass , raise , return , try , while , with , yield ,
```

## Performing Mathematics

```
import math
```

```
print("2 power 3",math.pow(2,3))
print("Rounded up 3.2",math.ceil(3.2))
print("Rounded Down3.7",math.floor(3.7))
print("Square root of 16",math.sqrt(16))
print("Sin of 30", math.sin(30))
print("Cosin of 45",math.cos(45))
print("tan of 60", math.tan(60))

```

```
2 power 3 8.0
Rounded up 3.2 4
Rounded Down3.7 3
Square root of 16 4.0
Sin of 30 -0.9880316240928618
Cosin of 45 0.5253219888177297
tan of 60 0.320040389379563
```

## Random

```
import random
```

```
nums = random.sample(range(1,49),6)
print("Your Lucky Lotto Number Are:",nums)
```

```
Your Lucky Lotto Number Are: [38, 44, 19, 48, 31, 13]
```

```
random.seed(10)
print(random.random())
```

```
0.5714025946899135
```

```
# Returns a randomly selected element from the range created by the start,
# stop and step arguments.
print(random.randrange(3,9,2))
```

```
7
```

```python
import random
for i in range(5):
    print(random.randint(1, 10),end=' ')
```

```
1 7 3 10 6
```

## Ending a Program Early with sys.exit ()

```python
import sys
while True:
    print('Type exit to exit.')
    response = input()
    if response == 'exit':
        sys.exit()
    print('You typed ' + response + '.')
```

```
Type exit to exit.
one
You typed one.
Type exit to exit.
two
You typed two .
Type exit to exit.
exit

An exception has occurred, use %tb to see the full traceback.

SystemExit
```

## Questions for Practice:

**1.** What is Flow Chart ? Give the meaning of different flow chart symbols.

**2.** Write a Flow chart and Python program for the following:

a. To find the average of two numbers.
b. To find the simple interest given the value of P,T and R
c. To find the maximum of two numbers
d. To find the sum of first 50 natural numbers
e. To find the factorial of a given number N.

**3.** Compare == and = operator.

**4.** What are operators? Explain the following Operators with example:

a. Binary Boolean Operators: and, or & not

**5.** What is Flow Control? Explain the different Elements of Flow Control?

**6.** Define Block and explain what nested blocks with example are.

**7.** Define condition control statement. Explain the different types of Condition Control Statement.

**8.** Explain with flow chart and programming example , the following condition control statements :

*a. if*
*b. if – else*
*c. nested if else*
*d. if – elif ladder*

***9.*** What is iteration or looping ? Describe the different types of looping statements.

***10.*** Explain with syntax , flow chart and programming example the following looping operations.

*a.* While loop
*b.* For loop

***11.*** Discuss the working of range() function with programming example.

***12.*** Give the output of the following :

*a.* range(10)

*b.* range(1,11)

*c.* range(0,30,5)

*d.* range(0,-9,-1)

***13.*** What is infinite loop ? Explain with example.

**14.** What are nested Loops ? Explain with examples.

**15.** Discuss the following with examples :

*a.* break

*b.* continue

*c.* else statement with loop

*d.* pass

***16.*** What are Python Modules ? Explain with examples how to import Python Modules .

***17.*** What is the difference between break and continue statements.

***18.*** What is the purpose of else in loop?

***19.*** Write logical expressions for the following :

*a. Either A is greater than B or A is less than C*

*b. Name is Snehith and age is between 18 and 35.*

*c. Place is either Mysore or Bengaluru but not "Dharwad".*

***20.*** *Convert the following while loop into for loop :*

```
x =10
while (x<20):
    print(x+10)
    x+=2
```

***21.*** *Explain while and for loop . Write a program to generate Fibonacci series upto the given limit by defining FIBONACCI(n) function.*

***22.*** *Mention the advantage of continue statement. Write a program to compute only even numbers sum within the given natural number using continue statement.*

***23.*** *Demonstrate the use of break and continue keywords in looping structures using a snippet code.*

***24.*** *With syntax explain the finite and infinite looping constructs in python. What is the break and continue statements .[ VTU June /July 2019]*

## Programming Questions for Practice:

**1.**Construct a logical expressions to represent each of the following conditions :

a. Mark is greater than or equal to 100 but less than 70
b. Num is between 0 and 5 but not equal to 2
c. Answer is either 'N' or 'n'
d. Age is greater than or equal to 18 and gender is male
e. City is either 'Kolkata' or 'Mumbai'

**2.**Write a program to check if the number of positive or negative and display an appropriate message.

**3.**Write a program to convert temperature in Fahrenheit to Celsius.

**4.**Write a program to display even numbers between 10 and 20.

**5.**Write a program to perform all the mathematical operations of calculator.

**6.**Write a program to accept a number and display the factorial of that number.

**7.**Write a program to convert binary number to decimal number.

**8.**Write a program to find the sum of the digits of a number.

**9.**Write a program to display prime number between 30.

**10.**Write a program to find the best of two test average marks out of three tests marks accepted from the user.

**11.**Write a program to find the largest of three numbers . [ VTU June /July 2019]

**12.**Write a program to check whether the given year is leap or not. [ VTU June /July 2019]

**13.**Write a program to generate and print prime number between 2 to 50.

**14.**Write a program to find those numbers which are divisible by 7 and multiple of 5 , between 1000 and 3000.

**15.**Write a program to guess a number between 1 to 10.

**16.**Write a program that accepts a word from the user and reverse it.

**17.**Write a program to count the number of even and odd numbers from a series of numbers.

**18.**Write a program that prints all the numbers from 0 to 10 except 3, 7 and 10.

**19.**Write a program to generate Fibonacci series between 0 and 50.

**20.**Write a program which iterates the integers from 1 to 50. For multiple of three print “Fizz” instead of the numbers and fro the multiples of five print “Buzz” . For numbers which are multiples of both three and five print “FizzBuzz”.

**21.**Write a program that accepts a string and calculate the number of digits and letters.

**22.**Write a program to check the validity of password input by users.

**23.**Write a program to find the numbers between 100 and 400 where each digit of a number is an even number . The numbers obtained should be printed in a comma-separated sequence.

**24.**Write a program to print alphabet patterns ‘A’ ,D, ‘E’, ‘G’,’L’, ‘T’ and ‘S’ with * symbol.

**25.** Write a python program to create the multiplication table (from 1 to 20) of a number.

**26.** Write a program to print the following patterns :

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

```
1
22
333
4444
55555
666666
7777777
88888888
999999999
```

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

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

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

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

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

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

```
        1
      2 3 2
    3 4 5 4 3
  4 5 6 7 6 5 4
5 6 7 8 9 8 7 6 5
```

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

```
          1
        1   1
      1   2   1
    1   3   3    1
  1  4    6   4   1
 1  5   10   10  5   1
```

```
1
2 3
4 5 6
7 8 9 10
```