UNIT I

1.1 - ALGORITHMS

An Algorithm is defined as step by step procedure for solving a problem. An Algorithm is a well-defined computational procedure consists of a set of instructions that takes some values as an input, then manipulate them by following prescribed texts in order to produce some values as output.

INPUT
ALGORITHM
OUTPUT

1.1.1 –Two Phases of Algorithmic Problem Solving

- Derivation of an algorithm that solves the problem
- Conversion of an algorithm into code (or) program

1.1.2 –Data types and Data Structures:

In Algorithms the data are numbers, words, list & files. Algorithm provides the logic and data provides the values.

Program = Algorithm + Data Structures

Data Structures refer to the types of data used and how the data are organized in the program. The Programming languages provide simple data type such as Integers, real numbers and characters.

1.1.3 – Characteristics of an Algorithm:

An algorithm should have the following characteristics.

Precision – the steps are precisely stated (defined).

Uniqueness – results of each step are uniquely defined and only depend on the input and the result of the preceding steps.

Finiteness – the algorithm stops after a finite number of instructions are executed.

Effectiveness – algorithm should be most effective among many different ways to solve a problem.

Input – the algorithm receives input.

Output – the algorithm produces output.

Generality – the algorithm applies to a set of inputs.

1.1.4 – Qualities of an Algorithm:

The following factors will determine the quality of an algorithm.

Accuracy	Algorithm should provide accurate results.
Memory	It should require minimum computer memory.
Time	The time taken to execute any program is considered as a main quality.
Sequence	The procedure of an algorithm must be in a sequential form.

1.1.5 – Representation of an Algorithm:

An algorithm can be represented in a number of ways which include

Natural Language	usually verbose and ambiguous.
Flow Chart	pictorial representation of an algorithm
Pseudo Code	resembles common elements of an language and no particular format
Programming Language	Algorithms are the basis of programs. Expressed in High level language.

Examples:

1. Write an algorithm to find the area of circle.

```
Step1: Start
Step2: Read the value of radius as r
Step3: Calculate area = 3.14*r*r
Step4: Print the value of area.
Step5: Stop
```

2. Write an algorithm to find the biggest of two numbers.

```
Step1: Start
Step2: Read the value of a and b
Step3: Compare the value of a sand b if a>b then
        Print ‘A is largest” otherwise print ‘b is largest’
Step4: Stop
```

3. Write an algorithm for calculating total marks of specified number of subjects given as 66, 99, 98, 87, 89.

Step1: Start

Step2: Read Numbers as N

Step3: Initialize the value of Total as 0 and as 1

Step4: Repeat step 5 to step7 until i is less than n

Step5: Read the marks

Step6: Add mark to the total

Step7: Increment the value of i

Step6: Print the value of total

Step7: Stop

1.2 – Building Blocks of an Algorithm

There are four building blocks of algorithm which are

a) Instructions / Statement

i. *Simple Statement*

ii. *Compound Statement*

b) State

c) Control Flow

i. *Sequence Control Structure*

ii. *Selection Control Structure*

iii. *Case Structure*

d) Functions

a) Instructions / Statement

In Computer programming, a statement is a smallest standalone element which expresses some action to be covered. It is an instruction written in a high level language that commands the computer to perform a specified action. A program written in such language is formed by sequence of one or more statements.

Ex:

An Algorithm to add two numbers:

Step-1: Start

Step-2: Declare variables num1, num2 and sum

Step-3: Read values num1, num2

Step-4: Add num1 and num2 and assign the result to sum

```
        sum→num1+num2
Step-5: Display sum
Step-6: Stop
```

- There are two types of statements they are
    i. *Simple Statement*
    ii. *Compound Statement*

i) Simple Statement:

- Simple Statement consists of a single logical line.

Ex:

- Expression Statement
- Assignment Statement
- Print Statement
- Import Statement

ii) Compound Statement:

- Compound Statements consist of a group of statements. They control the execution of other statements. It consists of one or more clauses.

Ex:

- if – statement
- while – statement
- for – statement
- try – statement

b) State

An algorithm is a deterministic automation for accomplishing a goal which given an initial state, will terminate in a defined end state and internal state for performing an algorithm.

- Initial State starts by keyword "START" or "BEGIN".
- Internal State starts from data, which is read as an input, written to an output and stored for processing. The stored data is regarded as a part of Internal State.
- End State is nothing but the keyword "END"

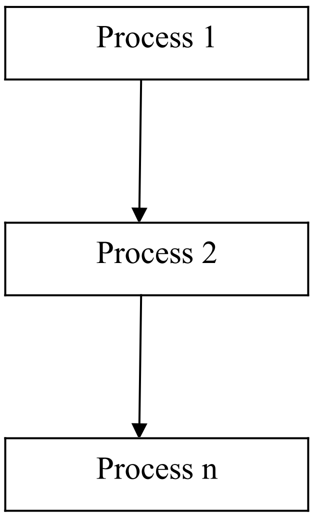
c) Control Flow

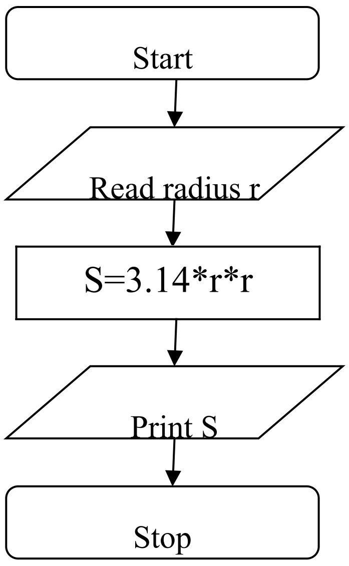
Control Flow or Flow of Control is the order function calls, instructions and statement are executed or evaluated when a program is running. Program Control structures are defined as the program statements that specifies the order in which statements are executed. There are three types of control structures. They are.

i. Sequence Control Structure
ii. Selection Control Structure
iii. Case Structure

i) Sequence Control Structure

- Sequence Control Structure is used to perform the actions one after another. It performs process A and then performs process B and so on. This structure is represented by writing one process after another.
- The logic is from top to bottom approach.





ii) Selection Control Structure

Selection Control Structure or Decision Control Structure allows the program to make a choice between two alternate paths whether it is True or False.

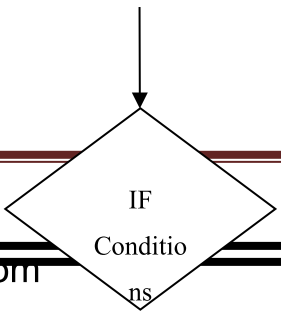
1) IF….THEN
2) IF….THEN…ELSE

1) IF….THEN

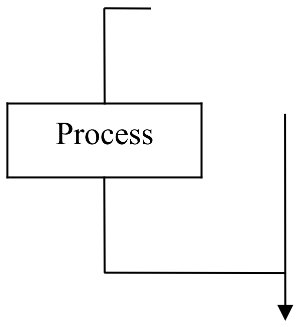
- This choice makes between two processes.
  - If the condition is true. It performs the process.
  - If the condition is false. It skips over the process.

Pseudo code *Flow Chart*

```
IF Condition THEN
```



```
        Process A
        …
        …
        …
END IF
```



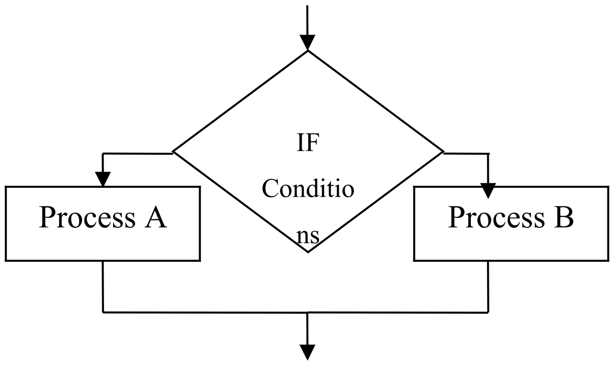
2) IF….THEN…ELSE

- In this structure
    - If the condition is true. It performs the process A.
    - Else the condition is false. It executes process B.
- The process execution is depending on the condition.

Pseudo code

```
IF Condition THEN
        Process A
ELSE
        Process B
END IF
```

Flow Chart



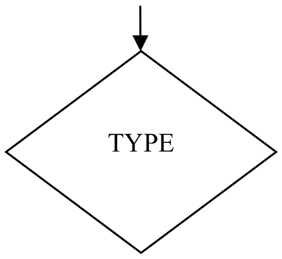
iii) Case Structure:

- This is a multi-way selection structures that is used to choose one option from many options. If the value is equal to type 1 then it executes process-1, If the value is equal to type 2 then it executes process-2 and so on. END CASE is used to indicate the end of the CASE Structure.

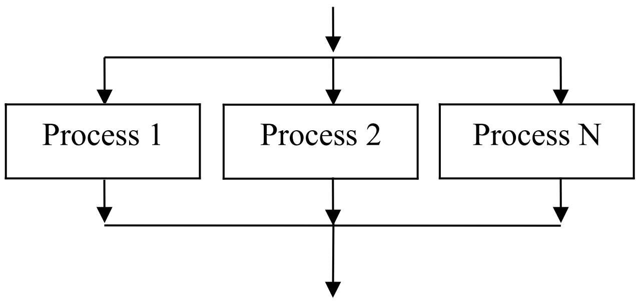
Pseudo code

```
CASE TYPE
case Type-1:
    Process 1
case Type-2:
```

Flow Chart



```
    Process 2
…
…
…
case Type-n:
    Process n
END CASE
```



d) Functions

Functions are "self-contained" modules of code that accomplish a specific task. Functions usually "take in" data as INPUT, process it, and "return" a result as OUTPUT. Once a function is written, it can be used over and over and over again. Functions can be "called" from the inside of other functions. Functions "Encapsulate" a task (they combine many instructions into a single line of code).

Ex:

```
>>>def add(num1,num2):                    #Function Definition
        num3=num1+num2
        print("The sum is",num3)
        return num3
>>>a=10
>>>b=20
>>>c=add(a,b)                             #Function Call
```

Flow of a Function:

When a function is **"called"** the program "leaves" the current section of code and begins to execute the first line inside the function. Thus the function "flow of control" is:

- The program comes to a line of code containing a "function call".
- The program enters the function (starts at the first line in the function code).
- All instructions inside of the function are executed from top to bottom.
- The program leaves the function and goes back to where it started from.
- Any data computed and RETURNED by the function is used in place of the function in the original line of code.

Need for Functions

- Functions allow us to create sub-steps. (Each sub-step can be its own function and a group of Instructions)
- They allow us to reuse code instead of rewriting it.
- Functions allow us to keep our variable namespace without any confusion.
- Functions allow us to test small parts of our program in isolation from the rest.

<u>The Steps to Write a Function</u>

- Understand the purpose of the function.
- Define the data that comes into the function (in the form of parameters).
- Define what data variables are needed inside the function to accomplish its goal.
- Decide the data produces as output to accomplish this goal.

<u>Parts of a "black box" (or) Parts of a function</u>

Functions can be called "black boxes" because we don't know how they work. We just know what is supposed to go into them, and what is supposed to come out of them.

1. **The Name** - describes the purpose of the function. Usually a verb or phrase, such as "compute_Average", or just "average".
2. **The Inputs** - called parameters. Describe what data is necessary for the function to work.
3. **The Calculation** - varies for each function
4. **The Output** – The calculated values from the function and "returned" via the output variables.

<u>Function Workspace</u>

Every function has its own Workspace which means that every variable inside the function is only usable during the execution of the function. After that we cannot use the variable. The variables can be created inside the function or passed as an argument.

<u>Function Workspace</u>

Every function has its own Workspace which means that every variable inside the function is only usable during the execution of the function. After that we cannot use the variable. The variables can be created inside the function or passed as an argument.

<u>Formal Parameters vs. Actual Parameters</u>

- Inside the circle function, the variable **‘r’** will be used in place to calculate radius.

- The parameter "r" is called a Formal parameter, this just means a place holder name for radius.

The variable **'radius'** is the Actual parameter, this means "what is actually used" for this call to the function

Example:

Write a Python Program to find area of a circle:

```
>>>def circle(r):                    #Where 'r' is the Formal Parameter
        rad=3.14*r*r
        print("The area of circle is",rad)
>>>radius=9
>>>circle(radius)                    #Where 'radius' is the Actual Parameter
```

1.3 NOTATIONS (CLASSIFICATIONS OF AN ALGORITHM):

The Notations of an algorithm is classified into three

i. *Pseudo code*
ii. *Flow Chart*
iii. *Programming Language*

1.3.1. Pseudo Code:

- Pseudo code is a kind of structure English for designing algorithm.
- Pseudo code came from two words.
    - Pseudo Means imitation
    - Code means instruction
- Pseudo code is a text based tool for human understanding
- Pseudo cannot be compiled nor executed, and there are no real format or syntax rules
- The benefits of pseudo code is it enables the programmer to concentrate on the algorithm without worrying about all syntactic details

_Ex:_

```
READ num1, num2
result = num1 + num2
WRITE result.
```

Basic Guidelines for writing Pseudo code:

- *Write only one statement per line*
    - Each statement in your pseudocode should express just one action for the computer
- *Capitalize Initial Keyword*

        - The keywords should be written in capital letters. Ex – READ, WRITE, IF, ELSE, WHILE
    - *Indent to show hierarchy (order)*
        - We use a particular Indentation in each design.
        - For sequence statement Indent start from same column
        - For selection & looping statement Indent will leave some space and fall inside the column.
            - <u>*Ex:*</u>

```
IF a>b then
    print a
ELSE
    print b
```

    - *End multi line Structures*
        - Each structure must be ended properly to provide more clarity
            - *Ex:* END IF
    - *Keep statements language Independent.*
        - The statement should be Independent to the language we are using. The language features should be used properly.

<u>Advantages of Pseudo Code.</u>

- Can be done easily on a word processor.
- Easily modified.
- Implements structured concepts well.
- No special symbols are used.
- It is simple because it uses English like statements.

<u>Disadvantage of pseudo Code.</u>

- It is not visual.
- There is no standard format.
- Cannot be compiled not executed.

<u>*Examples:*</u>

1. Write a pseudocode to find the area of circle.

```
BEGIN
READ radius  r
INITIALIZE pi=3.14
COMPUTE Area=pi*r*r
PRINT r
```

2. Write a pseudocode to find the biggest of two numbers

```
BEGIN
```

```
READ A, B
IF A>B
    PRINT “B is big”
ELSE
   PRINT “A is big”
```

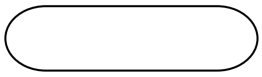
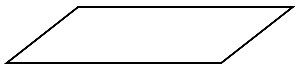
3. Write a pseudocode for calculating total marks of specified number of subjects given as 66, 99, 98, 87, 89

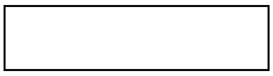
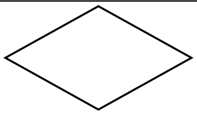
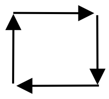
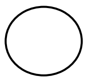
```
BEGIN
INITIALIZE  total=0,i=1
WHILE i<n
            READ mark
            total=total+mark
            INCREMENT i
PRINT i
END
```

1.3.2. Flowchart

- A flow chart is a diagrammatic representation that illustrates the sequence of operations to be performed to arrive the solution.
- Each step in the process is represented by a different symbol and has a short description of the process steps.
- The flow chart symbols are linked together with arrows showing flow directions.

Flow Chart Symbols:

S.No	Name of the Symbol	Symbol	Meaning
1.	Terminal		Represent the start and stop of the program
2.	Input / Output		Denoted either an input or output operation.

3.	Process		Denotes the process to be carried out.
4.	Decision		Represent decision making and branching
5.	Flow Lines		Represent the sequence of steps and directions of flow
6.	Connectors		Represent a circle and a letter or digit inside the circle. Used to connect two flow charts

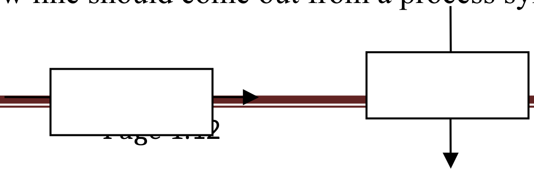
Types of Flow Chart:

- ***High Level Flow Chart:***
  - Shows major steps in the process.
  - Which also includes intermediate outputs of each step.

- ***Detailed Flow Chart:***
  - Which provides detailed picture of a process by mapping all steps
  - Which gives the detail about all the steps
- ***Deployment or Matrix Flow Chart:***
  - Which maps out the process in terms of who is doing the steps.
  - Which is in the matrix format.
  - Which shows various participants and the flow of steps among those participants.

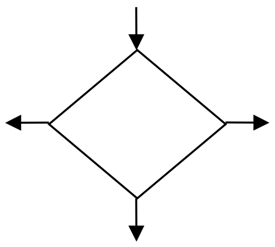
Basic Guidelines for preparing Flow Chart:

- Flow chart should have all necessary requirements and should be listed out in logical order.
- The Flow Chart should be clear neat and easy to follow. There should not be any ambiguity.
- The usual direction of the flow chart is from left to right and top to bottom.
- Only one flow line should come out from a process symbol

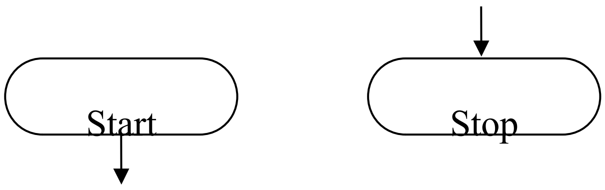


(or)

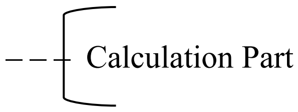
- Only one flow line should enter a decision symbol but two or three flow lines can leave.



- Only one flow line is used in conjunction with terminal symbol



- Write within standard symbols briefly. You can also use annotation symbol to describe the data more clearly.



- If the flowchart becomes complex then it is better to use connector symbols to reduce the number of flow lines.
- Flow chart should have a start and stop.
- Validity of a flow chart can be tested by passing through it with a simple test data.

Advantages of Flow Chart:

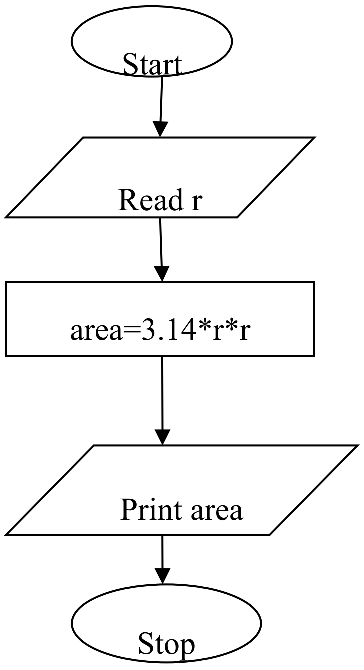
- Flow Charts are better in communication.
- A Problem can be analyzed in an effective way.
- Program Flow Charts Serve as a good documentation.
- Flow Charts acts as a blue print during system analysis and program development phase.
- The maintenance of operating program becomes easy with the help of flow chart.

Disadvantages of Flow Chart:

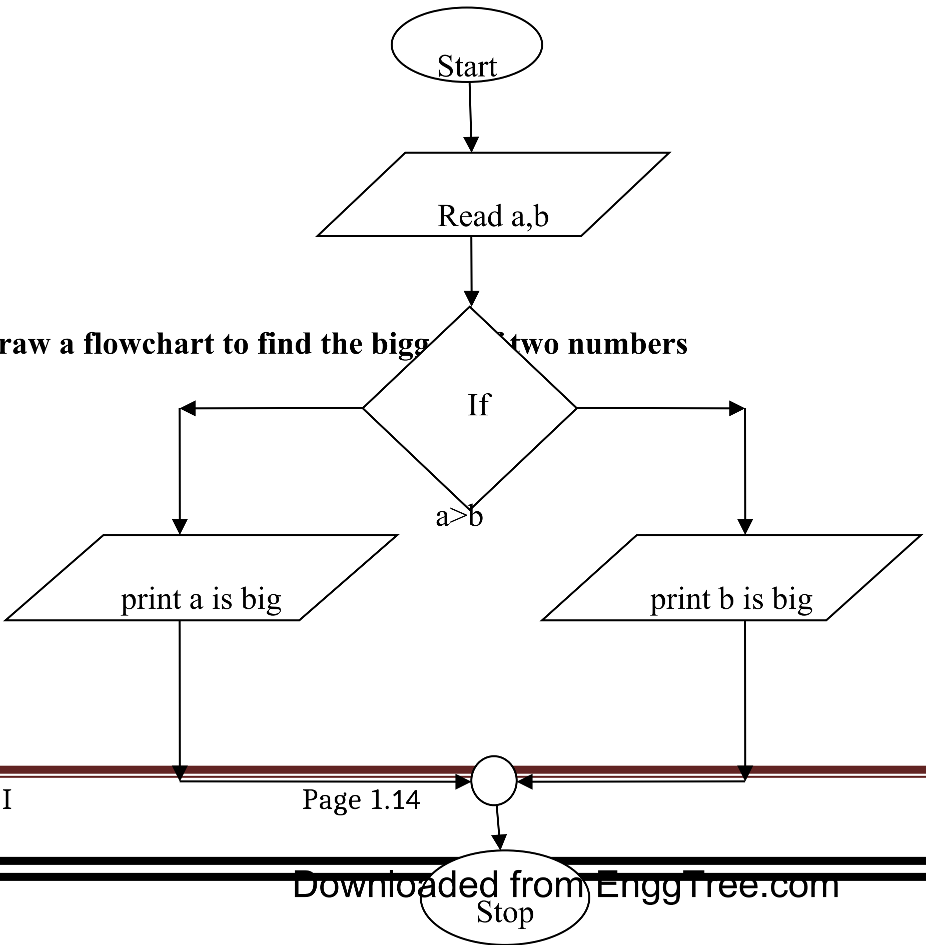
- Sometimes the program logic is quite complicated, In that case flowchart becomes complex and clumsy.
- If alterations are required the flowchart may require re-drawing completely.
- As the flowchart symbols cannot be typed, reproduction of flowchart becomes a problem.

Example:

1. Draw a flowchart to find the area of circle.

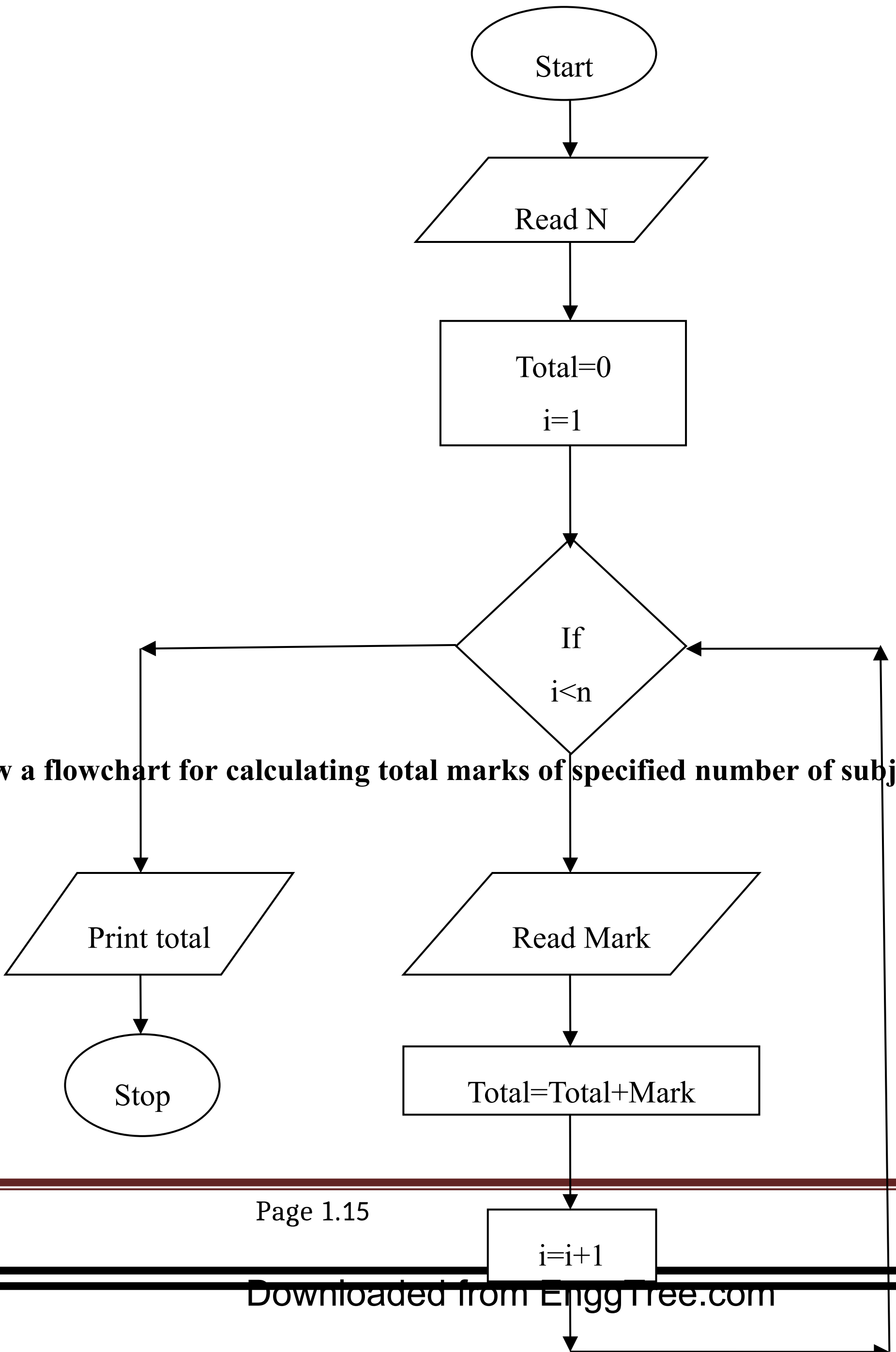


2. Draw a flowchart to find the bigg[illegible] two numbers



yes No

3. Draw a flowchart for calculating total marks of specified number of subject



Yes No

1.3.3 Programming Languages:

A computer is the ideal machine to execute computational algorithms because:

- A computer can perform arithmetic operations
- It can also perform operations only if it satisfies some condition.

A Programming language is a specialized language used to instruct the computer to solve a particular problem.

Types of Computer Programming Languages

There are three types of languages used in computer

i. *Machine Language*
ii. *Assembly Language*
iii. *Programming Language*

i) Machine Language

- Machine Language is also Known as Binary language or Low Level Language.
- Machine Language consists of 0's and 1's.
- Each Computer has its own machine language.
- The Machine Language encodes the instructions for the computer.

Ex:

01010010101

ii) Assembly Language

An Assembly language consists of English like mnemonics. There is one mnemonic for each machine instructions of the computer.

An assembler is used to convert the assembly language into an executable machine code.

Ex for An Assembly language:

```
Start
add x,y
sub x,y
…
…
End
```

Difference between Pseudo Code and Assembly Language.

S.No	Pseudo Code	Assembly Language
1.	Pseudo Codes are not compiled or executed.	Assembly Language should be compiled and should be executed.
2.	Pseudo Code should not have a specified format.	Assembly Language should have a specified format known as mnemonics.

iii) High Level Programming Language

- A High Level Programming Language consist of people language to simplify the computer algorithms.
- A High Level Programming Language allows the programmer to write sentence in the language which can be easily translated into machine language.
- The sentence written in High Level Programming Language is called as statements.

Ex for High Level Programming Language:

```
main()
{
if(x<y)
{
print("x is greater")
}
else
{
print("y is greater")
}
}
```

Some High Level Programming Language:

1. Fortan
2. C
3. C++
4. Java
5. Pearl

Define Compiler:

- A Compiler is a computer software that transforms computer code written in High Level Programming Language into Machine Language (0,1) as well as vice versa.

Define Interpreter:

- An Interpreter translates a High Level Programming Language into an immediate form of Machine Level Language. It executes instructions directly without compiling.

Define Assembler:

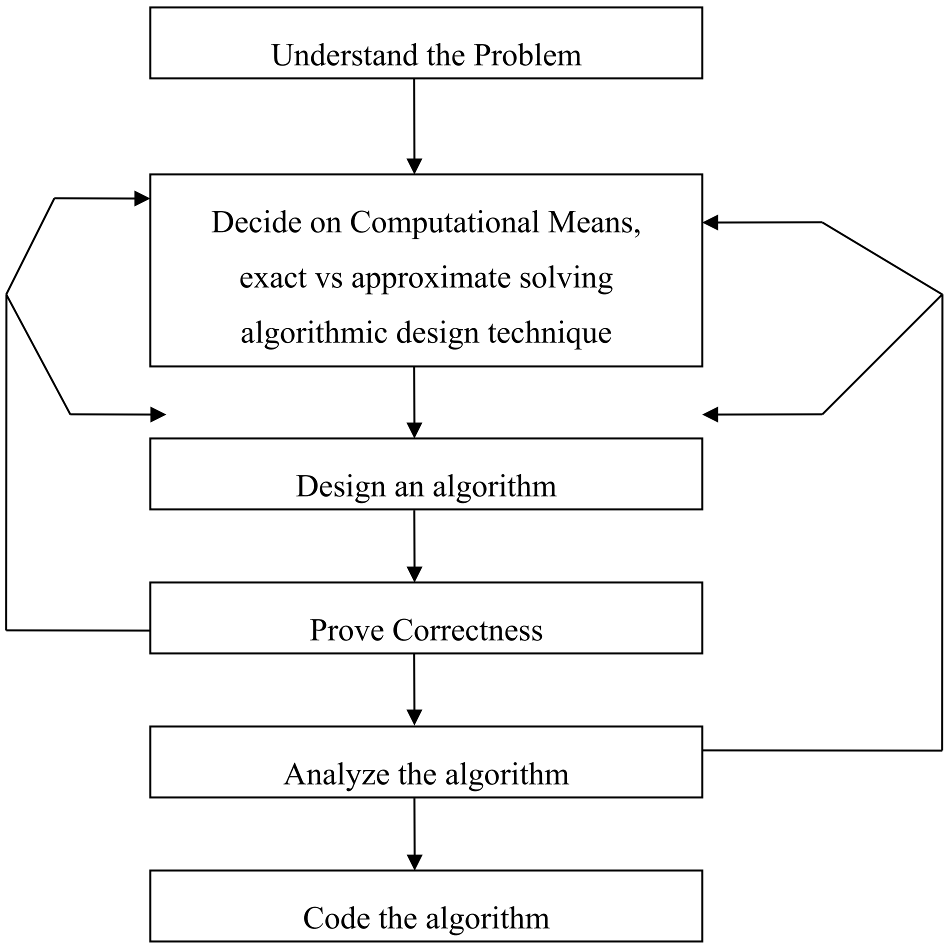
- An Assembler is used to convert the Assembly Language into an executable Machine Level Language.

Define Computer Program:

- A Computer Program is a combination of Computer algorithm and a Programming language. Most programs are English like languages.
- A Computer Program will solve the problem specified in the algorithm.

1.4 Algorithmic Problem Solving:

A sequence of steps involved in designing and analyzing an algorithm is shown in the figure below.



i) Understanding the Problem

- This is the first step in designing of algorithm.
- Read the problem's description carefully to understand the problem statement completely.
- Ask questions for clarifying the doubts about the problem.
- Identify the problem types and use existing algorithm to find solution.

- An Input to an algorithm specifies an instance of the problem and range of the input to handle the problem.

ii) Decision making

The Decision making is done on the following:

a) Ascertaining the Capabilities of the Computational Device

- After understanding the problem one need to ascertain the capabilities of the computational device the algorithm is intended for.
- **Sequential Algorithms:**
    - Instructions are executed one after another. Accordingly, algorithms designed to be executed on such machines.
- **Parallel Algorithms:**
    - An algorithm which can be executed a piece at a time on many different processing device and combine together at the end to get the correct result.

iii) Choosing between Exact and Approximate Problem Solving:

- The next principal decision is to choose between solving the problem exactly (or) solving it approximately.
- An algorithm used to solve the problem exactly and produce correct result is called an exact algorithm.
- If the problem is so complex and not able to get exact solution, then we have to choose an algorithm called an approximation algorithm. i.e., produces an approximate answer.
  Ex: extracting square roots.

iv) Deciding an appropriate data structure:

- In Object oriented programming, data structures is an important part for designing and analyzes of an algorithm.
- Data Structures can be defined as a particular scheme or structure to organize the data.

Algorithms+ Data Structures = Programs

v) Algorithm Design Techniques

- An algorithm design technique (or "strategy" or "paradigm") is a general approach to solve problems algorithmically that is applicable to a variety of problems from different areas of computing.
- First they provide guidance for designing an algorithm.
- Second, they classify algorithm according to the design idea.

vi) Methods of specifying an algorithm:

- Once an algorithm is designed, we need to specify it some way. There are three ways to represent an algorithm.
    - Pseudo code – is structured English.
    - Flowchart – a pictorial representation of an algorithm.
    - Programming Language – a high level language to instruct computer to do some action.

vii) Proving an algorithm's correctness:

- Once an algorithm has been specified, it has to be proved for its correctness.
- An algorithm has to prove with the input to yield a required output with respective of time. A Common technique is mathematical induction.

viii) Analyze the algorithm:

- For an algorithm the most important is efficiency. In fact, there are two kinds of algorithm efficiency. They are:
    - **Time efficiency**, indicating how fast the algorithm runs, and
    - Space efficiency, indicating how much extra memory it uses.
- The efficiency of an algorithm is determined by measuring both time efficiency and space efficiency.

Some factors to analyze an algorithm are:

- Time efficiency of an algorithm
- Space efficiency of an algorithm
- Simplicity of an algorithm
- Generality of an algorithm

ix) Coding of an algorithm:

- The coding / implementation of an algorithm is done by a suitable programming language like C, C++, JAVA, PYTHON.
- The transition from an algorithm to a program can be done correctly and it should be checked by testing.
- After testing, the programs are executed and this solves our problem.

1.5 Simple Strategies for Developing Algorithms (Iteration, Recursion:

Iteration:

An Iterative function is one that repeats some parts of the codes again and again until one condition gets false.

Ex:

Write a Python program to print 10 values:

```
>>>i=0
>>>while i<=10:
        print("The Value of i is",i)
        i++
>>>
```

Recursion:

A Recursive function is the one which calls itself again and again to repeat the code. The recursive function does not check any condition. It executes like normal function definition and the particular function is called again and again.

Difference between Recursion and Iteration:

S.No	Recursion	Iteration
1.	Recursion is achieved through repeated function calls.	Iteration is explicitly a repetition structure.
2.	Recursion terminates when a base case is recognized.	Iteration terminates when loop continuation test becomes false.
3.	Recursion causes a copy of function and additional memory space is occupied.	Iteration normally occurs within a loop so extra memory is omitted.

Examples:

Algorithm, Pesudocode, Flowchart for finding Factorial using Recursion

1. Algorithm to find the factorial of given number using recursion

Step1: Start

Step2: Read the value of n

Step3: Call factorial(n) and store the result in f

```
Step 4: Print the factorial f
Step5: Stop
```

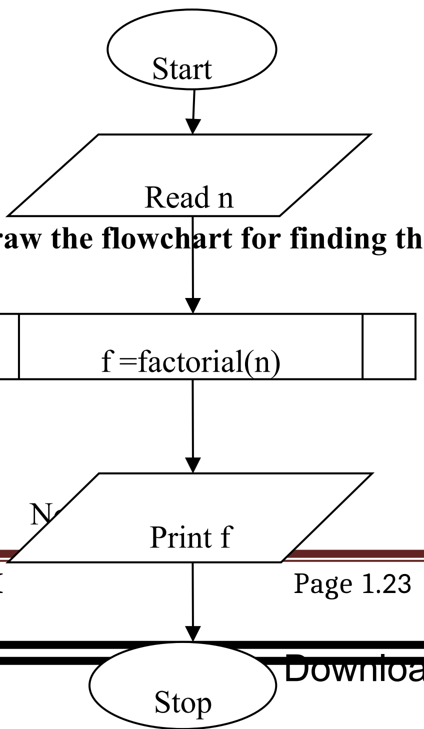
Subprogram

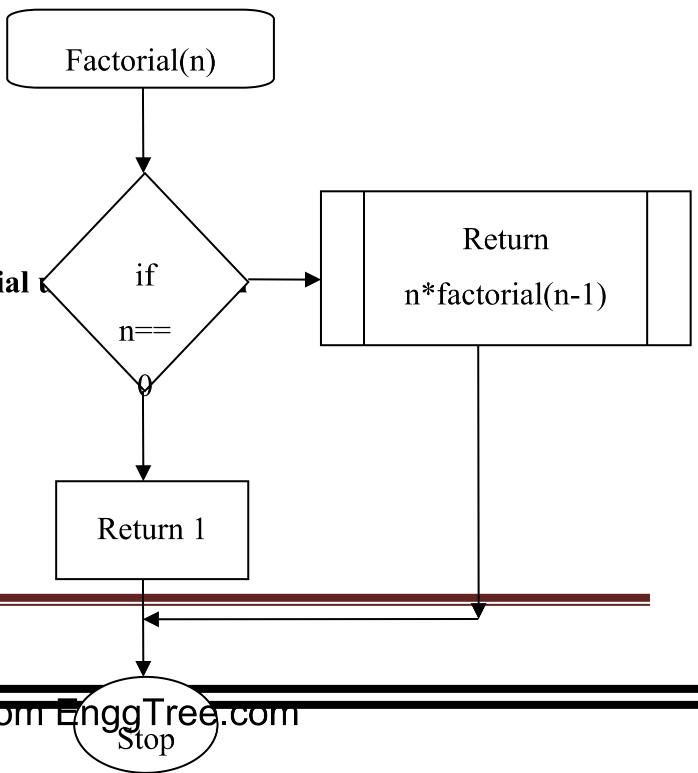
```
Step1: Start factorial(n)
Step2: If n is equal to zero then return the value 1
                    else return n*factorial(n-1).
Sep 3: Stop factorial
```

2. Write the pseudocode for finding the factorial using recursion

```
BEGIN factorial
READ n
f=factorial(n)
PRINT f
subprogram
Begin Factorial (n)
          IF n==0 THEN return 1
          ELSE return n*factorial(n-1)
END
```

3. Draw the flowchart for finding the factorial





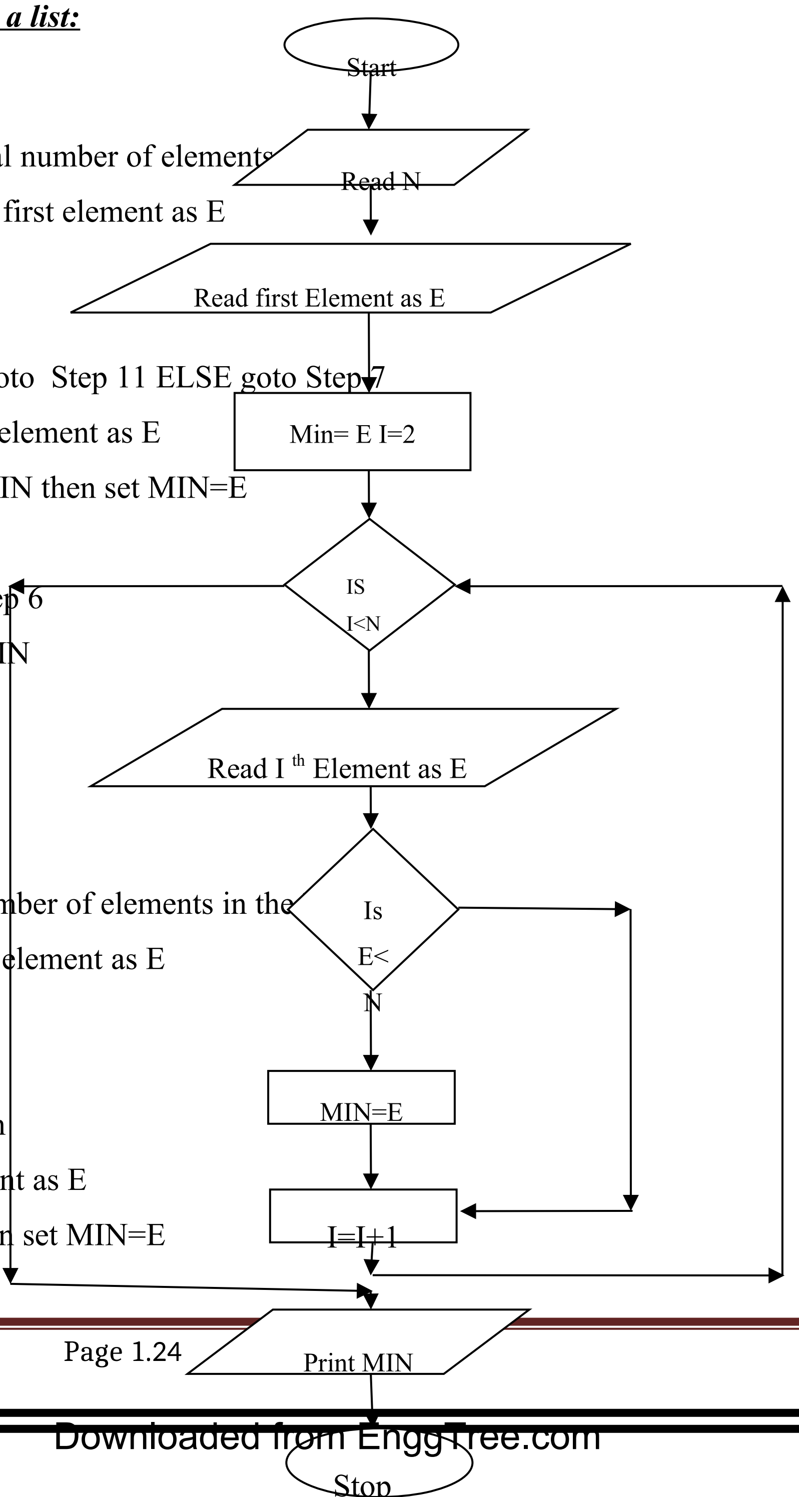
yes

ILLUSTRATIVE EXAMPLES

1.1. Find Minimum in a list:

Algorithm:

```
Step1:Start
Step2:Read total number of elements
Step3:Read the first element as E
Step4:MIN=E
Step5:SET i=2
Step6:If  i>n  goto  Step 11 ELSE goto Step 7
Step7:Read ith element as E
Step8:if  E < MIN then set MIN=E
Step9:i=i+1
Step10:goto Step 6
Step11:Print MIN
Step12:stop
```



Pseudocode

```
BEGIN
READ total number of elements in the
READ the first element as E
SET MIN=E
SET i=2
WHILE   I <= n
Read  ith element as E
if  E < MIN then set MIN=E
```

```
 i=i+1
Print MIN
END
```

Flow chart:

N

Y

N

Y

1.2. Find how to insert a card in a list of sorted cards:

Insterting a card in a list of sorted card is same as inserting an element into a sorted array. Start from the high end of the list, check to see where we want to insert the data. If the element is less than then move the high element one position and next check with next element repeat the process until the correct position and then insert the element.

Position	0	1	2	3	4	5
Original list	4	6	9	10	11	
7>11 ✗	4	6	9	10		11
7>10 ✗	4	6	9		10	11
7>9 ✗	4	6		9	10	11
7>6 ✓	4	6	7	9	10	11

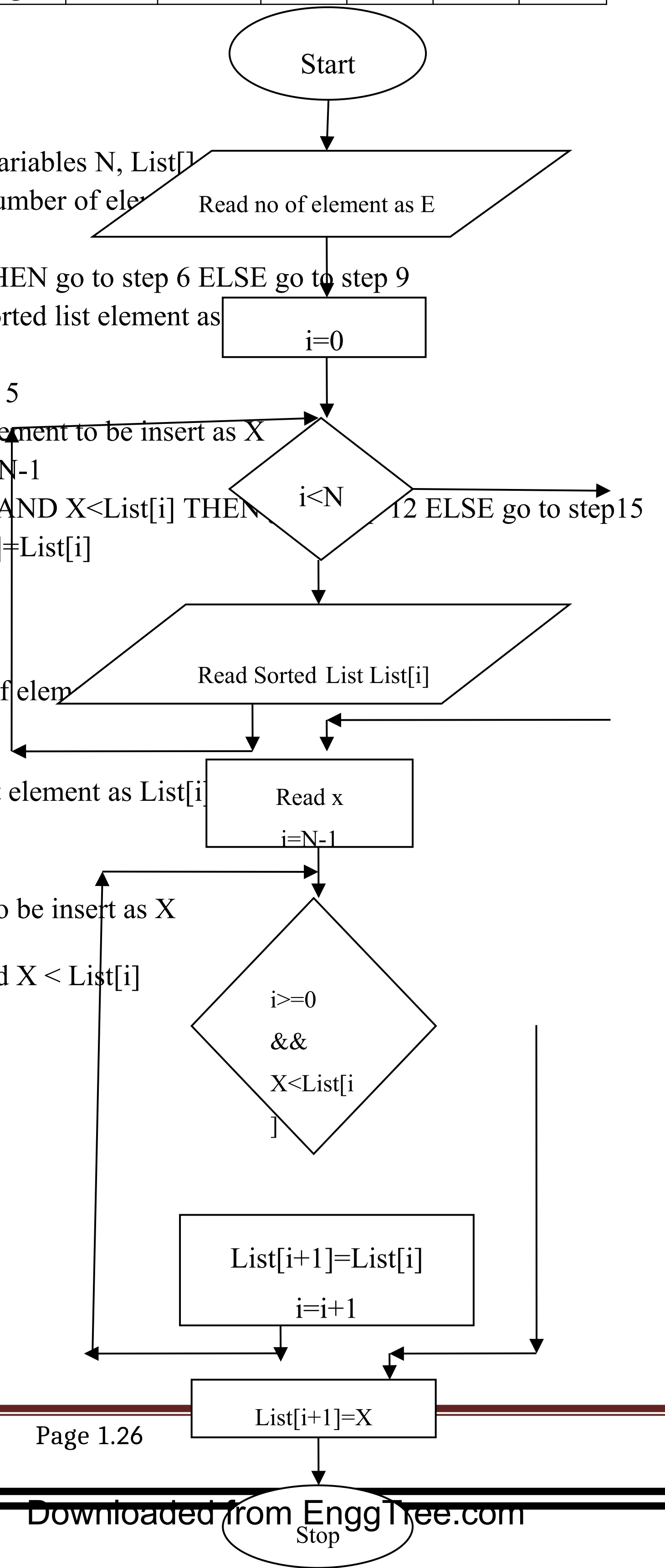
Algorithm:

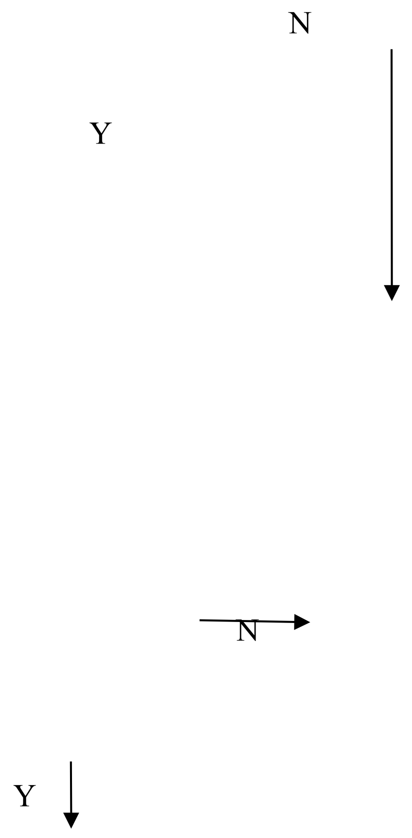
```
Step 1: Start
Step 2: Declare variables N, List[]
Step 3: READ Number of ele
Step 4: SET i=0
Step 5: IF i<N THEN go to step 6 ELSE go to step 9
Step 6: READ Sorted list element as
Step 7: i=i+1
Step 8: go to step 5
Step 9: READ Element to be insert as X
Step 10: SET i = N-1
Step 11: IF i>=0 AND X<List[i] THEN 12 ELSE go to step15
Step 12: List[i+1]=List[i]
Step 13: i=i-1
```

Pseudocode

```
READ Number of elem
SET i=0
WHILE i<N
READ Sorted list element as List[i]
i=i+1
ENDWHILE
READ Element to be insert as X
SET i = N-1
WHILE i >=0 and X < List[i]
List[i+1] =List[i]
i = i – 1
END WHILE
List[i+1] = X
```

Flow Chart





<u>1.3. Find how to guess an integer number within a range:</u>

The computer randomly selects an integer from 1 to N and ask you to guess it. The computer will tell you if each guess is too high or too low. We have to guess the number by making guesses until you find the number that the computer chose.

.

<u>Algorithm</u>

Step 1: Start

Step 2: SET Count =0

Step 3: READ Range as N

Step 4: SELECT an RANDOMN m 1 to N as R

Step 5: READ User Guessed Number as G

Step 6: Count = Count +1

Step 7: IF R==G THEN go to step 10 ELSE go to step 8

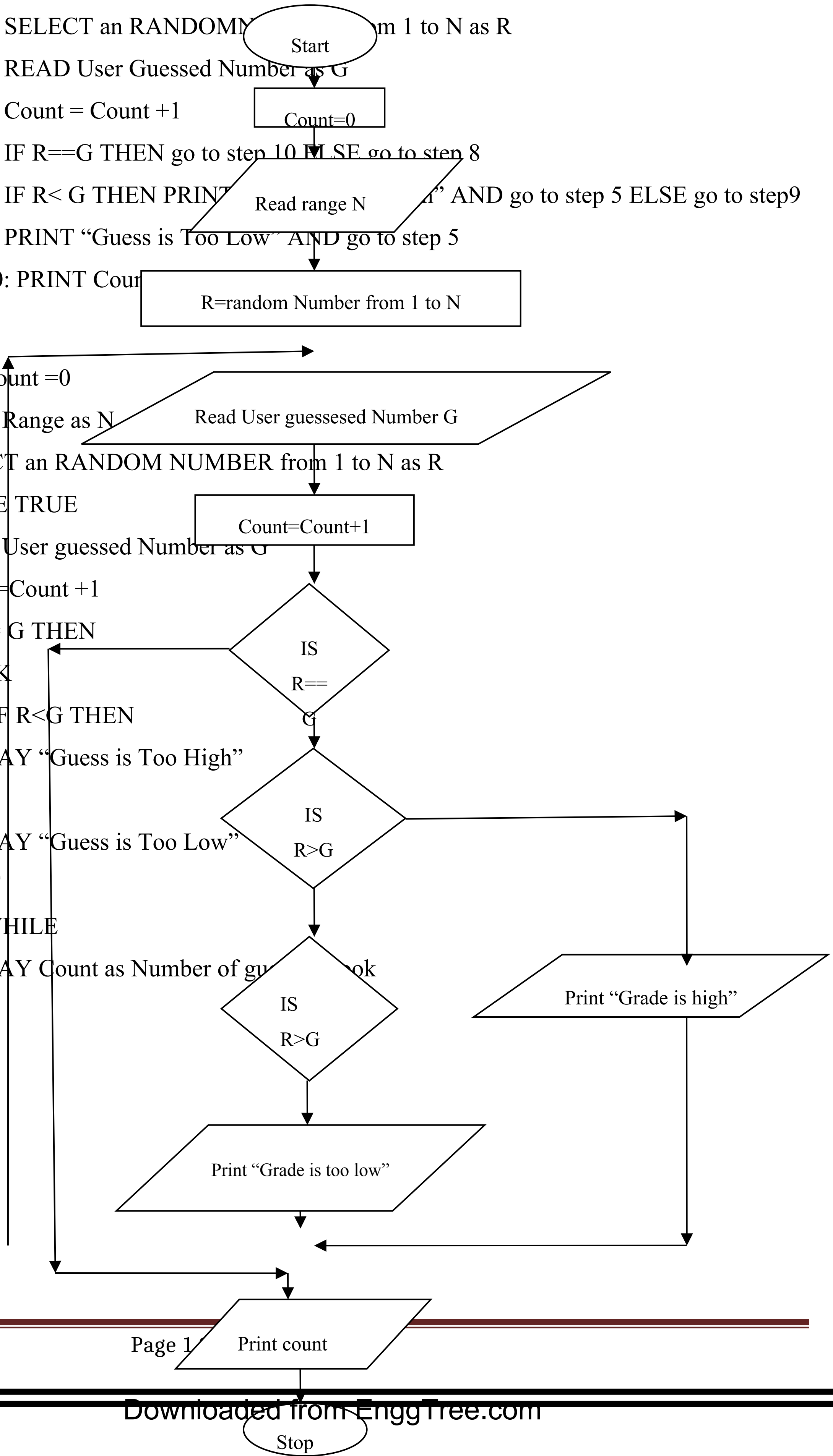
Step 8: IF R< G THEN PRINT ” AND go to step 5 ELSE go to step9

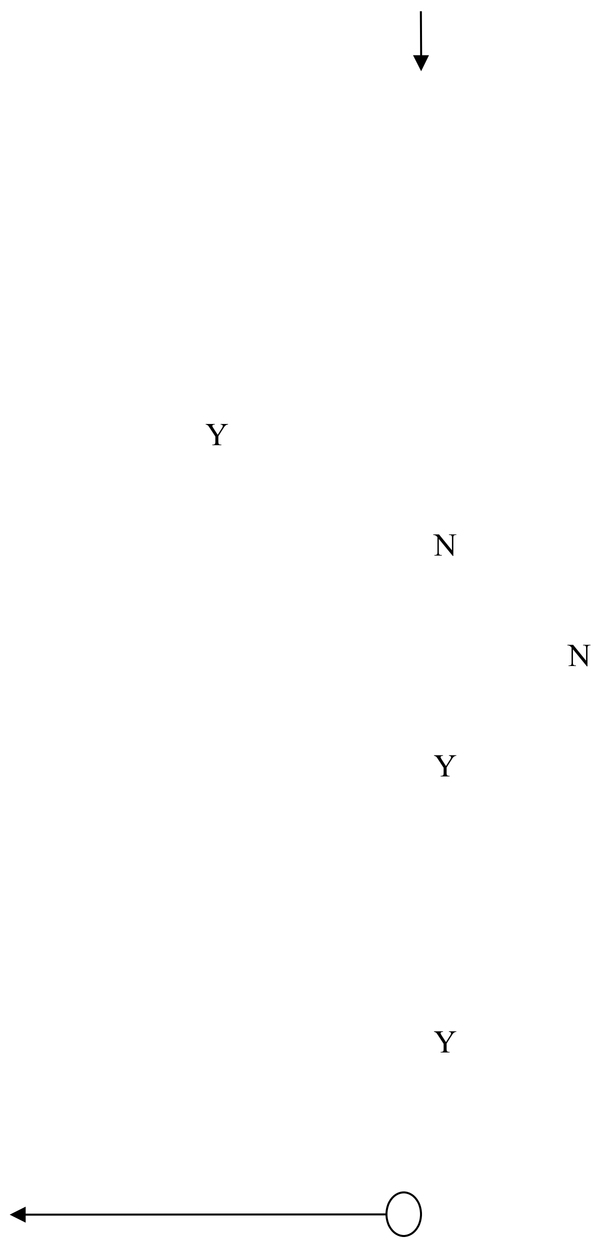
Step 9: PRINT “Guess is Too Low” AND go to step 5

Step 10: PRINT Cour

Pseudocode:

```
SET Count =0
READ Range as N
SELECT an RANDOM NUMBER from 1 to N as R
WHILE TRUE
READ User guessed Number as G
Count =Count +1
IF R== G THEN
BREAK
ELSEIF R<G THEN
DISPLAY “Guess is Too High”
ELSE
DISPLAY “Guess is Too Low”
ENDIF
ENDWHILE
DISPLAY Count as Number of gu ok
```





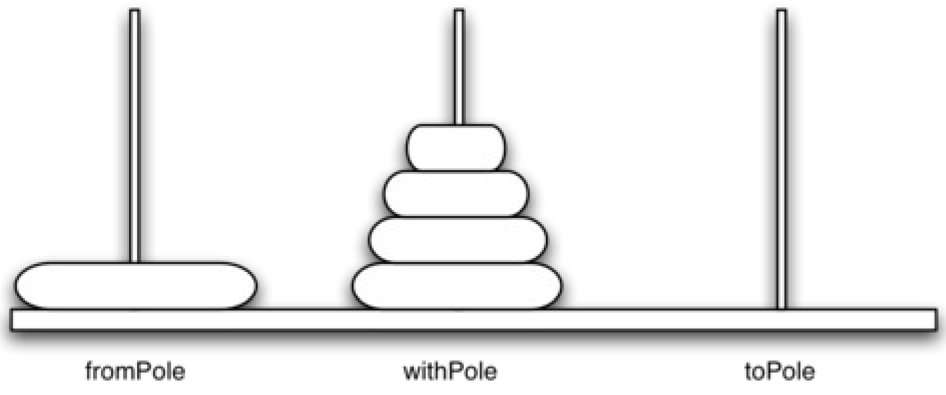
1.4 Find the towers of Hanoi:

The Tower of Hanoi puzzle was invented by the French mathematician it has three poles and a stack of disks, each disk a little smaller than the one beneath it. Their assignment was to transfer all the disks from one of the three poles to another.

- This should be done with two important constraints.
    - We can only move one disk at a time.

- We cannot place a larger disk on top of a smaller one.

Figure 1 shows an example of a configuration of disks in the middle of a move from the first pole to the third pole.



An outline to solve this problem, from the starting pole, to the goal pole, using an intermediate pole:

1. Move a tower of height-1 to an intermediate pole, using the final pole.
2. Move the remaining disk to the final pole.
3. Move the tower of height-1 from the intermediate pole to the final pole using the original pole.

we can use the three steps above recursively to solve the problem.

Python Program to solve Towers of Hanoi:

```
>>>def moveTower(height,fromPole, toPole, withPole):
    if height >= 1:
        moveTower(height-1,fromPole,withPole,toPole)
        moveDisk(fromPole,toPole)
        moveTower(height-1,withPole,toPole,fromPole)
>>>def moveDisk(fp,tp):
    print("moving disk from",fp,"to",tp)
>>>moveTower(3,"A","B","C")
```

Output:

```
moving disk from A to B
moving disk from A to C
moving disk from B to C
moving disk from A to B
moving disk from C to A
moving disk from C to B
moving disk from A to B
```

A recursive Step based algorithm for Tower of Hanoi

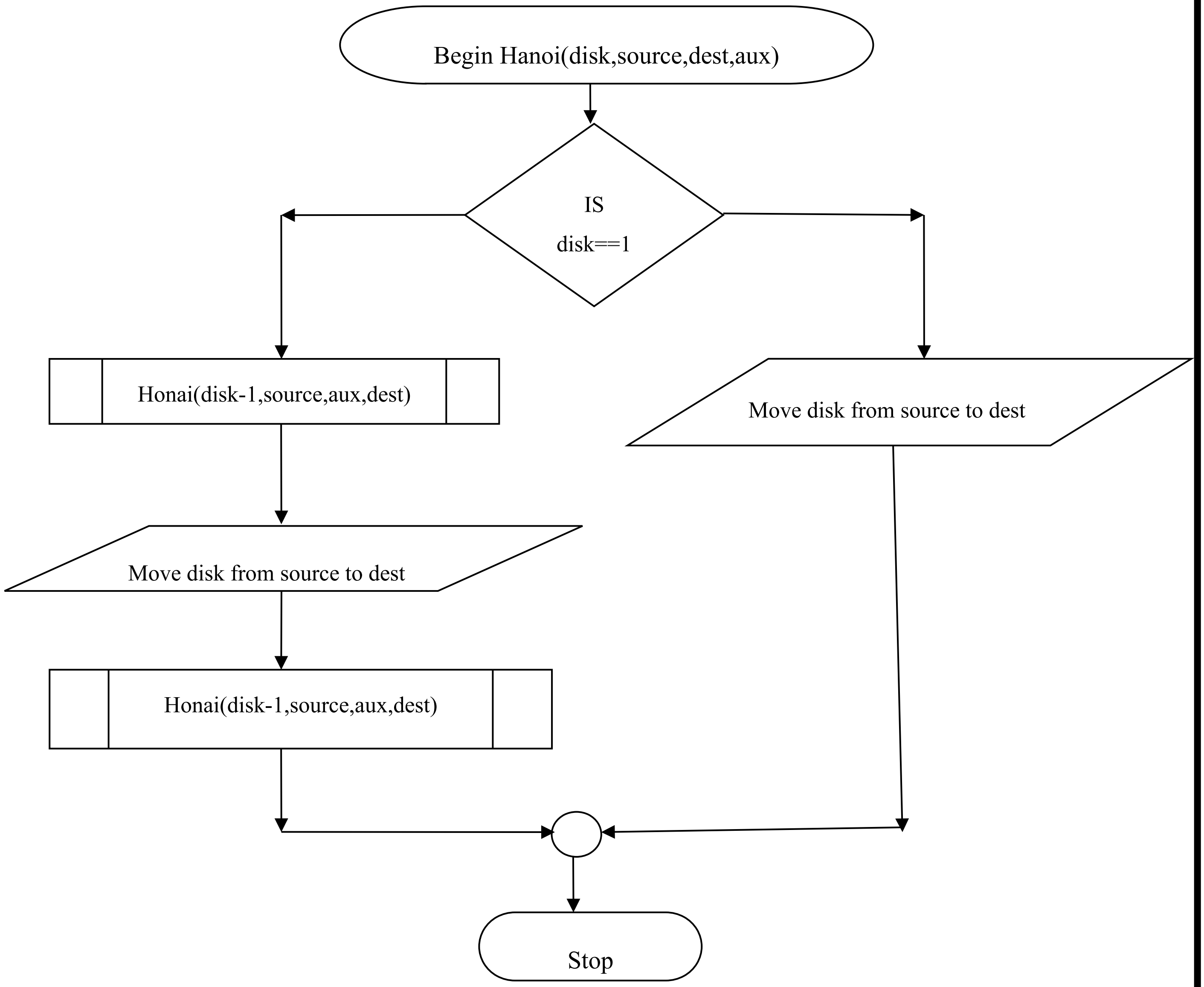
```
Step 1: BEGIN Hanoi(disk, source, dest, aux)
Step 2: IF disk == 1 THEN go to step 3 ELSE go to step 4
Step 3: move disk from source to dest AND go to step 8
Step 4: CALL Hanoi(disk - 1, source, aux, dest)
Step 5: move disk from source to dest
Step 6: CALL Hanoi(disk - 1, aux, dest, source)
Step 7: END Hanoi
```

A recursive Pseudocode for Tower of Hanoi

```
Procedure Hanoi(disk, source, dest, aux)
IF disk == 1 THEN
move disk from source to dest
ELSE
Hanoi(disk - 1, source, aux, dest) // Step 1
move disk from source to dest // Step 2
Hanoi(disk - 1, aux, dest, source) // Step 3
END IF
```

END Procedure

Flow Chart for Tower of Hanoi Algorithm


Begin Hanoi(disk,source,dest,aux)
IS
disk==1
Honai(disk-1,source,aux,dest)
Move disk from source to dest
Move disk from source to dest
Honai(disk-1,source,aux,dest)
Stop

WORKSHEETS

1.1 Draw a Flowchart and write a pesudocode to add two numbers.

Flowchart:

Flowchart:

Flowchart:

Flowchart:

Flowchart:

Flowchart:

Flowchart:

Flowchart:

Pseudocode:

TWO MARKS

1. What is an algorithm? (*University question*)

Algorithm is an ordered sequence of finite, well defined, unambiguous instructions for completing a task. It is an English-like representation of the logic which is used to solve the problem. It is a step- by-step procedure for solving a task or a problem. The steps must be ordered, unambiguous and finite in number.

2. Write an algorithm to find minimum of 3 numbers in a list. (*University question*)

ALGORITHM : Find Minimum of 3 numbers in a list

- Step 1: Start
- Step 2: Read the three numbers A, B, C
- Step 3: Compare A and B.

  If A is minimum, go to step 4 else go to step 5 Step 4: Compare A and C.

  If A is minimum, output "A is minimum" else output "C is minimum". Go to step 6.
- Step 5: Compare B and C.

  If B is minimum, output "B is minimum" else output "C is minimum".
- Step 6: Stop

3. List the building blocks of an algorithm.

- Statements
- Sequence
- Selection or Conditional
- Repetition or Control flow
- Functions

4. **Define statements. List its type**

Statements are instructions in Python designed as components for algorithmic problem solving. For example. An input statement collects a specific value from the user for a variable within the program. An output statement writes a message or the value of a program variable to the user's screen.

5. Write the pseudo code to calculate the sum and product of two numbers and display.

```
BEGIN
INITIALIZE variables sum, product, number1, number2
READ number1, number2 sum = number1+ number2
PRINT "The sum is ", sum
```

```
COMPUTE product = number1 * number2
PRINT "The Product is ", product
END
```

6. What is a function?

Functions are "self-contained" modules of code that accomplish a specific task. Functions usually "take in" data, process it, and "return" a result. Once a function is written, it can be used over and over again. Functions can be "called" from the inside of other functions.

7. Write the pseudo code to calculate the sum and product displaying the answer on the monitor screen.

```
BEGIN
INITIALIZE variables sum, product, number1, number2 of type real
READ number1, number2 sum = number1 +number2
PRINT "The sum is ", sum
COMPUTE product = number1 * number2
PRINT "The Product is ", product
END program
```

8. **Give the rules for writing Pseudo codes.**

- Write one statement per line.
- Capitalize initial keywords.
- Indent to show hierarchy.
- End multiline structure.
- Keep statements to be language independent

.

9. Give the difference between flowchart and pseudo code.

- Flowchart is a graphical representation of the algorithm.
- Pseudo code is a readable, formally English like language representation.

10. Define a flowchart.

A flowchart is a diagrammatic representation of the logic for solving a task. A flowchart is drawn using boxes of different shapes with lines connecting them to show the flow of control. The purpose of drawing a flowchart is to make the logic of the program clearer in a visual form.

11. Give an example of Iteration. a = 0

```
for i from 1 to 3     // loop three times
{
a = a + I             // add the current value of i to a
}
print a               // the number 6 is printed (0 + 1; 1 + 2; 3 + 3)
```

12. Write down the rules for preparing a flowchart.

- A flowchart should have a start and end,
- The direction of flow in a flowchart must be from top to bottom and left to right
- The relevant symbols must be used while drawing a flowchart.

13. Mention the characteristics of an algorithm.

- Algorithm should be precise and unambiguous.
- Instruction in an algorithm should not be repeated infinitely.
- Ensure that the algorithm will ultimately terminate.
- Algorithm should be written in sequence.
- Algorithm should be written in normal English.
- Desired result should be obtained only after the algorithm terminates.

14. Define the two modes in Python.

Python has two basic modes: script and interactive.

The normal mode is the mode where the scripted and finished .py files are run in the Python interpreter. Interactive mode is a command line shell which gives immediate feedback for each statement, while running previously fed statements in active memory

15. Give the various data types in Python

Python has five standard data types

- Numbers
- String
- List
- Tuple
- Dictionary

16. List out the simple steps to develop an algorithm.

Algorithm development process consists of five major steps.

- Step 1: Obtain a description of the problem.
- Step 2: Analyze the problem.
- Step 3: Develop a high-level algorithm.
- Step 4: Refine the algorithm by adding more detail.
- Step 5: Review the algorithm.

17.Give the differences between recursion and iteration

Recursion	Iteration
Function calls itself until the base condition is reached.	Repetition of process until the condition fails.
Only base condition (terminating condition) is specified.	It involves four steps: initialization, condition, execution and updation.
It keeps our code short and simple.	Iterative approach makes our code longer.
It is slower than iteration due to overhead of maintaining stack.	Iteration is faster.
It takes more memory than iteration due to overhead of maintaining stack.	Iteration takes less memory.

18. What are advantages and disadvantages of recursion?

Advantages	Disadvantages
Recursive functions make the code look clean and elegant.	Sometimes the logic behind recursion is hard to follow through.
A complex task can be broken down into simpler sub-problems using recursion.	Recursive calls are expensive (inefficient) as they take up a lot of memory and time.
Sequence generation is easier with recursion than using some nested iteration.	Recursive functions are hard to debug.

19. Define control flow statement.

A program's 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.

20. Write an algorithm to accept two number. compute the sum and print the result. (*University question*)

Step 1: Start

Step 2: READ the value of two numbers

Step 3:Compute sum of two numbers

Step 4 :Print the sum

Step 5:Stop

21. Write an algorithm to find the minimum number in a given list of numbers. (*University question)*

- Step 1: Start
- Step 2:Read the total number of element in the list as N
- Step3:Read first element as E
- Step 4:MIN=E
- Step 5:Set i=2
- Step6:IF i>n goto Step 11 ELSE goto Step 7
- Step 7:Read i th element as E
- Step 8:IF E<MIN then set MIN=e
- Step 9:i=i+1
- Step 10:goto step 6
- Step 11:print MIN
- Step12:Stop

22. Outline the logic to swap the contents of two identifiers without using the third variable *(University question)*

- Step1: Start
- Step 2:Read A,B
- Step 3 :A=A+B
- Step 4:B=A-B
- Step5:A=A-B
- Step 6:Print A ,B
- Step 7:Stop

UNIT II DATA, EXPRESSIONS, STATEMENTS

Python interpreter and interactive mode; values and types: int, float, boolean, string, and list; variables, expressions, statements, tuple assignment, precedence of operators, comments; Illustrative programs: exchange the values of two variables, circulate the values of n variables, distance between two points.

PROGRAM

The most important skill for a computer Engineer is **problem solving.** Problem solving means the ability to formulate problem, think creatively about solution clearly and accurately. The process of learning to program is an excellent opportunity to practice Problem solving skills.

"A program is a sequence of instructions that specifies how to perform a computation" .The computation might be mathematical i e Solving equations, Finding the roots of polynomial, symbolic computation such as searching and replacing text in a document.

Basic Instruction in Language

Input: Get data from Keyboard.

Output**:** Display data on the screen

Math: Perform the mathematical operations.

Conditional Execution**:** Check condition and execute certain code.

Repetition**:** Perform some action repeatedly.

2.1 INTRODUCTION

Python

Python is a high level programming language like C, C++ designed to be easy to read, and less time to write. It is an open source and it is an interpreter which directly executes the program without creating any executable file. Python is portable which means python can run on different platform with less or no modifications.

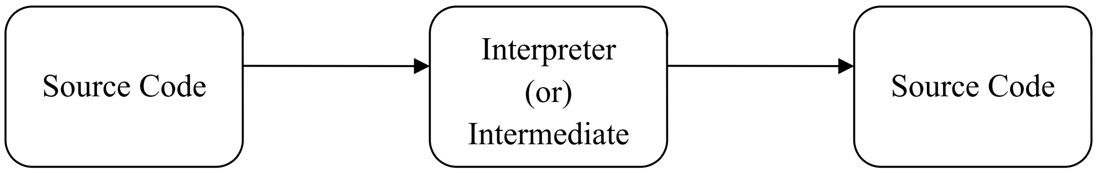
Features of Python:

- Python is publicly available open source software.
- Python is easy to learn.
- Python is easy to read.
- Python is easy to maintain.
- Python provides automatic memory management.
- Python is portable.
- Python support database and GUI(Graphical User Interface)

2.2 PYTHON INTERPRETER AND INTERACTIVE MODE

2.2.1 Python Interpreter

Python Interpreter translates high level instruction into an immediate form of machine level language. It executes instructions directly without compiling.



The Python interpreter is usually installed in C:/Program Files/Python3.6. In windows operating python can also be found in the start menu. All Programs→ Python 3.6 → Python 3.6 (Shell) and Python IDLE.

2.2.2 Python Interactive mode

- Interactive mode is a command line which gives immediate feedback for each statement. Python interactive mode can be start by two methods - Python 3.6 (Shell) and Python IDLE.
- Python 3.6 (Shell), A prompt will appear and it usually have 3 greater than signs (>>>). Each Statements can be enter after this (>>>) symbol. For continuous lines three dots (…) will represent the multiple lines.
- Python IDLE (Integrated Development for Learning Environment) which provides a user friendly console to the python users. Different colors are used to represent different keywords.
- IDLE starts by python version, after a line (>>>) three greater than symbols will be displayed. The statements will be executed in that line.

Example:

```
>>> 1+1
2
>>>5+10
15
```

2.2.3 In Script mode

- In script mode, type python program in a file and store the file with .py extension and use the interpreter to execute the content of the file which is called a script.
- Working in script mode is convenient for testing small piece of code because you can type and execute them immediately, But for more than few line we can use script since we can modify and execute in future.

2.2.4 Debugging

- Programming is error –prone, programming errors are called bugs and process of tracking them is called debugging.
- Three kinds of error can occur in a program:
    - Syntax Error
    - Runtime Error
    - Semantic error.

Syntax error:

- Syntax refers to the structure of a program and rules about the structure. Python can only execute a program if the syntax is correct otherwise the interpreter display an error message.

Runtime Error:

- The error that occurs during the program execution is called run time error.

Semantic Error:

- The computer will not generate any error message but it will not do the right thing since the meaning of the program is wrong.

2.2.5 Integrated Development Environment (IDE)

We can use any text editing software to write a Python script file. We just need to save it with the .py extension. IDE is a piece of software that provides useful features like code hinting, syntax highlighting and checking, file explorers etc. to the programmer for application development.

IDLE is a graphical user interface (GUI) that can be installed along with the Python programming language.

Example:

- Type the following code in any text editor or an IDE and save it as ***helloWorld.py***

```
print("Hello world!")
```

To run the script, type python ***helloWorld.py*** in the command window and the output as follows:

```
Hello world!
```

2.3 VALUES AND DATA TYPES

2.3.1Values

A value is one of the fundamental things; a program works with like a word or number. Some example values are 5, 83.0, and 'Hello, World!'. These values belong to different **types**: 5 is an **integer**, 83.0 is a **floating-point number**, and 'Hello, World!' is a **string.** If you are not sure what type a value has, the interpreter can tell you:

```
>>>type(5)
<class 'int'>
>>>type(83.0)
<class 'float'>
>>>type('Hello, World!')
<class 'str'>
```

In these results, the word ―class‖ is used in the sense of a category; a type is a category of values. Integers belong to the type int, strings belong to str and floating-point numbers belong to float. The values like '5' and '83.0' look like numbers, but they are in quotation marks like strings.

```
>>>type('5')
<class 'str'>
>>>type('83.0')
<class 'str'>
```

2.3.2 Standard Datatypes

A datatype is a category for values and each value can be of different types. There are 7 data types mainly used in python interpreter.

a) Integer
b) Float
c) Boolean
d) String
e) List
f) Tuple
g) Dictionary

a) Integer

Let Integer be positive values, negative values and zero.

Example:

```
>>>2+2
    4
>>>a=-20
```

print() → 20

>>> type(a) → <type 'int'>

b) Float

A floating point value indicates number with decimal point.

Example:

>>> a=20.14

>>>type(a) → <type 'float'>

c) Boolean

A Boolean variable can take only two values which are **True or False.** True and False are simply set of integer values of 1 and 0.The type of this object is bool.

Example:

>>>bool(1)

True

>>>bool(0)

False

>>>a=True

>>>type(a) → <type 'bool'>

>>>b=false   #Prints error

>>>c='True'

>>>type(c) → <type 'str'>

The boolean type is a subclass of the int class so that arithmetic using a boolean works.

>>>True + 1

2

>>>False * 85

0

# A Boolean variable should use Capital T in true & F in False and shouldn't be enclosed within the quotes.

>>>d=10>45 → #Which returns False

Boolean Operators

Boolean Operations are performed by 'AND', 'OR', 'NOT'.

Example:

True and True → True

True and False → False

True or True → True

False or True → False

not False → True

d) String

A String is an ordered sequence of characters which can be created by enclosing characters in single quotes or double quotes.

Example:

```
>>>a="Hello"
>>>type(a)
 <type 'str'>
```

Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end. The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator.

Example:

```
>>>str = 'Python Programming'
>>>print(str)                          # Prints complete string
>>>print(str[0])                       # Prints first character of the string
>>>print(str[-1])                      # Prints last character of the string
>>>print(str[2:5])                     # Prints characters starting from 3rd to 5th
>>>print(str[2:])                      # Prints string starting from 3rd character
>>>print(str * 2)                      # Prints string two times
>>>print(str + " Course")              # Prints concatenated string
```

Output

```
Python Programming
P
g
tho
thon Programming
Python ProgrammingPython Programming
Python Programming Course
```

String Functions:

For the following string functions the value of ***str1*** and ***str2*** are as follows:

>>>str1=”Hello”

>>>str2=”World”

S.No	Method	Syntax	Description	Example
1.	+	String1 + String2	It Concatenates two Strings	print(str1+str2)→ HelloWorld
2.	*	String*3	It multiples the string	str1*3 → HelloHelloHello
3.	len()	len(String)	Returns the length of the String	len(str1) →5
4.	centre()	centre(width,fullchar)	The String will be centred along with the width specified and the charecters will fill the space	str1.centre(20,+) → ++++Hello++++
5.	lower()	String.lower()	Converts all upper case into lower case	str1.lower() → hello
6.	upper()	String.upper()	Converts all lower case into upper case	str1.upper() → HELLO
7.	split()	String.split(“Char”)	splits according to the character which is present inside the function	str1.split(“+”) → H+E+L+L+O
8.	ord()	ord(String)	It converts a string in to its corresponding value	ord(‘a’)→ 96
9.	chr()	chr(Number)	It converts a number in to its corresponding String	chr(100)-->’d’
10.	rstrip()	rstrip()	It removes all the spaces at the end	rstrip(a) → it returns -1
11.	\n	print(“String\n”)	New Line Character	print(“Hello\n”)
12.	\t	print(“String\t”)	It provides Space	print(“Hello\t”)
13.	\’	print(“String\’String”)	Escape Character (/) is used to print single quote or double quote in a String	print(“Hello I\’m Fine”)
14.	\”	print(“String\”String”)		print(“Hello I\”m Fine”)

e) List

A list is an ordered set of values, where each value is identified by an index. The values that make up a list are called its elements. A list contains items separated by commas and enclosed within square brackets ([]). Lists are mutable which means the items in the list can be add or removed later.

The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1. The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.

Example:

```
>>>list = [ 'Hai', 123 , 1.75, 'vinu', 100.25 ]
>>>smalllist = [251, 'vinu']
>>>print(list)                  # Prints complete list
>>>print(list[0])               # Prints first element of the list
>>>print(list[-1])              # Prints last element of the list
>>>print(list[1:3])             # Prints elements starting from 2nd till 3rd
>>>print list([2:])             # Prints elements starting from 3rd element
>>>print(smalllist * 2)         # Prints list two times
>>>print(list + smalllist)      # Prints concatenated lists
```

Output

```
['Hai', 123, 1.75, 'vinu', 100.25]
Hai
100.25
[123, 1.75]
[1.75, 'vinu', 100.25]
[251, 'vinu', 251, 'vinu']
['Hai', 123, 1.75, 'vinu', 100.25, 251, 'vinu']
```

f) Tuple

Tuple are sequence of values much like the list. The values stored in the tuple can be of any type and they are indexed by integers. A tuple consists of a sequence of elements separated by commas. The main difference between list and tuples are:" List is enclosed in square bracket ([ ]) and their elements and size can be changed while tuples are enclosed in parenthesis (( )) and cannot be updated.

Syntax:

```
tuple_name=(items)
```

Example:

```
>>> tuple1=('1','2','3','5')
>>>tuple2=('a','b','c')
>>>tuple3='3','apple','100'
>>>print(tuple2)                         #print tuple2 elements
>>>print(tuple2[0])                      #print the first element of tuple2
```

```
>>>print(tuple2 + tuple3)          #print the concatenation of tuple2 and tuple3
>>>print(tuple3[2])                #print the second element of tuple3
```

Output:

```
('a','b','c')
('a')
('1','2','3','5','a','b','c')
('3')
```

g) Dictionary

Dictionaries are an unordered collection of items. Dictionaries are enclosed by curly braces '{ }' .The element in dictionary is a comma separated list of keys: value pairs where keys are usually numbers and strings and values can be any arbitrary python data types. The value of a dictionary can be accessed by a key. and values can be accessed using square braces '[ ]'

Syntax:

```
dict_name= {key: value}
```

Example:

```
>>>dict1={}
>>>dict2={1:10,2:20,3:30}
>>>dict3={'A':'apple','B':'200'}
>>>Dict={'Name':'john','SSN':4576,'Designation':'Manager'}
```

2.4 PYTHON KEYWORDS:

Keywords are reserved words that cannot be used as ordinary identifiers. All the keywords except True, False and None are in lowercase .

False	class	finally	is	return	None
continue	for	lambda	try	True	def
from	nonlocal	while	and	del	global
not	with	as	elif	if	or
yield	assert	else	import	pass	break
except	in	raise	and	print	exec

2.5 PYTHON IDENTIFIERS:

Identifiers are names for entities in a program such as class, variables and functions etc.

Rules for defining Identifiers:

Identifiers can be composed of uppercase, lowercase letters, underscore and digits bur should start only with an alphabet or an underscore.

- Identifiers can be a combination of lowercase letters (a to z) or uppercase letters (A to Z) or digits or an underscore.
- Identifiers cannot start with digit
- Keywords cannot be used as identifiers.
- Only (_) underscore special symbol can be used.

Valid Identifiers: sum total _ab_ add_1

Invalid Identifies: 1x x+y if

2.6 VARIABLES

A variable is nothing but a reserved memory location to store values. A variable in a program gives data to the computer.

Ex:

```
>>>b=20
>>>print(b)
```

2.7 PYTHON INDENTATION

Python uses indentation. Block of code starts with indentation and ends with the unintended line. Four whitespace character is used for indentation ans is preferred over tabs.

Ex:

```
x=1
if  x==1:
        print("x is 1")
Result:
x is 1
```

2.8 EXPRESSIONS

An Expression is a combination of values, variables and operators.

Ex:

```
>>>10+20
12
```

2.9 STATEMENTS

A Statement is an instruction that a python interpreter can execute.IN python enf of a statement is marked by a newline character.

c=a+b

Multiline statement can be used in single line using semicolon(;)

```
>>a=1;b=10;c=a +b
```

Ex:

```
>>>b=20
>>>print(b)
>>>print("\"Hello\"")
```

Difference between a Statement and an Expression

A statement is a complete line of code that performs some action, while an expression is any section of the code that evaluates to a value. Expressions can be combined —horizontally into larger expressions using operators, while statements can only be combined vertically by writing one after another, or with block constructs. Every expression can be used as a statement, but most statements cannot be used as expressions

2.10 TUPLE ASSIGNMENTS

Tuple Assignment means assigning a tuple value into another tuple.

Ex:

```
 t=('Hello','hi')
>>>m,n=t
>>>print(m) → Hello
>>>print(n) → hi
>>>print(t) → Hello,hi
```

In order to interchange the values of the two tuples the following method is used.

```
>>>a=('1','4')
>>>b=('10','15')
>>>a,b=b,a
>>>print(a,b)
(('10','15'), ('1','4'))
```

2.11 COMMENTS

Comments are non-executable statements which explain what program does. There are two ways to represent a comment.

2.11.1 Single Line Comment

Begins with # hash symbol

Ex:

```
>>>print("Hello world")                                    # prints the string
```

2.11.2 Multi Line Comment

Multi line comment begins with a double quote and a single quote and ends with the same

Ex:

>>>"'This is a multi line comment'"

2.12 OPERATORS:

Operators are the construct which can manipulate the value of operands.

Eg: 4+5=9

Where 4, 5, 9 are operand

+ is Addition Operator

= is Assignment Operator

Types of Operator:

1. Arithmetic Operator
2. Comparison Operator (or) Relational Operator
3. Assignment Operator
4. Logical Operator
5. Bitwise Operator
6. Membership Operator
7. Identity Operator

1. Arithmetic Operator

It provides some Arithmetic operators which perform some arithmetic operations

Consider the values of a=10, b=20 for the following table.

Operator	Meaning	Syntax	Description
+	Addition	a+b	It adds and gives the value 30
-	Subtraction	a-b	It subtracts and gives the value -10
*	Multiplication	a*b	It multiplies and gives the value 200
/	Division	a/b	It divides and gives the value 0.5
%	Modulo	a%b	It divides and return the remainder 0
**	Exponent	a**b	It performs the power and return 10^{20}
//	Floor	a//b	It divides and returns the least quotient

Example Program:

1.Write a Python Program with all arithmetic operators

```
>>>num1 = int(input('Enter First number: '))
>>>num2 = int(input('Enter Second number '))
>>>add = num1 + num2
```

```
>>>dif = num1 - num2
>>>mul = num1 * num2
>>>div = num1 / num2
>>>modulus = num1 % num2
>>>power = num1 ** num2
>>>floor_div = num1 // num2
>>>print('Sum of ',num1 ,'and' ,num2 ,'is :',add)
>>>print('Difference of ',num1 ,'and' ,num2 ,'is :',dif)
>>>print('Product of' ,num1 ,'and' ,num2 ,'is :',mul)
>>>print('Division of ',num1 ,'and' ,num2 ,'is :',div)
>>>print('Modulus of ',num1 ,'and' ,num2 ,'is :',modulus)
>>>print('Exponent of ',num1 ,'and' ,num2 ,'is :',power)
>>>print('Floor Division of ',num1 ,'and' ,num2 ,'is :',floor_div)
```

Output:

```
>>>
Enter First number: 10
Enter Second number 20
Sum of  10 and 20 is : 30
Difference of  10 and 20 is : -10
Product of 10 and 20 is : 200
Division of  10 and 20 is : 0.5
Modulus of  10 and 20 is : 10
Exponent of  10 and 20 is : 100000000000000000000
Floor Division of  10 and 20 is : 0
>>>
```

2. Comparison Operator (or) Relational Operator

These operators compare the values and it returns either True or False according to the condition. Consider the values of a=10, b=20 for the following table.

Operator	Syntax	Meaning	Description
==	a==b	Equal to	It returns false
!=	a!=b	Not Equal to	It returns true
>	a>b	Greater than	It returns false
<	a<b	Lesser than	It returns true
>=	a>=b	Greater than or Equal to	It returns false
<=	a<=b	Lesser than or Equal to	It returns true

3. Assignment Operator

Assignment operators are used to hold a value of an evaluated expression and used for assigning the value of right operand to the left operand.

Consider the values of a=10, b=20 for the following table.

Operator	Syntax	Meaning	Description
=	a=b	a=b	It assigns the value of b to a.
+=	a+=b	a=a+b	It adds the value of a and b and assign it to a.
- =	a-=b	a=a-b	It subtract the value of a and b and assign it to a.
*=	a*=b	a=a*b	It multiplies the value of a and b and assign it to a.
/=	a/=b	a=a/b	It divides the value of a and b and assign it to a.
%=	a%=b	a=a%b	It divides the value of a and b and assign the remainder to a.
**=	a**=b	a=a**b	It takes 'a' as base value and 'b' as its power and assign the answer to a.
//=	a//=b	a=a//b	It divides the value of a and b and takes the least quotient and assign it to a.

4. Logical Operator

Logical Operators are used to combine two or more condition and perform logical operations using Logical AND, Logical OR, Logical Not.

Consider the values of a=10, b=20 for the following table.

Operator	Example	Description
AND	if(a<b and a!=b)	Both Conditions are true
OR	if(a<b or a!=b)	Anyone of the condition should be true
NOT	not (a<b)	The condition returns true but not operator returns false

5. Bitwise Operator

Bitwise Operator works on bits and performs bit by bit operation.

Consider the values of a=60, b=13 for the following table.

Operator	Syntax	Example	Description
&	Binary AND	a&b= 12	It do the and operation between two operations
|	Binary OR	a|b= 61	It do the or operation between two operations
~	Binary Ones Complement	~a=61	It do the not operation between two operations
<<	Binary Left Shift	<<a	It do the left shift operation
>>	Binary Right Shift	>>a	It do the right shift operation

A	B	A&B	A|B	~A
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

1. Write a Python Program with all Bitwise Operator

```
a = 10                                              # 10 = 0000 1010
b = 20                                              # 20 = 0001 0100
c = 0
c = a & b;                                         # 0 = 0000 0000
print ("Line 1 - Value of c is ", c)
c = a | b;                                          # 30 = 0001 1110
print ("Line 2 - Value of c is ", c)
c = ~a;                                             # -11 = 0000 1011
print ("Line 3 - Value of c is ", c)
c = a << 2;                                        # 40 = 0011 1000
print ("Line 4 - Value of c is ", c)
c = a >> 2;                                        # 2 = 0000 0010
print ("Line 5 - Value of c is ", c)
```

Output:

```
Line 1 - Value of c is  12
Line 2 - Value of c is  61
Line 3 - Value of c is  -61
Line 4 - Value of c is  240
Line 5 - Value of c is  15
```

6. Membership Operator

Membership Operator test for membership in a sequence such as strings, lists or tuples. Consider the values of a=10, b=[10,20,30,40,50] for the following table.

Operator	Syntax	Example	Description
in	value *in* String or List or Tuple	a in b returns True	If the value is '**in**' the list then it returns True, else False
not in	value *not in* String or List or Tuple	a not in b returns False	If the value is '**not in**' the list then it returns True, else False

Example:

x='python programming'

```
print('program' not in x)
print('program'  in  x)
print(' Program' in  x)
```

Output:

```
False
True
False
```

7. Identity Operator

Identity Operators compare the memory locations of two objects.

Consider the values of a=10, b=20 for the following table.

Operator	Syntax	Example	Description
is	variable 1 *is* variable 2	a is b returns False	If the variable 1 value **is** pointed to the same object of variable 2 value then it returns True, else False
is not	variable 1 *is not* variable 2	a is not b returns False	If the variable 1 value **is not** pointed to the same object of variable 2 value then it returns True, else False

Example:

```
x1=7
y1=7
x2='welcome'
y2='Welcome'
print (x1 is y1)
print (x2 is y2)
print(x2 is not y2)
```

Output:

```
True
False
True
```

2.13 PRECEDENCE OF PYTHON OPERATORS

The combination of values, variables, operators and function calls is termed as an expression. Python interpreter can evaluate a valid expression. When an expression contains more than one operator, the order of evaluation depends on the **Precedence** of operations.

For example, Multiplication has higher precedence than Subtraction.

```
>>> 20 – 5*3
5
```

But we can change this order using Parentheses () as it has higher precedence.

```
>>> (20 - 5) *3
45
```

The operator precedence in Python are listed in the following table.

Table :Operator precedence rule in Python

S. No	Operators	Description
1.	()	Parentheses
2.	**	Exponent
3.	+x, -x, ~x	Unary plus, Unary minus, Bitwise NOT
4.	*, /, //, %	Multiplication, Division, Floor division, Modulus
5.	+, -	Addition, Subtraction
6.	<<, >>	Bitwise shift operators
7.	&	Bitwise AND
8.	^	Bitwise XOR
9.	|	Bitwise OR
10.	==, !=, >, >=, <, <=,	is, is not, in, not in Comparison, Identity, Membership operators
11.	not	Logical NOT
12.	and	Logical AND
13.	or	Logical OR

2.14 ASSOCIATIVITY OF PYTHON OPERATORS

If more than one operator exists in the same group. These operators have the same precedence. When two operators have the same precedence, associativity helps to determine which the order of operations. Associativity is the order in which an expression is evaluated that has multiple operator of the same precedence. Almost all the operators have left-to-right associativity. For example, multiplication and floor division have the same precedence. Hence, if both of them are present in an expression, left one evaluates first.

Example:

```
>>> 10 * 7 // 3
23
>>> 10 * (7//3)
20
>>> (10 * 7)//3
23
```

10 * 7 // 3 is equivalent to (10 * 7)//3.

Exponent operator ** has right-to-left associativity in Python.

```
>>> 5 ** 2 ** 3
390625
>>> (5** 2) **3
15625
>>> 5 **(2 **3)
390625
```

2 ** 3 ** 2 is equivalent to 2 ** (3 ** 2).

PRECEDANCE OF ARITHMETIC OPERATORS

Precedence	Operator	Description
1	**, ()	Exponent, Inside Parenthesis
2	/, *, %, //	Division, Multiplication, Modulo, Floor
3	+, -	Addition, Subtraction

2.15 FUNCTIONS

A **function** is a named sequence of statements that performs a specific task. Functions help programmers to break complex program into smaller manageable units. It avoids repetition and makes code reusable.

Types of Functions

Functions can be classified into

- BUILT-IN FUNCTIONS
- USER DEFINED FUNCTIONS

2.15.1 BUILT-IN FUNCTIONS

The Python interpreter has a number of functions that are always available for use. These functions are called built-in functions. The syntax is

```
function_name(parameter 1, parameter 2)
```

i) type()

```
>>>type(25)
<class 'int'>
```

The name of the function is **type().** The expression in parentheses is called the **argument** of the function. The result, for this function, is the type of the argument. Function takes an argument and returns a result. The result is also called the **return value**.

Python provides functions that convert values from one type to another. The **int()** function takes any value and converts it to an integer, if it can, or it shows error otherwise:

ii) Casting:

```
>>>int('25')
25
>>>int('Python')
valueError: invalid literal for int(): Python
```

int() can convert floating-point values to integers, but it doesn't round off; it chops off the fraction part:

```
>>>int(9.999999)
9
>>>int(-2.3)
-2
```

float() converts integers and strings to floating-point numbers:

```
>>>float(25)
25.0
>>>float('3.14159')
3.14159
```

Finally, *str()* converts its argument to a string:

```
>>>str(25)
```

```
'25'
```

iii) range()

The range() constructor returns an immutable sequence object of integers between the given start integer to the stop integer.

Python's range() Parameters

The range() function has two sets of parameters, as follows:

i) range(stop)

stop: Number of integers (whole numbers) to generate, starting from zero.

eg. range(3) == [0, 1, 2].

ii) range([start], stop[, step])

start: Starting number of the sequence.

stop: Generate numbers up to, but not including this number.

step: Difference between each number in the sequence.

Example:

```
>>>range(10)
[0,1,2,3,4,5,6,7,8,9]
>>>range(5,10)
[5,6,7,8,9]
>>>range[10,1,-2]
[10,8,6,4,2]
```

iv) Printing to the Screen

print() function will prints as strings ,everything in a comma separated sequence of expressions, and it will separate the results with single blanks by default.

Example:

```
>>> x=10
>>> y=7
>>>print('The sum of',x, 'plus', y, 'is', x+y)
```

Output:

```
The sum of 10 plus 7 is 17
```

print statement can pass zero or more expressions separated by commas.

v) Reading Keyboard Input:

Python provides a built-in function to read a line of text as a standard input, which by default comes from the keyboard. This function is:

- input()

The input() Function

The *input([prompt])* function print the string which is in the prompt and the cursor point will wait for an input.

```
>>>str = input("Enter your input: ");
Enter your input: 10
>>> print("The input is : ", str)
10
```

2.15.2 USER-DEFINED FUNCTIONS

If the user create own functions then these functions are called ***user-defined functions***.

Function Definition

Syntax:

A function definition is a heart of the function where we will write main operation of that function.

Syntax:

```
def function_name(parameter 1, parameter 2):
        #Function Definition statements
```

Components of function definition

1. Keyword **def** marks the start of function header.
2. A function name to uniquely identify it.
3. **Parameters** (arguments) through which pass values to a function. They are **optional.**
4. A colon (:) to mark the end of function header.
5. **Optional documentation string** (docstring) to describe what the function does.
6. One or more valid python statements that make up the function body. Statements must have same indentation level.
7. An **optional return** statement to return a value from the function.

Example:

```
>>>def welcome(person_name):
        """This function welcome the person passed in as parameter"""
        print(" Welcome " , person_name , " to learn Python")
```

Using Function or Function Call

Once we have defined a function, we can call it from another function, program or even the Python prompt. To call a function we simply type the function name with appropriate parameters.

Syntax:

```
function_name(parameter 1, parameter 2)
```

Example:

```
>>> welcome('Students')
```

Output:

```
Welcome Students to learn Python.
```

The return statement:

The return statement is used to exit a function and go back to the place from where it was called.

Syntax:

```
return variable_name
```

This statement can contain expression which gets evaluated and the value is returned. If there is no expression in the statement or the return statement itself is not present inside a function, then the function will return the None object.

Example:

```
>>>def absolute_value(num):
        """This function returns the absolute value of the entered number"""
        if num >= 0:
                return num
        else:
                return -num

>>>print(absolute_value(5))
>>>print(absolute_value(-7))
```

Output:

```
5
7
```

2.15.3 FLOW OF EXECUTION

The order in which statements run is **called the flow of execution**. Execution always begins at the first statement of the program. Statements are run one at a time, in order from top to bottom. Function definitions do not alter the flow of execution of the program, but when the function is called Instead of going to the next statement, the flow jumps to the body of the function, runs the statements there, and then comes back to pick up where it left off.

2.15.4 PARAMETERS AND ARGUMENTS

Inside the function, the a**rguments** are assigned to variables **called parameters**. Here is a definition for a function that takes an argument:

Function Arguments

Types of Formal arguments:

- Required arguments
- Default arguments
- Keyword arguments
- Variable-length arguments

Required Arguments

Required arguments are the arguments passed to a function in correct positional order. Here, the number of arguments in the function call should match exactly with the function definition.

Example:

```
>>>def add(a,b):     # add() needs two arguments, if not it shows error
        return a+b
>>>a=10
>>>b=20
>>>print("Sum of ", a ,"and ", b, "is" , add(a,b))
```

Output:

```
Sum of 10 and 20 is 30
```

Default Arguments:

A default argument is an argument that assumes a default value if a value is not provided in the function call for that argument.

Example:

```
>>>def add(a,b=0):
        print ("Sum of ", a ,"and ", b, "is" ,a+b)
>>>a=10
>>>b=20
>>>add(a,b)
>>>add(a)
```

Output:

```
Sum of 10 and 20 is 30
Sum of 10 and 0 is 10
```

Keyword Arguments:

Keyword arguments are related to the function calls. When you use keyword arguments in a function call, the caller identifies the arguments by the parameter name.

Example:

```
>>>def add(a,b):
        print ("Sum of ", a ,"and ", b, "is" ,a+b)
>>>a=10
>>>b=20
>>>add(b=a,a=b)
```

Output:

```
Sum of 20 and 10 is 30
```

Variable-Length Arguments:

The special syntax *args in function definitions in python is used to pass a variable number of arguments to a function. It is used to pass a non-keyworded, variable-length argument list.

- The syntax is to use the symbol * to take in a variable number of arguments; by convention, it is often used with the word args.

- What *args allows you to do is take in more arguments than the number of formal arguments that you previously defined. With *args, any number of extra arguments can be tacked on to your current formal parameters

Example:

```
>>>def myFun(*argv):
        for arg in argv:
                print (arg)
>>>myFun('Hello', 'Welcome', 'to', 'Learn Python')
```

Output:

```
Hello
Welcome
to
Learn Python
```

The Anonymous Functions or Lambda Functions

In Python, anonymous function is a function that is defined without a name. While normal functions are defined using the def keyword, in Python anonymous functions are defined using the lambda keyword. Hence, anonymous functions are also called lambda functions.

Syntax:

```
lambda arguments: expression
```

Example:

```
>>>double = lambda x: x * 2
        print(double(5))
```

Output:

```
10
```

In the above program, lambda x: x * 2 is the lambda function. Here x is the argument and x * 2 is the expression that gets evaluated and returned.

The same Anonymous function can be written in **normal function** as

```
>>>def double(x):
        return x * 2
>>>double(5)
```

2.16 MODULES

Modules

a) Importing Modules

b) Built-in Modules

Define Module

A module allows you to logically organize the python code. Grouping related code into a module makes the code easy to use. A module is a file consisting python code. A module can define functions, classes and variables. A module can also include runnable code.

Example:

The Python code for a module **add** normally resides in a file named **addition.py**.

support.py

```
>>>def add(a,b):
        result=a+b
        return result
```

a) Importing Modules

We can invoke a module by two statements

i. import statement
ii. from…import statement

i) The import Statement

You can use any Python source file as a module by executing an import statement in some other Python source file.

Syntax:

```
import  module
```

Example:

```
import addition                        # Import module addition
addition.add(22,33)              # Now we can call the function in that module as
```

Result:

```
Addition of two number is 55
```

ii) The from...import Statement

Python's from statement lets to import specific attributes from a module into the current namespace. Multiple function can be imported by separating by their names with commas .

Syntax:

```
from  module_name  import function_name
```

Example:

addition.py

```
>>>def add(a,b):
        result=a+b
        return result
```

subt.py

```
>>>def sub(a,b):
        result=a-b
        return result

>>>from example import add,sub
>>> print(addition.add(9,2))
11
>>>print(subt.sub(5,2))
3
```

b) Built-in Modules

Python have many built-in modules such as random, math, os, date, time, URLlib21.

i*) random module:*

This module is used to generate random numbers by using randint function.

Ex:

```
import random
print(random.randint(0,5))
print(random.random())
print( random.random()*10)
my_data=[156,85,"john',4.82,True]
print(random.choice(my_data))
```

Output:

```
1
0.675788
5.2069
4.82
```

ii) math module:

This math module provides access to mathematical constants and functions.

Ex:

```
>>>import math
>>>math.pi                          #Pi, 3.14
```

```
>>>math.e                    #Euler's number
>>>math.degrees(2)           #2 rads=114.59 degrees
>>>math.sin(2)               #sin of 2
>>> math.cos(0.5)            #cos of 0.5
>>> math.tan(0.23)           #tan of 0.23
>>> math.sqrt(49)            #sqrt of 49 is 7
>>>math.factorial(5)         #factorial of 5 is 1*2*3*4*5=120
```

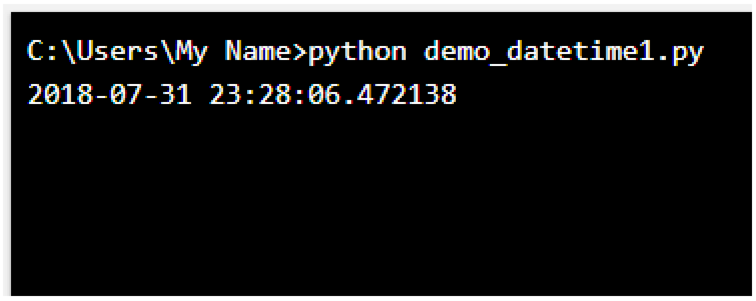
<u>iii) date & time module</u>

It is useful for web development. This is module is use to display date and time

<u>Ex:</u>

```
>>>import datetime
>>> x = datetime.datetime.now()
>>>print(x)
```

<u>Output:</u>



```
C:\Users\My Name>python demo_datetime1.py
2018-07-31 23:28:06.472138
```

<u>iii)Calender module:</u>

This module is used to display calendar

```
>>>import calendar
>>>cal=calendar.month(2019,5)
>>>print(cal)
```

<u>Output:</u>

Calendar is displayed

ILLUSTRATIVE EXAMPLES

Note- *Always get an Input from the USER.*

1. Write a python Program to find the area of a triangle

```
a = float(input('Enter first side: '))
b = float(input('Enter second side: '))
c = float(input('Enter third side: '))                # calculate the semi-perimeter
s = (a + b + c) / 2                                    # calculate the area
area = (s*(s-a)*(s-b)*(s-c)) ** 0.5
print("The area of the triangle is f",area)           # displays the area
```

Output:

```
>>> Enter first side: 6
    Enter second side: 8
    Enter third side: 10
   The area of the triangle is f 24.0
>>>
```

2. Write a python Program to find the quadratic equation

```
import cmath                                  # import complex math module
a = float(input('Enter a: '))                 # To take coefficient from the users
b = float(input('Enter b: '))
c = float(input('Enter c: '))                 # calculate the discriminant
d = (b**2) - (4*a*c)
sol1 = (-b-cmath.sqrt(d))/(2*a)               # find two solutions
sol2 = (-b+cmath.sqrt(d))/(2*a)
print("The solution are {0} and {1}".format(sol1,sol2)) # displays the quadratic equation
```

Output:

```
>>> Enter a: 1
    Enter b: 5
    Enter c: 6
    The solution are (-3+0j) and (-2+0j)
```

3. Write a Python program to swap two numbers using a temporary variable.

```
x = input('Enter value of x: ')               # To take input from the user
y = input('Enter value of y: ')
temp = x                                      # create a temporary variable
x = y
y = temp
print('The value of x after swapping: {}'.format(x))
print('The value of y after swapping: {}'.format(y))
```

Output:

```
>>> Enter value of x: 10
     Enter value of y: 20
    The value of x after swapping: 20
    The value of y after swapping: 10
```

4. Write a Python Program to find the distance between two points

```
import math
def distance(x1,y1,x2,y2):                          # Defining the Function Distance
        dx=x2-x1
        dy=y2-y1
        print("The value of dx is", dx)
        print("The value of dy is", dy)
        d= (dx**2 + dy**2)
        dist=math.sqrt(d)
        return dist
x1 = float(input("Enter the first Number: "))       #Getting inputs from the user
x2 = float(input("Enter the Second Number: "))
y1 = float(input("Enter the third number: "))
y2 = float(input("Enter the forth number: "))
print("The distance between two points are",distance(x1,x2,y1,y2))
                                                    #Calling the function distance
```

Output:

```
>>> Enter the first Number: 2
    Enter the Second Number: 4
    Enter the third number: 6
    Enter the forth number: 12
    The value of dx is 4.0
    The value of dy is 8.0
   The distance between two points are 8.94427190999916
```

6. Write a program to circulate the values of list.

```
def  circulate(list,n):
        return list[n:]+list[:n]
>>>print("The Values Shift two places in Clockwise: "circulate([1,2,3,4,5,6,7],2))
>>>print("The Values Shift three places in Anti Clockwise: "circulate([1,2,3,4,5,6,7],-3))
```

Output:

```
The Values Shift two places in Clockwise: [3,4,5,6,7,1,2,]
The Values Shift three places in Anti Clockwise: [5, 6, 7, 1, 2, 3, 4]
```

WORKSHEETS

Note: - Always Get an Input from the User

2.1. Write a python program to find the area of a circle

Program:

Output:

2.2 Write a python program to convert Kilometers to Miles (Conversion factor = 0.621)

Formula: miles=kilometer*conversion_factor

Program:

Output:

2.3. Write a python program to find the average of 5 numbers

Program:

Output:

2.4. Write a python program to find Simple Interest

Formula: Simple Interest = (P x T x R)/100

Program:

Output:

2.5. Write a python program to find Compound Interest

Formula: Compound Interest = P(1+ R/100)$^{\text{Time}}$

Program:

Output:

2.6. Write a python program to o display your details like name, age, address in three different lines.

Program:

Output:

TWO MARKS

1. What is a value? What are the different types of values? (University question)

A value is one of the fundamental things – like a letter or a number – that a program manipulates. Its types are: integer, float, , strings , lists and Dicitionary.

2. Define a variable and write down the rules for naming a variable.

A name that refers to a value is a variable. Variable names can be arbitrarily long. They can contain both letters and numbers, but they have to begin with a letter. It is legal to use uppercase letters, but it is good to begin variable names with a lowercase letter.

3. Define keyword and enumerate some of the keywords in Python. (University question)

A keyword is a reserved word that is used by the compiler to parse a program. Keywords cannot be used as variable names. Some of the keywords used in python are: and, del, from, not, while, is, continue.

4. Define statement and what are its types?

A statement is an instruction that the Python interpreter can execute. There are two types of statements: print and assignment statement.

5. What do you meant by an assignment statement?

An assignment statement creates new variables and gives them values:

Eg 1: Message = “hello”

Eg 2: n = 17

6. What is tuple? (*University question*)

A tuple is a sequence of immutable Python objects. Tuples are sequences, like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets. Creating a tuple is as simple as putting different comma-separated values. Comma-separated values between parentheses can also be used.

Example: tup1 = ('physics', 'chemistry', 1997, 2000);

7. What is an expression?

An expression is a combination of values, variables, and operators. An expression is evaluated using assignment operator.

Examples: Y=x + 17

8. What do you mean by an operand and an operator? Illustrate your answer with relevant example.

An operator is a symbol that specifies an operation to be performed on the operands. The data items that an operator acts upon are called operands. The operators +, -, *, / and ** perform addition, subtraction, multiplication, division and exponentiation.

Example: 20+32

In this example, 20 and 32 are operands and + is an operator.

9. What is the order in which operations are evaluated? Give the order of precedence.

The set of rules that govern the order in which expressions involving multiple operators and operands are evaluated is known as rule of precedence. Parentheses have the highest precedence followed by exponentiation. Multiplication and division have the next highest precedence followed by addition and subtraction.

10. Illustrate the use of * and + operators in string with example.

The * operator performs repetition on strings and the + operator performs concatenation on strings.

Example:

```
>>> ‘Hello*3’
Output:  HelloHelloHello
>>>’Hello+World’
Output: HelloWorld
```

11. What is the symbol for comment? Give an example.

# is the symbol for comments in python.

Example:

```
# compute the percentage of the hour that has elapsed
```

12. What is function call?

A function is a named sequence of statements that performs a computation. When we define a function, we specify the name and the sequence of statements. Later, we can “call” the function by its name called as function call.

13. Identify the parts of a function in the given example.

```
>>>     betty = type("32")
>>>     print betty
```

The name of the function is type, and it displays the type of a value or variable. The value or variable, which is called the argument of the function, is enclosed in parentheses. The argument is 32. The function returns the result called return value. The return value is stored in betty.

14. What is a local variable?

A variable defined inside a function. A local variable can only be used inside its function.

15. What is the output of the following?

a. float(32)

b. float("3.14159")

Output:

a. 32.0 The float function converts integers to floating-point numbers.

b. 3.14159 The float function converts strings to floating-point numbers.

16. What do you mean by flow of execution?

In order to ensure that a function is defined before its first use, we have to know the order in which statements are executed, which is called the flow of execution. Execution always begins at the first statement of the program. Statements are executed one at a time, in order from top to bottom.

17. Write down the output for the following program.

```
first = 'throat'
second = 'warbler' print first + second
```

Output:

```
throatwarbler
```

18. Give the syntax of function definition.

```
def NAME( LIST OF PARAMETERS ):
        STATEMENTS
```

19. Explain the concept of floor division.

The operation that divides two numbers and chops off the fraction part is known as floor division.

20. What is type coercion? Give example.

Automatic method to convert between data types is called type coercion. For mathematical operators, if any one operand is a float, the other is automatically converted to float.

Eg:

```
>>>     minute = 59
>>>     minute / 60.0
0.983333333333
```

21. Write a math function to perform √2 / 2.

```
>>>     math.sqrt(2) / 2.0
0.70710678118
```

22. What is meant by traceback?

A list of the functions that tells us what program file the error occurred in, and what line, and what functions were executing at the time. It also shows the line of code that caused the error.

23. Write a program to accept two numbers multiply them and print them. (*University question*)

```
A=10
B=2
multiplication=A*B
print (“Multiplication of two numbers :” ,multiplication)
```

UNIT III CONTROL FLOW, FUNCTIONS, STRINGS

Conditionals: Boolean values and operators, conditional (if), alternative (if-else), chained conditional (if-elif-else); Iteration: state, while, for, break, continue, pass; Fruitful functions: return values, parameters, local and global scope, function composition, recursion; Strings: string slices, immutability, string functions and methods, string module; Lists as arrays. Illustrative programs: square root, gcd, exponentiation, sum an array of numbers, linear search, binary search.

BOOLEAN VALUES:

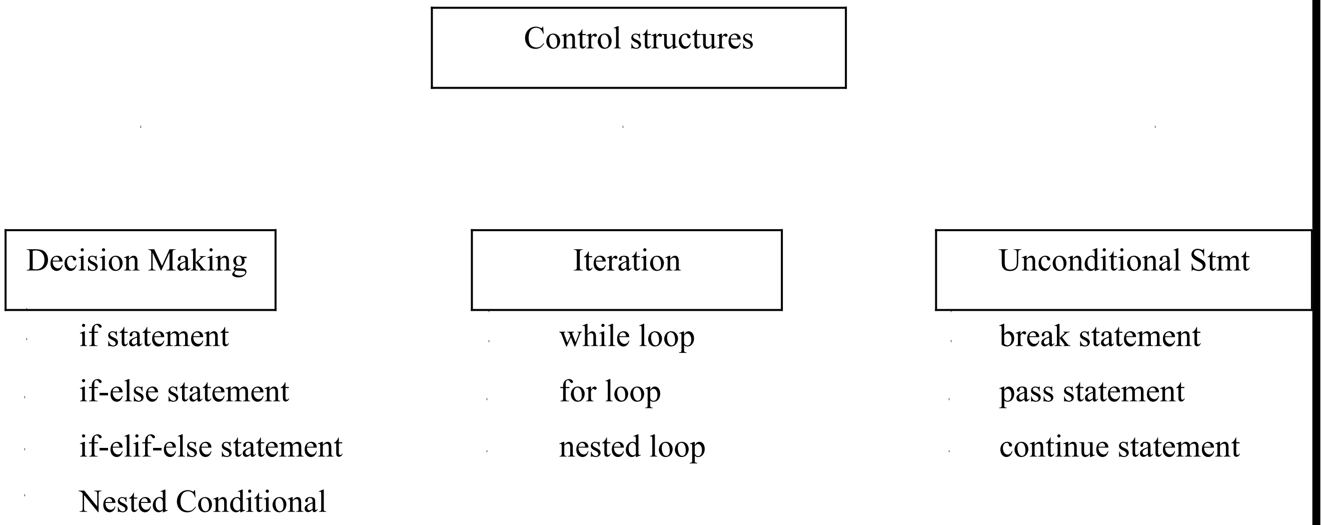
Boolean values can be tested for truth value, and used for IF and WHILE condition. There are two values True and False. 0 is considered as False and all other values considered as True.

Boolean Operations:

Consider x=True, y= False

Operator	Example	Description
and	x and y- returns false	Both operand should be true
or	x or y- returns true	Anyone of the operand should be true
not	not x returns false	Not carries single operand

3.1 CONTROL STRUCTURES



3.1.1 Decision Making (or) Conditionals (or) Branching

The execution of the program depends upon the condition. The sequence of the control flow differs from the normal program. The decision making statements evaluate the conditions and produce True or False as outcome.

Types of conditional Statement

1. if statement
2. if-else statement
3. if-elif-else statement
4. Nested Conditional

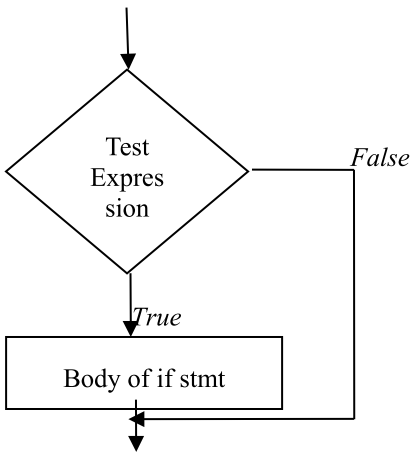
1.if statement

If statement contains a logical expression using which data is compared and a decision is made based on the result of comparison.

Syntax

```
if expression:
        true statements
```

Flow Chart



Example:

```
a=10
if a==10:
        print("a is equal to 10")
```

Output:

```
a is equal to 10
```

2. if else statement

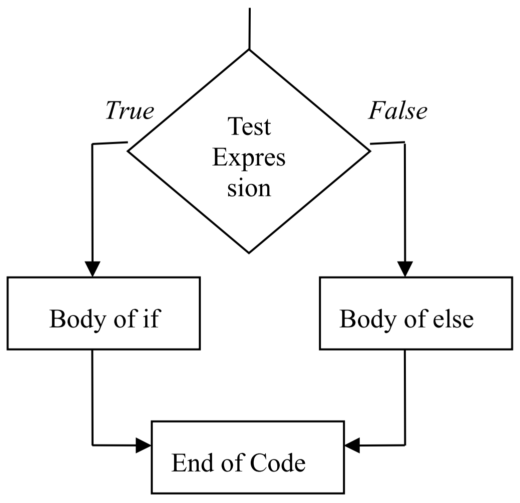
The second form of if statement is "alternative execution" in which there are two possibilities and the condition determines which block of statement executes.

Syntax

```
if  test_expression:
        true statements
else:
        false statements
```

If the testexpression evaluates to true then truestatements are executed else falsestatements are executed.

Flow Chart



Example:

```
a=10
if a==10:
        print("a is equal to 10")
else:
        print("a is not equal to 10")
```

Output:

```
a  is equal to 10
```

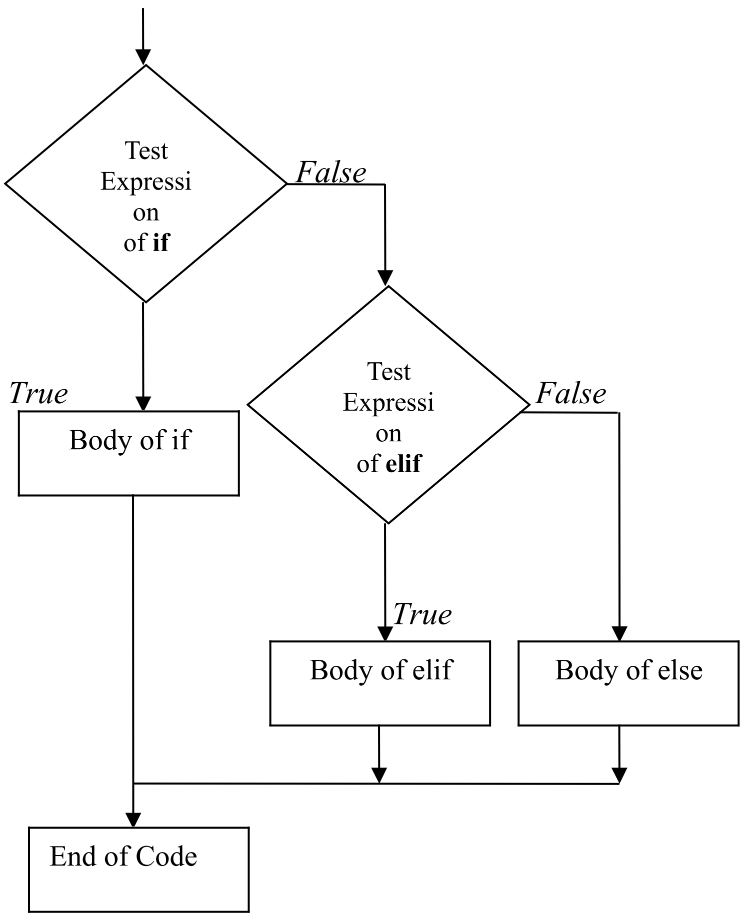
3.elif else Statement(chained conditionals)

The elif statement or chained conditional allows you to check multiple expressions for true and execute a block of code as soon as one of the conditions evaluates to true. The elif statement has more than one statements and there is only one if condition.

Syntax

```
if expression 1:
        true statements
elif expression 2:
        true statements
elif expression 3:
        true statements
else:
        false statement
```

Flow Chart



Example:

```
a=9
if a==10:
          print("a is equal to 10")
elif a<10:
          print("a is lesser than 10")
elif a>10:
          print("a is greater than 10")
 else
          print("a is not equal to 10")
```

Output::

```
a is less than 10
```

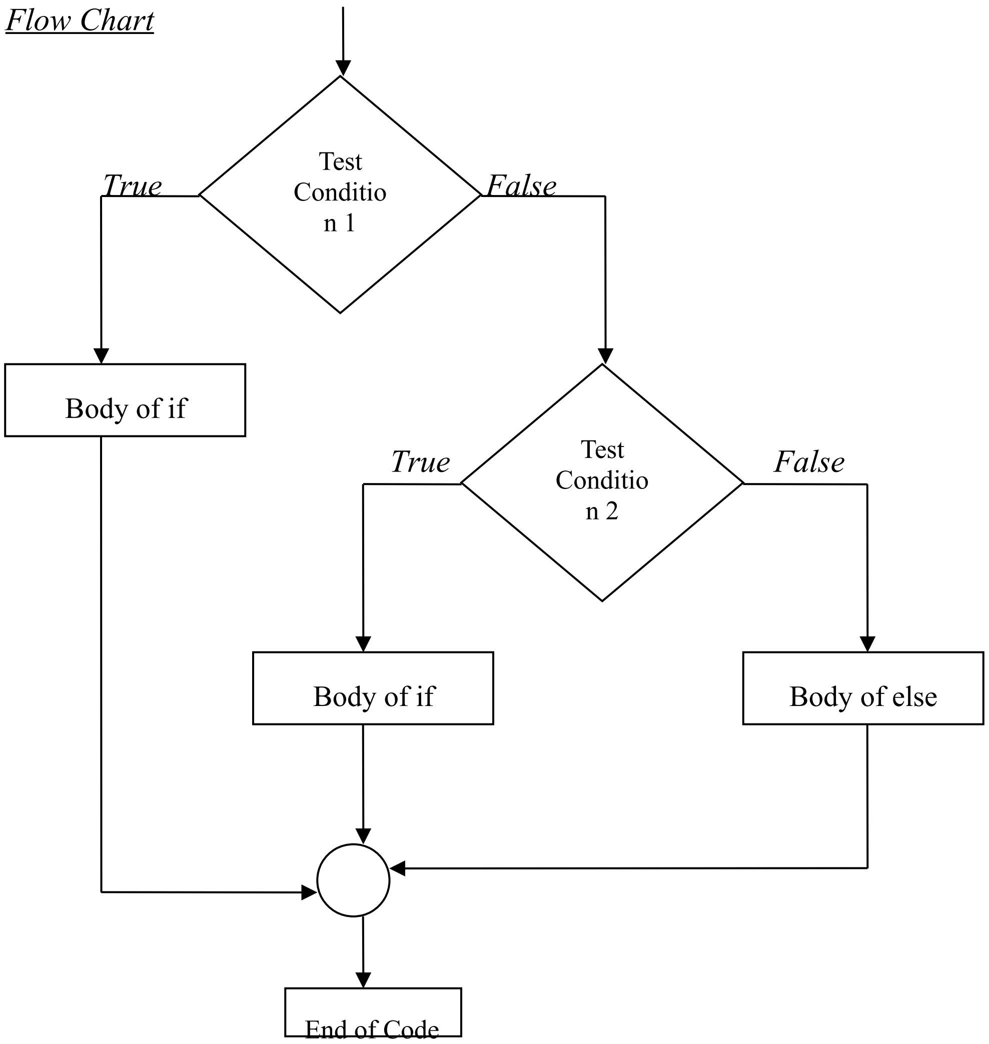
<u>4. Nested Conditionals (or) Nested if-else</u>

One conditional can also be nested with another condition.(ie) we can have if…elif….else statement inside another if …elif…else statements.

<u>Syntax</u>

```
if expression 1:
          true statements
else:
          if expression 2:
          true statements
          else:
          false statement
```

<u>Flow Chart</u>



Example:

```
a=10
if a==100:
        print("a is equal to 10")
else:
        if a>100:
                print("a is greater than 100")
        else
                print("a is lesser than 100")
```

Output:

```
a is lesser than 100
```

3.1.2 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

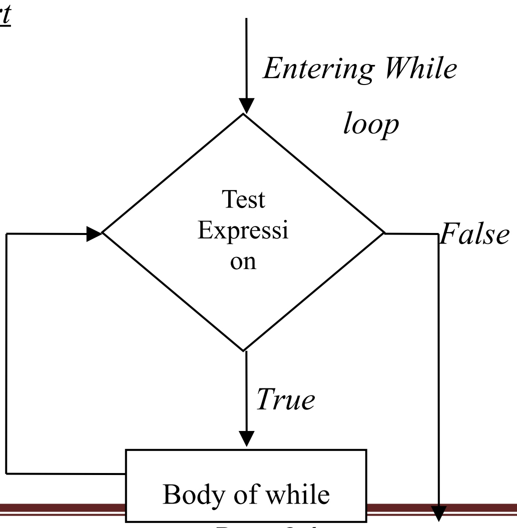
1. while loop

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



Exit Loop

Example-1:

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

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

Output:

```
        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)
```

Output:

```
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.

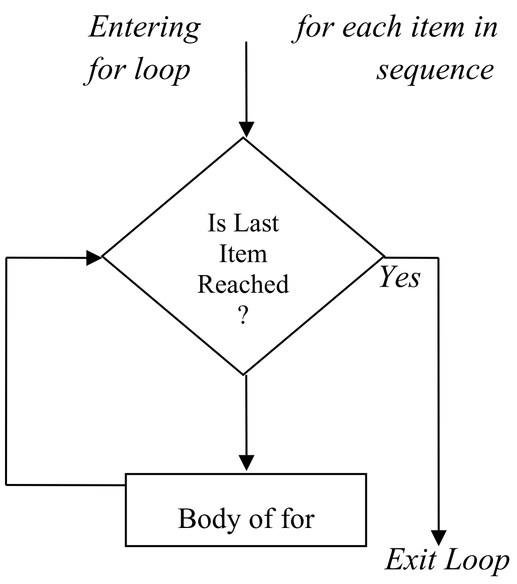
For loop in python starts with the keyword *"for"*followed by an arbitrary variable name,which holds its value in the following sequence objects.

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
```

3.1.3.Unconditional Statement

In a situation the code 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

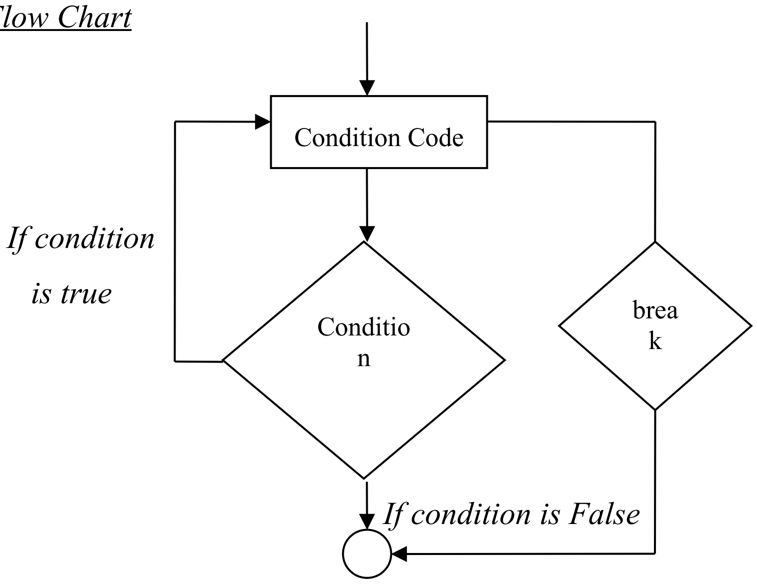
1.break statement

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
```

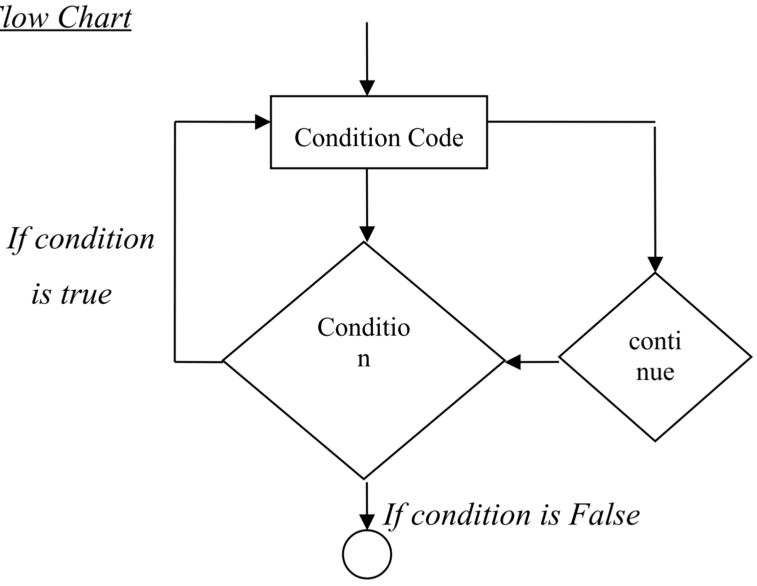
2. continue statement

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.

Syntax

```
pass
```

Example:

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

Output:

```
        python
        bye
```

3.2 Fruitful Functions:

Function that returns value are called as fruitful functions. The return statement is followed by an expression which is evaluated, its result is returned to the caller as the "fruit" of calling this function.

$$\boxed{\text{Input the value} \rightarrow \text{fruitful function} \rightarrow \text{return the result}}$$

len(variable) – which takes input as a string or a list and produce the length of string or a list as an output.

range(start, stop, step) – which takes an integer as an input and return a list containing all the numbers as the output.

Example-1:

Write a python program to find distance between two points:

```
        import math
        def distance(x1,y1,x2,y2):                    # Defining the Function Distance
                dx=x2-x1
                dy=y2-y1
                print("The value of dx is", dx)
                print("The value of dy is", dy)
                d= (dx**2 + dy**2)
                dist=math.sqrt(d)
                return dist
        x1 = float(input("Enter the first Number: "))        #Getting inputs from  user
        x2 = float(input("Enter the Second Number: "))
        y1 = float(input("Enter the third number: "))
        y2 = float(input("Enter the forth number: "))
        print("The distance between two points are",distance(x1,x2,y1,y2))
```

#Calling the function distance

Output:

```
>>> Enter the first Number: 2
    Enter the Second Number: 4
    Enter the third number: 6
    Enter the forth number: 12
    The value of dx is 4.0
    The value of dy is 8.0
   The distance between two points are 8.94427190999916
>>>
```

Explanation for Example 1:

Function Name – 'distance()'
Function Definition – def distance(x1, y1, x2, y2)
Formal Parameters - x1, y1, x2, y2
Actual Parameter – dx, dy
Return Keyword – return the output value 'dist'
Function Calling – distance(x1, y1, x2, y2)

Parameter in fruitful function

A function in python

- Take input data, called parameter
- Perform computation
- Return result

```
def funct(param1,param2):
        statements
        return value
```

-

Once the function is defined,it can be called from main program or from another function.

Function call statement syntax

```
Result=function_name(param1,param2)
```

Parameter is the input data that is sent from one function to another. The parameters are of two types

1. Formal parameter

- The parameter defined as part of the function definition.
- The actual parameter is received by the formal parameter.

2. Actual parameter

- The parameter is defined in the function call

Example-1:

```
def  cube(x):
        return x*x*x                              #x is the formal parameter
a=input("Enter the number=")
b=cube(a)                                         #a is the actual parameter
print"cube of given number=",b
```

Output:

```
Enter the number=2
Cube of given number=8
```

3.3 Composition:

A Composition is calling one function from another function definition.

Example:

Write a python program to add three numbers by using function:

```
def addition(x,y,z):                                      #function 1
        add=x+y+z
        return add
def get():                                                #function 2
        a=int(input("Enter first number:"))
        b=int(input("Enter second number:"))
        c=int(input("Enter third number:"))
        print("The addition is:",addition(a,b,c))         #Composition function calling
get()                                                     #function calling
```

Output:

```
Enter first number:5
Enter second number:10
Enter third number:15
The addition is: 30
```

3.4 SCOPE :GLOBAL AND LOCAL

Scope is the portion of the program from where a namespace can be accessed directly without any prefix. When a reference is made inside a function, the name is searched in the local namespace, then in the global namespace and finally in the built-in namespace.

Local scope Example:

```
 def   outer_function():
          a=”I am in india”
          def  inner_function():
                    a=” I am in TamilNadu”
                    print(“a=”a)
          inner_function()
          print(‘a=”,a)

>>>a=”I am in world”
>>>outer_function()
>>>print(‘a=’,a)
```

Output:

```
a=I am in TamilNadu
a=I am in India
a=I am in World
```

Global scope Example:

```
 def   outer_function():
          global a
          a=”I am in india”
          def  inner_function():
                    a=” I am in TamilNadu”
                    print(“a=”a)
          iner_function()
          print(‘a=”,a)

>>>a=”I am in world”
>>>outer_function()
>>>print(‘a=’,a)
```

Output:

```
a=I am in TamilNadu
a=I am in TamilNadu
a=I am in TamilNadu
```

3.5 Recursion:

A Recursive function is the one which calls itself again and again to repeat the code. The recursive function does not check any condition. It executes like normal function definition and the particular function is called again and again

Syntax:

```
def function(parameter):
          #Body of function
```

Example-1:

Write a python program to find factorial of a number using Recursion:

(*Positive value of n , then n! can be calculated as n!=(n-1)….2.1 it can be written as (n-1)! Hence n! is the product of n and (n-1)!* n!=n.(n-1)!)

```
def fact(n):
        if(n<=1):
                return n
        else:
                return n*fact(n-1)
n=int(input("Enter a number:"))
print("The Factorial is", fact(n))
```

Output:

```
>>> Enter a number:5
     The Factorial is 120
>>>
```

Explanation:

First Iteration - 5*fact(4)
Second Iteration - 5*4* fact(3)
Third Iteration - 5*4*3*fact(2)
Fourth Iteration - 5*4*3*2* fact(1)
Fifth Iteration - 5*4*3*2*1

Example-2:

Write a python program to find the sum of a 'n' natural number using Recursion:

```
def nat(n):
        if(n<=1):
                return n
          else:
                return n+nat(n-1)
>>>n=int(input("Enter a number:"))
>>>print("The Sum is", nat(n))
```

Output:

```
>>> Enter a number: 5
     The Sum is 15
>>>
```

Explanation:

First Iteration – 5+nat(4)
Second Iteration – 5+4+nat(3)
Third Iteration – 5+4+3+nat(2)
Fourth Iteration – 5+4+3+2+nat(1)
Fifth Iteration – 5+4+3+2+1

3.6 String:

A string is a combination of characters. A string can be created by enclosing the characters within single quotes or double quotes. The character can be accessed with the bracket operator.

Example:

```
var1='Hello'
var2="Python Programming"
print(var1)                        # Prints Hello
print(var2)                        # Prints Python Programming
```

Traversal with a for Loop

A lot of computations involve in processing a string, one character at a time. Often they start at the beginning, select each character in turn, do something to it, and continue until the end. This pattern of processing is called a traversal.

Example:

```
>>>index = 0
>>>str="Python"
>>>while index < len(str):
        letter = str[index]
        print(letter, end="")
        index = index+1
```

Output:

```
Python
```

3.6.1 String Slices:

A segment of a string is called a slice. Selecting a slice is similar to selecting a character.

Syntax:

```
variable [start:stop]
<String_name> [start:stop]
```

Example

Char	a=	"B	A	N	A	N	A"
Index from Left		0	1	2	3	4	5
Index from Right		-6	-5	-4	-3	-2	-1

```
>>>print(a[0]) → prints B             #Prints B Alone 0th Position
>>>print(a[5]) → prints A             #Prints A Alone Last Position
>>>print(a[-4]) → print N             #Print From Backwards -4th Position
>>>a[:] → 'BANANA'                    #Prints All
>>>print(a[1:4]) → print ANA          #Print from 1st Position to 4th Position
>>> print(a[1:-2]) → ANA              #Prints from 1st position to -3th Position
```

3.6.2 Strings are Immutable:

Strings are immutable which mean we cannot change the exiting string.

>>>var1="Hello"

>>>var1[0]="J" *#Displays Error*

3.6.3 Strings Methods (or) Types of functions in String:

Consider the value of a, b, c be

a="Hello", b="hELLO", c="1234+56"

S.No	Method	Syntax	Description	Example
1.	len()	len(String)	Returns the length of the String	len(a) →5
2.	center()	String.center(width,fill char)	The String will be centered along with the width specified and the characters will fill the space	a.center(20,+) → ++++Hello++++
3.	lower()	String.lower()	Converts all upper case into lower case	b.lower() → hello
4.	upper()	String.upper()	Converts all lower case into upper case	a.upper() → HELLO
5.	capitalize()	String.capitalize()	It converts the first letter into capital	b.capitalize() → Hello
6.	split()	String.split("Char")	splits according to the character which is present inside the function	c.split("+") → ['1234','56']
7.	join()	String1.join(String2)	It concatenates the string with the sequence	a.join(b) → Hello hELLO
8.	isalnum()	String.isalnum()	It checks the string is alpha numeric or not. If the string contains 1 or more alphanumeric characters it returns 1, else its returns 0	c.isalnum() → returns 1
9.	isalpha()	String.isalpha()	Returns true if it has at least 1 or more alphabet characters, else it return false	b.isalpha() →returns 1
10.	isdigit()	String.isdigit()	Returns true if it has at least 1 or more digits, else it return false	b.isdigit() →returns 0
11.	islower()	String.islower()	Returns true if the string has all Lower case characters, else it return false	b.islower() → returns 0
12.	isupper()	String.isupper()	Returns true if the string has all Upper case characters, else it return false	b.isupper()→ returns 0

13.	isnumeric()	String.isnumeric()	Returns true if the string contains only numeric character or false otherwise	a.isnumeric()→ return 0
14.	isspace()	String.isspace()	Returns true if the string contains only wide space character or false otherwise	a.isspace() → returns e=" " e.isspace() → returns
15.	istitle()	String.istitle()	Returns true if the string is properly titled or false otherwise	d="Hello How R U" d.istitle() →returns 1
16.	isdecimal()	String.isdecimal()	Returns true if the string contains decimal value or false otherwise	c.isdecimal() → returns 1
17.	title()	String.title()	Returns title cased, all the characters begin with upper case	d="hello how h u" d.title() → "Hello How R U"
18.	find()	String.find("text", start,end)	If the string is found it returns index position or it returns -1	a.find("e",0,4) → returns 1 (index position)
19.	endswith()	String.endswith("text" , beg, end)	The string will check for whether the character is ending with the specified character. If it found it returns true, else false	a.endswith(i,0) → returns false
20.	index()	String.index('text',beg , end)	It is same as find(). but it raises exception when the string is not found	a.index('l',0) → returns 2
21.	count()	String.count('text',beg , end)	It counts howmany times a string appears	a.count('l',0) → returns 2
22.	rfind()	String.rfind('text',beg, end)	It finds a string from right to left	a.rfind('i') →-1
23.	rindex()	String.rindex('text', beg, end)	Same as index() but moves from right to left	a.rindex('l',0) → returns 3
24.	rjust()	String.rjust(width, fillchar)	It will justify the character into right and fill with the character	a.rjust(10,'-')→ -----Hello
25.	ljust()	String.ljust(width, fillchar)	It will justify the character into left and fill with the character	a.ljust(10,a,'+')→ Hello+++++
26.	rstrip()	rstrip()	It removes all the spaces at the end	rstrip(a) → it returns -
27.	startswith()	startswith(text, beg, end)	It checks whether the character starts with the specified one	a.stratswith(H,0) → returns true

3.6.4 Strings Modules:

This module contains a number of functions to process standard python strings. Using import string' we can invoke string functions.

Example: Using the string module

```
import string
text = "Monty Python's Flying Circus"
print("upper", "=>", string.upper(text))
print("lower", "=>", string.lower(text))
print("split", "=>", string.split(text))
print("join", "=>", string.join(string.split(text), "+"))
print("replace", "=>", string.replace(text, "Python", "Java"))
print("find", "=>", string.find(text, "Python"), string.find(text, "Java"))
print("count", "=>", string.count(text, "n"))
```

Output:

```
upper => MONTY PYTHON'S FLYING CIRCUS
lower => monty python's flying circus
split => ['Monty', "Python's", 'Flying', 'Circus']
join => Monty+Python's+Flying+Circus
replace => Monty Java's Flying Circus
find => 6 -1
count => 3
```

The 'in' Operator

The word in is a boolean operator that takes two strings and returns True if the first appears as a substring in the second:

```
>>>'t' in 'python'
True
>>> 'jan' in 'python'
False
```

3.9 List as Array

To store such data, in Python uses the data structure called list (in most programming languages the different term is used — "array").

Arrays are sequence types and like lists, except that the type of objects stored in them is constrained. ***A list (array) is a set of objects.*** Individual objects can be accessed using ordered indexes that represent the position of each object within the list (array).

The list can be set manually by enumerating of the elements the list in square brackets, like here:

Primes = [2, 3, 5, 7, 11, 13]

Rainbow = ['Red', 'Orange', 'Yellow', 'Green', 'Blue', 'Indigo', 'Violet']

The list Primes has 6 elements, namely:

Primes[0] = 2, Primes[1] = 3, Primes[2] = 5, Primes[3] = 7,Primes[4] = 11, Primes[5] = 13.

<u>ILLUSTRATIVE EXAMPLES</u>

Note- *Always get an Input from the USER.*

1. Program to find the square root using Newton Method.

2. Program to find the GCD of two numbers

3. Program to find the exponential of a number

4. Program to find the sum of n numbers.

5. Program to find the maximum and minimum in a list

6. Program to perform the linear search

7. Program to perform Binary search

1. Write a python Program to Check if a Number is Positive, Negative or 0

<u>Using if...elif...else</u>

```
num = float(input("Enter a number: "))
if num > 0:
    print("Positive number")
elif num == 0:
     print("Zero")
else:
    print("Negative number")
```

<u>Output:</u>

```
>>> Enter a number: 5
    Positive number
```

<u>3.2 Using Nested if</u>

```
num = float(input("Enter a number: "))
if num >= 0:
    if num == 0:
        print("Zero")
     else:
         print("Positive number")
else:
    print("Negative number")
```

<u>Output:</u>

```
>>> Enter a number: 5
    Positive number
```

3.3 Write a Python Program to Check a year is Leap Year or not.

```
year = int(input("Enter a year: "))              # To get year (integer input) from the user
if (year % 4) == 0:
        if (year % 100) == 0:
                if (year % 400) == 0:
                        print("{0} is a leap year".format(year))
                else:
                        print("{0} is not a leap year".format(year))
        else:
                print("{0} is a leap year".format(year))
else:
        print("{0} is not a leap year".format(year))
```

Output:

```
>>>
Enter a year: 2000
2000 is a leap year
>>>
Enter a year: 1991
1991 is not a leap year
```

3.4 Write a Python Program to Print the Fibonacci sequence

```
nterms = int(input("How many terms? "))
                                                        # first two terms
n1 = 0
n2 = 1
count = 0
                                                        # check if the number of terms is valid
if nterms <= 0:
        print("Please enter a positive integer")
elif nterms == 1:
        print("Fibonacci sequence upto",nterms,":")
        print(n1)
else:
        print("Fibonacci sequence upto",nterms,":")
        while count < nterms:                           # Starting of While loop
                print(n1,end=' , ')
                nth = n1 + n2
                                                        # update values
                n1 = n2
                n2 = nth
                count += 1                              # Ending of While loop
```

Output:

```
How many terms? 10
Fibonacci sequence upto 10 :
0 , 1 , 1 , 2 , 3 , 5 , 8 , 13 , 21 , 34 ,
>>>
```

3.5 Write a Python Program to Check a number is Armstrong Number or not.

```
num = int(input("Enter a number: "))
sum = 0                                   # initialize sum
temp = num                                # find the sum and cube of each digit
while temp > 0:
        digit = temp % 10
        sum += digit ** 3
        temp //= 10
if num == sum:                            # display the result
        print(num,"is an Armstrong number")
else:
        print(num,"is not an Armstrong number")
```

Output:

```
>>> Enter a number: 121
    121 is not an Armstrong number
>>>
```

3.6 Write a Python Program to Find LCM of two numbers

```
        def lcm(x, y):
                if x > y:
                        greater = x
                else:
                        greater = y
                while(True):
                        if((greater % x == 0) and (greater % y == 0)):
                                lcm = greater
                                break
                        greater += 1
                return lcm
num1 = int(input("Enter first number: "))
num2 = int(input("Enter second number: "))
print("The L.C.M. of", num1,"and", num2,"is", lcm(num1, num2))
```

Output:

```
>>> Enter first number: 10
    Enter second number: 15
    The L.C.M. of 10 and 15 is 30
>>>
```

3.7 Write a Python Program to Add Two Matrices

```
X = [[12,7,3],
     [4 ,5,6],
     [7 ,8,9]]
Y = [[5,8,1],
     [6,7,3],
     [4,5,9]]
result = [[0,0,0],
          [0,0,0],
          [0,0,0]]
```

```
                                                    # iterate through rows
     for i in range(len(X)):
                                                    # iterate through columns
          for j in range(len(X[0])):
               result[i][j] = X[i][j] + Y[i][j]
     for r in result:
          print(r)
```

Output:

```
>>>
[17, 15, 4]
[10, 12, 9]
[11, 13, 18]
>>>
```

3.8 Write a Python Program to Transpose a Matrix

```
     X = [[12,7],
         [4 ,5],
         [3 ,8]]
     result = [[0,0,0],
              [0,0,0]]                              # iterate through rows
     for i in range(len(X)):
                                                    # iterate through columns
          for j in range(len(X[0])):
               result[j][i] = X[i][j]

     for r in result:
          print(r)
```

Output:

```
>>>
[12, 4, 3]
[7, 5, 8]
>>>
```

3.9 Python Program to Multiply Two Matrices

```
                                                    # 3x3 matrix
     X = [[12,7,3],
         [4 ,5,6],
         [7 ,8,9]]
                                                    # 3x4 matrix
     Y = [[5,8,1,2],
         [6,7,3,0],
         [4,5,9,1]]
                                                    # result is 3x4
     result = [[0,0,0,0],
               [0,0,0,0],
               [0,0,0,0]]
                                                    # iterate through rows of X
```

```
for i in range(len(X)):
                                                        # iterate through column Y
        for j in range(len(Y[0])):
                                                        # iterate through rows of Y
                for k in range(len(Y)):
                        result[i][j] += X[i][k] * Y[k][j]
for r in result:
        print(r)
```

Output:

```
>>> [114, 160, 60, 27]
    [74, 97, 73, 14]
    [119, 157, 112, 23]
```

3.10 Write a Python Program to Check Whether a String is Palindrome or Not

```
my_str = 'madame'
my_str = my_str.casefold()                  # it suitable for case less comparison
rev_str = reversed(my_str)                  # reverse the string
if list(my_str) == list(rev_str):           # check the string is equal to its reverse
        print("It is palindrome")
else:
        print("It is not palindrome")
```

Output:

```
>>>
 It is not palindrome
```

3.11 Write a Python Program to count the number of each vowel in a string.

```
vowels = 'aeiou'                                    # string of vowels
ip_str = 'Hello, have you tried our turorial section yet?'   # change this value
ip_str = input("Enter a string: ")
ip_str = ip_str.casefold()                    # make it suitable for caseless comparisions
count = {}.fromkeys(vowels,0)   # make a dictionary with each vowel a key and value 0
for char in ip_str:                # count the vowels
    if char in count:
        count[char] += 1
print(count)
```

Output:

```
>>>{'o': 5, 'i': 3, 'a': 2, 'e': 5, 'u': 3}
```

WORKSHEETS

NOTE: Always get an Input from the User

3.1. Write a python program for Sum of squares of first n natural numbers

$1^2+2^2+3^2+\ldots\ldots\ldots+n^2$

Program:

Output:

3.2. Write a python program to find two numbers which are divisible by 13

Program:

Output:

3.3. Write a python program to find Factors of Number

25 = 1, 5, 25

Program:

Output:

3.4. Write a python program to print even length words in a string

str=”This is a python program” →This is python

Program:

Output:

3.5. Write a python program to accept the strings which contains all vowels

Input : ABeeIghiObhkUul → Output : Accepted

Program:

Output:

3.6. Write a python program using slicing rotate a string

Input : s = "qwertyu" , d = 2

Output : Left rotation : "ertyuqw" , Right rotation : "yuqwert"

Program:

Output:

3.7. Write a python program Find words which are greater than given length k

Input : str = "string is easy in python" ,   k = 3

Output : string easy python

Program:

Output:

3.8. Write a python program to split and join a string

Input : ['Python', 'is', 'Easy']

Output : Python-is-Easy

Program:

Output:

TWO MARKS

1. Define Boolean expression with example.

A boolean expression is an expression that is either true or false. The values true and false are called Boolean values.

Eg :

```
>>>     5 == 6
>>>     False
```

True and False are special values that belongs to the type bool; they are not strings.

2. What are the different types of operators?

- Arithmetic Operator (+, -, *, /, %, **, //)
- Relational operator (== , !=, <>, < , > , <=, >=)
- Assignment Operator (=, += , *= , - =, /=, %= ,**=)
- Logical Operator (AND, OR, NOT)
- Membership Operator (in, not in)
- Bitwise Operator (& (and), | (or) , ^ (binary Xor), ~(binary 1's complement , << (binary left shift), >> (binary right shift))
- Identity(is, is not)

3. Explain modulus operator with example.

The modulus operator works on integers and yields the remainder when the first operand is divided by the second. In Python, the modulus operator is a percent sign (%). The syntax is the same as for other operators:

Eg:

```
>>>     remainder = 7 % 3
>>>     print remainder 1
```

So 7 divided by 3 is 2 with 1 left over.

4. Explain relational operators.

The == operator is one of the relational operators; the others are:

```
x! = y # x is not equal to y
x > y # x is greater than y
x < y # x is less than y
x >= y # x is greater than or equal to y
```

x <= y # x is less than or equal to y

5. Explain Logical operators(University question)

There are three logical operators: and, or, and not. For example, x > 0 and x < 10 is true only if x is greater than 0 and less than 10. n%2 == 0 or n%3 == 0 is true if either of the conditions is true, that is, if the number is divisible by 2 or 3. Finally, the not operator negates a Boolean expression, so not(x > y) is true if x > y is false, that is, if x is less than or equal to y. Non-zero number is said to be true in Boolean expressions.

6. What is conditional execution?

The ability to check the condition and change the behavior of the program accordingly is called conditional execution. Example:

If statement:

The simplest form of if statement is:

Syntax:

if statement:

Eg:

```
if x > 0:
        print 'x is positive'
```

The boolean expression after 'if' is called the condition. If it is true, then the indented statement gets executed. If not, nothing happens.

7. What is alternative execution?

A second form of if statement is alternative execution, that is, if …else, where there are two possibilities and the condition determines which one to execute.

Eg:

```
if x%2 == 0:
        print 'x is even'
else:
        print 'x is odd'
```

8. What are chained conditionals?

Sometimes there are more than two possibilities and we need more than two branches. One way to express a computation like that is a chained conditional:

Eg:

```
if x < y:
        print 'x is less than y'
elif x > y:
        print 'x is greater than y'
else:
        print 'x and y are equal'
```

elif is an abbreviation of "else if." Again, exactly one branch will be executed. There is no limit on the number of elif statements. If there is an else clause, it has to be at the end, but there doesn't have to be one.

9. Explain while loop with example. Eg:

```
def countdown(n):
        while n > 0:
                print n
                n = n-1
        print 'Blastoff!'
```

More formally, here is the flow of execution for a while statement:

1. Evaluate the condition, yielding True or False.
2. If the condition is false, exit the while statement and continue execution at the next statement.
3. If the condition is true, execute the body and then go back to step 1

10. Explain 'for loop' with example.

The general form of a for statement is

Syntax:

```
for variable in sequence:
        code block
```

Eg:

```
x = 4
for i in range(0, x):
        print i
```

11. What is a break statement?

When a break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.

Eg:

```
while True:
        line = raw_input('>')
        if line == 'done':
                break
print line
print'Done!'
```

12. What is a continue statement?

The continue statement works somewhat like a break statement. Instead of forcing termination, it forces the next iteration of the loop to take place, skipping any code in between.

Eg:

```
for num in range(2,10):
        if num%2==0;
                print "Found an even number", num
                continue
print "Found a number", num
```

13.Compare return value and composition.

Return Value:

Return gives back or replies to the caller of the function. The return statement causes our function to exit and hand over back a value to its caller.

Eg:

```
def area(radius):
        temp = math.pi * radius**2 return temp
```

Composition:

Calling one function from another is called composition.

Eg:

```
def circle_area(xc, yc, xp, yp):
        radius = distance(xc, yc, xp, yp) result = area(radius)
        return result
```

14.What is recursion?

The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function.

Eg:

```
def factorial(n):
        if n == 1:
                return 1
        else:
                return n * factorial(n-1)
```

15. Explain global and local scope.

The scope of a variable refers to the places that we can see or access a variable. If we define a variable on the top of the script or module, the variable is called global variable. The variables that are defined inside a class or function is called local variable.

Eg:

```
def my_local():
        a=10
        print("This is local variable")
```

Eg:

```
a=10
def my_global():
        print("This is global variable")
```

16.Compare string and string slices.

A string is a sequence of character.

Eg: fruit = 'banana'

String Slices :

A segment of a string is called string slice, selecting a slice is similar to selecting a character. Eg: >>> s ='Monty Python'

```
>>>     print s[0:5] Monty
>>>     print s[6:12] Python
```

17. Define string immutability.

Python strings are immutable. 'a' is not a string. It is a variable with string value. We can't mutate the string but can change what value of the variable to a new string.

18.Mention a few string functions.

s.captilize() – Capitalizes first character of string s.count(sub) – Count number of occurrences of sub in string

s.lower() – converts a string to lower case

s.split() – returns a list of words in string

19. What are string methods?

A method is similar to a function—it takes arguments and returns a value—but the syntax is different. For example, the method upper takes a string and returns a new string with all uppercase letters. Instead of the function syntax upper(word), it uses the method syntax word.upper().

```
>>> word = 'banana'
>>>     new_word = word.upper()
>>>     print new_word
BANANA
```

20. Explain about string module.

The string module contains number of useful constants and classes, as well as some deprecated legacy functions that are also available as methods on strings.

Eg: import string

string.upper(text) → converts to upper case

string.lower(text) → converts to lower case

21. What is the purpose of pass statement?

Using a pass statement is an explicit way of telling the interpreter to do nothing.

Eg:

```
def bar():
        pass
```

If the function bar() is called, it does absolutely nothing.

UNIT IV LISTS, TUPLES, DICTIONARIES

Lists: list operations, list slices, list methods, list loop, mutability, aliasing, cloning lists, list parameters; Tuples: tuple assignment, tuple as return value; Dictionaries: operations and methods; advanced list processing - list comprehension; Illustrative programs: simple sorting, histogram, Students marks statement, Retail bill preparation.

4.1 LISTS

Define List

A list is an ordered set of values, where each value is identified by an index. The values in a list are called its elements or items. The items can be of different types (int, float, string). To create a new list, the simplest way is to enclose the elements in square bracket [ ]. Lists are mutable which means the items in the list can be add or removed later.

Example:

```
>>>[ ]                                #empty list
>>>[1,2,3]                            #list of integers
>>>['physics','chemistry','python']   #list of strings
>>>[1,'hello',3.4]                    #list with mixed datatypes
>>>list1=['a','b,'c''d']
>>>print(list1)
```

List can have another list as an item. This is called nested list.

Mylist=['mouse',[8,6,5], 3.2]

List are mutable.

Lists are mutable which means the items in the list can be added or removed later.

```
>>>mark=[98,87,94]
>>>mark[2]=100
>>>print(mark)        #Prints [98,87,100]
```

To access the elements in a list

The syntax for accessing an element is same as string. The square brackets are used to access the elements. The index value within the square brackets should be given.

```
>>>list1=[] #Empty list
>>>list2=[1,2,3,4,5,6,7,8]
>>>list3=['Hello',3.5,'abc',4]
print(list3[1]) → 3.5
```

List Length:

The function **len** returns the length of a list, which is equal to the number of elements.

len(list2) → 8

len(list3) → 4

List Membership:

The memberrship operator **"in"** and **"not in"** can also be used in a list to check whether the element is present in the list or not.

Ex:

list3=['Hello',3.5,'abc',4]

'Hello' in list3 → returns True

4.1.1 LIST OPERATIONS:

1. + Operator which concatenates two lists.

```
>>>list2=[1,2,3,4,5,6,7,8]
>>>list3=['Hello',3.5,'abc',4]
>>>print(list2+list3)
```

Output

```
1,2,3,4,5,6,7,8, 'Hello',3.5,'abc',4
```

2. * Operator multiples the list to the specific numbers

```
>>>list2=[1,2,3,4,5,6,7,8]
>>>list2*2
```

Output

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

4.1.2 LIST SLICE

A subsequence of a sequence is called a slice and the operation that extracts a subsequence is called slicing. For slicing we use square brackets [ ]. Two integer values splitted by **(:).**

Syntax:

List_Name[Starting_Value : Ending_Value]

Ex:

```
>>>a=['a','b','c','d','e']
```

List	a=	'a'	'b'	'c'	'd'	'e'
Index from Left		0	1	2	3	4
Index from Right		-5	-4	-3	-2	-1

>>>print(a[:]) → ['a', 'b', 'c', 'd', 'e']	*#Prints ALL*
>>> print(a[1:]) → ['b', 'c', 'd', 'e']	#Print from 1st Position to Last Position
>>> print(a[1:3]) → ['b', 'c']	#Print from 1st Position to Last – 1 Position
>>> print(a[:-1]) → ['a', 'b', 'c', 'd']	#Print from Backwards except -1th Position
>>> print(a[1:-1]) → ['b', 'c', 'd']	#Print from 1st Position till -1th Position

4.1.3 LIST METHODS (or) TYPES OF FUNCTIONS IN LIST

*Consider the values of **list a and list b** be*

>>>a=[‘apple’,’mango’,’lime’]

>>>b=[‘grape’]

S.No	Name	Syntax	Description	Example
1.	append()	listname.append()	The method append() will add the item to the end of a list	a.append(‘orange’)
2.	insert()	listname.insert(index,item)	This method inserts an item at a particular place and two arguments (index,item)	a.insert(1,’banana’)
3.	extend()	listname.extend(item1, item2)	This method is used to combine two list with the items in the argument.	a.extend(‘grape’) (or) a.extend(b)
4.	remove()	listname.remove(item)	This method will remove an item in the list.	a.remove(’apple’)
5.	pop()	listname.pop(index)	This method returns the item by the index position and removes it.	a.pop(1) >>>mango
6.	index()	listname.index(item)	This method will return index value of list and takes index value as argument.	a.index(‘lime’) >>>2
7.	copy()	dest_list=listname.copy()	This method is used to copy a list to another list.	c=a.copy()
8.	reverse()	listname.reverse()	This method is used to reverse	a.reverse()

			the items in a list.	
9.	count()	listname.count(item)	This method is used to count the duplicate items in the list which takes the item as arguments.	a.count('lime') >>>1
10.	sort()	listname.sort()	This method is used to arrange the list from ascending to descending alphabetically.	a.sort() >>>a=['apple', 'lime', 'mango']
11.	clear()	listname.clear()	This method is used to clear all the values in the list.	a.clear() -->[]

4.1.4 LIST LOOP:

A loop is to access all the elements in a list.

>>>a=['apple','mango','lime','orange']

>>>print(a) → ['apple','mango','lime','orange'] # *displays all at a time*

>>>print(a[0]) → 'apple'

i) for var in a:

print(a) #*displays all at a time*

Output:

The loop goes on 4 times and print all values

'apple','mango','lime','orange'

'apple','mango','lime','orange'

'apple','mango','lime','orange'

'apple','mango','lime','orange'

ii) for var in a:

print(var) #*displays a item at a time*

Output:

The loop goes on 4 times and print items one by one.

'apple'

'mango'

'lime'

'orange'

iii) >>>i=0

```
>>> for var in a:
            print("I like",a[i])
            i=i+1
```

Output:

The value of i makes the loop to goes on 4 times and print items one by one.

```
'apple'
'mango'
'lime'
'orange'
```

1. Write a python program to print the items in the list using while loop.

```
>>>a=['apple','mango','lime','orange']
>>>i=0
>>>while len(a)>i:
        print("I like",a[i])
        i=i+1
```

Output:

The value of i makes the loop to goes on 4 times and print items one by one.

```
'apple'
'mango'
'lime'
'orange'
```

4.1.5 LIST ARE MUTABLE:

Unlike String, List is mutable (changeable) which means we can change the elements at any point.

Ex:

```
>>>a=['apple','mango','lime','orange']
>>>a[0]='grape'
>>>print(a) → 'grape', 'mango','lime','orange'
```

4.1.6 LIST ALIASING:

Since variables refer to objects, if we assign one variable to another, both variables refer to same objects.(One or more variable can refer the same object)

```
>>>a=[1,2,3]
```

```
>>>b=a
>>>a is b                 # Displays True
```

4.1.7 LIST CLONING:

If we want to modify a list and also keep copy of the original we can use cloning to copy the list and make that as a reference.

Ex:

```
a=[1,2,3]
b=a[:]
print(b) → [1,2,3]
```

4.1.8 LIST PARAMETER:

Passing a list as an argument actually passes a reference to the list, not the copy of the list. We can also pass a list as an argument to the function.

Ex:

```
>>> def mul(a_list):   #a_list is a list passing as a parameter
        for index,value in enumerate(a_list):
                a_list[index]=2*value
        print(a_list)
>>> a_list=[1,2,3,4,5]
>>> mul(a_list)
```

Output:

```
[2, 4, 6, 8, 10]
```

4.2 TUPLE

Tuples are sequence of values much like the list. The values stored in the tuple can be of any type and they are indexed by integers.

The main difference between list and tuple is **Tuple is immutable**. Tuple is represented using '( )'

Syntax:

```
tuple_name=(items)
```

Ex:

```
>>> tuple1=('1','2','3','5')
>>>tuple2=('a','b','c')
>>>tuple3='3','apple','100'
```

TUPLES ARE IMMUTABLE:

The values of tuple cannot be changed.

```
>>> tuple1=('1','2','3','5')
tuple1[1]='4' #It Shows Error
```

Tuples can be immutable but if you want to add an item we can add it by

```
t1=('a','b')
t1=('A',)+t1[1:]  → t1=('A','b)
```

The disadvantage in this method is we can only add the items from the beginning.

4.2.1 TUPLE ASSIGNMENT

Tuple Assignment means assigning a tuple value into another tuple.

Ex:

```
 t=('Hello','hi')
>>>m,n=t
>>>print(m) → Hello
>>>print(n) → hi
>>>print(t) → Hello,hi
```

In order to interchange the values of the two tuples The following method is used.

```
>>>a=('1','4')
>>>b=('10','15')
>>>a,b=b,a
>>>print(a,b)
(('10','15'), ('1','4'))
```

COMPARING TUPLES

The comparison operator works with tuple and other sequence. It will check the elements of one tuple to another tuple. If they are equal it return true. If they are not equal it returns false.

```
>>>t1=('1','2','3','4','5')
>>>t2=('a','b','c')
>>>t1<t2  #It returns false
```

4.2.2 TUPLE AS RETURN VALUE

In a function a tuple can return multiple values where a normal function can return single value at a time.

Example-1:

Using a built-in function **divmod** which return quotient and remainder at the same time.

```
>>>t=divmod(7,3)
>>>print(t) → (2,1)
>>>quot,rem=divmod(7,3)
>>>print(quot) → 2
>>>print(rem) →1
```

Example-2:

```
def swap(a,b,c):
        return(c,b,a)
a=100
b=200
c=300
>>>print("Before Swapping",a,b,c)
>>>print("After Swapping",swap(a,b,c))
```

Output:

```
Before Swapping 100,200,300
After Swapping 300,200,100
```

4.3 DICTIONARIES

Dictionaries is an unordered collection of items. Dictionaries are a kind of hash table. The value of a dictionary can be accessed by a key. Dictionaries are enclosed by curly braces '{ }' and values can be accessed using square braces '[ ]'

Syntax:

```
dict_name= {key: value}
```

A Key can be any Immutable type like String, Number, Tuple. A value can be any datatype. The values can be repeated and the keys should not be repeated.

Ex:

```
>>>dict1={}
>>>dict2={1:10,2:20,3:30}
>>>dict3={'A':'apple','B':'200'}
```

>>>dict4={(1,2,3):'A',(4,5):'B'}

>>>dict5={[1,2,3]:'A',[4,5]:'B'} *#Error, Only Immutable types can be assigned in Keys*

4.3.1ACCESS, UPDATE, ADD, DELETE ELEMENTS IN DICTIONARY:

Accessing Values in Dictionary

To access dictionary elements, you can use the familiar square brackets along with the key to obtain its value.

Ex:

>>>d={'name':'xyz','age':23}

>>>d['name'] → 'xyz' *# since 'name' is a String datatype, it should be represented within quotes*

>>>d[name] → *shows error*

By get()method

>>>d.get('name') → 'xyz'

Update Values in Dictionary

You can update a dictionary by adding a new entry or a key-value pair, modifying an existing entry.

Ex:

>>> d={'name':'xyz','age':23}

>>>print(d) → {' name':'xyz','age':23}

>>>d['age'=24] *#modifying existing element*

>>>print(d) → {' name':'xyz','age':23}

By update method

>>>d1={'place':'abc'}

>>>d.update(d1)

print(d) → {'place':'abc',' name':'xyz','age':23}

Adding Values in Dictionary

>>>d['gender']='m' *#Adding new entry*

>>> print(d) → {'gender':'m', 'place':'abc',' name':'xyz','age':23}

Deleting or Removing Values in Dictionary

You can either remove individual dictionary elements or clear the entire contents of a dictionary.

>>>del d['name'] →{'gender':'m', 'place':'abc','age':23}

>>>d.clear()   *# remove all entries in dictionary*

>>>del d *#delete entire dictionary*

4.3.2 METHODS ON DICTIONARIES:

Consider the value of **Dictionary d and d1** as follows:
d={‘a’:1,’b’:2,’c’:3,’d’:4}
d1={‘e’:5,’f’:6}

S.No	Name	Syntax	Description	Example
1.	len()	len(dictonary)	Gives the total length of the dictionary.	len(d) → 4
2.	keys()	dictionary.keys()	Return the dictionary's keys.	>>> d.keys() ['a', 'c', 'b', 'd']
3.	values()	dictionary.values()	Return the dictionary's Values	>>> d.values() [1, 3, 2, 4]
4.	items()	dictionary.items()	Return the dictionary's Keys and Values	>>> d.items() [('a', 1), ('c', 3), ('b', 2), ('d', 4)]
5.	key in dict	key in dict	Returns True if the Key is in Dictionary, else returns false.	>>> 'a' in d True
6.	key not in dict	key not in dict	Returns True if the Key is not in Dictionary, else returns false.	>>> 'e' not in d True
7.	has_key (key)	dictionary.has_key(key)	Checks whether the dictionary has the specified key.	>>> d.has_key('a') True
8.	get()	dictionary.get(key)	Get the value and Return the value of key.	>>> d.get('b') 2

9.	update()	Dest_dictionary.update(Source_dict)	Update the dictionary with the key from existing keys.	>>>d.update(d1)
				>>> print(d)
				{'a': 1, 'c': 3, 'b': 2, 'e': 5, 'd': 4, 'f': 6}

10.	cmp()	cmp(dictionary1, dictionary2)	Compares elements of both dictionary. Returns 0 if the elements are same, Else returns 1.	>>> cmp(d,d1) 1

11	copy()	new_dict = original_dict.copy()	Return a copy of the dictionary.	>>> d2=d1.copy()
				>>> print(d2)
				{'e': 5, 'f': 6}

12.	pop()	dictionary.pop(‘key’)	Remove the item with key	>>> d.pop('f')

			and return its value	6
13.	popitems()	dictionary.popitem()	Remove and return an item with its key and value.	>>> d.popitem() ('a', 1)
14.	clear()	dictionary.clear()	Remove all items form the dictionary.	d.clear()

4.4 LIST COMPREHENSION:

List comprehension is an elegant way to define and create list in Python. These lists have often the qualities of sets. It consists of brackets containing an expression followed by a for clause, then zero or more for or if clauses.

Ex-1:

```
>>>x = [i for i in range(10)]
>>>print x
```

Output:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Ex-2:

```
>>>squares = []
>>>for x in range(10):
        squares.append(x**2)
    print (squares)
>>>squares = [x**2 for x in range(10)] # List comprehensions to get the same result:
>>>print (squares)
```

Output:

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

ILLUSTRATIVE EXAMPLES

Note- *Always get an Input from the USER.*

1.Write a Python Program to find Largest, Smallest, Second Largest, Second Smallest in a List without using min() & max() function.

```
>>>def find_len(list1):
        length = len(list1)
        list1.sort()
        print("Largest element is:", list1[length-1])
        print("Smallest element is:", list1[0])
        print("Second Largest element is:", list1[length-2])
        print("Second Smallest element is:", list1[1])
 >>>list1=[12, 45, 2, 41, 31, 10, 8, 6, 4]
>>>Largest = find_len(list1)
```

Output:

```
Largest element is: 45
Smallest element is: 2
Second Largest element is: 41
Second Smallest element is: 4
```

2. Find the output of the following program

Program 2.1:

```
>>>subject= ['Physics', 'Chemistry', 'Computer']
>>>mark=[98,87,94]
>>>empty=[]
>>> print(subject, mark, empty)
```

Output:

```
['Physics', 'Chemistry', 'Computer'], [98, 87, 94], []
```

Program 2.2:

```
>>>mark=[98,87,94]
>>> mark[2]=100
>>> print(mark)
```

Output:

```
 [98, 87, 100]
```

Program 2.3:

```
>>> subject= ['Physics', 'Chemistry', 'Computer']
>>> 'Chemistry' in subject
```

Output:

```
True
```

Program 2.4:

```
>>> subject= ['Physics', 'Chemistry', 'Computer']
>>> for s in subject:
        print(s)
```

Output:

```
Physics
Chemistry
Computer
```

Program 2.5:

```
>>> for i in range(len(mark)):
        mark[i] = mark[i] * 2
>>> print(mark)
```

Output:

```
[196, 174, 200]
```

WORKSHEETS

NOTE: Always get an Input from the User

4.1. Write a python program to interchange first and last elements in a list

Input : [1, 2, 3] →Output : [3, 2, 1]

Program:

Output:

4.2. Write a python program to swap two elements in a list

Input : List = [1, 2, 3, 4, 5], pos1 = 2, pos2 = 5

Output : [1, 5, 3, 4, 2]

Program:

Output:

4.3. Write a python program to find length of list without len()

Program:

Output:

4.4. Write a python program to check if element exists in list

Program:

Output:

4.5. Write a python program for Reversing a List

Program:

Output:

4.6. Write a python program Count occurrences of an element in a list

Program:

Output:

4.7. Write a python program to print even numbers in a list

Program:

Output:

4.8. Write a python program to print duplicates from a list of integers

Program:

Output:

TWO MARKS

1. What is a list?

A list is an ordered set of values, where each value is identified by an index. The values that make up a list are called its elements. Lists are similar to strings, which are ordered sets of characters, except that the elements of a list can have any type.

2. Solve a)[0] * 4 and b) [1, 2, 3] * 3.

```
>>>  [0] * 4 [0, 0, 0, 0]
>>>     [1, 2, 3] * 3 [1, 2, 3, 1, 2, 3, 1, 2, 3]
```

3.Let list = [’a’, ’b’, ’c’, ’d’, ’e’, ’f’]. Find a) list[1:3] b) t[:4] c) t[3:] .

```
>>>     list = [’a’, ’b’, ’c’, ’d’, ’e’, ’f’]
>>>     list[1:3] [’b’, ’c’]
>>>     list[:4] [’a’, ’b’, ’c’, ’d’]
>>>     list[3:] [’d’, ’e’, ’f’]
```

4. Mention any 5 list methods.

append() ,extend () ,sort(), pop(),index(),insert and remove()

5. State the difference between lists and dictionary.

List is a mutable type meaning that it can be modified whereas dictionary is immutable and is a key value store. Dictionary is not ordered and it requires that the keys are hashable whereas list can store a sequence of objects in a certain order.

6. What is List mutability in Python? Give an example.

Python represents all its data as objects. Some of these objects like lists and dictionaries are mutable, i.e., their content can be changed without changing their identity. Other objects like integers, floats, strings and tuples are objects that cannot be changed. Example:

```
>>>     numbers = [17, 123]
>>>     numbers[1] = 5
>>>     print numbers [17, 5]
```

7. What is aliasing in list? Give an example.

An object with more than one reference has more than one name, then the object is said to be aliased. Example: If a refers to an object and we assign b = a, then both variables refer to the same object:

```
>>>     a = [1, 2, 3]
>>>     b = a
>>>     b is a True
```

8. Define cloning in list.

In order to modify a list and also keep a copy of the original, it is required to make a copy of the list itself, not just the reference. This process is called cloning, to avoid the ambiguity of the word “copy”.

9. Explain List parameters with an example.

Passing a list as an argument actually passes a reference to the list, not a copy of the list. For example, the function head takes a list as an argument and returns the first element:

```
def head(list):
        return list[0]
output:
>>>     numbers = [1, 2, 3]
>>>     head(numbers)
```

10. Write a program in Python to delete first element from a list.

```
 def deleteHead(list): del list[0]
        Here’s how deleteHead is used:
>>>     numbers = [1, 2, 3]
>>>     deleteHead(numbers)
>>>     print numbers [2, 3]
```

11. Write a program in Python returns a list that contains all but the first element of the given list.

```
def tail(list): return list[1:]
```

Here's how tail is used:

```
>>> numbers = [1, 2, 3]
>>> rest = tail(numbers)
>>>     print rest [2, 3]
```

12. What is the benefit of using tuple assignment in Python?

It is often useful to swap the values of two variables. With conventional assignments a temporary variable would be used. For example, to swap a and b:

```
>>>     temp = a
>>>     a = b
>>>     b = temp
>>>     a, b = b, a
```

13.Define key-value pairs.

The elements of a dictionary appear in a comma-separated list. Each entry contains an index and a value separated by a colon. In a dictionary, the indices are called keys, so the elements are called key-value pairs.

14. Define dictionary with an example.

A dictionary is an associative array (also known as hashes). Any key of the dictionary is associated (or mapped) to a value. The values of a dictionary can be any Python data type. So dictionaries are unordered key-value-pairs.

Example:

```
>>> eng2sp = {}         # empty dictionary
>>>     eng2sp['one'] = 'uno'
>>>     eng2sp['two'] = 'dos'
```

15. How to return tuples as values?

A function can only return one value, but if the value is a tuple, the effect is the same as returning multiple values. For example, if we want to divide two integers and compute the quotient and remainder, it is inefficient to compute x/y and then x%y. It is better to compute them both at the same time.

```
>>>     t = divmod(7, 3)
>>>     print t (2, 1)
```

16. List two dictionary operations.

- Del -removes key-value pairs from a dictionary
- Len - returns the number of key-value pairs

17.Define dictionary methods with an example.

A method is similar to a function—it takes arguments and returns a value— but the syntax is different. For example, the keys method takes a dictionary and returns a list of the keys that appear, but instead of the function syntax keys(eng2sp), method syntax eng2sp.keys() is used.

```
>>>     eng2sp.keys() [’one’, ’three’, ’two’]
```

18.Define List Comprehension.

List comprehensions apply an arbitrary expression to items in an iterable rather than applying function. It provides a compact way of mapping a list into another list by applying a function to each of the elements of the list.

19.Write a Python program to swap two variables.

```
x = 5
y = 10
temp = x
x = y
y = temp
print('The value of x after swapping: {}'.format(x))
print('The value of y after swapping: {}'.format(y))
```

20.Write the syntax for list comprehension.

The list comprehension starts with a '[' and ']', to help us remember that the result is going to be a list. The basic syntax is [expression for item in list if conditional].

UNIT V FILES, MODULES, PACKAGES

Files and exception: text files, reading and writing files, format operator; command line arguments, errors and exceptions, handling exceptions, modules, packages; Illustrative programs: word count, copy file. Voter's age validation, Marks range validation (0-100)

5.1 - FILES AND EXCEPTION

5.1.1 FILES

A File is a collection of data or information which has a name, called filename. All the information is stored in the computer using a file. The file is a named location on disk to store related information. It is used to store data in volatile memory. There are many different types of files.

- Data File
- Text File
- Program File
- Directory File

Text File:

A **text file** is a sequence of characters stored on a permanent medium like a hard drive, flash memory, or CD-ROM. Text files are identified with the .txt file extension.

Reading and Writing to Text Files

Python provides inbuilt functions for creating, writing and reading files. There are two types of files that can be handled in python, normal text files and binary files (written in binary language,0s and 1s).

- **Text files:** In this type of file, each line of text is terminated with a special character called EOL (End of Line), which is the new line character ("\n") in python by default.
- **Binary files:** In this type of file, there is no terminator for a line and the data is stored after converting it into machine understandable binary language.

In order to perform some operations on files we have to follow below steps

- Opening
- Reading or writing
- Closing

5.1.2 FILE OPERATIONS

When the user want to do some operations in the file there is a sequence of steps that should be followed.

i. Open the File
ii. Read or Write data in the file
iii. Close the File

i .Open the File

Python has a built-in function open() to open a file. This function returns a file object and has two arguments which are **file name & opening mode**. The opening modes are reading, writing or appending.

Syntax:

```
file_object=open("filename", "mode")
```

where file_object is the variable to add the object and the mode tells the interpreter which way the file will be used.

Ex:

```
f=open("text.txt")
f=open("E:/Python/text.txt", 'r')
f=open("E:/Python/text.txt", 'w')
```

FILE OPENING MODE

S.No	Mode	Description
1.	"r"	Opening a file for reading(default mode)
2.	"w"	Opening a file for writing
3.	"x"	Opening a file for exclusive operations
4.	"a"	Opening a file for appending at the end of the file
5.	"b"	Opening a file in binary mode
6.	"t"	Opening a file in text mode
7.	"+"	Opening a file for updating and it comes with one operation or two operations.

ii. Read or Write data in the file

There are two methods to access the data in the file.

1. read()
2. write()

5.1 Write a Python Program to write a file and access a file

```
f=open("F:/Python/test.txt","w+")
f.write("Python Programming is Interesting")
f.seek(0)
print(f.read())
f.close()
```

There are several opening modes apart from the main opening modes.

S.No	Modes	Description
1.	rb	Opens a file for reading only in binary format.
2.	r+	Opens a file for both reading and writing.
3.	rb+	Opens a file for both reading and writing in binary format.
4.	wb	Opens a file for writing only in binary format.
5.	w+	Opens a file for both writing and reading in binary format.
6.	wb+	Opens a file for both writing and reading in binary format.
7.	a+	Opens a file for both appending and reading.

1. read() Method

This method reads a string from an open file.

Syntax:

```
file_object.read()
```

Ex:

```
f=open("F:/Python/test.txt","r")
print(f.read())
print(f.read(2))
f.close()
```

2.write() Method

This method writes a string from an open file.

Syntax:

```
file_object.write("String")
```

Ex:

```
f=open("F:/Python/test.txt","w+")
print(f.write("This is my first file"))
f.seek(0)
print(f.read())
f.close()
```

iii. Close the File

To close a file object we will use close() method. The file gets closed and cannot be used again for reading and writing.

Syntax:

```
file_object.close()
```

Ex:

```
f=open("F:/Python/test.txt","r")
print(f.read())
f.close()
```

How to read a text file?

1. File Operation
2. Opening Modes
3. Read Functions

   There are two functions to read a file line by line, then there are two methods

   - readline()
   - readlines()

1.readline()

Every time a user runs the method, it will return a string of characters that contains a single line of information from the file.

Syntax:

```
file_object.readline()
```

Ex:

Consider the file test.txt contain these three lines

```
This is the First Line in the file
This is the Second Line in the file
This is the Third Line in the file
```

Now the Python source code is

```
f=open("F:/Python/test.txt","r")
print(f.readline())
f.close()
```

Output:

```
This is the First Line in the file          #Returns the first line
```

2.readlines()

This method is used to return a list containing all the lines separated by the special character (\n).

Syntax:

```
file_object.readlines()
```

Ex:

Consider the same file

Now the Python source code is

```
f=open("F:/Python/test.txt","r")
print(f.readline())
f.close()
```

Output:

```
['This is the First Line in the file\n', 'This is the Second Line in the file\n', 'This is the Third Line in the file']
```

5.1.3 LOOPING OVER A FILE OBJECT

5.2. Write a python program to read a function without using read() function.

```
>>> f=open("test.txt","r")
>>>for line in f:
        print(line)
>>>f.close()
```

5.3. Write a python program to access a file using with statement.

Syntax:

```
with open("filename") as file:
```

Ex:

```
with open("test.txt") as f:
for line in f:
        print(line)
```

5.1.4 FILE MANIPULATIONS

File manipulation means change or access the content in the file.

1. File Positions
2. Renaming and Delete a File
3. Directories in Python

1. File Positions

There are two methods to access the positions of the file.

1. tell()

2.seek()

i) tell() method

This method is used to tell the current position within a file. It starts from the beginning of the file and this method followed by read() and write() method.

Syntax:

```
file_object.tell()
```

Ex:

```
>>> f=open("F:/Python/test.txt","r")
>>>print(f.tell())                          #0th poisition
>>>print(f.read())
>>>print(f.tell())                          #Last Position
>>>f.close()
```

Output:

```
0
This is the First Line in the file
This is the Second Line in the file
73
```

ii) seek() method

This method is used to change the current file position.

Syntax:

```
file_object.seek(offset, from)
```

seek() method set the file's current position at the offset. The default argument of **offset is 0**. The offset represents the number of the bytes to be moved.

The **from argument** represents three values.

0 → represents the beginning of the file
1→ represents the current position as reference
2→ represents the end of the file

Ex:

```
>>> f=open("F:/Python/test.txt","r")
>>>print(f.tell())                          #0th poisition
>>>print(f.read())
>>>f.seek(0,0)
>>>f.seek(0,1)
>>>f.seek(0,2)
>>>f.close()
```

2. Renaming and Delete a File

Two file processing operations are there, they are

i. rename() method
ii. remove() method

i) rename() method

The rename() method takes two argument, the current filename and new filename.

Syntax:

```
os.rename(current_filename, new_filename)
```

Ex:

```
>>>import os
>>>os.rename("test.txt","new.txt ")
```

ii) remove() method

The remove() method is used to delete the file. The argument contains file name.

Syntax:

```
os.remove(filename)
```

Ex:

```
>>>import os
>>>os.rename("new.txt ")
```

3. Directories in Python

All files are contained within various directories. The os module has several methods to create, remove and change directories.

S.No	Name	Syntax	Description	Example
1.	mkdir()	os.mkdir(“new_dir”)	This method is used to create a directory	os.mkdir(“test”)
2.	chdir()	os.chdir(“new_dir”)	This method is used to change a directory	os.chdir(“new_dir”)
3.	getcwd()	os.getcwd()	This method is used to display current directory	os.getcwd()
4.	rmdir()	rmdir(‘dir_name’)	This method is used to remove a directory	os.rmdir(‘new_dir’)

5.1.5 FORMAT OPERATOR

The argument of write has to be a string, so if we want to put other values in a file, we have to convert them to strings. The easiest way to do that is with str:

```
>>> f=open('stringsample.txt','w')
>>> f.write(5)             #TypeError
>>> f.write(str(5))
```

An alternative is to use the **format operator**, %. The first operand is the **format string**, which contains one or more **format sequences**, which specify how the second operand is formatted. The result is a string.

Example:

```
>>>var=8
>>>print("The Value is : %d"%var)
```

Output:

```
The Value is : 8
```

Some of the format strings are.

Sl No	Conversion	Meaning
1	d	Signed integer decimal.
2	i	Signed integer decimal
3	o	Unsigned octal.
4	u	Unsigned decimal.
5	x	Unsigned hexadecimal (lowercase).
6	X	Unsigned hexadecimal (uppercase).
7	e	Floating point exponential format (lowercase).
8	E	Floating point exponential format (uppercase).
9	f	Floating point decimal format.
10	F	Floating point decimal format.

5.1.6 COMMAND LINE ARGUMENTS

It is possible to pass some values from the command line to python programs when they are executed. These values are called command line arguments and it can be used to control program from outside instead of hard coding.

The command line arguments are handled using sys module. We can access command-line arguments via the **sys.argv** This serves two purposes −

- sys.argv is the list of command-line arguments.
- len(sys.argv) is the number of command-line arguments.

Here sys.argv[0] is the program name.

Example 1

Consider the following script command.py

```
>>>import sys
>>>program_name = sys.argv[0]
>>>arguments = sys.argv[1:]
>>>count = len(arguments)
>>>print(program_name)
>>>print(arguments)
>>>print(count)
```

Output:

```
>>>C:\Python30>python.exe command.py Hello World
command.py Hello World 11
```

5.2 - ERRORS AND EXCEPTIONS:

5.2.1 - ERRORS

Errors or mistakes in a program are often referred to as bugs. The process of finding and eliminating the errors is called debugging. Errors can be classified into three major groups:

i. Syntax Error
ii. Runtime Error
iii. Logical Error

i) Syntax Error

Python will find these kinds of errors when it tries to parse your program, and exit with an error message without running anything. Syntax errors are mistakes in the use of the Python Language, and are analogous to spelling or grammar mistakes in a language like English.

Common Python Syntax Errors include:

- Leaving out a Keyword
- Putting a keyword in a wrong place
- Misspelling a Keyword
- Incorrect Indent
- Unnecessary Spaces
- Empty Spaces

Ex:

```
>>>my function(x,y):
        return x+y
>>>else:
        print("Hello")
>>>if mark>=50
        print("Passed")
>>>else mark:
print("Not Passed")
```

ii) Runtime Error

If a program is syntactically correct that is free from syntax errors, however the program exits unexpectedly it is due to runtime error. When a program comes to halt because of runtime error, then it has crashed.

Common Python Runtime Errors include:

- Division by Zero
- Performing an operation on different data type

- Using an identifier which has not been defined
- Accessing a list element which doesn't exist
- Trying to access a file which doesn't exist

Ex:

```
>>>a=10
>>>b=0
>>>c=a/b
```

iii) Logical Error

Logical Errors are the most difficult one to fix. They occur when the program runs without crashing but it produces an incorrect result. The error is occurred by a mistake in program logic.

Common Python Logical Errors include:

- Using the wrong variable name
- Indenting a block to the wrong level
- Using Integer division instead of float division
- Getting Operator Precedence wrong (Priority)

5.2.1 – HANDLING EXCEPTION

- Define Exception
- How to handle Exception?
- Types of Exception
    - Built-in Exception
    - User Defined Exception
- Assertion

Exception

An Exception is an event which occurs during the execution of a program that disrupts the normal flow of the program's instructions. If a python script encounters a situation it cannot cope up, it raises an exception.

5.2.2 HOW TO HANDLE EXCEPTION?

There are four blocks which help to handle the exception. They are

i. try block
ii. except statement
iii. else block
iv. finally block

i) *try block*

In the try block a programmer will write the suspicious code that may raise an exception. One can defend their program from a run time error by placing the codes inside the try block.

Syntax:

```
try:
    #The operations here
```

ii) except statement

Except statement should be followed by the try block. A single try block can have multiple except statement. The except statement which handles the exception. Multiple except statements require multiple exception names to handle the exception separately.

Syntax:

```
except Exception 1:
    #Handle Exception 1
except Exception 2:
    #Handle Exception 2
```

iii) else block

If there is no exception in the program the else block will get executed.

Syntax:

```
else:
    #If no Exception, it will execute
```

iv) finally block:

A finally block will always execute whether an exception happens or not the block will always execute.

Syntax:

```
finally:
    #Always Execute
```

The flow as follows

Syntax:

```
try:
    #The operations here
except Exception 1:
    #Handle Exception 1
except Exception 2:
    #Handle Exception 2
else:
    #If no Exception, it will execute
finally:
    #Always Execute
```

Ex:

5.4 Write a Python program to write a file with Exception Handling

```
try:
    f=open("test.txt", 'w+')
    f=write("My First File")
    f.seek(0)
    print(f.read())
except IOError:
    print("File not Found")
else:
    print("File not Found")
    f.close()
finally:
     print("Close a file")
```

Ex:

5.5 Write a python program to read a file which raises an exception.

Assume test.txt is not created in the computer and the following program is executed which raises an Exception.

```
try:
    f=open("test.txt", 'w+')
    f=write("My First File")
    f.seek(0)
    print(f.read())
except IOError:
    print("File not Found")
else:
    print("File not Found")
    f.close()
finally:
     print("Close a file")
```

Except clause with no Exception

An except statement without the exception name, the python interpreter will consider as default 'Exception' and it will catch all exceptions.

Syntax

```
try:
    #The operations here
except:
     #Handles Exception
else:
    #If no Exception, it will execute
finally:
    #Always Execute
```

Except clause with multiple Exception

An except statement can have multiple exceptions. We call it by exception name.

Syntax

```
try:
    #The operations here
except (Exception 1, Exception 2):
     #Handles Exception
else:
    #If no Exception, it will execute
finally:
    #Always Execute
```

Ex:

5.6 Write a python program to read a file with multiple exceptions.

```
try:
    f=open("test.txt", 'w+')
    f=write("My First File")
    print(f.read())
except (IOError, ValueError, ZeroDivisionError):
    print("File not Found")
else:
    print("File not Found")
    f.close()
```

```
finally:
    print("Close a file")
```

Argument of an Exception

An Exception can have an argument which is a value that gives additional information the problem. The content of an argument will vary by the exception.

Syntax

```
try:
    #The operations here
except Exception Type, Argument:
     #Handles Exception with Argument
else:
    #If no Exception, it will execute
finally:
    #Always Execute
```

Raising an Exception:

You can raise exceptions in several ways by using raise statement.

Syntax

```
raise Exception, Argument:
```

Ex:

```
>>>def fun(level):
        if level<10:
                raise "Invalid Level", level
>>>fun(5)               #raise an Exception
>>>fun(11)
```

5.2.3 TYPES OF EXCEPTION:

There are two types of Exception:

i. Built-in Exception
ii. User Defined Exception

i) Built-in Exception

There are some built-in exceptions which should not be changed.

The Syntax for all Built-in Exception

except Exception_Name

S.No	Exception Name	Description
1.	Exception	It is the Base class for all Exceptions
2.	ArithmeticError	It is the Base class for all Errors that occur on numeric calculations.
3.	ZeroDivisionError	Raised when a number is divided by zero
4.	IOError	Raised when an Input or Output operation fails such as open() function. When a file is not exist in the folder
5.	TypeError	Raised when operation or function is invalid for a specified data type.
6.	ValueError	Raised when built-in function for a data type has valid arguments and it has invalid values.
7.	RuntimeError	Raised when a generated error does not fall into any category.
8.	KeyboardInterrupt	Raised when the user interrupts program execution by pressing Ctrl+C
9.	FloatingPointError	Raised when floating calculation Fails.
10.	AssertionError	Raised in case of failure of assert statement.
11.	OverFlowError	Raised when a calculation exceeds maximum limit for a numeric types.
12.	StandardError	Base class for all built-in exception except 'StopIteration and SystemExit'
13.	StopIteration	Raised when next() method of an iteration does not exist
14.	SystemExit	Raised by the sys.exit() function
15.	SyntaxError	Raised when there is an error in Python Syntax
16.	IndentationError	Raised when Indentation is not specified properly

ii) User Defined Exception

In Python a user can create their own exception by deriving classes from standard Exceptions. There are two steps to create a user defined exception.

Step-1

A User Defined Exception should be derived from standard Built-in Exceptions.

Step-2

After referring base class the user defined exception can be used in the program

Ex:

```
class NetworkError(RuntimeError):
        def __init__ (self,arg):
                self.args=arg
try:
    raise NetworkError("Bad host name")
except NetworkError,e:
    print(e.args)
```

5.2.4 ASSERTION

An assertion is a sanity check which can turn on (or) turn off when the program is in testing mode.

The assertion can be used by assert keyword. It will check whether the input is valid and after the operation it will check for valid output.

Syntax:

```
assert(Expression)
```

Ex:

```
>>>def add(x):
        assert(x<0)
        return(x)
>>>add(10)
```

5.3 MODULES

A module allows you to logically organize the python code. Grouping related code into a module makes the code easy to use. A module is a file consisting python code. A module can define functions, classes and variables. A module can also include runnable code.

Ex

The Python code for a module **support** normally resides in a file named **support.py**.

support.py

```
>>>def print_func( par ):
        print "Hello : ", par
        return
```

We can invoke a module by two statements

i. import statement
ii. from…import statement

i) The import Statement

You can use any Python source file as a module by executing an import statement in some other Python source file.

Syntax:

```
import module
```

Ex

```
import support                    # Import module support
support.print_func("xyz")         # Now we can call the function in that module as
```

Output:

```
Hello : Zara
```

ii) The from...import Statement

Python's from statement lets you import specific attributes from a module into the current namespace.

Syntax:

```
from module_name import function_name
```

*The from...import * Statement*

It is also possible to import all names from a module.

Syntax:

```
from module_name import *
```

5.4 PACKAGES

A package is a hierarchical file directory structure that defines a single Python application environment that consists of modules and subpackages and sub-subpackages, and so on.

Consider a file Pots.py available in Phone directory. This file has following line of source code −

```
>>>def Pots():
        print "I'm Pots Phone"
```

Similar way, we have another two files having different functions with the same name as above

- Phone/Isdn.py file having function Isdn()
- Phone/G3.py file having function G3()

We can import by

```
>>>import Phone
>>>Phone.Pots()
>>>Phone.Isdn()
>>>Phone.G3()
```

Output:

```
I'm Pots Phone
I'm 3G Phone
I'm ISDN Phone
```

TWO MARKS

1. What is a text file?

A text file is a file that contains printable characters and whitespace, organized in to lines separated by newline characters.

2. Write a python program that writes "Hello world" into a file.

```
 f =open("ex88.txt",'w')
f.write("hello world")
f.close()
```

3. Write a python program that counts the number of words in a file.

```
f=open("test.txt","r")
content =f.readline(20)
words =content.split()
print(words)
```

4. What are the two arguments taken by the open() function?

The open function takes two arguments : name of the file and the mode of operation.

Example: f = open("test.dat","w")

5. What is a file object?

A file object allows us to use, access and manipulate all the user accessible files. It maintains the state about the file it has opened.

6. What information is displayed if we print a file object in the given program?

```
 f= open("test.txt","w")
print f
```

The name of the file, mode and the location of the object will be displayed.

7. What is an exception?

Whenever a runtime error occurs, it creates an exception. The program stops execution and prints an error message. For example, dividing by zero creates an exception:

```
print 55/0
ZeroDivisionError: integer division or modulo
```

8. What are the error messages that are displayed for the following exceptions?

a. Accessing a non-existent list item → IndexError: list index out of range

b. Accessing a key that isn't in the dictionary → KeyError: what

c. Trying to open a non-existent file → IOError: [Errno 2] No such file or directory: 'filename'

9. What are the two parts in an error message?

The error message has two parts: the type of error before the colon, and specific about the error after the colon.

10. How do you handle the exception inside a program when you try to open a non-existent file?

```
filename = raw_input('Enter a file name: ')
try:
        f = open (filename, "r")
except IOError:
        print 'There is no file named', filename
```

11. How does try and execute work?

The try statement executes the statements in the first block. If no exception occurs, then except statement is ignored. If an exception of type IOError occurs, it executes the statements in the except branch and then continues.

12. What is the function of raise statement? What are its two arguments?

The raise statement is used to raise an exception when the program detects an error. It takes two arguments: the exception type and specific information about the error.

13. What is a pickle?

Pickling saves an object to a file for later retrieval. The pickle module helps to translate almost any type of object to a string suitable for storage in a database and then translate the strings back in to objects.

14. What are the two methods used in pickling?

The two methods used in pickling are pickle.dump() and pickle.load(). To store a data structure, dump method is used and to load the data structures that are dumped , load method is used.

15. What is the use of the format operator?

The format operator % takes a format string and a tuple of expressions and yields a string that includes the expressions, formatted according to the format string.

16. What are modules?

A module is simply a file that defines one or more related functions grouped together. To reuse the functions of a given module, we need to import the module. Syntax: import <modulename>

17. What is a package?

Packages are namespaces that contain multiple packages and modules themselves. They are simply directories.

Syntax: from <mypackage> import <modulename>

18. What is the special file that each package in Python must contain?

Each package in Python must contain a special file called _init__.py

19. How do you delete a file in Python?

The remove() method is used to delete the files by supplying the name of the file to be deleted as argument.

Syntax: os.remove(filename)

20. How do you use command line arguments to give input to the program?

Python sys module provides access to any command-line arguments via sys.argv. sys.argv is the list of command-line arguments. len(sys.argv) is the number of command-line arguments.