NOTES OF LESSON

GE3151- PROBLEM SOLVING AND

PYTHON PROGRAMMING

UNIT I   ALGORITHMIC PROBLEM SOLVING

INTRODUCTION

PROBLEM SOLVING
Problem solving is the systematic approach to define the problem and creating number of solutions.

The problem solving process starts with the problem specifications and ends with a correct program.

PROBLEM SOLVING TECHNIQUES
Problem solving technique is a set of techniques that helps in providing logic for solving a problem.

Problem solving can be expressed in the form of
1. Algorithms.
2. Flowcharts.
3. Pseudo codes.
4. Programs

1.ALGORITHM

It is defined as a sequence of instructions that describe a method for solving a problem. In other words it is a step by step procedure for solving a problem

- Should be written in simple English
- Each and every instruction should be precise and unambiguous.
- Instructions in an algorithm should not be repeated infinitely.
- Algorithm should conclude after a finite number of steps.
- Should have an end point
- Derived results should be obtained only after the algorithm terminates.

Qualities of a good algorithm

The following are the primary factors that are often used to judge the quality of the algorithms.

Time – To execute a program, the computer system takes some amount of time. The lesser is the time required, the better is the algorithm.

Memory – To execute a program, computer system takes some amount of memory space. The lesser is the memory required, the better is the algorithm.

Accuracy – Multiple algorithms may provide suitable or correct solutions to a given problem, some of these may provide more accurate results than others, and such algorithms may be suitable

Building Blocks of Algorithm
As algorithm is a part of the blue-print or plan for the computer program. An algorithm is constructed using following blocks.
- Statements
- States
- Control flow
- Function

Statements

Statements are simple sentences written in algorithm for specific purpose. Statements may consists of **assignment statements, input/output statements, comment statements**

Example:

- Read the value of 'a' //This is input statement
- Calculate c=a+b   //This is assignment statement
- Print the value of c   // This is output statement

Comment statements are given after // symbol, which is used to tell the purpose of the line.

States

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

An algorithm will **definitely** have **start state** and **end state**.

Control Flow

Control flow which is also stated as flow of control, determines what section of code is to run in program at a given time. There are three types of flows, they are

1. Sequential control flow
2. Selection or Conditional control flow
3. Looping or repetition control flow

Sequential control flow:

The name suggests the sequential control structure is used to perform the action one after another. Only one step is executed once. The logic is top to bottom approach.

Example

Description: To find the sum of two numbers.

1. Start
2. Read the value of 'a'
3. Read the value of 'b'
4. Calculate sum=a+b
5. Print the sum of two number
6. Stop

Selection or Conditional control flow

Selection flow allows the program to make choice between two alternate paths based on condition. It is also called as **decision structure**

Basic structure:

IF*CONDITION* is **TRUE** then

perform some action

ELSE IF *CONDITION* is **FALSE** then

perform some action

The conditional control flow is explained with the example of finding greatest of two numbers.

Example

Description: finding the greater number

1. Start
2. Read a

3

3. Read b
4. If a>b then
    4.1. Print a is greater
else
    4.2. Print b is greater
5. Stop

Repetition control flow

       Repetition control flow means that one or more steps are performed repeatedly until some condition is reached. This logic is used for producing loops in program logic when one one more instructions may need to be executed several times or depending on condition.

    Basic Structure:
        Repeat until*CONDITION*is true
           Statements
    Example
        Description: to print the values from 1 to n

1. Start
2. Read the value of 'n'
3. Initialize i as 1
4. Repeat step 4.1 until i< n

    4.1. Print i
5. Stop

Function

       A function is a block of organized, reusable code that is used to perform a single, related action. Function is also named as **methods, sub-routines**.

    Elements of functions:
1. Name for declaration of function
2. Body consisting local declaration and statements
3. Formal parameter
4. Optional result type.

    Basic Syntax
        function_name(parameters)
           function statements
        end function

<u>Algorithm for addition of two numbers using function</u>
<u>Main function()</u>
      Step 1: Start
      Step 2: Call the function add()
      Step 3: Stop
  <u>sub function add()</u>
      Step1: Functionstart

Step 2: Get a, b Values
Step 3: add c=a+b
Step 4: Print c
Step 5: Return

2.NOTATIONS OF AN ALGORITHM

Algorithm can be expressed in many different notations, including **Natural Language, Pseudo code, flowcharts and programming languages**. Natural language tends to be verbose and ambiguous. Pseudocode and flowcharts are represented through structured human language.

A notation is a system of characters, expressions, graphics or symbols designs used among each others in problem solving to represent technical facts, created to facilitate the best result for a program

Pseudocode

Pseudocode is an **informal high-level description** of the operating principle of a **computer program** or **algorithm**. It uses the basic structure of a normal programming language, but is intended for human reading rather than machine reading.

It is text based detail design tool. **Pseudo means false** and **code** refers to **instructions written in programming language**.

Pseudocode cannot be compiled nor executed, and there are no real formatting or syntax rules. The pseudocode is written in normal English language which cannot be understood by the computer.

Example:

Pseudocode: To find sum of two numbers
READ num1,num2
sum=num1+num2
PRINT sum

Basic rules to write pseudocode:

1. Only one statement per line.

    Statements represents single action is written on same line. For example to read the input, all the inputs must be read using single statement.

2. Capitalized initial keywords

    The keywords should be written in capital letters. Eg: **READ, WRITE, IF, ELSE, ENDIF, WHILE, REPEAT, UNTIL**

    Example:

    Pseudocode: Find the total and average of three subjects
    RAED name, department, mark1, mark2, mark3
    Total=mark1+mark2+mark3
    Average=Total/3
    WRITE name, department,mark1, mark2, mark3

3. Indent to show hierarchy

    Indentation is a process of showing the boundaries of the structure.

4. End multi-line structures

    Each structure must be ended properly, which provides more clarity.

    Example:

Pseudocode: Find greatest of two numbers

READ a, b

IF a>b then

PRINT a is greater

ELSE

PRINT b is greater

ENDIF

5. Keep statements language independent.

Pesudocode must never written or use any syntax of any programming language.

Advantages of Pseudocode

- Can be done easily on a word processor
- Easily modified
- Implements structured concepts well
- It can be written easily
- It can be read and understood easily
- Converting pseudocode to programming language is easy as compared with flowchart
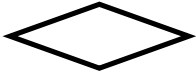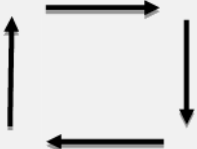
Disadvantages of Pseudocode

- It is not visual
- There is no standardized style or format
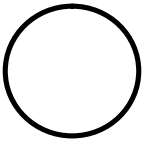
Flowchart

A **graphical representation** of an algorithm. Flowcharts is a diagram made up of boxes, diamonds, and other shapes, connected by arrows.

Each shape represents a step in process and arrows show the order in which they occur.

Table 1: Flowchart Symbols

S.No	Name of symbol	Symbol	Type	Description
1.	Terminal Symbol		Oval	Represent the start and stop of the program.
2.	Input/ Output symbol		Parallelogram	Denotes either input or output operation.
3.	Process symbol		Rectangle	Denotes the process to be carried
4.	Decision symbol		Diamond	Represents decision making and branching
5.	Flow lines		Arrow lines	Represents the sequence of steps and direction of flow. **Used to connect symbols**.

6.	Connector		Circle	A connector symbol is represented by a circle and a letter or digit is placed in the circle to specify the link. This symbol is **used to connect flowcharts**.

Rules for drawing flowchart

1. In drawing a proper flowchart, all necessary requirements should be listed out in logical order.
2. The flow chart should be clear, neat and easy to follow. There should not be any room for ambiguity in understanding the flowchart.
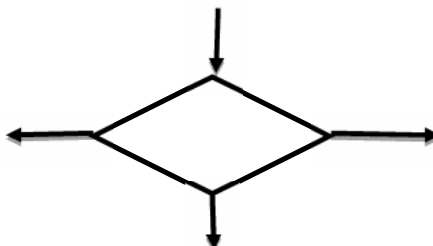3. The usual directions of the flow of a procedure or system is from left to right or top to bottom.
   Only one flow line should come out from a process symbol.
4. Only one flow line should enter a decision symbol, but two or three flow lines, one for each possible answer, cap leave the decision symbol.
5. Only one flow line is used in conjunction with terminal symbol.
6. If flowchart becomes complex, it is better to use connector symbols to reduce the number of flow lines.
7. Ensure that flowchart has logical start and stop.
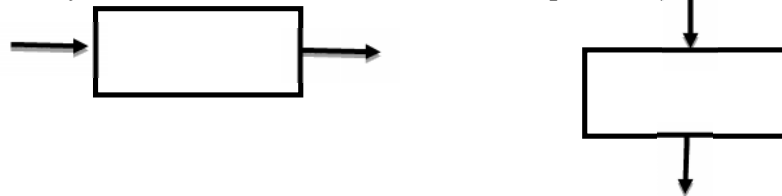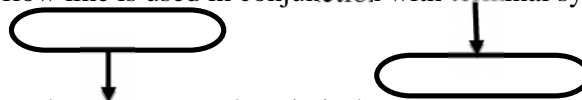
<u>**Advantages of Flowchart**</u>

Communication:
   Flowcharts are better way of communicating the logic of the system.

Effective Analysis
   With the help of flowchart, a problem can be analyzed in more effective way.

Proper Documentation
   Flowcharts are used for good program documentation, which is needed for various purposes.

Efficient Coding

The flowcharts act as a guide or blue print during the system analysis and program development phase.

Systematic Testing and Debugging

The flowchart helps in testing and debugging the program

Efficient Program Maintenance

The maintenance of operating program becomes easy with the help of flowchart. It helps the programmer to put efforts more efficiently on that part.

Disadvantages of Flowchart
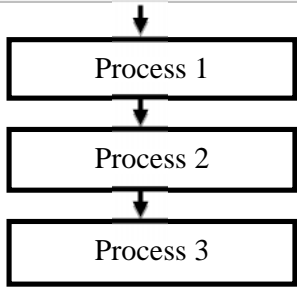
Complex Logic: Sometimes, the program logic is quite complicated. In that case flowchart becomes complex and difficult to use.
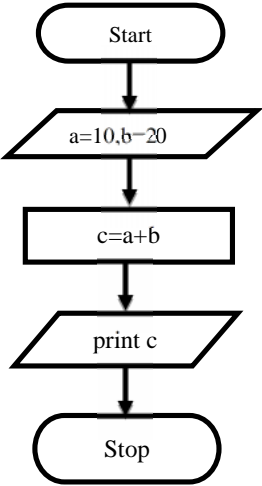
Alteration and Modification: If alterations are required the flowchart may require re-drawing completely.

Reproduction: As the flowchart symbols cannot be typed, reproduction becomes problematic.

Control Structures using flowcharts and Pseudocode

Sequence Structure

Pseudocode	Flow Chart
General Structure	
Process 1 …. Process 2 … Process 3	
Example	
READ a READ b Result c=a+b PRINT c	

Conditional Structure

- Conditional structure is used to check the condition. It will be having two outputs only (True or False)
- **IF** and **IF…ELSE** are the conditional structures used in Python language.
- **CASE** is the structure used to select multi way selection control. It is not supported in Python.

Pseudocode	Flow Chart
General Structure	

IF condition THEN
     Process 1
ENDIF	

Example	
READ a READ b IF a>b THEN PRINT a is greater	

IF… ELSE

IF…THEN…ELSE is the structure used to specify, if the condition is true, then execute Process1, else, that is condition is false then execute Process2

Pseudocode	Flow Chart
General Structure	

IF condition THEN
     Process 1
ELSE
     Process 2
ENDIF	

Example	

READ a READ b IF a>b THEN PRINT a is greater	
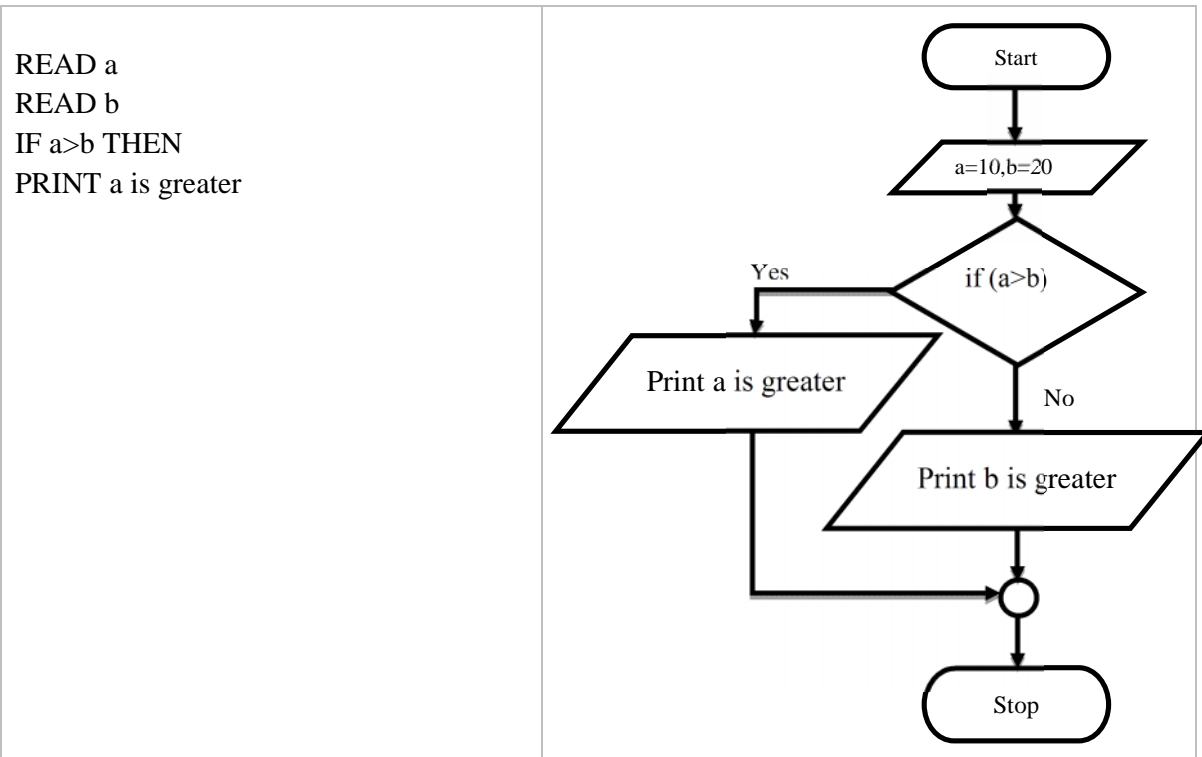
Iteration or Looping Structure

- Looping is generally used with **WHILE** or **DO...WHILE** or **FOR** loop.
- WHILE and FOR is entry checked loop

Pseudocode	Flow Chart
General Structure	
WHILE condition Body of the loop ENDWHILE	
Example	
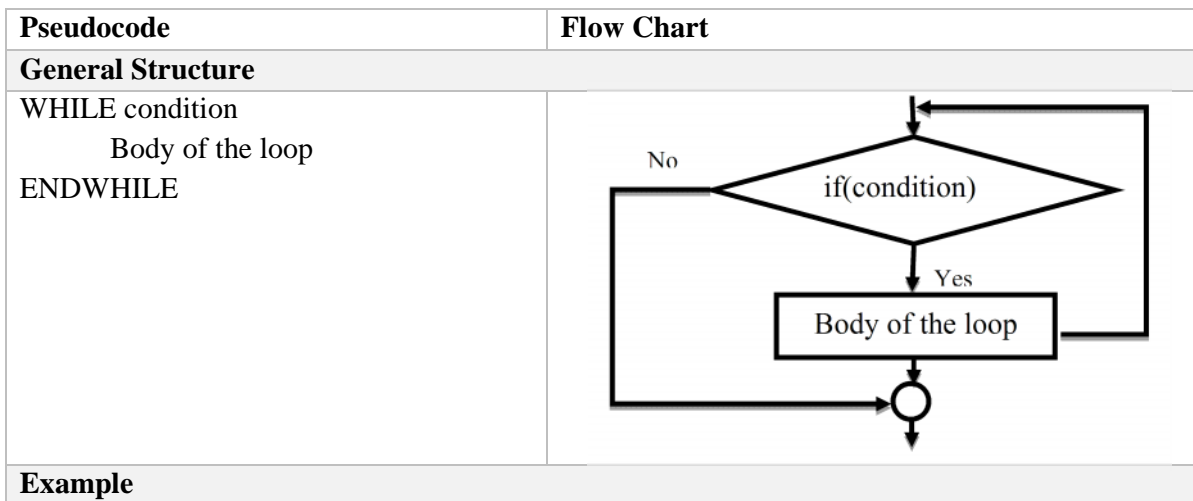
- DO…WHILE is exit checked loop, so the loop will be executed at least once.

| INITIALIZE a=1
WHILE a<10 THEN
    PRINT a
    a=a+1
ENDWHILE | |
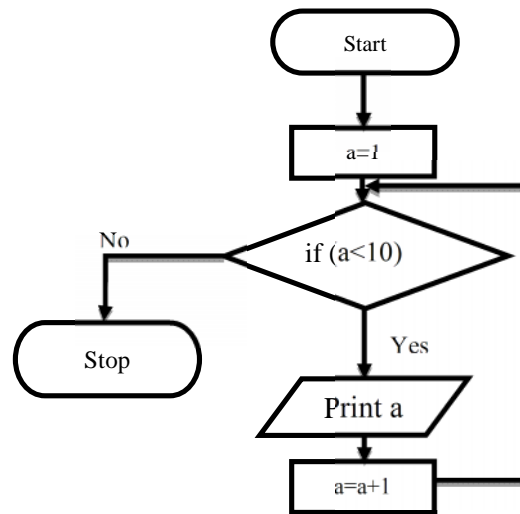
- o In python DO…WHILE is not supported.
- o If the loop condition is true then the loop gets into infinite loop, which may lead to system crash

Programming Language

- A programming language is a vocabulary and set of grammatical rules for instructing a computer or computing device to perform specific tasks. In other word it is set of instructions for the computer to solve the problem.
- Programming Language is a formal language with set of instruction, to the computer to solve a problem. The program will accept the data to perform computation.

Program= Algorithm +Data

Need for Programming Languages

- Programming languages are also used to organize the computation
- Using Programming language we can solve different problems
- To improve the efficiency of the programs.

Types of Programming Language

In general Programming languages are classified into three types. They are

- Low – level or Machine Language
- Intermediate or Assembly Language
- High – level Programming language

Machine Language:

Machine language is the lowest-level programming language (except for computers that utilize programmable microcode). Machine languages are the only languages understood by computers. It is also called as low level language.

Example code:100110011
111001100

Assembly Language:

An assembly language contains the same instructions as a machine language, but the instructions and variables have names instead of being just numbers. An assembler language consists of mnemonics, mnemonics that corresponds unique machine instruction.

Example code: start
addx,y
subx,y

High – level Language:

A high-level language (HLL) is a programming language such as C, FORTRAN, or Pascal that enables a programmer to write programs that are more or less independent of a particular type of computer. Such languages are considered high-level because they are closer to human languages and further from machine languages. Ultimately, programs written in a high-level language must be translated into machine language by a compiler or interpreter.

Example code: print("Hello World!")

High level programming languages are further divided as mentioned below.

Language Type	Example
Interpreted Programming Language	Python, BASIC, Lisp
Functional Programming Language	Clean, Curry, F#
Compiled Programming Language	C++,Java, Ada, ALGOL
Procedural Programming Language	C,Matlab, CList
Scripting Programming Language	PHP,Apple Script, Javascript
Markup Programming Language	HTML,SGML,XML
Logical Programming Language	Prolog, Fril
Concurrent Programming Language	ABCL, Concurrent PASCAL
Object Oriented Programming Language	C++,Ada, Java, Python

Interpreted Programming Language:

Interpreter is a program that executes instructions written in a high-level language.

An interpreter reads the source code one instruction or one line at a time, converts this line into machine code and executes it.

Figure : Interpreter

Compiled Programming Languages

Compile is to transform a program written in a high-level programming language from source code into object code. This can be done by using a tool called compiler.

A compiler reads the whole source code and translates it into a complete machine code program to perform the required tasks which is output as a new file.
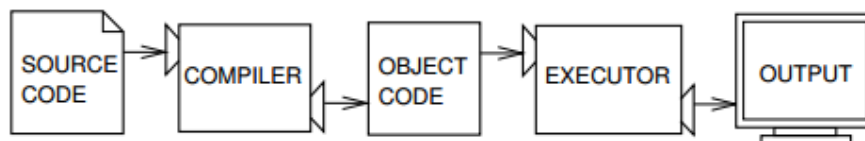
Figure: Compiler

Interpreted vs. Compiled Programming Language

Interpreted Programming Language	Compile Programming Language
Translates one statement at a time	Scans entire program and translates it as whole into machine code
It takes less amount of time to analyze the source code but the overall execution time is slower	It takes large amount of time to analyze the source code but the overall execution time is comparatively faster
No intermediate object code is generated, hence are memory efficient	Generates intermediate object code which further requires linking, hence requires more memory
Continues translating the program until first error is met, in which case it stops. Hence debugging is easy.	It generates the error message only after scanning the whole program. Hence debugging is comparatively hard.
Eg: Python, Ruby	Eg: C,C++,Java

3.ALGORITHMIC PROBLEM SOLVING:

Algorithmic problem solving is solving problem that require the formulation of an algorithm for the solution.
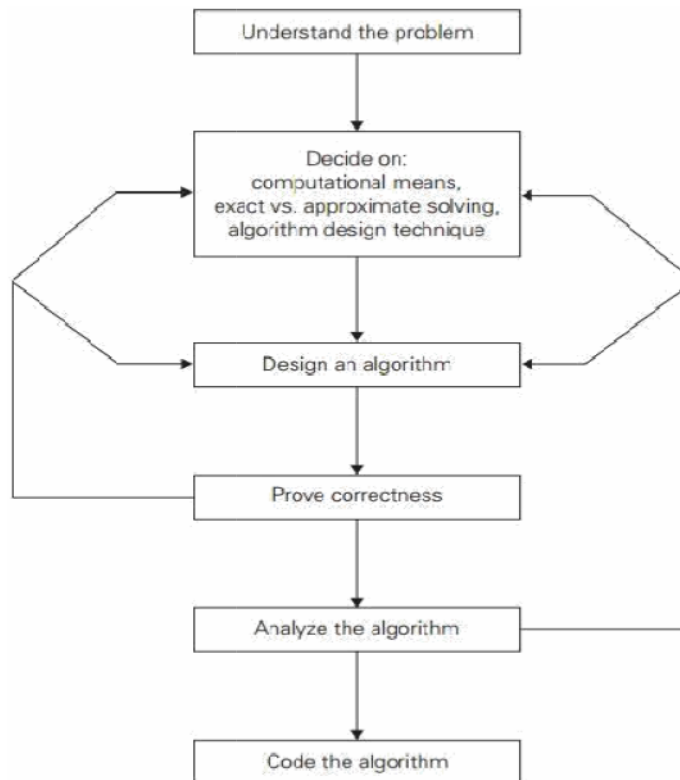
FIGURE 1.2 Algorithm design and analysis process.

Understanding the Problem

- It is the process of finding the input of the problem that the algorithm solves.
- It is very important to specify exactly the set of inputs the algorithm needs to handle.
- A correct algorithm is not one that works most of the time, but one that works Correctly for *all* legitimate inputs.

Ascertaining the Capabilities of the Computational Device

If the instructions are executed one after another, it is called sequential algorithm

Choosing between Exact and Approximate Problem Solving
- The next principal decision is to choose between solving the problem exactly or solving it approximately.
- Based on this, the algorithms are classified as exact *algorithm* and *approximation algorithm.*
- Data structure plays a vital role in designing and analysis the algorithms.
- Some of the algorithm design techniques also depend on the structuring data specifying a problem's instance
- Algorithm+ Data structure=programs.

Algorithm Design Techniques
- An *algorithm design technique* (or "strategy" or "paradigm") is a general approach to solving problems algorithmically that is applicable to a variety of problems from different areas of computing.
- Learning these techniques is of utmost importance for the following reasons.
- First, they provide guidance for designing algorithms for new problems,
  Second, algorithms are the cornerstone of computer science.

Methods of Specifying an Algorithm
- *Pseudocode* is a mixture of a natural language and programming language-like constructs. Pseudocode is usually more precise than natural language, and its usage often yields more succinct algorithm descriptions.In the earlier days of computing, the dominant vehicle for specifying algorithms was a *flowchart*, a method of expressing an algorithm by a collection of connected geometric shapes containing descriptions of the algorithm's steps.

- **Programming language** can be fed into an electronic computer directly. Instead, it needs to be converted into a computer program written in a particular computer language. We can look at such a program as yet another way of specifying the algorithm, although it is preferable to consider it as the algorithm's implementation.

- Once an algorithm has been specified, you have to prove its *correctness*. That is, you have to prove that the algorithm yields a required result for every legitimate input in a finite amount of time.

- A common technique for proving correctness is to use mathematical induction because an algorithm's iterations provide a natural sequence of steps needed for such proofs.

- It might be worth mentioning that although tracing the algorithm's performance for a few specific inputs can be a very worthwhile activity, it cannot prove the algorithm's correctness conclusively. But in order to show that an algorithm is incorrect, you need just one instance of its input for which the algorithm fails.

Analyzing an Algorithm

1. *Efficiency*.
   Time efficiency: indicating how fast the algorithm runs,
   Space efficiency: indicating how much extra memory it uses

2. *simplicity*.
   - An algorithm should be precisely defined and investigated with mathematical expressions.
   - Simpler algorithms are easier to understand and easier to program.
   - Simple algorithms usually contain fewer bugs.

Coding an Algorithm
   - Most algorithms are destined to be ultimately implemented as computer programs. Programming an algorithm presents both a peril and an opportunity.
   - A working program provides an additional opportunity in allowing an empirical analysis of the underlying algorithm. Such an analysis is based on timing the program on several inputs and then analyzing the results obtained.

4.Simple strategies for developing algorithm:
They are two commonly used strategies used in developing algorithm
   1. Iteration
   2. Recursion

Iteration
   - The iteration is when a loop repeatedly executes till the controlling condition becomes false
   - The iteration is applied to the set of instructions which we want to get repeatedly executed.
   - Iteration includes initialization, condition, and execution of statement within loop and update (increments and decrements) the control variable.

A sequence of statements is executed until a specified condition is true is called iterations.
   1. for loop
   2. While loop

Syntax for For:	Example: Print n natural numbers

FOR(*start-value* to *end-value)* DO
    statement
... ENDFOR
	BEGIN
	GET n
	 INITIALIZE i=1
	 FOR (i<=n) DO
	     PRINT i
	     i=i+
	1
	ENDFOR
	END

Syntax for While:	Example: Print n natural numbers

WHILE (condition) DO
    statement
    ...
ENDWHILE
	BEGIN
	GET n
	INITIALIZE i=1
	WHILE(i<=n) DO
	     PRINT i
	     i=i+1
	ENDWHILE
	END
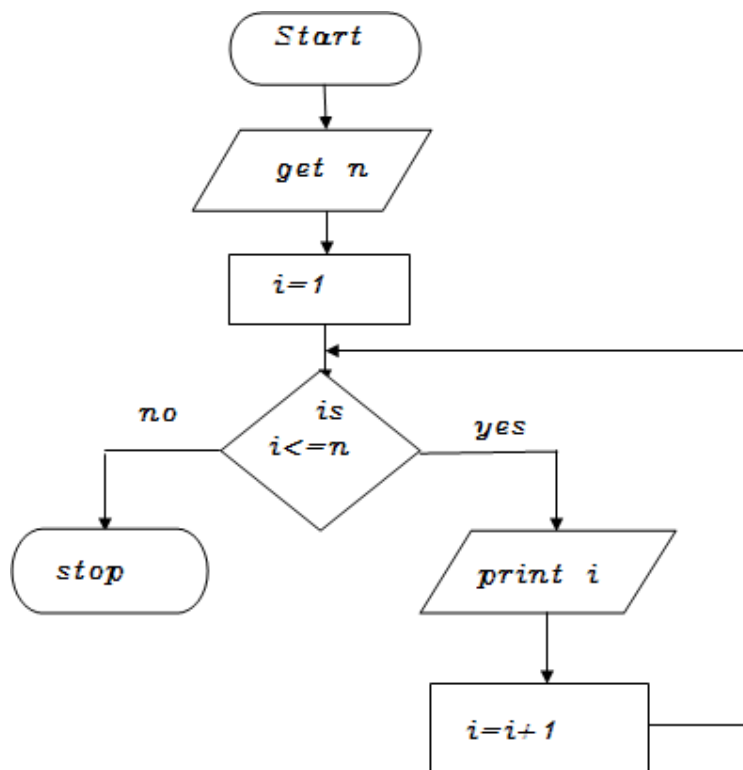
Recursions:

❖ A function that calls itself is known as recursion.

❖ Recursion is a process by which a function calls itself repeatedly until some specified condition has been satisfied.

Algorithm for factorial of n numbers using recursion:

Main function:

Step1: Start
Step2: Get n
Step3: call factorial(n)
Step4: print fact
Step5: Stop

Sub function factorial(n):

Step1: if(n==1) then fact=1 return fact
Step2: else fact=n*factorial(n-1) and return fact

FLOW CHART
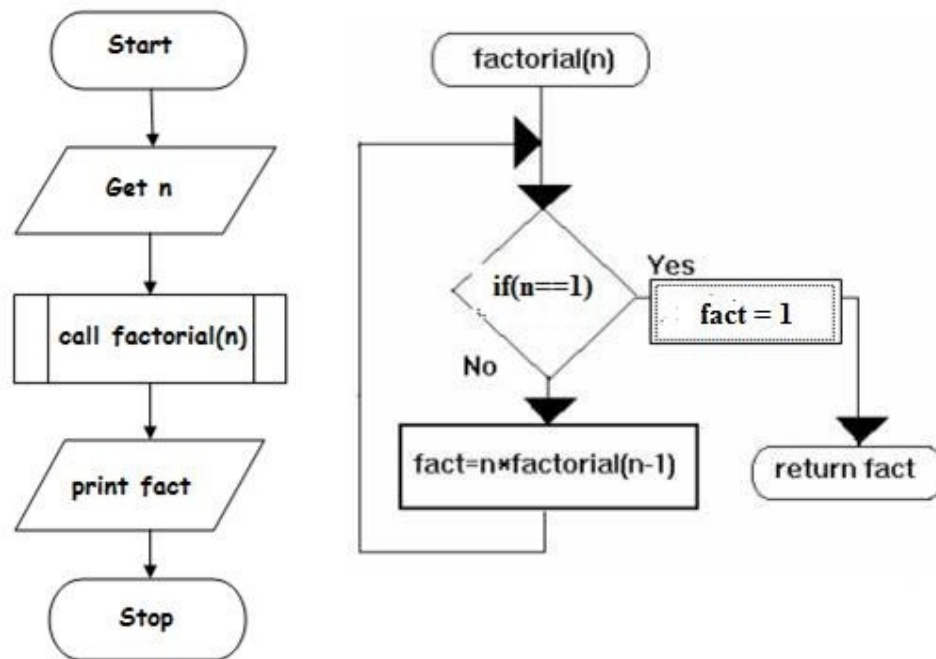
Pseudo code for factorial using recursion:

Main function:

BEGIN
GET n
CALL
factorial(n)
PRINT fact
BIN

Sub function factorial(n):

IF(n==1) THEN
        fact=1
        RETURN fact
ELSE
        RETURN fact=n*factorial(n-1)

5. ILLUSTRATIVE PROBLEMS

1. Guess an integer in a range

Algorithm:

Step1: Start

Step 2: Declare n, guess

Step 3: Compute guess=input

Step 4: Read guess

Step 5: If guess>n, then

  Print your guess is too high

 Else

Step6:If guess<n, then

  Print your guess is too low

 Else

Step 7:If guess==n,then

  Print Good job

 Else

  Nope

Step 6: Stop

Pseudocode:

BEGIN

COMPUTE guess=input

READ guess,

IF guess>n

PRINT Guess is high

ELSE

IF guess<n

PRINT Guess is low

ELSE

IF guess=n

 PRINT Good job

ELSE

Nope

END

Flowchart:
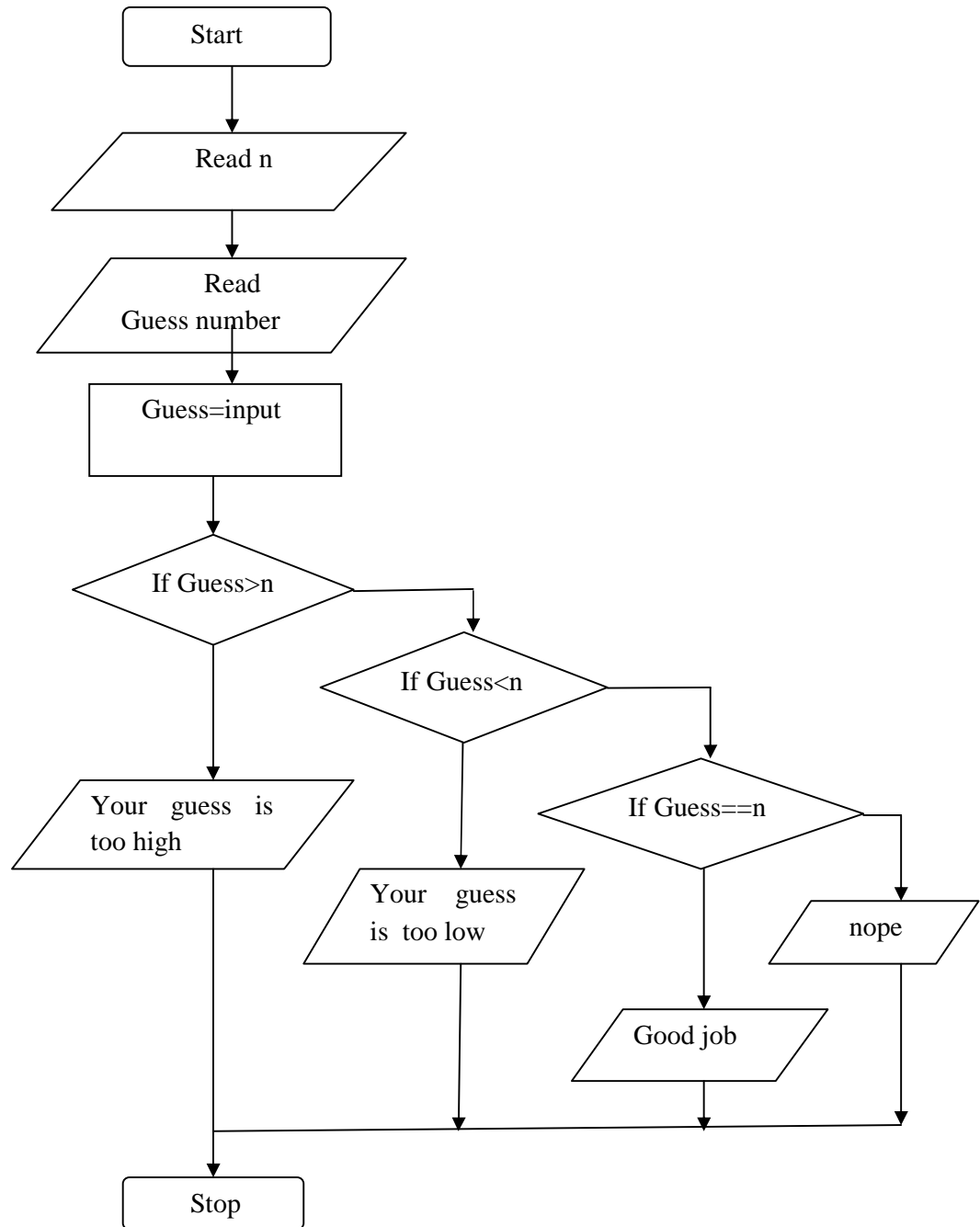
2.Find minimum in a list
Algorithm:

      Step 1: Start
      Step 2: Read n
      Step 3:Initialize i=0
      Step 4: If i<n, then goto step 4.1, 4.2 else goto step 5
      Step4.1: Read a[i]
      Step 4.2: i=i+1 goto step 4
      Step 5: Compute min=a[0]
      Step 6: Initialize i=1
      Step 7: If i<n, then go to step 8 else goto step 10
      Step 8: If a[i]<min, then goto step 8.1,8.2 else goto 8.2
      Step 8.1: min=a[i]
      Step 8.2: i=i+1 goto 7
      Step 9: Print min
      Step 10: Stop

Pseudocode:

      BEGIN
      READ n
      FOR i=0 to n, then
      READ a[i]
      INCREMENT i
      END FOR
      COMPUTE min=a[0]
      FOR i=1 to n, then
      IF a[i]<min, then
      CALCULATE min=a[i]
      INCREMENT i
      ELSE
      INCREMENT i
      END IF-ELSE
      END FOR
      PRINT min
      END

Flowchart:
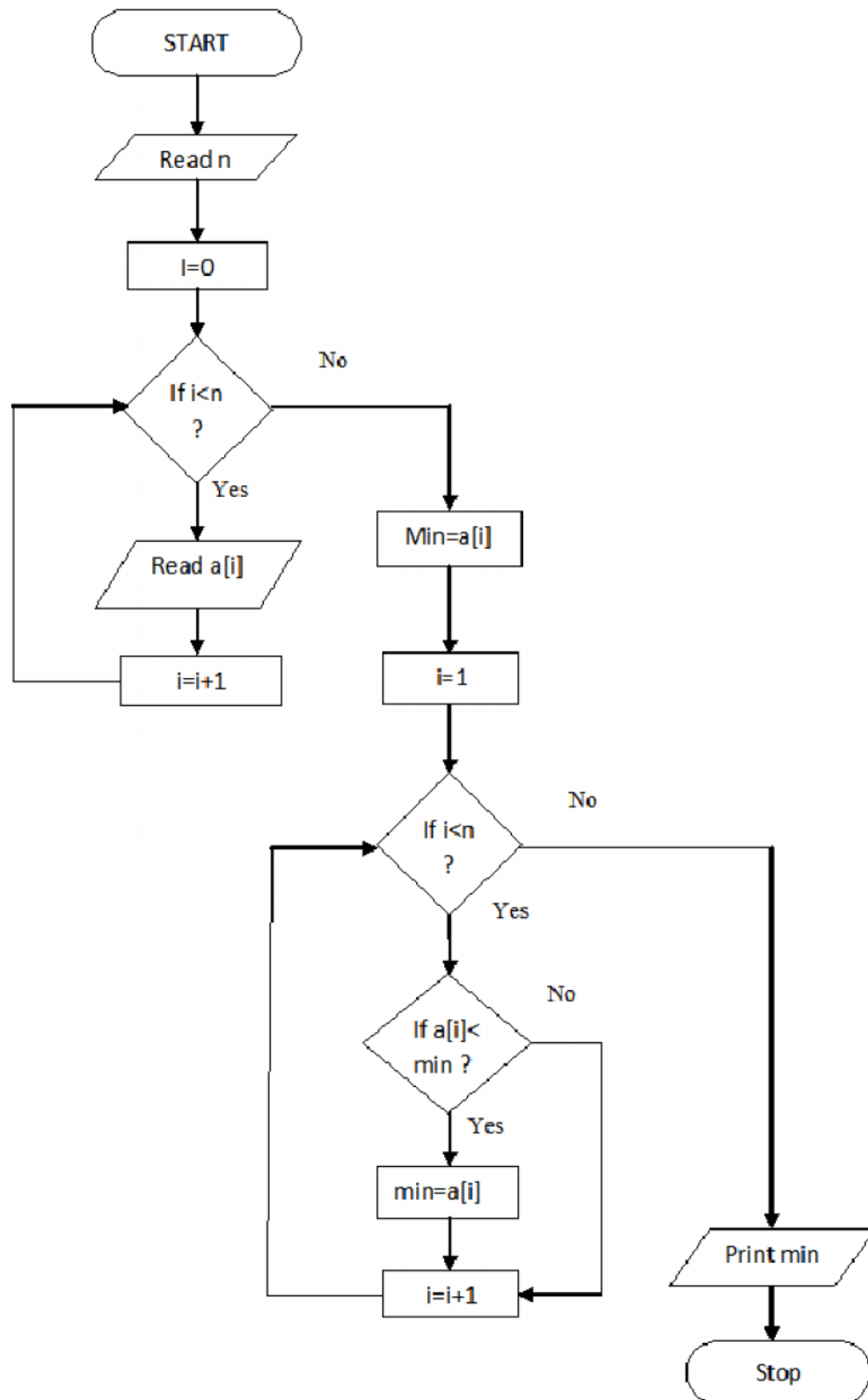
3. Insert a card in a list of sorted cards

Algorithm:
> Step 1: Start
> Step 2: Read n
> Step 3:Initialize i=0
> Step 4: If i<n, then goto step 4.1, 4.2 else goto step 5
> Step4.1: Read a[i]
> Step 4.2: i=i+1 goto step 4
> Step 5: Read item
> Step 6: Calculate i=n-1
> Step 7: If i>=0 and item<a[i], then go to step 7.1, 7.2 else goto step 8
> Step 7.1: a[i+1]=a[i]
> Step 7.2: i=i-1 goto step 7
> Step 8: Compute a[i+1]=item
> Step 9: Compute n=n+1
> Step 10: If i<n, then goto step 10.1, 10.2 else goto step 11
> Step10.1: Print a[i]
> Step10.2: i=i+1 goto step 10
> Step 11: Stop

Pseudocode:
> BEGIN
> READ n
> FOR i=0 to n, then
> READ a[i]
> INCREMENT i
> END FOR
> READ item
> FOR i=n-1 to 0 and item<a[i], then
> CALCULATE a[i+1]=a[i]
> DECREMENT i
> END FOR
> COMPUTE a[i+1]=a[i]
> COMPUTE n=n+1
> FOR i=0 to n, then
> PRINT a[i]
> INCREMENT i
> END FOR
> END

Flowchart:


```
START
  │
  ▼
Read n
  │
  ▼
i=0
  │
  ▼
If i<n ? ──No──► Read item
  │                  │
 Yes                 ▼
  ▼               i=n-1
Read a[i]            │
  │                  ▼
  ▼          If i>=0 & item<a[i]? ──No──► a[i+1] = a[i]
i=i+1            │                              │
                Yes                             ▼
                 ▼                            n=n+1
            a[i+1]=a[i]                         │
                 │                              ▼
                 ▼                            i=0
              i=i-1                            │
                                               ▼
                                         If i<n ? ──No──► Stop
                                               │
                                              Yes
                                               ▼
                                         Print a[i]
                                               │
                                               ▼
                                           i=i+1
```
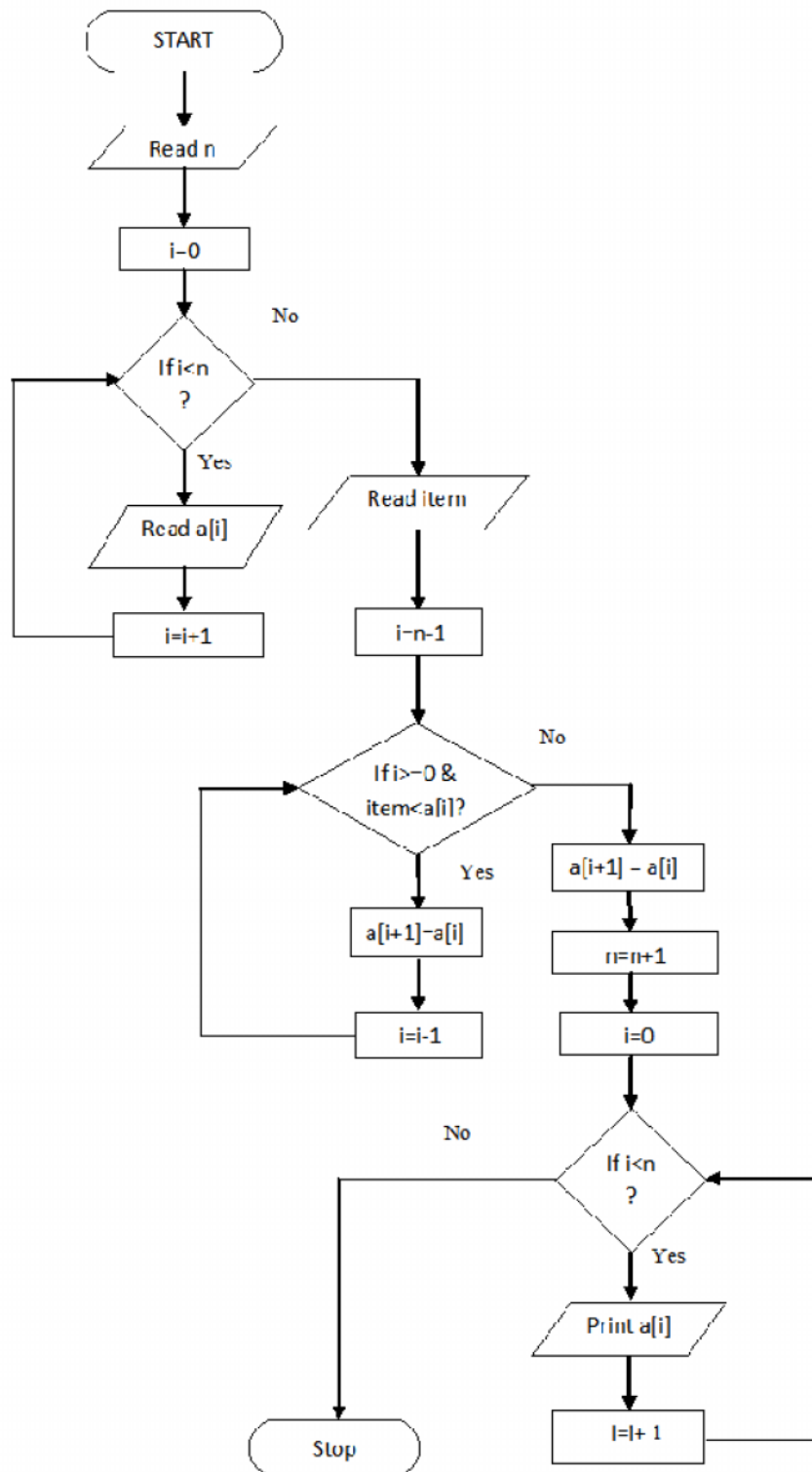
4.Tower of Hanoi

Tower of Hanoi, is a mathematical puzzle which consists of three towers (pegs) and more than one rings.

Tower of Hanoi is one of the best example for recursive problem solving.

Pre-condition:

These rings are of different sizes and stacked upon in an ascending order, i.e. the smaller one sits over the larger one. There are other variations of the puzzle where the number of disks increase, but the tower count remains the same.
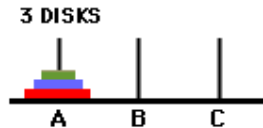
Post-condition:

All the disk should be moved to the last pole and placed only in ascending order as shown below.

Rules

The mission is to move all the disks to some another tower without violating the sequence of arrangement. A few rules to be followed for Tower of Hanoi are

- Only one disk can be moved among the towers at any given time.
- Only the "top" disk can be removed.
- No large disk can sit over a small disk.

Tower of Hanoi puzzle with n disks can be solved in minimum 2^n-1 steps. This presentation shows that a puzzle with 3 disks has taken $2^3 - 1 = 7$ steps.

Algorithm

To write an algorithm for Tower of Hanoi, first we need to learn how to solve this problem with lesser amount of disks, say   1 or 2. We mark three towers with name, **source**, **aux** (only to help moving the disks) and **destination**.

Input: one disk

If we have only one disk, then it can easily be moved from source to destination peg.

Input: two disks

If we have 2 disks –

- First, we move the smaller (top) disk to aux peg.
- Then, we move the larger (bottom) disk to destination peg.
- And finally, we move the smaller disk from aux to destination peg.

Input: more than two disks

- So now, we are in a position to design an algorithm for Tower of Hanoi with more than two disks. We divide the stack of disks in two parts. The largest disk (n[th] disk) is in one part and all other (n-1) disks are in the second part.

- Our ultimate aim is to move disk **n** from source to destination and then put all other (n1) disks onto it. We can imagine to apply the same in a recursive way for all given set of disks.
- The steps to follow are –

      Step 1 – Move n-1 disks from source to aux

      Step 2 – Move nth disk from source to dest

      Step 3 – Move n-1 disks from aux to dest
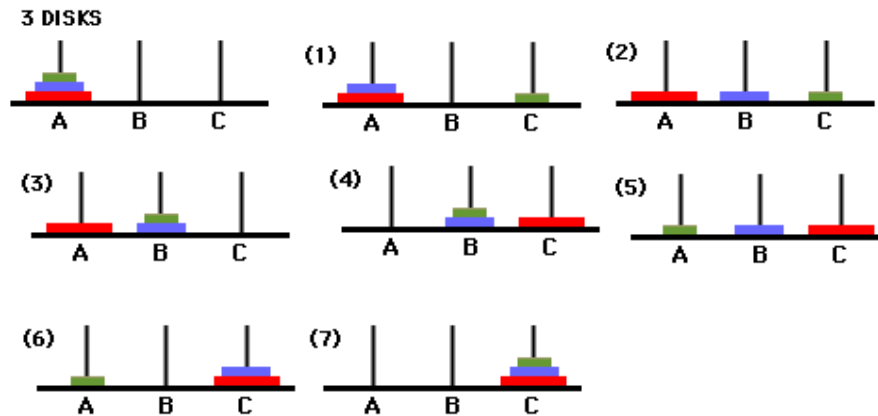
A recursive algorithm for Tower of Hanoi can be driven as follows –

```
START
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
STOP
```

FLOW CHART

```
                    ┌─────────────┐
                    │    Start    │
                    └─────────────┘
                           │
                           ▼
                 ╱─────────────────────╱
                ╱  Enter disk i.e number ╱
               ╱   of disks             ╱
              ╱─────────────────────╱
                           │
                           ▼
              ┌──┬─────────────────┬──┐
              │  │  Call the function │  │
              │  │ Hanoi(n,A,C,B) ⊗ │  │
              └──┴─────────────────┴──┘
                           │
                           ▼
                    ┌─────────────┐
                    │    Stop     │
                    └─────────────┘
```

```
                          ⊗
                           │
                           ▼
                    ◇─────────────◇
                   ╱   If disk==1?  ╲
                    ◇─────────────◇
                           │
                           ▼
                 ╱─────────────────────╱
                ╱  Print move disk from  ╱
               ╱   A to C               ╱
              ╱─────────────────────╱
                           │
                           ▼
              ┌──┬─────────────────┬──┐
              │  │ Call function Hanoi with │  │
              │  │ disk-1,A,B,C      │  │
              └──┴─────────────────┴──┘
                           │
                           ▼
                 ╱─────────────────────╱
                ╱  Print  move  disk    ╱
               ╱   from A to C          ╱
              ╱─────────────────────╱
                           │
                           ▼
              ┌──┬─────────────────┬──┐
              │  │ Call  function  Hanoi │  │
              │  │ with disk-1,C,A,B  │  │
              └──┴─────────────────┴──┘
                           │
                           ▼
                    ┌─────────────┐
                    │   Return    │
                    └─────────────┘
```
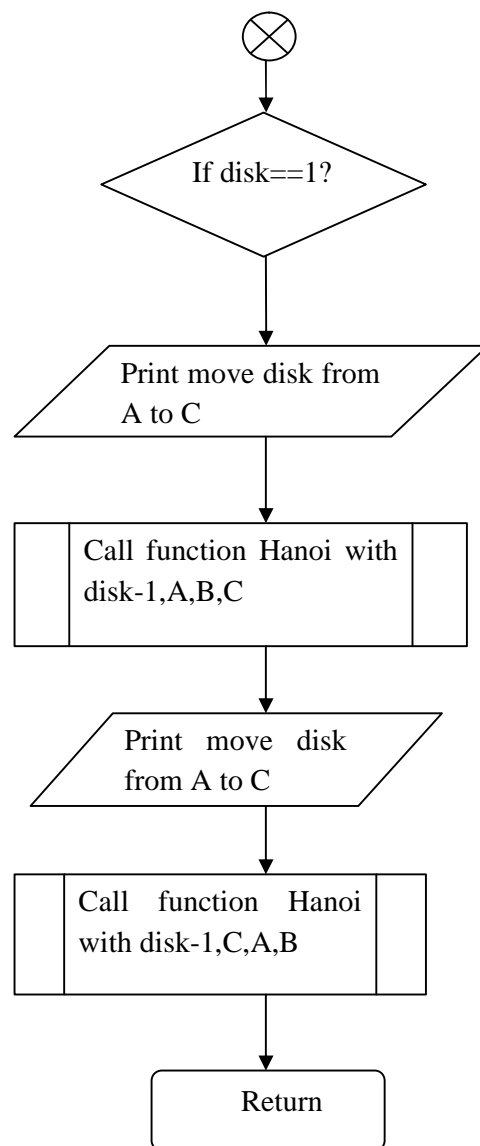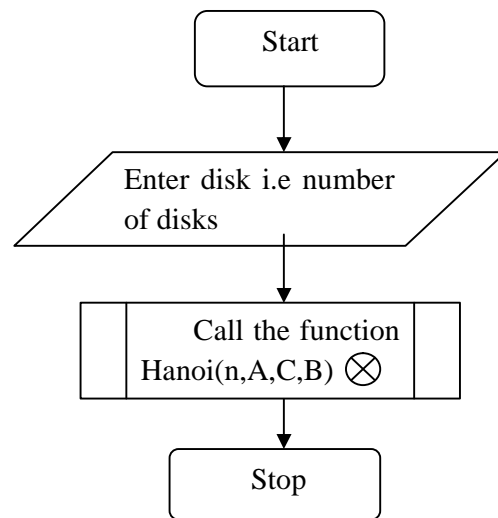
5. Draw a flow chart to find greatest among three numbers.(AU 2018)
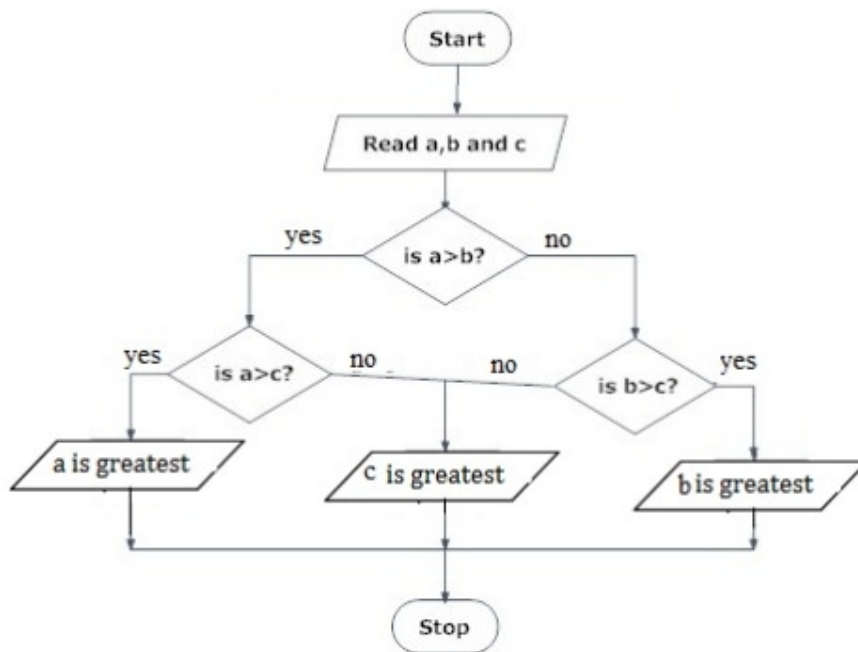
6. Draw a flow chart to find sum of n numbers(AU 2018)
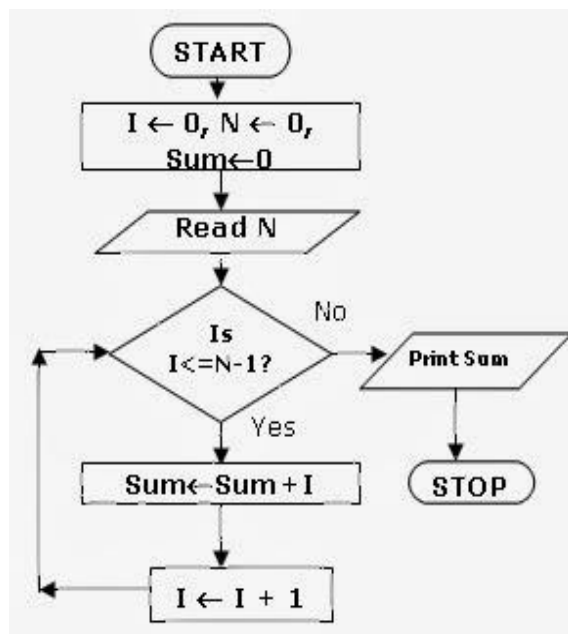
2 MARKS

1. What is an algorithm?

   An algorithm is a finite number of clearly described, unambiguous do able steps that can be systematically followed to produce a desired results for given input in the given amount of time. In other word, an algorithm is a **step by step procedure to solve a problem** with finite number of steps.

2. What is Pseudo code?

   Pseudocode is an **informal high-level description** of the operating principle of a **computer program** or **algorithm**. **Pseudo means false** and **code** refers to **instructions written in programming language**.

3. What is Problem Solving?

   Problem solving is the systematic approach to define the problem and creating number of solutions. The problem solving process starts with the problem specifications and ends with a correct program.

4. Distinguish between algorithm and program.

	Algorithm	Program
1.	**Systematic logical approach** which is a well-defined, step-by-step procedure that allows a computer to solve a problem.	It is exact code written for problem **following all the rules of the programming language.**
2.	An algorithm is a finite number of clearly described, unambiguous do able steps that can be systematically followed to produce a desired results for given input in the given amount of time.	The program will accept the data to perform computation. **Program=Algorithm + Data**

5. Define Flow chart.

   A **graphical representation** of an algorithm. Flow charts is a diagram made up of boxes, diamonds, and other shapes, connected by arrows.

6. Write an algorithm to accept two numbers, compute the sum and print the result.
   Step 1: Start
   Step 2: Declare variables num1,num2 and sum,
   Step 3: Read values num 1 and num2.
   Step 4: Add and assign the result to sum.
      Sum←num1+num2
   Step 5: Display sum

7. Differentiate between iteration and recursion.

S.No	Iteration	Recursion
1.	Iteration is a process of executing certain set of instructions repeatedly, without calling the self function.	Iteration is a process of executing certain set of instructions repeatedly, by calling the self function repeatedly.
2.	Iterative methods are more efficient because of better execution speed.	Recursive methods are less efficient.
3.	It is simple to implement.	Recursive methods are complex to implement.

8. What is Programming language? With example.

   Programming Language is a formal language with set of instruction, to the computer to solve a problem. Java, C, C++, Python, PHP.

9. What are the steps for developing algorithms.
   • Problem definition
   • Development of a model
   • Specification of Algorithm
   • Designing an Algorithm
   • Checking the correctness of Algorithm
   • Analysis of Algorithm
   • Implementation of Algorithm
   • Program testing
   • Documentation Preparation

10. What are the Guidelines for writing pseudo code?
   • Write one statement per line
   • Capitalize initial keyword
   • Indent to hierarchy
   • End multiline structure
   • Keep statements language independent.

11.Draw a flow chart to find whether the given number is leap year or not.

```
                    ┌─────────────┐
                    │    Start    │
                    └─────────────┘
                           │
                           ▼
                    ╱─────────────╱
                   ╱  Read year  ╱
                  ╱─────────────╱
                           │
                           ▼
                    ╱◆───────────◆╲
                   ◆      If       ◆
        ┌──────────◆  (year%4==0)  ◆──────────┐
        │           ◆───────────◆             │
        │                                     │
        │                                     │
        ▼                                     ▼
  ╱──────────────────╱              ╱──────────────────────╱
 ╱ Print 'Leap year' ╱             ╱ Print 'Not Leap year' ╱
╱──────────────────╱              ╱──────────────────────╱
        │                                     │
        │         ┌─────────────┐             │
        └────────▶│    Stop     │◀────────────┘
                  └─────────────┘
```
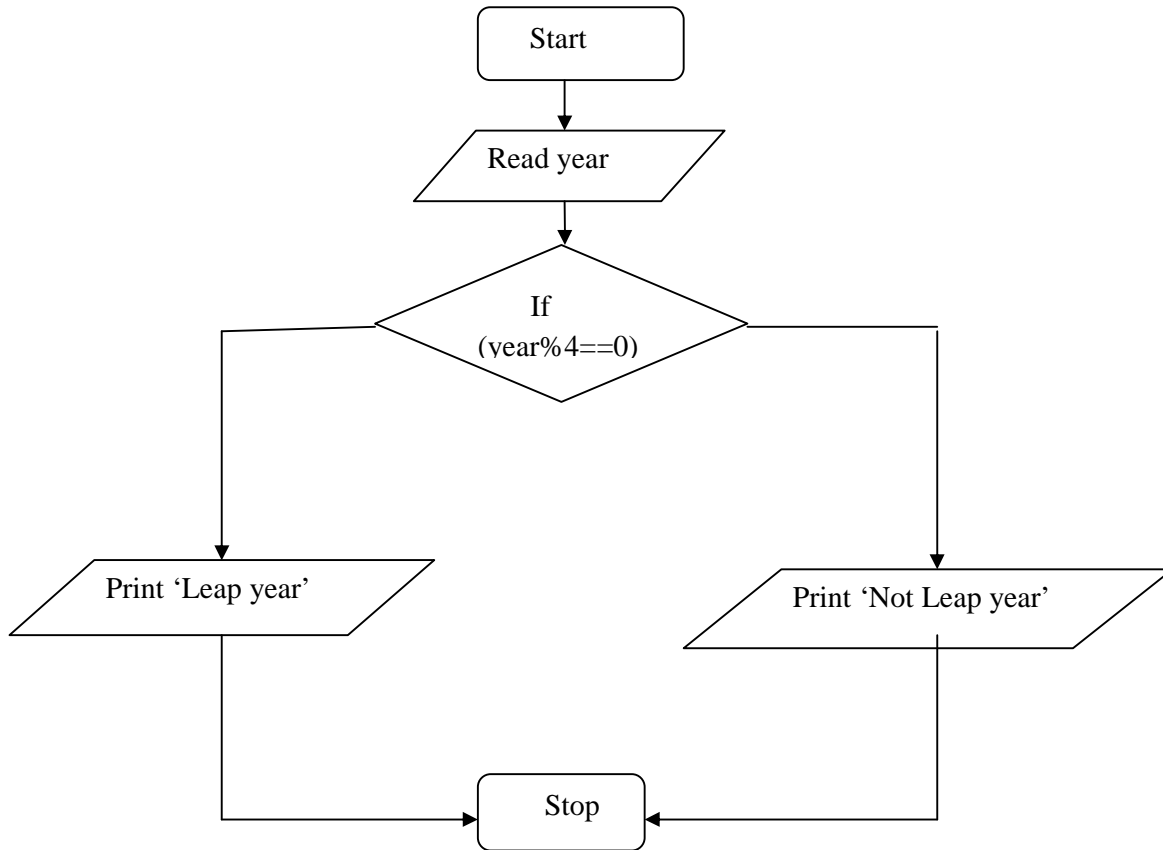
UNIT II
DATA, EXPRESSIONS, STATEMENTS

Python interpreter and interactive mode; values and types: int, float, boolean, string, and list; variables, expressions, statements, tuple assignment, precedence of operators, comments; Modules and functions, function definition and use, flow of execution, parameters and arguments; Illustrative programs: exchange the values of two variables, circulate the values of n variables, distance between two points.

1. **INTRODUCTION TO PYTHON:**

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language.
It was created by Guido van Rossum during 1985- 1990.
Python got its name from "Monty Python's flying circus". Python was released in the year 2000.

- **Python is interpreted**: Python is processed at runtime by the interpreter. You do not need to compile your program before executing it.
- **Python is Interactive**: You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented**: Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Python is a Beginner's Language:** Python is a great language for the beginner-Level programmers and supports the development of a wide range of applications.

Python Features:
- **Easy-to-learn:** Python is clearly defined and easily readable. The structure of the program is very simple. It uses few keywords.
- **Easy-to-maintain:** Python's source code is fairly easy-to-maintain.
- **Portable:** Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Interpreted:** Python is processed at runtime by the interpreter. So, there is no need to compile a program before executing it. You can simply run the program.
- **Extensible:** Programmers can embed python within their C,C++,JavaScript , ActiveX, etc.
- **Free and Open Source:** Anyone can freely distribute it, read the source code, and edit it.
- **High Level Language:** When writing programs, programmers concentrate on solutions of the current problem, no need to worry about the low level details.
- **Scalable:** Python provides a better structure and support for large programs than shell scripting.

Applications:
- ❖ Bit Torrent file sharing
- ❖ Google search engine, YouTube
- ❖ Intel, Cisco, HP,IBM
- ❖ i–Robot
- ❖ NASA
- ❖ Face book, Drop box

Python interpreter:

Interpreter: To execute a program in a high-level language by <u>translating it one line ata time.</u>
Compiler: To translate a program written in a high-level language into a low-level language <u>all at once</u>, in preparation for later execution.

Compiler	Interpreter
Compiler Takes **Entire** program as input	Interpreter Takes **Single** instruction as input
Intermediate Object Code is **Generated**	**No** Intermediate is Object Code **Generated**
Conditional Control Statements are Executes **faster**	Conditional Control Statements are Executes **slower**
Memory Requirement is **More**(Since Object Code is Generated)	**Memory Requirement** is **Less**
Program need not be **compiled** every time	Every time higher level program is converted into lower level program
Errors are displayed after **entire program** is checked	**Errors** are displayed for **every instruction** interpreted (if any)
Example : C Compiler	**Example** : PYTHON

Modes of python interpreter:
       Python Interpreter is a program that reads and executes Python code. It uses 2 modes of Execution.
1. Interactive mode
2. Script mode

Interactive mode:
❖ Interactive Mode, as the name suggests, allows us to interact with OS.
❖ When we type Python   statement, **<u>interpreter displays the result(s) immediately.</u>**

Advantages:
❖ Python, in interactive mode, is good enough to learn, experiment or explore.
❖ Working in interactive mode is convenient for beginners and for <u>testing small pieces of code.</u>
   Drawback:
❖ We <u>cannot save the statements</u> and have to retype all the statements once again to re-run them.

In interactive mode, you type Python programs and the interpreter displays the result:

>>> 1 + 1
2

The chevron, >>>, <u>is the prompt the interpreter uses</u> to indicate that it is ready for you to enter code. If you type 1 + 1, the interpreter replies 2.
>>> print ('Hello, World!')
Hello, World!

This is an example of a print statement. It displays a result on the screen. In this case, the result is the words.

Script mode:

- ☐ In script mode, we type python program in a file and then use interpreter to execute the content of the file.
- ☐ Scripts can be saved to disk for future use. **Python scripts have the extension .py**, meaning that the filename ends with**.py**
- ☐ Save the code with **filename.py** and run the interpreter in script mode to execute the script.

Example1:

```
print(1)
x = 2
print(x)
```

Output:
>>>1
2

Interactive mode	Script mode
A way of using the Python interpreter by typing commands and expressions at the prompt.	A way of using the Python interpreter to read and execute statements in a script.
Can't save and edit the code	Can save and edit the code
If we want to experiment with the code, we can use interactive mode.	If we are very clear about the code, we can use script mode.
we cannot save the statements for further use and we have to retype all the statements to re-run them.	we can save the statements for further use and we no need to retype all the statements to re-run them.
We can see the results immediately.	We can't see the code immediately.

Integrated Development Learning Environment(IDLE):

- ☐ Is a **graphical user interface** which is completely written in Python.
- ☐ It is bundled with the default implementation of the python language and also comes with optional part of the Python packaging.

Features of IDLE:

- ☐ Multi-window text editor with syntax highlighting.

☐ Auto completion with **smart indentation**.

☐ **Python shell** to display output with syntax highlighting.

2.VALUES AND DATATYPES

Value:

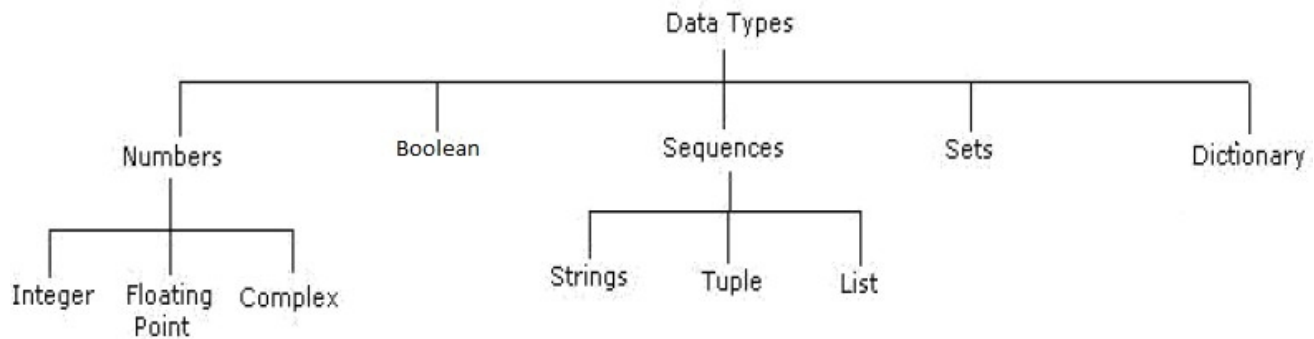**Value** can be any letter, number or string.

Eg, Values are 2, 42.0, and 'Hello, World!'. (These values belong to <u>different datatypes.)</u>

Data type:

Every value in Python has a data type.

It is a set of values, and the allowable operations on those values.

Python has four standard data types:

Numbers:

❖ Number data type stores **Numerical Values.**

❖ This data type is immutable [i.e. values/items cannot be changed].

❖ Python supports integers, floating point numbers and complex numbers. They are defined as,

Integers	Long	Float	Complex
- They are often called just integers or **int**. - They are positive or negative whole numbers with no decimal point.	-They are long integers. -They can also be represented in octal and hexadecimal representation.	-They are written with a decimal point dividing the integer and the fractional parts.	-They are of the form **a + bj**, where a and b are floats and j represents the square root of -1 (which is an imaginary number). -The real part of the number is a, and the imaginary part is b.
Eg, 56	**Eg, 5692431L**	**Eg, 56.778**	**Eg, square root of-1 is a complex number**

Sequence:

❖ A sequence is an **ordered collection of items**, indexed by positive integers.

❖ It is a combination **of mutable** (value can be changed) **and immutable** (values cannot be changed) datatypes.

❖ There are three types of sequence data type available in Python, they are

1. **Strings**
2. **Lists**
3. **Tuples**

Strings:

➢ A String in Python consists of a <u>series or sequence of characters - letters, numbers, and special characters.</u>

➢ Strings are marked by quotes:

- Single quotes('   ')   E.g., 'This a string in single quotes'
- double quotes(" ")   E.g., '"This a string in double quotes'"
- triple quotes(""" """ """)E.g., """This is a paragraph. It is made up of multiple lines and sentences."""

➢ Individual character in a string is accessed using a subscript(index).

➢ Characters can be accessed using indexing and slicing operations .Strings are

Immutable i.e the contents of the string cannot be changed after it is created.

Indexing:

String A	H	E	L	L	O
Positive Index	0	1	2	3	4
Negative Index	-5	-4	-3	-2	-1

- Positive indexing helps in accessing the string from the beginning
- Negative subscript helps in accessing the string from the end.
- Subscript 0 or –ven(where n is length of the string) displays the first element.
  Example: A[0] or A[-5] will display "H"
- Subscript 1 or –ve (n-1) displays the second element.
  Example: A[1] or A[-4] will display "E"

Operations on string:

i. Indexing
ii. Slicing
iii. Concatenation
iv. Repetitions
v. Membership

Creating a string	>>> s="good morning"	Creating the list with elements of different data types.
Indexing	>>>**print(s[2])** o >>>**print(s[6])** O	❖ Accessing the item in the position0 ❖ Accessing the item in the position2
Slicing(ending position -1)	>>>**print(s[2:])** od morning	- Displaying items from 2ndtill last.

Slice operator is used to extract part of a data type	>>>**print(s[:4])** Good	- Displaying items from 1st position till 3rd.
Concatenation	>>>**print(s+"friends")** good morning friends	-Adding and printing the characters of two strings.
Repetition	>>>**print(s*2)** good morning good morning	Creates new strings, concatenating multiple copies of the same string
in, not in (membership operator)	>>> s="good morning" **>>>"m" in s True** **>>> "a" not in s** **True**	Using membership operators to check a particular character is in string or not. Returns true if present.

Lists

- ❖ List is an ordered sequence of items. Values in the list are called elements /items.
- ❖ It can be written as a list of comma-separated items (values) between **square brackets[]**.
- ❖ Items in the lists can be of different datatypes.

Operations on list:

      Indexing
       Slicing
       Concatenation
       Repetitions
       Updation, Insertion, Deletion

Creating a list	>>>**list1=["python", 7.79, 101, "hello"]** >>>**list2=["god",6.78,9]**	Creating the list with elements of different data types.
Indexing	>>>**print(list1[0])** python >>>**list1[2]** 101	❖ Accessing the item in the position0 ❖ Accessing the item in the position2
Slicing(ending position -1) Slice operator is used to extract part of a string, or some part of a list Python	>>>**print(list1[1:3])** [7.79, 101] >>>**print(list1[1:])** [7.79, 101, 'hello']	- Displaying items from 1st till2nd. - Displaying items from 1st position till last.
Concatenation	>>>**print(list1+list2)** ['python', 7.79, 101, 'hello', 'god',	-Adding and printing the items of two lists.

	6.78, 9]	
Repetition	>>>**list2*3** **['god', 6.78, 9, 'god', 6.78, 9, 'god',** **6.78, 9]**	Creates new strings, concatenating multiple copies of the same string
Updating the list	>>>**list1[2]=45** >>>print(list1) ['python', 7.79, 45, 'hello']	Updating the list using index value
Inserting an element	>>>**list1.insert(2,"program")** >>> print(list1) ['python', 7.79, 'program', 45, 'hello']	Inserting an element in 2nd position
Removing an element	>>>**list1.remove(45)** >>> print(list1) ['python', 7.79, 'program', 'hello']	Removing an element by giving the element directly

Tuple:

❖ A tuple is same as list, except that the set of elements is **enclosed in parentheses** instead of square brackets.

❖ **A tuple is an immutable list.** i.e. once a tuple has been created, you can't add elements to a tuple or remove elements from the tuple.

❖ Benefit of Tuple:

❖ Tuples are faster than lists.

❖ If the user wants to protect the data from accidental changes, tuple can be used.

❖ Tuples can be used as keys in dictionaries, while lists can't.

Basic Operations:

Creating a tuple	>>>**t=("python", 7.79, 101, "hello")**	Creating the tuple with elements of different data types.
Indexing	>>>**print(t[0])** python >>>**t[2]** 101	❖ Accessing the item in the position0 ❖ Accessing the item in the position2
Slicing(ending position -1)	>>>**print(t[1:3])** (7.79, 101)	❖ Displaying items from1st till2nd.
Concatenation	>>>**t+("ram", 67)** ('python', 7.79, 101, 'hello', 'ram', 67)	❖ Adding tuple elements at the end of another tuple elements
Repetition	>>>**print(t*2)** ('python', 7.79, 101, 'hello', 'python', 7.79, 101, 'hello')	❖ Creates new strings, concatenating multiple copies of the same string

Altering the tuple data type leads to error. Following error occurs when user tries to do.

```
>>>t[0]="a"
Trace back (most recent call last):
 File "<stdin>", line 1, in <module>
Type Error: 'tuple' object does not support item assignment
```

Mapping

-This data type is unordered and mutable.

-Dictionaries fall under Mappings.

Dictionaries:

❖ Lists are ordered sets of objects, whereas **dictionaries are unorderedsets**.

❖ Dictionary is created by using **curly brackets**. i,e.{ }

❖ Dictionaries **are accessed via keys** and not via their position.

❖ 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**(The association of a key and a value is called a key- value pair)

Dictionaries don't support the sequence operation of the sequence data types like strings, tuples and lists.

Creating a dictionary	**>>> food = {"ham":"yes", "egg" : "yes", "rate":450 }** >>>print(food) {'rate': 450, 'egg': 'yes', 'ham': 'yes'}	Creating the dictionary with elements of different data types.
Indexing	>>>>**print(food["rate"])** 450	Accessing the item with keys.
Slicing(ending position -1)	>>>**print(t[1:3])** (7.79, 101)	Displaying items from 1st till 2nd.

If you try to access a key which doesn't exist, you will get an error message:

```
>>>words = {"house" : "Haus", "cat":"Katze"}
>>>words["car"]
Traceback (most recent call last): File
"<stdin>", line 1, in <module>KeyError: 'car'
```

Data type	Compile time	Run time
int	a=10	a=int(input("enter a"))
float	a=10.5	a=float(input("enter a"))
string	a="panimalar"	a=input("enter a string")
list	a=[20,30,40,50]	a=list(input("enter a list"))
tuple	a=(20,30,40,50)	a=tuple(input("enter a tuple"))

3.Variables,Keywords Expressions, Statements, Comments, Docstring ,Lines And Indentation, Quotation In Python, Tuple Assignment:

VARIABLES:

- ❖ A variable allows us to store a value by assigning it to a name, which can be used later.
- ❖ Named memory locations to store values.
- ❖ Programmers generally choose names for their variables that are meaningful.
- ❖ It can be of any length. No space is allowed.
- ❖ We don't need to declare a variable before using it. In Python, we simply assign a value to a variable and it will exist.

Assigning value to variable:
Value should be given on the right side of assignment operator(=) and variable on left side.

```
>>>counter =45
print (counter)
```

Assigning a single value to several variables simultaneously:

```
>>> a=b=c=100
```
Assigning multiple values to multiple variables:

```
>>>a,b,c=2,4,"ram"
```

KEYWORDS:

- ❖ Keywords are the reserved words in Python.
- ❖ We cannot use a keyword as name, function name or any other identifier.
- ❖ They are used to define the syntax and structure of the Python language.
- ❖ Keywords are case sensitive.

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	

IDENTIFIERS:

Identifier is the name given to entities like class, functions, variables etc. in Python.

- ❖ Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore (_).

- ❖ all are valid example.
- ❖ An identifier cannot start with a digit.
- ❖ Keywords cannot be used as identifiers.
- ❖ Cannot use special symbols like!, @, #, $, % etc. in our identifier.
- ❖ Identifier can be of any length.

Example:

Names like myClass, var_1, and **this_is_a_long_variabl**e

Valid declarations	Invalid declarations
Num	Number 1
Num	num1
Num1	addition of program
_NUM	1Num
NUM_temp2	Num.no
IF	if
Else	else

STATEMENTS AND EXPRESSIONS:

Statements:

      -Instructions that a Python interpreter can executes are called statements.

    -A statement is a unit of code like creating a variable or displaying avalue.

```
>>> n = 17
>>>print (n)
```

  Here, The first line is an assignment statement that gives a value to n. The second line is a print statement that displays the value of n.

Expressions:

     -An expression is a **combination of values, variables, and operators.**

   - A value all by itself is considered an expression, and also a variable.

   - So the following are all legal expressions:

```
>>> 42
42
>>> a=2
>>>a+3+2 7
>>> z=("hi"+"friend")
>>>print(z) hifriend
```

INPUT AND OUTPUT

INPUT: Input is data entered by user (end user) in the program. In python, **input
() function** is available for input.

Syntax for input() is:
variable = input ("data")

Example:

```
>>> x=input("enter the name:")
enter the name: george
```

```
>>>y=int(input("enter the number"))
enter the number 3
```

#python accepts string as default data type. Conversion is required for type.

OUTPUT: Output can be displayed to the user using Print statement .

Syntax:
print (expression/constant/variable)

Example:

```
>>> print ("Hello")
Hello
```

COMMENTS:

- ☐ A **hash sign (#)** is the beginning of a comment.
- ☐ Anything written after # in a line is ignored by interpreter.

  Eg: percentage = (minute * 100)/60   **# calculating percentage of an hour**

- ☐ Python **does not have multiple-line commenting feature.** You have to comment each line individually as follows:

Example:

```
# This is a comment.
# This is a comment, too.
 # I said that already.
```

DOCSTRING:

- ☐ Docstring is short for documentation string.
- ☐ It is a string that occurs as the first statement in a module, function, class, or method definition. We must write what a function/class does in the docstring.
- ☐ **Triple quotes** are used while writing docstrings.

Syntax:
functionname__doc.__ **Example:**

```
def double(num):
    """Function to double thevalue"""
    return2*num
>>>print (double.__doc__)
Function to double the value
```

LINES AND INDENTATION:

- ☐ Most of the programming languages like C, C++, Java use braces { } to define a block of code. But, python uses indentation.
- ☐ Blocks of code are denoted by line indentation.
- ☐ It is a space given to the block of codes for class and function definitions or flow control.

Example:

```
a=3
b=1
if a>b:
    print("a is greater")
else:
    print("b is greater")
```

QUOTATION INPYTHON:

Python accepts single ('), double (") and triple (''' or """) quotes to denote string literals. <u>Anything that is represented using quotations are considered as string.</u>

- ☐ Single quotes('   ')   Eg, 'This a string in single quotes'
- ☐ double quotes(" ")   Eg, "'This a string in double quotes'"
- ☐ triple quotes("""   """)   Eg, This is a paragraph. It is made up of multiple lines and sentences."""

TUPLE ASSIGNMENT

- ☐ An assignment to all of the elements in a tuple using a single assignment statement.
- ☐ Python has a very powerful **tuple assignment** feature that allows a tuple of variables on the left of an assignment to be assigned values from a tuple on the right of the assignment.
- ☐ The left side is a tuple of variables; the right side is a tuple of values.
- ☐ Each value is assigned to its respective variable.
- ☐ All the expressions on the right side are evaluated before any of the assignments. This feature makes tuple assignment quite versatile.
- ☐ Naturally, the number of variables on the left and the number of values on the right have to be the same.

```
>>>(a, b, c, d) = (1, 2, 3)
ValueError: need more than 3 values to unpack
```

Example:

-It is useful to swap the values of two variables. With **conventional assignment statements**, we have to use a temporary variable. For example, to swap a and b:

Swap two numbers	Output:
a=2;b=3 print(a,b) temp = a a = b b = temp print(a,b)	(2, 3) (3, 2) >>>

-Tuple assignment solves this problem neatly:

```
(a, b) = (b, a)
```

-**One way to think of tuple assignment is as <u>tuple packing/unpacking.</u>**
In tuple packing, the values on the left are 'packed' together in a tuple:

```
>>>b = ("George",25,"20000")      # tuplepacking
```

-In tuple unpacking, **<u>the values in a tuple on the right are 'unpacked 'into the variables/names on the right:</u>**

```
>>>b = ("George", 25, "20000")      # tuple packing
>>>(name, age, salary)=b      # tupleunpacking
>>>name
'George'
>>>age
25
>>>salary
'20000'
```

-The right side can be any kind of sequence (string, list,tuple)
<u>Example:</u>
-To split an email address in to user name and a domain

```
>>>mailid='god@abc.org'
>>>name,domain=mailid.split('@')
>>>print name god
>>> print (domain) abc.org
```

4.OPERATORS:

- ☐ Operators are the constructs which can manipulate the value of operands.
- ☐ Consider the **expression *4 + 5 = 9*.** Here, **4 and 5 are called operands** and <u>**+ is called operator**</u>
- ☐ Types of Operators:
  -Python language supports the following types of operators
  - Arithmetic Operators
  - Comparison (Relational)Operators
  - Assignment Operators
  - Logical Operators
  - Bitwise Operators
  - Membership Operators
  - Identity Operators

Arithmetic operators:

They are used to perform **mathematical operations** like addition, subtraction, multiplication etc.
Assume, a=10 and b=5

Operator	Description	Example
+ Addition	Adds values on either side of the operator.	a + b = 30
- Subtraction	Subtracts right hand operand from left hand operand.	a – b = -10
* Multiplication	Multiplies values on either side of the operator	a * b = 200
/ Division	Divides left hand operand by right hand operand	b / a = 2
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 0
** Exponent	Performs exponential (power) calculation on operators	a**b =10 to the power 20
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed	5//2=2

Examples
```
a=10
b=5
print("a+b=",a+b)
print("a-b=",a-b)
print("a*b=",a*b)
print("a/b=",a/b)
print("a%b=",a%b)
print("a//b=",a//b)
print("a**b=",a**b)
```

Output:
```
a+b=15
a-b= 5
a*b= 50
a/b= 2.0
a%b=0
a//b=2
a**b= 100000
```

Comparison (Relational)Operators:

- Comparison operators are used to compare values.
- It either returns True or False according to the condition. **Assume, a=10 and b=5**

Operator	Description	Example
==	If the values of two operands are equal, then the condition	(a == b) is

	becomes true.	not true.
!=	If values of two operands are not equal, then condition becomes true.	(a!=b) is true
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

Example

a=10
b=5
print("a>b=>",a>b)
print("a>b=>",a<b)
print("a==b=>",a==b)
print("a!=b=>",a!=b)
print("a>=b=>",a<=b)
print("a>=b=>",a>=b)

Output: a>b=>
True a>b=>
False a==b=>
False a!=b=>
True a>=b=>
False a>=b=>
True

Assignment Operators:

-Assignment operators are used in Python to assign values to variables.

Operator	Description	Example
=	Assigns values from right side operands to left side operand	c = a + b assigns value of a + b into c
+= Add AND	It adds right operand to the left operand and assign the result to leftoperand	c += a is equivalent to c = c + a
-= Subtract AND	It subtracts right operand from the left operand and assign the result to left operand	c -= a is equivalent to c = c -a

Operator	Description	Example
*= Multiply AND	It multiplies right operand with the left operand and assign the result to left operand	c *= a is equivalent to c = c *a
/= Divide AND	It divides left operand with the right operand and assign the result to left operand	c /= a is equivalent to c = c /ac /= a is equivalent to c = c /a
%= Modulus AND	It takes modulus using two operands and assign the result to left operand	c %= a is equivalent to c = c % a
**= Exponent AND	Performs exponential (power) calculation on operators and assign value to the left operand	c **= a is equivalent to c = c ** a
//= Floor Division	It performs floor division on operators and assign value to the left operand	c //= a is equivalent to c = c // a

Example

a =21
b =10
c = 0
c = a + b
print("Line 1 - Value of c is ",c)
c += a
print("Line 2 - Value of c is ", c)
c *= a
print("Line 3 - Value of c is ",c)
c /= a
print("Line 4 - Value of c is ", c)
c = 2
c %=a
print("Line 5 - Value of c is ",c)
c **= a
print("Line 6 - Value of c is ",c)
c //= a
print ("Line 7 - Value of c is ", c)

Output

Line 1 - Value of c is 31
Line 2 - Value of c is 52
Line 3 - Value of c is 1092
Line 4 - Value of c is 52.0
Line 5 - Value of c is2
Line 6 - Value of c is 2097152
Line 7 - Value of c is99864

Logical Operators:

-Logical operators are the and, or, not operators.

Operator	Meaning	Example
and	True if both the operands are true	x and y
or	True if either of the operands is true	x or y
not	True if operand is false (complements the operand)	not x

Example
a = True
b = False
print('a and b is', a and b)
print('a or b is' ,a or b)
print('not a is', not a)

Output
x and y is False
x or y is True
not x is False

Bitwise Operators:

- A **bitwise operation** operates on one or more **bit** patterns at the level of individual bits

Example: Let x = 10 (0000 1010 in binary)and
y = 4 (0000 0100 in binary)

Operator	Meaning	Example
&	Bitwise AND	x& y = 0 (0000 0000)
|	Bitwise OR	x | y = 14 (0000 1110)
~	Bitwise NOT	~x = -11 (1111 0101)
^	Bitwise XOR	x ^ y = 14 (0000 1110)
>>	Bitwise right shift	x>> 2 = 2 (0000 0010)
<<	Bitwise left shift	x<< 2 = 40 (0010 1000)

Example
a = 60   # 60 = 0011 1100
b = 13   # 13 = 0000 1101
c = 0
c = a & b;   # 12 = 0000 1100
print "Line 1 - Value of c is ", c
c = a|b;   # 61 = 00111101
print "Line 2 - Value of c is ", c
c = a^b;   # 49 = 00110001
print "Line 3 - Value of c is ", c
c =~a;   # -61 = 11000011

Output
Line 1 - Value of c is 12
Line 2 - Value of c is 61
Line 3 - Value of c is 49
Line 4 - Value of c is-61
Line 5 - Value of c is 240
Line 6 - Value of c is 15

print "Line 4 - Value of c is ", c
c = a<<2;   # 240 = 11110000
print "Line 5 - Value of c is ", c
c = a>>2;   # 15 = 00001111
print "Line 6 - Value of c is ", c

Membership Operators:

❖ Evaluates to find a value or a variable is in the specified sequence of string, list, tuple, dictionary or not.
❖ Let, **x=[5,3,6,4,1].** To check particular item in list or not, **in and not in** operators areused.

Operator	Meaning	Example
in	True if value/variable is found in the sequence	5 in x
not in	True if value/variable is not found in the sequence	5 not in x

Example:
x=[5,3,6,4,1]
>>>**5 in x**
True
>>>**5 not in x**
False

Identity Operators:
• They are used to check if two values (or variables) are located on the same partof the memory.

Operator	Meaning	Example
is	True if the operands are identical (refer to the same object)	x is True
is not	True if the operands are not identical (do not refer to the same object)	x is not True

Example
x =5
y =5
x2 = 'Hello'
y2= 'Hello'
print(x1 is not y1)
print(x2 is y2)

Output
False
True

48

5.OPERATOR PRECEDENCE:

When an expression contains **more than one operator, the order of evaluation**
depends on the order of operations.

Operator	Description
**	Exponentiation (raise to the power)
~ + -	Complement, unary plus and minus (method names for the last two are +@ and -@)
* / % //	Multiply, divide, modulo and floor division
+ -	Addition and subtraction
>><<	Right and left bitwise shift
&	Bitwise 'AND'
^ |	Bitwise exclusive `OR' and regular `OR'
<= <>>=	Comparison operators
<> == !=	Equality operators
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

-For mathematical operators, Python follows mathematical convention.
-The acronym **PEMDAS** (Parentheses, Exponentiation, Multiplication, Division, Addition, Subtraction) is a useful way to remember the rules:

- Parentheses have the highest precedence and can be used to force an expression to evaluate in the order you want. Since expressions in parentheses are evaluated first, 2 * (3-1)is 4, and (1+1)**(5-2) is8.
- You can also use parentheses to make an expression easier to read,asin(minute * 100) / 60, even if it doesn't change the result.
- Exponentiation has the next highest precedence, so 1 + 2**3 is 9, not 27, and2 *3**2 is 18, not 36.
- Multiplication and Division have higher precedence than Addition and Subtraction. So 2*3-1 is 5, not 4, and 6+4/2 is 8, not5.
- Operators with the same precedence are evaluated from left to right (except exponentiation).

Examples:

a=9-12/3+3*2-1 a=? a=9-4+3*2-1 a=9-4+6-1 a=5+6-1 a=11- 1 **a=10**	A=2*3+4%5-3/2+6 A=6+4%5-3/2+6 A=6+4-3/2+6 A=6+4- 1+6 A=10-1+6 A=9+6 **A=15**	find m=? m=-43||8&&0||-2 m=- 43||0||-2 m=1||-2 **m=1**

6.**Functions, Function Definition And Use, Function call, Flow Of Execution, Function Prototypes, Parameters And Arguments, Return statement, Arguments types, Modules**

-

FUNCTIONS:

> **Function is a sub program which consists of set of instructions used to perform a specific task. A large program is divided into basic building blocks called function.**

Need For Function:
- When the program is too complex and large they are divided into parts. Each part is separately coded and combined into single program. Each subprogram is called as function.
- Debugging, Testing and maintenance becomes easy when the program is divided into subprograms.
- Functions are used to avoid rewriting same code again and again in a program.
- Function provides code re-usability
- The length of the program is reduced.

Types of function:

Functions can be classified into two categories:
   i) user defined function
   ii) Built in function

i) Built in functions
- Built in functions are the functions that are <u>already created and stored</u> inpython.
- These built in functions are always available for usage and accessed by a programmer. It cannot be modified.

Built in function	Description
>>>**max(3,4)** 4	# returns largest element
>>>**min(3,4)** 3	# returns smallest element
>>>**len("hello")** 5	#returns length of an object
>>>**range(2,8,1)** [2, 3, 4, 5, 6, 7]	#returns range of given values
>>>**round(7.8)** 8.0	#returns rounded integer of the given number
>>>**chr(5)** \x05'	#returns a character (a string) from an integer
>>>**float(5)** 5.0	#returns float number from string or integer
>>>**int(5.0)** 5	# returns integer from string or float
>>>**pow(3,5)** 243	#returns power of given number
>>>**type(5.6)** <type 'float'>	#returns data type of object to which it belongs
>>>**t=tuple([4,6.0,7])** (4, 6.0, 7)	# to create tuple of items from list
>>>**print("good morning")** Good morning	# displays the given object
>>>**input("enter name:")** enter name : George	# reads and returns the given string

ii) User Defined Functions:
- User defined functions are the functions that programmers create for their requirement anduse.
- These functions can then be **combined to form module** which can be used in other programs by importing them.
- Advantages of user defined functions:
  - Programmers working on large project can divide the workload by making different functions.
  - If repeated code occurs in a program, function can be used to include those codes and execute when needed by calling that function.

Function definition: (Sub program)
- def keyword is used to define a function.
- Give the function name after def keyword followed by parentheses in which arguments are given.
- End with colon (:)
- Inside the function add the program statements to be executed
- End with or without return statement

Syntax:

def fun_name(Parameter1,Parameter2…Parameter n): statement1
 	
     statement2…
 	
     statement n return[expression]

Example:

def my_add(a,b):
    c=a+b
    return c

Function Calling: (Main Function)

➢ Once we have defined a function, we can call it from another function, program or even the Pythonprompt.

➢ To call a function we **simply type the function name with appropriate arguments.**

Example:

```
x=5
y=4
my_add(x,y)
```

Flow of Execution:

- The order in which statements are executed 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.
- Function definitions do not alter the flow of execution of the program, but remember that statements inside the function are not executed until the function is called.
- Function calls are like a bypass in the flow of execution. Instead of going to the next statement, the flow jumps to the first line of the called function, executes all the statements there, and then comes back to pick up where it left off.

Note: When you read a program, don't read from top to bottom. Instead, follow the flow of execution. This means that you will read the **def** statements as you are scanning from top to bottom, but you should skip the statements of the function definition until you reach a point where that function is called.

Function Prototypes:

i. Function without arguments and without return type
ii. Function with arguments and without return type
iii. Function without arguments and with return type
iv. Function with arguments and with return type

i) Function without arguments and without return type
- In this type no argument is passed through the function call and no output is return to main function
- The sub function will read the input values perform the operation and print the result in the same block

ii) Function with arguments and without return type
- Arguments are passed through the function call but output is not return to the main function

iii) Function without arguments and with return type
- In this type no argument is passed through the function call but output is return to the main function.

iv) Function with arguments and with return type
- In this type arguments are passed through the function call and output is return to the main function

Without Return Type	
Without argument	**With argument**

def add():
    a=int(input("enter   a"))
    b=int(input("enter   b"))
    c=a+b
    print(c)
add()
	def add(a,b):
	    c=a+b
	    print(c)
	a=int(input("enter a"))
	b=int(input("enter b"))
	 add(a,b)

OUTPUT: enter a5 enter b 10 15	OUTPUT: enter a5 enter b 10 15

With return type	
Without argument	**With argument**

def add():
    a=int(input("enter a"))
    b=int(input("enterb"))
    c=a+b
    return c
c=add()
print(c)
	def add(a,b):
	    c=a+b
	    return c
	a=int(input("enter a"))
	b=int(input("enter b"))
	c=add(a,b)
	print(c)

OUTPUT: enter a5 enter b 10 15	OUTPUT: enter a5 enter b 10 15

Parameters And Arguments:

Parameters:

- <u>Parameters are the value(s) provided in the parenthesis</u> when we write function header.
- These are the values required by function to work.
- If there is more than one value required, all of them will be listed in parameter list separated by **comma.**
- Example: defmy_add(a,b):

Arguments :

- Arguments are the value(s) provided in function call/invoke statement.
- List of arguments should be supplied in same way as parameters are listed.
- Bounding of parameters to arguments is done 1:1, and so there should be same number and type of arguments as mentioned in parameter list.
- Example:my_add(x,y)

RETURN STATEMENT:

- The **return statement is used to exit a function** and go back to the place from where it was called.
- If the return statement has no arguments, then it will not return any values. But exits from function.

Syntax:
return[expression]

Example:
```
def my_add(a,b):
   c=a+b
return c
x=5
y=4
print(my_add(x,y))
```
Output:
9

ARGUMENT TYPES:

1. Required Arguments
2. Keyword Arguments
3. Default Arguments
4. Variable length Arguments

Required Arguments : The number of arguments in the function call should match exactly with the function definition.

```
defmy_details( name, age ):
print("Name: ", name)
   print("Age ", age)
   return
my_details("george",56)
```

Output:

```
Name:
georgeAge56
```

Keyword Arguments:

Python interpreter is able to use the keywords provided to match the values with parameters even though if they are arranged in out of order.

```
def my_details( name, age ):
  print("Name: ", name)
  print("Age ", age)
  return
my_details(age=56,name="george")
```

Output:

```
Name:
georgeAge56
```

DefaultArguments:

Assumes a default value if a value is not provided in the function call for that argument.

```
defmy_details( name, age=40 ):
  print("Name: ", name)
  print("Age ", age) return
my_details(name="george")
```

Output:

```
Name:
georgeAge40
```

Variable lengthArguments

If we want to specify more arguments than specified while defining the function, variable length arguments are used. It is denoted by * symbol before parameter.

```
def my_details(*name ):
    print(*name)
my_details("rajan","rahul","micheal", ärjun")
```

Output:

```
rajanrahulmichealärjun
```

7.MODULES:

- ➢ **A module is a file containing Python definitions ,functions, statements and instructions.**
- ➢ Standard library of Python is extended as modules.
- ➢ **To use these modules in a program**, **programmer needs to import the module.**

➢ Once we import a module, we can reference or use to any of its functions or variables in our code.

- There is large number of standard modules also available in python.
- Standard modules can be imported the same way as we import our user- defined modules.
- Every module contains many functions.
- *To access one of the function , you have to specify the name of the module and the name of the function separated by dot .***This format is called dot notation.**

Syntax:

import module_namemodule_name.function_name(variable)

Importing Builtin Module:	**Importing User Defined Module:**
import math x=math.sqrt(25) print(x)	import calx=cal.add(5,4) print(x)

There are **four ways to import a module** in our program, they are

Import: It is simplest and most common way to use modules in our code. **Example:** import math x=math.pi print("The value of pi is", x) Output: The value of pi is 3.141592653589793	**from import** : It is used to get a specific function in the code instead of complete file. **Example:** from math import pi x=pi print("The value of pi is", x) Output: The value of pi is 3.141592653589793
import with renaming: We can import a module by renaming the module as our wish. Example: import math as m x=m.pi print("The value of pi is", x) Output: The value of pi is 3.141592653589793	**import all:** We can import all names(definitions) form a module using * Example: from math import * x=pi print("The value of pi is", x) Output: The value of pi is 3.141592653589793

Built-in python modules are,

1.math– mathematical functions:

some of the functions in math module is,

➕ math.ceil(x) - Return the ceiling of x, the smallest integer greater

than or equal to x

- math. floor(x) - Return the floor of x, the largest integer less than or equal to x.
- math. factorial(x)-Return x factorial.
- math.gcd(x,y)-Return   the greatest common divisor of the integers a and b
- math.sqrt(x)- Return the square root of x
- math.pi - The mathematical constant   = 3.141592
- math.e – returns The mathematical constant e = 2.718281

2 .**random**-Generate pseudo-random numbers
- random.**randrange**(*stop*) random.**randrange**(*start*, *stop*[, *step*]) random.**uniform**(*a*, *b*)
- -Return a random floating point number

8.ILLUSTRATIVE PROGRAMS

Program for SWAPPING(Exchanging)of values	Output
a = int(input("Enter a value ")) b = int(input("Enter b value")) c = a a = b b =c print("a=",a,"b=",b,)	Enter a value 5 Enter b value 8 a=8 b=5
Program to find distance between twopoints	**Output**
import math x1=int(input("enter x1")) y1=int(input("enter y1")) x2=int(input("enter x2")) y2=int(input("enter y2")) distance =math.sqrt((x2-x1)**2)+((y2- y1)**2) print(distance)	enter x17 enter y16 enter x25 enter y27 2.5
Program to circulate n numbers	**Output:**
a=list(input("enter the list"))	enter the list '1234'

print(a) for i in range(1,len(a),1): print(a[i:]+a[:i])	['1', '2', '3', '4'] ['2', '3', '4', '1'] ['3', '4', '1', '2'] ['4', '1', '2', '3']

2marks:
1. What is Python?

    Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language.

2. Enlist some features of python.
    * Easy-to-learn.
    * Easy-to-maintain.
    * Portable
    * Interpreted
    * Extensible
    * Free and Open Source
    * High Level Language

3. What is IDLE?

    Integrated Development Learning Environment (IDLE) is a **graphical user interface** which is completely written in Python. It is bundled with the default implementation of the python language and also comes with optional part of the Python packaging.

4. Differentiate between interactive and script mode.

Interactive mode	Script mode
A way of using the Python interpreter by typing commands and expressions at the prompt.	A way of using the Python interpreter to read and execute statements in a script.
Cant save and edit the code	Can save and edit the code
we cannot save the statements for further use and we have to retype all the statements to re-run them.	we can save the statements for further use and we no need to retype all the statements to re-run them.
We can see the results immediately.	We cant see the code immediately.

5. What are keywords? Give examples.
    * Keywords are the reserved words in Python.
    * We cannot use a keyword as variable name, function name or any other identifier.
    * They are used to define the syntax and structure of the Python language.
    * Keywords are case sensitive.

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	

6. What is a tuple?

❖ A tuple is same as list, except that the set of elements is **<u>enclosed in parentheses</u>** instead of square brackets.

❖ **A tuple is an immutable list.**i.e. once a tuple has been created, you can't add elements to a tuple or remove elements from the tuple.

7. Outline the logic to swap the contents of two identifiers without using third variable.

Swap two numbers	Output:
a=2;b=3 print(a,b) a = a+b b= a-b a= a-b print(a,b)	(2, 3) (3, 2) >>>

8. State about logical operators available in python with example.

Logical operators are " and, or, not" operators.

Operator	Meaning	Example
and	True if both the operands are true	x and y
or	True if either of the operands is true	x or y
not	True if operand is false (complements the operand)	not x

Example

a = True
b = False
print('a and b is',a and b)
print('a or b is',a or b)
print('not a is',not a)

Output

a and b is False
a or b is True
not a is False

9. What are the needs used for Function?

• When the program is too complex and large they are divided into parts. Each part is separately coded and combined into single program. Each subprogram is called as function.

• Debugging, Testing and maintenance becomes easy when the program is divided into subprograms.

• Functions are used to avoid rewriting same code again and again in a program.

• Function provides code re-usability

• The length of the program is reduced.

10. What is return statement?

The **return statement is used to exit a function** and go back to the place from where it was called. If the return statement has no arguments, then it will not return any values. But exits from function.

> **Syntax:**
> return[expression]

11. What are the types of arguments?

- Required Arguments
- Keyword Arguments
- Default Arguments
- Variable length Arguments

12. Define a module.

A module is a file containing Python definitions, functions, statements and instructions. Standard library of Python is extended as modules. To use these modules in a program**,** programmer needs to import the module.

13. **What is meant by interpreter?**

An interpreter is a computer program that executes instructions written in a programming language. It can either execute the source code directly or translate the source code in a first step into a more efficient representation and executes this code.

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 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.

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, scope: local and global, composition ,recursion; Strings: string slices, immutability, string functions and methods, string module; Lists as arrays. Illustrative programs: square root, gcd, exponentiation, sum the array of numbers, linear search, binary search.

1) Conditional Statements

- Conditional if
- Alternative if… else
- Chained if…elif…else
- Nested if….else

Conditional (if):

conditional (if) is used to test a condition, if the condition is true the statements inside if will be executed.

syntax:

```
if(condition 1):
        Statement 1
```

Flowchart:
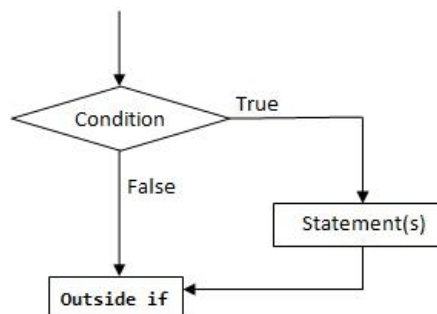
Program to provide bonus mark if the category is sports	output
m=eval(input("enter ur mark out of 100")) c=input("enter ur categery G/S") if(c=="S"): m=m+5 print("mark is",m)	enter ur mark out of 100 85 enter ur categery G/S S mark is 90

Alternative (if-else):

In the alternative the condition must be true or false. In this **else** statement can be combined with **if** statement. The **else** statement contains the block of code that executes when the condition is false. If the condition is true statements inside the if get executed otherwise else part gets executed. The alternatives are called branches, because they are branches in the flow of execution.

syntax:

```
if(condition 1):
    Statement 1
else:
    Statement 2
```

Flowchart:
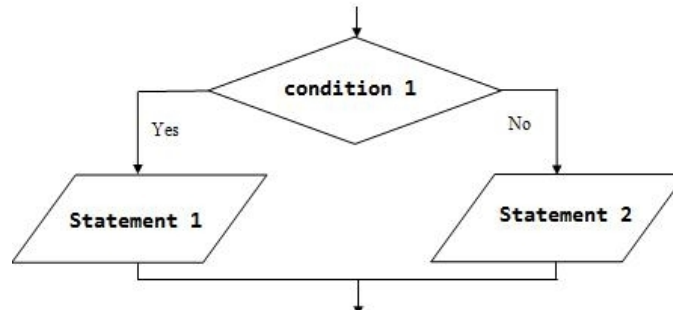
Examples:
1. odd or even number
2. positive or negative number
3. leap year or not

Odd or even number	Output

n=eval(input("enter a number"))
if(n%2==0):
   print("even number")
else:
   print("odd number")
	enter a number4
	even number

positive or negative number	**Output**

n=eval(input("enter a number"))
if(n>=0):
   print("positive number")
else:
   print("negative number")
	enter a number8
	positive number

leap year or not	**Output**

y=eval(input("enter a year"))
if(y%4==0):
   print("leap year")
else:
   print("not leap year")
	enter a year2000
	leap year

Chained conditionals (if-elif-else)

- The elif is short for else if.

- This is used to check more than one condition.

- If the condition1 is False, it checks the condition2 of the elif block. If all the conditions are False, then the else part is executed.

- Among the several if...elif...else part, only one part is executed according to the condition.

  The if block can have only one else block. But it can have multiple elif blocks.

  - The way to express a computation like that is a chained conditional.

syntax:

```
if(condition 1):
     statement 1
elif(condition 2):
     statement 2
elif(condition 3):
     statement 3
else:
     default statement
```

Flowchart:
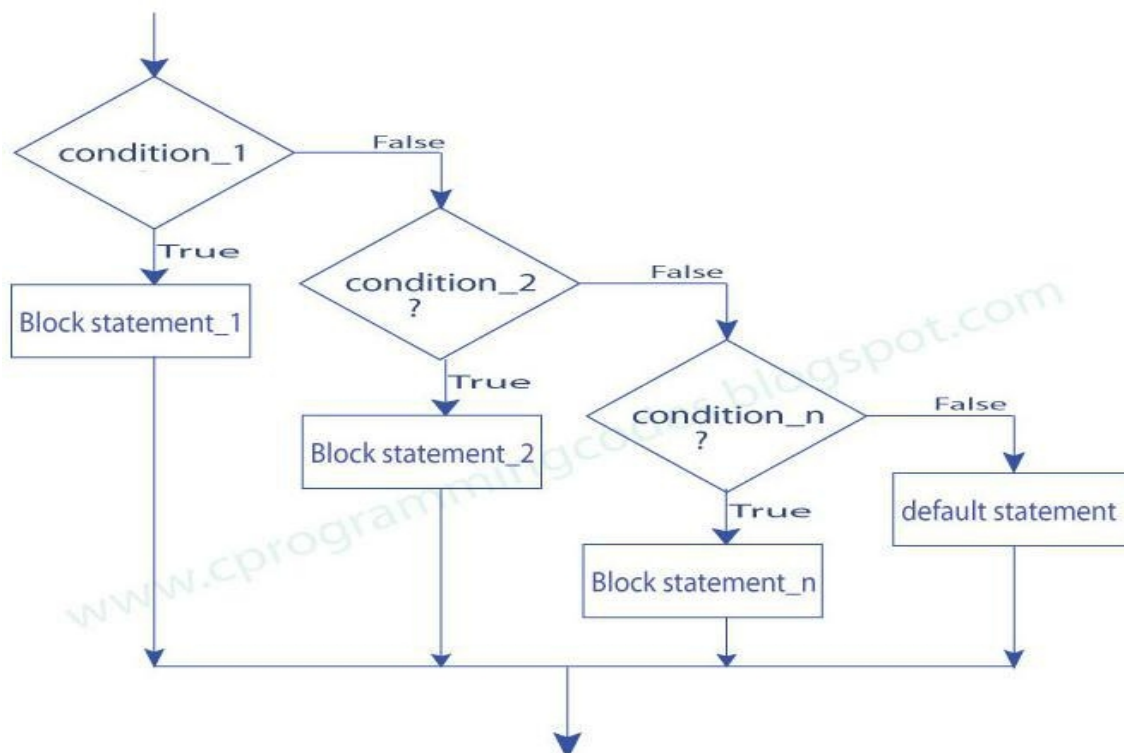
Example:

1. student mark system
2. traffic light system

student mark system	Output

mark=eval(input("enter ur mark:"))
if(mark>=90):
   print("grade:S")
elif(mark>=80):
   print("grade:A")
elif(mark>=70):
   print("grade:B")
elif(mark>=50):
   print("grade:C")
else:
   print("fail")
	enter ur mark:78
	grade:B

traffic light system	**Output**

colour=input("enter colour of light:")
if(colour=="green"):
   print("GO")
elif(colour=="yellow"):
   print("GET READY")
else:
   print("STOP")
	enter colour of light:green
	GO

Nested conditionals

One conditional can also be nested within another. Any number of condition can be nested inside one another. In this, if the condition is true it checks another if condition1. If both the conditions are true statement1 get executed otherwise statement2 get execute. if the condition is false statement3 gets executed

Syntax

```
if (condition):
        if(condition 1):
              statement 1
        else:
              statement 2
    else:
        statement 3
```
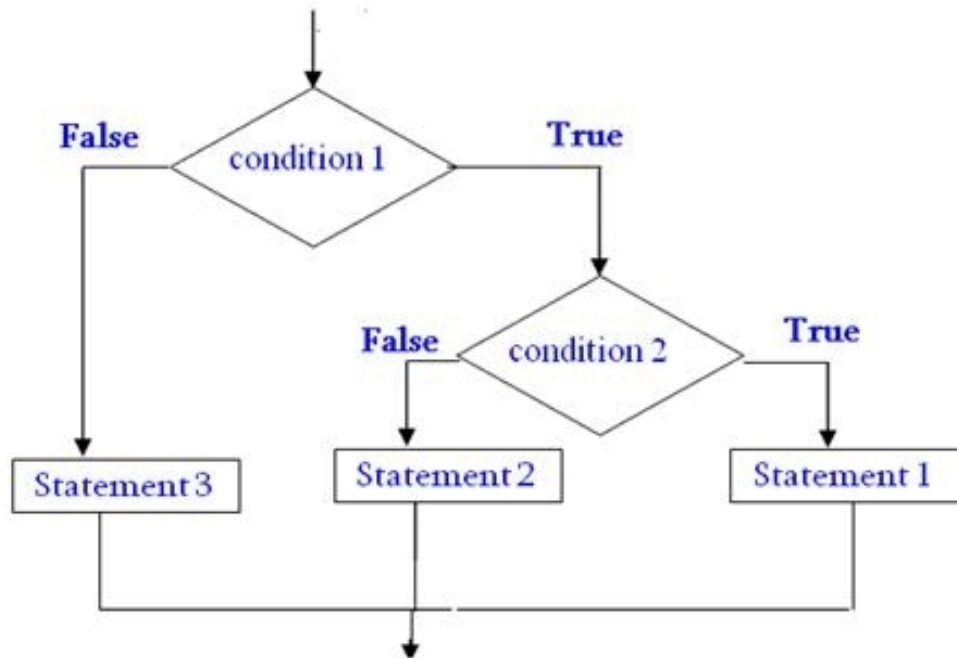
Example:

1. greatest of three numbers
2. positive negative or zero

greatest of three numbers	output
a=eval(input("enter the value of a"))	enter the value of a 9
b=eval(input("enter the value of b"))	enter the value of a 1
c=eval(input("enter the value of c"))	enter the value of a 8
if(a>b):	the greatest no is 9
if(a>c):	
print("the greatest no is",a)	
else:	
print("the greatest no is",c)	
else:	
if(b>c):	
print("the greatest no is",b)	
else:	
print("the greatest no is",c)	

positive negative or zero	output
n=eval(input("enter the value of n:"))	enter the value of n:-9
if(n==0):	the number is negative
print("the number is zero")	
else:	
if(n>0):	
print("the number is positive")	

```
else:
    print("the number is negative")
```

2.Iteration Or Control Statements.

- state
- while
- for
- break
- continue
- pass

State:

Transition from one process to another process under specified condition with in a time is called state.

While loop:

While loop statement in Python is used to repeatedly executes set of statement as long as a given condition is true.

In while loop, test expression is checked first. The body of the loop is entered only if the test expression is True. After one iteration, the test expression is checked again. This process continues until the test expression evaluates to False.

In Python, the body of the while loop is determined through indentation.

The statements inside the while start with indentation and the first unintended line marks the end.

Syntax:

```
inital value
while(condition):
    body of while loop
    increment
```

Flow chart:
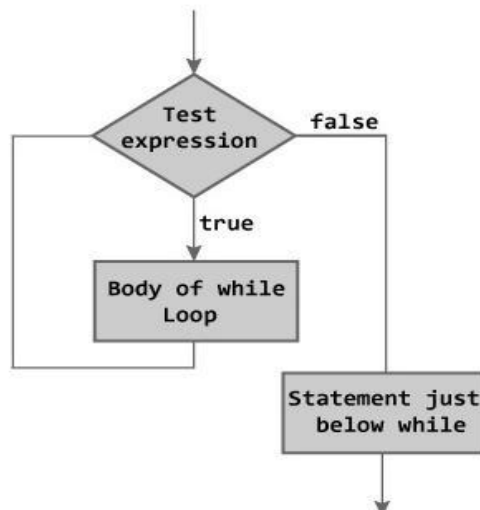
Examples:

1. program to find sum of n numbers:
2. program to find factorial of a number
3. program to find sum of digits of a number:
4. Program to Reverse the given number:
5. Program to find number is Armstrong number or not
6. Program to check the number is palindrome or not

Sum of n numbers:	output

n=eval(input("enter n"))
i=1
sum=0
while(i<=n):
  sum=sum+i
  i=i+1
print(sum)
	enter n
	10
	55

Factorial of a numbers:	output

n=eval(input("enter n"))
i=1
fact=1
while(i<=n):
  fact=fact*i
  i=i+1
print(fact)
	enter n
	5
	120

Sum of digits of a number:	output

n=eval(input("enter a number"))
sum=0
while(n>0):
  a=n%10
  sum=sum+a
  n=n//10
print(sum)
	enter a number
	123
	6

Reverse the given number:	output

n=eval(input("enter a number"))
sum=0
while(n>0):
  a=n%10
  sum=sum*10+a
  n=n//10
print(sum)
	enter a number
	123
	321

Armstrong number or not	output

n=eval(input("enter a number"))
org=n
sum=0
while(n>0):
  a=n%10
  sum=sum+a*a*a
  n=n//10
if(sum==org):
  print("The given number is Armstrong number")
else:
  print("The given number is not Armstrong number")
	enter a number153
	The given number is Armstrong number

Palindrome or not	output

n=eval(input("enter a number"))
org=n
sum=0
while(n>0):
  a=n%10
  sum=sum*10+a
  n=n//10
if(sum==org):
  print("The given no is palindrome")
else:
  print("The given no is not palindrome")
	enter a number121
	The given no is palindrome

For loop:

- *for in range:*
  - We can generate a sequence of numbers using range() function. range(10) will generate numbers from 0 to 9 (10 numbers).
  - In range function have to define the start, stop and step size as range(start,stop,step size). step size defaults to 1 if not provided.

syntax

```
for i in range(start,stop,steps):
    body of for loop
```

Flowchart:
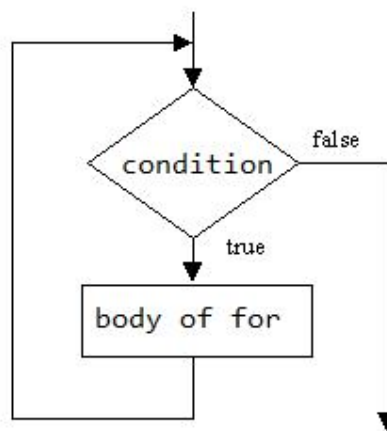
For in sequence

- The for loop in Python is used to iterate over a sequence (list, tuple, string). Iterating over a sequence is called traversal. Loop continues until we reach the last element in the sequence.
- The body of for loop is separated from the rest of the code using indentation.

```
for i in sequence:
    print(i)
```

Sequence can be a list, strings or tuples

s.no	sequences	example	output
1.	For loop in string	for i in "Ramu": print(i)	R A M U

			2
2.	For loop in list	for i in [2,3,5,6,9]: print(i)	3 5 6 9
3.	For loop in tuple	for i in (2,3,1): print(i)	2 3 1

Examples:

1. Program to print Fibonacci series.
2. check the no is prime or not

Fibonacci series	output
a=0	Enter the number of terms: 6
b=1	Fibonacci Series:
n=eval(input("Enter the number of terms: "))	0 1
print("Fibonacci Series: ")	1
print(a,b)	2
for i in range(1,n,1):	3
c=a+b	5
print(c)	8
a=b	
b=c	

check the no is prime or not	output
n=eval(input("enter a number"))	enter a no:7
for i in range(2,n):	The num is a prime number.
if(n%i==0):	
print("The num is not a prime")	
break	
else:	
print("The num is a prime number.")	

3.Loop Control Structures

BREAK

- Break statements can alter the flow of a loop.
- It terminates the current
- loop and executes the remaining statement outside the loop.
- If the loop has else statement, that will also gets terminated and come out of the loop completely.

Syntax:

break

```
while (test Expression):

    // codes
    if (condition for break):

        break

    // codes
```

Flowchart
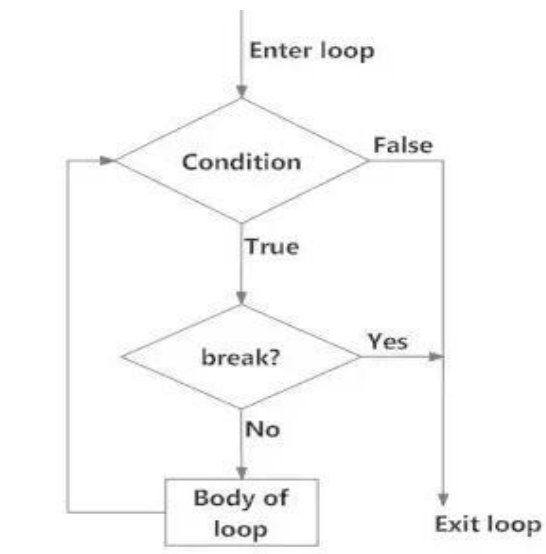
example	Output

for i in "welcome":
  if(i=="c"):
    break
  print(i)
	w
	 e
	 l

It terminates the current iteration and transfer the control to the next iteration in the loop.

Syntax: Continue

```
while (test Expression):

    // codes
    if (condition for continue):

        continue

    // codes
```

Flowchart
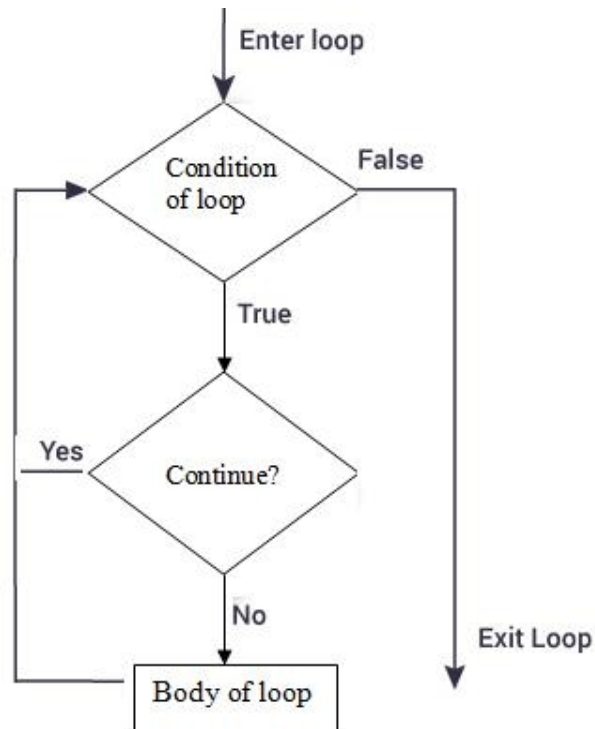
Example:	Output

for i in "welcome":
  if(i=="c"):
    continue
  print(i)
	w
	e
	l
	o
	m
	e

PASS

- It is used when a statement is required syntactically but you don't want any code to execute.

- It is a null statement, nothing happens when it is executed.

    pass
    break

Example	Output

for i in "welcome":
    if (i == "c"):
        pass
    print(i)
	w
	e
	l
	c
	o
	m
	e

Difference between break and continue

break	continue
It terminates the current loop and executes the remaining statement outside the loop.	It terminates the current iteration and transfer the control to the next iteration in the loop.
syntax: break	syntax: continue

for i in "welcome":
    if(i=="c"):
        break
    print(i)
	for i in "welcome":
	    if(i=="c"):
	        continue
	    print(i)

w e l	w e l o m e

else statement in loops:

else in for loop:

- If else statement is used in for loop, the else statement is executed when the loop has reached the limit.
- The statements inside for loop and statements inside else will also execute.

example	output

for i in range(1,6):
 	
   print(i)
 	
else:
 	
   print("the number greater than 6")
	1
	
	2
	
	3
	
	4
	
	5 the number greater than 6

else in while loop:

▢ If else statement is used within while loop , the else part will be executed when the condition become false.

▢ The statements inside for loop and statements inside else will also execute.

Program	output

i=1
 	
while(i<=5):
 	
   print(i)
 	
   i=i+1
 	
else:
 	
   print("the number greater than 5")
	1
	
	2
	
	3
	
	4
	
	5
	
	the number greater than 5

4) Fruitful Function

- Fruitful function
- Void function
- Return values
- Parameters
- Local and global scope
- Function composition
- Recursion

A function that returns a value is called fruitful function.

Example:

      Root=sqrt (25)

Example:

```
def add():
        a=10
        b=20
        c=a+b
        return c
c=add()
print(c)
```

Void Function

A function that perform action but don't return any value.

Example:

print("Hello")

Example:

```
def add():
      a=10
      b=20
      c=a+b
      print(c)
add()
```

Return values:

return keywords are used to return the values from the function.

example:

return a – return 1 variable

return a,b– return 2 variables

return a+b– return expression

return 8– return value

PARAMETERS / ARGUMENTS(refer 2nd unit)

Local and Global Scope

Global Scope

- The *scope* of a variable refers to the places that you can see or access a variable.
- A variable with global scope can be used anywhere in the program.
- It can be created by defining a variable outside the function.

Example	output

a=50
 	
def add():
 	
Global Variable
 	
  b=20
  c=a+b
  print© **Local Variable**
 	
def sub():
  b=30
  c=a-b
  print©
print(a)
	70
	
	20
	
	50

Local Scope A variable with local scope can be used only within the function .

Example	output

def add():
    b=20
 	
    c=a+b
    Local Variable
    print©
	70

def sub():
    b=30
 	
    c=a-b
    Local Variable
 	
    print©
	20

print(a)	error
print(b)	error

Function Composition:

- Function Composition is the ability to call one function from within another function
- It is a way of combining functions such that the result of each function is passed as the argument of the next function.
- In other words the output of one function is given as the input of another function is known as function composition.

find sum and average using function composition	output

def sum(a,b):
    sum=a+b
    return sum
def avg(sum):
        avg=sum/2
    return avg
a=eval(input("enter a:"))
b=eval(input("enter b:"))
sum=sum(a,b)
avg=avg(sum)
print("the avg is",avg)
	enter a:4
	enter b:8
	the avg is 6.0

Recursion

A function calling itself till it reaches the base value - stop point of function call. Example: factorial of a given number using recursion

Factorial of n	Output

def fact(n):
  if(n==1):
    return 1
  else:
    return n*fact(n-1)
 	
n=eval(input("enter no.   to find fact:"))
fact=fact(n)
print("Fact is",fact)
	enter no. to find fact:5
	Fact is 120

Explanation

Final value = 120

5! = 5 * 24 = 120 is returned

4! = 4 * 6 = 24 is returned

3! = 3 * 2 = 6 is returned

2! = 2 * 1 = 2 is returned

1! = 1 * 1 = 1 is returned

1 is returned
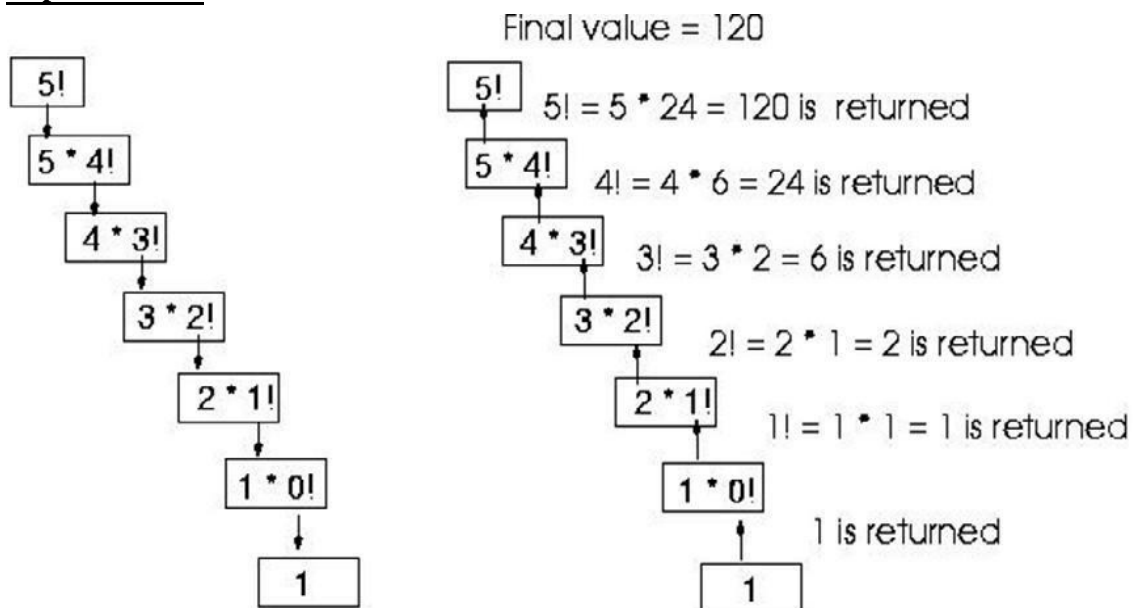
Examples:

1. sum of n numbers using recursion
2. **exponential of a number using recursion**

Sum of n numbers	Output

def sum(n):
  if(n==1):
    return 1
  else:
    return n*sum(n-1)
 	
n=eval(input("enter no. to find sum: "))
 	
sum=sum(n)
print("Fact is",sum)
	enter no. to find sum:10
	Fact is 55

5)Explain about Strings and its operation:

- String is defined as sequence of characters represented in quotation marks (either single quotes (') or double quotes (").
- An individual character in a string is accessed using a index.
- The index should always be an integer (positive or negative).
- A index starts from 0 to n-1.
- Strings are immutable i.e. the contents of the string cannot be changed after it is created.
- Python will get the input at run time by default as a string.
- Python does not support character data type. A string of size 1 can be treated as characters.

1. single quotes ('   ')

2. double quotes (" ")

3. triple quotes(""" """)

Operations on string:

1. Indexing
2. Slicing
3. Concatenation
4. Repetitions
5. Member ship

String A	H	E	L	L	O
Positive Index	0	1	2	3	4
Negative Index	-5	-4	-3	-2	-1

indexing	>>>a="HELLO" >>>print(a[0]) >>>H >>>print(a[-1]) >>>O	Positive indexing helps in accessing the string from the beginning Negative subscript helps in accessing the string from the end.
Slicing:	Print[0:4] – HELL Print[:3] – HEL Print[0:]- HELLO	The Slice[start : stop] operator extracts sub string from the strings. A segment of a string is called a slice.
Concatenation	a="save" b="earth" >>>print(a+b) Save earth	The + operator joins the text on both sides of the operator.
Repetitions:	a="panimalar " >>>print(3*a)	The * operator repeats the string on the left hand side times the value on right

78

	panimalarpanimalar panimalar	hand side.

Membership:	>>> s="good morning"
	 >>>"m" in s
	True
	>>> "a" not in s
	True
		Using membership operators to check a
		particular character is in string or not.
		Returns true if present

- A part of a string is called string slices.
- **The process of extracting a sub string from a string is called slicing.**

Slicing: **a="HELLO"**	Print[0:4] – HELL Print[:3] – HEL Print[0:]- HELLO	The Slice[n : m] operator extracts sub string from the strings. A segment of a string is called a slice.

Immutability:

- Python strings are "immutable" as they cannot be changed after they are created.
- Therefore [ ] operator cannot be used on the left side of an assignment.

operations	Example	output
element assignment	a="PYTHON" a[0]='x'	TypeError: 'str' object does not support element assignment
element deletion	a="PYTHON" del a[0]	TypeError: 'str' object doesn't support element deletion
delete a string	a="PYTHON" del a print(a)	NameError: name 'my_string' is not defined

string built in functions and methods:

A **method** is a function that "belongs to" an object.

Syntax to access the method

Stringname.method()

a="happy birthday"
here, a is the string name.

	syntax	example	description
1	a.capitalize()	>>> a.capitalize() ' Happy birthday'	capitalize only the first letter in a string
2	a.upper()	>>> a.upper() 'HAPPY BIRTHDAY'	change string to upper case
3	a.lower()	>>> a.lower() ' happy birthday'	change string to lower case
4	a.title()	>>> a.title() ' Happy Birthday '	change string to title case i.e. first characters of all the words are capitalized.
5	a.swapcase()	>>> a.swapcase() 'HAPPY BIRTHDAY'	change lowercase characters to uppercase and vice versa
6	a.split()	>>> a.split() ['happy', 'birthday']	returns a list of words separated by space
7	a.center(width,"fillchar")	>>>a.center(19,"*") '***happy birthday***'	pads the string with the specified "fillchar" till the length is equal to "width"
8	a.count(substring)	>>> a.count('happy') 1	returns the number of occurences of substring
9	a.replace(old,new)	>>>a.replace('happy', 'wishyou happy') 'wishyou happy birthday'	replace all old substrings with new substrings

10	a.join(b)	>>> b="happy"
		>>> a="-"
		>>> a.join(b)
		'h-a-p-p-y'
			returns a string concatenated with the elements of an iterable. (Here "a" is the iterable)

11	a.isupper()	>>> a.isupper() False	checks whether all the case-based characters (letters) of the string are uppercase.
12	a.islower()	>>> a.islower() True	checks whether all the case-based characters (letters) of the string are lowercase.
13	a.isalpha()	>>> a.isalpha() False	checks whether the string consists of alphabetic characters only.

- A module is a file containing Python definitions, functions, statements.
- Standard library of Python is extended as modules.
- To use these modules in a program, programmer needs to import the module.
- Once we import a module, we can reference or use to any of its functions or variables in our code.
- There is large number of standard modules also available in python.
- Standard modules can be imported the same way as we import our user-defined modules.

Syntax:

import module_name

Example	output
import string print(string.punctuation) print(string.digits) print(string.printable) print(string.capwords("happ y birthday")) print(string.hexdigits) print(string.octdigits)	!"#$%&'()*+,-./:;<=>?@[\]^_`{\|}~ 0123456789 0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJ KLMNOPQRSTUVWXYZ!"#$%&'()*+,- ./:;<=>?@[\]^_`{|}~ Happy Birthday 0123456789abcdefABCDEF 01234567

Escape sequences in string

Escape Sequence	Description	example
\n	new line	>>> print("hai \nhello") hai hello
\\	prints Backslash (\)	>>> print("hai\\hello") hai\hello
\'	prints Single quote (')	>>> print("'") '
\"	prints Double quote (")	>>>print("\"") "
\t	prints tab sapace	>>>print("hai\thello") hai hello
\a	ASCII Bell (BEL)	>>>print("\a")

6) Array:

Array is a collection of similar elements. Elements in the array can be accessed by index. Index starts with 0. Array can be handled in python by module named array.

To create array have to import array module in the program.
Syntax :
> import array
Syntax to create array:
> **Array_name = module_name.function_name('datatype',[elements])**
example:
> **a=array.array('i',[1,2,3,4])**
> a- array name
> array- module name
> i- integer datatype

Example

Program to find sum of array elements	Output
import array sum=0 a=array.array('i',[1,2,3,4]) for i in a: sum=sum+i print(sum)	10

Convert list into array:
fromlist() function is used to append list to array. Here the list is act like a array.

Syntax:
arrayname.fromlist(list_name)

Example

program to convert list into array	Output
import array sum=0 l=[6,7,8,9,5] a=array.array('i',[]) a.fromlist(l) for i in a: sum=sum+i print(sum)	35

Methods of an array

a=[2,3,4,5]

	Syntax	example	Description
1	array(data type, value list)	array('i',[2,3,4,5])	This function is used to create an array with data type and value list specified in its arguments.
2	append()	>>>a.append(6) [2,3,4,5,6]	This method is used to add the at the end of the array.
3	insert(index,element)	>>>a.insert(2,10) [2,3,10,5,6]	This method is used to add the value at the position specified in its argument.
4	pop(index)	>>>a.pop(1) [2,10,5,6]	This function removes the element at the position mentioned in its argument, and returns it.
5	index(element)	>>>a.index(2) 0	This function returns the index of value
6	reverse()	>>>a.reverse() [6,5,10,2]	This function reverses the array.
7	count()	a.count()	This is used to count number of

7.ILLUSTRATIVE PROGRAMS:

Square root using newtons method:		Output:
```def newtonsqrt(n): root=n/2 for i in range(10): root=(root+n/root)/2 print(root) n=eval(input("enter number to find Sqrt: ")) newtonsqrt(n)```		enter number to find Sqrt: 9 3.0
GCD of two numbers		**output**
```n1=int(input("Enter a number1:")) n2=int(input("Enter a number2:")) for i in range(1,n1+1): if(n1%i==0 and n2%i==0): gcd=i print(gcd)```		Enter a number1:8 Enter a number2:24 8
Exponent of number		Output:
```def power(base,exp): if(exp==1): return(base) else: return(base*power(base,exp-1)) base=int(input("Enter base: ")) exp=int(input("Enter exponential value:")) result=power(base,exp) print("Result:",result)```		Enter base: 2 Enter exponential value:3 Result: 8
sum of array elements:		**output:**
```a=[2,3,4,5,6,7,8] sum=0 for i in a: sum=sum+i print("the sum is",sum)```		the sum is 35
Linear search		**output**
```a=[20,30,40,50,60,70,89] print(a) search=eval(input("enter a element to search:")) for i in range(0,len(a),1): if(search==a[i]): print("element found at",i+1) break else: print("not found")```		[20, 30, 40, 50, 60, 70, 89] enter a element to search:30 element found at 2

Binary search	output

``
a=[20, 30, 40, 50, 60, 70, 89]
print(a)
search=eval(input("enter a element to search:"))
start=0
stop=len(a)-1
while(start<=stop):
    mid=(start+stop)//2
    if(search==a[mid]):
        print("element found at",mid+1)
        break
    elif(search<a[mid]):
        stop=mid-1
    else:
        start=mid+1
else:
    print("not found")
``
	[20, 30, 40, 50, 60, 70, 89]
	enter a element to search:30
	element found at 2

Two marks:

1. **What is a Boolean value?**
   - Boolean data type have two values. They are 0 and 1.
   - 0 represents False
   - 1 represents True
   - True and False are keyword.

Example:
```
>>> 3==5
False
>>> 6==6
True
>>> True+True
2
>>> False+True
1
>>> False*True
0
```

2. **Difference between break and continue.**

break	continue
It terminates the current loop and executes the remaining statement outside the loop.	It terminates the current iteration and transfer the control to the next iteration in the loop.
syntax: break	syntax: continue

for i in "welcome":
  if(i=="c"):
    break
  print(i)
	for i in "welcome":
	  if(i=="c"):
	    continue
	  print(i)

w e l	w e l o m e

3. **Write a Python program to accept two numbers, multiply them and print the result.**

```
number1 = int(input("Enter first number: "))
number2 = int(input("Enter second number: "))
mul = number1 * number2
print("Multiplication of given two numbers is: ", mul)
```

4. **Write a Python program to accept two numbers, find the greatest and print the result.**

```
number1 = int(input("Enter first number: "))
number2 = int(input("Enter second number: "))
if(number1>number2):
        print('number1 is greater',number1)
else:
        print('number2 is greater',number2)
```

5. Define recursive function.

Recursion is a way of programming or coding a problem, in which a function calls itself one or more times in its body. Usually, it is returning the return value of this function call. If a function definition fulfils the condition of recursion, we call this function a recursive function.

Example:

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

6. Write a program to find sum of n numbers:

n=eval(input("enter n"))	enter n
i=1	10
sum=0	55
while(i<=n):	
sum=sum+i	
i=i+1	
print(sum)	

7. **What is the purpose of pass statement?**

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

- It is used when a statement is required syntactically but you don't want any code to execute.

- It is a null statement, nothing happens when it is executed.

    pass
    break

Example	Output
for i in "welcome":	w
if (i == "c"):	e
pass	l
print(i)	c
	o
	m
	e

8. **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

9. **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")

10. **Mention a few string functions.**

    s.captilize() – Capitalizes first character of string

    s.count(sub) – Count number of occurrences of string

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

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

1. Insertion sort

Insertion sort is an elementary sorting algorithm that sorts one element at a time. Most humans, when sorting a deck of cards, will use a strategy similar to insertion sort. The algorithm takes an element from the list and places it in the correct location in the list. This process is repeated until there are no more unsorted items in the list.

Example:
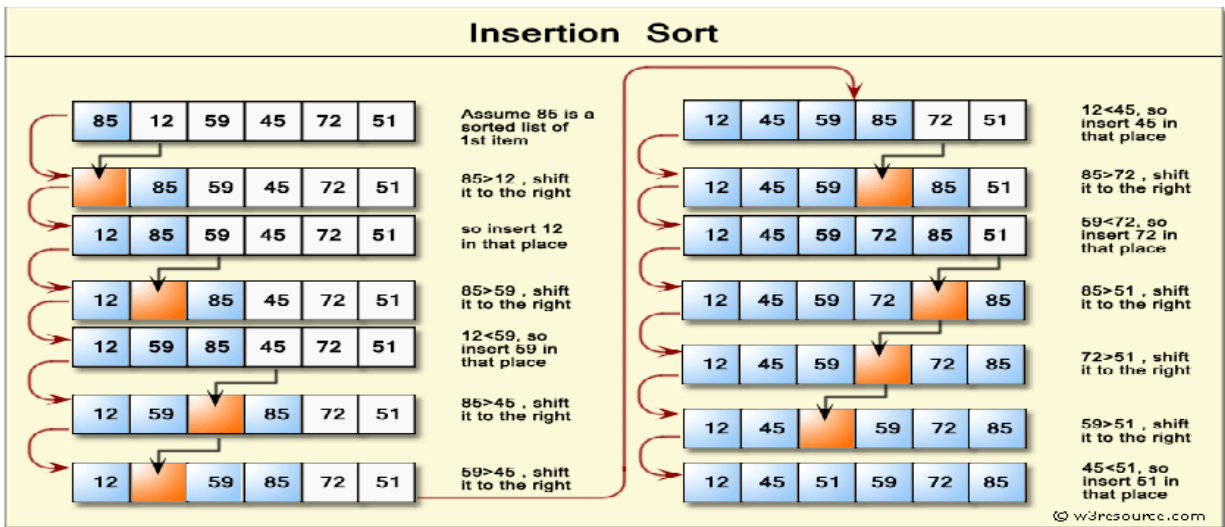
Program:

```
a=list()
n=int(input("Enter size of list"))
for i in range(n):
    a.append(int(input("Enter list elements")))
 print("Before sorting",a)
for i in range(1,n):
  key=a[i]
  j=i-1
  while j>=0 and key<a[j]:
    a[j+1]=a[j]
    j-=1
    a[j+1]=key
print("After sorting(using insertion sort)",a)
```

Output

Enter size of list6
Enter listelements4
Enter listelements33
Enter list elements6
Enter listelements22
Enter list elements6
Enter list elements-9
Before sorting [4, 33, 6, 22, 6, -9]
After sorting(using insertion sort) [-9, 4, 6, 6, 22, 33]

2. Selection Sort

The selection sort algorithm starts by finding the minimum value in the array and moving it to the first position. This step is then repeated for the second lowest value, then the third, and so on until the array is sorted.
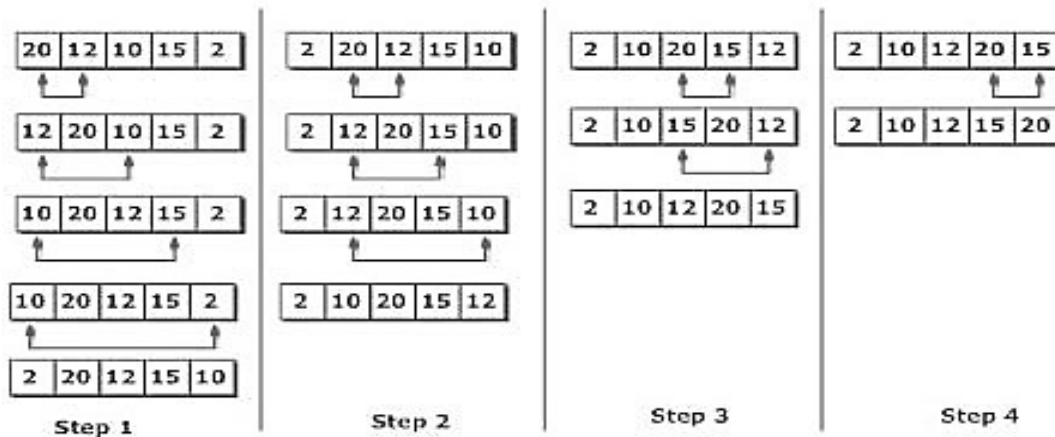
Example

Figure: Selection Sort

```
Program
a=list()
n=int(input("Enter size of list"))
for i in range(n):
    a.append(int(input("Enter list elements")))
print("List before sorting",a)
for i in range(0,n):
  j=i+1
  for j in range(j, n):
    if a[i]> a[j]:
        temp=a[i]
        a[i]=a[j]
        a[j]=temp
print("Sorted list(using Selection Sort)=",a)
```

Output:
Enter size of list5
Enter list elements12
Enter list elements-5
Enter list elements4
Enter listelements48
Enter listelements98
List before sorting [12, -5, 4, 48, 98]
Sorted list(using Selection Sort)= [-5, 4, 12, 48, 98]

3. Quadratic Equation:

Formula :

$$ax^2+bx+c \qquad = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$$

Program

```
import cmath
 a = int(input("Enter the coefficients a:"))
b=int(input("Enter the coefficients b: "))
c = int(input("Enter the coefficients c: "))
 d = b**2-4*a*c # discriminant
 x1 = (-b+cmath.sqrt((b**2)-(4*(a*c))))/(2*a)
 x2 = (-b-cmath.sqrt((b**2)-(4*(a*c))))/(2*a)
 print ("This equation has two solutions: ", x1, " or", x2)
```

Output

Enter the coefficients a: 5

Enter the coefficients b: 1

Enter the coefficients c: 2

This equation has two solutions: (-0.1+0.6244997998398398j) or (-0.1-0.6244997998398398j)

Enter the coefficients a: 1

Enter the coefficients b: -5
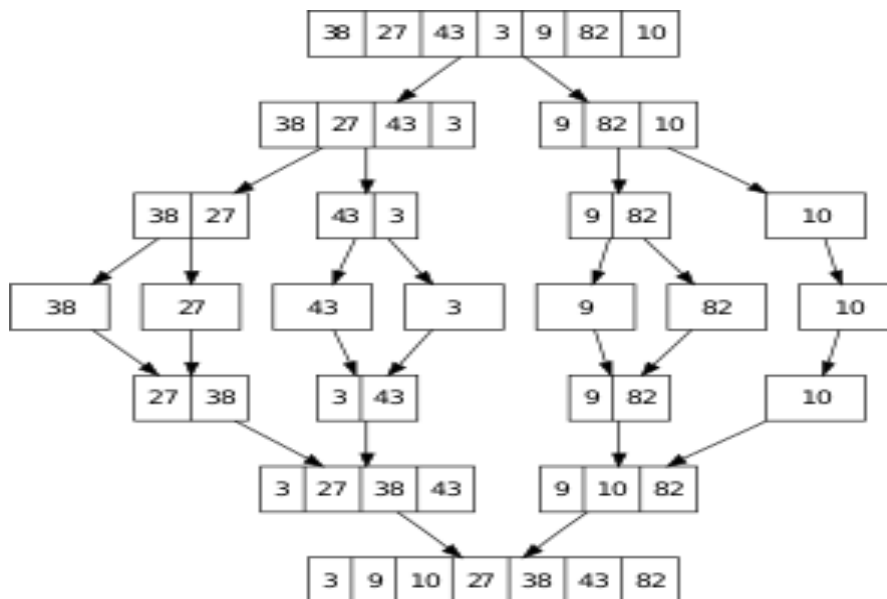
Enter the coefficients c: 6

This equation has two solutions: (3+0j) or (2+0j)

4. Merge sort

Merge sort works as follows:

a. Divide the unsorted list into n sub lists, each containing 1 element (a list of 1 element is considered sorted).

b. Repeatedly merge sub lists to produce new sorted sub lists until there is only 1 sub list remaining. This will be the sorted list.

Example

Program:

```python
def merge(left, right):
    result = []
    i, j = 0, 0
    while (i < len(left) and j<len(right)):
        if left[i] < right[j]:
            result.append(left[i])
            i+= 1
        else:
            result.append(right[j])
            j+= 1
    result=result+left[i:]
    result=result+right[j:]
    return result
def mergesort(list):
    if len(list) < 2:
        return list
    middle = len(list)//2
    left = mergesort(list[:middle])
    right = mergesort(list[middle:])
    return merge(left, right)
a=list()
n=int(input("Enter size of list"))
for i in range(n):
    a.append(int(input("Enter list elements")))
print("Unsorted list is",a)
print("Sorted list using merge sort is",a)
```

Output
Enter size of list5
Enter list elements21
Enter list elements1
Enter list elements-8
Enter list elements14
Enter list elements18
Unsorted list is [21, 1, -8, 14, 18]
Sorted list using merge sort is [-8, 1, 14, 18, 21]

5. LIST

- o List is a sequence of values, which can be of different types. The values in list are called "elements" or "items"
- o Each elements in list is assigned a number called "position" or "index"
- o A list that contains no elements is called an empty list. They are created with empty brackets[]
- o A list within another list is nested list

Creating a list :

The simplest way to create a new list is to enclose the elements in square brackets ([])

[10,20,30,40]

[100, "python" , 8.02]

1.LIST OPERATIONS:

1. Concatenation of list
2. Repetition of list

Concatenation: the '+' operator concatenate list

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

>>> b = [4,5,6]

>>> c = a+b

>>> Print (a*2) => [1,2,3,1,2,3]

Repetition: the '*' operator repeats a list a given number of times

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

>>> b = [4,5,6]

>>> print (a*2)= [1,2,3,1,2,3]

2. List looping: (traversing a list)

1. Looping in a list is used to access every element in list
2. "for loop" is used to traverse the elements in list

```
eg: mylist = ["python","problem",100,6.28]
    for i in range (len (mylist)):
        print (mylist [i])
```

3.List Slices:

A subset of elements of list is called a slice of list.

```
Eq: n = [1,2,3,4,5,6,7,8,9,10]
    print (n[2:5])
    print (n[-5])
    print (n[5: ])
    print (n[ : ])
```

4. Aliasing and cloning:

- when more than one variables refers to the same objects or list, then it is called aliasing.

> a= [5,10,50,100]
> b=a
> b[0] = 80
> print ("original list", a) = [5,10,50,100]
> print ("Aliasing list", b) = [80,5,10,50,100]

- Here both a & b refers to the same list. Thus, any change made with one object will affect other, since they are mutable objects.
- in general, it is safer to avoid aliasing when we are working with mutable objects

5. Cloning:

- Cloning creates a new list with same values under another name. Taking any slice of list create new list.
- Any change made with one object will not affect others. the easiest way to clone a new list is to use "slice operators"

> a = [5,10,50,100]
> b= a[:]
> b[0] = 80
> Print (" original list", a) = [5,10,50,100]
> Print (" cloning list", b) = [5,10,50,100]

List parameter:

- List can be passed as arguments to functions the list arguments are always passed by reference only.
- Hence, if the functions modifies the list the caller also changes.

> Eq: def head ():
>   del t[0]
>   >>> letters = ['a','b','c']
>   >>> head (letters)
>   >>> letters
>   ['b','c']

In above,

The parameters 't' and the variable 'letters' or aliases for the same objects

An alternative way to write a function that creates and return a new list

> Eq: def tail (t):
>   return t [1:]
>   >>> letters = ['a','b','c']
>   >>> result = tail (letters)
>   >>> result
>   ['b','c']

In above,

The function leaves the original list unmodified and return all element in list except first element

6. TUPLES:

A tuple is a sequence of value which can be of any type and they are indexed by integers. Values in tuple are enclosed in parentheses and separated by comma. The elements in the tuple cannot be modified as in list (i.e) tuple are immutable objects

Creating tuple:

Tuple can be created by enclosing the element in parentheses separated by comma

    t = ('a','b','c','d')

To create a tuple with a single element we have to include a final comma

    >>> t = 'a',
    >>> type (t)
        < class 'tuple'>

Alternative way to create a tuple is the built-in function tuple which mean, it creates an empty tuple

    >>> t = tuple ()
    >>> t
    >>> ( )

Accessing element in tuple:

If the argument in sequence, the result is a tuple with the elements of sequence.

    >>>t= tuple('python')
    >>> t
        ('p','y','t','h','o','n')
      t = ('a','b',100,8.02)
      print (t[0]) = 'a'
      print (t[1:3]) = ('b', 100 , 8.02)

Deleting and updating tuple:

Tuple are immutable, hence the elements in tuple cannot be updated / modified
But we can delete the entire tuple by using keyword 'del'

Eg 1: a = (' programming', 200, 16.54, 'c', 'd')
#Try changing an element.

      a[0] = 'python'   <-------- Error,modifying not possible
      print (a [0])

Eg: # Deletion of tuple

      a = ('a','b','c','d')
      del (a) :-------- delete entire tuple
      del a [1] <--------- error,deleting one element in tuple not possible

Eg: # replacing one tuple with another

      a = ('a','b','c','d')
      t = ('A',) + a[1:]
      print (t) <------ ('a','b','c','d')

Tuple Assignment:

- •Tuple assignment is often useful to swap any number of values
- •the number of variables in left and right of assignment operators must be equal
- •A single assignment to paralleling assign value to all elements of tuple is the major benefit of tuple assignment

Eg: Tuple swapping in python

```
A= 100
B= 345
C= 450
print (" A & B:", A,"&",B)
    # Tuple assignments for two
variables A,B = B,A
print (" A&B after tuple assignment : ",A,"&",B)
    # Tuple assignment can be done for no of
variables A,B,C = C,A,B
print (" Tuple assignment for more variables:",
A,"&",B,"&",C) Output
A & B: 100 & 345
 A&B after tuple assignment : 345 & 100
 Tuple assignment for more variables: 450 & 345 & 100
```

Tuple as return value:

- •Generally, function can only return one value but if the value is tuple the same as returning the multiple value
- •Function can return tuple as return value

Eg: # the value of quotient & remainder are returned as tuple

```
def mod_div
   (x,y): quotient
   = x/y remainder
   = x%y
   return quotient, remainder
# Input the seconds & get the hours minutes &
second sec = 4234
minutes,seconds= mod_div
(sec,60)
hours,minutes=mod_div(minutes,
60)
print("%d seconds=%d hrs:: %d min:: %d
sec"%(sec,hours,minutes,seconds)) Output:
4234onds=1 hrs:: 10 min:: 34 sec
```

7.   Histogram

```
def histogram( items ):              Output
  for n in items:                       **
    output = '                         ***
    times = n                       ******
    while( times > 0 ):            *****
     output += '*'
     times = times - 1
    print(output)


histogram([2, 3, 6, 5])
```

Two marks:

1. Write a program to create list with n values

```
a=list()
n=int(input("Enter the size of list"))
for i in range (n):
    a.append(int(input("Enter the list element")))
print("Created List=",a)
```

Output
```
Enter the size of list 5
Enter the list of element20
Enter the list of element30
Enter the list of element78
Enter the list of element12
Enter the list of element65
Created List= [20, 30, 78, 12, 65]
```

2. What is dictionary?

A dictionary is an unordered set of key: value pair. In a list, the indices have to be integers; in a dictionary they can be any type. A dictionary contains a collection of indices, which are called keys, and a collection of values. Each key is associated with a single value. The association of a key and a value is called a key-value pair. Dictionary is created by enclosing with curly braces {}.

Eg:
```
>>>
dictionary={"RollNo":101,2:(1,2,3),"Name":"Ramesh",20:20.50,Loc":['Chennai']}
>>> dictionary
{'Name':'Ramesh', 'Loc':['Chennai'], 2:(1,2.3), 20: 20.0, 'RollNo': 101}
```

3. Write program to rotate values in the list.(counter-clock wise)

```
a=list()
n=int(input("Enter the number of list elements"))
for i in range (n):
    a.append(int(input("Enter list element")))
rotate=int(input("Enter the rotation value(Give negative value for
counter cock-wise)"))
print("Created List=",a)
print("List rotated is",a[rotate:]+a[:rotate] )
```

Output
```
Enter the number of list elements 5
Enter list element 30
Enter list element 98
Enter list element 45
Enter list element 49
Created List= [30, 98, 45, 49]
Enter the rotation value(Give negative value for counter cock-wise)-2
List rotated in counter clockwise [45, 49, 30, 98]
```

4. What is data structure? List out the data structures used in Python

A data structure is a particular way of organizing and storing data in a computer so that it can be accessed and modified efficiently.

Python data structures:-

1. List
2. Tuples
3. Dictionary

5. Compare all the three data structures in Python

	List	Tuples	Dictionary
Mutable	List is mutable	Tuples are immutable	Keys must be immutable. Values may mutable
Indexing	A positive integer is used for indexing and always starts with zero. Reverse index is supported.	A positive integer is used for indexing and always starts with zero. Reverse index is supported.	Indexing is done with 'key'. Index may be of any type. Values can be accessed only through key
Declaration	List=[05,'Ashok',450]	Tuple=('Sun','Mon')	Dictionary={"Key":"value"}

6. Difference between list append and list extend

1. append() is a function adds a new element to the end of a list.
2. extend() is a function takes a list as an argument and appends all of the elements.

append()	extend()
>>>a=[10,20,30]	>>>a=[10,20,30]
>>>b=[40,50]	>>>b=[40,50]
>>>a.append(b)	>>>a.extend(b)
>>>print(a)	>>>print(a)
[10,20,30,[40,50]]	[10,20,30,40,50]

7. What is mutability? Is tuple is mutable

In object-oriented and functional programming, an immutable object (unchangeable object) is an object whose state cannot be modified after it is created. This is in contrast to a mutable object (changeable object), which can be modified after it is created.

Tuple is immutable.

8. Write a program to add or change elements in a dictionary.

```
>>> dictionary={"Roll No":101,2:(20.00,30),"Name":"Ramesh",20:200.00, "Loc":['Chennai']}
>>> dictionary
{'Name': 'Ramesh', 'Loc': ['Chennai'], 2: (20.0, 30), 20: 200.0, 'Roll No': 101}
>>> dictionary['Roll No']=105
>>> dictionary
{'Name': 'Ramesh', 'Loc': ['Chennai'], 2: (20.0, 30), 20: 200.0, 'Roll No': 105}
```

9. How to convert a string to list of characters and words.

```
>>> str1="Hello"
>>> list1=list(str1)
>>> list1
['H', 'e', 'l', 'l', 'o']
```

10. What is zip operation in tuples. Give an example.

Zip is a built-in function that takes two or more sequences and returns a list of tuples where each tuple contains one element from each sequence. This example zips a string and a list:

```
>>> s = 'abc'
>>> t = [0, 1, 2]
>>> zip(s, t)
<zip object at 0x7f7d0a9e7c48>
```

The result is a zip object that knows how to iterate through the pairs. The most common use of zip is in a for loop:

```
>>> for pair in zip(s, t):
        print(pair)
('a', 0)
('b', 1)
('c', 2)
```

FILES, MODULES, PACKAGES

1. FILE AND ITS OPERATION

- File is a collection of record.
- A file stores related data, information, settings or commands in secondary storage device like magnetic disk, magnetic tape, optical disk, flash memory.

File Type

1. Text file

2. Binary file

Text file	Binary file
Text file is a sequence of characters that can be sequentially processed by a computer in forward direction Each line is terminated with a special character called the E0L or end of line character	A binary files store the data in the binary format(i.e .0's and 1's) It contains any type of data (pdf,images,word doc,spreadsheet,zip files,etc)

Mode in File

Module	Description
r	Read only
w	mode Write
a	only Appending
r+	only
	Read and write only

Differentiate write and append mode:

Write mode	Append mode
- It is used to write a string in a file - If file is not exist it creates a new file - If file is exit in the specified name, the existing content will overwrite in a file by the given string	- It is used to append (add) a string into a file - If file is not exist it creates a new file - It will add the string at the end of the old file

File Operation:

- ✓ Open a file
- ✓ Reading a file
- ✓ Writing a file
- ✓ Closing a file

1. Open ( ) function:

- Pythons built in open function to get a file object.
- The open function opens a file.
- It returns a something called a file object.
- File objects can turn methods and attributes that can be used to collect

Syntax:

file_object=open("file_name" , "mode")

Example:

fp=open("a.txt","r")

Create a text file

fp=open ("text.txt","w")

2. Read ( ) function

Read functions contains different methods

- read() – return one big string
- readline() – return one line at a time
- readlines() – return a list of lines

Syntax:

file_name.read ()

Example:

fp=open("a.txt","w")

print(fp.read())

print(fp.read(6))

print (fp.readline())

print (fp.readline(3))

print (fp.readlines())

 a.txt

A file stores related data, information, settings or commands in secondary storage device like magnetic disk, magnetic tape, optical disk, flash memory.

Output

Reading file using looping:

- Reading a line one by one in given file

  fp=open("a.txt","r")

  for line in fp:

  print(line)

3. Write ( ) function

This method is used to add information or content to existing file.

Syntax:

file_name.write( )

Example:

fp=open("a.txt","w")

fp.write("this file is a.txt")

fp.write("to add more lines")

fp.close()

Output: a.txt

> A file stores related data, information, settings or commands in secondary storage device like magnetic disk, magnetic tape, optical disk, flash memory.
> this file is a.txt to
> add more lines

4. Close ( ) function

It is used to close the file.

Syntax:

File name.close()

Example:

fp=open("a.txt","w")

fp.write("this file is a.txt")

fp.write("to add more lines")

fp.close()

Splitting line in a text line:

```
fp=open("a.txt","w")

for line in fp:

        words=line.split()

print(words)
```

2. Write a program for one file content copy into another file:

```
source=open("a.txt","r")

destination=open("b.txt","w")

for line in source:

        destination.write(line)

source. close()

destination.close()
```

Output:

Input a.txt	Output b.txt
A file stores related data, information, settings or commands in secondary storage device like magnetic disk, magnetic tape, optical disk, flash memory	A file stores related data, information, settings or commands in secondary storage device like magnetic disk, magnetic tape, optical disk, flash memory

3. Write a program to count number of lines, words and characters in a text file:

```
fp = open("a.txt","r")
line =0
word = 0
character = 0
for line in fp:
        words = line . split ( )
        line = line + 1
        word = word + len(words)
        character = character +len(line)
print("Number of line", line)
print("Number of words", word)
print("Number of character", character)
```

Output:

```
Number of line=5
Number of words=15
Number of character=47
```

4. ERRORS,EXCEPTION HANDLING

Errors
- Error is a mistake in python also referred as bugs .they are almost always the fault of the programmer.
- The process of finding and eliminating errors is called debugging
    - **Types of errors**
        - Syntax error or compile time error
        - Run time error
        - Logical error

Syntax errors
- Syntax errors are the errors which are displayed when the programmer do mistakes when writing a program, when a program has syntax errors it will not get executed
    - ✓ Leaving out a keyword
    - ✓ Leaving out a symbol, such as colon, comma, brackets
    - ✓ Misspelling a keyword
    - ✓ Incorrect indentation

Runtime errors
- If a program is syntactically correct-that is ,free of syntax errors-it will be run by the python interpreter
- However, the program may exit unexpectedly during execution if it encounters a runtime error.
- When a program has runtime error it will get executed but it will not produce output
    - ✓ Division by zero
    - ✓ Performing an operation on incompatible types
    - ✓ Using an identifier which has not been defined
    - ✓ Trying to access a file which doesn't exit

Logical errors
- Logical errors are the most difficult to fix
- They occur when the program runs without crashing but produces incorrect result
    - ✓ Using the wrong variable name
    - ✓ Indenting a blocks to the wrong level
    - ✓ Using integer division instead of floating point division
    - ✓ Getting operator precedence wrong

Exception handling

Exceptions
- An exception is an error that happens during execution of a program. When that Error occurs

Errors in python
- IO Error-If the file cannot be opened.
- Import Error -If python cannot find the module
- Value Error -Raised when a built-in operation or function receives an argument that has the right type but an inappropriate value
- Keyboard Interrupt -Raised when the user hits the interrupt
- EOF Error -Raised when one of the built-in functions (input() or raw_input()) hits an end-of-file condition (EOF) without reading any data

Exception Handling Mechanism
1. try –except
2. try –multiple except
3. try –except-else
4. raise exception
5. try –except-finally

1. Try –Except Statements
- The try and except statements are used to handle runtime errors
    Syntax:
        try :
            statements
        except :
            statements

The try statement works as follows:-
- ✓ First, the try clause (the statement(s) between the try and except keywords) is executed.
- ✓ If no exception occurs, the except clause is skipped and execution of the try statement is finished.
- ✓ If an exception occurs during execution of the try clause, the rest of the clause is skipped. Then if its type matches the exception named after the except keyword, the except clause is executed, and then execution continues after the try statement.

Example:
        X=int(input("Enter the value of X"))
        Y=int(input("Enter the value of Y"))
        try:
            result = X / (X – Y)
            print("result=".result)
        except ZeroDivisionError:
            print("Division by Zero")

Output:1	Output : 2
Enter the value of X = 10	Enter the value of X = 10
Enter the value of Y = 5	Enter the value of Y = 10
Result = 2	Division by Zero

2. Try – Multiple except Statements
- o Exception type must be different for except statements
    Syntax:
        try:
            statements
        except errors1:
            statements
        except errors2:
            statements
        except errors3:
            statements

Example

```
X=int(input("Enter the value of X"))
Y=int(input("Enter the value of y"))
try:
        sum = X + Y
        divide = X / Y
        print (" Sum of %d and %d = %d", %(X,Y,sum))
        print (" Division of %d and %d = %d", %(X,Y,divide))
except NameError:
        print(" The input must be number")
except ZeroDivisionError:
        print("Division by Zero")
```

Output:1	Output 2:	Output 3:
Enter the value of X = 10	Enter the value of X = 10	Enter the value of X = 10
Enter the value of Y = 5	Enter the value of Y = 0	Enter the value of Y = a
Sum of 10 and 5 = 15	Sum of 10 and 0 = 10	The input must be number
Division of 10 and 5 = 2	Division by Zero	

3. Try –Except-Else

- o The else part will be executed only if the try block does not raise the exception.

- o Python will try to process all the statements inside try block. If value error occur, the flow of control will immediately pass to the except block and remaining statements in try block will be skipped.

Syntax:
```
try:
        statements
except:
        statements
else:
        statements
```

Example

```
X=int(input("Enter the value of X"))
Y=int(input("Enter the value of Y"))
try:
        result = X / ( X – Y )
except ZeroDivisionError:
        print("Division by Zero")
else:
        print("result=".result)
```

Output:1	Output : 2
Enter the value of X = 10	Enter the value of X = 10
Enter the value of Y = 5	Enter the value of Y = 10
Result = 2	Division by Zero

4. Raise statement

- The raise statement allows the programmer to force a specified exception to occur.
  Example:
  >>> **raise NameError**('HiThere')
  Output:
  Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  NameError: HiThere
  ✓ If you need to determine whether an exception was raised but don't intend to handle it, a simpler form of the raise statement allows you to **re-raise** the exception:
  Example
  try:
  ...   raise NameError('HiThere')
  ... except NameError:
  ...   print('An exception flew by!')
  ...   raise
  Output:
  An exception flew by! Traceback
  (most recent call last):
   File "<stdin>", line 2, in <module>
  NameError: HiThere

5. Try –Except-Finally
  ✓ A finally clause is always executed before leaving the try statement, whether an exception has occurred or not.
  ✓ The finally clause is also executed "on the way out" when any other clause of the try statement is left via a break, continue or return statement.
  Syntax
  try:
      statements
  except:
      statements
  finally:
      statements
  Example
  X=int(input("Enter the value of X"))
  Y=int(input("Enter the value of Y"))
  try:
      result = X / (X – Y)
  except Zero DivisionError:
      print("Division by Zero")
  else:
      print("result=".result)
  finally:
      print ("executing finally clause")

Output:1	Output : 2
Enter the value of X = 10	Enter the value of X = 10
Enter the value of Y = 5	Enter the value of Y = 10
Result = 2	Division by Zero
executing finally clause	executing finally clause

5. MODULES IN PYTHON

- A python module is a file that consists of python definition and statements. A module can define functions, classes and variables.
- It allows us to logically arrange related code and makes the code easier to understand and use.

1.Import statement:

- An import statement is used to import python module in some python source file.

Syntax: import module1 [, module2 [...module]]

Example:

       >>>import math
       >>>print (math.pi)
           3.14159265

2.Import with renaming:

       The import a module by renaming it as follows,
       >>>import math as a
       >>>print("The value of pi is ",a.pi)
           The value of pi is 3.14159265

Writing modules:

- Any python source code file can be imported as a module into another python source file. For example, consider the following code named as support.py, which is python source file defining two function add(), display().

Support.py:

```
def add(a,b):

    print("The result is ",a+b)

    return

def display(p):

     print("welcome ",p)

     return
```

       The support.py file can be imported as a module into another python source file and its functions can be called from the new files as shown in the following code:

3. Import file name

```
import support          #import module support

support.add(3,4)        #calling add() of support module with two integers

support.add (3.5,4.7) #calling add() of support module with two real values

support.add ('a','b')   #calling add() of support module with two character values

support.add ("yona","alex")#calling add() of support module with two string values

support.display ('fleming')     #calling display() of support module with a string value
```

Output:

The result is 7
The result is 8.2
The result is ab
The result is yonaalex
Welcome, fleming

4. from……import statement:

✓ It allows us to import specific attributes from a module into the current namespace.

Syntax: from modulename import name1 [, name2[,……nameN]]

from support import add   #import module support

support.add(3,4)   #calling add() of support module with two integers

support.add(3.5,4.7)   #calling add() of support module with two real values

support.add('a','b')   #calling add() of support module with two character values

support.add ("yona","alex")#calling add() of support module with two string values

support.display ('fleming')   #calling display() of support module with a string value

Output:

The result is 7
The result is 8.2
The result is ab
The result is yonaalex
Welcome, fleming

5.OS Module

✓ The OS module in python provide function for interacting with operating system
✓ To access the OS module have to import the OS module in our program

import os

method	example	description
name	Osname 'nt'	This function gives the name of the operating system
getcwd()	Os,getcwd() ,C;\\Python34'	Return the current working directory(CWD)of the file used to execute the code
mkdir(folder)	Os.mkdir("python")	Create a directory(folder) with the given name
rename(oldname,newname)	Os.rename("python","pspp")	Rename the directory or folder
remove("folder")	Os.remove("pspp")	Remove (delete)the directory or folder

| getuid() | Os.getuid() | Return the current process's user id |
| environ | Os.nviron | Get the users environment |

6.Sys Module

- ✓ Sys module provides information about constant, function and methods
- ✓ It provides access to some variables used or maintained by the interpreter

import sys

methods	example	description
sys.argv	sys.argv	Provides the list of command line arguments passed to a python script
	sys.argv(0)	Provides to access the file name
	sys.argv(1)	Provides to access the first input
sys.path	sys.path	It provide the search path for module
sys.path.append()	sys.path.append()	Provide the access to specific path to our program
sys.platform	sys.platform 'win32'	Provide information about the operating system platform
sys.exit	sys.exit <built.in function exit>	Exit from python

Steps to Create the Own Module

- ✓ Here we are going to create a calc module ; our module contains four functions

i.e add(),sub(),mul(),div()

Program for calculator module	output

Module name ;calc.py
def add(a,b);
print(a+b)
def sub(a,b);
  print(a-b)
def mul(a,b);
  print(a*b)
 def div(a,b);
  print(a/b)
	import calculator
	calculator.add(2,3)
	
	Outcome
	>>>5

6. PACKAGES IN PYTHON

- ✓ A package is a collection of python module. Module is a single python file containing function definitions
- ✓ A package is a directory(folder)of python module containing an additional init py file, to differentiate a package from a directory
- ✓ Packages can be nested to any depth, provided that the corresponding directories contain their own init py file.
- ✓ __init py file is a directory indicates to the python interpreter that the directory should be treated like a python package init py is used to initialize the python package

Steps to Create a Package

Step1: create the package directory

- ✓ Create the directory (folder)and give it your packages name
- ✓ Here the package name is calculator

Name	Data modified	Type
1. pycache___	05-12-2017	File folder
2.calculater	**08-12-2017**	**File folder**
3. DLLs	10-12-2017	File folder

Step2: write module for calculator directory add save the module in calculator directory

- ✓ Here four module have create for calculator directory

Local Disk (C)>Python34>Calculator

Name	Data modified	Type	Size
1. add	08-12-2017	File folder	1KB
2. div	08-12-2017	File folder	1KB
3. mul	08-12-2017	File folder	1KB
4. sub	08-12-2017	File folder	1KB

add.py	div.py	mul.py	sub.py
def add(a,b); print(a+b)	def div(a,b); print(a/b)	def mul(a,b); print(a*b)	def sub(a,b); print(a-b)

Step3: add the __init__.py file in the calculator directory

- ✓ A directory must contain the file named init__.py in order for python to consider it as a package

Add the following code in the_init_.py file

```
from * add import add
from * sub import sub
from * mul import mul
from * div import div
```

Local Disk (C):/Python34>Calculator

Name	Data modified	Type	Size
1. init_____	08-12-2017	File folder	1KB
2. add	08-12-2017	File folder	1KB
3. div	08-12-2017	File folder	1KB
4. mul	08-12-2017	File folder	1KB
5. sub	08-12-2017	File folder	1KB

Step4: To test your package

✓ Import calculator package in your program and add the path of your package in your program by using sys.path.append()

Example

```
import calculator
importsys
sys.path.append("C:/Python34")
print ( calculator.add(10,5))
print ( calculator.sub(10,5))
print ( calculator.mul(10,5))
print ( calculator.div(10,5))
```

Output :

```
>>> 15
    5
    50
    2
```

Two marks:

1. Why do we go for file?

File can a persistent object in a computer. When an object or state is created and needs to be persistent, it is saved in a non-volatile storage location, like a hard drive.

2. What are the three different mode of operations of a file?

The three mode of operations of a file are,

    i.   Open – to open a file to perform file operations

    ii.   Read – to open a file in read mode

    iii.   Write – to open a file in write mode

3. State difference between read and write in file operations.

Read	Write
A "Read" operation occurs when a computer program reads information from a computer file/table (e.g. to be displayed on a screen). The "read" operation gets information out of a file.	A "Write" operation occurs when a computer program adds new information, or changes existing information in a computer file/table.
After a "read", the information from the file/table is available to the computer program but none of the information that was read from the file/table is changed in any way.	After a "write", the information from the file/table is available to the computer program but the information that was read from the file/table can be changed in any way.

4. Differentiate error and exception.

Errors

- Error is a mistake in python also referred as bugs .they are almost always the fault of the programmer.
- The process of finding and eliminating errors is called debugging
  - Types of errors
  - Syntax error or compile time error
  - Run time error
  - Logical error

Exceptions

An exception is an error that happens during execution of a program. When that Error occurs

5. Give the methods of exception handling.

1. try –except
2. try –multiple except
3. try –except-else
4. raise exception
5. try –except-finally

6. State the syntax for try…except block

The try and except statements are used to handle runtime errors

Syntax:

```
try :
    statements
except:
    statements
```

7. Write a program to add some content to existing file without effecting the existing content.

file=open("newfile.txt",'a)

file.write("hello")

newfile.txt	newfile.txt(after updating)
Hello!!World!!!	Hello!!!World!!!hello

8. What is package?

- A package is a collection of python module. Module is a single python file containing function definitions
- A package is a directory(folder)of python module containing an additional_ init_ py file, to differentiate a package from a directory
- Packages can be nested to any depth, provided that the corresponding directories contain their own __init py file

9. What is module?

A python module is a file that consists of python definition and statements. A module can define functions, classes and variables. ☐ It allows us to logically arrange related code and makes the code easier to understand and use.

10. Write the snippet to find the current working directory.

```
Import os
print(os.getcwd))
```

Output:

C:\\Users\\Mano\\Desktop

11. Give the use of 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:

>>> x = 52

>>> fout.write(str(x))

An alternative is to use the format operator, %. When applied to integers, % is the modulus operator. But when the first operand is a string, % is 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. For example, the format sequence '%d' means that the second operand should be formatted as an integer (d stands for "decimal"):

>>> camels = 42

>>>'%d' % camels '42'

The result is the string '42', which is not to be confused with the integer value 42.

12. Write the snippet to find the absolute path of a file.

import os

os.path.abspath('w

rite.py')

Output:

'C:\\Users\\Mano\\Desktop\\write.py'

13. What is the use of os.path.isdir() function.

os.path.isdir() is a function defined in the package os. The main function of isdir("some input") function is to check whether the passed parameter is directory or not. isdir() function will only **return** only **true or false.**

14. What is the use of os.path.isfile() function.

os.path.isfile () is a function defined in the package os. The main function of isfile ("some input") function is to check whether the passed parameter is file or not. isfile () function will only **return** only **true or false**.

15. What is command line argument?

sys.argv is the list of command line arguments passed to the Python program. Argv represents all the items that come along via the command line input, it's basically an array holding the command line arguments of our program.