

GE8151 PROBLEM SOLVING AND PYTHON PROGRAMMING

SYLLABUS

OBJECTIVES:

- To know the basics of algorithmic problem solving
- To read and write simple Python programs.
- To develop Python programs with conditionals and loops.
- To define Python functions and call them.
- To use Python data structures — lists, tuples, dictionaries.
- To do input/output with files in Python.

UNIT I ALGORITHMIC PROBLEM SOLVING

9

Algorithms, building blocks of algorithms (statements, state, control flow, functions), notation (pseudo code, flow chart, programming language), algorithmic problem solving, simple strategies for developing algorithms (iteration, recursion). Illustrative problems: find minimum in a list, insert a card in a list of sorted cards, guess an integer number in a range, Towers of Hanoi.

UNIT II DATA, EXPRESSIONS, STATEMENTS

9

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.

UNIT III CONTROLFLOW ,FUNCTIONS

9

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

UNIT IV LISTS,TUPLES,DICTIONARIES

9

Lists: list operations, list slices, list methods, list loop, mutability, aliasing, cloning lists, list parameters; Tuples: tuple assignment, tuple as return value; Dictionaries: operations and methods; advanced list processing - list comprehension; Illustrative programs: selection sort, insertion sort, mergesort, histogram.

UNIT V FILES,MODULES,PACKAGES

9

Files and exception: text files, reading and writing files, format operator; command line arguments, errors and exceptions, handling exceptions, modules, packages; Illustrative programs: word count, copy file.

TOTAL : 45 PERIODS

TEXT BOOKS

Allen B. Downey, ``Think Python: How to Think Like a Computer Scientist``, 2nd edition, Updated for Python 3, Shroff/O'Reilly Publishers, 2016 (<http://greenteapress.com/wp/thinkpython/>)
Guido van Rossum and Fred L. Drake Jr, —An Introduction to Python – Revised and updated for Python 3.2, Network Theory Ltd., 2011.

REFERENCES:

John V Guttag, —Introduction to Computation and Programming Using Python``, Revised and expanded Edition, MIT Press , 2013

Robert Sedgewick, Kevin Wayne, Robert Dondero, —Introduction to Programming in Python: An Inter-disciplinary Approach, Pearson India Education Services Pvt. Ltd., 2016.

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 $\text{sum}=a+b$
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. 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 or 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 offunction
2. Body consisting local declaration and statements
3. Formal parameter
4. Optional result type.

## Basic Syntax

```
function_name(parameters)
```

function statements

end function

### Algorithm for addition of two numbers using function

## Main function()

### Step 1: Start

**Step 2:** Call the function `add()`

### Step 3: Stop

**sub function add()**

### Step 1: Function start

### Step2: Get a, b Values

**Step 3:** add  $c=a+b$

**Step 4: Printc**

**Step 5: Return**

## **2. Notations of 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.

Pseudocode must never be 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




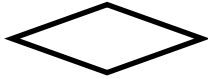
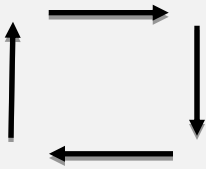
- It is not visual
- There is no standardized style or format

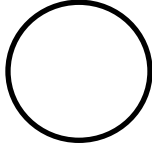
### Flowchart

A **graphical representation** of an algorithm. Flowcharts are 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**

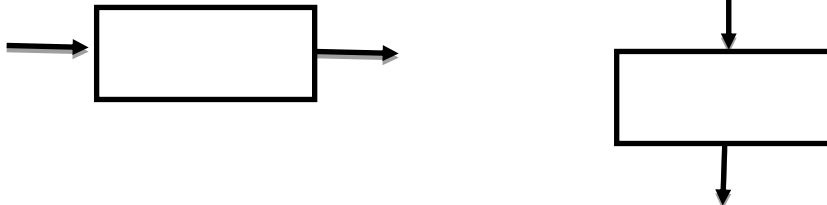
| S.No | Name of symbol       | Symbol                                                                              | Type          | Description                                                                             |
|------|----------------------|-------------------------------------------------------------------------------------|---------------|-----------------------------------------------------------------------------------------|
| 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. <b>Used to connect symbols.</b> |

|    |           |                                                                                   |        |                                                                                                                                                                     |
|----|-----------|-----------------------------------------------------------------------------------|--------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 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 <b>used to connect flowcharts</b> . |
|----|-----------|-----------------------------------------------------------------------------------|--------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|

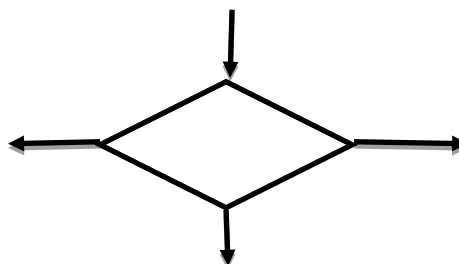
### Rules for drawing flowchart

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.

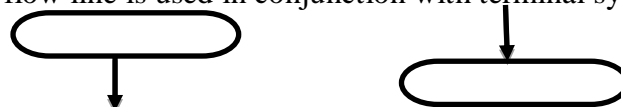
Only one flow line should come out from a process symbol.



4. Only one flow line should enter a decision symbol, but two or three flow lines, one for each possible answer, can 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.

### Advantages of Flowchart

#### **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

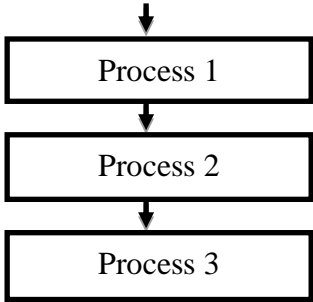
**Complex Logic:** Sometimes, the program logic is quite complicated. In that case flowchart becomes complex and difficult to use.

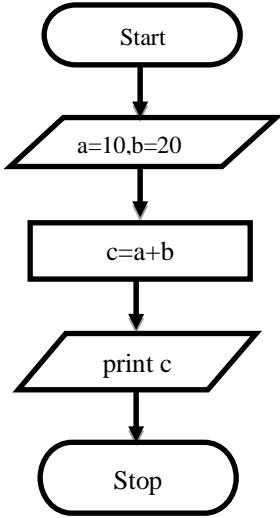
**Alteration and Modification:** If alterations are required the flowchart may require re-drawing completely.

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

### Control Structures using flowcharts and Pseudocode

#### Sequence Structure

| Pseudocode                                         | Flow Chart                                                                          |
|----------------------------------------------------|-------------------------------------------------------------------------------------|
| <b>General Structure</b>                           |                                                                                     |
| Process 1<br>....<br>Process 2<br>...<br>Process 3 |  |
| <b>Example</b>                                     |                                                                                     |

|                                               |                                                                                      |
|-----------------------------------------------|--------------------------------------------------------------------------------------|
| READ a<br>READ b<br>Result $c=a+b$<br>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                                                                                                                                                                                                                                                                         |
|-------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>General Structure</b>                              |                                                                                                                                                                                                                                                                                    |
| IF condition THEN<br>Process 1<br>ENDIF               | <pre> graph TD     Entry(( )) --&gt; Decision{if(condition)}     Decision -- Yes --&gt; Process1[Process 1]     Decision -- No --&gt; Process2[Process 2]           </pre>                                                                                                         |
| <b>Example</b>                                        |                                                                                                                                                                                                                                                                                    |
| READ a<br>READ b<br>IF a>b THEN<br>PRINT a is greater | <pre> graph TD     Start([Start]) --&gt; Input[/a=10,b=20/]     Input --&gt; Decision{if (a&gt;b)}     Decision -- Yes --&gt; Output[/Print a is greater/]     Decision -- No --&gt; Connector(( ))     Output --&gt; Connector     Connector --&gt; Stop([Stop])           </pre> |

## IF... ELSE

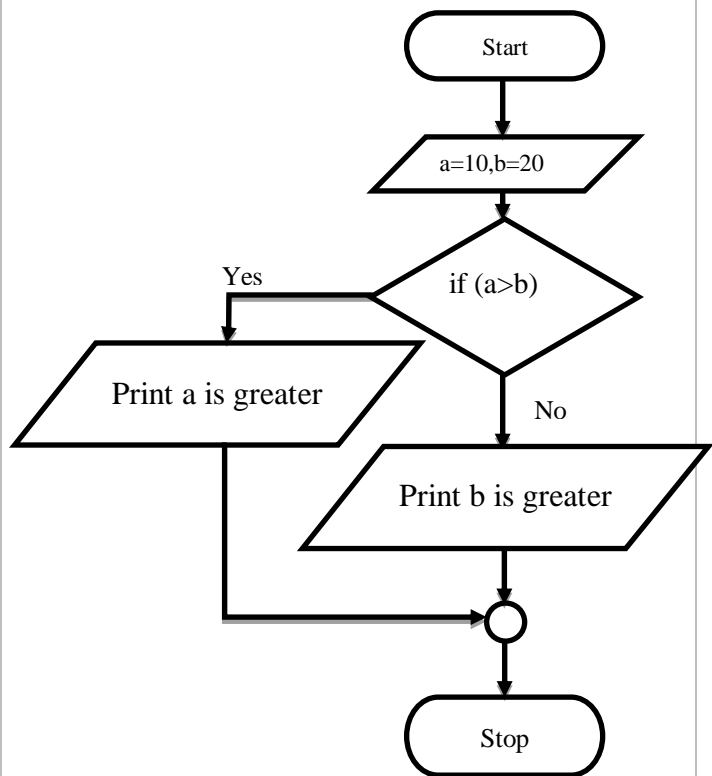
IF...THEN...ELSE is the structure used to specify, if the condition is true, then execute Process1, else, that is condition is false then execute Process2

| Pseudocode                                                   | Flow Chart                                                                                                                                                                 |
|--------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>General Structure</b>                                     |                                                                                                                                                                            |
| IF condition THEN<br>Process 1<br>ELSE<br>Process 2<br>ENDIF | <pre> graph TD     Entry(( )) --&gt; Decision{if(condition)}     Decision -- Yes --&gt; Process1[Process 1]     Decision -- No --&gt; Process2[Process 2]           </pre> |
| <b>Example</b>                                               |                                                                                                                                                                            |

```

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

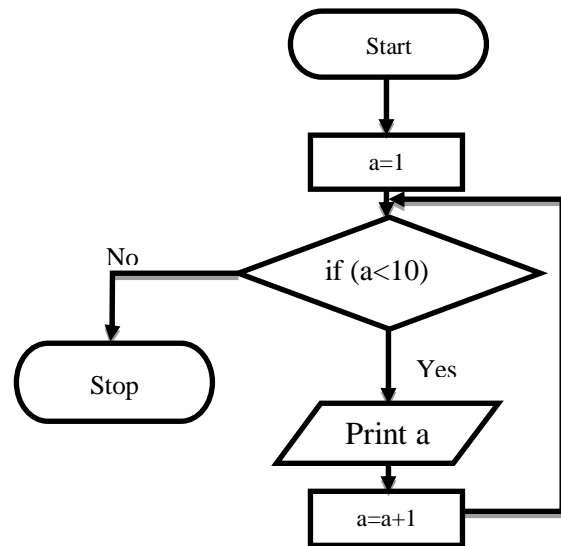
| Pseudocode                                      | Flow Chart                                                                                                                                                                                                   |
|-------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>General Structure</b>                        |                                                                                                                                                                                                              |
| WHILE condition<br>Body of the loop<br>ENDWHILE | <pre> graph TD     Entry(( )) --&gt; Decision{if(condition)}     Decision -- Yes --&gt; Body[Body of the loop]     Body --&gt; Entry     Decision -- No --&gt; Exit(( ))     Exit --&gt; ExitOut(( )) </pre> |
| <b>Example</b>                                  |                                                                                                                                                                                                              |

- **DO...WHILE** is exit checked loop, so the loop will be executed at least once.

```

INITLIZE a=1
WHILE a<10 THEN
 PRINT a
 a=a+1
ENDWHILE

```



- In python DO...WHILE is not supported.
- 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=Algoirthm+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 Programing 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.

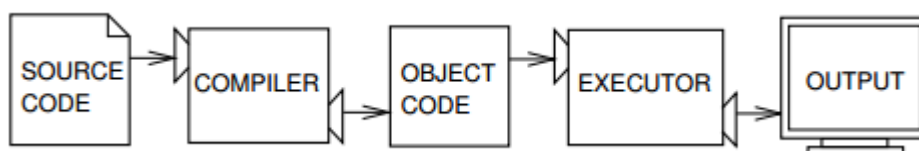


Figure: Compiler

### 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.

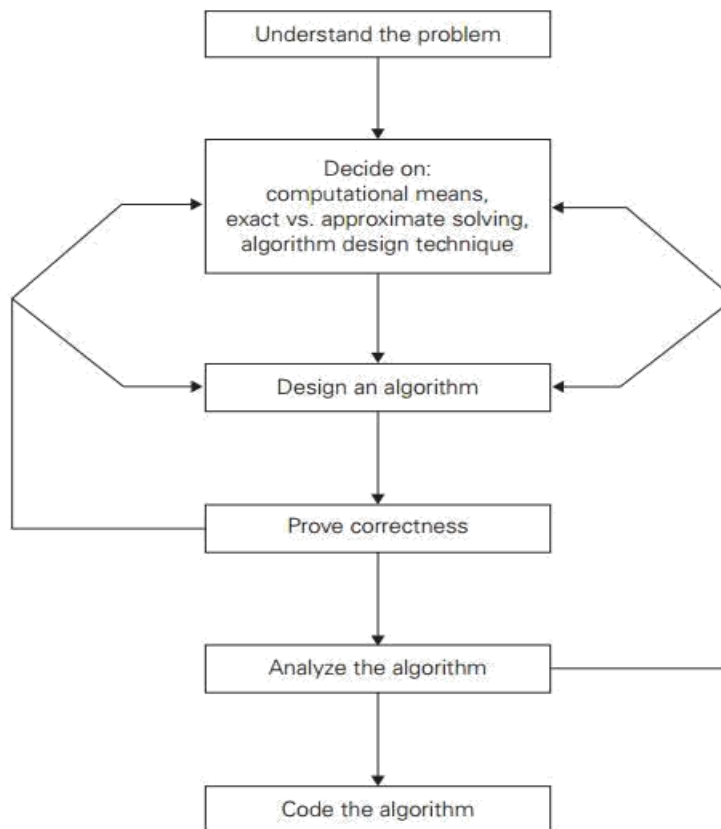


FIGURE 1.2 Algorithm design and analysis process.

#### 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

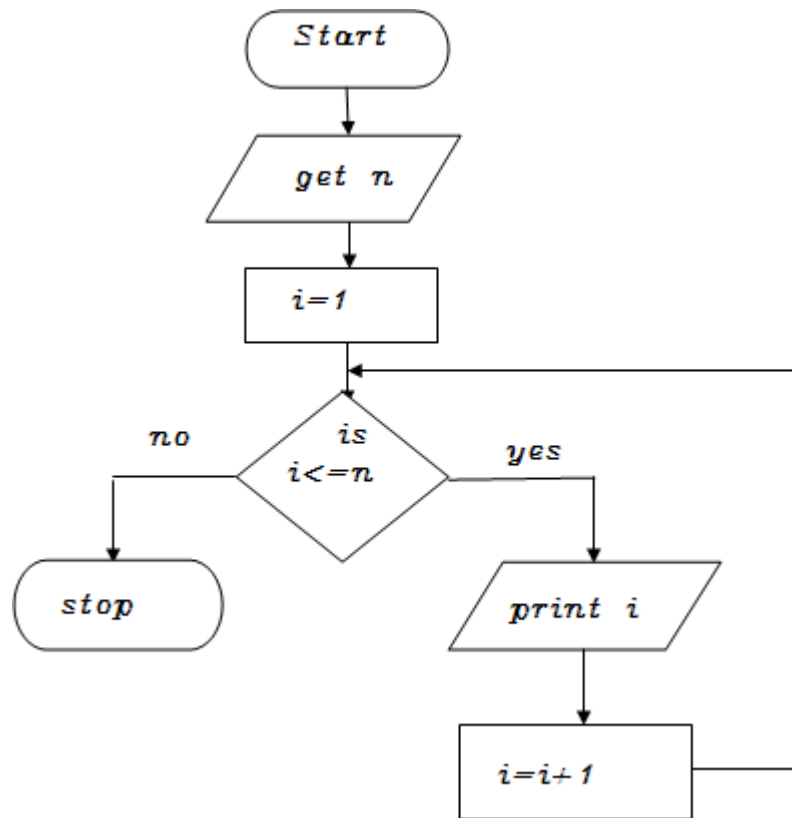
### **Iteration**

- 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

| <u><b>Syntax for For:</b></u>                                                      | <u><b>Example: Print n natural numbers</b></u>                                            |
|------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------|
| FOR( <i>start-value</i> to <i>end-value</i> ) DO<br><i>statement</i><br>... ENDFOR | BEGIN<br>GET n<br>INITIALIZE i=1<br>FOR (i<=n) DO<br>PRINT i<br>i=i+1<br>ENDFOR<br>END    |
| <u><b>Syntax for While:</b></u>                                                    | <u><b>Example: Print n natural numbers</b></u>                                            |
| WHILE (condition) DO<br>statement<br>...<br>ENDWHILE                               | BEGIN<br>GET n<br>INITIALIZE i=1<br>WHILE(i<=n) DO<br>PRINT i<br>i=i+1<br>ENDWHILE<br>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:**

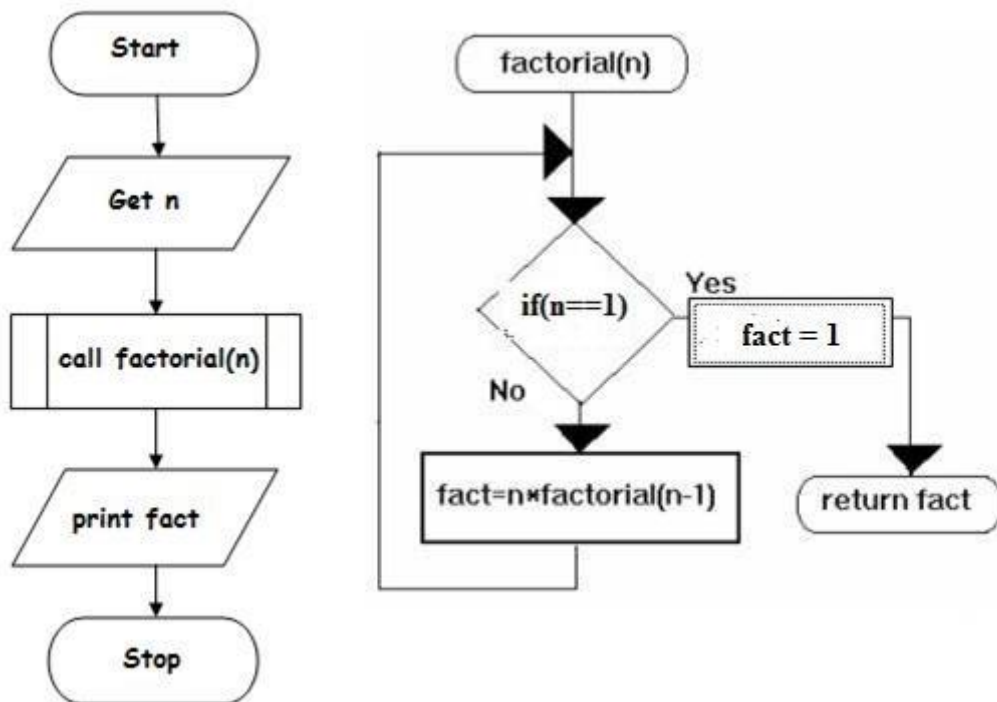
- Step1: Start
- Step2: Get n
- Step3: call factorial(n)
- Step4: print fact
- Step5: Stop

#### **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

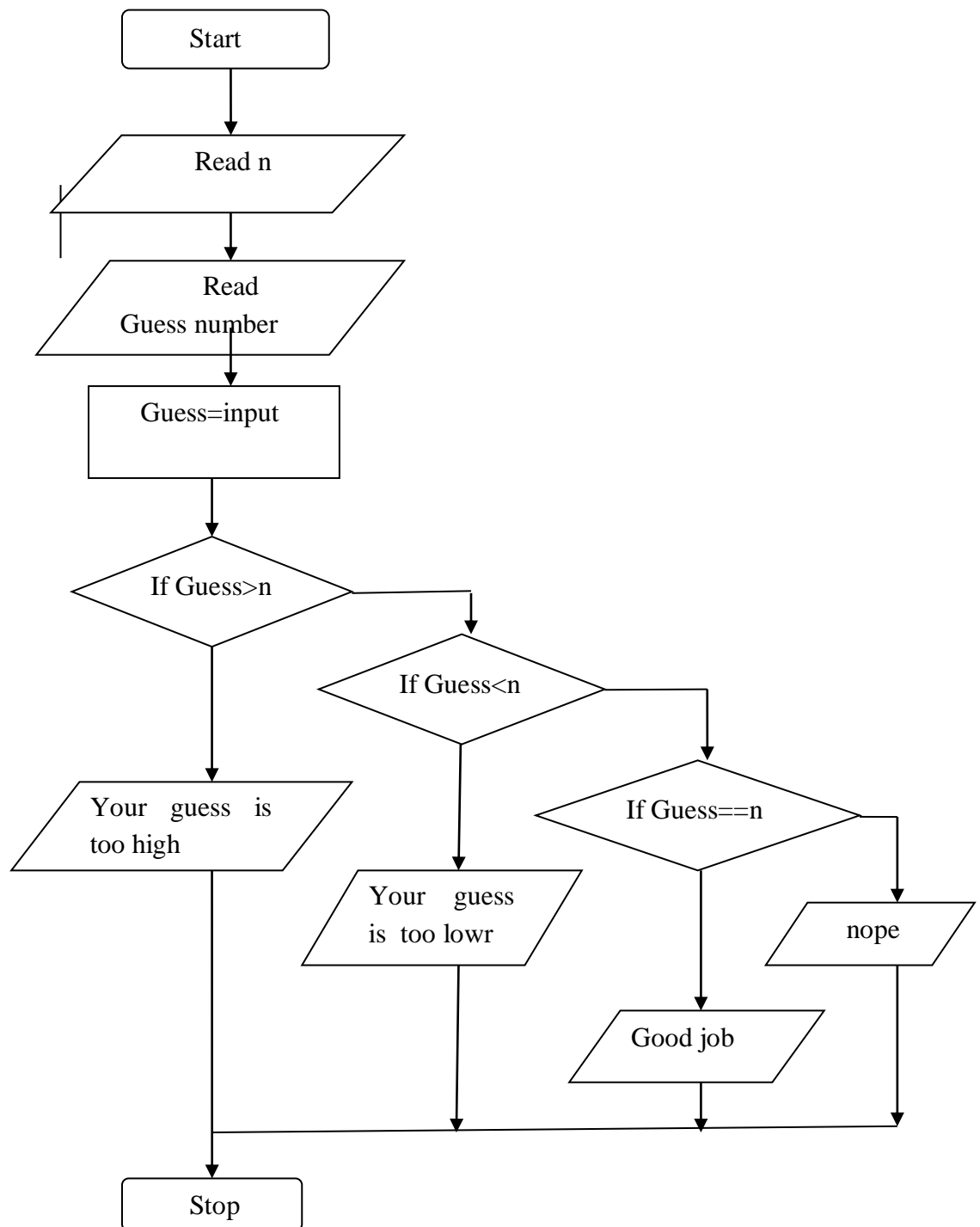
### Algorithm:

```
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

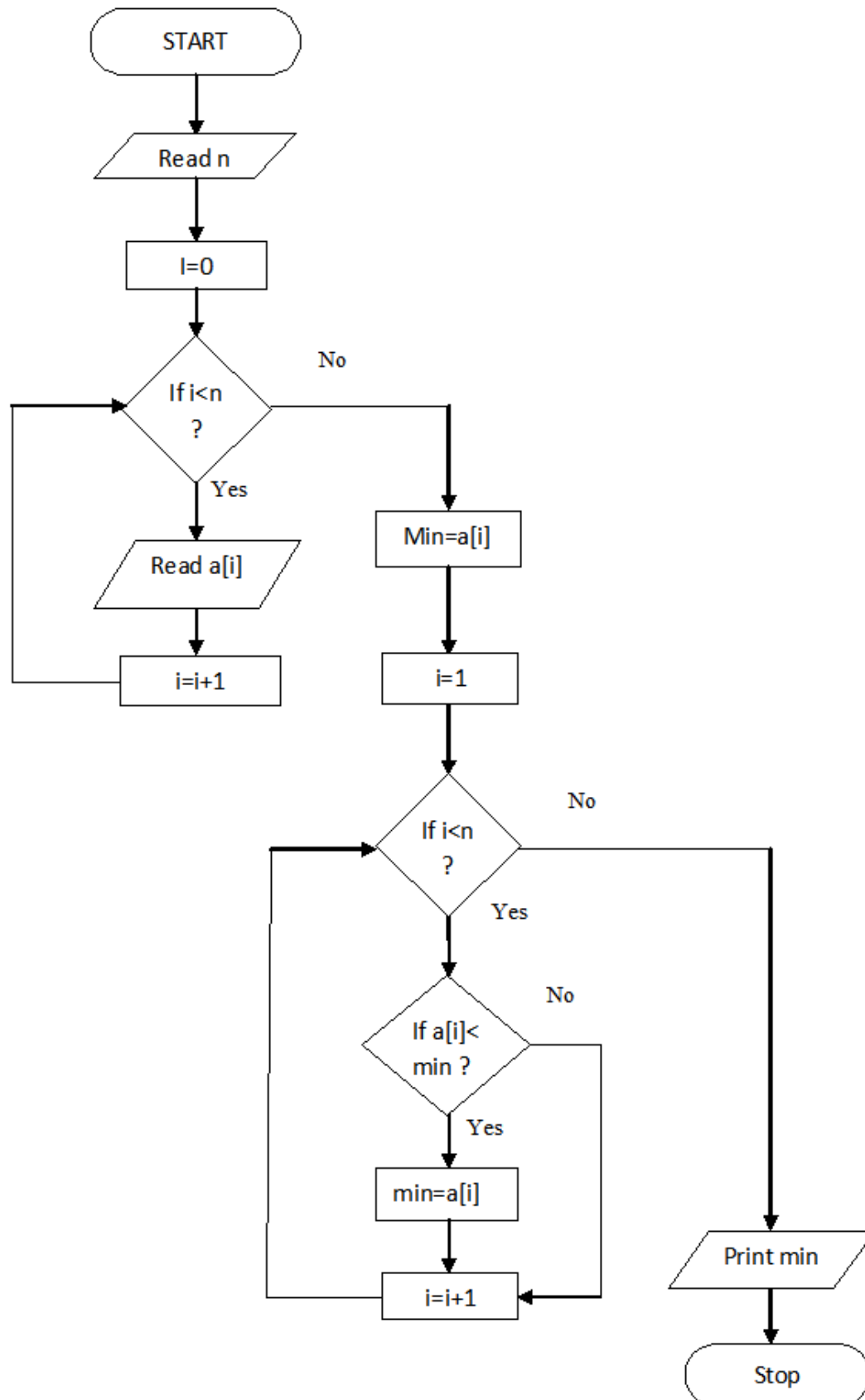
### Algorithm:

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  
Step 4.1: Read  $a[i]$   
Step 4.2:  $i=i+1$  goto step 4  
Step 5: Compute  $\text{min}=a[0]$   
Step 6: Initialize  $i=1$   
Step 7: If  $i < n$ , then go to step 8 else goto step 10  
Step 8: If  $a[i] < \text{min}$ , then goto step 8.1, 8.2 else goto 8.2  
Step 8.1:  $\text{min}=a[i]$   
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

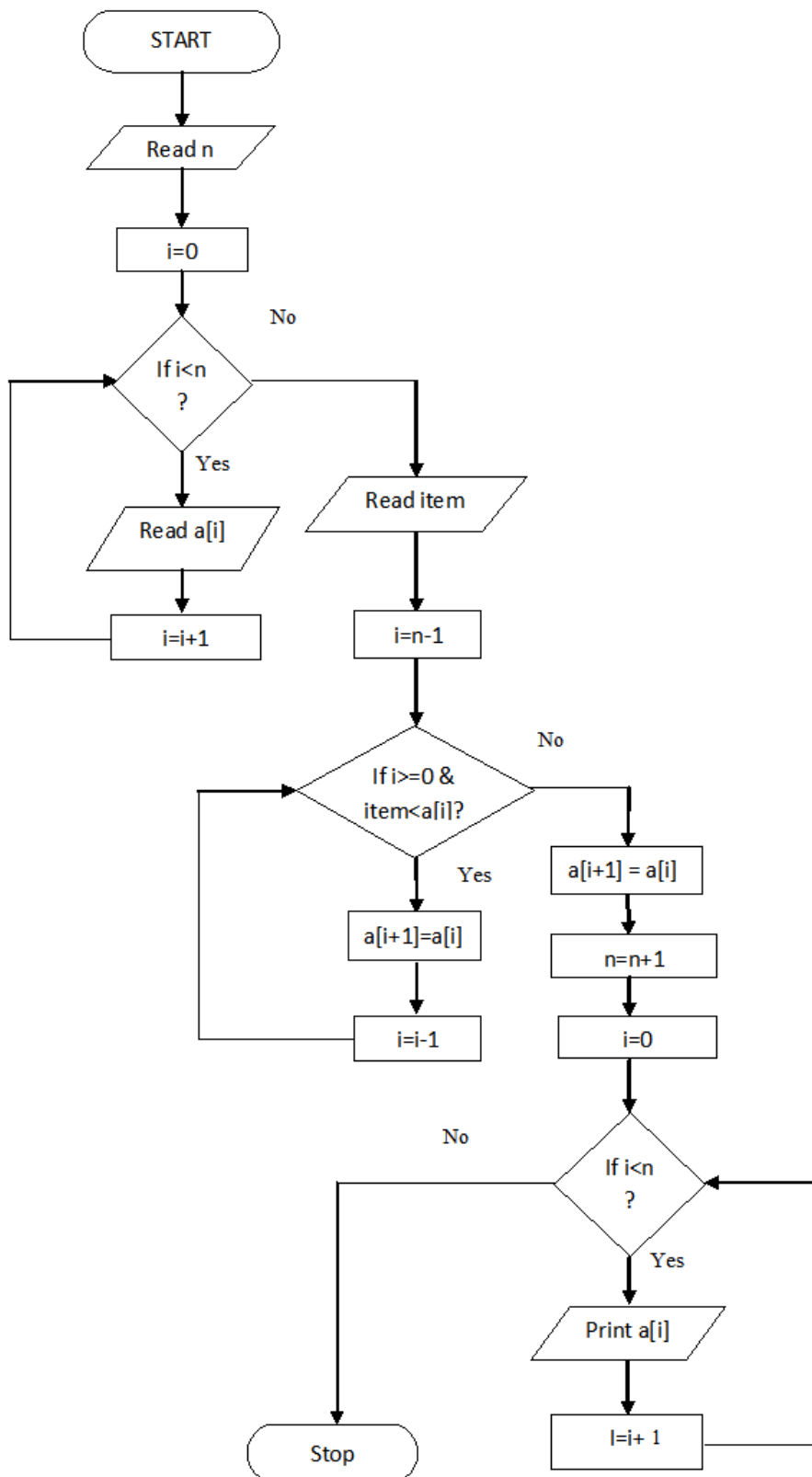
#### Algorithm:

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  
Step 4.1: 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  $i \geq 0$  and  $\text{item} < a[i]$ , 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  $a[i+1]=\text{item}$   
Step 9: Compute  $n=n+1$   
Step 10: If  $i < n$ , then goto step 10.1, 10.2 else goto step 11  
Step 10.1: Print  $a[i]$   
Step 10.2:  $i=i+1$  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:



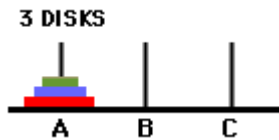
#### 4. Tower of Hanoi

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  $\rightarrow 1$  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.

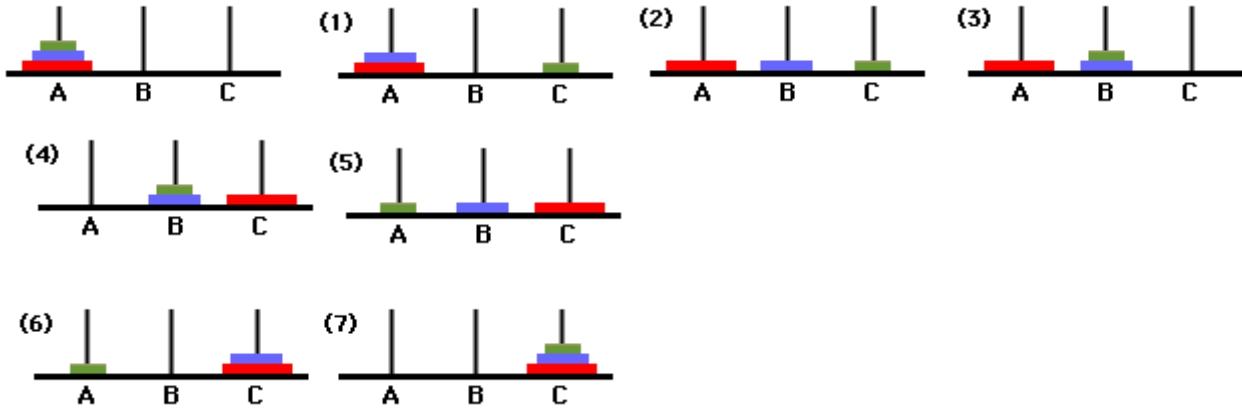
##### Input: more than two disks

- 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 ( $n^{\text{th}}$  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 (n-1) 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

**3 DISKS**



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

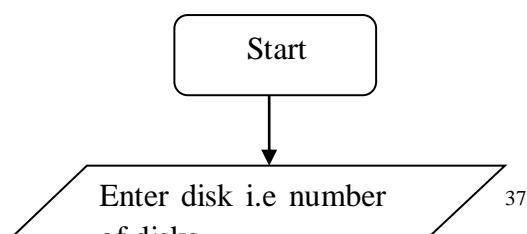
    Hanoi(disk - 1, aux, dest, source) // Step 3

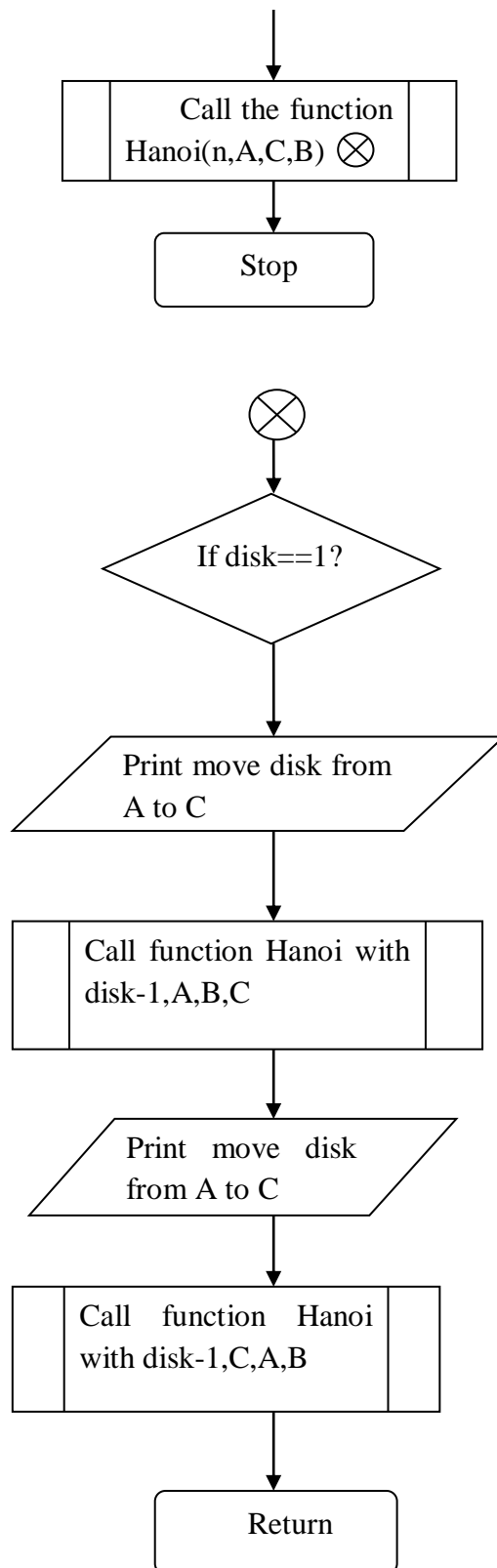
END IF

END Procedure

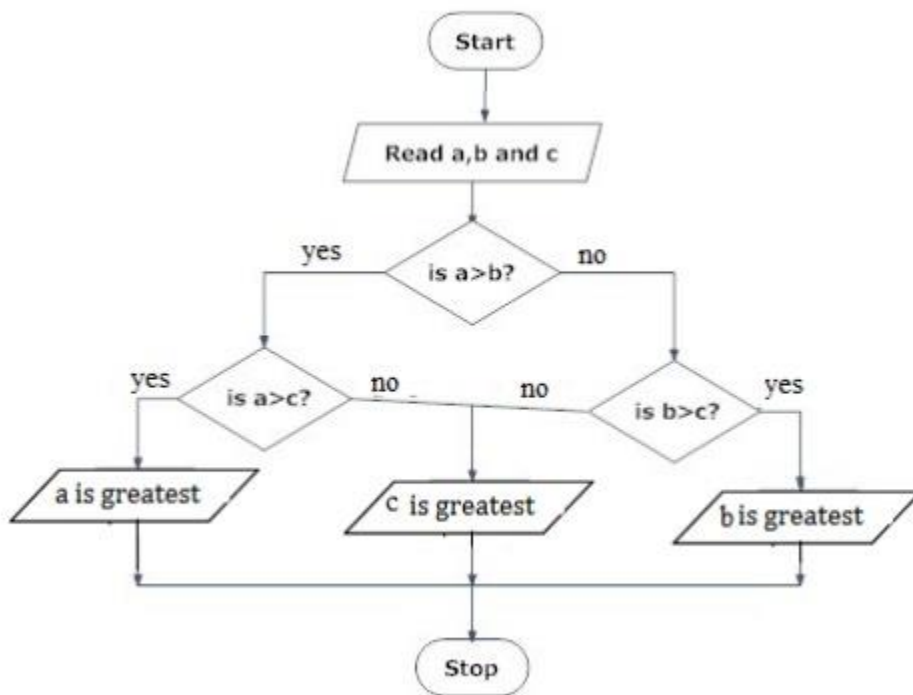
STOP

FLOW CHART

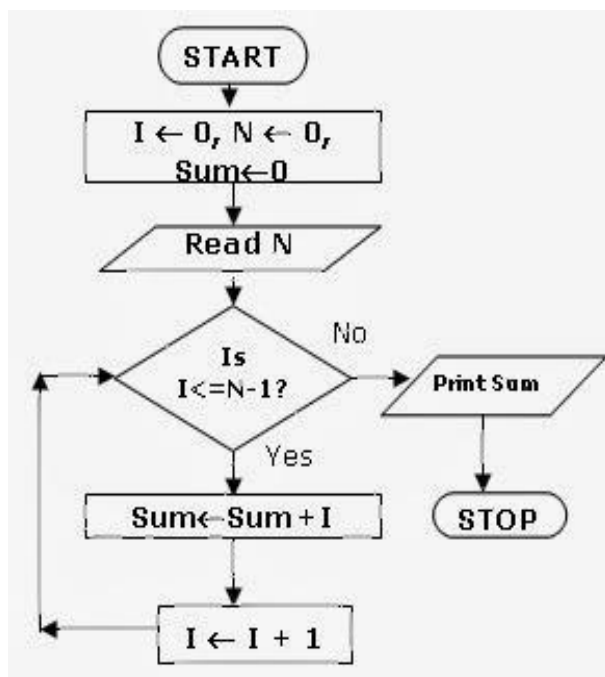




5. Draw a flow chart to find greatest among three numbers. (AU 2018)



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. | <b>Systematic logical approach</b> which is a well-defined, step-by-step procedure that allows a computer to solve a problem.                                                                 | It is exact code written for problem <b>following all the rules of the programming language</b> . |
| 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.<br><br><b>Program=Algoirthm+Data</b>     |

5. Define Flow chart.

A **graphical representation** of an algorithm. Flowcharts 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.

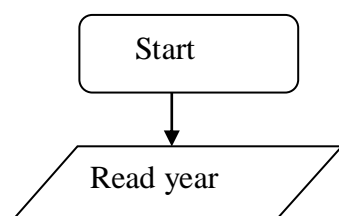
9. What are the steps for developing algorithms.

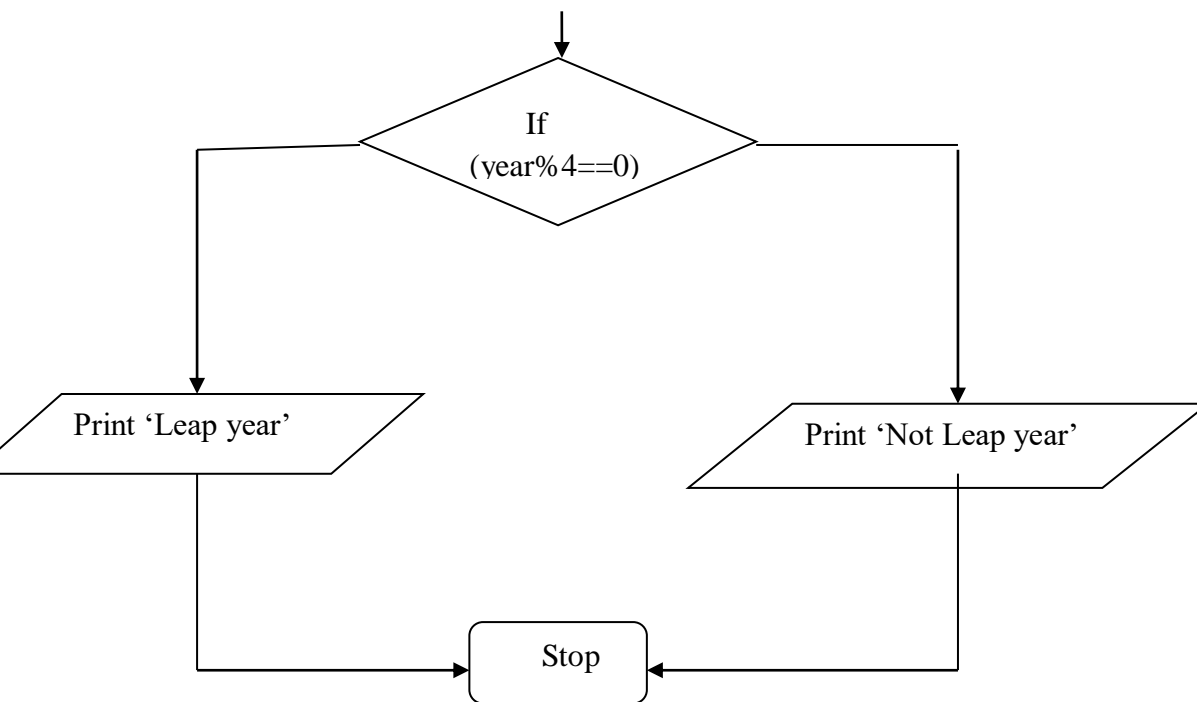
- Problem definition
- Development of a model
- Specification of Algorithm
- Designing an Algorithm
- Checking the correctness of Algorithm
- Analysis of Algorithm
- Implementation of Algorithm
- Program testing
- Documentation Preparation

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.





## 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 twopoints.

## 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.

### 1 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.

### 2 Applications:

- ❖ Bit Torrent file sharing
- ❖ Google search engine, YouTube
- ❖ Intel, Cisco, HP, IBM
- ❖ i-Robot
- ❖ NASA

### 3 Python interpreter:

**Interpreter:** To execute a program in a high-level language by translating it one line at a time.

**Compiler:** To translate a program written in a high-level language into a low-level language all at once, in preparation for later execution.

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

### 4 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

#### 5 Interactive mode:

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

#### 6 Advantages:

- ❖ Python, in interactive mode, is good enough to learn, experiment or explore.
- ❖ Working in interactive mode is convenient for beginners and for testing small pieces of code.

#### 7 Drawback:

- ❖ We cannot save the statements 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, >>>, is the prompt the interpreter uses** 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.

```

Python 2.7.13 Shell
File Edit Shell Debug Options Window Help
Python 2.7.13 (v2.7.13:a06454b1afa1, Dec 17 2016, 20:53:40) [MSC v.1500 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 2+5
7
>>> 2**6
64
>>> a=3
>>> b=6
>>> c=a-b
>>> print(c)
-3
>>> print("good morning")
good morning
>>> |

```

## 8 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.

### Example:

```

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.                                                                     |

## 9 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.

## 10 Features of IDLE:

- Multi-window text editor with syntax highlighting.

- Auto completion with **smart indentation**.
- Python shell** to display output with syntax highlighting.

## 11 2.VALUES AND DATATYPES

### Value:

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

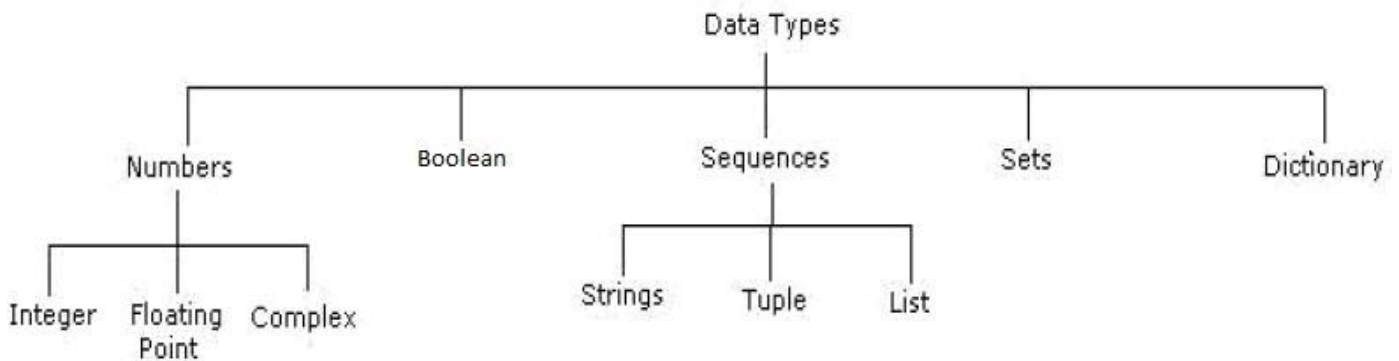
**Eg,** Values are 2, 42.0, and 'Hello, World!'. (These values belong to different datatypes.)

### 12 Data type:

Every value in Python has a data type.

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

### 13 Python has four standard data types:



### Numbers:

- ❖ Number data type stores **NumericalValues**.
- ❖ 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                                                                                                                                                                                                                                                                                                                                |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> <li>- They are often called just integers or <b>int</b>.</li> <li>- They are positive or negative whole numbers with no decimal point.</li> </ul> | <ul style="list-style-type: none"> <li>-They are long integers.</li> <li>-They can also be represented in <u>octal</u> and hexadecimal representation.</li> </ul> | <ul style="list-style-type: none"> <li>-They are written with a decimal point dividing the integer and the fractional parts.</li> </ul> | <ul style="list-style-type: none"> <li>-They are of the form <math>a + bj</math>, where <math>a</math> and <math>b</math> are floats and <math>j</math> represents the square root of -1 (which is an imaginary number).</li> <li>-The real part of the number is <math>a</math>, and the imaginary part is <math>b</math>.</li> </ul> |
| <b>Eg, 56</b>                                                                                                                                                                        | <b>Eg, 5692431L</b>                                                                                                                                               | <b>Eg, 56.778</b>                                                                                                                       | <b>Eg, square root of -1 is a complex number</b>                                                                                                                                                                                                                                                                                       |

### 14 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 series or sequence of characters - letters, numbers, and special characters.
- Strings are marked by quotes:
  - Single quotes(' ') E.g., 'This a string in singlequotes'
  - double quotes(" ") E.g., "This a string in doublequotes"
  - 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.

### 15 Indexing:

|                       |    |    |    |    |    |
|-----------------------|----|----|----|----|----|
| <b>String A</b>       | H  | E  | L  | L  | O  |
| <b>Positive Index</b> | 0  | 1  | 2  | 3  | 4  |
| <b>Negative Index</b> | -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.

### 16 Example: A[0] or A[-5] will display “H”

- Subscript 1 or -ve (n-1) displays the second element.

### 17 Example: A[1] or A[-4] will display “E”

### Operations on string:

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

|                                     |                                            |                                                                                  |
|-------------------------------------|--------------------------------------------|----------------------------------------------------------------------------------|
| <b>Creating a string</b>            | >>> s="good morning"                       | Creating the list with elements of different data types.                         |
| <b>Indexing</b>                     | >>>print(s[2])<br>o<br>>>>print(s[6])<br>O | ❖ Accessing the item in the position 0<br>❖ Accessing the item in the position 2 |
| <b>Slicing( ending position -1)</b> | >>>print(s[2:])<br>od morning              | - Displaying items from 2 <sup>nd</sup> till last.                               |

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

## 18 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 data types.

### Operations on

list: Indexing

Slicing

Concatenation

Repetitions

Updation, Insertion, Deletion

|                                                                                                                                           |                                                                                                 |                                                                                                                                                                                   |
|-------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Creating a list</b>                                                                                                                    | >>> <b>list1=["python", 7.79, 101, "hello"]</b><br>>>> <b>list2=["god", 6.78, 9]</b>            | Creating the list with elements of different data types.                                                                                                                          |
| <b>Indexing</b>                                                                                                                           | >>> <b>print(list1[0])</b> python<br>>>> <b>list1[2]</b><br>101                                 | <ul style="list-style-type: none"> <li>❖ Accessing the item in the position 0</li> <li>❖ Accessing the item in the position 2</li> </ul>                                          |
| <b>Slicing( ending position -1)</b><br><u>Slice operator is used to extract part of a string, or some part of a list</u><br><u>Python</u> | >>> <b>print(list1[1:3])</b><br>[7.79, 101]<br>>>> <b>print(list1[1:])</b> [7.79, 101, 'hello'] | <ul style="list-style-type: none"> <li>- Displaying items from 1<sup>st</sup> till 2<sup>nd</sup>.</li> <li>- Displaying items from 1<sup>st</sup> position till last.</li> </ul> |
| <b>Concatenation</b>                                                                                                                      | >>> <b>print( list1+list2)</b><br>['python', 7.79, 101, 'hello', 'god',                         | - Adding and printing the items of two lists.                                                                                                                                     |

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

## 19 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.

## 20 Basic Operations:

|                                     |                                                                                                |                                                                                                                                          |
|-------------------------------------|------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Creating a tuple</b>             | <pre>&gt;&gt;&gt;t=("python", 7.79, 101, "hello")</pre>                                        | Creating the tuple with elements of different data types.                                                                                |
| <b>Indexing</b>                     | <pre>&gt;&gt;&gt;print(t[0]) python &gt;&gt;&gt;t[2] 101</pre>                                 | <ul style="list-style-type: none"> <li>❖ Accessing the item in the position 0</li> <li>❖ Accessing the item in the position 2</li> </ul> |
| <b>Slicing( ending position -1)</b> | <pre>&gt;&gt;&gt;print(t[1:3]) (7.79, 101)</pre>                                               | ❖ Displaying items from 1st till 2nd.                                                                                                    |
| <b>Concatenation</b>                | <pre>&gt;&gt;&gt;t+("ram", 67) ('python', 7.79, 101, 'hello', 'ram', 67)</pre>                 | ❖ Adding tuple elements at the end of another tuple elements                                                                             |
| <b>Repetition</b>                   | <pre>&gt;&gt;&gt;print(t*2) ('python', 7.79, 101, 'hello', 'python', 7.79, 101, 'hello')</pre> | ❖ 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[o]="a"
ace back (most recent call last):
ile "<stdin>", line 1, in <module>
pe Error: 'tuple' object does not support item assignment
```

## 21 Mapping

- This data type is unordered and mutable.
- Dictionaries fall under Mappings.

## 22 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 theirposition.
- ❖ A dictionary is an associative array (also known as hashes). Any key of the dictionary is associated (or mapped) to avalue.
- ❖ 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        | <pre>&gt;&gt;&gt; food = {"ham": "yes", "egg" : "yes", "rate":450 } &gt;&gt;&gt; print(food) {'rate': 450, 'egg': 'yes', 'ham': 'yes'}</pre> | Creating the dictionary with elements of different data types. |
| Indexing                     | <pre>&gt;&gt;&gt;&gt; print(food["rate"]) 450</pre>                                                                                          | Accessing the item with keys.                                  |
| Slicing( ending position -1) | <pre>&gt;&gt;&gt; print(t[1:3]) (7.79, 101)</pre>                                                                                            | 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.

#### **23 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.

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

#### IDENTIFIERS:

**24 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.

## 25 Example:

Names like myClass, var\_1, and **this\_is\_a\_long\_variable**

| 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:

### 26 Statements:

- Instructions that a Python interpreter can execute are called statements.
- A statement is a unit of code like creating a variable or displaying a value.

```
>>> 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.

### 27 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")**



## 28 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:

```
int (expression/constant/variable)
```

## 29 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:

## 30 Example:

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

## 31 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.

## 32 Syntax:

functionname \_\_doc\_\_ Example:

```
def double(num):
 """Function to double the value"""
 return 2*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.

### 33 Example:

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

### QUOTATION IN PYTHON:

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

- Single quotes(' ') Eg, 'This a string in single quotes'
- double quotes(" ") Eg, "This a string in doublequotes"
- 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

### 34 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<br>print(a,b)<br>temp = a<br>a = b<br>b = temp<br>print(a,b) | (2, 3)<br>(3, 2)<br>>>> |

-Tuple assignment solves this problem neatly:

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

### 35 -One way to think of tuple assignment is as tuple packing/unpacking.

In tuple packing, the values on the left are 'packed' together in a tuple:

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

### 36 -In tuple unpacking, the values in a tuple on the right are 'unpacked' into the variables/names on the right:

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

-The right side can be any kind of sequence (string,list,tuple)

### 37 Example:

-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 **+ is called operator**
- 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

### 38 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                                                                              | $a ** b = 10 \text{ to the power } 20$ |
| //               | Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed | $5 // 2 = 2$                           |

### 39 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)
```

### 40 Output:

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

### 41 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.     |

## 42 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

## 43 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<br>assigns value of a + b into c |
| += Add AND      | It adds right operand to the left operand and assign the result to left operand        | 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          |

|                            |                 |                                                                                            |                                                                                |
|----------------------------|-----------------|--------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------|
| <b>*=</b><br>AND           | <b>Multiply</b> | It multiplies right operand with the left operand and assign the result to left operand    | $c *= a$ is equivalent to $c = c * a$                                          |
| <b>/=</b><br>AND           | <b>Divide</b>   | It divides left operand with the right operand and assign the result to left operand       | $c /= a$ is equivalent to $c = c / a$<br>$c /= a$ is equivalent to $c = c / a$ |
| <b>%=</b><br>AND           | <b>Modulus</b>  | It takes modulus using two operands and assign the result to left operand                  | $c \% a$ is equivalent to $c = c \% a$                                         |
| <b>**=</b> Exponent<br>AND |                 | Performs exponential (power) calculation on operators and assign value to the left operand | $c ** a$ is equivalent to $c = c ** a$                                         |
| <b>//=</b><br>Division     | <b>Floor</b>    | It performs floor division on operators and assign value to the left operand               | $c //= a$ is equivalent to $c = c // a$                                        |

#### 44 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)

```

#### 45 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 is 2
Line 6 - Value of c is 2097152
Line 7 - Value of c is 99864

```

#### 46 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   |

#### 47 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)
```

#### 48 Output

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

#### 49 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

```

## 50 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 are used.

| 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 |

## 51 Example:

```

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

```

## 52 Identity Operators:

- They are used to check if two values (or variables) are located on the same part of 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 |

## 53 Example

```

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

```

**Out**  
**put** False  
 True  
 e



## 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  $(1+1)**(5-2)$  is 8.
- You can also use parentheses to make an expression easier to read, as in  $(\text{minute} * 100) / 60$ , even if it doesn't change the result.
- Exponentiation has the next highest precedence, so  $1 + 2**3$  is 9, not 27, and  $2*3-1$  is 5, not 4, and  $6+4/2$  is 8, not 5.
- Operators with the same precedence are evaluated from left to right (except exponentiation).

#### 54 Example:

|                                                                                                          |                                                                                                                       |                                                                                       |
|----------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|
| $a=9-12/3+3*2-1$<br>$a=?$<br>$a=9-4+3*2-1$<br>$a=9-4+6-1$<br>$a=5+6-1$ $a=11-1$ <b><math>a=10</math></b> | $A=2*3+4\%5-3/2+6$<br>$A=6+4\%5-3/2+6$<br>$A=6+4-3/2+6$ $A=6+4-1+6$<br>$A=10-1+6$<br>$A=9+6$ <b><math>A=15</math></b> | find $m=?$<br>$m=-43\ 8\&\&0\ -2$ $m=-43\ 0\ -2$ $m=1\ -2$<br><b><math>m=1</math></b> |
|----------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|

### 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.

#### 55 Types of function:

Functions can be classified into two categories:

- user defined function
- Built in function

##### **i) Built in functions**

- Built in functions are the functions that are already created and stored in python.
- 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)<br>\x05'                             | #returns a character (a string) from an integer  |
| >>>float(5)<br>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)<br><type 'float'>                | #returns data type of object to which it belongs |
| >>>t=tuple([4,6.0,7])<br>(4, 6.0, 7)           | # to create tuple of items from list             |
| >>>print("good morning")<br>Good morning       | # displays the given object                      |
| >>>input("enter name:")<br>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 and use.
- 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 Python prompt.
- 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 returntype**

- In this type no argument is passed through the function call and no output is return to mainfunction
- The sub function will read the input values perform the operation and print the result in the sameblock

**ii) Function with arguments and without returntype**

- Arguments are passed through the function call but output is not return to the mainfunction

**iii) Function without arguments and with returntype**

- In this type no argument is passed through the function call but output is return to the mainfunction.

**iv) Function with arguments and with returntype**

- In this type arguments are passed through the function call and output is return to the mainfunction

**Without Return Type**

| Withoutargument                                                                                            | With argument                                                                                            |
|------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------|
| <pre>def add():     a=int(input("enter a"))     b=int(input("enter b"))     c=a+b     print(c) add()</pre> | <pre>def add(a,b):     c=a+b     print(c) a=int(input("enter a")) b=int(input("enter b")) add(a,b)</pre> |
| <b>OUTPUT:</b><br>enter a5<br>enter b 10<br>15                                                             | <b>OUTPUT:</b><br>enter a5<br>enter b 10<br>15                                                           |

**With return type**

| Withoutargument                                                                                                      | With argument                                                                                                     |
|----------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------|
| <pre>def add():     a=int(input("enter a"))     b=int(input("enterb"))     c=a+b     return c c=add() print(c)</pre> | <pre>defadd(a,b):     c=a+b     returnc a=int(input("enter a")) b=int(input("enter b")) c=add(a,b) print(c)</pre> |
| <b>OUTPUT:</b><br>enter a5<br>enter b 10<br>15                                                                       | <b>OUTPUT:</b><br>enter a5<br>enter b 10<br>15                                                                    |

## Parameters And Arguments:

### 56 Parameters:

- Parameters are the value(s) provided in the parenthesis 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: `def my_add(a,b):`

### 57 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.
- Bonding 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:

## 58 ARGUMENT TYPES:

4. Required Arguments
5. Keyword Arguments
6. Default Arguments
7. Variable length Arguments

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

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

### 59 Output:

```
ame:
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.

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

### 60 Output:

```
ame:
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")
```

### 61 Output:

```
ame:
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.

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

### 62 Output:

```
janrahulmichealärjun
```

## 7.MODULES:

- **A module is a file containing Python definitions ,functions, statements and instructions.**
- Standard library of Python is extended asmodules.
- **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.
  - oThere is large number of standard modules also available in python. oStandard modules can be imported the same way as we import our user- defined modules.
  - oEvery module contains many function.
    - oTo 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.**

### 63 Syntax:

| import<br>module_name.module_name.function_name(variable) |                                      |
|-----------------------------------------------------------|--------------------------------------|
| Importing Builtin Module:                                 | Importing User Defined Module:       |
| import math<br>x=math.sqrt(25)<br>print(x)                | import calx=cal.add(5,4)<br>print(x) |


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

|                                                                                                                                                                                                                                        |                                                                                                                                                                                                                                               |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b><u>Import:</u></b> It is simplest and most common way to use modules in our code.<br><b>Example:</b><br><pre>import math x=math.pi print("The value of pi is", x)</pre> Output: The value of pi is 3.141592653589793                | <b><u>from import :</u></b> It is used to get a specific function in the code instead of complete file.<br><b>Example:</b><br><pre>from math import pi x=pi print("The value of pi is", x)</pre> Output: The value of pi is 3.141592653589793 |
| <b><u>import with renaming:</u></b><br>We can import a module by renaming the module as our wish.<br><b>Example:</b><br><pre>import math as m x=m.pi print("The value of pi is", x)</pre> Output: The value of pi is 3.141592653589793 | <b><u>import all:</u></b><br>We can import all names(definitions) form a module using *<br><b>Example:</b><br><pre>from math import * x=pi print("The value of pi is", x)</pre> 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  $\pi = 3.141592$

`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

| <u>Program for SWAPPING(Exchanging )of values</u>                                                                                                                                                 | <u>Output</u>                                                                                  |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------|
| <pre>a = int(input("Enter a value ")) b = int(input("Enter b value")) c = a a = b b =c print("a=",a,"b=",b,)</pre>                                                                                | <pre>Enter a value 5 Enter b value 8 a=8 b=5</pre>                                             |
| <u>Program to find distance between twopoints</u>                                                                                                                                                 | <u>Output</u>                                                                                  |
| <pre>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)</pre> | <pre>enter x17 enter y16 enter x25 enter y27 2.5</pre>                                         |
| <u>Program to circulate n numbers</u>                                                                                                                                                             | <u>Output:</u>                                                                                 |
| <pre>a=list(input("enter the list"))</pre>                                                                                                                                                        | <pre>enter the list '1234'</pre>                                                               |
| <pre>print(a) for i in range(1,len(a),1):     print(a[i:]+a[:i])</pre>                                                                                                                            | <pre>['1', '2', '3', '4'] ['2', '3', '4', '1'] ['3', '4', '1', '2'] ['4', '1', '2', '3']</pre> |

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.

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

6. What is a 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.

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

| Swap two numbers                                               | Output:                               |
|----------------------------------------------------------------|---------------------------------------|
| <pre>a=2;b=3 print(a,b) a = a+b b= a-b a= a-b print(a,b)</pre> | <pre>(2, 3) (3, 2) &gt;&gt;&gt;</pre> |

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

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   |

#### 64 Example

#### Output

```
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)
```

```
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.

#### Syntax:

```
return[expression]
```

But exits from function.

11. What are the types of arguments?

- RequiredArguments
- KeywordArguments
- Default Arguments
- Variable lengthArguments

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) Explain about conditional statements with suitable examples.

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

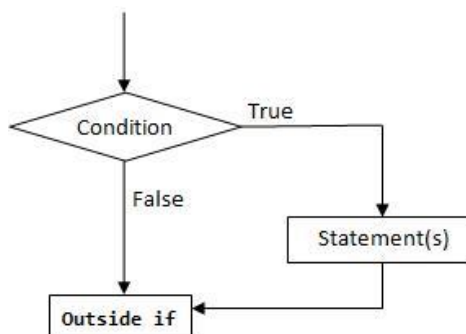
#### Conditional (if):

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                                                                      |
|----------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------|
| <pre> m=eval(input("enter ur mark out of 100")) c=input("enter ur category G/S") if(c=="S"):     m=m+5 print("mark is",m) </pre> | <pre> enter ur mark out of 100 85 enter ur category G/S S mark is 90 </pre> |

#### Alternative (if-else):

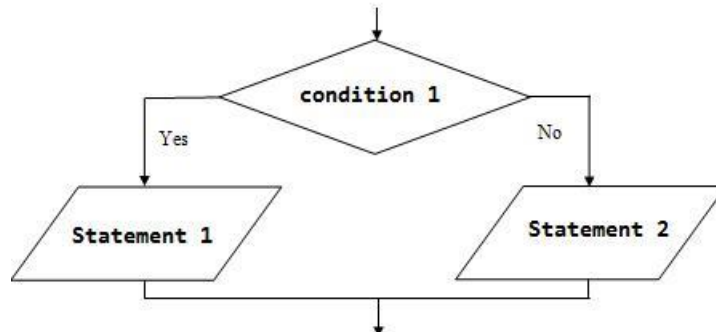
In the alternative the condition must be true or false. In this **else** statement can be combined with **if** statement. The **else** statement contains the block of code that executes when the condition is false. If the

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                                     |
|------------------------------------------------------------------------------------------------------------------------|--------------------------------------------|
| <pre>n=eval(input("enter a number")) if(n%2==0):     print("even number") lse:     print("odd number")</pre>           | <pre>enter a number4 even number</pre>     |
| positive or negative number                                                                                            | Output                                     |
| <pre>n=eval(input("enter a number")) if(n&gt;=0):     print("positive number") lse:     print("negative number")</pre> | <pre>enter a number8 positive number</pre> |
| leap year or not                                                                                                       | Output                                     |
| <pre>y=eval(input("enter a year")) if(y%4==0):     print("leap year") lse:     print("not leap year")</pre>            | <pre>enter a year2000 leap year</pre>      |

## 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.

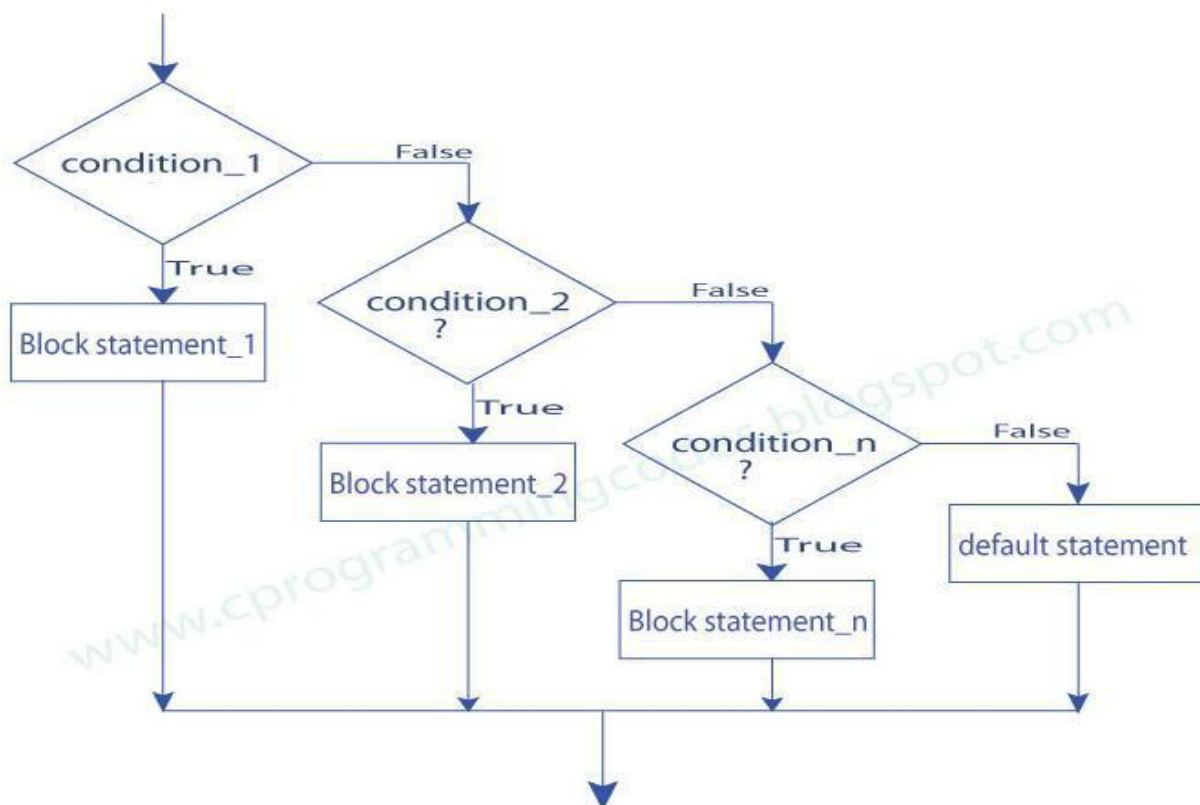
The if block can have only one else block. But it can have multiple elif blocks.

- 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                                      |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------|
| <pre> mark=eval(input("enter ur mark:")) if(mark&gt;=90):     print("grade:S") elif(mark&gt;=80):     print("grade:A") elif(mark&gt;=70):     print("grade:B") elif(mark&gt;=50):     print("grade:C") else:     print("fail") </pre> | <pre> enter ur mark:78 grade:B </pre>       |
| traffic light system                                                                                                                                                                                                                  | Output                                      |
| <pre> colour=input("enter colour of light:") if(colour=="green"):     print("GO") elif(colour=="yellow"):     print("GET READY") else:     print("STOP") </pre>                                                                       | <pre> enter colour of light:green GO </pre> |

### 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

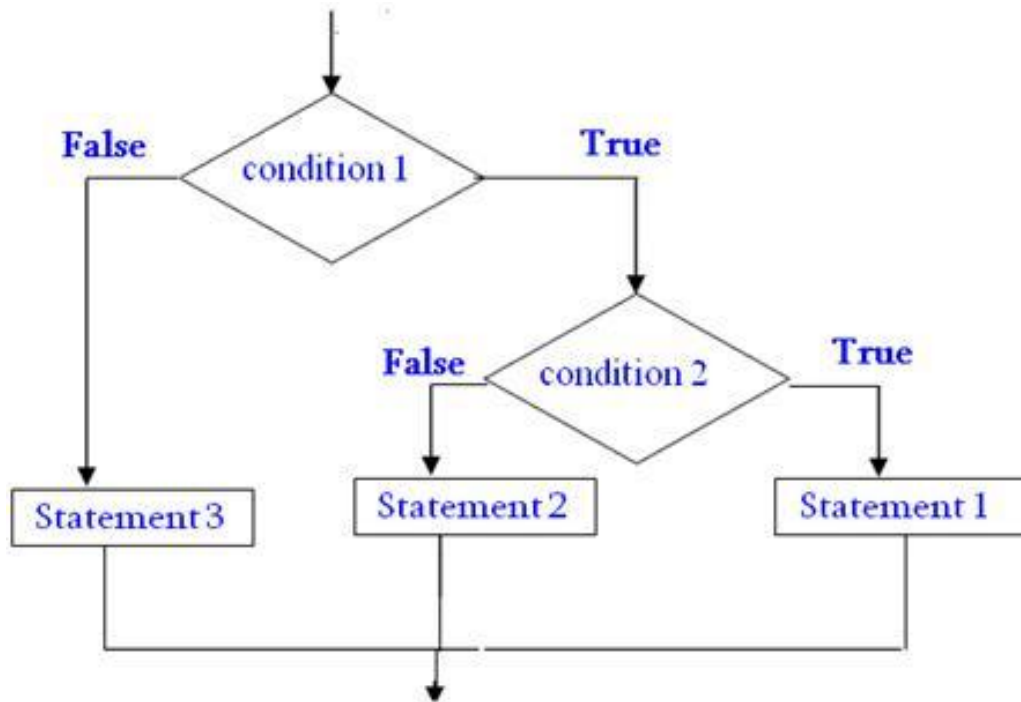
```

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

```



**Flowchart:**



**Example:**

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

| greatest of three numbers                                                                                                                                                                                                                      | output                                                                                               |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------|
| <pre>a=eval(input("enter the value of a")) b=eval(input("enter the value of b")) c=eval(input("enter the value of c")) if(a&gt;b):     if(a&gt;c):         print("the greatest no is",a)     else:         print("the greatest no is",c)</pre> | <pre>enter the value of a 9 enter the value of a 1 enter the value of a 8 the greatest no is 9</pre> |

```
else:
 if(b>c):
 print("the greatest no is",b)
 else:
 print("the greatest no is",c)
```

| positive negative or zero                                                                                                                                                                                   | output                                                    |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------|
| <pre>n=eval(input("enter the value of n:")) if(n==0):     print("the number is zero") else:     if(n&gt;0):         print("the number is positive")     else:         print("the number is negative")</pre> | <pre>enter the value of n:-9 the number is negative</pre> |

## 2) Explain about 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.

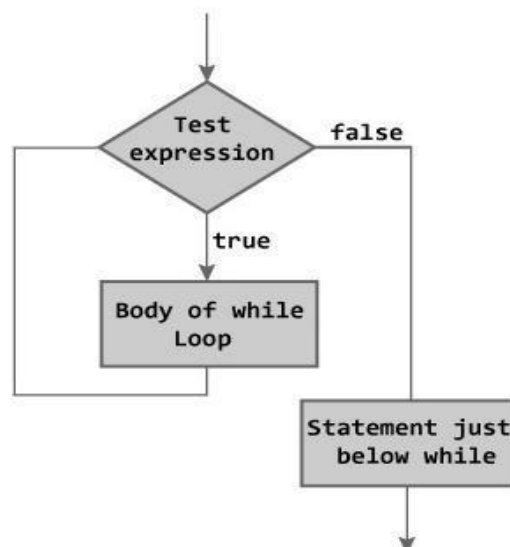
The statements inside the while start with indentation and the first unintended line marks the end.

### 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                          |
|-----------------------------------------------------------------------------------------------------------------|---------------------------------|
| <pre>n=eval(input("enter n")) i=1 sum=0 while(i&lt;=n):     sum=sum+i     i=i+1 print(sum)</pre>                | <pre>enter n 10 55</pre>        |
| Factorial of a numbers:                                                                                         | output                          |
| <pre>n=eval(input("enter n")) i=1 fact=1 while(i&lt;=n):     fact=fact*i     i=i+1 print(fact)</pre>            | <pre>enter n 5 120</pre>        |
| Sum of digits of a number:                                                                                      | output                          |
| <pre>n=eval(input("enter a number")) sum=0 while(n&gt;0):     a=n%10     sum=sum+a     n=n//10 print(sum)</pre> | <pre>enter a number 123 6</pre> |

| Reverse the given number:       | output         |
|---------------------------------|----------------|
| n=eval(input("enter a number")) | enter a number |
| sum=0                           | 123            |
| while(n>0):                     | 321            |
| a=n%10                          |                |
| sum=sum*10+a                    |                |
| n=n//10                         |                |
| print(sum)                      |                |

| Armstrong number or not                           | output                               |
|---------------------------------------------------|--------------------------------------|
| n=eval(input("enter a number"))                   | enter a number153                    |
| org=n                                             | The given number is Armstrong number |
| 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") |                                      |

| Palindrome or not                       | output                     |
|-----------------------------------------|----------------------------|
| n=eval(input("enter a number"))         | enter a number121          |
| org=n                                   | The given no is palindrome |
| 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") |                            |

## For loop:

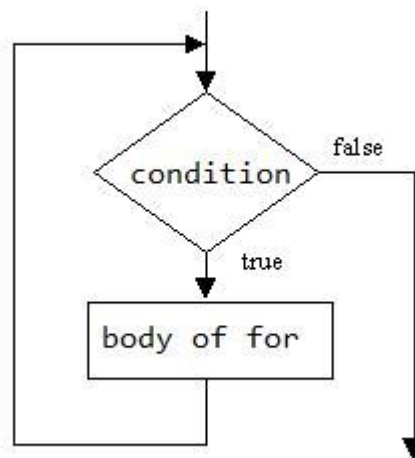
### □ for in range:

- 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

| .no | sequences          | example                      | output           |
|-----|--------------------|------------------------------|------------------|
| .   | For loop in string | for i in "Ramu":<br>print(i) | R<br>A<br>M<br>U |

|   |                   |                                   |                       |
|---|-------------------|-----------------------------------|-----------------------|
| 2 | For loop in list  | for i in [2,3,5,6,9]:<br>print(i) | 2<br>3<br>5<br>6<br>9 |
| 3 | For loop in tuple | for i in (2,3,1):<br>print(i)     | 2<br>3<br>1           |

### Exam

#### Examples:

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

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

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

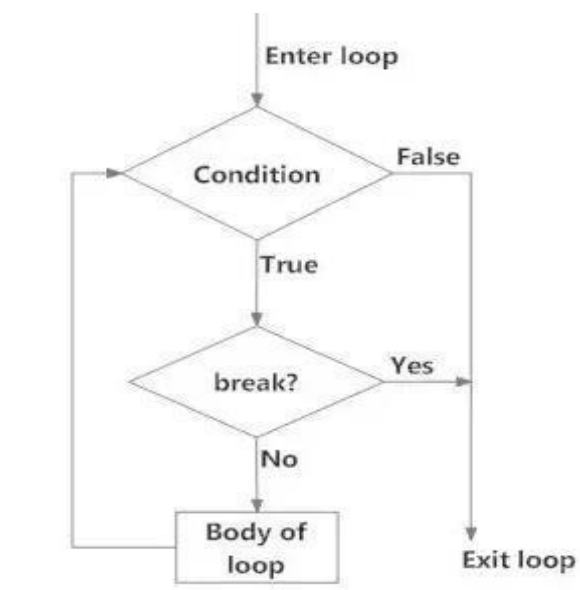
break

**while (test Expression):**

```
// codes
if (condition for break):
```



#### Flowchart



| example                                                                               | Output                   |
|---------------------------------------------------------------------------------------|--------------------------|
| <pre>for i in "welcome":<br/>    if(i=="c"):<br/>        break<br/>    print(i)</pre> | <pre>w<br/>e<br/>l</pre> |

## CONTINUE

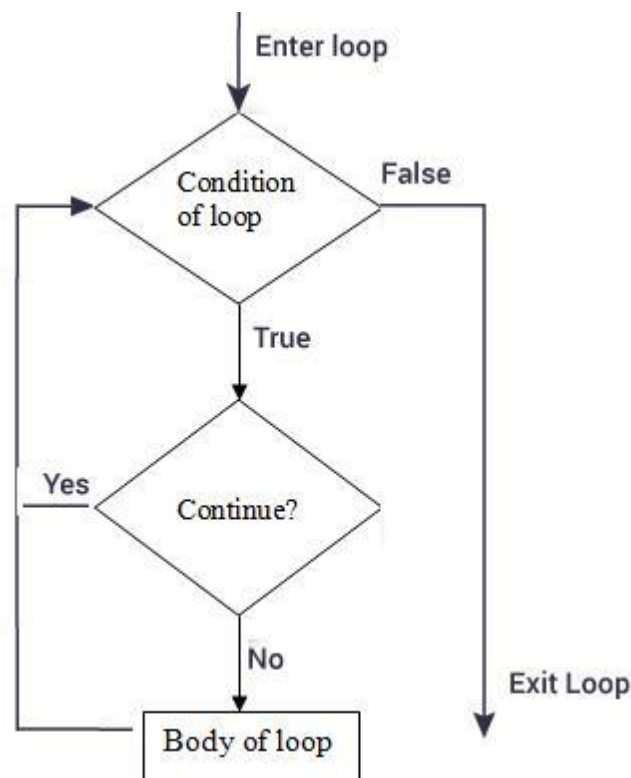
It terminates the current iteration and transfer the control to the next iteration in the loop.

**Syntax:** Continue

```
→ while (test Expression):
 // codes
 if (condition for continue):
 continue

 // codes
```

### Flowchart



| Example:                                                                                 | Output                     |
|------------------------------------------------------------------------------------------|----------------------------|
| <pre>for i in "welcome":<br/>    if(i=="c"):<br/>        continue<br/>    print(i)</pre> | w<br>e<br>l<br>o<br>m<br>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.



**Syntax:**

pass

break

| Example                                                                 | Output                   |
|-------------------------------------------------------------------------|--------------------------|
| <pre>for i in "welcome":     if (i == "c"):         pass print(i)</pre> | <pre>w e l c o m e</pre> |

**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:<br>break                                                                      | syntax:<br>continue                                                                             |
| <pre>for i in "welcome":     if(i=="c"     ):         break     print(i)</pre>        | <pre>for i in "welcome":     if(i=="c"):         continue     print(i)</pre>                    |
| <pre>w e l</pre>                                                                      | <pre>w e l o m e</pre>                                                                          |

**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                           |
|-------------------------------------------------------------------------------------------|----------------------------------|
| <pre>for i in range(1,6):     print(i) else:     print("the number greater than 6")</pre> | <p>the number greater than 6</p> |

#### **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                                                         |
|----------------------------------------------------------------------------------------------------|----------------------------------------------------------------|
| <pre>i=1 while(i&lt;=5):     print(i)     i=i+1 else:     print("the number greater than 5")</pre> | <p>1<br/>2<br/>3<br/>4<br/>5<br/>the number greater than 5</p> |

#### **4) Explain about 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 2<sup>nd</sup> 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                                    |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------|
| <pre>a=50<br/><br/>def add():<br/>    b=20<br/>    c=a+b<br/>    print©<br/><br/>def sub():<br/>    b=30<br/>    c=a-b<br/>    print©<br/>print(a)</pre> <div><div>Global<br/>Variable</div><div>Local<br/>Variable</div></div> | <div>70</div> <div>20</div> <div>50</div> |

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

| Example                                                                                                                                                                                                                                                                                                                                                           | output                                        |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------|
| <pre>def add():     b=20      c=a+b     print©</pre> <div style="border: 1px solid black; padding: 5px; display: inline-block; margin: 10px;">Local Variable</div> <pre>def sub():     b=30     c=a-b     print©</pre> <div style="border: 1px solid black; padding: 5px; display: inline-block; margin: 10px;">Local Variable</div> <pre>print(a) print(b)</pre> | <p>70</p> <p>20</p> <p>error</p> <p>error</p> |

### **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.

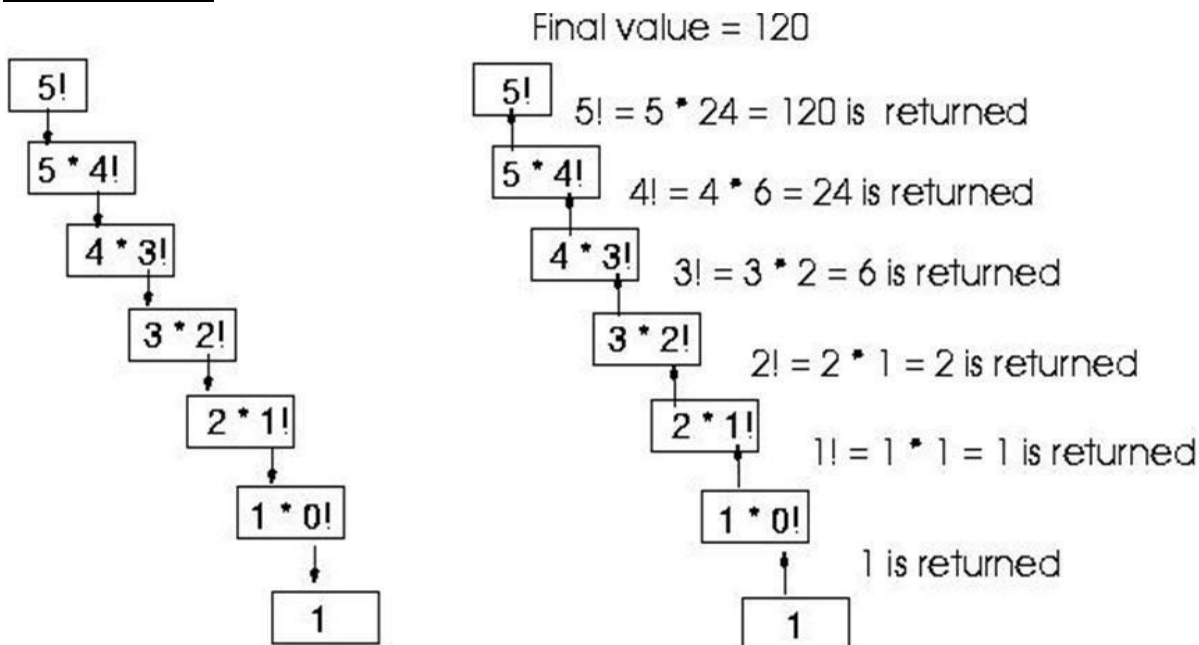
| find sum and average using function composition                                                                                                                                                      | output                                                  |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------|
| <pre>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)</pre> | <p>enter a:4</p> <p>enter b:8</p> <p>the avg is 6.0</p> |

### **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                                             |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------|
| <pre>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)</pre> | <p>enter no. to find fact:5</p> <p>Fact is 120</p> |

### Explanation



### Examples:

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

| Sum of n numbers                                                                                                                                                   | Output                                            |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------|
| <pre>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)</pre> | <p>enter no. to find sum:10</p> <p>Fact is 55</p> |

### 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 |

|                                 |                                                                                                                |                                                                                                                                                                                               |
|---------------------------------|----------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>indexing</b>                 | <pre>&gt;&gt;&gt;a="HELLO" &gt;&gt;&gt;print(a[0])  &gt;&gt;&gt;H &gt;&gt;&gt;print(a[-1]) &gt;&gt;&gt;O</pre> | <ul style="list-style-type: none"><li>□ Positive indexing helps in accessing the string from the beginning</li><li>□ Negative subscript helps in accessing the string from the end.</li></ul> |
|                                 | Print[0:4] – HEL                                                                                               | The Slice[start : stop] operator extracts                                                                                                                                                     |
| <b>Slicing:</b>                 | <pre>Print[ :3] – HEL Print[0: ]- HELLO</pre>                                                                  | sub string from the strings.<br>A segment of a string is called a slice.                                                                                                                      |
| <b>Concatenatio</b><br><b>n</b> | <pre>a="save" b="earth" &gt;&gt;&gt;print(a+b) Save earth</pre>                                                | The + operator joins the text on both sides of the operator.                                                                                                                                  |
| <b>Repetitions:</b>             | <pre>a="panimalar " &gt;&gt;&gt;print(3*a)</pre>                                                               | The * operator repeats the string on the left hand side times the value on right                                                                                                              |

|                    |                                                                                                    |                                                                                                                    |
|--------------------|----------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------|
|                    | <p>animalaranimalar<br/>animalar</p>                                                               | hand side.                                                                                                         |
| <b>Membership:</b> | <pre>&gt;&gt;&gt; s="good morning" &gt;&gt;&gt; "m" in s True &gt;&gt;&gt; "a" not in s True</pre> | <p>Using membership operators to check a particular character is in string or not.<br/>Returns true if present</p> |

|  |                       |
|--|-----------------------|
|  | <b>String slices:</b> |
|--|-----------------------|

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

|                                         |                                                                |                                                                                                                        |
|-----------------------------------------|----------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------|
| <p><b>Slicing:</b></p> <p>a="HELLO"</p> | <pre>Print[0:4] – HEL Print[ :3] – HEL Print[0: ]- HELLO</pre> | <p>The Slice[n : m] operator extracts sub string from the strings.</p> <p>A segment of a string is called a slice.</p> |
|-----------------------------------------|----------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------|

|                             |
|-----------------------------|
| <b><u>Immutability:</u></b> |
|-----------------------------|

- 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 | <pre>a="PYTHON" a[0]='x'</pre>       | <pre>TypeError: 'str' object does not support element assignment</pre> |
| element deletion   | <pre>a="PYTHON" del a[0]</pre>       | <pre>TypeError: object 'str' does not support element deletion</pre>   |
| delete a string    | <pre>a="PYTHON" del a print(a)</pre> | <pre>NameError: name 'my_string' is not defined</pre>                  |

### 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                                                                                |
|---|-----------------------------|--------------------------------------------------------------------|--------------------------------------------------------------------------------------------|
|   | a.capitalize()              | >>> a.capitalize()<br><br>' Happy birthday'                        | capitalize only the first letter in a string                                               |
|   | a.upper()                   | >>> a.upper()<br>'HAPPY BIRTHDAY'                                  | change string to upper case                                                                |
|   | a.lower()                   | >>> a.lower()<br>' happy birthday'                                 | change string to lower case                                                                |
|   | a.title()                   | >>> a.title()<br>' Happy Birthday '                                | change string to title case i.e. first characters of all the words are capitalized.        |
|   | a.swapcase()                | >>> a.swapcase()<br>'HAPPY BIRTHDAY'                               | change lowercase characters to uppercase and vice versa                                    |
|   | a.split()                   | >>> a.split()<br>['happy', 'birthday']                             | returns a list of words separated by space                                                 |
|   | a.center(width, "fillchar") | >>>a.center(19, "**")<br><br>'***happy birthday***'                | pads the string with the specified "fillchar" till the length is equal to "width"          |
|   | a.count(substring)          | >>> a.count('happy')<br>1                                          | returns the number of occurrences of substring                                             |
|   | a.replace(old, new)         | >>>a.replace('happy', 'wishyou happy')<br>'wishyou happy birthday' | replace all old substrings with new substrings                                             |
| 0 | a.join(b)                   | >>> b="happy"<br>>>> a="-"<br>>>> a.join(b)<br><br>'h-a-p-p-y'     | returns a string concatenated with the elements of an iterable. (Here “a” is the iterable) |
| 1 | a.isupper()                 | >>> a.isupper()<br>False                                           | checks whether all the case-based characters (letters) of                                  |



|   |             |                              |                                                                                     |
|---|-------------|------------------------------|-------------------------------------------------------------------------------------|
|   |             |                              | the string are uppercase.                                                           |
| 2 | a.islower() | >>> a.islower()<br>True      | checks whether all the case-based characters (letters) of the string are lowercase. |
| 3 | a.isalpha() | >>> a.isalpha()<br><br>False | checks whether the string consists of alphabetic characters only.                   |

**String modules:**

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

**Escape sequences in string**

| Escape Sequence | Description             | example                                                |
|-----------------|-------------------------|--------------------------------------------------------|
| \n              | new line                | <pre>&gt;&gt;&gt; print("hai \nhello") hai hello</pre> |
| \\              | prints Backslash (\)    | <pre>&gt;&gt;&gt; print("hai\\hello") hai\hello</pre>  |
| \'              | prints Single quote (') | <pre>&gt;&gt;&gt; print('') '</pre>                    |
| \"              | prints Double quote (") | <pre>&gt;&gt;&gt;print("") "</pre>                     |
| \t              | prints tab sapace       | <pre>&gt;&gt;&gt;print("hai\thello") hai hello</pre>   |
| \a              | ASCII Bell (BEL)        | <pre>&gt;&gt;&gt;print("\a")</pre>                     |

## 6) Explain about 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 |
|-------------------------------------------------------------------------------------------------|--------|
| <pre>import array sum=0 a=array.array('i',[1,2,3,4]) for i in a:     sum=sum+i print(sum)</pre> | 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 |
|----------------------------------------------------------------------------------------------------------------------|--------|
| <pre>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)</pre> | 35     |

## Methods in array

a=[2,3,4,5]

|  | Syntax                       | example                           | Description                                                                                        |
|--|------------------------------|-----------------------------------|----------------------------------------------------------------------------------------------------|
|  | 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. |
|  | append()                     | >>>a.append(6)<br>[2,3,4,5,6]     | This method is used to add the value at the end of the array.                                      |
|  | insert(index,element)        | >>>a.insert(2,10)<br>[2,3,10,5,6] | This method is used to add the value at the position specified in its argument.                    |
|  | pop(index)                   | >>>a.pop(1)<br>[2,10,5,6]         | This function removes the element at the position mentioned in its argument, and returns it.       |
|  | index(element)               | >>>a.index(2)<br>0                | This function returns the index of value                                                           |
|  | reverse()                    | >>>a.reverse()<br>[6,5,10,2]      | This function reverses the array.                                                                  |
|  | count()                      | a.count()                         | This is used to count number of                                                                    |

## 7.ILLUSTRATIVE PROGRAMS:

| Square root using newtons method:                                                                                                                                                                                                               | Output:                                                                                     |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------|
| <pre>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)</pre>                                                                      | <p>enter number to find Sqrt: 9<br/>3.0</p>                                                 |
| GCD of two numbers                                                                                                                                                                                                                              | output                                                                                      |
| <pre>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)</pre>                                                                                    | <p>Enter a number1:8<br/>Enter a number2:24<br/>8</p>                                       |
| Exponent of number                                                                                                                                                                                                                              | Output:                                                                                     |
| <pre>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)</pre> | <p>Enter base: 2<br/>Enter exponential value:3<br/>Result: 8</p>                            |
| sum of array elements:                                                                                                                                                                                                                          | output:                                                                                     |
| <pre>a=[2,3,4,5,6,7,8] sum=0 for i in a:     sum=sum+i print("the sum is",sum)</pre>                                                                                                                                                            | <p>the sum is 35</p>                                                                        |
| Linear search                                                                                                                                                                                                                                   | output                                                                                      |
| <pre>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")</pre>                | <p>[20, 30, 40, 50, 60, 70, 89]<br/>enter a element to search:30<br/>element found at 2</p> |

| Binary search                                                                                                                                                                                                                                                                                                                                                       | output                                                                                    |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------|
| <pre> 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&lt;=stop):     mid=(start+stop)//2     if(search==a[mid]):         print("elemrnt found at",mid+1)         break     elif(search&lt;a[mid]):         stop=mid-1     else:         start=mid+1 else:     print("not found") </pre> | <pre> [20, 30, 40, 50, 60, 70, 89] enter a element to search:30 element found at 2 </pre> |

## 2 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:<br>break                                                                      | syntax:<br>continue                                                                             |
| for i in "welcome":<br>if(i=="c"):<br>):<br>break<br>print(i)                         | for i in "welcome":<br><br>if(i=="c"):<br>continue<br>print(i)                                  |
| w<br>e<br>l                                                                           | w<br>e<br>l<br>o<br>m<br>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.



**Syntax:**

pass  
break

| Example                                                   | Output                          |
|-----------------------------------------------------------|---------------------------------|
| for i in "welcome":<br>if (i == "c"):<br>pass<br>print(i) | w<br>e<br>l<br>c<br>o<br>m<br>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.capitalize() – 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

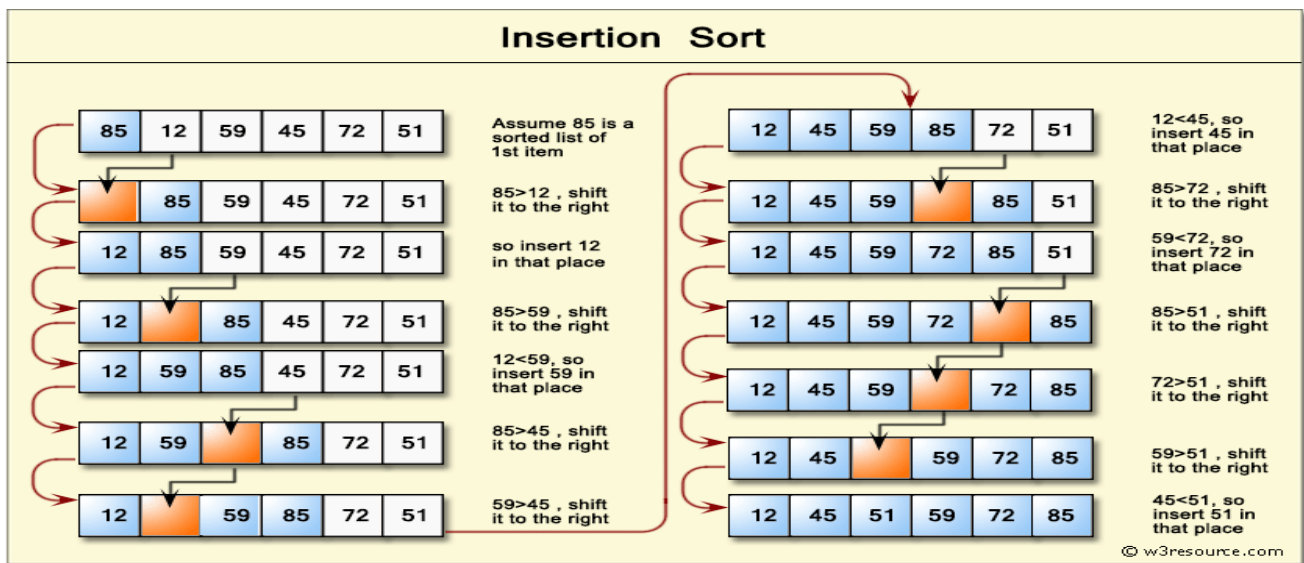


## UNIT IV LISTS, TUPLES, DICTIONARIES

### 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:



66

67 **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=j-1
 a[j+1]=key
print("After sorting(using insertion sort)",a)
```

## Output

```
Enter size
of list7 Enter
listelements4
Enter
listelements33 Enter
list elements6 Enter
listelements22 Enter
list elements6 Enter
list elements-9
Enter list
elements-2
```

Before sorting [4, 33, 6, 22, 6, -9, -2]

After sorting(using insertion sort) [-9, -2, 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

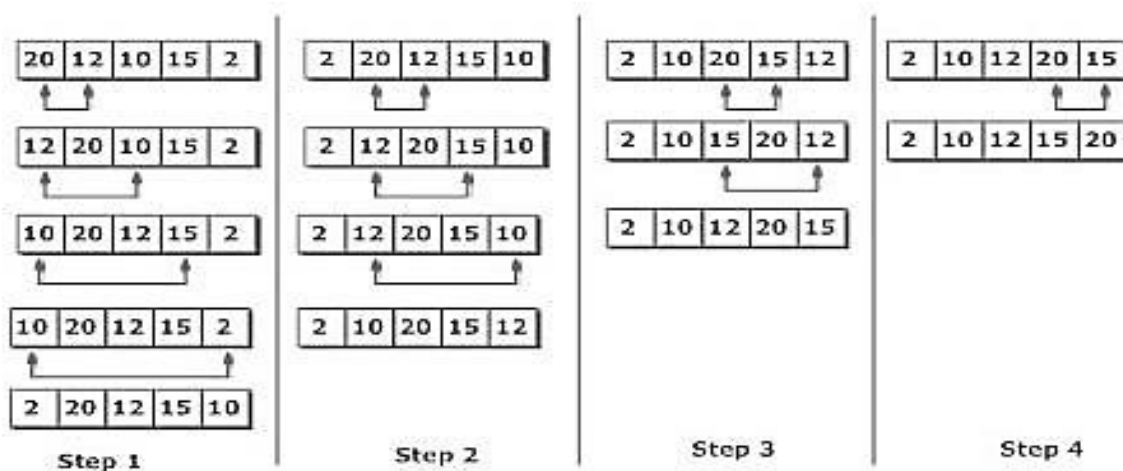


Figure: Selection Sort

### 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]=tem
 p
 print("Sorted list(using Selection Sort)=",a)

```

**Output:**

```

Enter size
of list5 Enter list
elements12 Enter
list elements-5
Enter list
elements4 Enter
list elements48
Enter list
elements98

```

List before sorting [12, -5, 4, 48, 98]

Sorted list(using Selection Sort)= [-5, 4, 12, 48, 98]

### 3. Quadratic Equation:

Formula :

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

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
```

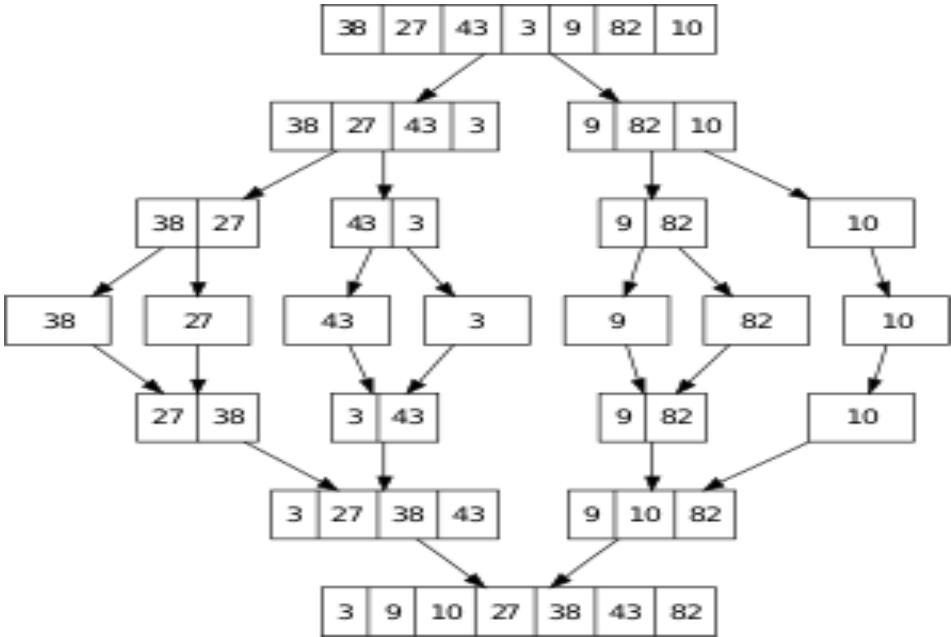
This equation has two solutions: (3+0j) or (2+0j)

### 4. Merge sort

Merge sort works as follows:

- a. Divide the unsorted list into n sublists, each containing 1 element (a list of 1 element is considered sorted).
- b. Repeatedly merge sublists to produce new sorted sublists until there is only 1 sublist remaining. This will be the sorted list.

Example



## Program:

```
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")
 print(a);
print("Sorted list
is")
print(mergesort(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 is [-8, 1, 14, 18, 21]
```



## 5. LIST

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

### 68 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. Eg: 

```
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]
```

## 69 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

70

71

72

## 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

### 73 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
>>> ()
```

### 74 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',100,8.0
2)
print (t[0]) = 'a'
print (t[1:3]) = ('b', 100 , 8.02)
```

### 75 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')
```

## 76 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
```

## 77 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:
4234 seconds=1 hrs:: 10 min:: 34 sec
```

## 7. Histogram

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

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 {}.

**78 Eg:**

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

|                    | <b>List</b>                                                                                      | <b>Tuples</b>                                                                                    | <b>Dictionary</b>                                                                              |
|--------------------|--------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------|
| <b>Mutable</b>     | List is mutable                                                                                  | Tuples are immutable                                                                             | Keys must be immutable. Values may mutable                                                     |
| <b>Indexing</b>    | 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 |
| <b>Declaration</b> | 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() |
|----------|----------|



|                                                                                                                            |                                                                                                                          |
|----------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------|
| <pre>&gt;&gt;&gt;a=[10,20,30] &gt;&gt;&gt;b=[40,50] &gt;&gt;&gt;a.append(b) &gt;&gt;&gt;print(a) [10,20,30,[40, 50]]</pre> | <pre>&gt;&gt;&gt;a=[10,20,30] &gt;&gt;&gt;b=[40,50] &gt;&gt;&gt;a.extend(b) &gt;&gt;&gt;print(a) [10,20,30,40, 50]</pre> |
|----------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------|

**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 0x7f7doa9e7c48>
```

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)
```

## 79 Unit - 5

### 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.

### 80 File Type

1. Text file
2. Binary file

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

### 81 Mode in File

| <b>Modul<br/>e</b> | <b>Descripti<br/>on</b>     |
|--------------------|-----------------------------|
| r                  | Read only                   |
| w                  | mode Write only             |
| a                  | Appending                   |
| r+                 | only<br>Read and write only |

#### **Differentiate write and append mode:**

| Write mode                                                                                                                                                                                                                                                       | Append mode                                                                                                                                                                                                        |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"><li>• It is used to write a string in a file</li><li>• If file is not exist it creates a new file</li><li>• If file is exit in the specified name, the existing content will overwrite in a file by the given string</li></ul> | <ul style="list-style-type: none"><li>• It is used to append (add) a string into a file</li><li>• If file is not exist it creates a new file</li><li>• It will add the string at the end of the old file</li></ul> |

#### **File Operation:**

- ✓ Open a file

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

## 1. Open ( ) function:

- Python's built-in open function is used to get a file object.
- The open function opens a file.
- It returns something called a file object.
- File objects have methods and attributes that can be used to collect

### 82 Syntax:

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

### 83 Example:

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

### 84 Create a text file

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

## 2. Read ( ) function

Read functions contain different methods

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

### 85 Syntax:

```
file_name.read()
```

### 86 Example:

```
fp=open("a.txt","w")
print(fp.read())
print(fp.read(6))
print(fp.readline())
print(fp.readline(3))
print(fp.readlines())
```

### 87 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.

#### **88 Syntax:**

```
file_name.write()
```

#### **89 Example:**

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

```
 fp.write("this file is
```

```
 a.txt") fp.write("to
```

```
 add more lines")
```

```
 fp.close()
```

#### **90 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.

### **4. Close ( ) function**

It is used to close the file.

#### **91 Syntax:**

```
filename.close()
```

#### **92 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.s
 plit() print(words)

```

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

```

source=open("a.txt","r")
destination=open("b.t
xt","w") for line in
source:
 destination.writ
e(line) source.close()
destination.close()

```

**93 Output:**

| <b>Input a.txt</b>                                                                                                                                      | <b>Output b.txt</b>                                                                                                                                     |
|---------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| 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)
```

**94 Output:**

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

#### 4. What is errors and its type? Explain the details about 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

##### **95 Types of errors**

- Syntax error or compile time error
- Run time error
- Logical error

##### **96 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

##### **97 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

##### **98 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

##### **99 Exception**

##### **handling**

##### **Exceptions**

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

##### **100 Errors in python**

- IOError-If the file cannot be opened.
- ImportError -If python cannot find the module
- ValueError -Raised when a built-in operation or function receives an argument that has the right type but an inappropriate value
- KeyboardInterrupt -Raised when the user hits the interrupt
- EOFError -Raised when one of the built-in functions (input() or raw\_input()) hits an end-of-file condition (EOF) without reading any data

## 101 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

#### 102 Syntax:

```
try :
 stateme
nts 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.

#### 103 Example:

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

|                                                                                        |                                                                                                 |
|----------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------|
| <b>Output:1</b><br>Enter the value of X = 10<br>Enter the value of Y = 5<br>Result = 2 | <b>Output : 2</b><br>Enter the value of X =<br>10 Enter the value of Y<br>= 10 Division by Zero |
|----------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------|

### 2. Try – Multiple except Statements

- Exception type must be different for except statements

#### 105 Syntax:

```
try:
 stateme
nts except
errors1:
 stateme
```

```
nts except
errors2:
 stateme
nts except
errors3:
 statements
```

**106 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))
107 except NameError:
 print(" The input must be number")
108 except ZeroDivisionError:
 print("Division by Zero")

```

|                                                                                                                              |                                                                                                                       |                                                                                                       |
|------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------|
| <b>Output:1</b><br>Enter the value of X = 10<br>Enter the value of Y = 5<br>Sum of 10 and 5 = 15<br>Division of 10 and 5 = 2 | <b>Output 2:</b><br>Enter the value of X = 10<br>Enter the value of Y = 0<br>Sum of 10 and 0 = 10<br>Division by Zero | <b>Output 3:</b><br>Enter the value of X = 10<br>Enter the value of Y = a<br>The input must be number |
|------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------|

**3. Try –Except-Else**

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

**109 Syntax:**

```

try:
 statements
except:
 stateme
else:
 nts

```

**110 Example**

```

 stateme
 nts

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

```

|                                                                                        |                                                                                                       |
|----------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------|
| <b>Output:1</b><br>Enter the value of X = 10<br>Enter the value of Y = 5<br>Result = 2 | <b>Output : 2</b><br>Enter the value of X =<br>10 Enter the value of Y<br>= 10<br>Division by<br>Zero |
|----------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------|

#### 4. Raise statement

- The raise statement allows the programmer to force a specified exception to occur.

##### 113 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:

##### 114 Example

```
try:
 ... raise NameError('HiThere')
... except NameError:
... print('An exception flew by!')
```

115 ... 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.

##### 116 Syntax

```
try:
 statements
except
 stateme
t:
 nts
finall
 stateme
y:
 nts
```

##### 117 Example

```
X=int(input("Enter the value
of X")) Y=int(input("Enter
the value of Y")) try:
result = X / (X - Y)
```



```

118 except ZeroDivisionError:
 print("Division by Zero")
119 else:
 print("result=".result)
120 finally:
 print ("executing finally clause")

```

|                                                                                                                    |                                                                                                                             |
|--------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <b>Output:1</b><br>Enter the value of X = 10<br>Enter the value of Y = 5<br>Result = 2<br>executing finally clause | <b>Output : 2</b><br>Enter the value of X =<br>10 Enter the value of Y<br>= 10 Division by Zero<br>executing finally clause |
|--------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------|

## 5. Explain the details about Modules in python

### Modules

- 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]]

#### 121 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
```

#### 122 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().

#### 123 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

**124 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

**125 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

## 126 import os

| method                  | example                       | description                                                                   |
|-------------------------|-------------------------------|-------------------------------------------------------------------------------|
| name                    | Osname 'nt'                   | This function gives the name of the operating system                          |
| getcwd()                | Os.getcwd()<br>,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

### 127 import sys

| methods           | example                                                  | description                                                                                                                               |
|-------------------|----------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| sys.argv          | sys.argv<br><br>sys.argv(0)<br>)<br><br>sys.argv(1)<br>) | Provides the list of command line arguments passed to a python script Provides to access the file name 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<br>'win32'                                  | Provide information about the operating system platform                                                                                   |
| sys.exit          | sys.exit<br><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<br>;calc.py<br>def<br>add(a,b); print(a+b)<br>def<br>sub(a,b);<br>print(a-b)<br>def<br>mul(a,b);<br>print(a*b)<br>def | import<br>calculator<br>calculator.add(2,<br>3)<br><br>Outcome<br>>>>5 |

|                                                |  |
|------------------------------------------------|--|
| <div><div>div(a,b);<br/>print(a/b)</div></div> |  |
|------------------------------------------------|--|

## 6. Explain the details about Package in python

### 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.
- ✓ \_\_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

### 128 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        |
| <b>2.calculater</b> | <b>08-12-2017</b> | <b>File folder</b> |
| 3. DLLs             | 10-12-2017        | File folder        |

### 129 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<br>add(a,b);<br>print(a+<br>b) | def div(a,b);<br>print(a/b) | def<br>mul(a,b)<br>;<br>print(a*<br>b) | def<br>sub(a,b)<br>;<br>print(a-<br>b) |



**130 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
```

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  |

### 131 **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
import sys
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))
```

### 132 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.**

| <b>Read</b>                                                                                                                                                                             | <b>Write</b>                                                                                                                                                               |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 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

**133 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

**134 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")
```

| <b>newfile.txt</b> | <b>newfile.txt(after updating)</b> |
|--------------------|------------------------------------|
| 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 anydepth, 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. 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.

#### 11. Write the snippet to find the current working directory.

```
import os
```

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

**Output:**

```
C:\\Users\\Mano\\Deskt
op
```

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

```
import os
```

```
os.path.abspath
```

```
('write.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