UNIT-I

Computational Thinking and Problem Solving

Fundamentals of computing - Computing Devices - Identification of computational Problems - Algorithms - Building blocks of algorithms (Statements, State, Control flow, functions), notation (pseudocode, 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.

Fundamentals of Computing

* A computer is a machine or device that performs processes, calculations and operations based on instructions provided by a software or hardware program.

Types of computer

We can categorize computer in two ways; on the basis of data handling capabilities and size.

* Analogue computer
* Digital computer
* Hybrid computer

Analogue Computer

* Analogue computer are designed to process analogue data.
* Analogue data is continuous data, that changes continually and cannot have discrete values. We can say that analogue computers are used where we don't need exact values such as speed, temperature, pressure and current.

Digital computer

Digital computer is designed to perform calculations and logical operations at high speed. It accepts the raw data as input in the form of digits or binary numbers (0 and 1) and processes it with programs stored in its memory to produce the output. All modern computers like laptop, desktops including smartphones that we used at home or office are digital computers.

Hybrid computer

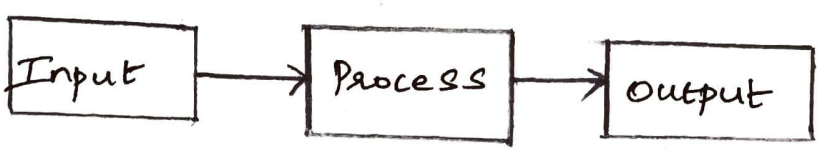
* Hybrid computers has features of both analogue and digital computer. It is fast like an analogue computer and has memory and accuracy like digital computers.
* It can process both continuous and discrete data.
* It accepts analogue signals and convert them into digital form before processing

Functionalities of a computer:

* Takes data as input
* Stores the data/instructions in its memory and use them when required.
* Processes the data and converts it into useful information
* Generates the output

Input - process - Output model

Computer input is called data and the output obtained after processing it, based on user's instructions is called information.



Computer Components

Any kind of computer consists of

* Hardware * Software

Hardware

It represents the physical components of a computer like its electronic parts. For example CPU, memory, Hard disk, Monitor, Printer, mouse etc.

Software

It represents the programs which perform different tasks on a computer system. It is a programming code which is executed by CPU, which can takes instructions from input devices like keyboard, mouse and can show output on output devices like monitor, printer etc.

Differences between Hardware and Software

Key	Hardware	Software
① Type	Hardware is a set of Physical parts of computer which actually executes the instruction.	Software is a program or set of instructions which are to be executed by CPU to do the intended task
② Development	A hardware is manufactured in factories	A software is developed engineered by software development companies
③ Dependency	A hardware cannot do any task with a Software instructing it	A software cannot execute if underlying hardware is not present
④ Tangible	A hardware can be touched being a physical electronic device.	Software being digital can be seen but cannot be touched.

Type	Hardware	Software
⑤ Categories	Hardware categories: Input devices, output devices, Storage devices, internal components of CPU and motherboard.	Software categories:- Programming, Application softwares and operating systems.
⑥ Virus Impact	Hardware remain unaffected by viruses.	Software is affected by virus being primary target.
⑦ Digital Transfer	A hardware can be only Physically transferred.	Software can be transferred electronically
⑧ Replacement	If hardware gets damaged, it is replaced with new one.	If software get damaged, it is reinstalled.

Computational Thinking

* Computational Thinking allows us to take a complex Problem, Understand what the problem is and develop possible solutions.

* Computational Thinking is a set of problem solving methods that involve expressing problems and their solutions in ways that a computer could also execute.

The four key techniques to computational Thinking:

* Decomposition
* Pattern Recognition
* Abstraction
* Algorithm

Decomposition:

* Breaking down the problems into smaller sections.

* Breaking down the problems can make complicated challenges more manageable. This enables other computational Thinking elements to be applied more effectively to complex challenges. The solutions to the smaller problems are then combined to solve the original, larger problem.

Pattern Recognition

* Recognizing if there is a pattern and determining the sequence.

* Examining the problem for patterns, or similarities to previously solved problems, can simplify the solution. Pattern Recognition can lead to grouping, organizing or streamlining problems for more efficient outcomes.

Abstraction:

* Generalization of problem - Focusing on the important information only, ignoring irrelevant detail.

* Abstraction allows us to create a general idea of what the problem is and how to solve it.

Algorithms

When solving a problem, it is important to create a plan for your solution. Algorithms are a strategy that can be used to determine the step by step instructions on how to solve the problem.

Identification of Computational Problems

* Many systems in the natural sciences can be represented by means of a mathematical expressions, known as mathematical Model. This expression may be fairly complex in form, and thus it may not be possible to make a reasonable application of the standard methods of evaluation to the expression.

* If this is true, then it is standard practice to make an attempt at evaluating the expressions by various approximation methods.

* For any problems that we must consider there are a number of steps that are suggested for one to take in the process of finding a computational solution of the problem, and then for assessing the viability of this solution.

The Five Step Process:

(1) Identify the problem.

(2) Express the problem in terms of a mathematical model.

(3) Construct a computational method for solving the model.

(4) Implement the computational method on a computer.

(5) Assess the results if there is a relatively large disparity between your solution and the actual solution or what you know can actually be true for the problem, then reassess the problem and try the process again.

Algorithm

An Algorithm is a set of instructions used for solving problems in a step-by-step manner. It is an ordered sequence of finite, well defined instruction for completing a task.

Characteristics of Algorithm:

* Algorithm has a finite number of inputs.
* Every instructions should be precise and unambiguous
* Ensure that the algorithm has proper termination.
* The desired output must be obtained only after the algorithm terminates.
* The algorithm can be applied to various set of inputs.

(Eg) Algorithm for Interchanging / Swapping two values.

Step 1 : Start the program
Step 2 : Read two numbers a, b
Step 3 : Set temp = a
a = b
b = temp
Step 4 : print a and b
Step 5 : Stop

Algorithm to find the radius of circle.

Step 1 : Start the program
Step 2 : Read radius r
Step 3 : calculate area = 3.14 * r * r
Step 4 : print area
Step 5 : Stop

Building Blocks Of Algorithm

An algorithm starts from an sequence of Instructions

* Instructions / statements
* State
* Control flow
* Functions

Instructions / Statements

An instruction written in a high level language. It directs to the computer to perform a action. It can be classified into simple statement, compound statement.

Simple Statement	Compound Statement
assignment : sum = 0	block : begin ..... end
goto : goto big	while : while ..... action
return : return sum	for : for (.....)
function call : add()	

State

The State of an algorithm is defined as its condition regarding current values or contents of the stored data.

Control Flow

The logic of the program may not always be a linear sequence of statements to be executed in order. It has been proven that any algorithm can be constructed from three building blocks.

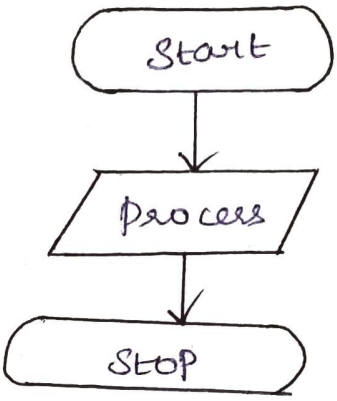
* Sequence * Selection * Iteration

Sequence control structure

The sequence logic is used for performing instructions one after another. The sequence control structures use top-down approach.

(e.g) Algorithm to find sum of 2 numbers

Step 1 : Start the program
Step 2 : Read two values a and b
Step 3 : Sum = a+b
Step 4 : Print sum
Step 5 : Stop

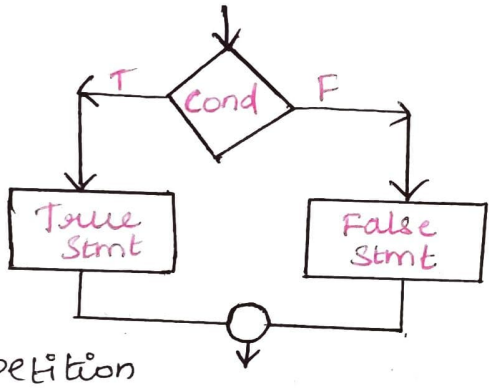


Selection Control Structure

The selection logic is used for making decisions. It is usually depicted as if ..., if ... else statements.

(e.g) Algorithm to test the 2 numbers are equal or not

Step 1 : Start
Step 2 : Read two numbers x, y
Step 3 : If x==y print "Equal" else print "not equal"
Step 4 : Stop



Iteration or repetition

which involves executing one or more steps for a number of times, can be implemented using constructs such as the while loops and for loops. These loops execute one or more steps until some condition is true.

Algorithm to find the sum of first n numbers

Step 1 : Start
Step 2 : Read n
Step 3 : Set i=1, sum=0
Step 4 : Repeat step 4 & step 5 while i<=n
Step 5 : Set sum=sum+i, Set i=i+1 (end of loop)
Step 6 : print sum
Step 7 : Stop

Function

Function is a subprogram which consists of blocks of code that performs a particular task, for complex problems the problem is divided into smaller and simpler tasks during algorithm design.

Function provide Reduction in line of code, code reuse, Better readability, easy to debug and test.

(e.g) Algorithm to find sum of 2 no's using function

Main function

Step1 : Start

Step2 : Call the function add()

Step3 : Stop

Sub fun add()

Step1 : function start

Step 2 : Read two values

Step3 : sum = a + b

Step 4 : Print sum

Step 5 : return

Notation

A series or system of written symbols used to represent numbers, amounts or elements in programming language is called as Notation.

* Pseudocode * Flowchart

Pseudocode

Pseudocode is a compact and informal high level description of an algorithm that uses the structure conventions of a programming language. "Pseudo" means initation, "code" means the set of statements or instructions written in a programming language. It is also called as "Program Design language" - PDL

Rules for writing pseudocode

* Write one statement perline
* Capitalize Initial keywords
* Indent to show hierarchy
* End multiline Structure
* Keep statements language independent

* Write one Statement perline

Each statement in the pseudocode should express just one action for the computer.

(e.g) TO calculate Student total and average

```
READ name, m1, m2, m3
CALC tot = m1+m2+m3
CALC Avg = tot /3
PRINT name, tot, Avg
```

* Capitalize Initial keywords

The keywords in the pseudocode should be written in all capital letters, because they begin the statement and they are command words, they give special meaning to the operation.

READ, WRITE, IF, ELSE, FOR ..... are keywords, that must be written in capital letters.

* Indent to show hierarchy

Indentation is the process of showing the "boundaries" of the structure. In case of sequence structure we would not have any indentation, but in the loop or selection structure we must use indentation

(e.g)

```
READ name, m1, m2, m3
CALC Tot = m1+m2+m3
CALC Avg = Tot/3
If Avg is greater than 90
    PRINT DISTINCTION
ENDIF
PRINT name
```

* End multiline structure

Each structure must be ended properly, which provide more clarity. It indicates where the loop or selection is completed or ended.

(e.g) ENDIF , ENDWHILE

* keep Statements language independent

Pseudocode can be written in any language.

Advantages:

* Converting Pseudocode to programming language is easy.
* It can be easily modified as compared to Flowchart
* It's implementation is very useful in structured design
* It can be read and understand easily

Disadvantages:

* There is no standardised style or format
* For a beginner it is more difficult to follow the logic or write pseudocode
* It is not visual

Flowchart

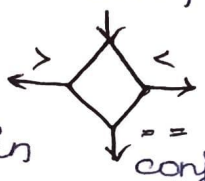
* A flowchart is a graphical or symbolic representation of a process. It is basically used to design and visualize the logic of the process.

When designing Flowchart, each step in the process is depicted by a different symbol and is associated with a short description.

Basic Guidelines to draw a flowchart.

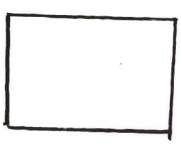
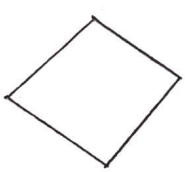
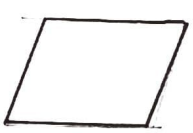
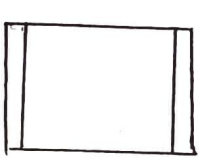
* The standard symbols should only be used.
* The arrow heads in the flowchart represents the direction of flow of control in the problem.
* The usual direction of the flow of procedure is from top to bottom or left to right.

* Only one flowline should come to a process symbol and only one flowline should come out from a process symbol 

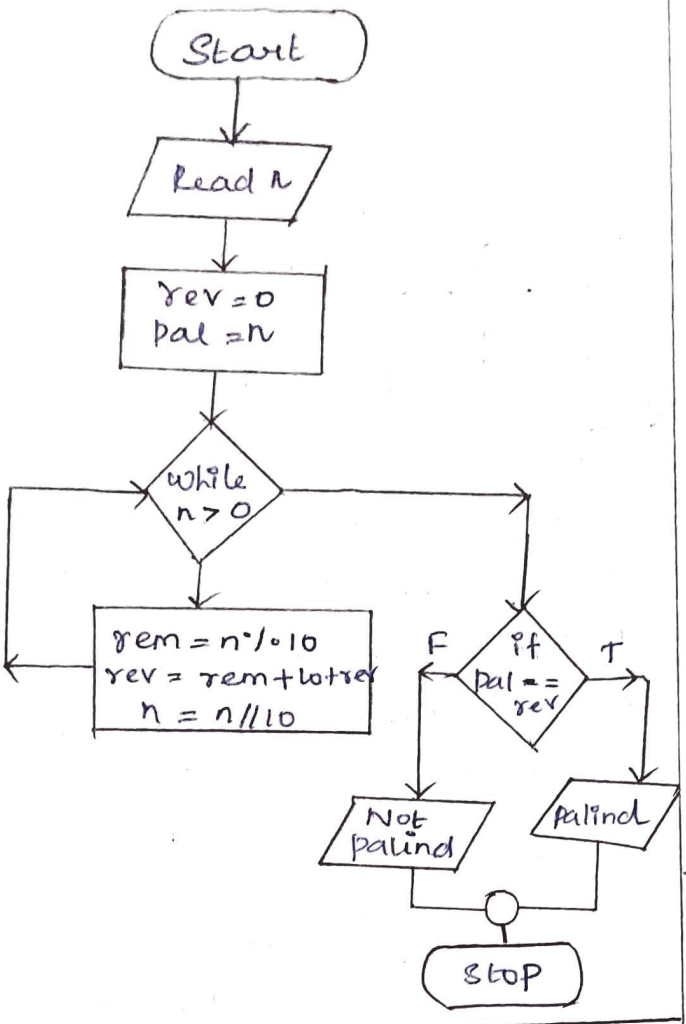
* only one flowline should enter a decision symbol, and two or three flowlines, one for each possible answer should leave the decision symbol. 

* only one flowline is used in conjunction with terminal symbol. 

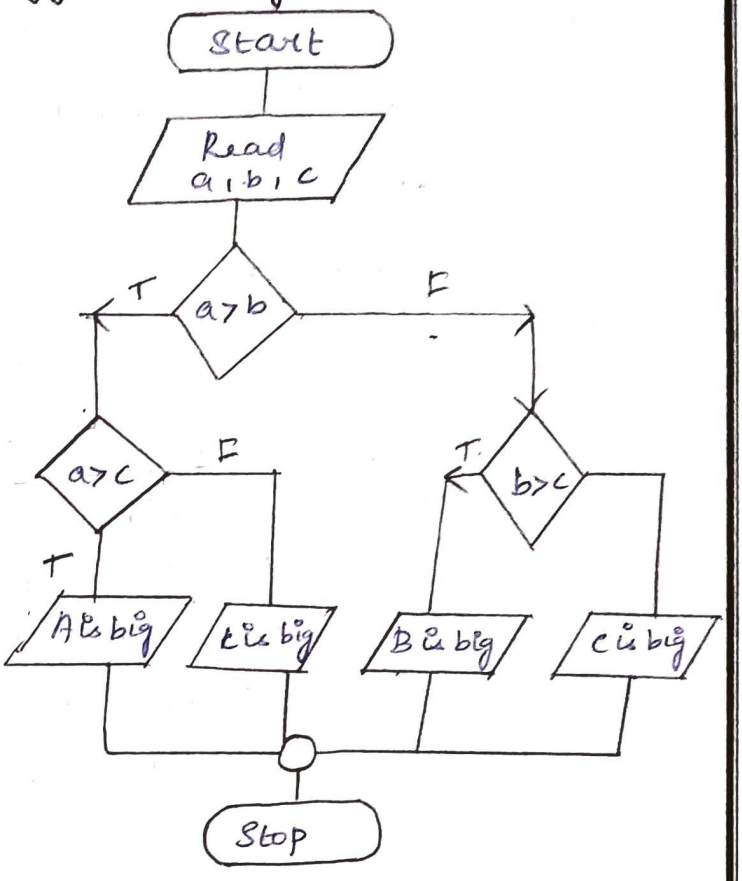
* Flowlines should not intersect with each other
* Ensure that the flowchart has a logical start and stop.
* Write with in the standard symbol briefly, As necessary use the annotation symbol to describe data or computational steps more clearly - - -[Read n values

Symbols	Description	Symbols	Description
	Terminal symbol Represent the start and stop of the program		Process Symbol It represents arithmetic and data movement instruction
	Decision symbol It is always used to depict a Yes/No que or True/False test		Input/output Used to get inputs from the user or display the results
	Flow Lines Represents the sequence of steps and direction of flow used to connect symbols.		Predefined processor (or) function It is a marker for another process step or series of process flow steps that are formally defined elsewhere
	connectors It is used to indicate a continued flow		

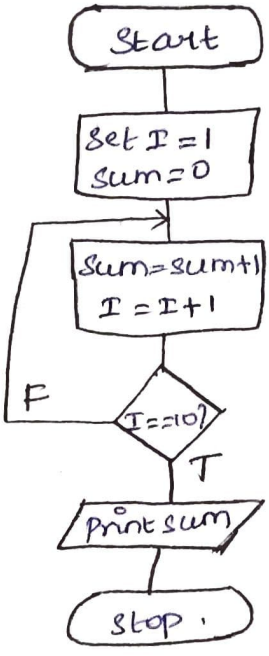
Draw a flowchart to check the given no is palindrome or not



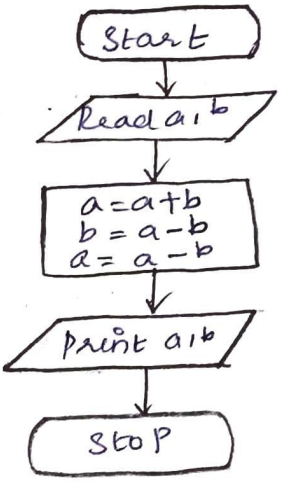
Draw a flowchart to find the biggest among three numbers.



To calculate sum of 10 natural numbers



Flowchart to swap 2 numbers



Advantages of flowchart	Limitations of using flowchart
* TO understanding logic clearly * Better communication * Effective analysis * Effective coding * Systematic Testing * Efficient Program maintanence	* complex logic * Alterations and modifications * Reproduction

Programming Language

A Programming language is a formal language that specifies a set of instructions that can be used to produce various kinds of output. Programming language generally consists of instructions for a computer.

It can be classified into

* Lowlevel Programming language
* Highlevel programming language

Lowlevel Programming Language:

Lowlevel progeamming language is a programming language that Provides little or no abstraction from a computer instruction set architecture. It refers either machine code or Assembly language.

machine code: It is the only language a computer can process directly without any transformation. It is a stream of binary data. (e.g) 0110110000001111

Assembly language:

Second generation programming language use a simple Processor called an assembler. Assembly language has little semantics or formal specification, being only a mapping of human-readable symbols, including symbolic addresses to Opcodes, addresses, numeric constants, strings and soon

In general assembler instructions have three components,

*Mnemonic - Easily remembered instruction name.

*Operand - The data to be used such a memory addresses or a register.

*Command - To make reading the code easier

(e.g.) ; LD ACC, #13500 ; Load the number into the accumulator

STO ACC, 16000 ; Store the contents of a memory location in acc

High level Programming Language

A High level language is any programming language that enables development of a program in a much more user-friendly programming context and is generally independent of the computer Hardware Architecture.

The first Highlevel language were introduce in 1950's. There includes BASIC, C, C++, COBOL, FORTRON, JAVA, PASCAL, PYTHON, PHP, RUBY etc....

BASIC → It is an easy to understand Programming language.

C → It is used for creating computer application.

C++ → System oriented applications, Graphic designing applications.

COBOL → Business, Finance and Administrative systems

FORTRAN → Scientific, Mathematical, Statistical and Engg type procedures.

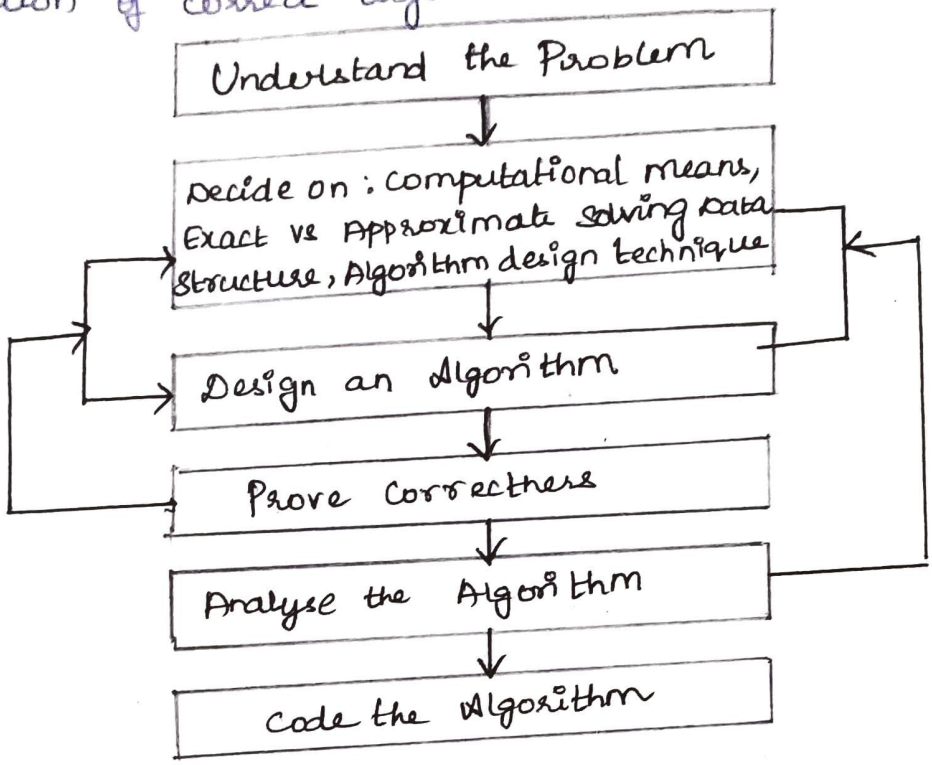
JAVA → Mobile Application, Web Apps, GOI app etc

PASCAL → Application Programming

PYTHON → Web Programming, Scripting language.

Algorithmic Problem Solving

Algorithmic problem solving is about the formulation and solution of problems, where the solution involves the principles and techniques that have been developed to assist in the construction of correct algorithm.



Understanding the problem:

Read the problem's description carefully to understand the problem statement completely. It is compared with earlier problems, if it is similar to them then the same algorithm is used, otherwise a newone have to be developed.

Determining the capabilities of the computational device.

After understanding the problem, the capabilities of the computing device should be known. For this the type of architecture, speed and memory availability of the device are noted.

Choice of computational devices like processor and memory is mainly based on space and time efficiency.

Exact / Approximation solution

An algorithm used to solve the problem exactly and produce correct result is called exact algorithm. If the problem is so complex and not able to get exact solution, then we have to choose an algorithm is called approximation algorithm. (e.g.), Extracting square roots, solving non-linear equation and evaluating definite integrals.

Select the appropriate data structures

A datatype is a well defined collection of data with a well defined set of operations on it. A data structure is basically a group of data elements that are put together under one name and which defines a particular way of storing and organizing data in a computer.

The elementary data structures are

* List → Allows fast access of data
* Sets → Treats data as elements of a set. Allows application of operations such as intersection, union and equivalence
* Dictionaries → Allows data to be stored as a keyvalue pair

Algorithm design Techniques

An algorithm design technique is a general approach to solving problems algorithmically, that is applicable to a variety of problems from different areas of computing.

Developing an algorithm is an art which may never be fully automated.

Methods of specifying an Algorithm

An algorithm is just a sequence of steps are instructions, that can be used to implement a solution. After writing the algorithm, it is specified either using a natural language or with the help of pseudocode or flowcharts.

Proving algorithms Correctness

Writing an algorithm is not enough. We need to prove that it computes solutions for all the possible valid inputs. This process is often referred to as algorithm validation. Algorithm validation ensures that the algorithm will work correctly irrespective of the programming language in which it will be implemented.

Analysing the Performance of algorithms

An algorithm is analysed to measure its performance in terms of CPU time and memory space required to execute the algorithm. This is a challenging task and is often used to compare different algorithms for a particular problem. The result of the comparison helps us to choose the best solution from all possible solutions.

Analysis of the algorithm also helps us to determine whether, the algorithm will be able to meet any Efficiency constraint that exists or not.

Code the Algorithm

The coding / Implementation of an algorithm is done by any suitable programming language. Implementing an algorithm correctly is necessary.

Simple Strategies for developing algorithm (Iteration, Recursion)

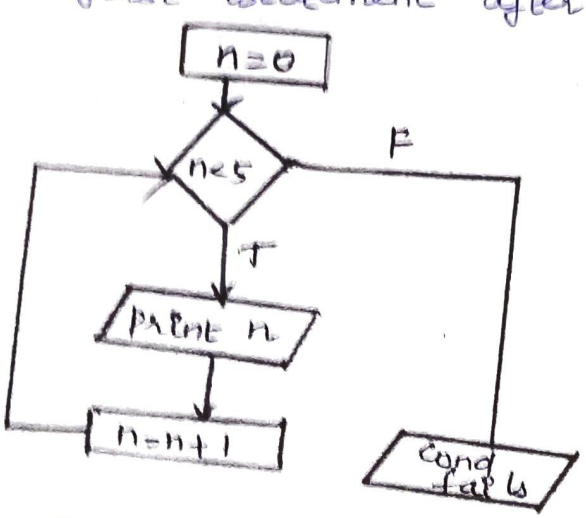
Iteration

Iteration or Repetition which involves executing one or more steps for a number of times until a condition is met. They use repetitive constructs like for loop and while loop. Key features are used to design a loop.

- Pass (or Iteration) - One complete execution of the body.
- loop entry - The point where flow of control passes to the body.
- loop test - The point at which the decision is made to re-enter the body or not.
- loop exit - The point at which the iteration ends and control passes to the next statement after the loop.
- Termination - The condition that causes the iteration to stop

while loop

In while loop the boolean expression is examined before each pass through the body of the loop. If the boolean expression is true, the loop body is executed. If it is false, execution proceeds to the first statement after the loop body.



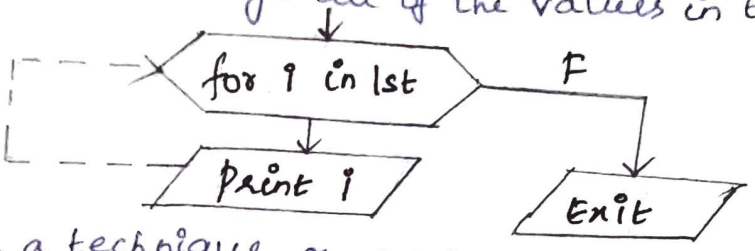
for loop

for loop is called as definite loops, because they executes an exact number of times.

Definite loops have explicit iteration variables that change each time through the loop. These iteration variables move through the sequence or set.

```
for i in [1,2,3,1,5]:        → element sequence
    Print(i)                 i → Iteration Variables.
```

The iteration variables iterates in the sequence. The body of code is executed once for each value in the sequence. The iteration variable moves through all of the values in the sequence.



Recursion

It is a technique of solving a problem by breaking it down into smaller and smaller subproblems until we get to a small enough problem that it can be easily solved. Recursion involves a function calling itself until a specified condition is met. Every recursive solution has two major cases

* **Base case**: In which the problem is simple enough to be solved directly without making any further calls to the same function.
* **Recursive case**: The problem is divided into simpler subparts. Second the function calls itself but with subparts of the problem obtained in the first step. Third the result is obtained by combining the solutions of simpler subparts.

(e.g); Calculate the factorial of an integer number

$n! = 1 \times 2 \times 3 \times \ldots \times n$ where n is an integer

we can also express this by $\boxed{n! = n * (n-1)!}$

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n * (n-1)! & \text{if } n > 0 \end{cases}$$

(e.g)

```
def fact(n):
    if n==1:
    else return 1
        return n * fact(n-1)
fact(3)  # function call.
```

Illustrative Problems

Find Minimum in a list

TO find the minimum value in an list is an easy task. Take the first number in the list and call it minimum. Compare the minimum's value with all other values in the list one by one. If we find the smaller element than the minimum call that element as minimum.

Algorithm:

Step1 : Start
Step2 : Read the count of numbers as n
Step3 : Set i = 0
Step4 : Read the first element as min
Step 5 : Repeat steps 6-8 while i<n-1
Step 6 : Read the next number as num
Step 7 : If min > num
set min = num
Step8 : Set i = i+1
Step9 : print min
Step10 : Stop.

(e.g); 12 4 7 3 9

let min = 12
compare min with the first input 4
(i.e) : min > num

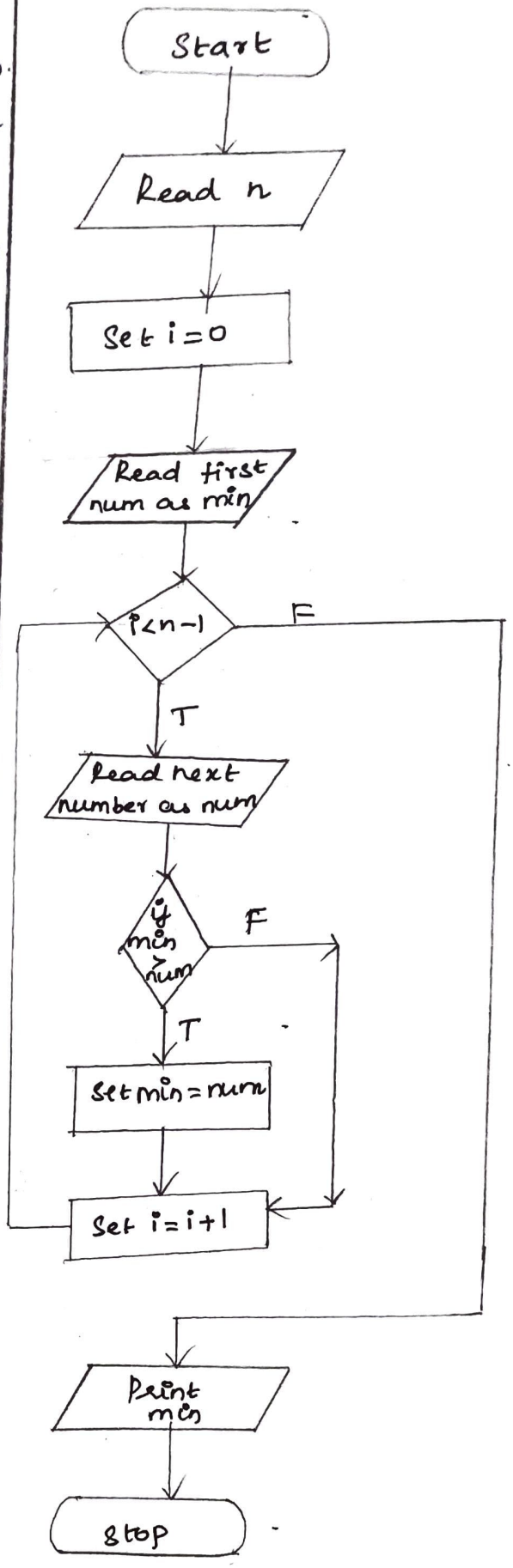
12 > 4 True so set min = 4
Now min = 4 compare this with the next input

4 > 7 False no change
Read next element 3, compare

4 > 3 True so set min = 3
Read next element as 9

3 > 9 False min = 3

TO find the minimum value by using this method, it requires n steps and need exactly n-1 comparisons.



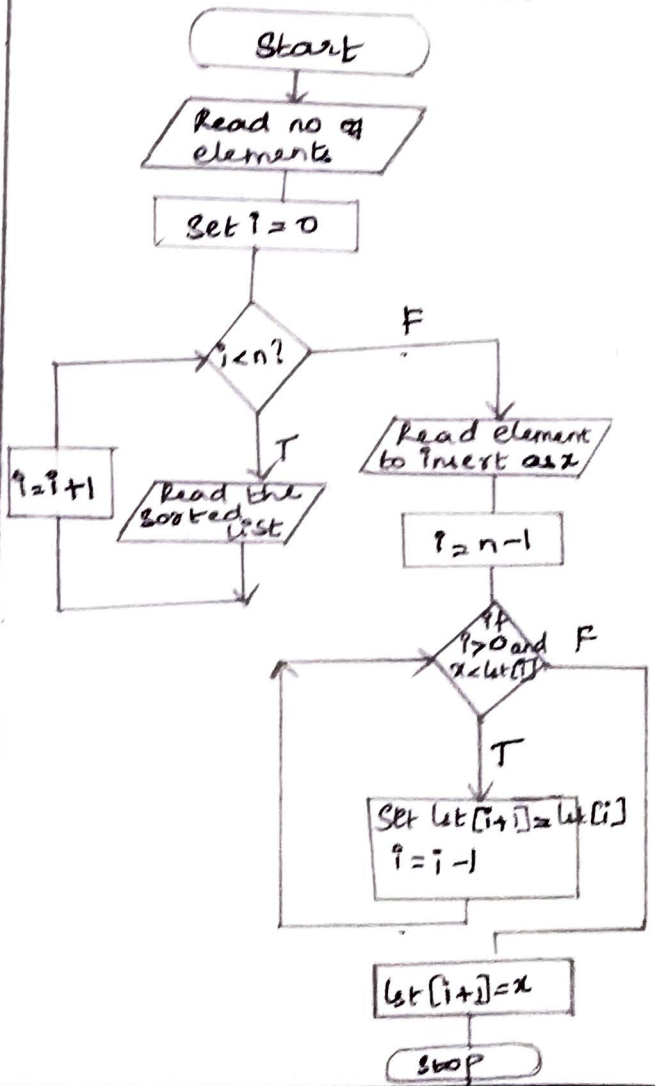
Insert a card in a list of sorted cards

Inserting a card in a list of sorted card is same as inserting an element into a sorted list of numbers. For this we start from the end of the list and compare the new element with the elements of the list to find a suitable position at which the new element has to be inserted, while comparing also shift the elements one step ahead to create a vacancy for the new element.

Position	0	1	2	3	4	5	Element to be inserted
original list	1	5	9	12	15		7
7 > 15 F	1	5	9	12		15	
7 > 12 F	1	5	9		12	15	
7 > 9 F	1	5		9	12	15	
7 > 5 F	1	5	7	9	12	15	

Algorithm

Step 1 : Start
Step 2 : Read number of elements n
Step 3 : Set i = 0
Step 4 : Repeat Steps 5 and 6 while i < n
Step 5 : Read the sorted list as lst[i]
Step 6 : Set i = i + 1
Step 7 : Read element to be inserted as x
Step 8 : Set i = n - 1
Step 9 : Repeat step 10 and 11 while i >= 0 and x < lst[i]
Step 10 : lst[i+1] = lst[i]
Step 11 : Set i = i - 1
Step 12 : lst[i+1] = x
Step 13 : Stop



Guess an Integer number in a range

The game is that the computer will randomly select an integer from 1 to n and ask the player to guess it. The computer reads the user input and tell that each guess is too high or too low. The good thing is that there is no limit on number of guess.

Algorithm

Step 1 : Start

Step 2 : Set i=0

Step 3 : Read the range of n

Step 4 : Set num as randomly selected number from 1 to n

Step 5 : Read val as a value guessed by the user

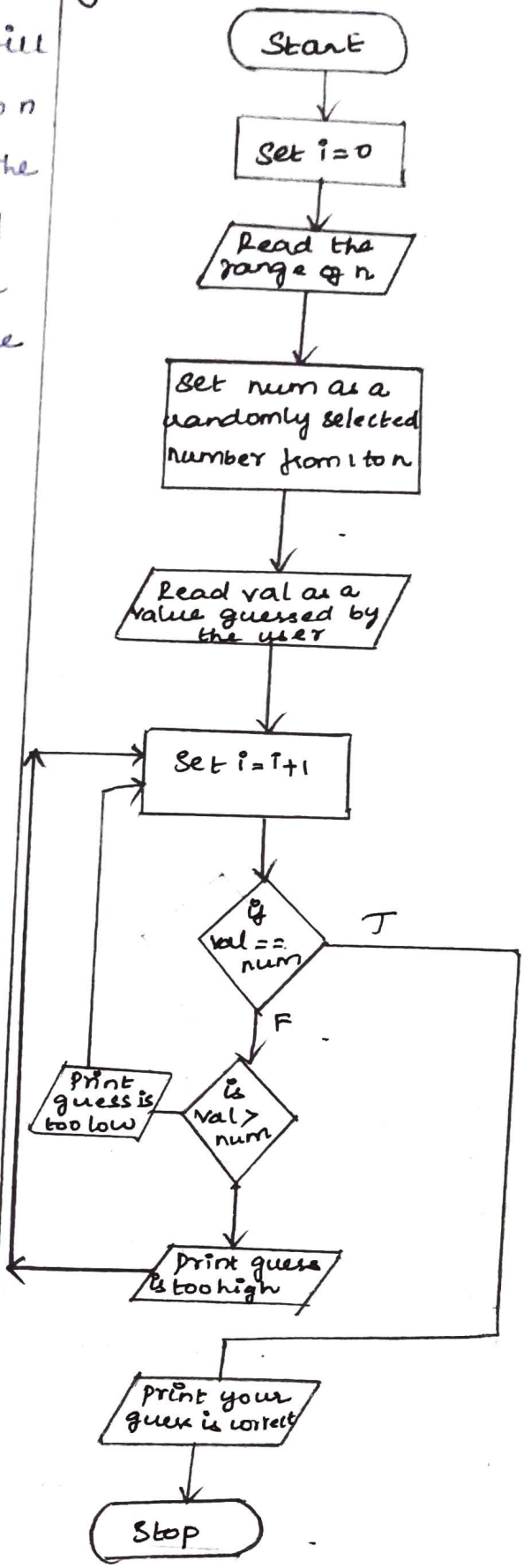
Step 6 : Set i = i+1

Step 7 : If val == num then print "Your Guess is correct" otherwise goto step 10

Step 8 : If val > num then print "Guess value is too high Play again" Goto step 5

Step 9 : If val < num then print "Value too low play again" Goto step 5

Step 10 : Stop



Tower of Hanoi

The tower of Hanoi is a mathematical game or puzzle. It consists of rods and number of disk of different sizes, which can slide onto any rod.

The objective of the puzzle is to move the entire stack to another rod, obeying the following rules.

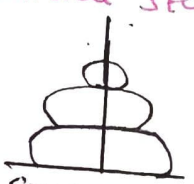
* Only one disk can be moved at a time.
* Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack.
* No disk may be placed on top of a smaller disk.

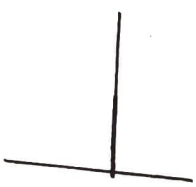
The minimal number of moves required to solve tower of Hanoi are 2^{n-1}, where n is the number of disks.

Example:

Let us understand this problem by 3 disks, so total of moves required = 7.

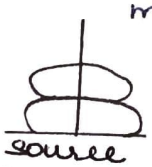
Initial step

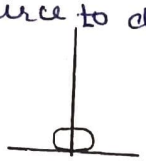




Step I

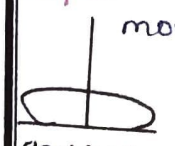
move disk from source to dest

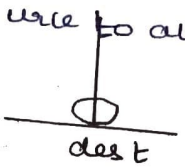




Step II

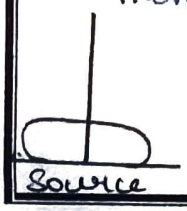
move disk from source to aux

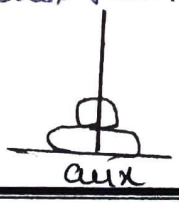




Step III

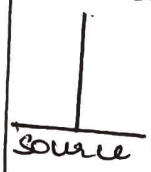
move disk from dest to aux

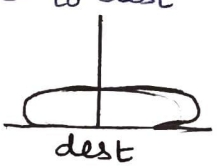




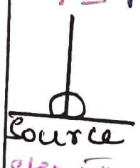
Step IV

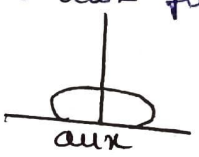
move disk from source to dest

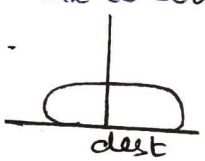




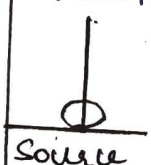
Step V move disk from aux to source

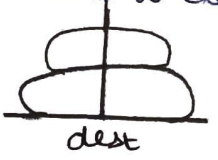




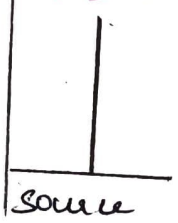


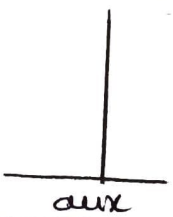
Step VI move disk from aux to dest

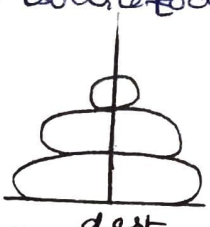




Step VII move disk from source to dest







Algorithm.

tower (n, source, aux, dest)

Step 1 : Begin

Step 2 : If n == 1

move disk from source to destination

then go to step 6

: else

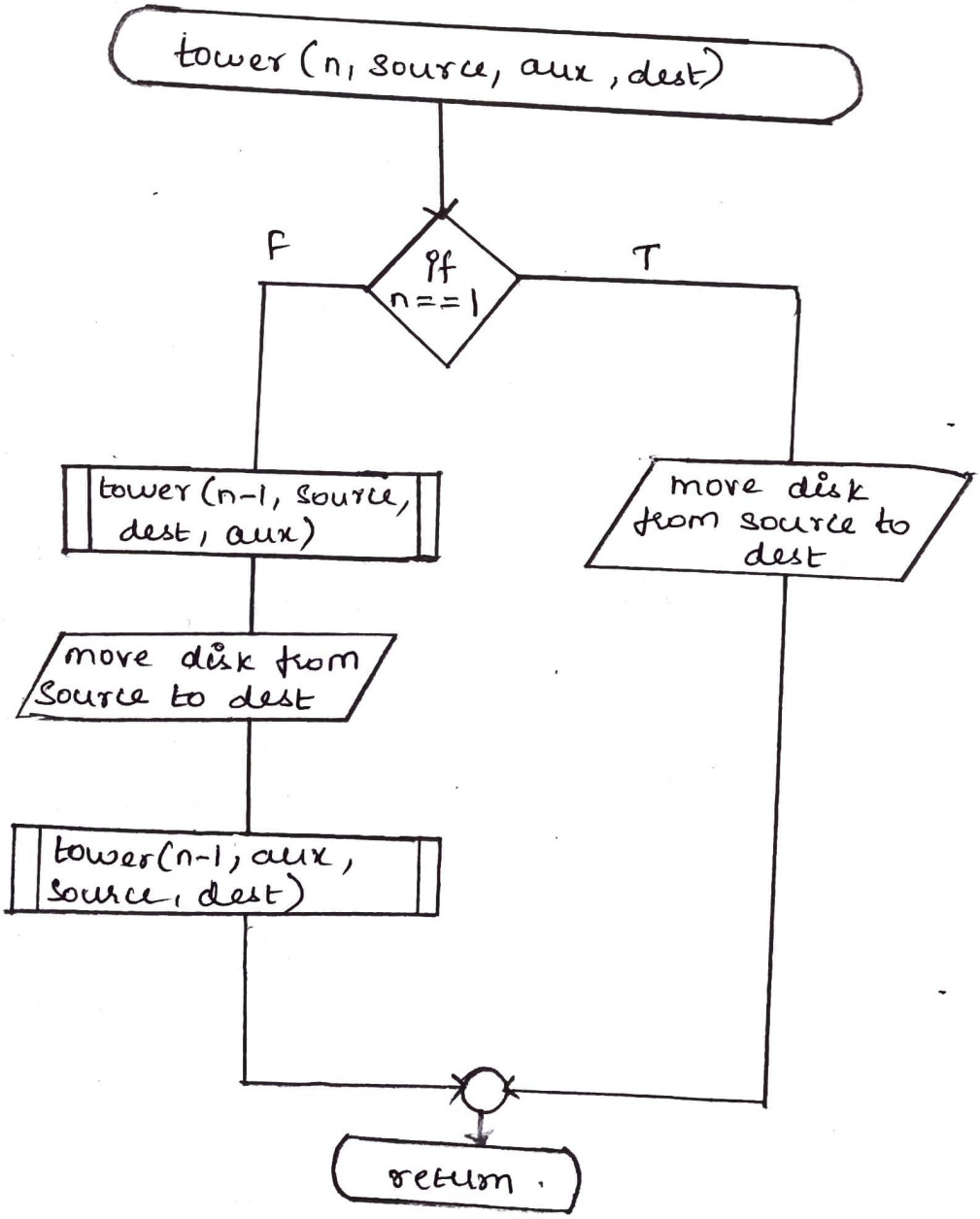
goto step 3

Step 3 : call tower (n-1, source, dest, aux)

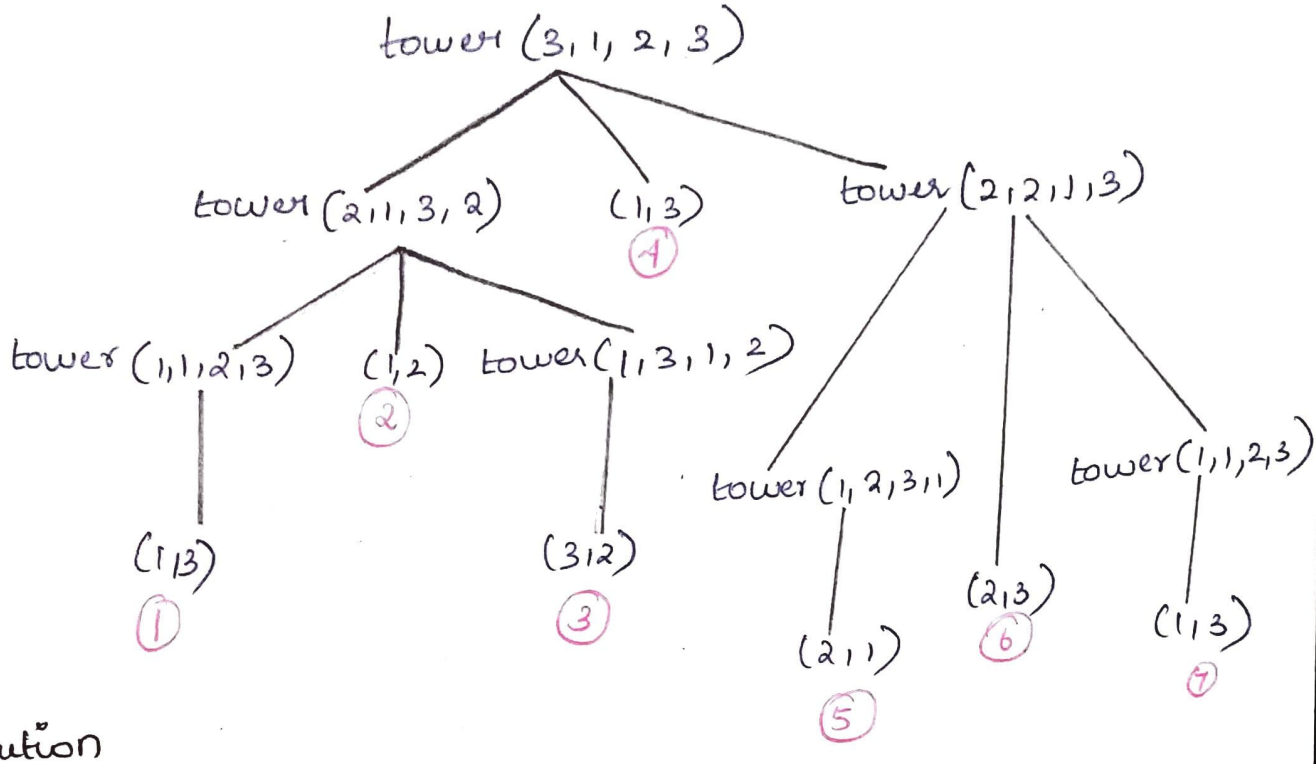
Step 4 : move disk from source to dest

Step 5 : call tower (n-1, aux, source, dest)

Step 6 : end.



Assume $n=3$, source $=1$, aux $=2$, dest $=3$



Solution

(1,3), (1,2), (3,2), (1,3), (2,1), (2,3), (1,3)

(1,3) move the disk from source to destination

(1,2) move the disk from source to aux

(3,2) move the disk from dest to aux

(1,3) move the disk from source to dest

(2,1) move the disk from aux to source

(2,3) move the disk from aux to dest

(1,3) move the disk from source to dest

Hence the disks moves from source to destination

UNIT - II DATA TYPES, EXPRESSIONS, STATEMENTS

Applications of python in solving various Engineering problems - Python Interpreter and interactive mode, debugging; values and Types: int, float, boolean, string and list; Variables, expressions, statements, tuple assignment, precedance of operators, comments; illustrative programs: exchange the values of two variables, circulate the values of n variables, distance between two points

Applications of python in solving various Engineering problems:

Introduction to python:

Python was developed by Guido Van Rossum during 1985-1990. python is a general purpose interpreted, interactive, object oriented and high level Programming language.

* python is interpreted:

Python is processed at runtime by the interpreter. So there is no need to compile a program before executing it.

* python is Interactive:

Programs in python works in Interactive mode which allows interactive testing and debugging a pieces of code.

* Easy to learn:

A python Program is clearly defined and easily readable. The structure of the program is very simple.

* Versatile:

Python supports development of a wide range of applications ranging from simple text processing to www browsers to games.

* Free and open source:

It is an open source software, so anyone can freely distribute it, read the source code, edit it.

High level language:

When writing programs in python, the programmers don't have to worry about the low level details.

Portable:

Python can run on a wide variety of hardware plat forms and has the same interface on all platforms.

Object oriented:

Python supports object oriented style or technique of Programming that encapsulates code within objects.

Applications of python:

Python is a high level general purpose programming language that is used to develop a wide range of applications including image processing, text processing, web and enterprise level applications using scientific and numeric data from network.

* Embedded scripting language.
* 3D software.
* Web development.

Python is an easily extensible language that Provides good integration with database and other web standards.

* GUI-based desktop applications
* Image processing and graphic design applications.
* Scientific and computational applications.
* Games
* Enterprise and business applications.
* Operating systems.
* Language development
* Network Programming and Prototyping.

Python. Interpreter.

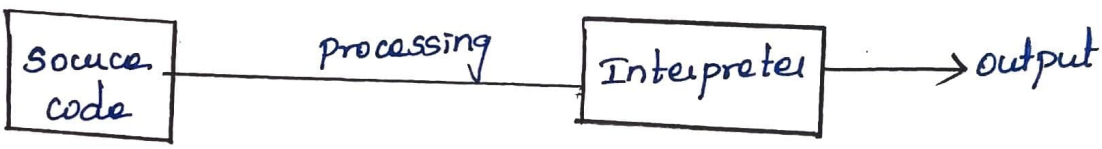
Interpreter:

To execute a program in a highlevel language by translating one line at a time is called interpreter.

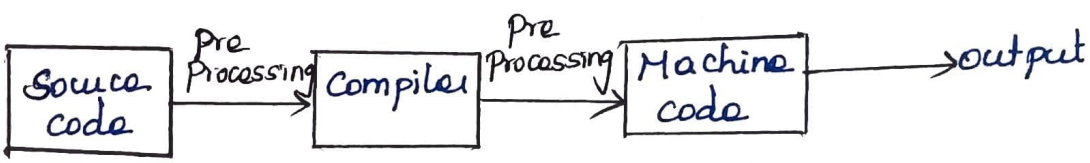
Compiler:

To translate a program written in a high level language into a low. level language all at once.

Interpreter:



Compiler:



Difference between compiler and Interpreter:

COMPILER	INTERPRETER
Compiler scans the entire program and translates it as a low level language.	Translates program one statement at a time.
Intermediate object code is Generated.	No Intermediate object code is generated.
Conditional control statements are Executes faster.	Conditional control statements are executes slower.
Memory Requirement is more, because object code is generated.	Memory requirement is less.

Program need not be compiled every time. Errors are displayed after entire program is checked. Ex : C, C++, compiler.	Every time higher level program is converted. Errors are displayed for every instruction Interpreted. Ex. Python, Ruby.

MODES OF PYTHON INTERPRETER.

Python Interpreter is a program that reads and executes Python code. It uses two modes of Execution.

* Interactive mode
* Script mode.

Interactive mode :

When we type the python program in interactive mode it displays the result immediately.

The cherron >>> is the prompt the interpreter uses to indicate that it is ready to enter code.

(eg) >>> 99+1
100
>>> Print ("Interactive mode in python")
Interactive mode in python.

ADVANTAGES :

* Python Interactive mode is good enough to learn, experiment or Explore.
* Working in interactive mode is convenient for beginners and for testing small codes.

DISADVANTAGE :

We cannot save the statements and have to retype all the statements once again to re-run them.

Script mode :

In script mode, type python program in a file and store the file with .py extension and use the interpreter to execute the contents of the file, which is called a script.

python script mode does not automatically display results. In order to see output from a python script, we will use the print statement. This statement takes a list of values and prints their string representation on the standard output file. The standard file output is typically directed to the terminal window.

Eg :
```
a = 25
b = 91
c = a+b
print("sum:", c)
```

output :

sum : 116.

Difference between Interactive and script mode.

Interactive mode	Script mode.
* A way of using the python Interpreter by typing statements as prompt	* A way of using the python Interpreter to read and execute statements in a script.
* Cannot save and edit the code	* can save and edit the code
* We can see the results immediately	* We cannot see the results immediately.

Python IDLE (integrated Development Learning Environment) works on different platforms. It contains the shell window an interactive interpreter, debugger and a multiwindow text editor that has features like python colorizing, smart indent and auto completion.

Debugging.

Debugging term is popularly used to process of locating and rectifying errors in a program. It is a systematic process of spotting and fixing the number of bugs or defects in a piece of software so that the software is behaving as expected. Debugging is harder for complex systems in particular when various subsystems are tightly coupled as changes in one system or interface may cause bugs to emerge in another.

Types of Errors:

* Syntax Errors
* Semantic Errors
* Runtime Errors

Data types available in python (or) Values and Types

The data stored in the memory can be of many types. Data type is the type of the data, that are going to access within the program.

Python has six basic data types

① Numeric ② string ③ List ④ Tuple ⑤ Dictionary ⑥ Boolean.

Numeric Data type:

Numbers as the name suggests, refers to a numeric value. We can use four types of numbers in python program. These include integers, long integers, floating point and complex numbers.

* whole numbers are referred to as integers.
* Bigger whole numbers are called long integers, long integers must have 'l' or 'L' as the suffix.
* The real numbers or fractional numbers are called floating point numbers. A floating point number is accurate upto 15 decimal places.
* Complex numbers are written in the form x+yj, where x is the real part and y is the imaginary part.

Eg Integer

```
>>> a=46298
>>> type(a)
<class 'int'>
```

Long

```
>>> x=17496532L
>>> type(x)
<class 'long'>
```

Float

```
>>> b=73.95
>>> type(b)
<class 'float'>
```

Complex:

```
>>> x=complex(5,8)
>>> type(x)
<class 'complex'>
```

Strings :

String is a sequence of characters, which may consist of letters, numbers, special symbols or a combination of these types represented by single ' ' or double " " quotes.

It is an Immutable data type, which means we cannot modify the string once it is created.

If we want to specify multiline string we can use triple quotes (''' ..... ''')

(eg)

```
>>> str = "university"
>>> type (str)
<class 'string'>
```

```
>>> str = 'hello'
>>> type (str)
<class 'string'>
```

```
>>> str = "python is very
easy to learn'''
>>> type (str)
<class 'string'>
```

A string consisting of only a pair of matching quotes are called empty string.

(eg) ' ' (or) " "

List :

Python offers a range of compound datatypes often referred to as sequence. List is an ordered sequence of values of any data types (int, float, string ...).

A list is created by placing all the elements inside a square bracket [ ], separated by commas.

(eg)
```
>>> lst = [10, 20, 98.5, "program", 'welcome']
>>> print (lst)
10, 20, 98.5, "program", 'welcome'
```

To access element from a list:

(i) List index.

We can use the index operator [] to access an element in a list. Index starts from 0. If a list having 7 elements will have index from 0 to 6.

(eg)
```
>>> lst = ['a', 'e', 'i', 'o', 'u']
>>> print (lst[1])
e
```

(ii) Negative Indexing:

Python allows negative indexing for its sequences. The index of -1 refers to the last item in the list, -2 to the second last item and so on.

```
>>> lst = ['abc', 1, 2, 3, 'xyz', 14.96]
>>> print (lst[-1])
14.96
```

Tuple:

A Tuple is a sequence of values. The values can be of any type and they are indexed as integers. Tuples are immutable. Tuples are comma separated list of values.

```
>>> tup = (10, 20, 'a', 'b', 'c')  (or)  tup = 10, 20, 'a', 'b', 'c'
```

It is not necessary to enclose tuples in parenthesis

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

(eg)
```
>>> t1 = 'a',
>>> type (t1)
<class 'tuple'>
```

Another way to create a tuple is by using built-in function 'tuple'. If we create a tuple with no arguments that is referred as empty tuple.

eg)
```
>>> t1 = tuple ('fruit')
>>> print (t1)
    ('f', 'r', 'u', 'i', 't')
>>> print t1[2]
    u.
```

Tuples are immutable, which means we cannot update or change the values of tuple elements. But we can replace one tuple with other.

```
>>> t = (1, 2, 3, 4, 5)
>>> t = ('a', ) + t[1: ]
>>> print(t)
    ('a', 2, 3, 4, 5)
```

Dictionary:

Dictionary datatype is a kind of hash table. It contains key_value pairs. Dictionaries are enclosed in curly braces {} and values can be assigned and accessed using square brackets.

A dictionary key can be almost any python type.

eg)
```
>>> dict = {'Name': 'AAA', 'Age' : 20, 'Add': 'xxx'}
>>> Print ( dict. keys ())
    dict-keys ( ['Name', 'Age', 'Add'])
>>> print (dict . values ())
    dict. values ( ['AAA', 20, 'xxx'])
>>> Print (dict)
    {'Name': 'AAA', 'Age': 20, 'Add': 'xxx'}
```

Boolean

The simplest built in type in Python is the bool type, it represents the truth values True or False.

(eg)
```
>>> x = True
>>> type(x)
<class 'bool'>

>>> y = False
>>> type(y)
<class 'bool'>
```

None Type :

None is a special constant in python. It is a Null value. Compare anything other than None will always return false. We can assign None to any Variable, but we cannot create other None type objects.

(eg)
```
>>> type (None)
<class 'NoneType'>
>>> None == 0
False
>>> a = None
>>> a == None
True.
```

Variables, Expressions and Statement.

Variables :

A Variable is a name that refers to a value. Variables are reserved memory locations that stores values. To be identified easily each variable is given an appropriate name.

(eg)
```
>>> subject = "python Programming"
>>> amt = 450
>>> pi = 3.14
```

The first statement creates a variable subject and assigns python programming to it, the second statement creates the variable amt and assigns 450 to it, the third statement creates the variable pi and assigns the value 3.14. Variables are examples of identifiers.

Rules:

* The first character of an identifier must be an underscore ('_') or a letter.
* Identifiers are case sensitive (eg) x and X are not the same.
* Punctuation characters such as @, $ and % are not allowed with in identifiers.
* keywords cannot be used as identifiers.
* Identifier can be of any length.

Expressions and statements:

An expression is a combination of values, variables and operators. An expression in python is any valid combination of operator and literals and variables.

(eg) sum = a+5 # arithmetic expression

a>b , a==b # Relational expression.

a and b # logical expression.

A statement is an instruction that the python interpreter can execute.

In python we have two sets of statements.

* Assignment statement
* Print Statement.

Assignment statements :

Assigns the value to a variable.

```
>>> a=100.
```

Print statement :

```
>>> print (a)
    100
```

Tuple assignment :

An assignment to all of the elements in a tuple using a single assignment statement

By using this feature allows a tuple of variables on the left of an assignment to be assigned values from a tuple on the right of the assignment.

(eg)
```
a,b = 10,20
print (a)
print (b)
```

(eg) Swap two Variables - Tuple assignment.

```
a=10
b=30
a,b = b,a
print (a,b)
```

output :

30,10.

Note :

The number of variables on the left and the number of values on the right have to be same..

Comments

* A comment in python starts with the hash character # and extends to the end of the physical line.
* Making use of comments in python is very easy, you can include a comment line into your code.
* Comments can be used to explain python code and make the code more readable.
* Comments can be used to prevent execution when testing code.
* It is also possible to use Triple Quotation (""...."") for multiline Comments.

Types of Comments in python:

There are three main kinds of comments in python.

They are.

* Single-Line Comments
* Multi line Comments.

Single line Comments:

A single line comment begins with a hash (#) symbol and is useful in mentioning that the whole line should be considered as a comment until the end of the line.

Eg `n=50 : # 50 value assigns to a Variable n.`

Multi line Comments:

Multi line Comment is useful when we need to Comment on many lines. In python triple double quote (""" """), (' ') and single quote (' ') are used for Multiline Commenting.

Eg
```
""" Author : Arun
    Dept : CSE
    Section : A """
```

Literals

Literal is a raw data given in a variable (or) constant. In python there are various types of literals.

* Numeric Literals
* String Literals
* Special literals
* Boolean literals

Numeric Literals :

Numeric Literals are immutable. Numeric literals can belong to 3 different Numerical Types.

* Integer literal * float literal * Complex literal.

Regular integer : Normal integer numbers.

(eg) >>> a = 525
>>> print (a) ⇒525

Octal literals (base 8) :

To indicate an octal literal we will use the prefix 0o (or) 0O (zero followed by upper (or) lower case O)

(eg) >>> x = 0ob output : b
>>> print (x)

Hexadecimal literal (base 16)

To indicate hexa decimal literal we will use the

Prefix 0x (or) 0X

(eg) >>> x = 0x Ab
>>> print (x)
166

(ie)
A b
b × 16^0 = b
A × 16^1 = 160 (+)
166

float literal : >>> f=45.63 >>> print(f) output : 45.63
Complex literal : >>> a = 4+3.5j >>> print (a) output : 4+3.5j
Boolean literal : >>> a = True >>> print (a) output : True
String literal : >>> s = "String" >>> print (s) output : String literal
Special literal : >>> a = none >>> if a==none output : Special literal

Precedence of operators

Operators are the constructs that are used to manipulate the value of operands. Some basic operands include +, -, * and /.

Python support different types of operators

* Arithmetic operators
* Comparison operators
* Assignment operators
* Logical operators
* unary operators
* Bitwise operators
* Membership operators
* Identity operators

Operators precedence and Associativity :-

When an expression has more than one operators then it is the relative priorities of the operators with respect to each other that determined the order in which the expression will be evaluated.

Operator	Description.
**	Exponentiation
~, +, -	Complement, unary plus, unary minus
*, /, %, //	Multiply, divide, modulo and floor division.
+, -	Addition and subtraction
>>, <<	left and right shift operator.
&, ^, |	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

ILLUSTRATIVE PROGRAMS

① Exchange the values of two variables:

(i) using third variable:

```
a = int (input ("Enter the first number"))
b = int (input ("Enter the second number"))
Print ("Before swap")
print ("a =", a, "b=", b)
temp = a
  a = b
  b = temp
Print ("After swap")
Print ("a =", a, "b=", b)
```

Output:

```
Enter the first number : 2
Enter the second number : 3
Before swap
  a = 2, b = 3
After swap
  a = 3, b = 2.
```

(ii) without using third variable:

```
a = int (input ("Enter the first number"))
b = int (input ("Enter the second number"))
Print ("Before swap")
print ("a =", a, "b=", b)
a = a+b
b = a-b
a = a-b
print ("After swap")
print ("a =", a, "b=", b)
```

output:

```
Enter the first number : 2
Enter the second number : 3
Before swap
  a = 2, b = 3
After swap
  a = 3, b = 2.
```

③ Circulate the values of n variables :-

```
def cir (l, n):
    for i in range (0, n):
        j = len(l) - 1
        while j>0:
            temp = l[j]
            l[j] = l[j-1]
            l[j-1] = temp
            j = j-1
        Print (i, 'Rotation', l)
l = [1, 2, 3, 4, 5]
cir (l, 3)
```

Output

0 Rotation [5, 1, 2, 3, 4]
1 Rotation [4, 5, 1, 2, 3]
2 Rotation [3, 4, 5, 1, 2]

④ Distance between two points :

```
import math
def distance (x1, y1, x2, y2):
    dist = math.sqrt ((x2-x1)**2 + (y2-y1)**2)
    return dist
d = distance (3, 4, 15, 16)
Print ("Distance between two points', d)
```

Output :

Distance between two points

16.970562748477714.

Extra Programs.

① Program to find the area and circumference of a circle.

```
r = float (input ("Enter the radius "))
area = 3.14 * r * r
Circum = 2 * 3.14 * r
Print ("Area :", area)
Print ("Circum:", Circum)
```

Output:

Enter the radius 3
Area : 28.259999999999
Circum : 18.84

② Program to Compute simple Interest.

```
P = int (input ("Enter the pamt "))
n = int (input ("Enter no. of years "))
r = float (input ("Enter the rate of Int "))
SI = (p*n*r) /100
print ("simple Int =", SI)
```

Output:

Enter the pamt 10000
Enter no. of years 5
Enter the rate of Int 12
Simple Interest = 6000.0

③ Python Program to Convert Celcius to faranheit.

```
C = float (input ("Enter the centigrade"))
f = (1.8 * C) + 32
Print ("The faranheit is =", f)
```

Output:

Enter the centigrade : 45
The faranheit is = 113.0

④ Program to find the sum and avg of three numbers.

```
a = int (input ("Enter a"))
b = int (input ("Enter b"))
c = int (input ("Enter c"))
Sum = a+b+c
avg = sum/3
Print ('sum:', Sum, 'Avg:', avg)
```

output:
Enter a 100 Enter b 200 Enter c 300
Sum 600, Avg 300.

⑤ Python Program to convert faranheit to centigrade.

```
f = float (input ("Enter the faranheit:"))
C = (f-32) / 1.8
Print ("centigrade =", C)
```

Output:

Enter the faranheit Value = 113
centigrade = 45.0

⑥ Program to exchange the value of two variables using tuple assignment

```
a = int (input ("Enter a"))
b = int (input ("Enter b"))
Print ("Before Swap")
Print ("a =", a, "b=", b)
a, b = b, a
Print ("After swap")
Print ("a =", a, "b=", b)
```

Output:

Enter a 10 Enter b 20
Before swap a=10, b=20
After swaps a=20, b=10.

⑦ Program to calculate area of a triangle (using Heron's formula)

```
a = float (input ("Enter the 1st side of
                  the triangle"))
b = float (input ("Enter the 2nd side of
                  the triangle"))
c = float (input ("Enter the 3rd side of
                  the triangle"))
s = (a+b+c)/2
area = (s*(s-a)*(s-b)*(s-c))**
                              0.5
print ("Area = ", area)
```

output:

```
Enter the 1st side of the triangle 12
Enter the 2nd side of the triangle 18
Enter the 3rd side of the triangle 10
Area = 56.568542494924.
```

Two Marks

1. Define python ?

python is an object oriented, high level language, interpreted, dynamic and multipurpose programming language.

2. Give the features of python

* Easy to use
* Expressive language
* Interpreted language
* Cross-platform language
* Free and open source
* object oriented language
* Extensible.

3. What is python interpreter ?

The engine that translates and runs python is called the python interpreter. There are two ways to use it: immediate mode and script mode. The >>> is called the python prompt. The interpreter uses the prompt to indicate that it is ready for instructions.

4. What is meant by value in python ?

A value is one of the fundamental things - like a letter or a number - that a program manipulates.

5. List the standard data types in python ?

Types : * Numbers * Strings * List * Tuples * Dictionary.

6. Difference between Interactive mode and script mode.

Interactive mode	Script mode.
1. A way of using the python interpreter by typing statements at prompt.	1. A way of using the python interpreter to read and execute statements in a script.
2. cannot save and edit the code	2. Can save & edit the code.
3. We can see the results immediately	3. We can not see the results immediately.

7. Listout the rules to be followed by an identifier.

* Identifier can be a combination of letters, digits & underscore.
* keywords cannot be used as identifiers.
* An identifier cannot start with a digit.
* We cannot use any special symbols like !, @, # ... in an identifier
* Identifiers can be of any length.

8. Define reserved words in python?

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 & structure of the python language.

(Ex) False, True, if, While, not, try.....

9. Define function.

Function is a subprogram which consists of sets of instructions used to perform a specific task. A Large program is divided into small building blocks called function.

10. Define Expressions and statements ?

Expressions :

An Expression is a combination of values, variables and operators. Eg a*b+c
a=b/c

Statements :

Instructions that a python interpreter can executes are called statement.

eg a=43 # assignment statement
print(a) # print statement.

11. Define 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, function definitions do not alter the flow of execution of the program.

12. Define parameters and arguments.

Parameters are the values provided in the parenthesis when we write function header.

(eg) def sum (a,b):

Arguments are the values provided in function call

(Eg) sum(x,y).

13. Define modules in python.

A module is a file containing python definition, function, statements and instructions.

14. Mention the types of arguments in python ?

* Keyword arguments
* Default arguments.
* Required arguments
* Variable length arguments.

15. List Some built in modules in python ?

* math module
* random
* threading
* Collections
* String
* time
* tkinter.

UNIT III

Control Flow, Functions, Strings

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

Conditionals: Boolean values

* The boolean can be one of two values, either True or False we use booleans in programming to make comparisons and to control the flow of the program.

* The equal to (==) operator compares two operands and produces True if they are equal and False otherwise.

(e.g) >>> 50 == 50 >> 99 == 98

True False

* True and False are special values that belong to the type bool; they are not strings:

(e.g) >>> type(True) >>> type(False)

<type 'bool'> <type 'bool'>

Operators

Operators are the constructs that are used to manipulate the value of operands. Python supports different types of operators:

* Arithmetic Operators
* Comparison operators
* Assignment operators
* Logical operators
* Bitwise operators
* Membership Operators
* Identity operators

Arithmetic Operators.

Operator	Description	Example a=21, b=5
+	Addition : Adds the operand	>>> print (a+b) 26
−	Subtracts right operand from the left operand or unary minus	>>> print (a-b) 16
*	Multiply two operands	>>> Print (a*b) 105
/	Divide left operand by the right one	>>> print (a/b) 4.2
%	Modulus - remainder of the division of the left operand by the right	>>> a % b 1
//	Floor division : Divides the operands and return the quotient	>>> a // b 4
**	Exponent : performs exponential calculation (i.e) left operand raised to the power of right	>>> b ** 2 25

Comparison Operator

Comparison operators also known as relational operators are used to compare the values on its either sides and determines the relation between them.

Operator	Description	Example a=25 b=50
==	Returns True if the two values are exactly equal	>>> print (a==b) False
!=	Returns True if the two values are not equal	>>> print (a!=b) True
>	Returns True, if the value at the operand on the left side is greater than right	>>> print (a>b) False

Operator	Description	Example a=25, b=50
<	Returns True if the left operand is less than right	>>>print (a<b) True
>=	Returns True if the left side value is greater than or equal to its right	>>>print (a>=b) False
<=	Returns True if the right side value is greater than or equal to its left	>>> print (a<=b) True.

Assignment Operator:

As the name suggests assign value to the operands. The in place operator is also known as short cut operators, that includes +=, -=, *=, /=, //=, %= and **=.

Operator	Description	Example a=10, b=5
=	Assign value of the operand on the right side of the operator to the operand on the left	a=b
+=	Adds and assign: Adds the operand on the left and right side of the operator and assigns the result to the operand on the left	a+=b (i,e) a=a+b
-=	Subtract and assign	a-=b (i,e) a=a-b
*=	Multiply and assign	a*=b (i,e) a=a*b
/=	Divide and assign	a/=b (i,e) a=a/b
%=	Modulus and assign	a%=b (i,e) a=a%b
//=	Floor division	a//b (i,e) a=a//b
**=	Exponent and assign	a**=b (i,e) a=a**b

Logical Operators:

Python supports three logical operators.

* logical and
* OR
* logical not

The logical expressions are evaluated from left to right.

* and
* or
* not

The logical and, or operator is used to evaluate two conditions or expressions with relational operators. If expressions on both the side of the logical operator are true, then the whole expression is true.

(e.g) (a>5) and (b>10)
(a>5) or (b>10)

Bitwise Operator.

Bitwise operators perform operations at the bit level.

Operator	Description	Example.
&	Bitwise AND - The bit in the first operand is AND with the corresponding bit in the second operand.	a=4, b=6 a ⇒ 0100 b ⇒ 0110 0100 a&b ⇒ 4
|	Bitwise OR - The bit in the first operand is ORed with the corresponding bit in the second operand	a = 0100 b = 0110 0110 a|b = 6
^	Bitwise XOR. The bit in the first operand is XORed with the corresponding bit in the second operand	a = 0100 b = 0110 0010 a^b = 2
~	Bitwise Not (or) Complement is a unary operator. It performs logical Negation on each bit of the operand	a = 4 0100 ~a = 1011 = 11
>> (Right Shift)	The operator shifts the specified number of bits discarding the right most bits	a = 22 a>>2 (i.e) 1011?
<< (left shift)	The data is shifted to the left by the specified number of bits and the trailing bits are replaced by zero	a=22 a<<2 = 88

Membership Operator

in and not in are the membership operators. They are used to test whether a value or variable is found in sequence (string, list, tuple, set and dict)

Operator	meaning	eg x=[1,2,3]
not in	True if value/var is not found in sequence.	5 not in x True
in	True if value/var is found in sequence	1 in x True

Identity Operator.

They are used to check if two values are located on the same part of the memory.

Operator	meaning	Example
is	True if the operands are identical (refer same obj)	a=[1,2,3], b=[1,2,3] a is b False
is not	True if the operands are not identical.	a is not b True

Decision making Statements

A decision is when a program has more than one choice of actions depending on a variable's value. In python, decision making statement can be classified into three types:

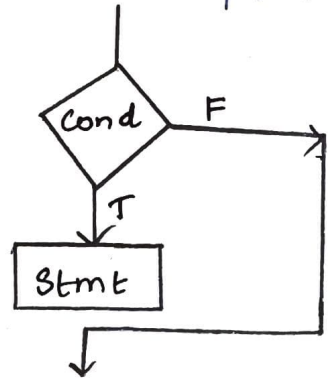
* conditional (if)
* alternative (if-else)
* chained conditional (if-elif-else)

Conditional statement (if):

The if Statement is the simplest form of decision control Statement. An if statement is a selection control statement. An if st on the value of a given Boolean Expression.

Syntax:

```
if condition :
    True statement
```

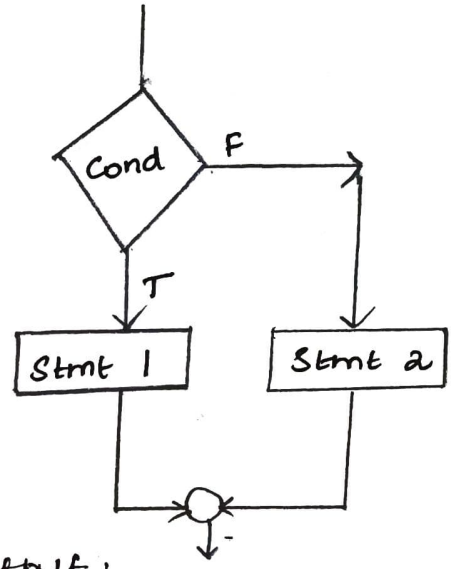


(e.g)
```
a = int(input('Enter a value'))
if(a>0):
    print('The given no is positive')
Print(a)
```

Output :-
Enter a value
20
The given no is positive
20

If-else (alternative statement)

The second form of the if statement is alternative execution, in which there are two possibilities and the condition determines which one gets executed. First the condition is evaluated, if the condition is true statement block 1 is executed otherwise statement block 2 is executed.



Syntax:

```
if condition :
        True Statement
else:
        False statement
```

(e.g)
```
# Pgm to check the given no is
odd or even.
n = int(input("Enter a number"))
if(n%2==0):
    print("Number is even", n):
else:
    print("Number is odd ", n)
```

Output:
Enter a number
10
Number is even, 10

Chained conditional (if-elif-else)

This statement is used to test additional conditions apart from the initial test expressions.

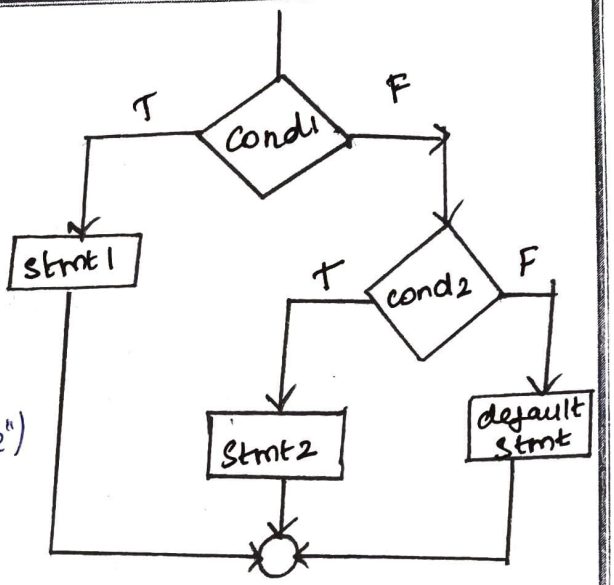
Syntax:

```
if (cond 1):
    statement 1
elif (cond 2):
    statement 2
else:
    default
    statement.
```

A series of if-elif statements can have a final else blocks, which is called if none of the if or elif expressions is True.

(e.g): TO test the given no is positive, negative or equal to zero.

```
n=int(input ("Enter a number"))
if(n==0):
    Print ("The entered number is zero")
elif (n>0):
    Print("The entered number is positive")
else:
    Print ("The entered number is negative" )
```



O/P

```
Enter a number
2
The entered number is positive.
```

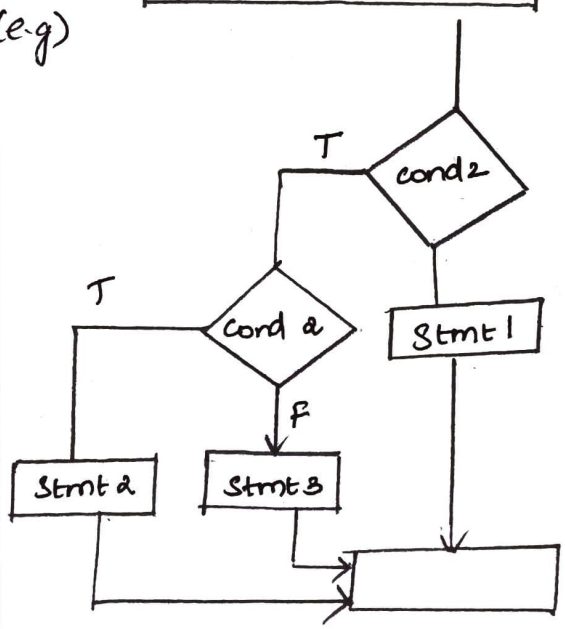
Nested if statement:

A statement that contains other statement is called a compound statement. To perform more complex checks, if statements can be nested.

Syntax:

```
if (test cond1):
    if(test cond 2):
        stmt block 2
    else:
        stmt block 3
else:
    Stmt block 1
```

(e.g)



(e.g) # Biggest among 3 no's.

```
a=int(input ('Enter a'))
b=int(input ('Enter b'))
c=int(input ('Enter c'))
if(a>b):
    if(a>c):
        Print ('a is big')
    else
        print ('c is big')
elif (b>c):
        Print ('b is big')
else:
    Print ('c is big')
```

Output

```
Enter a : 20
Enter b : 30
Enter c : 10
b is big
```

Iteration

A loop statement allows us to execute a statement or group of statement multiple times. Repeated execution of a set of statements is called iteration.

Python has two Iterative statement

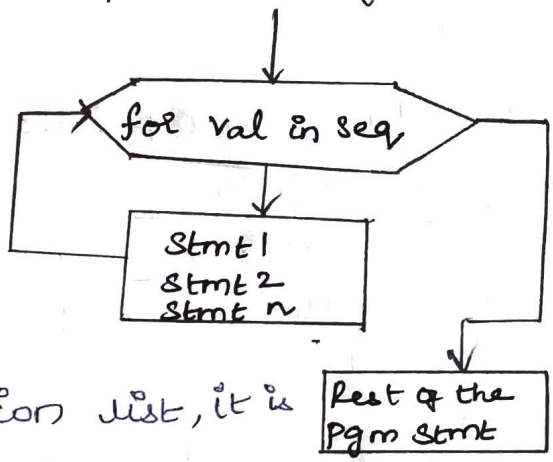
* for statement
* while statement

for statement:

The for statement in python goes through the list in any ordered sequence elements (i.e) list, tuples, the key of dictionary and any other iterables.

Syntax

```
for itervar in sequence:
    Statements.
```



(e.g);

```
for i in "example":
    Print(i)
```

If a sequence contains an expression list, it is evaluated first, then the first item in the sequence is assigned to the iterating variable. Next the statement block is executed, The statement block is executed until the entire sequence is exhausted.

(e.g)

```
for i in [1, 2, 3]:
    Print(i)
```

O/P

```
1
2
3
```

The range function:

The range() function is one of Python's builtin functions. It is used to indicate how many times the loop will be executed

```
range (start, upto, step)
range (start, end)
range (end)
```

(i) Start and step are optional

(ii) upto means end, but not including the end value.

(iii) Start, step, upto or end must be integer.

(e.g):
```
for i in range(5):
    print(i)
```
O/P
0
1
2
3
4

```
for i in range(1,4):
    Print(i)
```
O/P
1
2
3

```
for i in range(0,25,5):
    Print(i)
```
O/P
0
5
10
15
20

(e.g): write a python program to add first 10 numbers using for loop.

```
sum=0
for i in range(11):
    sum = sum+i
Print("sum", sum)
```
Output:
sum : 55

(e.g): write a python program to print even numbers

```
x=0
for x in range(0,11):
    if(x%2==0):
        Print(x)
    else:
        Print("No more even no's")
```
O/P
0
2
4
6
8
10

Nested for loop

The loop within the loop is called nested loop. In nested for loops two or more for statements are included in the body of the loop.

The number of iterations in this type of structure will be equal to the number of iterations in the outerloop multiplied by the number of iterations in the inner loop.

(e.g) TO print multiplication table

```
max=5
for i in range(1,max+1):
    for j in range(1,max+1):
        Print(i*j, "It", end=" ")
    Print() #print a newline
```
O/P
1 2 3 4 5
2 4 6 8 10
3 6 9 12 15
4 8 12 16 20
5 10 15 20 25

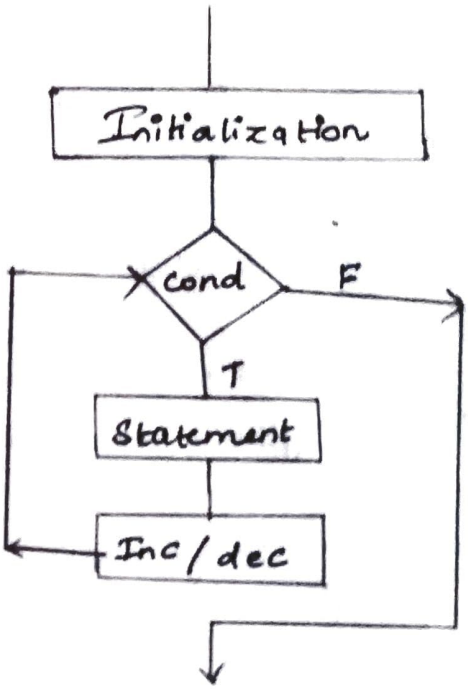
While loop

It is a repetitive control structure, used to execute the statements within the body, until the condition becomes false

The while loop is an entry controlled loop statement (i.e) the condition is evaluated first and if it is true, then the body of the loop is executed.

Syntax

```
Initialization
while (cond):
        body of the loop
```



Addition of first two numbers

```
i=1
sum=0
while (i<=10):
    sum=sum+i
    i=i+1
Print ("sum=", sum)
```

Nested while loop

When a while loop is present inside another while loop then it is called nested while loop.

(e.g)

```
i=1
j=5
while (i<4):
    while (j<8):
        Print (i," ",j)
        j=j+1
    i=i+1
```

o/p

1 , 5
2 , 6
3 , 7

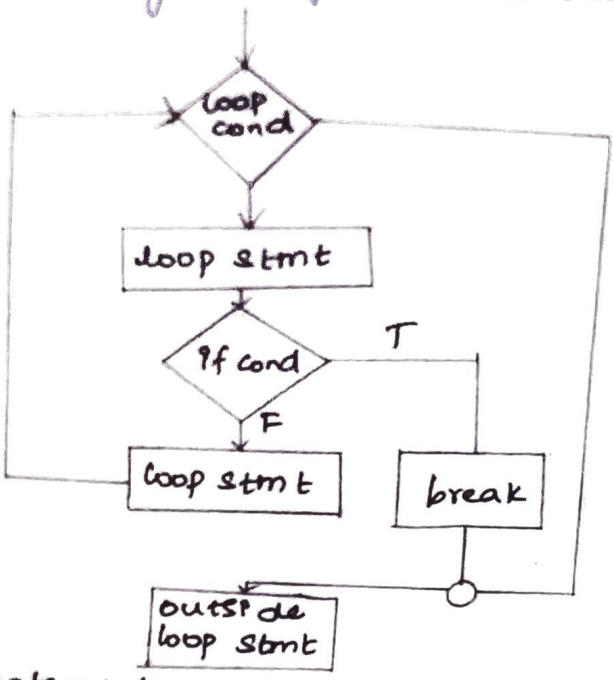
Loop Control Structures

Loop control statements change execution from its normal sequence. In python we have three loop control statements.

- break
- continue
- Pass

break statement

The break statement is used to terminate the execution of the nearest enclosing loop in which it appears. The break statement is usually associated with an if statement. When the keyword break is used inside any python loop, control is automatically transferred to the first statement after the loop.



(e.g)

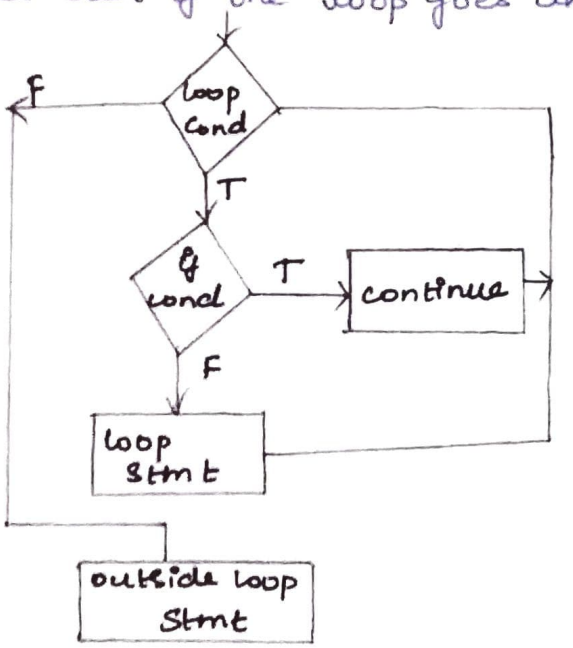
```
i=1
while (i<=10):
    Print(i, end=" ")
    if(i==5):
        break
    i=i+1
Print("loop is terminated")
```

o/p

1 2 3 4 5
loop is terminated.

Continue statement

The continue statement can only appear in the body of a loop. The continue statement results in the execution of the next iteration of the loop. Thus based on a condition a part of the loop is skipped and rest of the loop goes ahead for execution.



(e.g)

```
for i in range(8):
    if(i==5):
        Continue
    Print(i, end=" ")
```

o/p

0 1 2 3 4 6 7

Pass statement

The pass statement is used when a statement is required syntactically but no command or code has to be executed. It specified a null operation.

(e.g)
```
for i in range (1, 11):
    if (i == 5):
        pass
    Print (i, end = " ")
```

O/P

1 2 3 4 5 6 7 8 9 10

Fruitful functions

* A fruitful function is one in which there is a return statement with an expression. This means that a fruitful functions returns a value that can be utilized by the calling function for further processing.

(e.g)
```
def cube(x)
    return (x * x * x)

num=5
res = cube(num)
Print ("cube of ", num," = ", res)
```

O/P

cube of 5 = 125

The return statement must appear within the function. Once a value returns from a function, it immediately exits that function. Any code written after the return statement is never executed.

Parameters

Parameters are used in the function definition, arguments on the other hand are the values actually passed to a function when calling it. Therefore parameters define the number and type of arguments a function can accept.

(e.g)
```
def max(a,b): #max function accepts two parameters a and b
    if(a>b):
        Print("a is big",a)
    else:
        Print("b is big",b)
a=int(input('Enter a'))
b=int(input('Enter b'))
max(a,b) #a,b are function arguments
```

o/p
```
Enter a
25
Enter b
10
a is big 25
```

Local and Global scope

All variables in a program may not be accessible at all locations in that program. This depends on where we have declared a variable.

There are two basic scopes of variables in python

* Local Variable
* Global Variable

Local variable:

Variables that are defined inside a function body have a local scope. This means the local variables can be accessed only inside the function in which they are declared.

Global Variable:

Variables that are defined outside a function body have a global scope, where as global variables can be accessed throughout the program body by all functions.

(e.g)
```
def func():
    a=10 # local variable
    Print("value inside function:",a)
a=50 # global variable
func()
Print("value outside function:",a)
```

o/p
```
value inside function 10
value outside function 50
```

call by value

In call by value a copy of actual argument is passed to formal arguments. Any changes made to the formal arguments will not be visible outside the scope of the function.

```
def call by value (val):
    val = val + 5
    Print("Inside func val", val)
val = 95
call by value (val)
Print("Outside func aftercall val", val)
```

O/P

Inside func val = 100
Outside func aftercall val = 95

call by reference

In call by reference the location (address) of actual arguments is passed to formal arguments hence any changes made to formal arguments will also reflect in actual arguments.

```
def call by ref (val):
    val.append(5)
    Print("Inside func call", val)
Val = [1, 2, 3, 4]
Print("Before funcall val", val)
call by ref (val)
Print("After funcall val", val)
```

O/P

Before funcall val [1, 2, 3, 4]
Inside funcall [1, 2, 3, 4, 5]
After funcall val [1, 2, 3, 4, 5]

Function Composition in Python

Function composition combines two or more functions such that the result of each function is passed as the argument of the next function.

(e.g);

```
def sum(a, b):
    tot = a + b
    return (tot)
def avg(total):
    avg1 = total / 2
    return (avg1)
a = int(input('Enter a'))
b = int(input('Enter b'))
t = sum(a, b)
av = avg(t)
Print("Avg is : ", av)
```

O/P

Enter a : 20
Enter b : 10
Avg is : 15

Note:

Function composition is the ability to call one function within another function.

Recursion

A function calls itself is called Recursion. Every recursive solution has two major cases.

* **Base case** : In which the problem is simple enough to be solved directly without making any further calls to the same function
* **Recursive case** : In which first the problem is divided into simpler sub parts. Second, the function calls itself but with the sub parts of step 1, Third, the result is obtained by combining the solutions of simpler subparts.

(e.g) **Calculating factorial of a number**

TO calculate n! use the formula n*(n-1)!

Base case : n=1 the result is known to be 1 (i.e) 1! = 1

Recursive case : The factorial function will call itself but with a smaller value of n

(i.e) fact(n) = n * fact(n-1)

```
def fact(n):
    if(n==1):
        return 1
    else:
        return n*fact(n-1)
n=int(input("Enter the value of n:"))
print("Factorial of ",n," is ",fact(n))
```

Explanation

Problem 5!

5! = 5 x 4!

4! = 5 x 4 x 3!

3! = 5 x 4 x 3 x 2!

2! = 5 x 4 x 3 x 2 x 1!

1! = 1 ⇒ 5 x 4 x 3 x 2 x 1

5 x 4 x 3 x 2

5 x 4 x 6

5 x 24

120

o/p

```
Enter the value of n
5
Factorial of 5 is 120
```

(e.g) **Fibonacci Series - Recursion**

```
def fib(n):
    if(n<2):
        return 1
    return(fib(n-1)+fib(n-2))
n=int(input('Enter the no of terms'))
for i in range(n):
    print("Fibonacci series",i,fib(i))
```

o/p:

```
Enter the no of terms 4
fibonacci series 0 1
"  "  1 1
"  "  2 2
"  "  3 3
```

Strings

The Python string datatype is sequence of characters enclosed in single or double quotes.

(e.g) Str = "python"

Accessing strings

To access strings in python we use index in square brackets

(e.g) str[1] returns y

modifying strings in python

Strings in python are immutable. Once they have been created they cannot be modified.

String Concatenation

We concatenate two strings by using '+' operator. The plus operator takes the strings on the left side and clubs it with the string on the right side.

(e.g.,)
```
S1 = "Python"
S2 = "Programming"
Print ("S1+ " "+ S2)
```
O/P
Python Programming

The default data type of python is String

String slice

A substring of a string is called a slice. The slice operation is used to refer to subparts of sequence and strings. We can take subset of a string from the original string by using [] operator also known as slicing operator.

The Syntax of slice operation is s[start : end] where start specifies the beginning index of the substring and end specifies the end index of the substring.

(e.g) S = "programming"

Print (s[1:5]) ⇒ rogr #characters starting at index 1 and extending upto but not includes 5th index.

Print (s[:5]) ⇒ progr # defaults to start of the string.

Print [s[:]) ⇒ programming #defaults to the entire string

Print (s[1:]) ⇒ rogramming #defaults to the end of the string

Print (s[-1]) ⇒ g. # prints last character.

String functions and methods

Strings are an example of python objects. An object is an entity that contains both data as well as functions to manipulate that data.

Python supports many built in methods to manipulate strings. A method is just like a function.

Function	Usage	Example
Capitalize()	This function is used to Capitalize the first letter of the string.	Str = "welcome" Print (Str. Capitalize()) O/P Welcome
Count (str, beg, end)	counts no of times str occurs in a string	Str = "programming" msg = "program" print(msg.count(str,0, len(msg)) O/P 1
find (str, beg, end)	check if str is present in string.	msg = "python is easy to learn" print(msg. find ("is",0, len(msg))) O/P 7
isupper()	Returns True if string has atleast 1 character and every is an uppercase alphabet and False otherwise.	msg = "WELCOME" Print (msg.isupper()) O/P True
len(str)	Returns the length of the string.	Str = "WELCOME" print(len(str)) O/P 7
isdigit()	Returns True if string contains only digits and False otherwise	msg = "100" Print(msg.isdigit()) O/P True
isalnum()	Returns True if string has atleast 1 character and every character is either a number or an alphabet and false otherwise	msg = "python100") Print (msg.isalnum()) O/P True

Functions	Usage	Example.
ljust(width, fillchar)	Returns a string left justified to a total of width columns.	Str = "hello" Print (Str. ljust(10, '*')) O/P hello * * * * *
lower()	converts all characters in the string into lowercase	str = "WELCOME" Print (Str. lower()) O/P welcome
upper()	converts all characters in the string into uppercase	Str = "welcome' Print (str. upper()) O/P WELCOME
strip()	Removes all leading and trailing white space in string.	Str = "welcome " Print (Str. strip()) O/P welcome
title()	Returns string in title Case	Str = "Built_in String methods" print (Str. title()) O/P Built_in String Methods
min(str)	Returns the lowest alphabetical character from the string	str="hello' print (min(str)) O/P e
max(str)	Returns the highest alphabetical character from the string	Str = "hello" Print (max(str)) O/P d
split(delim)	Returns a list of substrings separated by the specified delimiter. If no delimiter is specified the whitespace is used to split strings	Str = "Big, small, abc, 123" Print (Str. split (',1)) O/P ['Big', 'small', 'abc', '123']

String Module

* String module consists of a number of useful constant classes and functions. These functions are used to manipulate string.

* Standard library of Python is extended as modules. To use these modules in a program, programmer needs to import the module

(e.g) import string

```
print(string.digits)      # 0123456789
print(string.octdigits)   # Refers to octal digits 0-7
print(string.uppercase)   # All the char are considered uppercase
print(string.punctuation) # ! " # $ % & ' ( ) * + , - . / : ; < = > ? @ [ \ ] ^ _ ` { | } ~
```

Escape Sequences in string

Escape sequence	Description
\n	newline
\\	Prints Backslash
\'	Prints single quote
\"	Prints Double quote
\t	Prints tabspace
\a	ASCII Bell

List as 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.

Syntax

arrayname = modulename.funname ('datatype', [elements])

(e.g.);

a = array.array('i', [1,2,3,4])

a → arrayname

array → modulename, functionname

i → integer datatype.

(e.g.);

```
import array
sum = 0
a = array.array('i', [1,2,3,4])
for i in a:
    sum = sum + i
print(sum)
```

o/p 10

Typecode	Description
'b'	Signed Integer of size 1 byte
'c'	Character of size 1 byte
'h'	Signed integer of size 2 bytes
'f'	floating point of size 4 bytes
'd'	floating point of size 8 bytes
'i'	signed integer of size 2 bytes
'w'	unicode character of size 4 bytes
'L'	unsigned integer of size 4 bytes
'I'	unsigned integer of size 2 bytes
'u'	unicode character of size 2 bytes

Array Methods.

1. array (datatype, valuelist)

This function is used to create an array with datatype and value list specified in its arguments

(e.g) array ('i', [5,6,12,40])

2. append()

This method is used to add the element at the end of the array

(e.g) a.append (20)

[5,6,12,40,20]

3. insert (index, element)

This method is used to add the value at the position specified in its argument

(e.g) a. insert (2, 15)

[5,6,15,12,40,20]

4. pop(index)

This function removes the element at the position mentioned in its argument and returns it.

(eg) a. pop(1)

[5,15,40,20]

5. index(element)

This function returns the index of value.

(e.g) a.index (40)

2

6. reverse()

This function reverses the array

a.reverse ()

[20,40,15,5]

7. count()

This is used to count number of elements in an array.

Illustrative Programs.

① Python Program to find the gcd of two numbers.

```
def hcf(a,b):
    if(b==0):
        return a
    else:
        return hcf(b, a%b)
a=int(input("enter the number"))
b=int(input("enter the number"))
Print("The gcd of a & b is :",hcf(a,b))
```

Output:

```
Enter the number : 60
Enter the number: 12
The gcd of a & b is : 12
```

② Python Program to compute the Power of a Number (or) exponentiation

```
n=int(input("Enter the number"))
exp = int(input("Enter the exponent"))
result =1
for i in range(1,exp+1):
    result = result * n
Print(result)
```

Output:

```
Enter the number : 4
Enter the exponent : 3
64
```

③ Python program to find the Square root of a number.

```
import math
n = int(input("Enter a number"))
S = math.sqrt(n)
Print("Square root is :",S)
```

Output:

```
Enter a number 25
Square root is 5
```

④ Python program to find the Sum of n array.

```
import array
sum=0
a=array.array('i',[1,10,20])
for i in a:
    sum=sum+i
Print(sum)
```

output.

```
31
```

⑤ Linear search

```
n=int(input("Enter the Size of list"))
lst =[]
flg = false
Print("Enter the elements")
for i in range(0,n):
    lst.append(int(input()))
Print(lst)
X= int(input("Enter the element to
                search"))
for i in range(0,n):
    if (x == lst[i]):
        flag = True
        break
if (flag == True):
    Print("The element is found")
else:
    Print("The element is not found")
```

Output:

Enter the size of list : 5

Enter the elements 2, 3 4 6 10

Enter the element to search 6

The element is found.

⑥ Binary Search

```
n= int(input("Enter the size of list"))
lst =[]
flag = False
Print("Enter the elements")
for i in range(0,n):
    lst.append(int(input())
Print(lst)
x= int(input("Enter the elements
                to search"))
low = 0
high = n-1
mid = (low+high)//2
while (low <= high):
    if (lst[mid] == x):
        print("Element is found")
        break
    elif (lst[mid] < x):
        low = mid+1
    else:
        high = mid-1
    mid = (low+high)//2
if(low > high):
    Print("Element is not found")
```

Output:

Enter the size of list : 5

Enter the elements 3 8 10 12 14

Enter the element to search 8

The element is found.

2 marks

① what are the types of control structure available in python?

(i) If statement: An if statement consists of a boolean expression followed by one or more statements.

(ii) If Else statement: An if statement can be followed by an optional Else statement, which executes when the boolean expression is FALSE

(iii) nested if statements: It can use one if or Else if statement inside another if (or) else if statement(s).

② what are the looping statements available in python?

(i) While loop: Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.

(ii) for loop: Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.

(iii) nested loop: It can use one or more loop inside any another while, for or do... while loop.

③ what is the difference between break and continue statement?

BREAK	CONTINUE
* To stop the current iteration and get out of that block	* To stop the current iteration and continue for next iteration
* Break : keyword is used	* continue : keyword is used
* It is used in switch case, for, while, do	* It is used in for, while
* statements after the break statement will not be executed	* Statement after continue statement will not be executed in current iteration

4) What is a global variable?

* A global variable is a variable declared in the main body of the source code, outside all functions.

* It will be visible throughout the program scope of this variable is available in all the functions.

* life as long as the programs execution doesn't come to an end

Example: This function uses global variable a

```
def fun(a):
    Print(a)
a=20          // Global scope.
fun(a)
```

5) what is fruitful function?

Fruitful functions are functions that return value while using fruitful functions, the return value must be handled properly by assigning it to a variable or use it as part of expression.

Example:

```
import math
X= math.sin(90)+1
Print (x)
```

output:

1.893996636

6) Define Recursion

* Recursion is a way of programming in which a function calls itself again and again until a condition is true.

* A recursive function calls itself and has a termination condition.

Example: def show():

show() return

7) Define Module? What are the ways to import modules in python?

Module is a python file that contains collection of related variables, functions, classes and other declarations.

Three different ways to import module:

1) using import 2) Importing individual objects 3) - import-(*)

8) What is a string?

* A string is a sequence of zero (or) more characters. An empty string contains no characters and has a length 0.

* The indices of a string's characters are numbered from 0 from the left end and numbered from -1 from the right end.

Example:

```
S = "All the best!"                    O/P
Print(s)                               All the best!
Print (s[0])                           A
print (s[2:5])                         l t
Print ((s*2)                           All the best! All the best!
```

9) What is string slices?

* A part of a string is called slice. The operator [m:n] returns the part of the string from mth index to nth index, including the character at mth index but excluding the character at nth index.

* If the first index is omitted, the slice starts at the beginning of the string.

* If the second index is omitted, the slice goes to the end of the string.

* If the first index is greater than or equals to the second, the slice is an empty string.

10) Why we call python string as immutable?

Python string is called as immutable, since its character can be accessed but the string cannot be modified.

Example:-

```
S = 'hello, python!'
S[0] = H
TypeError : 'str' Object does not support item
            assignment.
```

11) List any four methods of python strings?

* capitalize()
* isupper()
* islower()
* swapcase()

12) What is the use of string module in python?

The string module of python is a file that offers additional functions, classes and variable to manipulate standard strings.

13) How will you check in a string that all characters are alphanumeric?

isalnum() can be used which returns true if string has atleast 1 character and all other characters are alpha numeric

14) what is an array? (or) Define array.

* An array is a collection of same datatype elements. All elements are stored in continuous locations.

* Array index always start from '0'. It is represented using [] - square brackets.

Types of Array:

→ One dimensional array
→ Two dimensional array
→ Multi dimensional array.

15) Define List.

A list is a sequence of any type of values and can be created as a set of comma-separated values within square brackets. The values in a list are called "elements" or "items".

(E.g) list1 = ["ram", 'chennai', 2017] // list of diff types of items.

UNIT-IV LISTS, TUPLES, DICTIONARIES

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

Sequence :

A data structure is a group of data elements that are put together under one name. Data structure defines a particular way of storing and organizing data in a Computer. Sequence is the most basic data structure in python. In the Sequence data structure each element has a specific index. The index values starts from zero and is automatically incremented for the next element in the sequence.

List :

List is a sequence of values. The values in a list are comma-separated values, and it is also called as elements. List elements must enclosed by square brackets [....] and the values must b.

Syntax list Variable = [val1, val2 .... val n]

The key feature of a list is that it can have elements that belong to different data types or same data types.

Eg
```
>>> lst1 = [2, 4, b, 8]
>>> lst2 = [20, 14.68, 'hello', True, "python"]
```

A list that contains no elements is called an empty list.

eg >>> lst = [ ]

A list within another list is called Nested list.

Eg. >>> nlst = [1,5,8, [23, 28, 'abc'], 4]

To access the elements in the list:

To access values in list square brackets are used to slice along with the index to get value stored at the index, and also slice operation is used to access values in list.

Syntax : seq = lst [start : stop : step]

Eg:

```
lst = [5, 10, 15, 20, 25, 30]
print ("list elements ", lst)
print ("First element", lst[0])
print ("lst [1:3]:", lst[1:3])
print ("lst [::2]:", lst[::2]
```

output

```
list elements [5, 10, 15, 20, 25, 30]
First element 5
lst [1:3]: [10, 15]
lst [::2]: [5, 15, 25]
```

Updating Values in list:

Lists are mutable. The value of any element inside the list can be changed at any point of time. The elements of the list are accessible with their index value. The index always starts with 0 and ends with n-1

```
>>> lst = ['a', 'b', 'c', 10, 20, 30, 40]
>>> lst [3] = 75
>>> lst
['a', 'b', 'c', 75, 20, 30, 40].
```

Traversing a list:

Traversing a list means accessing all the elements in the list. Traversing can be performed by using for loop.

```
>>> lst = [98, 66, 73, 58, 48, 5]
>>> for i in lst:
        print (i)
```

o/p

```
98
66
73
58
48
5
```

```
>>> nlst = [1, 2, 3, [3, 4, 5], b]
>>> lst[3]
    [3, 4, 5]
>>> for i in nlst:
        print (i)
```

o/p

```
1
2
3
[3, 4, 5]
b.
```

List operations :

Operation	Description	Example	output
len	Returns length of list	>>> len([1, 2, 3])	3

Concatenation	Joins two lists. This is done by '+' operator	>>> a = [10, 5, b]
		>>> b = [2, 'a', 'x']
		>>> a+b
			[10, 5, b, 2, 'a', 'x']

Repetition	Repeats elements in the list. It is performed by '* operator.	>>> lst = [10, 'a'] >>> lst * 2	[10, 'a', 10, 'a']
in	checks if the value is present in the list	>>> 'a' in [10, 20, 30]	False
not in	checks if the value is not present in the list.	>>> 'a' not in [10, 20, 30]	True
max	Returns maximum value in the list	>>> lst = [72, 68, 45] >>> max (lst)	72
min	Returns minimum value in the list	>>> lst = [10, 26, 48] >>> min (lst)	10.
sum	Add the values in the list, that has numbers	>>> lst = [10, 20, 30] >>> sum (lst)	60

sorted	Returns a new sorted list. the original list is not sorted.	>>> lst = [5, 2, 4, 7, 1]
		>>> sorted (lst)
		>>> lst
			[1, 2, 4, 5, 7]
			[5, 2, 4, 7, 1]

all	Returns True if all items in iterable are true, otherwise it returns false	>>> lst = [1, 2, 3]
		>>> all (lst)
		>>> lst = [0, 1, 2]
		>>> all (lst)
			True
			False

any	Returns True if any element in the list is true. If the list is empty it returns False.	>>> lst=[0,1,2]
		>>> any(lst)
		>>> lst =[]
		>>> any(lst)
			True
			
			False.

List methods.

Python has various methods to help programmers work efficiently with lists.

Operation	Description and syntax	Example	Output.
count()	Counts the number of times an element appears in the list. Syntax : list.count(obj)	>>> lst=[5,6,7,6,7,6,8] >>> lst.count(6)	3

append()	Appends an element to the list (ie) add an element at the end of the list.
	Syntax: list.append(obj)
		>>> lst = [1,5,2]
		>>> lst.append(8)
		>>> lst
			[1,5,2,8]

index()	Returns the lowest index of object in the list. Gives a Value Error if object is not present in the list. Syntax: list.index(obj)	>>> lst =[5,26,'a',7] >>> lst.index('a')	2.

insert()	Insert object at the specified index in the list.
	Syntax: list.insert(index, obj)
		>>> lst =['a',5,8]
		>>> lst.insert(1,'b')
		>>> lst
			['a','b',5,8]

remove()	Removes or deletes objects from the list.
	Syntax: list.remove(obj)
		>>> lst =[5,6,23,48]
		>>> lst.remove(23)
		>>> lst
			[5,6,48]

pop()	Removes the element at the specified index from the list. Index is an optional parameter. If no index is specified then remove the last element from the list.
	Syntax: list.pop([index])
		>>> lst = [10,20,30,5,6]
		>>> lst.pop()
		>>> lst.pop(2)
		>>> lst
			6
			30
			[10,20,5]

reverse()	Reverse the elements in the list.
	Syntax:- list.reverse()
		>>> lst = [15,17,28,38]
		>>> lst.reverse()
		>>> lst
			[38,28,17,15]

sort()	Sorts the elements in the list
	Syntax : list.sort()
		>>> lst = [12,6,4,15,1]
		>>> lst.sort()
		>>> lst
			[1,4,6,12,15]

extend()	Adds the elements in a list to the end of another list using + (or) += on a list is similar to using extend()
	Syntax : list1.extend(list2)
		>>> lst = ['a','b',1]
		>>> lst2 = [10,20,30]
		>>> lst1.extend(list2)
		>>> lst1
		>>> lst2
			['a','b',1,10,20,30]
			[10,20,30]

Looping in lists:

Python's for and in constructs are useful when working with lists. The for var in list statement is an easy way to access each element in a list.

Eg Program to find the sum of elements in a list

```
lst = [10, 5, 20, 40]
sum = 0
for i in lst :
    sum = sum + i
print ('sum = ', sum)
```

output:

15

Using range() function

```
lst = [10, 5, 20, 40]
s = 0
for i in range (len(lst)):
    print ('index', i)
```

O/P

```
Index 0
Index 1
Index 2
Index 3
```

Mutability :

The list is a mutable data structure. This means the elements can be replaced, inserted or removed. A slice operator on the left side of an assignment operation can update single or multiple elements of a list.

```
>>> lst = [10, 'a', 'xyz', 75]
>>> lst [0] = 75
>>> lst [1:3] = [85, 'abc']
>>> print (lst)
```

Output

```
[75, 85, 'abc', 75]
```

List aliasing :

An object with more than one reference has more than one name, so we say that the object is aliased.

Eg

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

a → [1,2,3]
b → [1,2,3]

a refers to an object and assign b=a, then both Variable refer to the same object. The association of a variable with an object is called a reference.

Cloning List :

Assignment statements in python do not copy objects. They simply create bindings between two objects. for mutable Sequences a copy of an existing object may be required, so that one object can be changed without affecting another.

In lists, cloning operation can be used to create a copy of an existing list. so that changes made in one copy of list will not affect another. The copy contains the same elements as the original.

Method (1) list() function

```
newlist = list(oldlist)

lst = [1, 2, 3]
nlst = list(lst)
print('lst', lst)
print('nlst', nlst)
lst[1] = 70
print('lst', lst)
print('nlst', nlst)
```

Output:

```
lst [1, 2, 3]
nlst [1, 2, 3]
lst [1, 70, 3]
nlst [1, 2, 3]
```

Method (2) copy.copy() function

```
newlist = copy.copy(oldlist)

import copy
lst = [10, 20, 30, 40, 50]
nlst = copy.copy(lst)
lst[0] = 100
print('lst', lst)
print('nlst', nlst)
```

Output

```
lst [100, 20, 30]
nlst [10, 20, 30]
```

List parameters:

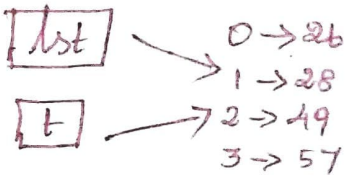
When we pass a list to a function, the function gets a reference to the list. If the function modifies a list parameter, the caller sees the change in the list.

Eg

```
def insval(t):
    t.insert(1,26)

lst = [26, 28, 49, 57]
print("Before insertion", lst)
insval(lst)
print("After insertion", lst)
```

The parameter t and the variable lst are aliases for the same object



Tuple

Tuple is a sequence of values of same or different types separated by commas. Tuple is a sequence of immutable objects. Tuples use parenthesis to define its elements. Once a tuple has been created, we can't add elements to a tuple or remove elements from the tuple.

Creating Tuples:

(i) Tuple with integer data items.

```
>>> t = (1,2,3)
>>> t
   (1,2,3)
```

(ii) Tuple with different datatypes

```
>>> t = ('hello', 3.14, True, 75, 98)
>>> t
   ('hello', 3.14, True, 75, 98)
```

(iii) Nested Tuple.

```
>>> t=(15, 18, 45, (78, 'abc', 37.98))
```

Creating a tuple with one element is some what different when we creating a tuple with one element, we need to add a comma after the element.

```
>>> tup = "program",      (or)  t=1,
```

Accessing Values in Tuples:

To access the values in a tuple, it is necessary to use the index number enclosed in square bracket along with the name of the tuple.

(i) using square brackets

```
>>> t= (10, 20, 30, 75, 95)
>>> print (t[3])
   75
```

(ii) using slicing.

```
>>> tup = (75, 85, 95, 100)
>>> tup[1:3)
   (85,95)
>>>  tup [ :2]
   (75,85)
```

Basic Tuple operations :

Operation	Description	Example	Output

Concatenation	Adding tuple elements at the end of another tuple elements	>>> a = (50,60)
		>>> b = ('25', 'xyz')
		>>> a+b
			(50,60,25, 'xyz')

Repetition	Repeating the tuple in n no. of times.	>>> a = (50, 60) >>> a * 2	(50,60, 50,60)

Membership in, not in	Returns True, if element is present in tuple, otherwise return false.	>>> a=(51, 63,98)
		>>> 63 in a
		>>> 98 not in a
			True
			False

Comparison	Returns true, if all elements in both tuple are same, otherwise returns false.	>>> a=1,2,3
		>>> b= 1,2,3
		>>> a==b
			True.

min	Returns minimum value in tuple.	>>> a= 5,9,10,12 >>> min(a)	5
max	Returns maximum value in tuple.	>>> a= 5,9,10,12 >>> max(a)	12
tuple	Returns (or) convert to tuple.	>>> tuple([10,20,30]) >>> tuple('hello')	(10, 20,30) ('h','e','l','l', 'o')
del	Delete the entire tuple.	>>> a=5,6,7 >>> del(a)	

Basic Tuple methods

index()	Returns the index of the first matched item. syntax: t.index(obj)	>>>t=(72,45,26) >>> t.index [45]	1
count()	Returns the count of the given element. syntax: t.count(obj)	>>>t=(5,6,5,4,5) >>> t.count(5)	3.

Tuple assignment :

Tuple assignment is a very attractive and powerful feature in python. It allows the assignment of values to a tuple of variables on the left side of the assignment from the tuple of values on the right side of the assignment.

The number of variables in the tuple on the left of the assignment must match the number of elements in the tuple on the right of the assignment.

Eg:
```
>>> (a,b,c) = (25,35,45)
>>> print (a,b,c)
    25 35 45
```

```
>>> tup = (10,20,30)
>>> x,y,z = tup
>>> print (x,y,z)
    10,20,30
```

Eg swap two numbers by tuple assignment.

```
>>> a,b = 70,65
>>> a,b = b,a
>>> print ('a:', a, 'b:', b)
    a:65 b:70
```

Tuples as return values

Tuples can also be returned by the function as return values generally the function returns anyone value, but by returning tuple a function can return more than one value.

Eg
```
def divfun (a,b):
    quo = a//b
    rem = a%b
    return (quo, rem)

a = int (input ())
b = int (input ())
c = divfun (a,b)
print (c)
```

Output
```
26
4
(6,2)
```

Variable length argument tuples

Variable number of argument can also be passed to a function. A variable name is proceded by an (*) asterisk collects the arguments into a tuple.

Eg
```
def sumfun (*t):
    i = 0
    sum = 0
    while i < len (t):
        sum = sum + t[i]
        i = i+1
    return sum
a = sumfun (10,20,30,40,50)
print (a)
print (sumfun (1,2,3))
```

output
```
150
6
```

Advantages of tuples

* Tuples are immutable
* Iterating through tuple is faster than iterating over a list
* Tuples can be used as key for a dictionary, but list cannot be used as keys
* Tuples are suited for storing data that is write protected
* Multiple values can be returned using a tuple.

Dictionaries

Dictionary is an unordered collection of elements. An element in dictionary has a key value pair. All elements in dictionary are placed inside the curly braces {...}

Elements in dictionary are accessed via keys and not by their position.

The values of a dictionary can be any data type. keys must be immutable data type (numbers, strings, tuple).

Operations on dictionary :

* Accessing an element
* update
* Add element
* Membership.

Operations	Description	Example.

Creating a dictionary	creating a dictionary with elements of different datatypes	>>> a = {'Name': 'AAA', 'No': 123}
		>>> a
		{'Name': 'AAA', 'No': 123}

Accessing an element	Accessing the elements by using keys.	>>> a['No']
		123
		>>> a[0]
		KeyError : 0

update	Assigning a new value to key. It replaces the old value by new value.	>>> a['No'] = 678
		>>> a
		{'Name': 'AAA', 'No': 678}

add element	Add new element into the dictionary with key	>>> a['Age'] = 18
		>>> a
		{'Name': 'AAA', 'No': 678, 'age': 18}

membership	Returns True, if the key is present in dictionary. otherwise returns false.	>>> a = {'Name': 'AAA', 'No': 678}
		>>> 'Name' in a
		True
		>>> 'Age' not in a
		True

Methods in dictionary

Method	Example	Description.

copy()	>>> a = {'sub' : 'python', 'Marks' : 95}
	>>> b = a.copy()
	>>> print(b)
	{'sub' : 'python', 'marks' : 95}
		It returns copy of dictionary. copy of dictionary is stored in dictionary b.

items()	>>> a = {1 : "one", 2 : "two"}
	>>> a.items()
	dict_items([(1, 'one', 2, 'two')])
		Returns a new view of the dictionary items. It displays a list of dictionary (key, value) tuple pairs

keys()	>>> a.keys() dict_keys([1, 2])	It displays list of keys in a dictionary.
values()	>>> a.values() dict_values(['one', 'two'])	It displays list of values in dictionary.

pop(key)	>>> a.pop(1)
	'two'
	>>> print(a)
	output
	{2 : 'two'}
		Remove the element with key and return its value from the dictionary.

setdefault (key, value)	>>> a = {'sub' : 'python', 'Marks' : 95}
	>>> a.setdefault('Dept', 'Mech')
	>>> print(a)
	{'sub' : 'python', 'Marks' : 95, 'Dept' : 'Mech'}
		If key is in the dictionary. returns its value, If key is not present, insert key with a value of dictionary and return dictionary.

update(dict)	>>> a = {1 : 'one'}
	>>> b = {2 : 'two'}
	>>> a.update(b)
	>>> print(a)
	{1 : 'one', 2 : 'two'}
		It will add the dictionary with the existing dictionary.

fromkeys()	>>> key = {'Apple', 'Mango'}
	>>> value = 'fruits'
	>>> d = dict.fromkeys(key, value)
	>>> d
	{'Apple' : 'fruits', 'Mango' : 'fruits'}
		It creates a dictionary from keys and values.

clear()	>>> a = {'Name' : 'AAA', 'Age' : 18}
	>>> a.clear()
	>>> print(a)
	{}
		Remove all elements from the dictionary.

len()	>>> len(d) 2	It returns the length of dictionary.
del()	>>> a = {'Name': 'AAA', 'Age': 18} >>> del(a)	It will delete the entire dictionary

List Comprehension

Python supports computed lists called list comprehension. It is a tool for transforming one list into another list.

Syntax:

lst = [expression for element in list if conditional]

this is equivalent to

```
for element in list:
    if cond:
        exp
```

It consists of brackets containing an expression, followed by a for clause, then zero or more for or if clauses. The list comprehension always returns a result list.

Program to combine three lines of code into one.

```
>>> cubes = [i**3 for i in range(6)]
>>> print(cubes)
    [0, 1, 8, 27, 64, 125]
```

We can also use the list comprehension to combine the elements of two lists.

```
>>> lst = [x*y for x in [20, 40, 60] for y in [2, 4, 6]]
>>> lst
    [40, 80, 120, 80, 160, 240, 160, 320, 480]
```

the above code is equivalent to

```
lst = []
for x in [20, 40, 80]:
    for y in [2, 4, 6]:
        lst.append (x * y)
print (lst)
```

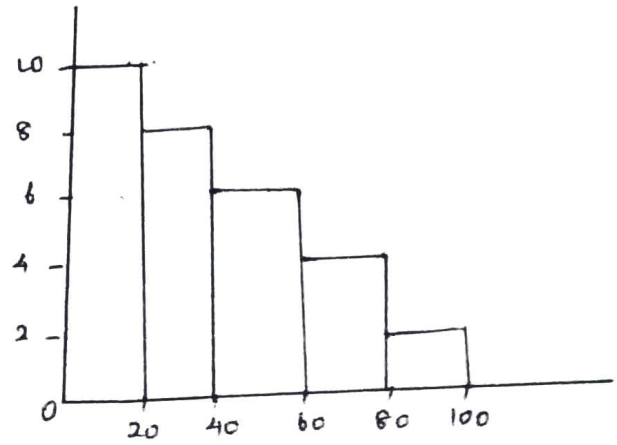
Illustrative problems

Histograms

A histogram shows the frequency on the vertical axis and the horizontal axis in another dimension.

Matplotlib can be used to create histograms. usually it has bins, where every bin has a minimum and maximum value. Each bin also has a frequency between x and infinite

(eg)
```
import numpy as np
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt
x = [21, 22, 23, 35, 6, 7, 8, 45, 65, 78, 100]
num_bins = 5
n, bins, patches = plt.hist (x, num_bins, facecolor = 'blue',
                                          alpha = 0.5)
plt.show ()
```



Illustrative programs

Selection sort

```
n = int (input ("Enter the size of list"))
lst = []
print ("Enter the elements")
for i in range (0, n):
    lst.append (int (input ()))
for i in range (0, n-1):
    for j in range (i+1, n):
        if lst[i] > lst[j]:
            temp = lst[i]
            lst[i] = lst[j]
            lst[j] = temp
print ("The sorted values are")
print (lst)
```

output

```
Enter the size of list . 5
Enter the elements
    53 23 12 19 8
The sorted values are
    [8, 12, 19, 23, 53].
```

Example: [53, 23, 12, 19, 8]

pass 1 : i=0.

j=1, 53>23 (T) → Exchange the values
23, 53, 12, 19, 8.

J=2, 23>12 (T) → Exchange the values
12, 53, 23, 19, 8.

j=3, 12>19 (F) → No changes.

j=4, 12>8 (T) → Exchange.
8, 53, 23, 19, 12

pass II i = 1
8 53, 23, 19, 12

j=2 53>23 (T) → change.
8, 23, 53, 19, 12

j=3
8 23, 53, 19, 12

23>19 (T) Exchange

j=4, 19>12 (T) → Exchange
8, 12, 53, 23, 19.

Pass III i=2:
8 12 53, 23, 19

J=3, 53>23 (T) → Exchange
8. 12 19, 53, 23

pass IV i=3
8, 12, 19 53, 23

53>23 (T) → Exchange the values.
8 12 19, 23, 53.

Insertion sort

```
n=int (input ("Enter the size of list"))
lst =[]
print ("Enter the elements")
for i in range (0,n):
    lst.apperd (int (input ()))
for i in range (1,n):
    j = i
    while j>0 and lst [j-1] > lst [j]:
        temp = lst [j]
        lst [j] = lst [j-1]
        lst [j-1] = temp
        j = j-1
Print ("Sorted list")
print (lst)
```

output

```
Enter the size of list : 5
Enter the elements : 27 14 36 24 8
Sorted list
    [8, 14, 24, 27, 36]
```

Example:

27 14 36 24 8

J=1

27 14 36 24 8

27>14 (T) → Exchange

14 27 36 24 8

j=2, 14 27 36 24 8

27>36 (F) No change

14 27 36 24 8

j=3, 14, 27, 36, 24, 8

36>24 (T) → change

14 27 24, 36, 8

J=2

27>24 (T) Exchange

14 24 27 36 8 → condition fails.

j=4. 14 24 27 36 8

36>8 (T) Exchange

14 24 27 8 36

j=3. 27>8 (T) Exchange.

14 24 8 27 36

j=2 24>8 (T) Exchange.

14 8 24. 27 36

j=1 ; 14>8 (T) - Exchange.

8 14 24 27 36

Product details [Retail bill preparation]

```
def bill (lst, n):
    total = 0
    for x in lst:
    price = unitprice [x]
    if prod [x] >0:
        total = total + price * 2
        prod [x] = prod [x] - n
    Print ("purchased product details")
    Print (lst)
    print ("Total amount :", total)
lst = ["biscuit", "chocolate", "pen",
       "pencil", "Erasei"]

n = 2
print ("product price details")
unitprice = {"biscuit": 25, "chocolate"
    : 10, "pen": 48, "pencil": 8
        "Erasei" : 4}
print (unitprice)
print ("product stock details")
prod = {"biscuit": 10, "chocolate": 80,
    "pen": 28, "pencil": 52,
        "Erasei": 5}
print (prod)
bill (lst, n)
print ("current stock details")
print (prod)
```

output:

```
Product price details.
{'biscuit': 25, 'chocolate': 10, 'pen': 48,
    'pencil': 8, 'Erasei': 4}
Product stock details.
{'biscuit': 10, 'chocolate': 80, 'pen': 28,
    'pencil': 52, 'Erasei': 5}
Purchased product details.
['biscuit', 'chocolate', 'pen', 'pencil',
    'Erasei']
Total amount: 190
current stock details.
{'biscuit': 8, 'chocolate': 78, 'pen': 26,
    'pencil': 50, 'Erasei': 3}
```

Students marks statement:

```
def average(numbers):
    total = sum(numbers)
    total = float(total)
    avg = total / (len(numbers))
    return avg
def get_average(student):
    Assessment = average(student[
        "Assessment"])
    Assignment = average(student
        ["Assignment"])
    Attendance = average(student
        ["Attendance"])
    tot = 0.6 * Assessment + 0.3 *
        Assignment + 0.1 * Attendance
    return tot
def get_letter_grade(score):
    if score >= 91:
        return "O"
    elif score >= 81:
        return "A+"
    elif score >= 71:
        return "A"
    elif score >= 61:
        return "B+"
    elif score >= 50:
        return "B"
    else:
        return "U"
def get_class_average(students):
    results = []
    for student in students:
        r = get_average(student)
    results.append(r)
    return average(results)
n1 = {"name": "kumar", "Assessment":
    [90.0, 97.0, 85.0, 92.0],
    "Assignment": [88.0, 70.0, 94.0],
    "Attendance": [85.0, 90.0]}
n2 = {"name": "veena", "Assessment":
    [100.0, 92.0, 98.0, 100.0],
    "Assignment": [82.0, 83.0, 91.0],
    "Attendance": [89.0, 97.0]}
n3 = {"name": "Deepak", "Assessment":
    [80.0, 87.0, 75.0, 50.0], "Assignment":
    [10.0, 75.0, 78,0],
    "Attendance": [100.0, 100.0]}
Students = [n1, n2, n3]
for student in students:
    print(student["name"])
    print(student["Assessment"])
    print(student["Assignment"])
    print(student["Attendance"])
print("Average marks")
print("kumar:", get_letter_grade
    (get_average(n1)))
print("veena:", get_letter_grade
    (get_average(n2)))
print("deepak:", get_letter_grade
    (get_average(n3)))
print("class Average Marks")
print(get_class_average(students))
print("class Average Grade")
print(get_letter_grade(get_class_
    average(students)))
```

Output

```
Kumar
[90.0, 97.0, 85.0, 92.0]
[88.0, 70.0, 94.0]
[85.0, 90.0]
Veena
[100.0, 92.0, 98.0, 100.0]
[82.0, 83.0, 91.0]
[89.0, 97.0]
Deepak.
[80.0, 87.0, 75.0, 50.0]
[70.0, 75.0, 78.0]
[100.0] [100,0]
Average marks
Kumar: A+
Veena = O
Deepak : B+
Class Average Marks 70.0
Class Average grade : B+
```

Two marks

1. Define List slice with example ?

A part of a list is called List slice. The operation [m:n] returns the part of the list from mth index to nth index, including the element at the mth index but excluding the element at no index.

eg
```
>>> lst = ['a','b','c','i',2,3]
>>> lst [:3]
['a','b','c']
```

---

2. what do you mean by list cloning with example ?

If we want to modify a list and also keep a copy of the original, we need to be able to make a copy of the list itself, not just the reference. This process is called as list cloning.

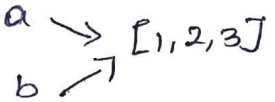
(eg)
```
>>> a = [1,2,3,4]
>>> b = a[1:3]
[2,3]
```

---

3. Define list aliasing with example.

If we assign one variable to another, both variable refers to the same object.

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



The True list has two different names a and b, it is called aliased. changes made with one alias affect the another.

```
>>> b[0] = 20
>>> a
[20,2,3]
```

---

4. Define nested list with example.

A nested list is a list that appears as an element in another list.

```
>>> nest = ['abc', 1, 2, [10,20]]
```

5. Illustrate negative indexing in list with an example?

Python supports negative indexing for sequence types like lists. The last item on the list takes the -1 index, the second to the last item has the -2 index and so on.

```
>>> lst = ['p', 'y', 't', 'h', 'o', 'n']
>>> lst[-1]
n
```

6. How can we delete an element in the list:

The keyword del can be used to delete one or more items on a list or the entire list itself.

eg) `del lst[index]`
`del lst[:]`
`del lst`

```
>>> lst = [1,2,3,4]
>>> del lst[2]
[1,2,4]
```

7. How to convert a string to a tuple?

A string can be converted as a tuple by using the tuple() function.

```
(eg) >>> tuple ('programming')
('p','r','o','g','r','a','m','m','i','n','g')
```

8. Write a python program to swap two values by using tuple assignment.

```
a = 25
b = 35
a,b = b,a
print (a,b)
```

35, 25

9. Point out the methods used in tuples?

count() → It returns the number of elements which is equal to the given element.

```
>>> tup = ('a', 'b', 'c', 'a')
>>> tup.count('a')
2
```

index()

↓,

It returns the index of the given element

```
Eg >>> tup = ('a','b','c')
   >>> tup.index(b)
   1
```

10. Explain how tuples are used as return values.

Functions can return tuples as return values.

(eg)
```
def swap (x,y):
    return (y,x)
a = 35
b = 26
a,b = swap (a,b)
print ('a=', a, "b=", b
```

O/P
a=26
b=35

11. Define coercion with Example (or) Data type Conversion.

Data type conversion or type casting changes one type of value/ Variable to another data type. It is also known as coercion.

(eg) >>> float (200)
200.0.

>>> int (350.75)
350.

12. Write a python program to generate a dictionary that contains numbers in the form of (x, x*x)

```
n = int (input ('Enter a number :'))
d = {x : x*x for x in range (1, n+1)}
print (d)
```

O/P
Enter a number : 5
{1:1, 2:4, 3:9, 4:16, 5:25}

13. Define lookup and reverse lookup.

Lookup is the process of finding the corresponding value for the given key from dictionary. (eg) value = dict [key]

Reverse lookup is the process of finding the key for a given value.

14. Define memorization.

Memorization effectively refers to remembering results of method calls based on the method inputs and then returning the remembered result rather than computing the result again.

UNIT-V

Files, Modules, Packages

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

Files

Files are used to store data permanently. It holds a specific name for itself. Files are divided into two categories.

* Text file * Binary file

* Text file is a stream of characters that can be sequentially processed by a computer in forward direction.

* A Binary file is a file which may contain any type of data, encoded in binary form for computer storage and processing purpose. Binary File is usually called as a non text file. Also there is no end of line terminator in a binary file.

Characteristics of a text file

* A text file can be opened by the help of any text editor.
* A text files have an extension of .txt.
* The end of line in a textfile in python is represented by \n.
* It is easy to recover data stored in text format.
* It takes more storage space than required

File Operations :-

In python there are four basic file related operations. They are

* Opening a file
* Reading from a file
* Writing to a file
* closing a file

} Process data

* Open file * processed data * close file

Opening a file

Before reading from or writing to a file, we must open it by using python's built-in open() function. This function creates a file object, which will be used to invoke methods associated with it.

Syntax :

```
file obj = open (filename, [accessmode])
```

filename → It is a string value which contains the name of the file to access

accessmode → It determines the mode in which the file has to be opened.

accessmode is an optional parameter and the default file access mode is read (r).

Different modes for Opening a file

Modes	Description
r	This is the default mode of opening a file which opens the file for reading only. The file pointer is placed at the beginning of the file.
rb	This mode opens a file for reading only in binary format. The file pointer is placed at the beginning of the file.
r+	This mode opens a file for both reading and writing. The file pointer is placed at the beginning of the file.

Modes	Description.
rbt	This mode opens the file for both reading and writing in the binary format. The file pointer is placed at the beginning of the file
w	This mode opens the file for writing only. If the files already exists, it overwrites the files. If not, it creates a new file.
wb	Opens a file in binary format for writing only. If the file already exists, it overwrites the file. If not, it creates a new file.
w+	Opens a file for both reading and writing. If the file doesnot exist, a newfile is created for reading as well as writing. If the file already exists and has some data stored in it, the contents are overwritten.
wbt	Opens a file in binary format for both reading and writing.
a	Opens a file for appending. The file pointer is placed at the end of the file, if the file exists, otherwise it creates a newfile for writing.
ab	Opens a file in binary format for appending. The file pointer is placed at the end of the file, if the file exists, Otherwise it creates a newfile for writing.
a+	Opens a file for reading and appending. The file pointer is placed at the end of the file, if the file exists, Otherwise it creates a newfile for writing.
ab+	Opens a file in binary format for reading and appending. The file pointer is placed at the end of the file. If the file exists, Otherwise it creates a newfile for writing

(e.g):

```
>>> f = open('file1.txt', "r") #opens in read mode
    f1 = open('file2.txt', "w") #opens in write mode
    f2 = open('file3.txt', "a") # opens file in append mode
```

Reading and Writing files:

The read() and write() are used to read data from file and write data to files respectively.

write() method:

The write() method is used to write a string to an opened file, while writing data to a file by using write() method does not add a newline character ('\n') to the end of the string.

Syntax: fileobj.write(string)

(e.g)

```
f = open("file1.txt", "w")
f.write("python file methods")
f.close()
print("data written into the file")
```

o/p

data written into the file.

writelines() method:

This method is used to write a list of strings.

(e.g):

```
f = open("file1.txt", "w")
lines = ["file program", "writelines method"]
f.writelines(lines)
f.close()
```

read() method:

This method is used to read a string from an already opened file.

Syntax: fileobj.read([count])

Count is an optional parameter which is passed to the read() method specifies the number of bytes to be read from the opened file.

(e.g):
```
f.open("file1.txt","r")
Print(f.read(4))
f.close()
```
o/p:
```
pyth
```

readline() method

It is used to read a singleline from the file. This method returns an empty string when the end of the file has been reached.

(e.g):
```
f=open("file1.txt","r")
Print(f.readline())
f.close()
```
o/p
```
Python file methods.
```

readlines() method:

It is used to read all lines in the file.
```
f = open("file1.txt","r")
Print(f.readlines())
f.close()
```

File Methods

The object of file provides functions to manipulate data. python has several file methods that can be used to handle files.

flush() This method will force out any unsaved data that exists in a program buffer to the actual file

Syntax:
```
file object.flush()
```

(e.g):
```
f=open("newfile.txt","w")
f.write("Flushmethod")
f.flush()
f.close()
```

fileno()

This method returns the integer file descriptor which is used to request I/O operations from the OS.

Syntax:

```
fileobject.fileno()
```

(e.g.):

```
f=open("newfile.txt","w")
Print(f.fileno())
f.close()
```

O/P 2.

isatty()

Returns True, if the file stream is interactive and False otherwise.

(e.g)

```
f=open("newfile.txt","w")
f.write("hello")
Print(f.isatty())
```

O/P False

seek()

In opening a file, the system points to the beginning of the file. A seek() operation moves the pointer to some other part of the file, so that we can read or write at the place.

Syntax:-

```
fileobject.seek(offset[,from])
```

Offset: It is the position of file read/write pointer.

from: It is an optional parameter, that has three possible values.

0: absolute file positioning (default value)

1: It seeks relative to current pos.

2: It seeks relative to the end of the file.

readline() (e.g)

```
f=open("newfile.txt","r+")
Print(f.readline())
f.seek(0,0)
Print(f.readline())
f.close()
```

tell()

The tell() method returns the current position of the file

Syntax:

fileobject.tell()

(e.g)

```
f = open("newfile.txt", "w")
f.write("tell() method")
Print(f.tell())
```

O/P
14

truncate()

Resizes the file to n bytes

Syntax:

fileobject.truncate([size])

If size is not specified, it resizes to current location.

(e.g)

```
f= open("file.txt", "w")
f.write("Resizes the file")
f.truncate(5)
f= open("file1.txt","r")
Print(f.read())
```

O/P
Resize

Split()

It splits the line based on a character

Syntax:

var.split()

(e.g)

```
with open("file1.txt","r") as f:
    line = f.readline()
    words = line.split()
    Print(words)
```

O/P
['Python', 'File', 'Method']

next()

It is used in a file when it acts as an iterator in a loop. This method returns the next input line, or raised stop Iteration when EOF is hile syntax:- next(file obj)

(e.g)

```
f=open("newfile.txt", "rwt")
for i in range(5):
    line = next(f)
    print(i, line)
f.close()
```

Format Operators

One of python's features is the string format operator %. Python uses C-style string formatting to create new formatted strings.

The "%" operator is used to format a set of variables enclosed in a tuple together with a format, string, which contains normal text together with argument specifiers

Format Symbol	conversion
%c	character
%s	String conversion str() prior to formatting
%i	Signed decimal integer
%d	Signed decimal integer
%u	Unsigned decimal integer
%o	octal integer
%x or %X	hexadecimal integer
%e or %E	Exponential Notation
%f	Floating point real number
%g	The shorter of %f and %e
%G	The shorter of %f and %E

Syntax : <"format"> % " (<values>)

(e.g)

```
Subject = "Python"
Credit = 3
Print ("Subject = %s and credit = %d" % (Subject, Credit))
Print ("Subject = %s and credit = %d" % ("English", 4))
```

O/P

```
Subject = Python and credit = 3
Subject = English and credit = 4
```

Format() Function

Python format() function has been introduced for handling complex string formatting more efficiently. This method of the built in string class provides functionality for complex variable substitutions and value formatting.

Syntax : {}. format (value)

Parameters:

(value): can be integer, floating point numeric constant, string, characters or even variables.

Example:-

```
f = open("file.txt", "r+")
f.write('python {}'.format('is easy to learn'))
```

F strings in python

f-strings provide a concise and convenient way to embed python expressions inside string literals for formatting.

Example1 :-

```
a = "python"
f.write(f'strings in {a}')
```

o/p:

strings in python

Example 2:

```
f = open("file.txt", 'r+')
a,b = 30, 40
f.write(f "Equation : {2*(a+b)}")
```

o/p:

Equation : 140

Command Line Arguments

The python standard Library contains many useful Modules. Examples are: math module, sys module etc.

Syntax:
```
import module_name
```
Example:
```
import sys
```

The System Module - Sys Module

The system module contains much system-specific functionality. They are accessed by the module name followed by the period operator and the functionality name.

A few functions available in sys module

Functionality Name	Description
sys.argv	The arguments passed to the command line will be enclosed within sys.argv in the form of a list
sys.exit([arg])	This allows exit from python. The arg 0 leads to successful termination. The value 2 for command line syntax errors and 1 for all other kind of errors.
sys.getsizeof(object[, default])	Returns the size of objects in bytes if given. The default argument allows to define a value which will be returned if the object type does not provide means to retrieve the size and cause a TypeError
sys.version	A string containing the version number of the python Interpreter plus additional information on the build number and compiler used.

The command line arguments are processed as strings.

eg: import sys
```
Print ("The python Script is: %s" %sys.argv[0])
for i in range (len(sys.argv)):
    If i>0:
        print ("Arg %d : %s" % (i, sys.argv[i]))
```

o/p:

The python Script is:
Arg 1 : 1
Arg 2 : 2
Arg 3 : 3

IDLE does not support passing command line arguments to a python Script at run time. Move to DOS command prompt and execute the python script with command line arguments.

e.g
```
import sys
sum=0
print (sys.argv)
for i in range (len(sys.argv)):
    if i>0:
        sum= sum + int(sys.argv[i])
Print ("The sum of command line arguments is: ", sum)
```

o/p:

```
argv
argv1
argv2
argv3
argv4
```

Error and Exception Handling

Errors or mistakes in a program are often referred to as bugs. Debugging is the process of finding and eliminating errors.

Errors can be classified into three major groups,

* Syntax errors
* Logical errors
* Runtime errors

Syntax Errors

Syntax Error arises due to poor understanding of the language.

* leaving out a keyword
* putting a keyword in the wrong place
* misspelling a keyword
* Incorrect Indentation
* leaving out a symbol, such as colon, comma (or) brackets

(e.g)
```
>>>i = 0
>>> if i==0 print(i)
```
Syntax error : invalid syntax

Logical error:

Logical error may occur due to wrong algorithm (or) logic to solve a particular program.

(eg)
```
>>>x=6
>>>y=10
>>>z= x+y/2
>>> print("Avg of 2 no is ",z)
```
O/P

Avg of 2 no is 11

This answer is wrong because the divison is evaluated before addition.

To rectify this problem we will simply add the Parenthesis $z = (x+y)/2$. Now we will get the correct answer 8.

Runtime error:

Runtime error may arises even if a statement or expression is syntactically correct, it may cause an error when an attempt is made to execute it. Errors detected during execution is called runtime error

(eg) >>>20/0

```
Traceback (most recent call last):
File "<Std in>", line1, in <module>
Zero divison Error: divison by zero
```

Exceptions

An exception is an iregular state that is originated by a runtime error in the program. An exception is an event which occurs during the execution of a program and disrupt the normal flow of the programs instruction.

Built-in Exceptions

The built in exceptions force our program to output an error when something in it goes wrong

Exception	Description
Exception	Base class for all Exceptions.
Arithmatic Error	Generated due to mathematical calculations
Zero Divison Error	Raised when a number is divided by zero
Attribute Error	Raised when attribute reference (or) assignment fails
Import Error	Raised when an import statement fails
Index Error	Raised when an index is not found in a sequence
Key Error	Raised when a key is not found in the sequence
Syntax Error	Raised when there is a syntax error in the program
Indentation Error	Raised when there is an indentation Problem in the program.

Exception	Description
Name Error	Raised when a variable is not found in local or global space
Io Error	Raised when input (or) output operation fails
Floating Point Error	Raised when a float point calculation could not be performed.
EOF Error	Raised when end-of-file is reached.

Handling Exceptions

We can handle exceptions in our program by using try block and except block. A critical operation which can raise exception is placed inside the try block and the code that handles exception is written in except block.

Syntax:

```
try:
    Statements
except ExceptionName:
    Statements
```

1) try.... except

Python provides a try statement for handling exceptions, first the try block is executed. If no exception occurs, the except block is skipped, otherwise the exception type matches the exception named after the except keyword, the execution continues after the try statement.

If no exception handler is found in the program then it is an unhandled exception and the program is terminated with an error message.

Eg) Program to handle the divide by zero exception

```
a=int(input("Enter the numerator"))
b=int(input("Enter the denominator"))
try:
    c=a/b
    print("quotient:",c)
except ZeroDivisonError:
    print("Denominator cannot be zero")
```

O/P

```
Enter the numerator
9
Enter the denominator
0
Denominator cannot be zero
```

2) Multiple Except Blocks

Python allows multiple except blocks for a single try block. The block which matches with the exception generated will get executed.

Syntax:

```
try:
    operations are done in this block
    ....
    ------
except Exception 1:
    If there is Exception 1, then execute this block
except Exception 2:
    If there is Exception 2, then execute this block
    ....
else:
    If there is no execution then execute this block
```

E.g)

```
try:
    a=int(input("Enter the numerator"))
    b=int(input("Enter the denominator"))
    c=a/b
    print("Result:",c)
except ZeroDivisonError:
    Print("Denominator cannot be zero")
except valueError:
    Print("plese check the input"))
```

O/P

```
Enter the numerator
5
Enter the denominator
a
Please check the input
```

3) Multiple Exceptions in a single block:

An except clause may name multiple exceptions as a parenthesized tuple. So whatever exception is raised, out of the exceptions specified, the same Except block will be executed.

Syntax:

```
try:
    Operations are done in this block
except (Exception1, Exception2):
    execute this block while there is exception
    from the given Exception.
else:
    execute this code block.
```

(e.g)

```
try:
    a=int(input("Enter the numerator"))
    b=int(input("Enter the denominator"))
    c=a/b
    print("Result:",c)
except (ZeroDivisionError, ValueError):
    print("An exception occurs")
else:
    print("Division operation performed successfully")
```

O/P

```
Enter the numerator
20
Enter the denominator
0
An exception occurs
```

4) Except Block without Exception:

We can also specify an except block without mentioning any exception. It is called default handler, it can be placed after all other except blocks.

Syntax:

```
try:
    operations are done in this block
except:
    Execute this block while there is exception
else:
    Execute this code block
```

(e.g)
```
try:
    a=int(input("Enter the numerator"))
    b=int(input("Enter the denominator"))
    c=a/b
    print("Result:", c)
except:
    print("An exception occurs")
else:
    print("Divison operation performed
                    successfully")
```

o/p

Enter the numerator
10
Enter the denominator
0
An exception occurs

5) The Finally block:

It is an optional block which is used to define clean-up actions that must be executed under all circumstances.

Syntax:
```
try:
    write any operations here
    - - -
    Due to any exception, operations
    written here will be skipped.
finally:
    This would always be executed.
```

(e.g)
```
try:
    print("Raising Exception")
    raise ValueError
finally:
    print("This would always be
                executed")
```

o/p

Raising Exception.
This would always be executed.
Traceback (most recent call last):
File "C:\Python36\fh.py", line 4, in <module>
raise ValueError
ValueError

From the above code we conclude that when an exception occurs in the try block and is not handled by an except clause, then it is reraised after executing the finally block.

Exception with arguments:

It is possible to have exception with arguments. An argument is a value that gives additional information about the problem. we can accept the exceptions argument by passing variable in the except clause.

Syntax:

```
def fun(arg):
try:
    do the operation
except Exceptl type, Argument :
    Print value
```

The variable can receive a single value or multiple values in the form of a tuple. This tuple usually contains the error string, the error name and the error location.

(e.g)

```
def fun(var):
    try:
        return int(var)
    except ValueError, Argument:
        Print ("The argument does not
                contain numbers", argument)
fun('abc')
```

o/p

```
The argument does not contain numbers.
invalid literal for int() with base 10:
'abc'
```

Raising an exception:

We can raise user-real exceptions by using the raise statement.

Syntax:

```
raise [Exception [, args [, traceback]]]
```

Here Exception is the name of exception to be raised, args is optional and specifies a value for the exception argument. The final argument traceback is always optional, and if present, is the traceback object used for the exception.

(eg) a=int(input('Enter the parameter value'))

```
try:
    if a<=0:
        raise ValueError('Not a +ve Integer')
except ValueError as err:
    print(err)
else:
    print('positive number', a)
```

O/p

Enter the parameter value
-5
Not a +ve integer

Userdefined Exception:

Python allows programmers to create their own exceptions by creating a newexception class. The newexception class is derived from the baseclass Exception.

(eg)

```
class myError(Exception):
    def __init__(self, val):
        self.val = val
    def __str__(self):
        return repr(self.val)

try:
    raise myError(10)
except myError as e:
    print("user defined Exception generated with value", e.val)
```

O/p

user defined Exception generated with value 10

Packages in Python

A package is a hierarchical file directory structure that defines a single python application environment that consists of modules and subpackages.

Steps to create a python package

* Create a directory and insert package name
* Insert necessary classes to it
* Create a __init__.py file in that directory

The __init__.py file is necessary, because with this file Python will know that this directory is a package directory, other than an ordinary directory.

eg

Step 1: Create the package directory.
We create a directory named student

```
>>> import os
>>> os.mkdir("student")
```

Step 2: Now we create a class named 'dept', create a file named dept.py inside the student directory and write the following code in it.

```
class dept:
    def __init__(self):
        self.data = ['CSE','MECH','MCT','EEE','ECE',
                     'BME','IT','CIVLL']
    def printdata(self):
        Print("Details of department")
        for i in self.data:
            Print("%s" %i)
```

Step 3: Now we create another class named 'grade', Create a file named grade.py inside the student directory and write the following code in it

```
class grade :
    def __init__(self):
        self.val =['O','A+','A','B+',B]
    def printval(self):
        Print("Details of grade value")
        for i in self.val:
            Print("%s"%.i)
```

Step4: Finally we create a file named __init__.py inside the student directory and write the following code in it

```
from dept import dept
from grade import grade
```

Step5: Now we create a simple file named main.py is the same directory and write the following codes in it

```
from student import dept
from student import grade
a = dept()
a.printdata()
b = grade()
b.printval()
```

If we run main.py output can be displays as

Details of department Class

CSE
MECH
MCT
EEE
ECE
CIVIL
BME
IT

Details of grade class

O
A+
A
B+
B

Modules

Modules refer to a file containing, python statement and definitions. We use Modules to break down large programs into small manageable and organized files.

A module is a file that contains proper python code with .py extension. Modules can be imported from other modules using the import command.

Creating Modules:

A module is a python file that has only definitions of Variables, functions and classes. Create a module with the suffix .py and then import it using the import command.

(e.g) Create a module arithmetic.py containing definitions of functions.

```
def add(x,y):
    sum = x+y
    return(sum)
def sub(x,y):
    return(x-y)
def mul(x,y):
    return(x*y)
def div(x,y):
    return(x/y)
```

Importing Modules

Python Modules are essentially reusable libraries of code that can be imported and used in a program. There are 3 ways to import a module.

* TO import the entire module by including
  import modulename
* TO import module with renaming
  import modulename as newname
* TO import specific functions from the module.
  from modulename import name1, [name2]....

TO access the definitions in a module, we use dot operator between the modulename and the function.

(e.g) import arithmetic
Print (arithmetic.add(80,15))
Print (arithmetic.div(100,5))

o/p
95
20

from ..... import statement

We can import specific function name from a module without importing the module as a whole.

(e.g) from arithmetic import sub
Print (sub(80,40))

o/p
40

from math import pi
print ("pi = ", pi)

Import all functions from modules

We can import all functions from a module. We use the '*' after the import statement.

from modulename import *

(e.g) from arithmetic import *
Print (div(30,3)

o/p
10

Import with renaming

We can also import Modules with an alias name.

(e.g) import modulename as newname
import arithmetic.py as calc
Print (calc.add(20,60))

o/p
80

Locating Modules

While importing a module in a program must be located and loaded into memory before it can be used. Python first searches for the modules in the current working directory. If the module is not found there, it then looks for the Module in the directories specified in the PYTHONPATH environment variable

import sys
sys.path

The sys module contains functions and variables which provide access to the environment in which the python Interpreter runs.

(e.g) Program that defines a function large, a module which will be used to find larger of two values.

```
# making own module larger.py
def big (a,b):
    if (a>b):
        return a
    else:
        return b
```

main.py

```
import larger
print("Big = ", larger.big(10,20))
print("Big = ", larger.big('A','x'))
```

Built in Modules

Python has several built in modules, they are math modules, random module, time module, calendar module etc....

(e.g)

```
import random
print(random.randint(0,5))
print(random.random())
```

O/P

```
3
0.5088984396
```

(e.g)

```
import calendar
cal = calendar.month(2019,9)
print(cal)
```

O/P

```
    Mo Tu Wed Th Fr Sa Su
                    1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30
```

Random Module

1.Seed Method

The seed() method is used to initialize the random number generator.
The random number generator needs a number to start with (a seed value), to be able to generate a random number.

```
import random
random.seed(10)
print(random.random())   o/p: 0.5714025946899135
```

2. randrange()

The randrange() method returns a randomly selected element from the specified range.
Syntax: random.randrange(*start*, *stop*, *step*)

```
import random
print(random.randrange(6, 9)) o/p:7
```

3. randint()

The randint() method returns an integer number selected element from the specified range.(it includes the start and end index)

```
import random
print(random.randint(4, 12))o/p:12
```

4. choice()

The choice() method returns a randomly selected element from the specified sequence.
The sequence can be a string, a range, a list, a tuple or any other kind of sequence.

```
import random
mylist = ["red", "green", "blue",15,75]
print(random.choice(mylist))    o/p:75
```

5.shuffle()

The shuffle() method takes a sequence, like a list, and reorganize the order of the items.

```
import random
mylist = ["red", "green", "blue",15,75]
random.shuffle(mylist)
print(mylist)  o/p: [["red", 15,"green", 75,"blue"].
```

6.uniform()

The uniform() method returns a random floating number between the two specified numbers (both included).

```
import random
print(random.uniform(25, 60))]   o/p 31.61817527553314
```

Math module

1.Ceil()

The math.ceil() method rounds a number UP to the nearest integer, if necessary, and returns the result.

```
import math
print(math.ceil(2.3))     o/p 3
print(math.ceil(6.8))     o/p 9
print(math.ceil(-8.2))    o/p -8
```

2.gcd

The math.gcd() method returns the greatest common divisor of the two integers *x* and *y*.

```
import math
print (math.gcd(13, 6))   o/p 1
print (math.gcd(16, 12))  o/p 4
```

3.floor()

The math.floor() method rounds a number DOWN to the nearest integer, if necessary, and returns the result

```
import math
print(math.floor(7.6)) o/p:7
print(math.floor(.8))  o/p:0
```

4.pow()

The math.pow() method returns the value of x raised to power y.If x is negative and y is not an integer, it returns a ValueError.

```
import math
print(math.pow(8, 4))  o/p 4076.0
```

5.trunc()

The math.trunc() method returns the truncated integer part of a number. This method will NOT round the number up/down to the nearest integer, but simply remove the decimals.

```
import math
print(math.trunc(11.77))  o/p:11
print(math.trunc(15.32))  o/p:15
```

6.sqrt()

The math.sqrt() method returns the square root of a number.The number must be greater than or equal to 0.

```
import math
math.sqrt(64)   o/p 8.0
math.sqrt(75)    o/p: 8.660254037844387
```

7.fabs()

The math.fabs() method returns the absolute value of a number, as a float.Absolute denotes a non-negative number. This removes the negative sign of the value if it has any.

```
import math
print(math.fabs(-6.73))   o/p:6.73
print(math.fabs(-14))     o/p:14
```

8. fmod

The math.fmod() method returns the remainder (modulo) of x/y.

```
Import math
print(math.fmod(29, 6))  o/p 5.0
```

Illustrative programs

1. Write a python program to count the number of lines and no. of words and no. of characters in a file.

```
import os, sys
fname = input ('Enter filename')
if os.path.isfile (fname):
    f. open(fname, 'r')
else:
    Print ("file not exist")
    sys. exit()
cl = cw = cc = 0
for line in f:
    words = line. Split()
        cl += 1
        cw+ = len (words)
        cc+ = len(line)
Print ("No. of lines ", cl)
Print ("No. of words", cw)
Print ("No. of char ", cc)
f. close()
```

file1. txt

English
Maths
Physics ....

O/P

```
Enter filename
    file1.txt
No. of lines 4
No. of words 6
No. of char 39
```

(2) write a python program to copy a text file to another text file.

```
f1 = input ("Enter source file')
f2 = input ('Enter dest file')
fr = open (f1, 'r')
fw = open(f2, 'w')
for line in fr.readlines():
    fw - write(line)
fr. close()
fw. close()
Print ("file copied successfully")
```

O/P

```
Enter source file
    file1.txt
Enter dest file
    file2.txt
file copied successfully.
```

3) Voter's age Validation

```
try:
    age = int(input("Enter the age : "))
    if (age >= 18):
        Print ("Eligible to vote")
    else:
        raise ValueError("Not eligible to vote")
except ValueError as err:
    Print(err)
```

Output:

```
Enter the age
20
Eligible to vote

Enter the age
12
Not eligible to vote
```

4) Marks Range validation (0-100):

```
try:
    mark = int(input("Enter the mark : "))
    if (mark >= 0) and (mark <= 100):
        Print ("valid marks Range")
    else:
        raise ValueError("Invalid Range")
except ValueError as err:
    Print(err).
```

output:

```
Enter the mark
60
valid marks Range

Enter the mark
-3
Invalid Range
```

User-Defined Exception

```
Class invalidAge(Exception):
    def display(self):
        Print("Sorry, Age cannot be below 18")

try:
    age = int(input())
    if (age < 18):
        raise invalidAge
except invalidAge as e:
    e.display()
else:
    Print("eligible")
```

output:

```
20
eligible

12
Sorry, Age cannot be
below 18
```

2 Marks

1) Write a python script to display the current date & time. [AU Jan 18, R17]

```
import datatime
now = datetime.datetime.now()
Print("current date and time using str method of datetime object")
Print(str(now))
```

output: current date and time using str method of datetime object:

2018-02-10 16:34:40.27829

2) Write notes on Modular Design [AU Jan 18, R17]

* Modules are prewritten pieces of code that are used to perform common tasks like generating random numbers, Performing mathematical operations.

* A Module is a file with .py extension that has definitions of all functions and variables that would use in other programs.

* Modules are used to breakdown large programs into small manageable and organized files and it provides reusability of code.

3) What is file?

A File is a sequence of characters stored on a permanent medium like hard drive, flash memory (or) CD-ROM

4) what is Format operator?

* Format operator is an operator (%) that takes a format string and a tuple as input and generates a string that includes the Elements of the tuple, formatted as specified by the format string

* when applied to integers % is the modulus operator But when the first operand is a string % is the format operator.

5) What is command line argument?

Python provides a getopt module that helps to to pass command-line options and arguments

$ python test.py arg1 arg2 arg3

6) what is modules?

Modules are a file containing Python statements and definitions.

7) what are packages?

* A package is just a way of collecting related modules together within a single tree like hierarchy. A well organised hierarchy of directories for easier access.

* Python has packages for directories and modules for files.

8) What are Exceptions?

Errors can also occur at runtime and these are called Exceptions

(e.g) A file trying to open does not exist (FileNot found Error), dividing a number by zero (zeroDivisionError), module trying to import a module and is not found (import Error) etc, are exceptions.