GE3151 - Problem Solving and python Programming

UNIT-I

Computational Thinking and problem Solving

Fundamentals of Computing:

Computer:

A computer is an information processing machine which consists of a number of interrelated components that work together with the aim of converting data into information.

* Data is entered through input devices
* Data is processed using the central processing unit
* The processed data (information) is displayed to users using output devices.

All these parts constitute the computer hardware.

Computer organization:

The basic hardware units of a computer system are,

1) Central processing Unit
2) Memory Unit
3) Input Unit
4) output Unit

Input Devices:

Input devices are electromechanical device. Human communicate with computers through input devices. Data and instruction are entered into memory of a computer through input devices. It captures information and translates it into a form that can be processed by cpu.

Eg: Keyboard, mouse, Joystick, Scanners.

output Devices:

Computers communicate with human being using output devices. output device take the machine-coded output result from cpu & convert them into a form that is easily readable. output unit processes data into useful information. The outputs are in the form of softcopy

and hardcopy. The physical form of output is known as hardcopy. The electronic version of an output that reside in computer memory/disk is known as softcopy.
Eg: Monitors, printers and audio response.

Central processing Unit:

It is the brain of a computer system. It converts data (input) into information (output). It controls all internal and external devices, performs arithmetic and logic operations and operate only on binary data ie data composed of 1s and 0s. It also controls the usage of main memory to store data and instructions and controls the sequence of operations. CPU consists of 3 main subsystems.

Arithmetic/Logic Unit(ALU):

It contains the electronic circuitry that executes all arithmetic & logical operations on the data. It comprises of 2 units.

* Arithmetic unit contains the arithmetic calculations such as addition, subtraction, multiplication and division.
* Logic unit enable the cpu to perform logical operation based on instruction provided to it. It can compare numbers, letters or special characters. It test 3 conditions: equal to, less than and greater than.

Control Unit:

It checks the correctness of sequence of operations. It fetches program instruction from primary unit, interprets them and ensures correct execution of program. It controls the input/output devices and directs the overall functioning of other units of computer.

Registers:

Special-purpose, high speed temporary memory units holds various types of information such as data, instructions, addresses and the intermediate results of calculations. It hold the information that cpu is currently working on. It can be thought of

as cpu's working memory.

Memory Unit:

Components such as input device, output device and cpu are not sufficient for the working of the computer. A Storage area is needed in a computer to store data and instructions either temporarily or permanently. The memory unit stores information is a group of memory called memory locations, as binary digits. Each memory location has a unique memory address. The computer memory is broadly classified as

* Primary memory (to handle data)
* Secondary memory (to store the output)

Primary Memory:

It is known as main memory. It Stores data and instructions for processing. A memory module interface with other parts of computer system through a set of address lines, data lines & control lines. These are called address bus, data bus & control bus. It is classified into

* Random Access Memory (RAM) - Volatile memory
* Read only Memory (ROM)

Secondary Memory:

Auxiliary memory or external memory or bulk memory. Instructions and data stored in storage devices are permanent in nature (non-volatile). The access time in secondary memory is much large than in primary memory. Common secondary storage devices are magnetic disks & tapes.

Memory operations:

Some operations are common to both primary and secondary memory. They are,

Read operation:

During this operation, data is retrieved from memory.

Write operation:

In this operation, data is stored in the memory.

Unit of Memory:

The smallest block of memory is considered to be a byte which comprises eight bits. The total memory space is measured in terms of bytes. The unit of memory is a byte. The memory capacity is also expressed in number of bytes.

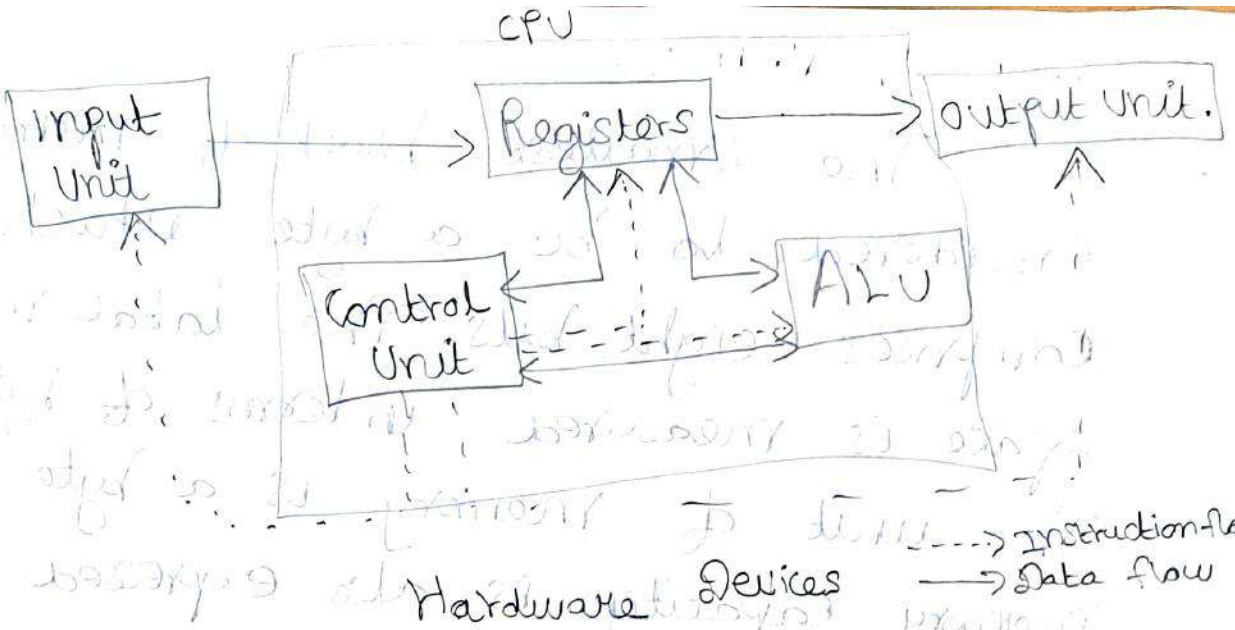
Byte:
The smallest meaningful storage block. It represents one addressable storage location in memory. It consists of eight bits.

KB:
one kilobyte is equal to 1024 bytes.

MB:
one megabyte is equal to 1024 kilobytes.

GIB:
one Gigabyte is equal to 1024 megabytes.

TB:
one terabyte is equal to 1024 gigabytes.



Hardware Devices

Classification of Computers:-

Based on physical size, performance and application areas, computers are divided into 4 major categories.

Micro Computer:-

A micro computer is a small, low cost digital computer. It's designed for individual users. Eg: IBM-pc pentium 100, IBM-pc pentium 200. It includes desktop, laptop & handheld.

Micro Computer - Desktop:-

It is known as personal computer (PC). It consists of system unit, display monitor, a keyboard, internal hard disk storage & other peripheral devices.

Micro Computers - Laptop:

It is a portable computer. It is called as "notebook". It can use anywhere and at anytime. It do not need external power supply.

Micro Computers - Hand Held:

It is a personal Digital Assistant (PDA). It's convenient to be stored in a pocket.

Mini Computers:

It is a mid-range computer. It is a small digital computer. It is capable of supporting from 4 to about 200 simultaneous users.

Mainframe Computer:

It is an ultra high performance computer made for high-volume, processor intensive computing. It's used by large businesses and for scientific purposes.

Eg: CDC 6600, IBM's ES000.

Super Computer:

It's designed to maximize the number of floating point operation per second. It has solving scientific & engineering

problems. It contains number of CPUs that operate in parallel to make it faster.
Eg: CRAY-3, Cyber 205 and PARAM.

Identification of Computational Problems:

It is fundamentally about computational problem solving ie solving problems by the use of computation. It is any type of calculation that includes both arithmetic and non-arithmetic steps and follows a well-defined model, Eg; Algorithm. Computational thinking can be split into 4 parts,

* Decomposition
* pattern Recognition
* Abstraction
* Algorithmic Design.

Requirements to solve Computational problems:

To solve a problem Computationally, 2 things are needed,

* A representation that captures all the relevant aspects of the problem.
* An algorithm that solves the problem by use of the representation.

Algorithms:

The term algorithm was derived from the name of Mohammed al-Khawarizmi, a persian mathematician in the ninth century. Al-Khawarizmi → Algorismus (in Latin) → Algorithm.

An algorithm is a well-defined computational procedure consisting of a set of instructions that takes some value or set of values as input and produces some value or set of values as output.

In other word, an algorithm is a procedure that accepts data, manipulate them to fill the required unknown with the desired value(s).

Input ⟹ Algorithm ⟹ output

People of different professions have their own form of procedure in their line of work and they call it different names. For instance, a cook follows a procedure commonly known as a recipe that converts the ingredients (input) into some culinary dish (output), after a certain number of steps.

Characteristics of a Good Algorithm:

1) **Precision:**
The steps are precisely stated (defined)

2) **Uniqueness:**
Result of each step are uniquely defined and only depend on the input and the result of the preceding steps.

3) **Finiteness:**
The algorithm stops after a finite number of instructions are executed.

4) **Effectiveness:**
Algorithm should be most effective among many different ways to solve a problem.

5) **Input:**
The algorithm receives input

6) **output:**
The algorithm produces output

7) **Generality:**
The algorithm applies to a set of inputs.

Building Blocks of Algorithm (Statements, State, Control flow, functions)

An algorithm is an effective method that can be expressed within a finite amount of space and time in a well-defined formal language for calculating a function. Starting from an initial state and initial input, the instructions describe a computation when executed proceeds through a finite number of states. The transition from one state to the next is not deterministic; that algorithms known as randomized algorithms.

1) Statements:

Statement is a single action in a computer. In a computer, it include some of the following actions.

* **Input Data:**
  Information given to the program
* **Process Data:**
  Perform operation on a given input
* **output Data:**
  processed result

2) State:

The State of an algorithm is defined as its condition regarding Stored data. The Stored data in an algorithm are Stored as Variables or Constants. The State Shows its current values or Contents.

3) Control flow:

In Computer Science, Control flow (or flow of control) is the order in which individual Statements, instructions or function calls of an algorithm are executed or evaluated. It can be executed in 3 ways.

i) Sequence:

A Sequence is a Series of Steps that occur one after the other, in the Same order every time.

Eg: Consider a car Starting off from a Set of lights. Imagine the road in front of the car is clear for many miles and the driver has no need to Slow down or Stop. The car will Start in first gear, then move to Second, third, fourth and finally fifth. A good driver (who doesn't have to Slow down or Stop) will move through the gears

in this sequence without skipping gears.

ii) Selection:

A selection is a decision that has to be made. If the condition test is true, one part of the program will be executed, otherwise it will execute the other part of the program.

Eg:

Consider a car approaching a set of lights. If the lights turns from green to yellow, the driver will have to decide whether to stop or continue through the intersection. If the driver stops, she will have to wait until the lights are green again.

iii) Iteration:

Iteration is sometimes called repetition, which means a set of steps which are repeated over and over until some event occurs.

Eg:

Consider the driver at red light, waiting for it to turn green. She will check the lights and wait. This sequence will be repeated until the lights turn green

4) Functions:

A function is a sub program which consists of block of code (set of instructions) that perform a particular task. For complex problems, the problem is been divided into smaller and simpler functions during algorithm design.

Notation (pseudo code, flowchart, programming language):

There are 4 different notation of representing algorithms.

1) Algorithm as Step-form:

It is a step-by-step procedure for solving a task or problem. The steps must be ordered, unambiguous and finite in number. It is English like representation of logic which is used to solve the problem. Some guidelines are,

* Keep in mind that algorithm is a Step-by-Step process.
* Use begin or start to start the process.
* Include variables and their usage
* If there are any loops, try to give sub number lists.

* Try to give Go back to Step number if loop or condition fails.
* Use jump Statement to jump from one Statement to another.
* Try to avoid unwanted raw data in algorithm.
* Define expressions
* Use break and Stop to terminate the process.

Eg: To find the greatest among 3 numbers (one way)

1) Start
2) Read the three numbers A, B, C
3) Compare A and B. If A is greater perform Step 4 else perform Step 5.
4) Compare A and C. If A is greater, output "A is greater" else output "C is greater".
5) Compare B and C. If B is greater, output "B is greatest" else output "C is greatest"
6) Stop

Eg: To find the greatest among 3 numbers (Another way)

1) Start
2) Read the three numbers A, B, C

3) Compare A and B. If A is greater, Store A in MAX, else store B in MAX.
4) Compare MAX and C. If MAX is greater, output "MAX is greater" else output "C is greater".
5) Stop.

2) pseudocode:

pseudocode ("sort of code") is another way of describing algorithms. It is called "pseudo" code because of its strong resemblance to "real" program code. pseudocode is essentially english with some defined rules of structure and some keywords that make it appear a bit like program code.

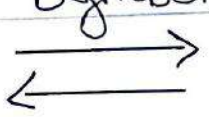
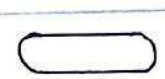
Some guidelines for writing pseudocode are,

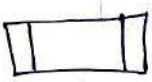
* For start and finish BEGIN MAINPROGRAM, END MAINPROGRAM. This is often abbreviated to BEGIN and END especially in smaller programs.
* for initialisation INITIALISATION, END INITIALISATION this part is optional., though it makes program structure clear
* for subprogram BEGIN SUBPROGRAM, END SUBPROGRAM
* for selection IF, THEN, ELSE, ENDIF

* for multi-way selection CASEWHERE, OTHERWISE, ENDCASE
* for pre-test repetition WHILE, ENDWHILE
* for post-test repetition REPEAT, UNTIL
* Keywords are written in capitals
* Structural elements come in pairs. eg for every BEGIN there is an END, for every IF there is an ENDIF, etc
* Indenting is used to show structure in the algorithm
* The names of subprograms are underlined

3) Flowcharts:

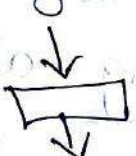
Flowcharts are a diagrammatic method of representing algorithms. The purpose of flowchart is making the logic of the program clear in a visual representation.

Symbol	Symbol Name	Description
	Flow Lines	Used to Connect Symbols
	Terminal	Used to Start, pause or halt in program logic
	Input/output	Represent the information entering or leaving system
	Processing	Represents arithmetic and logical instructions

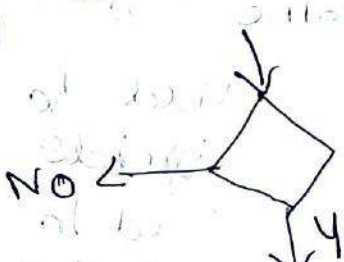
	Decision	Represent a decision to be made
	Connector	Used to join different flow lines
	Sub function	Used to call function

Rules for drawing a flowchart:

1) The flowchart should be clear, neat and easy to follow
2) The flowchart must have a logical start and finish
3) only one flow line should come out from a process symbol.



4) only one flow line should enter a decision symbol. However, two or three flow lines may leave the decision symbol.



5) only one flow line is used with a terminal symbol.

6) Within standard symbols, write briefly & precisely.

7) Intersection of flow lines should be avoided. (21)

Advantages of Flowchart:

1) Communication
2) Effective Analysis
3) Proper documentation
4) Efficient coding
5) Proper debugging
6) Efficient program maintenance

Disadvantages of Flowchart:

1) Complex logic
2) Alterations and modifications
3) Reproduction
4) Cost

4) Programming Language:

A programming language is a set of symbols and rules for instructing a computer to perform specific tasks. The programmers have to follow all the specified rules before writing program using programming language. The user has to communicate with the computer using language which it can understand.

Types of Programming Language:

1) Machine Language
2) Assembly Language
3) High Level Language

Algorithmic problem Solving:

It is solving problem that require the formulation of an algorithm for the solution.

1) Understanding the problem:

The problem given should be understood completely. check if it is similar to some standard problems and if a known algorithm exists. Otherwise a new algorithm has to be developed.

2) Ascertain the capabilities of the computational device:-

Once a problem is understood, need to know the capabilities of the computing device can be done by knowing the type of the architecture, speed and memory availability.

3) Exact / Approximate Solution:-

Once algorithm is developed, it is necessary to show that it computes

answer for all the possible legal inputs. The Solution is Stated in two forms, Exact Solution or approximate Solution. Examples of problems where an exact Solution cannot be obtained are

i) Finding a square root of number
ii) Solutions of nonlinear equations

4) Decide on appropriate data structure:-

Some algorithms do not demand any ingenuity in representing their inputs. Some others are infact are predicted on ingenious data structures. A data type is a well-defined collection of data with a well-defined set of operations on it. A data structure is an actual implementation of a particular abstract data type. The elementary data structures are Arrays: access lots of data fast. It can have arrays of any other data type. However, can't make arrays bigger if program decides it needs more space. Records: It organize Non-homogeneous data into logical packages to keep everything together. These packages do not include operations, just data fields. Record do

not help process distinct items in loops.
Sets:- It represent subsets of a set with operations as intersection, union and equivalence. Built-in sets are limited to a certain small size.

5) Algorithm design techniques:-

Creating an algorithm is an art which may never be fully automated. By mastering design strategies, it become easier to develop new and useful algorithms. Dynamic programming is one such technique. Some of the techniques are especially useful in fields other than computer science such as operation research and electrical engineering. Some important design techniques are linear, nonlinear and integer programming.

6) Methods of Specifying an algorithm:-

There are mainly 2 options for specifying an algorithm: use of natural language or pseudocode & flowcharts. A pseudocode is a mixture of natural language & programming language like constructs

A flowchart is a method of expressing an algorithm by a collection of connected geometric shapes.

7) Proving algorithms Correctness::

Once algorithm is developed it is necessary to show that it computes answer for all the possible legal inputs. It refer to this process as algorithm validation. The process of validation is to assure that algorithm will work correctly independent of issues concerning programming language it will be written in.

8) Analysing algorithms::

As an algorithm is executed, it uses the computers central processing unit to perform operations and its memory (both immediate & auxiliary) to hold the program and data. Analysis of algorithms and performance analysis refers to the task of determining how much computing time and storage an algorithm requires. It is a challenging area in which requires grate mathematical

Skill. An important result of study is it allows to make quantitative judgments about the value of one algorithm over another. Another result is it allows to predict whether the software will meet any efficiency constraint that exits.

Simple Strategies for developing algorithms (Iteration, Recursion):

There are two commonly used strategies used in developing an algorithm. They are Iteration & Recursion. Basically iteration & recursion perform the same kind of task.

1) Iteration:

In Iteration the control statements such as for loop, do-while loop or while is used. It continues its execution until the terminating condition is reached.

Eg: Consider an example of factorial.

It can define the factorial of some number n as a product of all the

integers from n to 1.

Eg: If the 5 factorial has to be calculated then it will be

$= 5 * 4 * 3 * 2 * 1$

$= 120$

Similarly $3! = 3 * 2 * 1$

$= 6$

$0! = 1$

The explanation mark is used to denote the factorial. The definition of factorial function as

$n! = 1$ if $n == 0$

otherwise $n! = n * (n-1) * (n-2) * \ldots * 1$ if $n > 0$

If the value of n is any of the 0, 1, 2, 3 or 4 then the definition will be

$0! = 1$

$1! = 1$

$2! = 2 * 1$

$3! = 3 * 2 * 1$

$4! = 4 * 3 * 2 * 1$

presenting an algorithm that takes the input as value of n and returns the result of n!

Algorithm for factorial function using iterative definition:

```
prod = 1;
x = n;
while (x > 0)
{
prod = prod * x;
x--;
}
return (prod);
```

Such an algorithm is called as an iterative algorithm because it calls explicit repetition of some process until certain condition is met.

Recursion:

It is a method of solving problems that involves breaking a problem down into smaller and smaller subproblems until get to a small enough problem that it can be solved trivially.

Properties of Recursion:

There are 3 important laws of recursion.

1) A recursive algorithm must have a base case.
2) A recursive algorithm must change its state and move toward the base case.
3) A recursive algorithm must call itself, recursively.

```
Fact(5)
  ↓
5* Fact(4)
  ↓
5*4* Fact(3)
  ↓
5*4*3* Fact(2)
  ↓
5*4*3*2* Fact(1)
  ↓
5*4*3*2*1   =120
```


Illustrative problems: Find Minimum in a list:

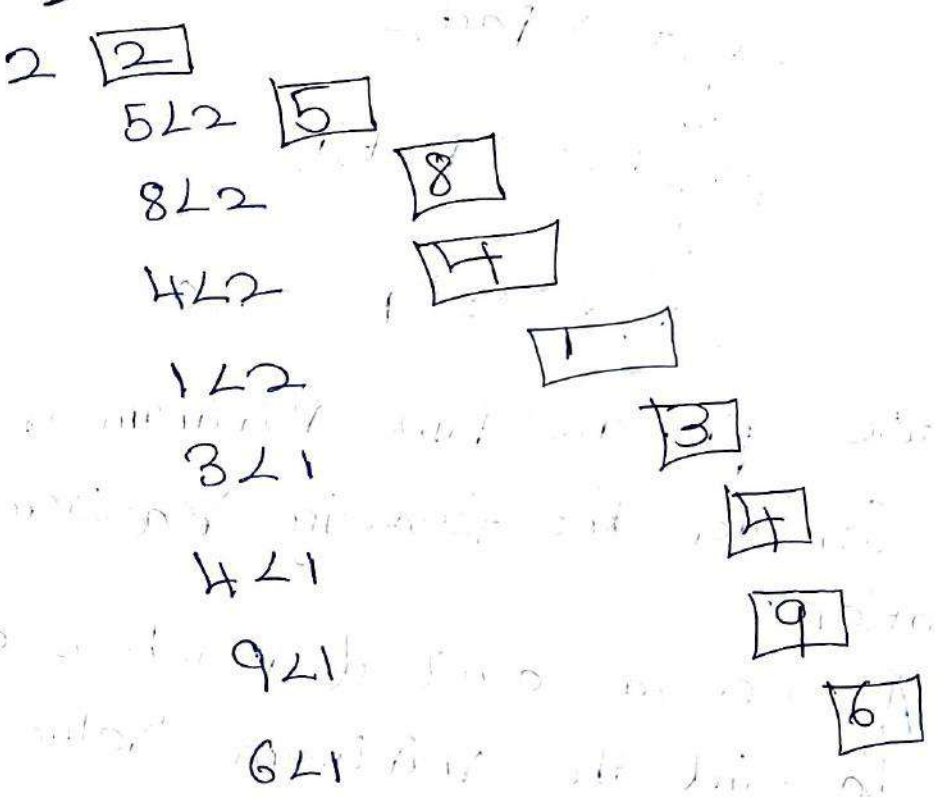
Consider the following requirement specification:

A user has a list of numbers and wishes to find the minimum value in the list. An algorithm is required which will allow the user to enter the numbers,

and which will calculate the minimum value that are input. The user is quite happy to enter a count of the numbers in the list before entering the numbers.

Finding the minimum value in a list of items is not difficult. Take the first element and compare its value against the values of all other elements. Once find a smaller element continue the comparisons with its value. Finally find the minimum.

2 5 8 4 1 3 4 9 6



1 is the minimum

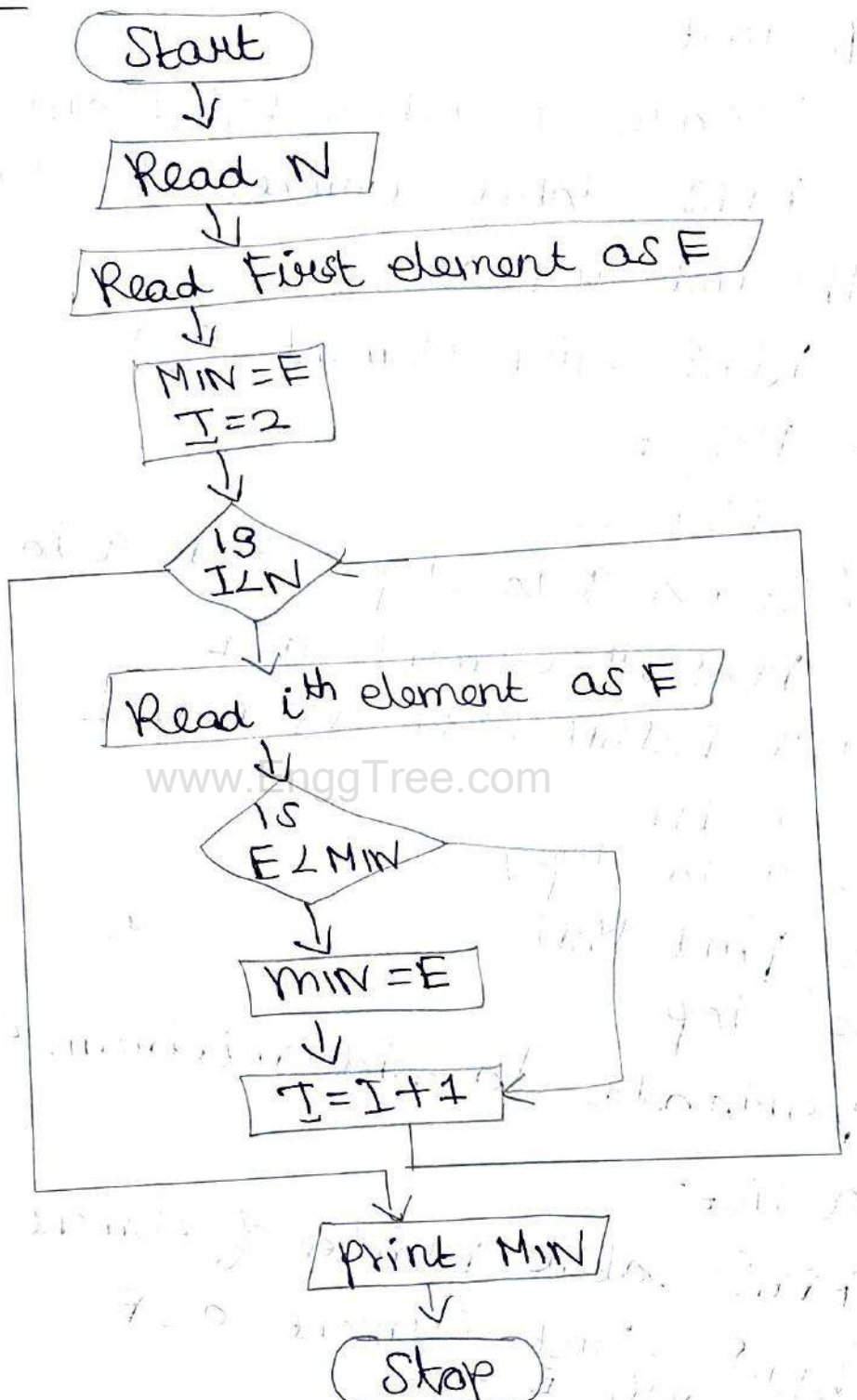
Step base algorithm to find minimum element in a list:

1. Start
2. Declare variables N, E, i and MIN
3. READ total number of element in the list as N.
4. READ first element as E
5. MIN = E
6. SET i = 2
7. IF i > n go to step 12 ELSE go to step 8
8. READ ith element as E
9. IF E < MIN THEN SET MIN = E
10. i = i + 1
11. go to Step 7
12. Print MIN
13. Stop

Pseudocode to find minimum element in a list:

```
READ Total number of element in List as N
READ first element as E
SET MIN = E
SET i = 2
WHILE i <= n
      READ ith element as E
      IF E < MIN THEN
            SET MIN = E
      END IF
      INCREMENT i by one
END WHILE
PRINT MIN
```

Flowchart to find minimum element in a list:



Illustrative problems: Insert a card in a list of Sorted Cards:

It is same as inserting an element into a sorted array. It start from the high end of the array and check to see it want to insert the data. If so fine. If not, move the preceding element up one and then check to see it want to insert x in the "hole" left behind. Repeat this step as necessary.

Thus the search for the place to insert X and the shifting of the higher elements of the array are accomplished together.

position	0	1	2	3	4	5
original List	4	6	9	10	11	
7 711 X	4	6	9	10	→	11
7710 X	4	6	9	→	10	11
779 X	4	6	→	9	10	11
776 ✓	4	6	7	9	10	11
			↑			

Element to be insert 7

Step based algorithm:

1. Start
2. Declare Variables N, list[], i and X.
3. READ number of element in Sorted list as N
4. SET i=0
5. IF i<N THEN go to step 6 ELSE go to step 9
6. READ Sorted list element as List[i]
7. i=i+1
8. go to step5
9. READ element to be insert as X
10. SET i=N-1
11. IF i>=0 and X<List[i] Then go to step12 ELSE go to step15
12. List[i+1]= List[i]
13. i=i-1
14. go to Step 11.
15. List[i+1]=X

Pseudo Code:

READ number of element in Sorted list as N

SET i=0

WHILE i<N

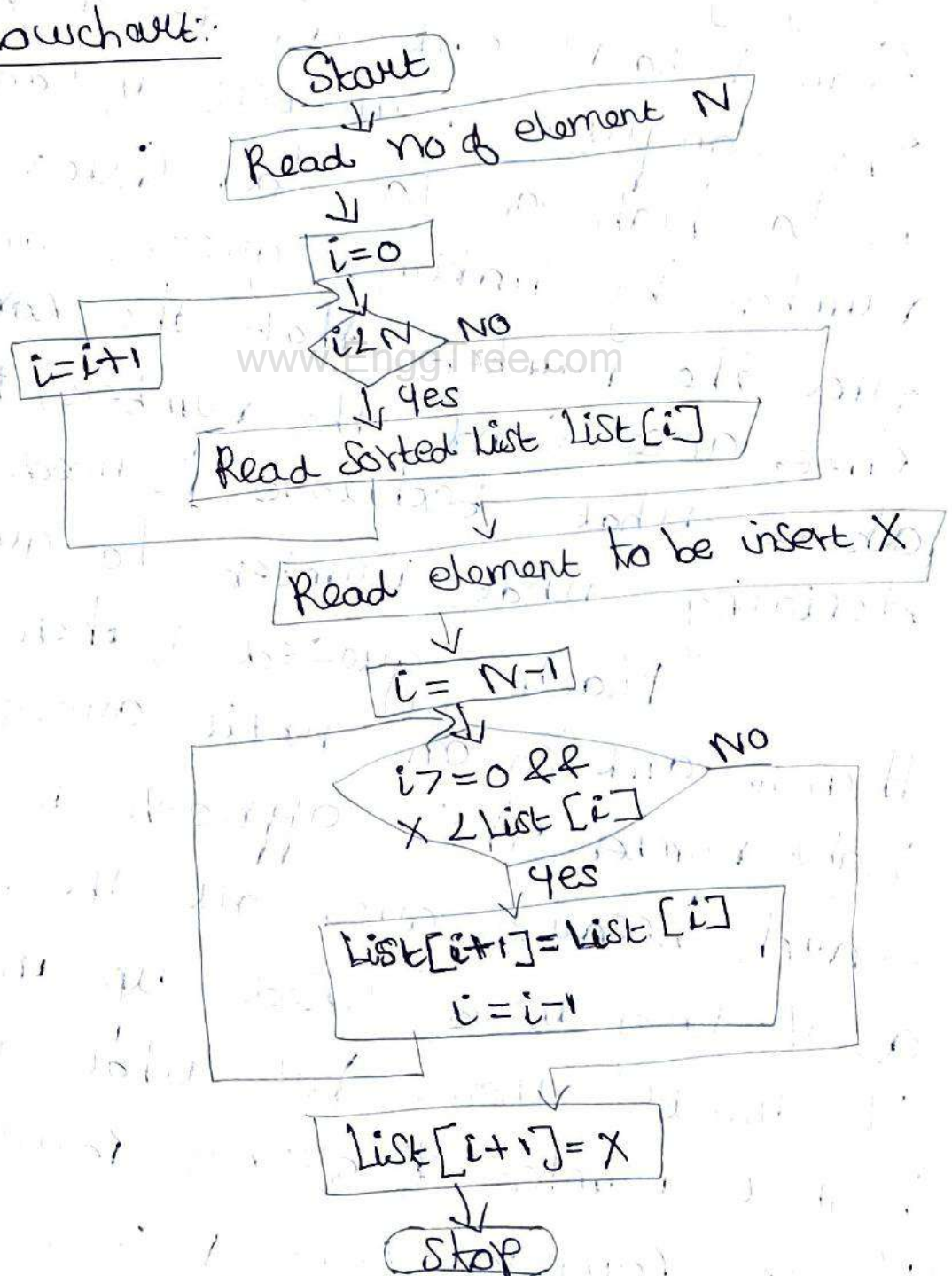
READ Sorted list element as List[i].

i=i+1

ENDWHILE

READ element to be insert as X...
SET i = N-1
WHILE i >= 0 and X < List[i]
    List[i+1] = List[i]
    i = i-1
END WHILE
List[i+1] = X

Flowchart:



Illustrative problems: Guess an integer number in a range:

Let's play a little game to give an idea of how different algorithms for the same problem can have wildly different efficiencies. The computer is going to randomly select an integer from 1 to N and ask to guess it. The computer will tell if each guess is too high or too low. Guess the number by making guesses until find the number that the computer chose. once found the number, think about what technique is used when deciding what number to guess next?

Maybe guessed 1, then 2, then 3, then 4 and so on until guessed the right number. This approach is linear search because guess all the numbers as if they were lined up in a row. It would work. But what is the highest number of guesses could need? If the computer selects N, would need N.

guesses. Then again could be really lucky, which would be when the Computer selects 1 and get the number on first guess. How about on average? If the Computer is equally likely to select any number from 1 to N then on average need N/2 guesses.

But could do something more efficient than just guessing 1, 2, 3, 4... right? Since the Computer tells whether a guess is too low, too high or correct can start off by guessing N/2. If the number that the Computer selected is less than N/2 then N/2 is too high, can eliminate all the numbers from N/2 to N from further consideration. If the number selected by the Computer is greater than N/2 then can eliminate 1 through N/2. Either way can eliminate about half the numbers. On next guess, eliminate half of the remaining numbers. Keep going, always eliminating half of the remaining numbers.

It call this halving approach binary search and no matter which number from 1 to N the computer has selected, should be able to find the number in almost $\log_2 N+1$ guesses with this technique. The following table shows the maximum number of guesses for linear and binary search for a few number sizes

Highest Number	Max Linear Search Guesses	Max Binary Search Guesses
10	10	4
100	100	7
1000	1000	10
10,000	10,000	14
1,00,000	1,00,000	17
10,00,000	10,00,000	20

Step base algorithm:

1) Start
2) SET Count = 0
3) READ range as N
4) SELECT an RANDOM NUMBER from 1 to N as R.

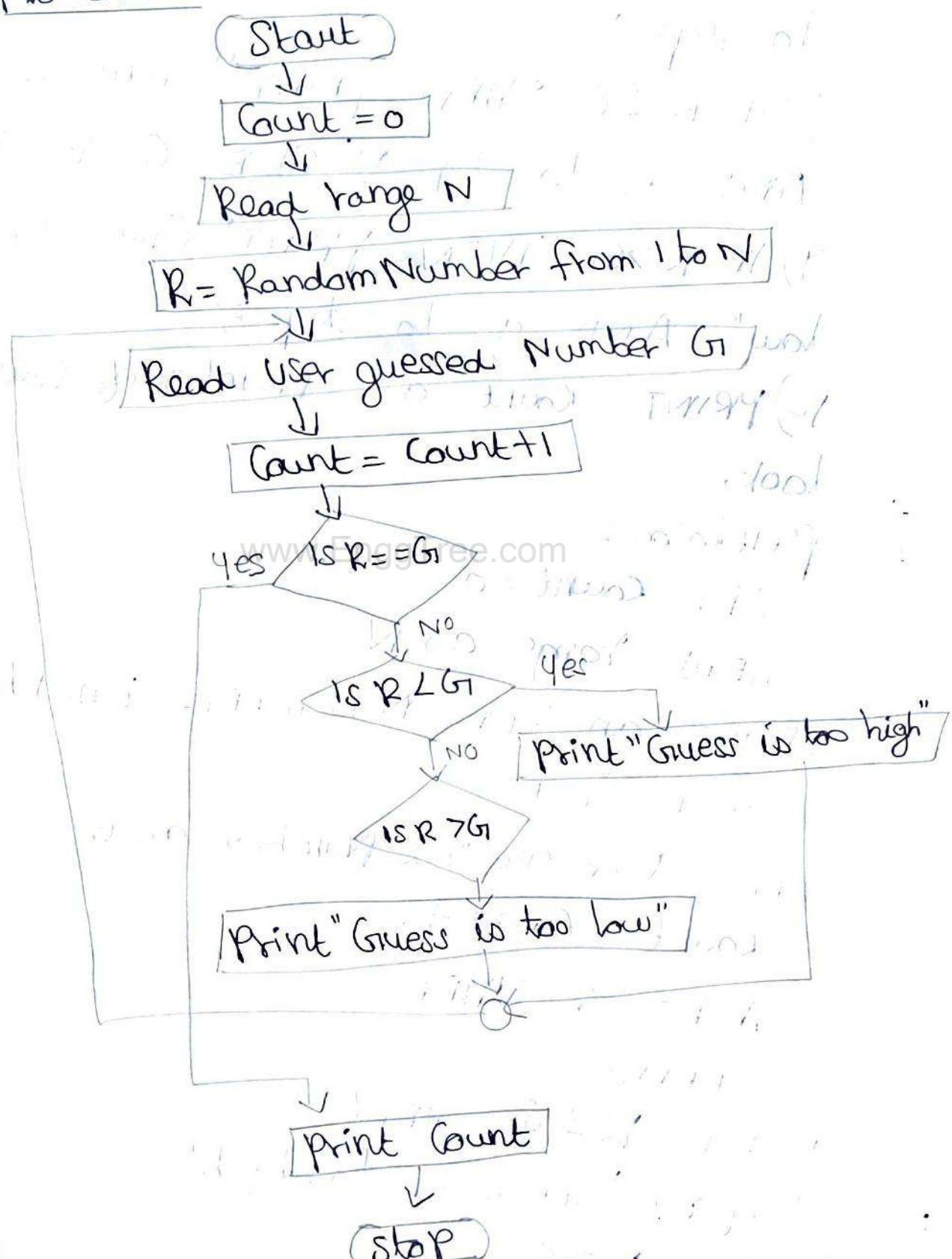
5) READ User Guessed Number as G1
6) count = Count + 1
7) If R = G1 THEN go to step 10 ELSE go to step 8
8) IF R < G1 THEN PRINT "Guess is too high" AND go to step 5 ELSE GO TO STEP 9
9) IF R > G1 THEN PRINT "Guess is too low" AND go to step 5
10) PRINT Count as Number of Guesses took.

pseudoCode:


```
SET count = 0
READ range as N
SELECT an RANDOM NUMBER from 1 to N as R
WHILE TRUE
READ User guessed Number as G1
  Count = Count + 1
  IF R == G1 THEN
    BREAK
ELSEIF R < G1 THEN
 DISPLAY "Guess is too high"
ELSEIF R > G1 THEN
 DISPLAY "Guess is too low"
```

ENDIF
ENDWHILE
DISPLAY Count as Number of guesses took.

Flowchart:



Eg: 1) 1-100:

R = 42, G1 = 50

R < G1. Guess is too high. So Consider 1-49

2) 1-49:

R = 42, G1 = 25

R > G1. Guess is too low. So Consider 26-49

3) 26-49:

R = 42, G1 = 37,

R > G1. Guess is too low. So Consider 38-49

4) 38-49:

R = 42, G1 = 43

R < G1. Guess is too high. So Consider 38-42

5) 38-42:

R = 42, G1 = 40

R > G1. Guess is too low. So Consider 41-42.

6) 41-42:

R = 42, G1 = 41

R > G1 Guess is too low. So answer is 42.

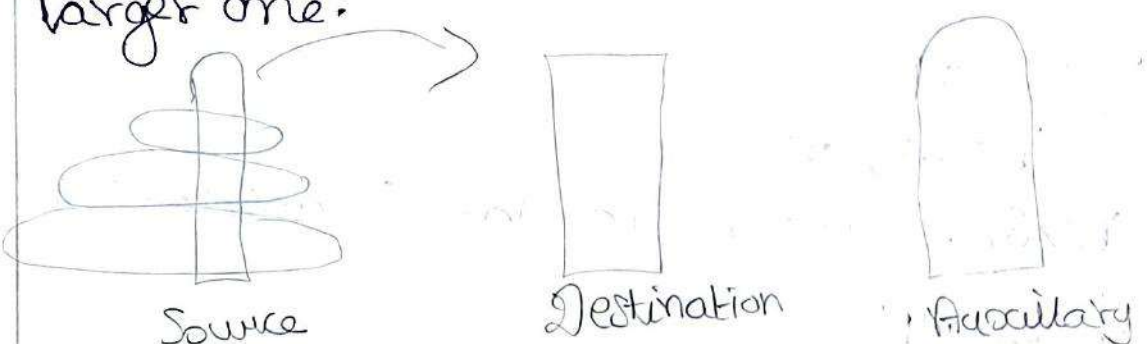
7) 42:

R = 42, G1 = 42. No of 7 guesses took.

Towers of Hanoi:

It is a mathematical puzzle

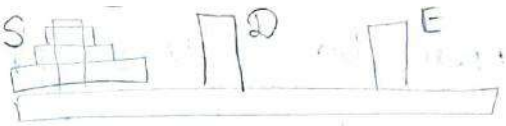
which consists of 3 towers (pegs) and more than one ring is as depicted. These rings are of different sizes and stacked upon in an ascending order ie the smaller one sits over the larger one.



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

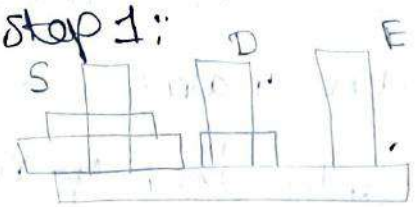
Rules:

1) only one disk can be moved among the towers at any given time.
2) only the "top" disk can be removed
3) No large disk can sit over a small disk.

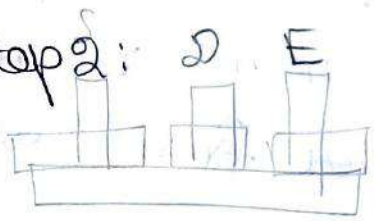


Initial State

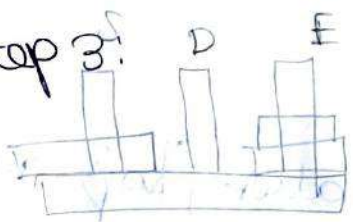
Step 1:



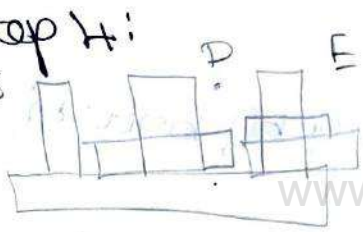
Step 2:



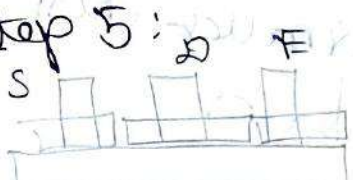
Step 3:



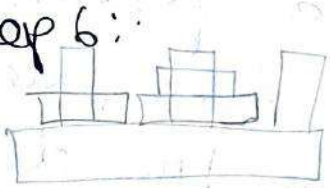
Step 4:



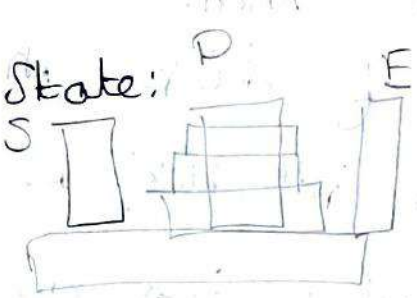
Step 5:



Step 6:



Final State:



Algorithm:

To write an algorithm for Tower of

Hanoi, learn how to solve this problem with lesser amount of disks, say 1 or 2. It mark 3 towers with name, source, destination and aux (only to help moving the disk). If having only one disk, then it can easily be moved from source to destination tower.

If having 2 disks:

1) First move the smaller (top) disk to aux tower.
2) Then move the larger (bottom) disk to destination tower.
3) Finally move the smaller disk from aux to destination tower.

So now are in a position to design an algorithm for Tower of Hanoi with more than two disks.

Divide the stack of disks in two parts. The largest disk (n^{th} disk) is in one part and all other $(n-1)$ disks are in the second part.

Ultimate aim is to move disk n from Source to destination and then put all other (n-1) disks onto it. It can apply the same in a recursive way for all given set of disks.

Steps to follow are-

1) Move n-1 disks from source to aux
2) Move nth disk from source to dest.
3) Move n-1 disks from aux to dest.

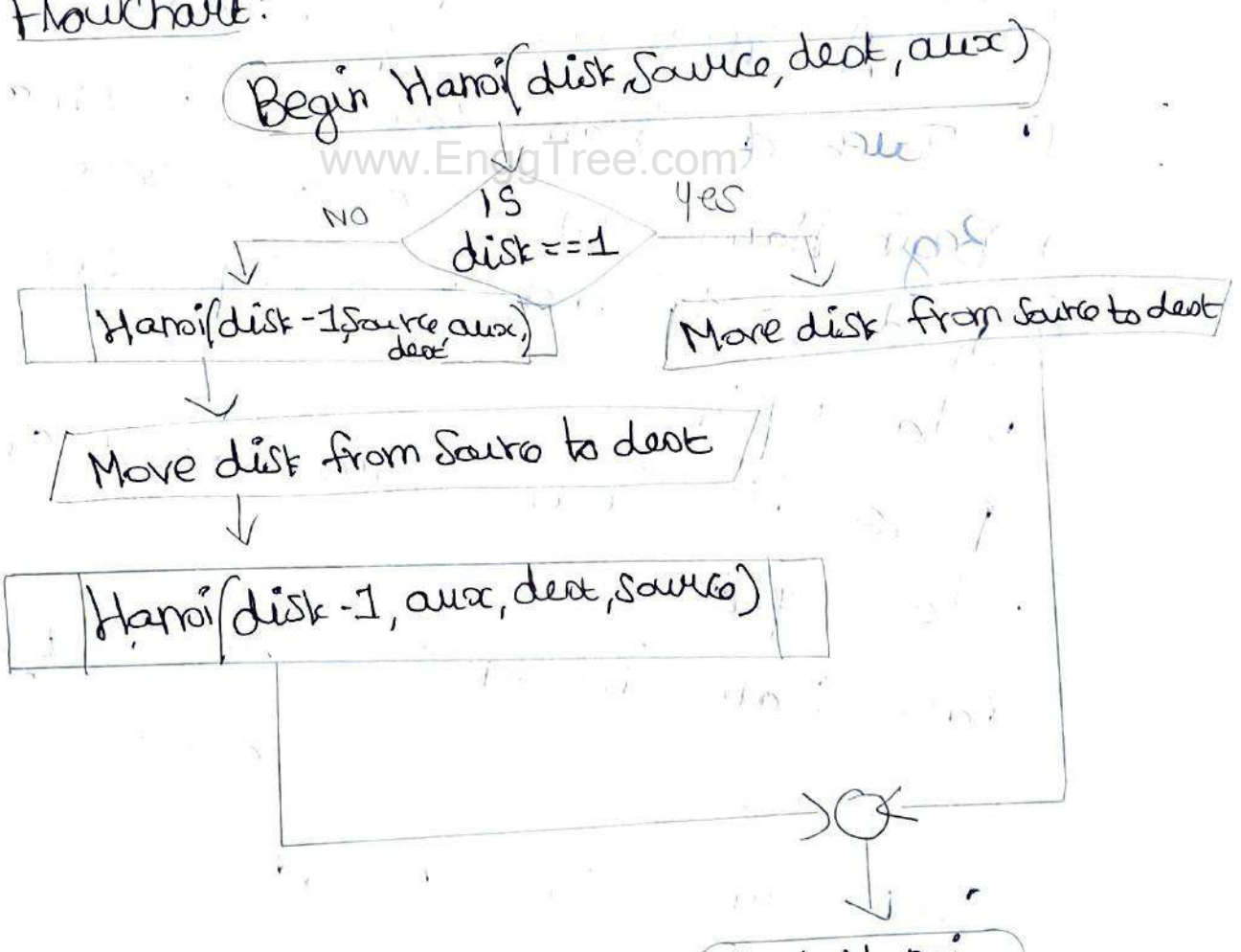
A recursive step based algorithm for Tower of Hanoi can be driven as-

1) Begin Hanoi (disk, Source, dest, aux)
2) If disk = 1 then go to Step 3 else go to step 4
3) Move disk from Source to dest and go to step 7
4) Call Hanoi (disk - 1, Source, aux, dest)
5) Move disk from Source to dest
6) Call Hanoi (disk - 1, aux, dest, Source)
7) End Hanoi

Pseudocode:

```
IF disk = 1 THEN
   move disk from source to dest
ELSE
Hanoi (disk - 1, source, aux, dest)
move disk from Source to dest
Hanoi (disk - 1, aux, dest, Source)
ENDIF
END Procedure
```

Flowchart:



23/12/24.

Tower of Hanoi puzzle with n disks can be solved in minimum $2^n - 1$ steps.

UNIT-II

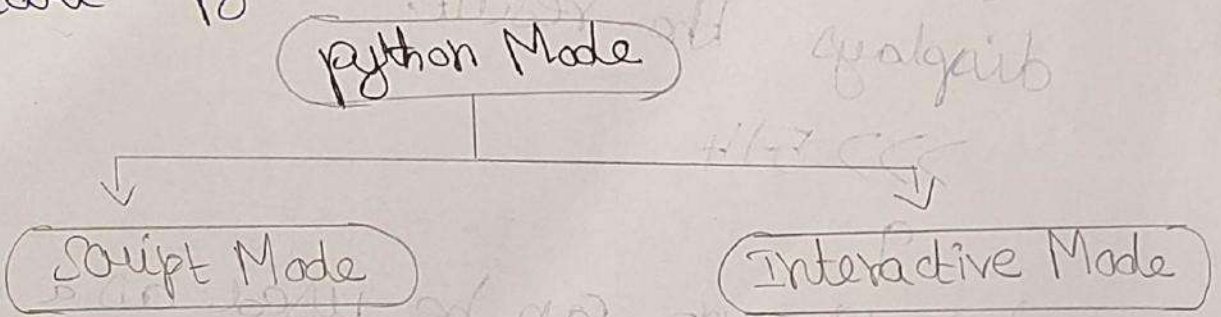
Data Types, Expressions, Statements

python interpreter and interactive Mode:

The python programming language implemented in December 1989 by Guido Van Rossum. It runs on multiple platforms like windows, Mac os X, Linux, Unix. It is a free and open Software.

Python Interpreter:

It is a program that reads and executes python code. On Linux, the python interpreter is installed as /usr/local/bin/python35. On windows machines, the python installation is placed in C:/python35. There are two ways to start python.



Interactive Mode:

It is a Command line Shell which

gives immediate feedback for each statement. When it starts, its output is,

```
Python 3.5.2 (v3.5.2:5fd33b5, Dec 4 2019, 04:14:3
[MSC v.1900 32 bit (Intel)]
on win32
Type "Copyright", "Credits" or "License()" for more inf
>>> 1+1
2
>>>
```

The first three lines contain information about the interpreter and the operating system it is running on. In this example, 3 begins with 3, which indicates that running python 3. The last line is a prompt that indicates that the interpreter is ready to enter code. If type a line of code and hit Enter, the interpreter displays the result.

```
>>> 5+4
9
```

This prompt can be used as a calculator. To exit this mode type exit() or quit() and press enter.

Script Mode:

It is also called as normal mode. It is a mode in which python commands are stored in a file and the file is saved using the extension .py.

Eg: 1) open python shell by clicking the python IDE. on File Menu click on New File option. Give some suitable file name with extension .py. A file will get opened and type some programming code. Now run code by clicking on Run on menubar. For running the script can also use F5 key.

PyScripter IDE is one of open source IDE. If using pyscripter, there is a green arrow button on top. press that button or press ctrl+F9 on keyboard to run the program.

```
print("Hello World!")
```

The output is

```
Hello World!
```

Values and types: int, float, boolean, String and list:

A Value is one of the basic things a program works with like a letter or a number. Value is stored in a Variable and it is a basic thing in python. Based on the data type different Values can be stored in a Variable. There are some standard data types in python. 5 is an integer, 83.0 is a floating-point number and "Hello world!" is a String.

1) Int:

It represents the Signed integer Value.

Eg: 100,

2) Float:

It is for representing the floating point real numbers.

Eg: -3.14, -88.33

3) Boolean:

Two new constants were added to the built in module True and False. True and False are simply set to integer values of 1 and 0 and a different type. The type object for this new type is named bool; the constructor for it takes any python value and converts it to True or False.

```
>>>bool(1)
True
>>> bool(0)
False
>>> True+1
2
>>> False+1
1
>>> False*85
0
>>> True*85
85
>>> True+True
2
>>> False+False
0
```

4) String:

The String is a collection of characters. It is presented within a quote.

Eg:

```
>>> mystr = "India"
>>> print[mystr[0]
  I
>>>Print[mystr[1:3]
  nd
```

String in python are identified as a contiguous set of characters represented in the quotation marks. python allows for either pairs of single or double quotes. Subsets of strings can be taken using the slice operator ([] and [:]) with indexes starting at 0 in the beginning of the String and working their way from -1 at the end.

The plus(+) sign is the String concatenation operator and the asterisk (*) is the repetition operator.

Eg:

```
Str = 'python Programming'
printStr
```

printstr[0]
It prints the first character of the String.
p

printstr[-1]
It prints the last character of the String
g

printstr[2:5]
It prints characters starting from 3rd to 5th
tho

printstr[2:]
It prints String starting from 3rd character
thon programming

printstr*2
It prints string two times
python programmingpython programming

printstr+"Course"
It prints concatenated String
python programming Course

5) List:-

The list is a compound data type which contain list of elements separated by comma and enclosed within the square brackets [ ]

```
>>> myList = [10, 20, 30]
>>> print (myList [0])
10
>>> print (myList [1:2])
[20]
>>> print (myList [0:2])
[10, 20]
>>>
```


Lists are the most versatile of python's compound data types. A list contains items separated by commas and enclosed within square brackets ([ ]). The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end-1. The plus (+) sign is the list concatenation operator and the asterisk (*)

is the repetition operator.

Eg:

List = ['Hai', 123, 1.75, 'Abar', 100.25]

Smallist = [251, 'Abar']

print list

['Hai', 123, 1.75, 'Abar', 100.25]

print list[0]

It print first element of the list.

Hai

print list[-1]

It print last element of the list

100.25

print list [1:3]

It prints elements starting from 2nd till 3rd.

[123, 1.75]

print list [2:]

It prints elements starting from 3rd element

[1.75, 'Abar', 100.25]

printsmall list * 2

It prints last 2 times in list

[251, 'Abar', 251, 'Abar']

print list + Smallist

It prints Concatenated lists.

['Hai', 123, 1.75, 'Abar', 100.25, 251, 'Abar']

Variables:

A Variable is a name that refers to a value. Variable reserved memory locations to store values. Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory.

Variable Names:

Programmers generally choose names for their variables that are meaningful.

Rules:

1) Variables names must start with a letter or an underscore such as

```
_mark
mark_
```

2) The remainder of Variable name may consists of letters, numbers and underscores.

```
Subject1
myzrdsubject
un_der_scores
```

3) Names are case sensitive.
Case_Sensitive, CASE_SENSITIVE & case_sensitive are each a different variable.

4) It can be any length.
5) There are some reserved keywords which cannot use as a variable name because python uses them for other things.
6) The interpreter uses keywords to recognize the structure of the program and they cannot be used as variable names.

Keywords:

python 3 has these keywords.

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

* No need to memorize this keywords. keywords are displayed in a different color; if try to use one as a variable name

If give a variable an illegal name, get a syntax error.

```
>>> 1book = 'python'
SyntaxError: invalid syntax
```

1book is illegal because it begins with a number.

```
>>> more@ = 100000
SyntaxError: invalid syntax
```

more@ is illegal because it contains an illegal character @

```
>>> class = "Fundamentals of programming'
SyntaxError: invalid syntax
```

class is illegal because it is a keyword.

Good Variable Name:

1) Choose meaningful name instead of short name.

roll_no is better than rn

2) Maintain the length of a variable name.

Roll_no_of_a_Student is too long.

3) Be consistent;

roll_no (or) RollNo

4) Begin a variable name with an underscore (_) character for a special case.

Expressions and Statements:

An expression is a combination of values, variables and operators. A value all by itself is considered an expression and is a variable. The following are all legal expressions:

```
>>> 50
50
>>> 10 < 5
False
>>> 50+20
70
```

When type an expression at the prompt, the interpreter evaluates it which means it finds the value of the expression.

A Statement is a unit of code that has an effect, like creating a variable or displaying a value.

```
>>> n=25
>>> print(n)
```

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

type a Statement, the interpreter executes it which means that it does whatever the Statement says. Statement dont have values.

```
>>>a=10
>>>a+20
30
>>> 2+3*4
14
```

The first statement in the above code is assignment Statement. The second Statement is an expression. The interpreter evaluates the expression and displays the result.

	Statement	Expression
1.	It is a complete line of code that performs some action	It is any section of code that evaluates to a value
2.	It can be combined horizontally into larger expressions using operators.	It can be combined vertically by writing one after another or with block constructs.
3.	Every expression can be used as a Statement	Most Statements cannot be used as expressions.

Tuple Assignment:

Tuple is a sequence of items of any type. It is a comma separated by list of values. It is enclosed within parenthesis. The main difference between list and tuples are: Lists are enclosed in brackets [] and their elements and size can be changed, while tuples are enclosed in parentheses () and cannot be updated. It allows tuple of variables on the left of an assignment to be assigned values from a tuple on the right of the assignment.

Eg:

```
>>>Student = ('AAA', 96, 'Std_X')
>>> print (Student)
('AAA', 96, 'Std_X')
>>>print (Student[0])
AAA
>>> print (student [1:3])
(96, 'Std_X')
```

It is to swap the values of two variables.

Eg: To swap a and b

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

This solution is cumbersome; tuple assignment is more elegant.

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

The left side is a tuple of variables, the right side is a tuple of expressions. The number of variables on the left and the number of values on the right have to be the same.

```
>>> a, b = 1, 2, 3
Value Error: too many values to unpack.
```

The right side can be any kind of sequence (string, list or tuple). To split an email address into a user name & domain

```
>>> addr = 'monty@python.org'
>>> uname, domain = addr.split('@')
```

The return value from split is a list with two elements; the first element is assigned to uname, the second to domain.

```
>>> uname
'monty'
>>> domain
'python.org'
```

Operators:

Operators are special symbols in python that carry out computation. The value that the operator operates is called the operand.

Eg: >>>10+5
15

Here + is the operator that performs addition. 10 and 5 are the operands. 15 is the output of the operation. python has a number of operators

1) Arithmetic operators:

It is used to perform mathematical operations like addition, subtraction, multiplication etc.

+ add 2 operands
- Subtract right operand from left
* multiply 2 operands
/ Divide left operand by right
% modulus - remainder of the division
// Floor division - division that result into whole number
** Exponent - left operand raised to the power of right

Eg: x = 7
y = 3
print('x+y =', x+y)
10
print('x-y =', x-y)
4
print('x*y =', x*y)
21
print('x/y =', x/y)
2.33
print('x//y =', x//y)
2
print('x%y =', x%y)
1
print('x**y =', x**y)
343

2) Comparison or Relational operators:-

It is used to compare values. It either returns True or False according to the condition.

> Greater than - True if left operand is greater than right

< Less than - True if left operand is less than the right

== Equal to - True if both operands are equal

!= Not equal to - True if operands are not equal.

>= Greater than or equal to - True if left operand is greater than or equal to the right

<= Less than or equal to - True if left operand is less than or equal to the right.

Eg: x = 5
y = 7

```
print('x>y is', x>y)
```

x>y is False

```
print('x<y is', x<y)
```

x<y is True

```
print('x==y is', x==y)
```

x==y is False

```
print('x!=y is', x!=y)
```

x!=y is True

```
print('x>=y is', x>=y)
```

x >= y is False

print('x <= y is', x <= y)

x <= y is True

3) Logical operators:

Logical operators are the and, or, not operators.

and True if both the operands are true

or True if either of the operands is true

not True if operand is false (complements)

Eg: X = True

Y = False

print('x and y is', x and y)

x and y is False

print('x or y is', x or y)

x or y is True

print('not X is', not X)

not X is False

4) Bitwise operators:

It act on operands. It operates bit by bit. Eg: 2 is 10 in binary and 7 is 111.

& Bitwise AND

| Bitwise OR

~ Bitwise NOT

^ Bitwise XOR

>> Bitwise right shift

<< Bitwise left shift

Eg: x = 10 (0000 1010)

y = 4 (0000 0100)

print('x & y =', x & y)

```
0000 1010
0000 0100
0000 0000
```

x & y

0 0 0

Alone / remaining 0

x & y = 0

print('x | y =', x | y)

```
0000 1010
0000 0100
0000 1110
```

x y x|y

0 0 0

1 1 1

Alone / remaining 1

x | y = 14 (0000 1110)

print('~x =', ~x)

0 means 1

1 means 0

~x = -11 (0000 1011) (1111 0101)

print('x ^ y =', x ^ y)

```
0000 1010
0000 0100
0000 1110
```

x y x^y

0 0 0

1 1 0

1 0 1

0 1 1

Along / remaining 1

x ^ y = 14

print('x >> 2 =', x >> 2)

0000 1010 → move out 2 places right

x >> 2 = 2 (0000 0010)

print('x << 2 =', x << 2)

0000 1010 → move out

x << 2 = 40 (0010 1000)

5) Assignment operators:

It is used in python to assign values to variables.

Eg: a = 10

It is a simple assignment operator that assigns the value 10 on the right to the variable a on the left.

Precedence of operators:

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

Eg: 20 - 5 * 3

= 20 - 15

= 5

Multiplication has higher precedence than subtraction.

But can change order using parentheses() as it has higher precedence.

```
>>> (20-5)*3
    15*3
    45
```

When more than one operator appears in an expression, the order of evaluation depends on the rules of precedence. The acronym PEMDAS is a useful way to remember the order of operations.

1) P: Parentheses have the highest precedence. Expression in parentheses is evaluated first.

Eg: = 1*(10-5)
= 1*5
= 5

2) E: Exponentiation has the next highest precedence. So, $2^3 = 2^{**}3 = 2 \times 2 \times 2 = 8$

3) MDAS: Multiplication and Division have the same precedence which is higher than Addition and Subtraction also have same precedence.

2+3*4 = 14 rather than 20

4) operators with the same precedence are evaluated from left to right.

= 3 - 2 + 1

= 2

As subtraction is performed first and then addition will be performed.

Comments:

Comments are the kind of statements that are written in program for program understanding purpose. By the comment statement it is possible to understand what exactly the program is doing. Use the hash (#) symbol to start writing a comment. It extends up to the newline character. Python interpreter ignores comment. As programs get bigger and more complicated they get more difficult to read. It is a good idea to add notes to your programs to explain in natural language what the program is doing.

Compute area of triangle using base & height

area = (base * height)/2

The comment appears on a line by itself in this case. It can also put comments at the end of a line.

area = (base * height)/2 # area of a triangle using base & height

Everything from # to the end of the line is ignored - it has no effect on execution of program. If have comments that extend multiple lines, one of way is to use hash(#) in the beginning of each line.

Eg: # This is a long comment
and it extends
to multiple lines

Another way of doing this is to use triple quotes either ''' or """. These triple quotes are used for multi-line strings. But they can be used as multi-line comment as well.

Eg """ This is also a perfect example of multi-line comments"""

Illustrative Programs: Exchange the Values of two Variables:

In python, Exchange of values of two variables (Swapping) can be done in different ways.

1) Using Third Variable:

```
Var 1 = input("Enter Value of Variable 1:")
Var 2 = input("Enter Value of Variable 2:")
temp = Var 1
Var 1 = Var 2
Var 2 = temp
print("After Swapping:")
print("First Variable =", Var 1)
print("Second Variable =", Var 2)
```

Output:

```
Enter Value of Variable 1: 5
Enter Value of Variable 2: 10
After Swapping
First Variable = 10
Second Variable = 5
```

In the above program line number 3,4,5 are used to swap the values of two variables. It use the temp variable to temporarily hold the value of var1. Then put the value of var2 in var1 and later temp in var2 In this way, the values get exchanged. In this method using one more variable other than var1, var2. So calling it as swapping with the use of third variable.

2) Using Tuple assignment method:

In this method, instead of line number 3,4,5 use a single tuple assignment statement.

```
var1, var2 = var2, var1
```

The left side is a tuple of variables; the right side is a tuple of expressions. Each value is assigned to its respective variable. All the expressions on the right side are evaluated before any of assignments

3) Using Arithmetic operators:-

If the Variables are both numbers, Swapping can be performed using Simple mathematical addition Subtraction relationship or multiplication division relationship.

```
x = x+y
y = x-y
x = x-y
```

It is for addition and subtraction.

```
X = X * y
y = x/y
X = x/y
```

It is for multiplication and division.

4) Exchange the values by using function:

```
def swap (a,b):
    temp = a
    a = b
    b = temp
    print (a)
    print (b)
```

o/p: swap (10, 20)

20
10

Circulate the Values of n Variables:

Problem of circulating a python list by an arbitrary number of items to the right or left can be done easily by list slicing operator.

1	2	3	4	5	6	7

Consider the above list; circulation of the above list by n position can be easily achieved by slicing the array into two and concatenating them. Slicing is done as n^{th} element to end element + beginning element to $n-1^{th}$ element. Suppose n=2 means, given list is rotated 2 positions towards left side

3	4	5	6	7	1	2

Left circulated list

Suppose n=-2 means given list is rotated 2 position towards right side as,

6	7	1	2	3	4	5

Right circulated list

So the Simple function to perform this circulation operation is,

```
def circulate (list, n):
return list[n:] + list[:n]
>>> circulate([1,2,3,4,5,6,7],2)
[3,4,5,6,7,1,2]
>>> circulate([1,2,3,4,5,6,7]-2)
[6,7,1,2,3,4,5]
```

Distance between Two points:

The formula for calculating distance between two points is,

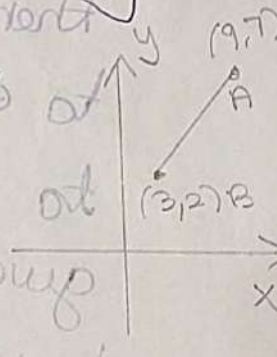
$$dist = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2}$$

$$dist = \sqrt{(9-3)^2 + (7-2)^2}$$

$$= \sqrt{6^2 + 5^2}$$

$$= \sqrt{61}$$

$$= 7.8102$$



Program:-

```
import math
print("Enter Value of X1")
X1 = int(input())
```

```
print("Enter Value of x2")
x2 = int(input())
print("Enter Value of y1")
y1 = int(input())
print("Enter Value of y2")
y2 = int(input())
dist = math.sqrt((x2-x1)**2+(y2-y1)**2)
print("The distance between two points is:",dist)
```

Output:

```
Enter Value of x1 = 9
Enter Value of x2 = 3
Enter Value of y1 = 7
Enter Value of y2 = 2
The distance between two points is
7.810249675
```

22/12/21

UNIT-III

CONTROL FLOW, FUNCTIONS, STRINGS

Conditionals:

Flow of execution of instruction can be controlled using conditional statements. Conditional statements have some boolean expression. Boolean expression can have relational operators or logical operators or both.

Boolean Values:

There are two types of boolean values - true or false. The boolean expression can be represented using the operator.

Eg: >>> 3==3
True
>>> 3==5
False

In above example, == operator is used for obtaining the boolean value of the expression. It can check the data type of True and False with the help of type function

```
>>> type(True)
<class 'bool'>
>>> type(False)
<class 'bool'>
```

Some of the valid boolean expressions are

```
>>> True
True
>>> 5 == 4
False
>>> 3+2 == 5
True
```

operators:

operands are special symbols that are used in computations.

Eg: +, -, * and / are used for performing arithmetic operations. The values that operator uses for performing computation are called operands.

Various operators used in python are,

1) Arithmetic operators:

It is used for performing arithmetic operations.

+:
Addition operator is used for performing addition of two numbers.

Eg: 10+20=30

-:
Subtraction operator is used for performing subtraction.

Eg: 20-10=10

*:
Multiplication operator is used for performing multiplication.

Eg: 10*10=100.

/:
Division operator is used for performing division of two numbers.

Eg: 10/2=5

%:
Mod operator returns the remainder value.

Eg: 10%2=0

**:
It is an exponentiation operator (power).

Eg: 2**3=8

//: It is a floor division operator. It result is the quotient in which the digits after decimal point are removed. Eg: 10//3=3

2) Comparison operators:

It Compares the values and establish the relationship among them.

==:

If two values are equal then condition becomes true

Eg: 10 == 10

!=

If two operands are not equal then the condition becomes true.

<>

It is similar to !=. It means if two values are not equal then it returns true.

<:

It is less than operator. If left operand is less than the right operator then the return value is true.

Eg: >>> 10<20
True

>:

It is greater than operator. If left operand is greater than the right operator then the return value is true.

Eg: >>> 20 > 10
True

<=:
It is less than equal to operator. If left operand is less than the right operator or equal to right operator then the return value is true.

>>> 10 <= 10
True

>=:
It is greater than equal to operator. If left operand is less than the right operator or equal to the right operator then the return value is true.

>>> 20 >= 10
True

3) Logical operators:
There are 3 types of logical operators and, or, not

and: If both the operands are true then entire expression is true. Eg: a and b
or: If either first or second operator is true. Eg: a or b

not:

If operand is false, then entire expression is true.

Eg: not a

>>>a = True
>>>b = False
>>> a and b
False
>>> a or b
True
>>> not a
False

4) Bitwise operators:

It works on the bits of given value. These bits are binary numbers ie 0 or 1.

Eg: The number 2 is 010 = a
The number 3 is 011 = b

&: It is bitwise and operator.

Eg: a & b = 010

0	0	0
0	1	0
1	0	0
1	1	1

|: It is or operator

Eg: a | b = 011

0	0	0
0	1	1
1	0	1
1	1	1

~: It is bitwise not operator.

Eg: ~a = 101

^ It is bitwise XOR operator. 0 0 0, 0 1 1, 1 0 1, 1 1 0

Eg: a xor b = 001

<< It is left shift operator.

>> It is right shift operator.

5) Assignment operators:

It is used to assign the values to variables.

=: It is an operator in which value is assigned to a variable. Eg: a = 5

+=: Eg: a += 5 means a = a+5

-=: Eg: a -= 5 means a = a-5

Similarly *=, /= operators are used for performing arithmetic multiplication and division operation.

6) Membership operators:

There are two types of membership operators - in and not in. It is used to find out whether a value is a member of a sequence such as string or list.

in: It returns True if a sequence with specified value is present in object.

```
>>> Color = ["red", "blue", "green"]
>>> print ("blue" in Color)
```

True

not in: It returns true if a sequence with specified value is not present in the object.

```
>>> print("yellow" in Color)
False
>>> print("yellow" not in Color)
True
```

7) Identity operators:

The is operator returns true if both the operand point to same memory location. Similarly is not returns true if both the operand point to different memory location.

```
>>> Color1 = ["red", "blue", "green"]
>>> Color2 = ["red", "blue", "green"]
>>> Color3 = Color1
>>> print(Color1 is Color2)
False
>>> print(Color1 is Color3)
True
>>> print(Color1 is not Color3)
True
```

8) Modulus operator:

The '%' operator is a modulo operator that gives the remainder from the division of first argument by second.

```
>>> 10%3
1
>>> 10.10%3.3
0.2000
```

The operator // is used for floor division. It returns the integral part of the quotient.

```
>>> 10//3.5
2.0
```

9) String operators:

String is collection of characters. It is possible to perform concatenation and repetition operations using the operators like + and *

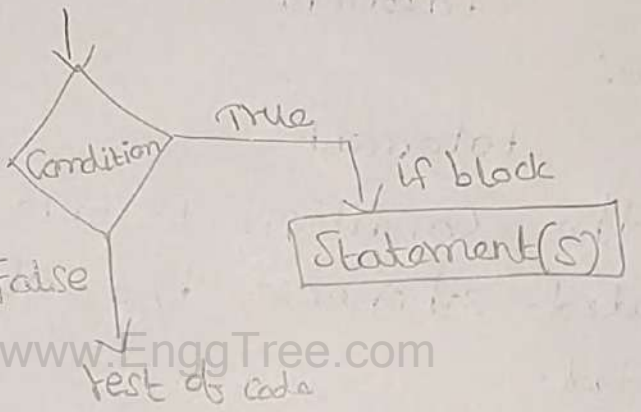
```
>>> Str1 = "Hello"
>>> Str2 = "friend"
>>> Str1 + Str2
"Hellofriend"
>>> "welcome" * 2
"welcome welcome"
```

Conditional Statements:

There are various types of Conditional statements.

1) If Statement:

It is used to test particular condition. If the condition is true then it executes the block of statements is called as if block.



Syntaxc:

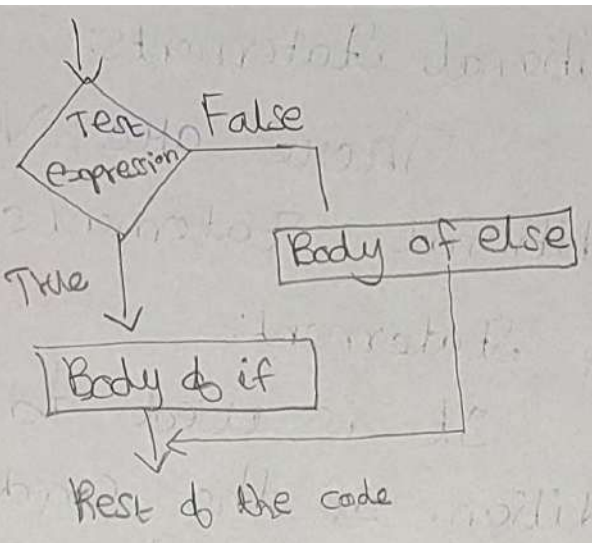
```
if Condition:
    Statement
```

Eg:

```
if a < 10:
    print("The number is less than 10")
```

2) Alternative Statements:

The if-else statement provides an else block combined with if statement which is executed in false case condition.



Syntax:

```
if Condition:
    Statement
else:
    Statement
```

If the Condition is true, then if-block is executed. otherwise, the else-block is executed.

Eg:

```
print("Enter Value of n")
n = int(input())
if n%2==0;
    print("Even Number")
else:
    print("odd Number")
```

output:

```
Enter Value of n: 7
odd Number
```

3) Chained Conditionals:

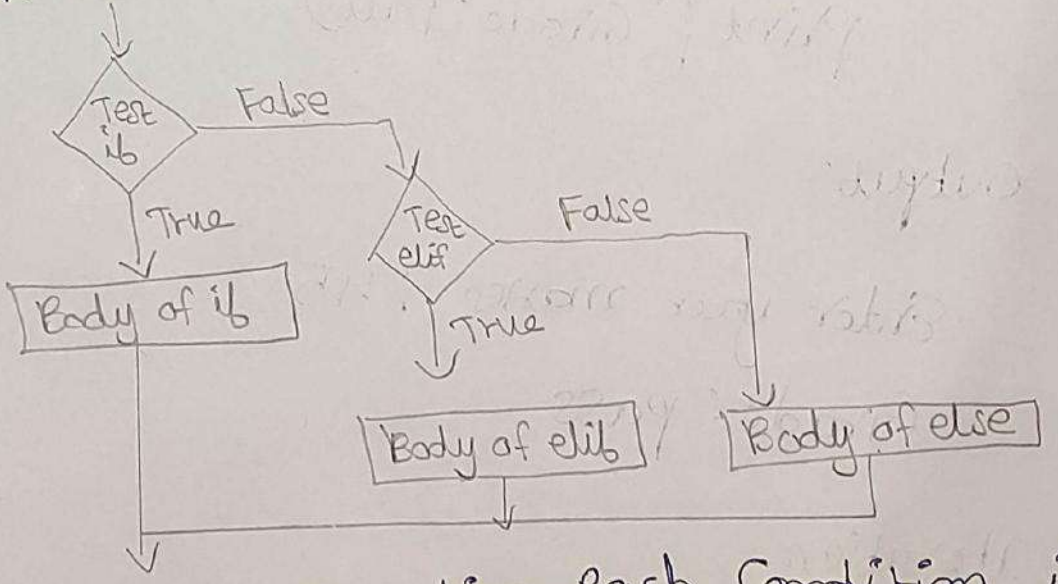
Sometimes there are more than

two possibilities. These possibilities can be expressed using chained conditionals.

Syntax:

```
if Condition:
   Statement
elif Condition:
   Statement
...
else:
   Statement
```

The chained conditional execution will be that each condition is checked in order. The elif is basically abbreviation of else if. There is no limit on the number of elif statements. If there is else clause then it should be at the end.



In chained execution, each condition is checked in order and if one of condition is true then corresponding branch runs and then

Statement ends. In this case if there are any remaining condition then those condition wont be tested.

Eg:

```
print ("Enter your marks")
m=int (input())
if m>=75;
    print("Grade: Distinction")
elif m >=60;
    print("Grade: First class")
elif m>=50;
    print ("Grade: Second class")
elif m >= 40;
    print ("Grade: pass")
else;
    print ("Grade: Fail")
```

Output:

```
Enter your marks : 45
Grade: pass
```

Iteration:

It is a technique that allows to execute a block of statements repeatedly.

Repeated execution of a set of statements is called iteration. The programming constructs used for iteration are while, for, break, continue and pass.

1) State:

The simplest form of statement is assignment statement. It is specified using = operator.

```
>>> x=10
>>> print(x)
10
```

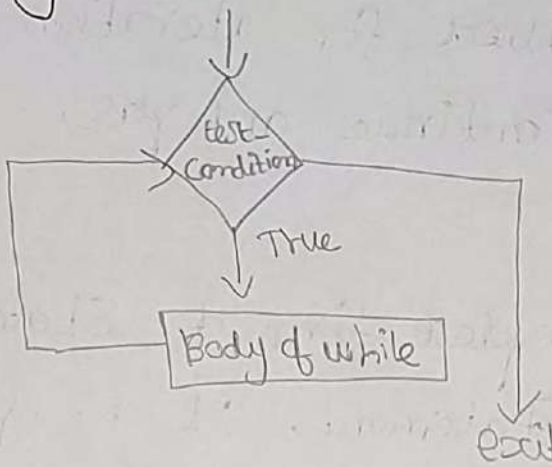
Reassigning variables is useful but should use it with caution. If values of variables change frequently, it make code difficult to read and debug. Similarly can update the values by using operators.

```
>>> a = 10
>>> a = a+1
>>> a
11
```

2) While:

The while statement is popularly used for representing iteration.

While test_condition:
    body of while



The flow of execution is specified as-
i) Using the condition determine if given expression is true or false.
ii) If the expression is false then exit the while statement.
iii) If the expression is true then execute body of while and goback to step 1 in which again the condition is checked.


```
while i<=10;
    i=i+1
```

The body of while contains the statement which will change the value of variable used in test condition. Hence finally after performing definite number of iterations,

the test condition gets false and the control exists the while loop. If the condition never gets false, then the while body executes for infinite times. Such while loop is called infinite loop. There is another version of while statement in which else is used.

```
while test_condition:
    body of while
else:
    statement
```

Eg:


```
while i<=10:
    i = i+1
else:
    print("Invalid value of i")
```

3) for:

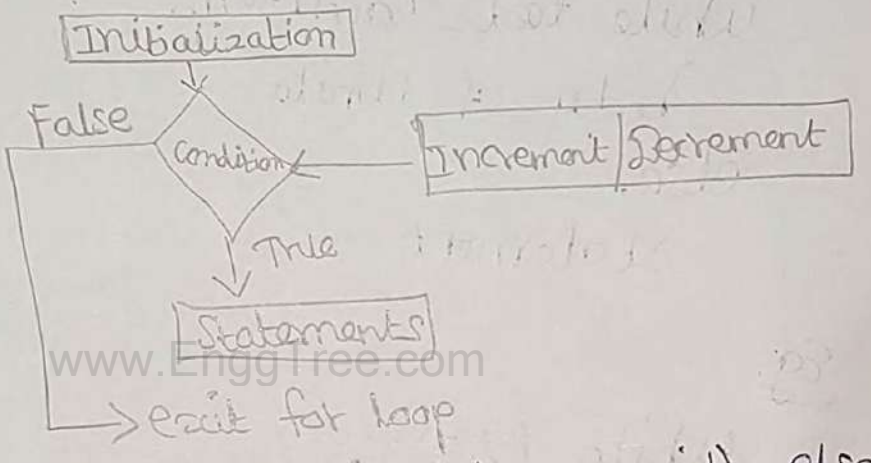
The for loop is another popular way of using iteration. The syntax of for loop is,

```
for variable in sequence:
    body of for loop
```

The variable takes the value of item inside

the sequence on each iteration. Loop continues until reach the last item in sequence. The body of for loop is separated from rest of the code using indentation.

Eg: for val in numbers:
    val = val+1



Similarly can have for loop with else statement.

```
for variable in sequence:
    body of for loop
else:
    Statement
```

Eg:

```
for i in myList:
    print(i)
else:
    print("the number is not present in list")
```

4) break:

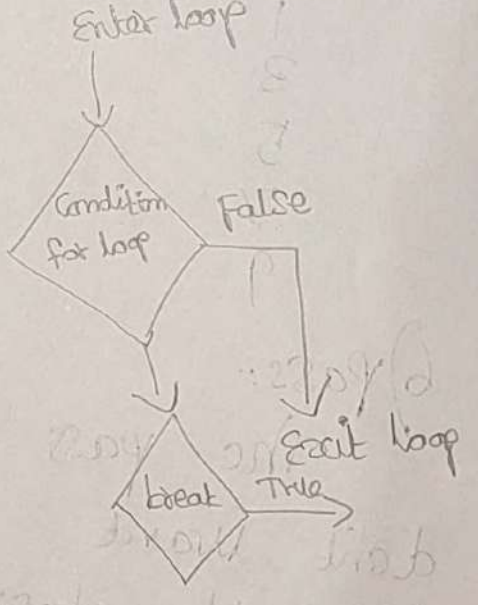
It is used to transfer the control to the end of the loop. When break statement is applied then loop gets terminated and the control goes to the next line pointing after loop body.

Syntax:

```
break
```

Eg:

```
for i in range(1,11):
    if i==5:
    print("Element {} found!!!".format(i))
      break
    print(i)
```



Output:

```
1
2
3
4
Element 5 found!!!
```

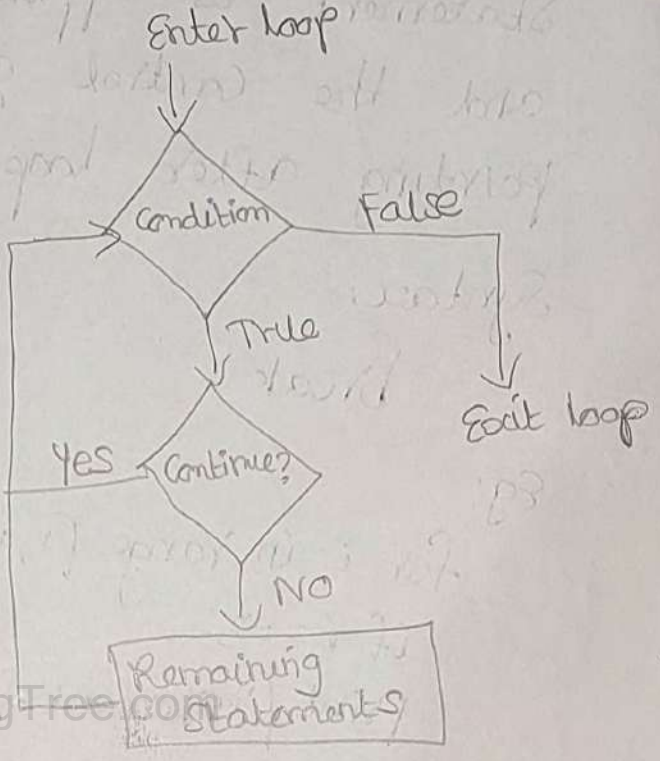
5) Continue:

It is used to skip some statements inside the loop. It is used with decision

making statements such as if..else. It forces to execute the next iteration of the loop to execute.

Syntax:

```
Continue
```



Eg:

```
i=0
while i<10;
  i=i+1
if i%2==0:
  Continue
  print(i)
```

Output:

```
1
3
5
7
9
```

6) Pass:

The pass statement is used when dont want to execute any statement. Thus the desired statements can be bypassed.

Syntax:

```
pass
```

Eg:

```
for i in range(1,5):
    if i==3:
        pass
        print("Reached at pass statement")
print("The current number is", i)
```

output:

```
The current number is 1
The current number is 2
Reached at pass Statement
The current number is 3
The current number is 4
```


Fruitful Functions:

There are two types of functions.

i) The functions that return some value.

ii) The functions that does not return the value.

The fruitful functions are the functions that return values.

1) Return values:

The value can be returned from a function using the keyword return.

Syntax:

return [expression_list]

Eg:

Write a function that returns area of circle.

```
def area(r):
    result = 3.14*r**2
    print("The area of circle")
    return (result)
```

output:

```
area(10)
```

The area of circle = 314.0

2) Parameters:

It can pass different number of parameters to the function.

Eg: Write a python program using function to find GCD of two numbers.

```
def gcd(a,b):
    while(b):
    a,b = b, a%b
    return a
num1 = int(input("Enter first number:"))
num2 = int(input("Enter second number:"))
```

print("The G.C.D of", num1, "and", num2, "is",
gcd(num1, num2))

output:

Enter first number: 12
Enter second number: 15
The G.C.D of 12 and 15 is 3

3) Local and Global Scope:

The global variables are those variables that are declared and defined outside the function.

The local variables are those variables that are declared and defined inside a function.

A global variable is one that can be accessed anywhere. A local variable is opposite, it can only be accessed within its frame. The difference between the global and local is that global variables can be accessed locally but not modified locally inherently.

Eg: def fun():
print(a)
a = 10
fun()

output:
10

Eg:

```
def f():
    print(a)
    a=100  // error is raised
a=10
f()
print(a)
```

Add global keyword to change variable.

```
def f():
    global a
    print(a)
    a=100
a=10
f()
print(a)
```

output:
10
100

4) Function composition:

It is a way of combining functions such that the result of each function is passed as argument of next functio

Eg: The composition of two functions f and g is denoted f(g(x)). X is the argument of g, the result of g is passed as the argument of f, and the result of the composition is the result of f.

Eg: Create a simple function for addition of two numbers.

```
def add(a,b):
    return a+b
```

Create a simple function for multiplication of two numbers.

```
def mul(c, num):
    return c*num
```

Create a main function in which the two functions used in above two steps are called.

```
def mainfun(x,y):
    Z = add(x,y)
    result = mul(Z,10)
    return result
```

```
def add(a,b):
    return a+b
def mul(c,num):
    return c*num
def mainFun(x,y):
    z=add(x,y)
    result=mul(z,10)
    return result
```

output:

```
MainFun(10,20)
300
```


Recursion:

Recursion is a property in which one function calls itself repeatedly in which the values of function parameter get changed on each call.

Properties of Recursion:

1) A recursive function must have a base case.
2) A recursive function must change its state and move towards base case.

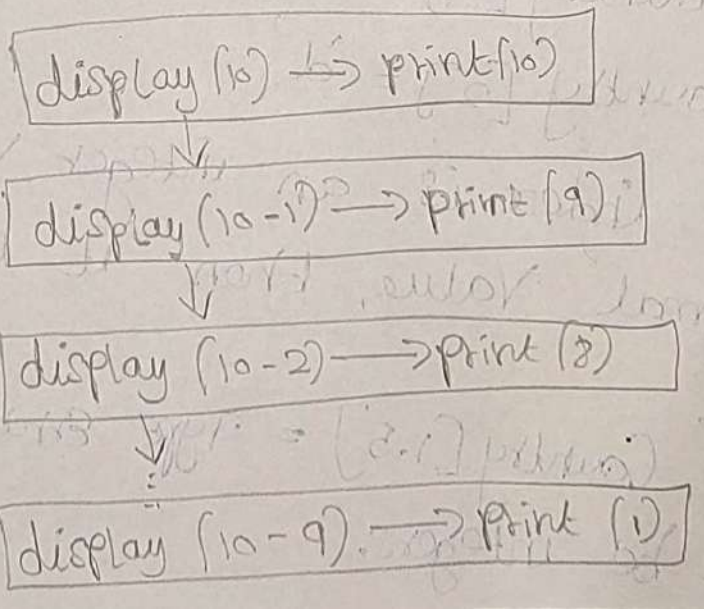
3) A recursive function must call itself, recursively.

Eg:
```
def display(n):
    if n<=0:
        return
    else:
        print(n)
        display(n-1)
```

output:

display(10)

10
9
8
7
6
5
4
3
2
1



Eg:

```
def factorial (n):
    if n==0:
        return 1
    else:
        result = n* factorial (n-1)
        return result
print(factorial (5))
```

output:

120

Strings:

String is basically the seaquence of characters. Any desired character can be accessed using indexc.

Eg: Country = "India"

Country [1] = 'n'

Country [2] = 'd'

The index is an integer value, if it is a decimal value, then it will raise an error.

Country [1.5] = Type Error: String indices must be integers.

The String can be created using double quote or single quotes.

Eg: msg = "Hello"
print(msg)
Hello

msg = 'Goodbye'
print(msg)
Goodbye

Finding Length of a String:-

It is an in-built function to find length of the String. It is len function.

Eg: msg = 'Goodbye'
print(msg)
Goodbye
len(msg)
7

1) String Slices:-

String Slice is an extracted chunk of characters from the original String. It can obtain the String slice with the help of string indices.

Eg: msg = "Good Morning"

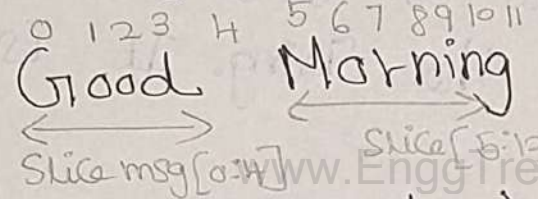
msg[0:4]

'Good'

msg[5:12]

'Morning'

The String from 0 to less than 4 index will be displayed. In the next command the String from 5th index to 11th index is displayed.



It can omit the beginning index. The beginning index is considered as 0.

Eg: msg[:4]

'Good'

Similarly, can omit ending index. The String will be displayed upto its ending character.

Eg: msg[5:]

'Morning'

If didn't specify any starting index or ending index then the String from starting

index 0 to ending index as last character will be considered and the entire String will be displayed.

Eg: msg[:]

'Good Morning'

2) String Immutability:

Strings are immutable ie cannot change the existing Strings.

Eg: msg = "Good Morning"

msg[0] = 'g'

output:

TypeError: 'str' object doesn't support item assignment.

To make the desired changes need to take new String and manipulate it as per our requirement.

```
msg = 'Good Morning'
new_msg = 'g' + msg[1:]
Print (new_msg)
```

output:

good Morning

In the above example the new_msg String is created to display "good morning"

instead of "Good Morning". The String slice from character 1 to end of String is concatenated with the character 'g'. The concatenation is performed using the operator +.

3) String Functions and Methods:

There are various string functions and methods.

i) String Concatenation:

Joining of two or more strings is called concatenation. In python use + operator for concatenation of two strings.

Eg:
```
msg1 = "Good"
msg2 = "Morning"
print(msg1+msg2)
```

output:

Good Morning

ii) String Comparison:

It can be done using the

relational operators like <, >, =.

Eg: msg1 = "aaa"
msg2 = "aaa"
msg1 == msg2
True
msg1 = "aaa"
msg2 = "bbb"
print (msg1 < msg2)
True

The string comparison is made based on alphabetical ordering. All the upper case letters appear before all the lower case letters.

iii) String Repetition:

It can repeat the string using * operator.

Eg: msg = "welcome!"
print (msg * 3)
welcome! welcome! welcome!

iv) Membership Test:

The membership of particular

character is determined using the keyword in.

Eg: msg = "welcome"

```
'm' in msg
True
't' in msg
False
```

4) String Module:

It contains number of constants and functions to process the strings. To use the string module in python program need to import it at the beginning.

i) capwords:

It is a function that converts first letter of the string into capital letter.

Syntax:

```
String.capwords(string)
```

Eg:

```
import string
Str = 'i love python'
print(Str)
print(Str.capwords(Str))
```

Output

i love python

I Love Python

ii) upper function and lower Case:

For converting the string into upper case letter use str.upper() function instead of string.upper. Similarly str.lower() function is for converting the string into lower case.

Eg:

```
import String
text1 = 'i love programming'
text 2 = 'PROGIRAMMING IN PYTHON'
print("original String:", text1)
print("String in Upper Case:", str.upper(text1))
print("original String:", text2)
print("String in Lower Case:", str.lower(text2))
```

output:

original String: i love programming

String in Upper Case: I LOVE PROGRAMMING

original String: PROGIRAMMING IN PYTHON

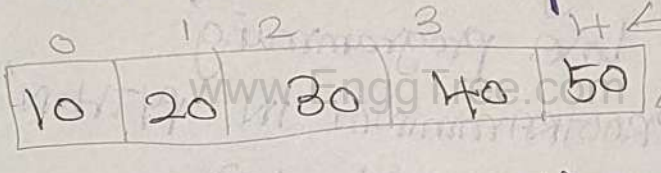
String in Lower Case: programming in python

List as arrays:-

The array is a data structure in which elements are of same data type. A list in python is an ordered collection of items which can be of any type. The elements in array are separated by comma and are enclosed within square bracket.

Eg: arr = [10, 20, 30, 40, 50]

The arr can be represented by,



The values are arranged sequentially as,

arr[0] = 10
arr[1] = 20
arr[2] = 30
arr[3] = 40
arr[4] = 50

1) Creation of Arrays:-

It can create an array using

the array name and list of elements.

Eg:

```
arr = [10,20,30,40]
```

will create an array containing the elements 10,20,---40. It can be represented using for loop.

Eg:

```
arr = [10,20,30,40]
print("The elements in array are..")
for i in range(len(arr)):
    print(arr[i])
```

output:

The elements in array are

```
10
20
30
40
```


2) operations on Arrays:

i) Appending a value:

It can add the element in the array at the end.

Eg:

```
arr=[10,20,30,40]
```

```
print("The elements in array are..")
for i in range(len(arr)):
    print(arr[i])
arr.append(50)
print("Now the elements in array are...")
for i in range(len(arr)):
    print(arr[i])
```

output:

```
The elements in array are..
10
20
30
40
Now the elements in array are...
10
20
30
40
50
```

Thus the value 50 is appended in array.

ii) Inserting the element in the list:-

It can insert the value at any desired location using insert() function.

Syntax:-

```
insert(index, value)
```

Eg: arr = [10, 20, 30, 40]

```
print("The elements in array are...")
for i in range(len(arr)):
    print(arr[i])
arr.insert(2,25)
print("Now the elements in array are...")
for i in range(len(arr)):
    print(arr[i])
```

output:

The elements in array are...

10
20
30
40

Now the elements in array are..

10
20
25
30
40

(ii) Extending the array:

It can extend one array by joining another array to it. For that purpose the extend() function is used.

Syntax:

```
extend(new_array)
```

Eg: arr = [10,20,30,40]

```
print("The elements in array are...")
for i in range(len(arr)):
    print(arr[i])
new_arr = [50,60,70]
arr.extend(new_arr)
for i in range(len(arr)):
    print(arr[i])
```

output:

```
The elements in array are...
10
20
30
40
10
20
30
40
50
60
70
```


iv) Removing the element from the array:

Any desired element can be deleted from array using remove() method.

Syntax:

```
remove(index_of_element)
```

Eg:

```
arr = [10,20,30,40]
print("The elements in array are...")
for i in range(len(arr)):
    print(arr[i])
arr.remove(30)
print("Now the elements in array are...")
for i in range(len(arr)):
    print(arr[i])
```

output:

```
The elements in array are...
10
20
30
40
Now the elements in array are...
10
20
40
```


v) Removing last element from array:

For removing the last element from the array then pop() function is used.

Syntax:

```
pop()
```

Eg:

```
arr=[10,20,30,40]
```

```
print("The elements in array are..")
for i in range(len(arr)):
 print(arr[i])
arr.pop()
print("Now the elements in array are..")
for i in range(len(arr)):
 print(arr[i])
```

output:

The elements in array are..

10
20
30
40

Now the elements in array are..

10
20
30

vi) Reversing the elements of array:

It can reverse the contents of the array using reverse() function.

Eg:

```
arr=[10,20,30,40]
print("The elements in array are..")
for i in range(len(arr)):
 print(arr[i])
```

```
arr.reverse()
print("Now the elements in array are..")
for i in range(len(arr)):
 print(arr[i])
```

output:

The elements in array are..

10
20
30
40

Now the elements in array are..

40
30
20
10

vii) Counting the occurence of element in array

It can count the number of times the particular element appears in the array using the count method.

Eg:

```
arr=[10,20,30,40,50,20,30,20]
print("The elements in array are..")
for i in range(len(arr)):
 print(arr[i])
print("The element 20 appears for",arr.count(20),"times in array")
```

output:

The elements in array are..

10
20
30
40
50
20
30
20

The element 20 appears for 3 times in array.

Illustrative programs:

1. Square Root:

It is used for obtaining the Square root of a given number.

Eg:

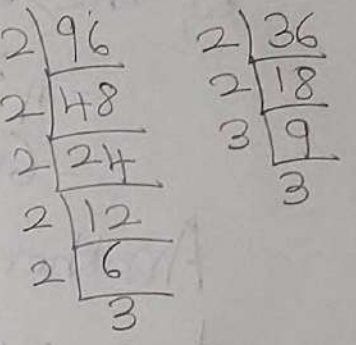
```
print("Enter the number:")
num = float(input())
sqrt_num = num ** 0.5
print("The square root of", num, "is", sqrt_num)
```

output:

Enter the number: 25
The square root of 25 is 5

2) GCD:

The GCD is the Greatest Common Divisor is a largest integer that can exactly divide both numbers without a remainder.



Eg: 96 = 2X2X2X2X2X3

36 = 2X2X3X3

GCD = 2X2X3

= 12

Hence GCD of 96 and 36 is 12.

Eg:

```
print("Enter first number:")
a=int(input())
print("Enter second number:")
b=int(input())
rem=a%b
while rem!=0:
    a=b
    b=rem
    rem=a%b
print("gcd of given number is:",b)
```

output:

```
Enter first number:96
Enter second number:36
GCD of given number is 12
```

3) Exponentiation:-

The Simplest way to find the exponentiation of a number x^y in python is, the use of ** operator.

```
=4**3
=64
```

Another way to find the exponentiation of a number x^y in python is the use of pow() function that is available in math module.

```
import math
math.pow(4,3)
=64
```

Eg:

```
def expo(base, degree):
    result = 1
    i = 1
    while i <= degree:
        result = base * result
        i += 1
    print("Result is", result)
expo(2,3)
```

output:

```
Result is 8
```

4) Sum of Numbers:-

For sum of numbers have to store the numbers in an array. By traversing the elements of array each number is added with each other. The resultant sum is then printed.

Eg: Consider 5 numbers stored in an array as,

0	1	2	3	4
10	20	30	40	50

Sum = (10+20+30+40+50)

= 150

Eg:

```
print("Enter total number of elements in array")
n=int(input())
i=0
sum=0
a=[i for i in range(n)]
for i in range(0,n):
    print("Enter the element")
    a[i]=int(input())
print("the elements in array are:")
for i in range(0,n):
    print(a[i])
```

```
for i in range (0,n):
    sum = sum + a[i]
print ("The sum of all elements in array", sum)
```

output:-

```
Enter total number of elements in array:5
Enter the element:10
Enter the element:20
Enter the element:30
Enter the element:40
Enter the element:50
The elements in array are:-
10
20
30
40
50
The sum of all elements in array 150.
```


5) Linear Search:-

In linear search method, the key element is compared against every element of the array. If the key element matches with the array element then declare element is found otherwise the element is declared as not found.

Eg:
```
print("enter total number of elements in array")
n=int(input())
i=0
a=[i for i in range(n)]
for i in range(0,n):
    print("Enter the element:")
    a[i]=int(input())
print("The elements in array are..")
for i in range(0,n):
    print(a[i])
print("Enter the key element to be searched")
key=int(input())
for i in range(0,n):
    if a[i]==key:
        found=True
        break
    else:
        found=False
if found==True:
    print("the element is found")
else:
    print("the element is not found")
```

Output:

Enter the total number of elements in array: 5
Enter the element: 10
Enter the element: 20
Enter the element: 30
Enter the element: 40
Enter the element: 50
The elements in array are...
10
20
30
40
50

Enter the key element to be searched: 40
The element is found.

6) Binary Search:

Search a sorted array by repeatedly dividing the search interval in half. Begin with an interval covering the whole array. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half. otherwise narrow it to upper half. Repeatedly check until the value is

found or the interval is empty.

1) Compare x with the middle element.

2) If x matches with middle element, return the mid index.

3) Else if x is greater than mid element, then x can only lie in right half, Subarray after mid element, So recur for right half.

4) Else (x is smaller) recur for left half.

It is an efficient searching technique.

Eg:

```
def binary_search(a,n,key):
    low=0
    high=n
while(low<=high):
  mid=int((low+high)/2)
if(key==a[mid]):
  return mid
elif(key<a[mid]):
  high=mid-1
else:
  low=mid+1
return -1
n=int(input("Enter size of list"))
a=[i for i in range(n+1)]
for i in range(0,n):
```

```
print("Enter the element:")
a[i]= int(input())
print("Enter the element to be searched:")
k=int(input())
position= binary_search(a,n,k)
if(position != -1):
print("Entered number {} is present at position
                {}".format(k,position))

else:
print("Entered number {} is not present in list
                ".format(k))
```

output:

```
Enter the size of list:5
Enter the element:10
Enter the element:20
Enter the element:30
Enter the element:40
Enter the element:50
Enter the element to be searched:40
Entered number 40 is present at position:3
Enter the element to be searched:90
Entered number 90 is not present in list.
```

Explanation:

The prerequisite for this searching technique is that array should be sorted.

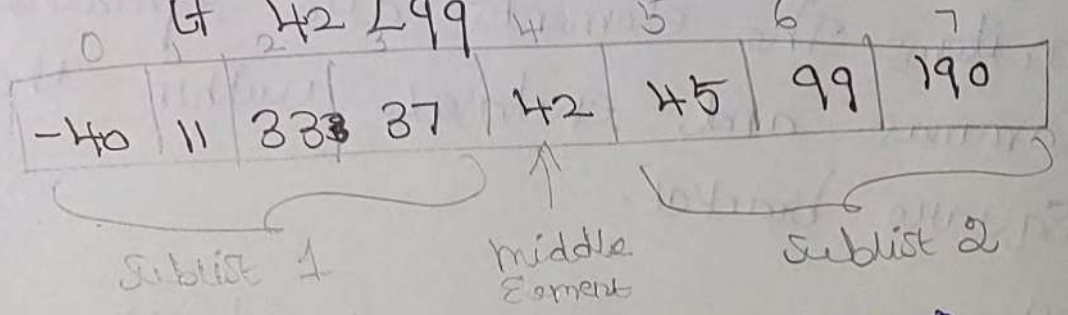
The necessity of this method is that all elements should be sorted.

0	1	2	3	4	5	6	7
40	-11	33	37	42	45	99	190

Now the key element is to be searched is = 99. key = 99. Find the middle element of the array. Compare it with key.

if middle < key

If 42 < 99

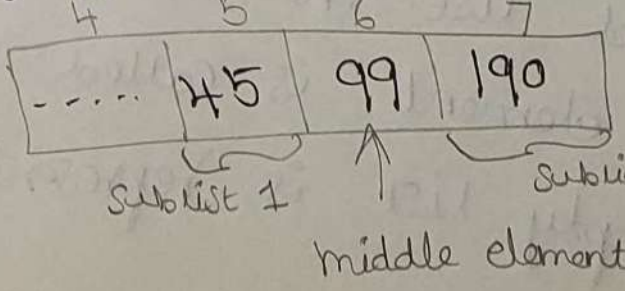


if 42 < 99 search the sublist 2.

4	5	6	7
...	45	99	190

key = 99

Middle element is 99 and key is also 99. Declare that the element is found and it is at index 6.



Unit-IV

Lists, Tuples, Dictionaries

Lists:-

List is a Sequence of Values. String is also Sequence of Values. But string is a sequence of characters. List can be of any type. The Values in List are called elements or items. The elements are separated by commas and enclosed within the square bracket.

Eg:

[10, 20, 30, 40] - List of integers

['aaa', 'bbb', 'ccc'] - List of strings

['a', 10, 20, 'b', 33.33] - mixed list

[10, 20, ['a', 'b', 'c']] - nested list

The list within another list is called nested list. The list that contains no element. is called empty list. The empty list is represented by [].

List operations:-

There are 2 operations that can be

performed using operators such as + and * [3]

1) Concatenation using +:

The list can be created and can be joined using + operator.

Eg: >>>[10,20,30]

[10,20,30]

```
>>>L1 = [10,20,30]
>>>L2 = [40,50,60]
>>>L = L1+L2
>>> L
[10,20,30,40,50,60]
```


2) Repetition using *:

The * is used to repeat the list for number of times.

```
>>> [10] * 5
[10,10,10,10,10]
>>> [10,20,30] * 3
[10,20,30,10,20,30,10,20,30]
```

List slices::

The : operator is used within

square bracket that it is a list slice and not the index of list.

Eg: >>> a=[10,20,30,40,50,60]

```
>>> a[1:4]
[20,30,40]
>>> a[:5]
[10,20,30,40,50]
>>> a[4:]
[50,60]
>>> a[:]
[10,20,30,40,50,60]
```

If omit the first index then list is considered from the beginning. If omit the last second index then slice goes to end.

If omit both the first and second index then list will be displayed from the beginning to end.

Lists are mutable. It can change the elements of the list.

Eg: >>> a = [10,20,30,40,50]

```
>>> a[2:4] = [111,222,333]
>>> a
[10,20,111,222,333,50]
```

List Methods:-

1) Append ():

It is used to add the element at the last position of the list.

Syntax:

```
list.append(element)
```

Eg:

```
>>> a=[10,20,30]
>>> a.append(40)
>>> a
[10,20,30,40]
>>> b=['A','B','C']
>>> b.append('D')
>>> b
['A','B','C','D']
```

2) Extend ():

The extend function takes the list as an argument and appends the list at end of old list.

Syntax:

```
List1.extend(List2)
```

where List2 is added at end of List1.

Eg: >>> a = [10,20,30]
>>> b = ['a','b','c']
>>> a.extend(b)
>>> a
[10, 20, 30, 'a', 'b', 'c']

3) insert():

It allows to insert the desired element at specified position.

Syntax:

List.insert(position, element)

where the first parameter is a position at which element is to be inserted. The element to be inserted is the second parameter in this function.

Eg: a = [10,20,30]
a.insert(2,25)
print("List a = ",a)

output:

List a = [10,20,25,30]

4) pop():

It removes the element specified by the index.

Syntax:

List.pop(index)

The index represent the location of element in list and the element specified by index will be deleted and returned. If no parameter is passed, the default index -1 is passed as an argument which returns the last element.

Eg: a = [10, 20, 30, 40, 50]

```
ret_val = a.pop(2)
print("The deleted element is =", ret_val)
print("The modified list is =", a)
```

output:

The deleted element is = 30

The modified list is = [10, 20, 40, 50]

5) remove:

The remove method deletes the element which is passed as parameter. It actually removes the first matching element.

Syntax:

List.remove(element)

Eg:

```
>>> a = [10, 20, 30, 20, 40, 50]
>>> a.remove(20)
>>> print(a)
[10, 30, 20, 40, 50]
```


6) erase():

This method erases all the elements of the list. After this operation list becomes empty.

Syntax:

List.clear()

Eg:

```
>>> a = [10, 20, 30, 40, 50]
>>> a.clear()
>>> print(a)
[ ]
```

7) del():

This method deletes all the elements within the given range.

Syntax:

List.del(a:b)

The element within the range a to b gets deleted by this method.

Eg:

```
>>> a = ['a', 'b', 'c', 'd', 'e']
>>> del a[2:4]
>>> a
['a', 'b', 'e']
```


8) sort():

This method sort or arranges the elements in increasing order.

Syntax:

List.sort()

Eg:

```
>>> a=['x','z','u','v','y','w']
>>> a.sort()
>>> a
['u','v','w','x','y','z']
```

9) reverse():

This method is used for reversing the elements in the list.

Syntax:

List.reverse()

Eg:

```
>>> a=[10,20,30,40,50]
>>> a.reverse()
>>>print("List after reversing is :",a)
```

output:

List after reversing is :[50,40,30,20,10]

10) count():

This method returns a number for how many times the particular element appears in the list.

Syntax:

List.count(element)

Eg:

```
>>>a=[10,20,30,30,10,40,50]
>>>print("Count for element 10 is:",a.count(10))
```

output:

Count for element 10 is:3

List Loop:

The loop is used in list for traversing purpose. The for loop is used to traverse the list elements.

Syntax:

```
for VARIABLE in LIST:
    BODY
```

Eg:

```
>>>a=['a','b','c','d','e']
>>> for i in a:
    print(i)
```

output:

```
a
b
c
d
e
```


To update the elements of list, then pass index as argument to for loop.

Eg:

```
>>> a=[10,20,30,40]
>>> for i in range(len(a)):
        a[i]=a[i]+1
>>>a
[11,21,31,41]
```

If specify the [] empty list then the body of for loop will never get executed.

Eg: >>> for i in []:
    print("The line will never execute")

Mutability:-

Strings are immutable. It can not modify the strings. But List are mutable. It is possible to change the values of list. If the bracket operator is present on left side, then element is identified and list element is modified.

Eg: >>> a = ['AAA', 'BBB', 'CCC']
>>> a[1] = 'XXX'
>>> a
['AAA', 'XXX', 'CCC']

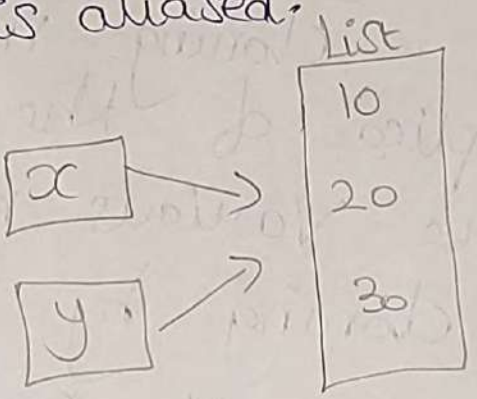
Using the in operator can check whether particular element belongs to the list or not. If the given element is present in list it returns True otherwise False.

Eg: >>> a = ['AAA', 'XXX', 'CCC']
>>> 'XXX' in a
True
>>> 'BBB' in a
False

Aliasing:

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

Eg: x = [10, 20, 30]




```
y = x
y is x
True
y
[10,20,30]
```

The association of a variable with object is called reference. Reference y is created using object a. If any change is made in one reference then other object gets affected.

Eg: x = [10, 20, 30]

```
y = x
y
[10,20,30]
y[0] = 111
y
[111,20,30]
x
[111,20,30]
```

If change the reference list y then list x also gets changed automatically.

Cloning Lists:

Cloning means creating exact replica of the original list. There are various methods using clone.

1) Cloning by assignment:

It can assign one list to another new list. It creates a new reference to the original list.

```
a=['a','b','c']
b=a
b
['a','b','c']
```

List b is a clone of original list a. But as the reference is created change in one list causes change in another list.

2) Cloning by Slicing:

It can create a clone by slicing. During this process, [:] operator

is used for creating the clone.

Eg: a=[10,20,30]

b = a[:]

print("List a =", a)

List a = [10, 20, 30]

print("List b = ", b)

List b = [10,20,30]

b[1] = 222

print("List b = ", b)

List b = [10, 222, 30]

print("List a =", a)

List a = [10, 20, 30]

clone b is created from list a. But change in one list does not affect the other list. The operator [:] means copy the elements in one list from beginning to end into another list.

3) clone by copy constructor:-

It can create a clone for the list using copy constructor. It is similar to clone by slicing. The list function

is used to create the clone.

Eg: a = ['A', 'B', 'C']

b = list(a)

b

['A', 'B', 'C']

a

['A', 'B', 'C']

If changes in clone list, the other list doesnot get changed.

Eg: a

['A', 'B', 'C']

b[1] = 'z'

b

['A', 'z', 'C']

a

['A', 'B', 'C']

List parameters:

A list can be passed as a parameter to the function. The parameter is passed by reference. Any change made in list inside the function will affect the list even after returning the function to the main.

Eg:

```
def Insert_End(a):
    a.append(40)

a=[10,20,30]
Insert_End(a)
a
[10,20,30,40]
```

The function Insert_End is for inserting the element at the end of list. Here pass the list a as parameter to the function. Inside the function, the element is added at the end of list. Hence, after the making call to function, list in which the element is inserted at the end.

Tuples:

Tuple is a sequence of values. It is similar to list but there lies difference between tuple and list.

Tuple is said to immutable. It means once created, can not change the tuple.

Eg: T1 = (10, 20, 30, 40)

T2 = ('a', 'b', 'c', 'd')

T3 = ('A', 10, 20)

T4 = ('aaa', 'bbb', 30, 40)

The empty tuple can be created as, T1 = ()

Tuple is written within the parenthesis. Even if tuple contains a single value, the comma is used as a separator.

Eg: T = (10,)

The tuple index starts at 0.

Eg: t1 = (10, 20, 'AAA', 'BBB')

```
print (t1[0])
10
Print (t1[1:3])
(20, 'AAA')
```

Tuple Assignment:

It can create a tuple by using assignment operator. Multiple assignments are possible at a time using tuple assignment.

Eg: a, b = 10, 20
print(a)
10
print(b)
20

The left side is a tuple of variables; the right side is a tuple of expressions. Each value is assigned to its respective variable. All the expressions on the right side are evaluated before any of the assignments. The number of variables on left and right have to be same. It also separate out the values in expression in the tuple by using the split function.

Eg: Student = 'AAA,123'

```
name, roll = Student.split(',')
print(name)
AAA
print(roll)
123
```

The expression is split at comma and the two separated values are collected in two different variables. It have name and roll variables in which the corresponding values are stored.

Tuple as return value:

Function returns a single value, but if the value is tuple, then multiple values can be returned.

Eg:

```
def student_Info():
    name = 'AAA'
    roll = 101
    return name, roll
['AAA', 101]
```

Eg: t= Student_Info()

```
print (t[0])
AAA
print (t[1])
101
t
('AAA', 101)
```

Dictionaries:-

In python, dictionary is unordered collection of items. These items are in the form of key-value pairs. The dictionary contains the collection of indices called keys and collection of values. Each key is associated with a single value. The association of keys with values is called key-value pair or item. Dictionary always represent the mappings of keys with values. Thus each key maps to a value.

Create Dictionary:-

Items of the dictionary are written within the {} brackets and are separated by commas. The key value pair is represented using : operator. That is key:value. The keys are unique and are of immutable types - such as string, number, tuple. It can also create a dictionary using the word dict()

Eg:
my_dictionary = dict({0:'Red', 1:'Green', 2:Blue

Access elements in dictionary:-

It can access the elements in dictionary using the keys.

Eg: Student_dict = {'name': AAA, 'roll': 10}
print(student_dict['name'])
print(student_dict['roll'])
AAA
10

operations:-

Various operations that can be

performed on dictionary are

1) Adding item to dictionary:

It can add the item to the dictionary.

Eg: my_dictionary = dict({0:'Red', 1:'Green', 2:'Blue'})

```
print(my_dictionary)
{0:'Red', 1:'Green', 2:'Blue'}
my_dictionary[3] = 'yellow'
print(my_dictionary)
{0:'Red', 1:'Green', 2:'Blue', 3:'yellow'}
```

2) Removing item from dictionary:

For removing an item from dictionary use the keyword del.

Eg:

```
del my_dictionary[2]
print(my_dictionary)
{0:'Red', 1:'Green', 3:'yellow'}
```

3) Updating the Value of dictionary:

It can update the value of the dictionary by directly assigning the value to corresponding key position.

Eg:

```
my_dictionary=dict({0:'Red',1:'Green',2:'Blue'})
print(my_dictionary)
{0:'Red',1:'Green',2:'Blue'}
my_dictionary[1]='yellow'
print(my_dictionary)
{0:'Red',1:'yellow',2:'Blue'}
```

4) Checking Length:

The len() function give the number of pairs in the dictionary.

Eg:

```
my_dictionary=dict({0:'Red',1:'Green',2:'Blue'})
len(my_dictionary)
3
```

Methods in dictionary:

Following are some commonly used methods in dictionary.

1) clear Method:

This method removed all the items from the dictionary. It does not take any parameter and does not return anything.

Eg:

```
my_dictionary={1:'AAA',2:'BBB',3:'CCC'}
print(my_dictionary)
```

{1: 'AAA', 2: 'BBB', 3: 'CCC'}
my_dictionary.clear()
print(my_dictionary)
{}

2) Copy Method:

It returns the copy of the dictionary. It does not take any parameter and returns a shallow copy of dictionary.

Eg:

my_dictionary={1:'AAA', 2:'BBB', 3:'CCC'}
print(my_dictionary)
{1: 'AAA', 2: 'BBB', 3: 'CCC'}
new_dictionary = my_dictionary.copy()
print(new_dictionary)
{1: 'AAA', 2: 'BBB', 3: 'CCC'}

3) Fromkey method:

It creates a new dictionary from the given sequence of elements with a value provided by the user.

Syntax:

dictionary.fromkeys(sequence [, value])

It returns a new dictionary with the given sequence of elements as the keys of the dictionary. If the value argument is set, each element of the newly created dictionary is set to the provided value.

Eg: keys = {10, 20, 30}

values = 'Number'

new_dict = dict.fromkeys(keys, values)

print(new_dict)

{10: 'Number', 20: 'Number', 30: 'Number'}

4) Get Method:

The get() method returns the value for the specified key if key is in dictionary. It takes 2 parameters key and value. It can return either key or value or nothing.

Syntax:

dictionary.get(key [, value])

Eg:

```
Student={'name':'AAA','roll':10,'marks':98}
print("Name:", Student.get('name'))
Name:AAA
print("roll:", Student.get('roll'))
roll:10
print("marks:", Student.get('marks'))
marks:98
print("Address:", Student.get('address'))
Address:None
```

5) Value Method:

It returns the value object that returns view object that displays the list of values present in the dictionary.

Eg:

```
my_dictionary={marks1:'99',marks2:96,marks3
print(my_dictionary.values())
dict_values([99,96,97])
```

6) Pop Method:

It removes and returns an element from a dictionary having the

given key.

Syntax:

pop (key [, default])

where key is the key which is searched for removing the value. Default is the value which is to be returned when the key is not in dictionary.

Eg: my_dictionary={1:'Red', 2:'Blue', 3:'Green'}

Val=my_dictionary.pop(1)

Print("The popped element is:", Val)

The popped element is: Red

Val=my_dictionary.pop(5,3)

When the specified key is not present in list, then default value is returned.

Print("The popped element using default Value is:", Val)

The popped element using default Value is: 3

Advanced List processing - List Comprehension

Python supports a concept called

"List comprehensions". It can be used to construct lists in a very natural, easy way, like a mathematician is used to do. In python, can write the expression almost exactly like a mathematics with special cryptic syntax.

```
L=[x**2 for x in range(10)]
L
[0,1,4,9,16,25,36,49,64,81]
```

The above code creates a list of square numbers from 0 to 10. This list can be created using a simplified expression "x**2 for x in range(10)". This is basically list comprehension. It apply an arbitrary expression to items in an iteration. The list comprehension is achieved using the tools like map, filter and lambda.

1) Map:-

The map() function applies the values to every member of the list and returns the result. The map function takes two arguments.

result = map(function, sequence)

Eg:

```
def increment_by_three(x):
    return x+3
new_list = list(map(increment_by_three, range(10)))
new_list
[3,4,5,6,7,8,9,10,11,12]
```


There is a built in function for calculating length of the string ie len(). It can pass this function to map and find the length of every element present in list as list.

```
names = ['Aparna', 'satya', 'Jeremiah', 'karthik']
lengths = list(map(len, names))
lengths
[6, 5, 8, 7]
```

2) Filter:

It selects some of the elements and filters out others. If use if with list comprehension, it is almost equivalent to the filter built-in. It takes two arguments.

result = filter(function, sequence)

Eg:

```
def even_fun(x):
    return x%2==0
r = filter(even_fun,[1,2,3,4,5])
list(r)
[2,4]
```


Using filter() function the odd numbers are filtered out and only even numbers are displayed. filter() works in the similar manner like map() It applies function to all the elements of the list.

3) Lambda:

It is a simple inline function that is used to evaluate the expression using the given list of arguments.

Syntax:

Lambda argument_list: expression

where the argument_list consists of a common separated list of arguments and the expression is an arithmetic expression using these arguments.

Eg: Sum = Lambda x,y: x+y

Sum (1,3)

4

Thus the expression x+y is evaluated and the result of evaluation is returned.

4) Reduce:

The kind of function that combines sequence of elements into a single value is called reduce function. It also takes two arguments.

Result = reduce (function, sequence)

The function reduce() had been dropped from the core of python when migrating to python 3. To import functools to be capable of using reduce.

Eg: import functools

functools.reduce(lambda x,y: x+y,[1,2,3,4])

10

Illustrative programs:

Simple Sorting:

1) Selection Sort:

Scan the array to find its smallest element and swap it with the first element. Then, starting with the second element scan the entire list to find the smallest element and swap it with second element. Then starting from the third element the entire list is scanned in order to find the next smallest element. Continuing in this can sort the entire list.

Eg: Consider the elements 70, 30, 20, 50, 60, 10, 40

70	30	20	50	60	10	40
A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]

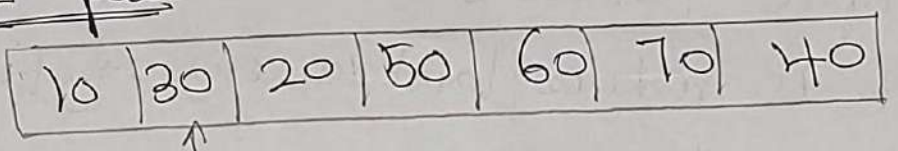
1st pass:

70	30	20	50	60	10	40
↑ i					↑ Smallest element found	

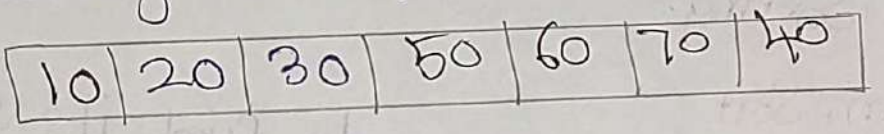
Now swap A[i] with smallest. Then get,

10	30	20	50	60	70	40

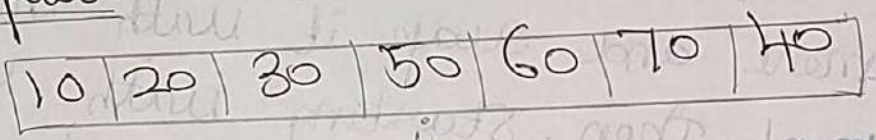
2nd pass:



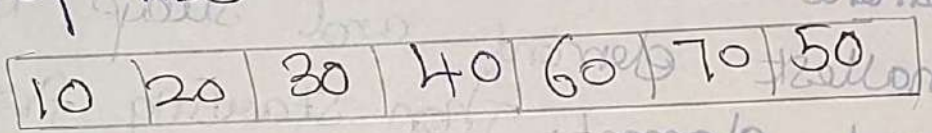
Swap A[i] with smallest element. The array becomes,



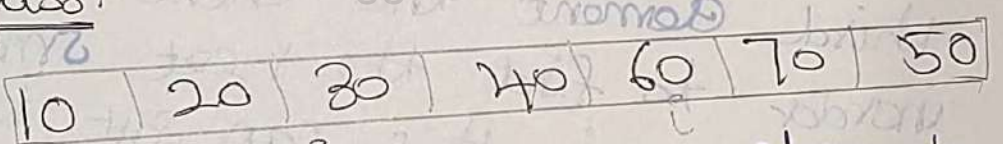
3rd pass:



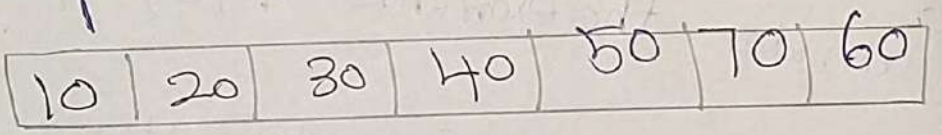
Swap A[i] with smallest element.



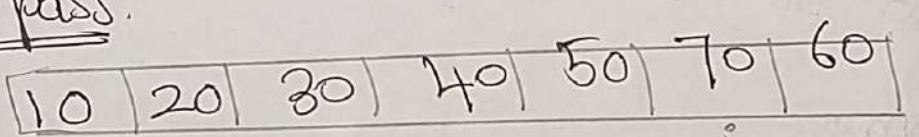
4th pass:



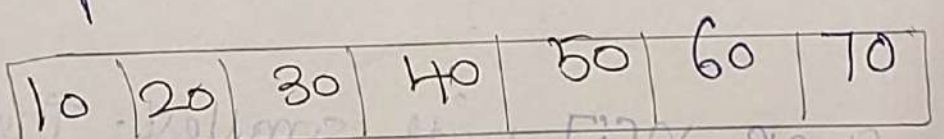
Swap A[i] with smallest element



5th pass:



Swap A[i] with smallest element



This is a sorted array.

```
def SelectionSort(a):
    for i in range(len(a)):
        least = i
    for k in range(i+1, len(a)):
        if a[k] < a[least]:
            least = k
    temp = a[least]
    a[least] = a[i]
    a[i] = temp
        a = [50,30,10,20,40,70,60]
print("original List is...")
print(a)
Selectionsort(a)
print("The Sorted List is...")
print(a)
```

Output:

original List is... 50,30,10,20,40,70,60

The Sorted List is... 10,20,30,40,50,60,70

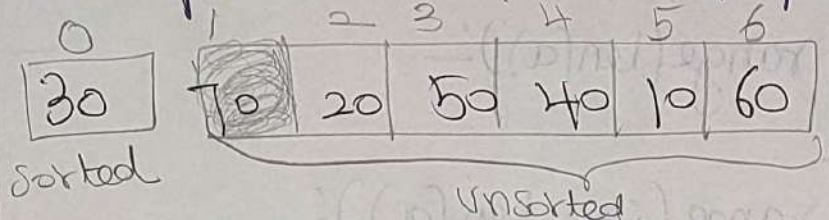
2) Insertion Sort:

In this method the elements are inserted at their appropriate place.

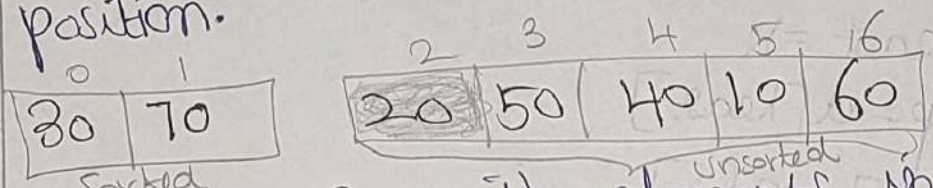
Eg: Consider a list of elements as,

30	70	20	50	40	10	60

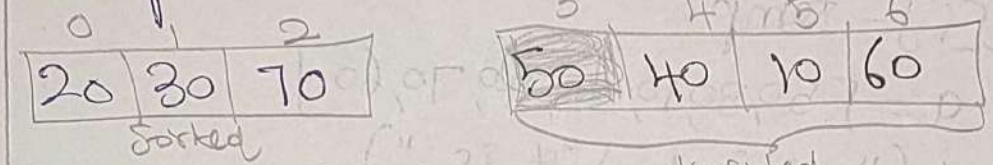
The process starts with first element

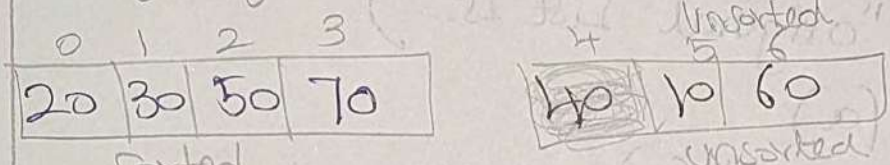


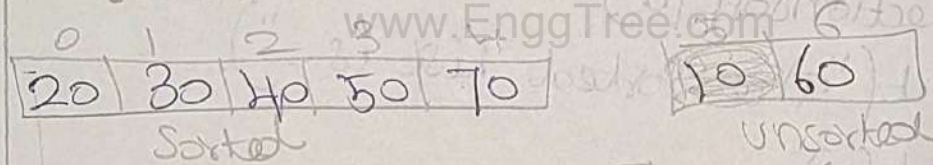
Compare 70 and 30 and insert it at its position.

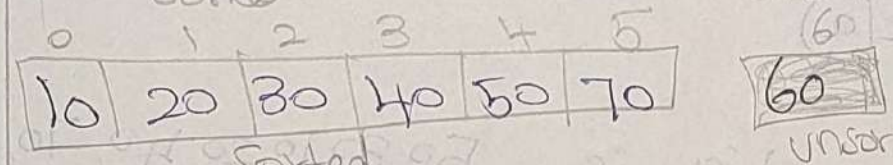


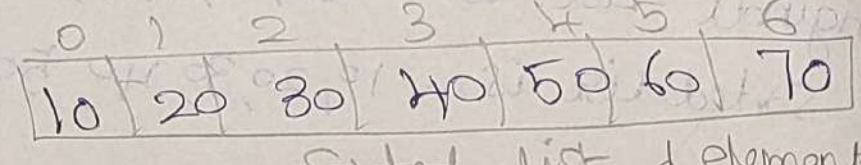
Compare 20 with elements in sorted zone












```
def insertionSort(a):
    for i in range(1, len(a)):
        tmp = a[i]
        k = i
        while k > 0 and tmp < a[k-1]:
            a[k] = a[k-1]
```

```
k-=1
a[k]=tmp
a=[50,30,10,20,40,70,60]
print("original List is...")
print(a)
insertionSort(a)
print("The Sorted List is...")
print(a)
```

output:

Original List is 50,30,10,20,40,70,60

The Sorted List is 10,20,30,40,50,60,70

3) Merge Sort:

Merge Sort on an input array with n elements consists of 3 steps.

i) partition array into two sub lists S_1 and S_2 with n/2 elements each.

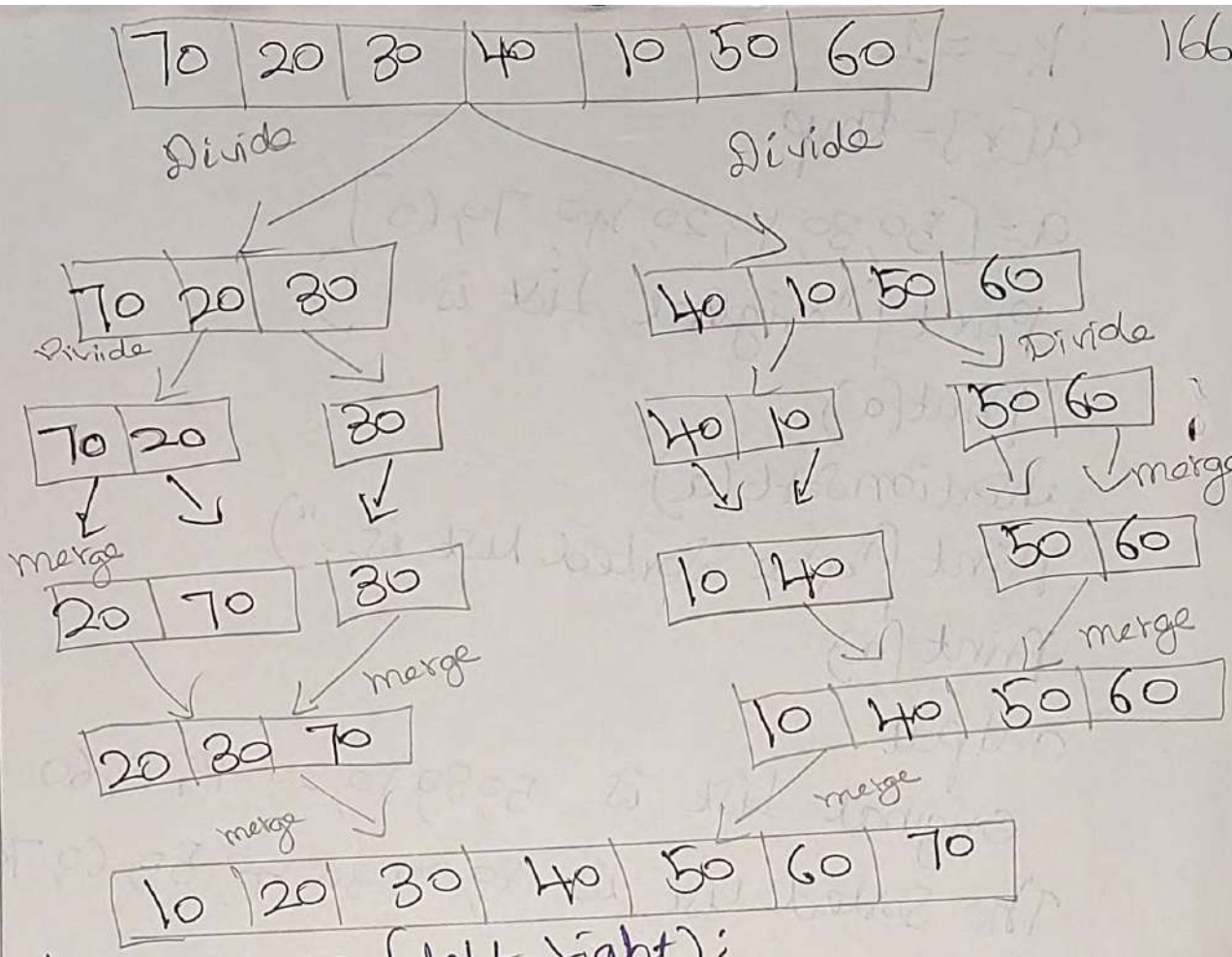
ii) Then Sort Sub list S_1 and Sub list S_2

iii) Merge S_1 and Sublist S_2 into a unique sorted group.

Eg: Consider elements as,

70,20,30,40,10,50,60

Split the list into two Sublists.




```
def merge(left, right):
    if not len(left) or not len(right):
        return left or right
    result = []
    i, j = 0, 0
    while(len(result) < len(left) + len(right)):
        if left[i] < right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
        if i == len(left) or j == len(right):
            result.extend(left[i:] or right[j:])
            break
```

```
return result
def mergesort(a):
    if len(a) < 2:
        return a
middle = int(len(a)/2)
left = mergesort(a[:middle])
right = mergesort(a[:middle:])
return merge(left, right)
a = [30, 50, 10, 40, 20]
print("Before Sorting")
print(a)
print("After Sorting")
print(mergesort(a))
```

output:

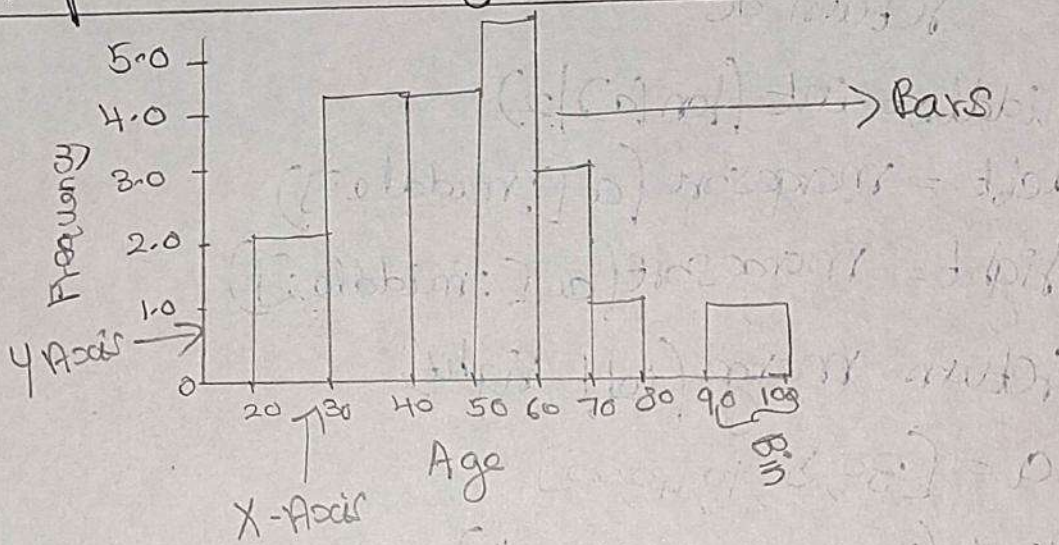
```
Before Sorting...
[50, 30, 10, 40, 20]
After Sorting...
[10, 20, 30, 40, 50]
```

Histogram:

Histogram is visual representation of the distribution of quantitative variable.

It is appeared as vertical bar graph.
It represents the distribution of variable.

Representation of Histogram:



Parts of Histogram:

Title:-

The title gives a brief description of what the information is represented by the histogram.

Horizontal or X-Axis:

It shows the range or the values of gaps in between variables. They are commonly called class intervals which represent or summarize large data sets.

Vertical or Y-Axis:

It represent the range values of frequencies or number of times the class

intervals occured.

Bars:

The bars represent the object, item, event or person from where the variables arise. Their height denotes their respective frequencies, while the bar placement along the X-axis indicates their respective interval values. The width is the same for all bars.

Bin:

Histograms use bins to display data - where a bin represent a given range of values.

It is very simple to plot the histogram in python, but for that need to install few libraries - namely numpy, scipy and matplotlib.

Step 1: For installing numpy, open the command prompt. import numpy

Step 2: For installing scipy, open the command prompt. import scipy

Step 3: For installing matlib, open the command prompt. import matplotlib

Program for Histogram:

Step 1: The histogram is drawn from Gaussian distribution. This is a normal distribution. For plotting numbers use,

gaussian_numbers = normal (size = 1000)

Step 2: The normal distribution function can be obtained using, from numpy.random import normal

The hist() method is used to generate histogram. The numbers that are used for generating histogram are passed to hist() method as parameter.

Step 3: Similarly there are title, Xlabel and Ylabel functions for giving title, label to x-axis and y-axis.

Step 4: Finally use show method for displaying the histogram.

Step 5: At the beginning import matplotli

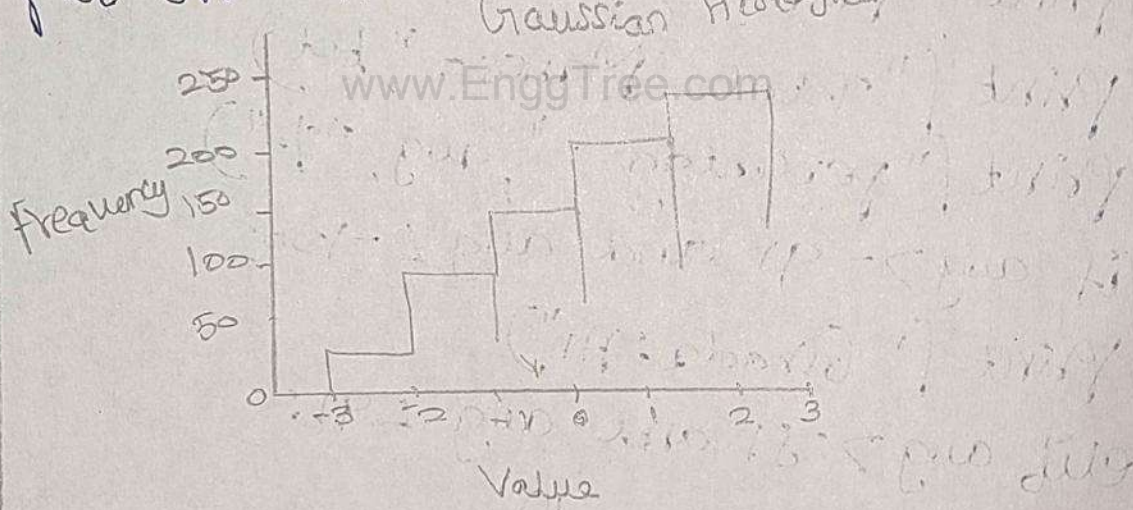
import matplotlib.pyplot as plt

open the New file & write the program

for plotting the histogram.

histogram.py

```
import matplotlib.pyplot as plt
from numpy.random import normal
gaussian_numbers = normal(size=1000)
plt.hist(gaussian_numbers)
plt.title("Gaussian Histogram")
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.show()
```



Students Marks Statement:

It contains the roll number, name, marks and grade of the student.

```
print("Enter roll number of student")
roll = int(input())
print("Enter name of student")
```

```
name = input()
mark = []
tot = 0
print("Enter marks obtained in 5 subjects:")
for i in range(5):
    mark.insert(i, input())
for i in range(5):
    tot = tot + int(mark[i])
avg = tot/5
print("Roll Number:", roll)
print("Name:", name)
print("Total Marks =" tot)
print("percentage =", avg, "%")
if avg >= 91 and avg <=100:
print("Grade: A1")
elif avg >=81 and avg <91:
print("Grade: A2")
elif avg >=71 and avg <81:
print("Grade: B1")
elif avg >= 61 and avg <71:
print("Grade: B2")
elif avg >=51 and avg <61:
    print("Grade: C1")
```

```
elif avg >= 41 and avg < 51:
    print("Grade: C2")
elif avg >= 33 and avg < 41:
    print("Grade: D")
elif avg >= 21 and avg < 33:
    print("Grade: E1")
elif avg >= 0 and avg < 21:
    print("Grade: E2")
else:
    print("Invalid Input")
```

output:


```
Enter roll number of student: 101
Enter name of student: AAA
Enter marks obtained in 5 subjects:
77
66
44
33
68
Roll Number: 101
Name: AAA
Total Marks = 288
Percentage = 57.6%
Grade: C1
```

Retail Bill preparation:


```
Product_name = []
product_ quantity = []
product_price = []
Company_ name = 'Student Store'
Company_address = 'Grace'
company_city = 'Tuticorin'
message = 'Thanks for shopping with us today!'
for i in range(3):
product_name.append(input('Enter name of product:'))
product_quantity.append(int(input("Enter quantity
                                  of Product:')))
product_price.append(int(input("Enter price of product:")))
print("BILL")
print('\t\t{}'.format(company_name.title()))
print("\t\t{}".format(company_address.title()))
print("\t\t{}".format(company_city.title()))
print('\tproduct Name\tQuantity\tprice')
i=0
for i in range(3):
print(product_name[i], product_quantity[i], product_
                                  price[i])
print('total')
total =0
i=0
for i in range(3):
  total += (product_price[i]*product_quantity[i])
print('{}'.format(total))
```

print('{}'.format(message))

output:

Enter name of the product: pen
Enter the quantity of product: 20
Enter the price of product: 20
Enter name of the product: pencil
Enter the quantity of product: 10
Enter the price of product: 10
Enter name of product: Eraser
Enter the quantity of Product: 10
Enter the price of product: 5

BILL
Student Store
Grace
Tuticorin

Product Name	Quantity	price
pen	20	20
pencil	10	10
Eraser	10	5

Total : Rs.550

Thanks for shopping with us today!

UNIT-V

Files, Modules, packages

Files:

Files is a named location on the disk to store information. It is used to store the information permanently.

Types of Files:

There are 2 types of files.

1) Text File:

The text ASCII file is a simple file containing collection of characters. Various operations that can be performed on text files are, opening the file, reading the file, writing to the file, appending data to file.

2) Binary File:

It is a file which contains data encoded in binary form. It can be images, sound, spreadsheets, videos. It cannot read using text editor like notepad.

Text Files:

It is the type of file that store textual information. Various operations are,

1) opening a File:

In python, there is a built in function open() to open a file.

Syntax:

File_object = open(file_name, mode)

where File_object is used to handle to the file.

Eg:

F = open("test.txt")

file named test.txt is opened and file object is in variable F.

There are various mode of file in which it is opened.

Mode	purpose
r	open file for reading
w	open file for writing. If file is not created then create new file and then write. If file is already existing then truncate it.

x	open a file for creation only. If file already exist, the operation fails
a	open file for appending mode. It is a mode in which data is inserted at end of existing text. A new file is created if it doesn't exist.
t	open file in text mode
b	open file in binary mode
+	open file for updation ie reading and writing.

Eg:
fo = open("test.txt", w) → write mode
fo = open("test.txt", rt) → reading in text mode

2) closing a File:

After performing all operations, it is necessary to close the file. closing of file is necessary because it frees all the resources that are associated with file.

Eg: fo = open("test.txt", rt)
fo.close()

3) Reading from Files:

For reading file, open the file in r mode and is used for reading the file is read() method.

Eg:
```
inf = open("D:\\test.txt", 'r')
print(inf.read())
inf.close()
```

Readline() method:

It allows to read a single line from file. When file reaches to end, it returns an empty string.

Eg:
```
inf = open('D:\\test.txt', 'r')
print(inf.readline())
inf.close()
```

Readlines() method:

It is used to print all the lines in the program.

Eg:
```
inf = open('D:\\test.txt', 'r')
print(inf.readlines())
inf.close()
```

4) Writing to Files:

For writing contents to file, must open it in writing mode. Use 'w' or 'a' or

as a file mode. It is used to write the contents to the file.

Syntax:

File_object.write(contents)

Eg:

```
fo = open('D:\\test.txt', 'wt')
fo.write("Grace clg of Engg")
fo.close()
```

writelines() method:

It is used to write list of strings to the file.

Eg:

```
fo = open('D:\\test.txt', 'wt')
lines = ["welcome to pspp\n", "It is easy\n", "It is language"]
fo.writelines(lines)
fo.close()
```

b) Appending the files:

It means inserting records at the end of the file. It must open the file in 'a' mode.

Eg:

```
fo = open('D:\\test.txt', 'a')
fo.write("python has widescope in future")
fo.close()
```

Format operator:

It is specified using %: operator. To display integer values then format sequence is %d. To display string values then format sequence is %s

Eg: 'There are %d colors in a rainbo' % 7
'There are 7 colors in a rainbow'

'I like %s color' % 'blue'
I like blue color

'There are %d letters in the word %s' % (4, 'blue')
'There are 4 letters in the word blue'

'The value of %s is %f' % ('pi', 3.14)
'The value of pi is 3.14'

'There are %d %d %d numbers' % (1,2)
TypeError

'There are %d cows' % 'three'
TypeError: %d format: a number is required, not str

Command Line Arguments:

The sys module is used to use the command line arguments. There are 3 important steps to be followed while accessing the command line arguments.

1) Import the sys module
2) It can use sys.argv for getting list of command line arguments.
3) The len(sys.argv) gives total number of command line arguments.

Eg: The name of script file is cmdline.py

```
import sys
print("Number of arguments:", len(sys.argv), "arguments")
print("Argument List:", str(sys.argv))
print("The name of this program is:", str(sys.argv[0]))
```

output:

```
Number of arguments: 4 arguments
Argument List: ['cmdline.py', '11', '22', '33']
The name of this program is: cmdline.py
```


Errors and Exceptions:

Errors are normally referred as bugs in the program. They are almost always the fault of the programmer. The process of finding and eliminating errors is called debugging.

There are mainly 2 types of errors.

1) Syntax Errors:

The python find the syntax error when it parses the source program. Once it find a syntax error, the python will exit the program without running anything. Commonly occuring syntax errors are

i) putting a keyword at wrong place.
ii) Misspelling the keyword
iii) Incorrect indentation
iv) Forgetting the symbols such as comma, brackets, quotes
v) Empty block.

2) Runtime Errors:

If a program is syntactically correct, is free of syntax errors- it will run by python interpreter. The program may exit unexpectedly during execution if it encounters a runtime error. The run-time errors are not detected while parsing the source program, but will occur due to some logical mistake.

i) Trying to access the file which doesn't

exists.

ii) performing the operation of incompatible type elements.

iii) using an identifier which is not defined.

iv) Division by zero.

Such types of errors are handled using exception handling mechanism.

Handling Exceptions:

An exception is an event that occurs during the execution of a program that interrupts the normal flow of the program.

When a python script encounters a situation it cannot cope with, it raises an exception. When a python script raises an exception, it must either handle the exception immediately otherwise it terminates and quits. The exception handling mechanism using try.. except.. else blocks. The suspicious code is placed in try block. After try

block place the except block which handles the exception elegantly. If there is no exception then else block statements gets executed.

Syntax:

```
try:
    Code
except Exception 1:
    Code
except Exception 2:
    Code
    ..
    ..
else:
    Code
```

Eg:

```
try:
    n=int(input("Enter some number:"))
except:
    print("you have entered wrong data")
else:
    print("you have entered:",n)
```

Output:

```
Enter some number:a
you have entered wrong data
Enter some number:10
    you have entered:10
```

Use of finally:

The finally clause will be executed at the end of the try-except block, if there is no exception, if an exception is raised and handled, If an exception is raised and not handled and even if exit the block using break, continue or return. It can use finally clause for clean up code that want to be executed.

Eg:

```
try:
    age = int(input("Enter your age"))
except ValueError:
    print("Invalid age")
else:
    print("your age is:", age)
    finally:
        print("Good Bye")
```

output:

```
Enter your age: 10
your age is: 10
Good bye
Enter your age: ten
Invalid age
Good bye
```

Modules:

Modules are imported using import command.

1) from.. import Statement:

There are many variables and functions present in the module. It can use them by using import statement. It can use any variable or function present within that module. The from.. import statement can use only selected variables or functions present within that module.

Eg:

```
from math import sqrt
print("Square root (25)=", sqrt(25))
```

output:

```
Square root (25)=5.0
```

To use some different name for standard function present in module use as keyword.

Eg:

```
from math import sqrt as my_sq_root
print("Square root(25)=", my_sq_root(25))
```

output:

```
Square root(25)=5.0
```

2) Name of the module:

Every module has a name. One can u

the name of the module using __name__ attribute of the module.

Eg: print("welcome")
print("Name of this module is:", __name__)

The __name__ is a built-in variable that is set when the program starts. If the program is running as a script, __name__ has the value __main__.

3) Creating a Module:

It can create own module. Every python program is a module. It means every file save using .py extension is a module.

i) Create a file having extension .py. It have created a file printMsg.py. The function fun is defined.

printMsg.py
```
def fun(usr):
    print("welcome", usr)
```

ii) open the python shell and import above module and then call the functionality present in module.

```
import printMsg
printMsg.fun("parth")
Welcome parth
```

4) dir() function:

It is an inbuilt function. It is used to display functions, variables or classes used in the module.

Eg:
```
def display(usr):
    print(usr)
usr = "Admin"
display(usr)
print(dir())
```

output:
```
Admin
['__annotations__', '__doc__', '__file__', '__name__', '__package__',
'usr']
```

b) python Module:

It is basically a file containing some functions and statements. When python file is executed directly it is considered as main module. It is recognized as __main__ and provide the basis for a complete python. The main module can import any number of other modules. But main module can't be imported into some other module.

6) Modules and Namespace:

Namespace is basically a collection of different names. Different Namespaces can co-exist at a given time but are isolated. If two names (variables) are same and are present

in same scope then it will cause name clash. To avoid such situation use the keyword namespace. Each module has its own namespace. It includes all names of its functions and variables. If use two functions having same name belonging to 2 different modules at a time in same other module then that might create name clash.

i) Create first module for addition. functionality

```
def display(a,b)
    return a+b
```

ii) Create second module for multiplication functionality

```
def display(a,b)
    return a*b
```

iii) If call above modules in driver program for performing both addition and multiplication get error because of name clash for function.

```
import FirstModule
import SecondModule
print(display(10,20))
print(display(5,4))
```

iv) To resolve this ambiguity, call these functionalities using their module names.

```
import FirstModule
import SecondModule
print(FirstModule.display(10,20))
print(SecondModule.display(5,4))
```

o/p:
30
20

vii) Global, Local and Built-in Namespace:

There are 3 commonly used categories of namespaces.

The global namespace contains the module is currently executing. The local namespace is a namespace for defining the names in a local function. The built-in namespace is a namespace in which the built-in functionality can be invoked.

```
import math
def even_number(number):  -> global namespace
    num = number  -> local namespace
    if (num%2==0)
    return num
    else
      return 0
print("Enter some number")
num=int(input())
if(even_number(num)!=0):
print("the square root of", num, "is", math.sqrt(num))
```

built-in namespace

viii) private Variables:

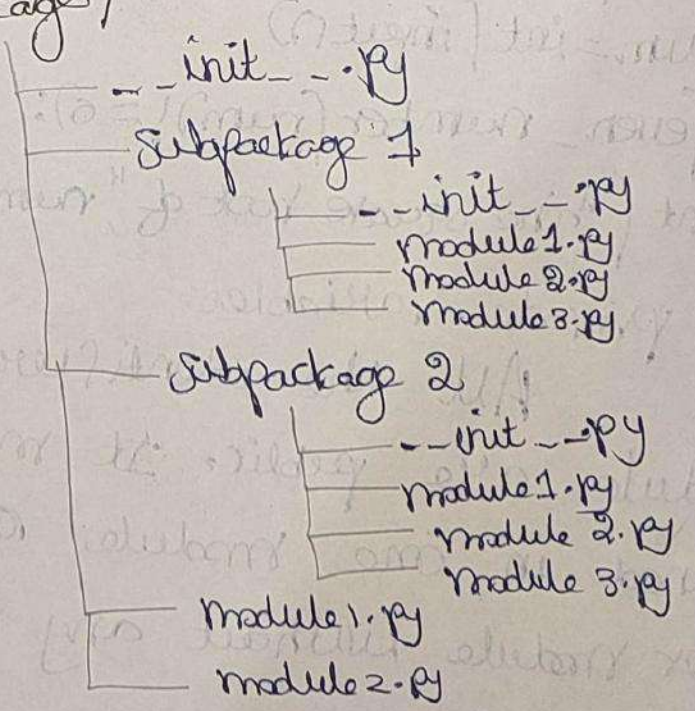
All the identifiers defined in module are public. It means the identifier defined in one module can be invoked by other module without any restriction. But

there are private Variables. The Variables begin with two underscore(_) are called private Variable. These are the variables which are used only within that module. other module cannot access these private Variables.

If write import * from modulename then all the identifiers except the private Variables can be imported.

Packages:

Packages are namespaces which contain multiple packages and modules. They are simply directories. Along with packages and modules the package contain a file named __init__.py. Infact to be a package, there must be a file called __init__.py in folder.



It can access the packages and various Subpackages and modules.

import My_package
import My_package.module1
from My_package.import module2
import My_package.subpackage1

When importing main package then there is no need to import subpackage.

Eg: import My_package
My_package.subpackage1.my_function1()

Thus no need to impot subpackage_1. The primary use of __init__.py is to initialize python packages. Simply putting the sub directories and modules inside a directory doesnot make a directory a package, it needs a __init__.py inside it. Then only python treats that directory as package.

How to Create a package?

To Create a package, perform arithmetic operations addition, multiplication, division and subtraction operations by Creating a package.

1) Create a folder named MyMaths in

working drive on D: drive

2) Inside MyMaths directory create __init__.py file. Just create an empty file.

3) Create a folder named Add inside MyMaths. Inside Add folder create file named addition.py. The addition.py will be

```
def add_fun(a,b):
    return(a+b)
```

4) Similarly the folder Sub inside MyMaths is created. Inside create a file Subtraction.py The code for this file is,

```
def sub_fun(a,b):
    return(a-b)
```

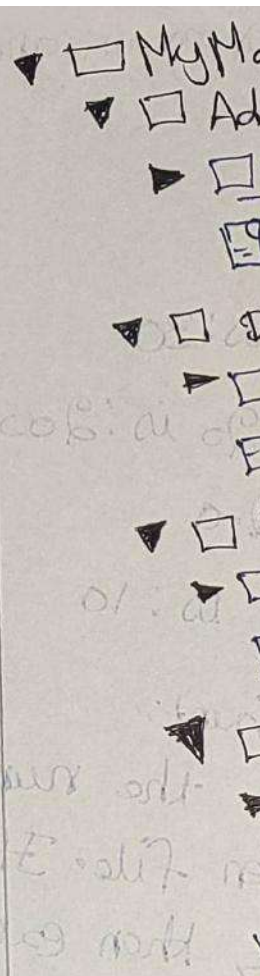
5) Similarly the folder Mul inside MyMaths is created. Inside create a file multiplication.py

```
def mul_fun(a,b):
    return(a*b)
```

6) Similarly the folder Div inside MyMaths is created. Inside create a file division.py

```
def div_fun(a,b):
    return(a/b)
```

7) The overall directory structure is given below. __pycache__ is automatically generated directory.



8) Now on D-drive create a driver program which will invoke all the above functionalities present in MyMaths package. It is named as testing_arithmetic.py

```
import MyMaths.Add.addition
import MyMaths.Mul.multiplication
import Mymaths.Div.division
import MyMaths.Sub.subtraction
print("the addition of 10&20 is :",MyMaths.Add.addition.
                                    add_fun(10,20))
print("the multiplication of 10&20 is :",MyMaths.Mul.
                multiplication.mul_fun(10,20))
print("the division of 10&5 is :",MyMaths.Div.division.
                                    div_fun(10,5)
print("the subtraction of 20&10 is :",MyMaths.Sub.
        Subtraction.Sub_fun(20,10))
```

9) Now run this testing program and will get the output as,

Output:

The addition of 10 and 20 is: 30
The multiplication of 10 and 20 is: 200
The division of 10 and 5 is: 2.0
The subtraction of 20 and 10 is: 10

Illustrative programs: Word Count:

This program counts the number of words present in the given file. If the specified file is not present, then exception is raised.

Program:

```
try:
  inFile = open("D:\\test.txt", 'r')
except:
  print("Error: File not found")
else:
  data = inFile.read()
  words = data.split()
  print(words)
  print("Total number of words are...", len(words))
```

Input File:

The input file is used for reading and for word counting is,

Welcome to Grace dg of Engg, Tuticorin

output:

Total number of words are: 7

Illustrative programs: Copy File

In this program, the contents of one file are copied line by line to another file. Create a file named one.txt in which some contents are stored. These contents are copied to another file named two.txt.

Program:

```
with open("D:\\one.txt") as inFile:
    with open("D:\\two.txt", "w") as outFile:
        for line in inFile:
            outFile.write(line)
    inFile.close()
    outFile.close()
    print("The contents of original files are...")
    with open("D:\\one.txt") as inFile:
        print(inFile.read())
    inFile.close()
    print("The contents of copied File are")
    with open("D:\\two.txt") as inFile:
        print(inFile.read())
    inFile.close()
```

Input File:

The input file one.txt is as,

one.txt:

Welcome to Grace clg of Engg

output:

Welcome to Grace clg of Engg

Illustrative programs: Voter's Age Validation:

```
try:
  age = int(input("Enter your age:"))
  if age >18:
    print("Eligible to vote!!")
  else:
    print("Not eligible to vote!!")
except ValueError as err:
  print(err)
finally:
  print("Good Bye!!")
```

output:

```
Enter your age:20
Eligible to vote!!
Good Bye!!
Enter your age:17
Not eligible to vote!!
Good Bye!!
Enter your age: twenty
invalid literal for int() with base 10: 'twenty'
Good Bye!!
```

Illustrative programs: Marks Range Validation (0-100):

```
while True:
    try:
        num = int(input("Enter a number between 1 and
                         100: "))
    if num in range(1,100):
        print("Valid number!!!")
        break
    else:
        print("Invalid number. Try again")
    except:
        print("That is not a number. Try again")
```

Output:

```
Enter a number between 1 and 100: -1
Invalid number. Try again
Enter a number between 1 and 100: 99
  Valid number!!!
Enter a number between 1 and 100: 1000
Invalid number. Try again
Enter a number between 1 and 100: five
This is not a number. Try again
```