GE 3151 - PROBLEM SOLVING AND PYTHON PROGRAMMING

UNIT-I COMPUTATIONAL THINKING AND PROBLEM SOLVING

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

UNIT - I

Fundamentals of computing

Computer

Computer is an electronic device that can be programmed to accept data (input) process it and generate results (outputs).

Computer performs both simple and complex operations with speed and accuracy.

Generations of a computer

The computer has evolved from large and simple calculating machine to a smaller but much more powerful machine.

Based on various stages of development, computers can be categorized into different generations.

Each generation of computer is designed based on new technological development resulting in better, cheaper and smaller computer that are more powerful, faster and efficient than their predecessors

First Generation (1940 - 1956) using Vaccum Tubes:

Hardware Technology: Vaccum Tubes are used for circuitry and magnetic drums are used for memory.
→ Punched Cards were used for input and outputs was displayed as printouts.

Software Technology: The instructions were in Machine language and this language was 0's and 1's for coding of the instructions.
→ The computation time was in milliseconds.
→ These computers were enormous in size and required a large room for installation.

Disadvantages:
Generated a lot of heat and were expensive to operate ☆ machines were prone to frequent malfunctioning.

Example:
UNIVAC - Universal Automatic computer
ENIAC - Electronic Numerical Integrator & calculator
EDVAC - Electronic Descrete variable Automatic computer.

Second Generation (1956 - 1963) using Transistors:

Hardware Technology:
This generation computers used magnetic core technology for primary memory and used magnetic tapes and disks for secondary storage.
Inputs was through punch cards and output was through printouts.

Software Technology:
Assembly language → This Language used ADD for Addition and SUB for Substraction for coding of the instructions.
→ The computation time was in micro seconds. → Transistors are smaller in size compared to vacuum tubes, thus the size of the computer was also reduced.
Disadvantages: Generated lot of heat but less than 1st generation computer ☆ less maintenance required.

Third Generation (1964 - 1971) using Integrated Circuits.

Hardware Technology:

Uses IC Chips. An IC chip, contains transistors Wiring and other components on a single silicon Chip.

The use of IC chip increased the speed and efficiency of computers.

Software Technology:

Keyboard and Monitor were interfaced through operating system.

OS allowed different applications to run at the same time.

⇒ High level languages were used exclusively for programming instead of machine language and Assembly Language.

⇒ The computation time was in Nanoseconds.

⇒ The size of these computers were small compared to 2nd generation computers.

Advantages:

This generation computer used for less power and generated less heats.

The cost of the computer was less

Fourth Generation (1971 to present): using microprocessors

This generation use large scale Integration (LSI) and very large Integration (VLSI) Technology.

Microprocessor is a Chip containing millions of transistors and components and designed using LSI and VLSI technology.

This generation computers gave rise to Personal Computers (PC)

Semiconductor memory replaced magnetic core memories, resulting in fast random access to memory.

Secondary storage device like magnetic disks became smaller in physical size and larger in capacity.

Software Technology:

OS like MS-DOS and MS-Windows were developed. This generation of computers supports GUI (Graphical User Interface).

High Level Programming language are used for writing of programs.

GUI is a user friendly interface that allows users to interact with computer via menus and icons.

Computing Characteristics:

The computation time is in picoseconds.

Physical Appearance:

Smaller than computers of previous generation.

Advantage:

* This computers are portable & more reliable & they generate more Less heat and require less maintenance.

Fifth Generation (Present and Next):

Artificial Intelligence:

5th Generation computers use super large scale Integrated (SCSI) chips that are able to store millions of components on a single chips.

This generation computers are based on Artificial Intelligence (AI). They try to simulate the human way of thinking and reasoning.

Artificial Intelligence includes area like Expert System (ES), Natural Language processing (NLP) Speech Recognition, voice Recognition, Robotics.

Components of a computer:

The computer is the combination of Hardware and software. Hardware is the Physical compent of a computer like motherboard, memory devices, monitor, keyboard, etc..

Both hardware and software together make the computers system to function.

The computer system hardware comprising of three main components:

i) Input/ output unit
ii) Central processing unit (CPU)
iii) Memory unit

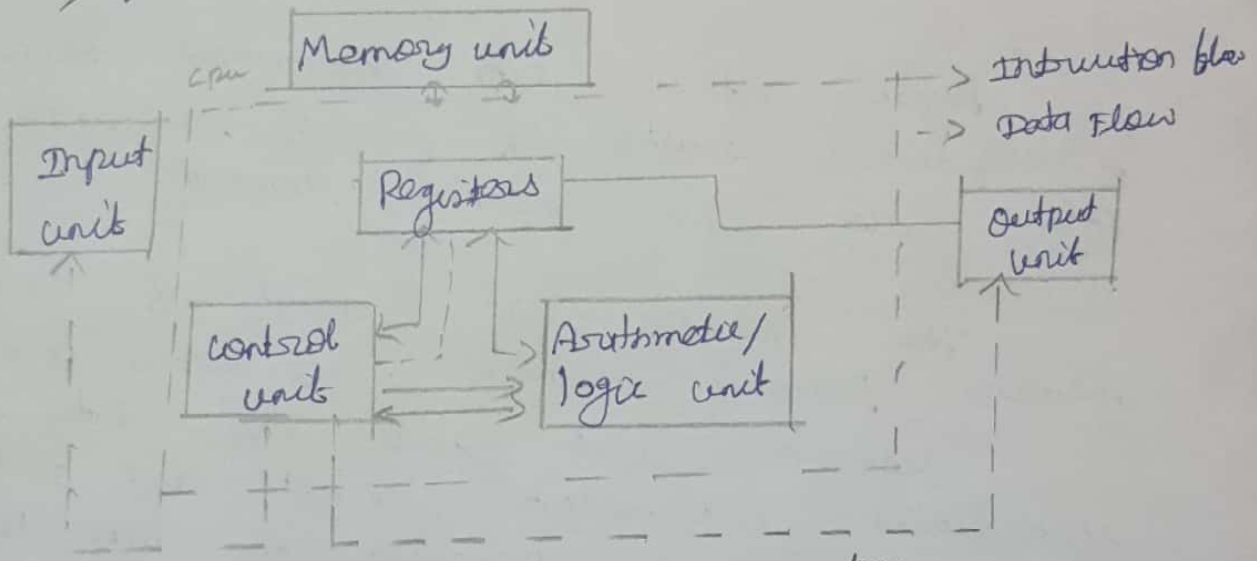


Fig: The computer system Interaction

I: Input / output unit

Input unit → The input is provided to the computer using devices like keyboard, trackball, and mouse.

Output unit → It provides the processed data output devices are monitor and printer.

II: Control processing unit (CPU)

CPU or the processor is often called the brain of computer.

CPU consists of
→ Arithmetic Logic unit (ALU)
→ control unit
→ set of Register

ii) Arithmetic Logic Unit (ALU)

ALU consists of two units

→ Arithmetic unit: performs arithmetic operations

→ Addition, subtraction, Multiplication & division

→ Logic unit: performs logic operations.

→ comparision of Numbers, letters & characters

iii) Control unit

CU coordinates the input & output devices of a computer.

The Control unit organizes the processing of data and instructions.

i) Registers:

Register store data, instructions, address and intermediate results of processing. So registers are often called CPU's working Memory

iii Memory unit:

The memory unit consist of cache memory, primary memory & secondary memory.

Memory unit stores the data and instructions intermediate results and output temporarily during processing of data this memory called Main Memory / primary memory

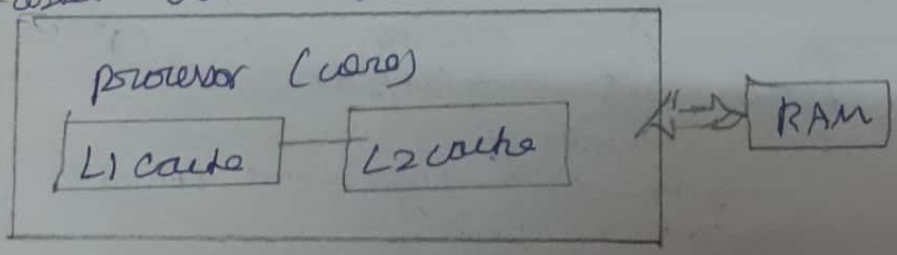
ex: RAM, Random Access Memory

ROM Read Only Memory.

i) Cache memory: High speed memory.

→ Cache memory is placed between RAM & CPU

→ Faster data access.



Primary Memory:

It is the main memory of computer stores data temporarily.

RAM → It provides temporary storage for data and instruction

It is volatile

It stores data when computer is on

Limited storage capacity, due to high cost.

ROM → It provides permanent storage

→ It is not volatile

→ Information will not be erased even if the computer is turned off.

Secondary Memory:

It store data and instructions permanently

It provides back up storage for data & Instructions.

Hard disk drive. Secondary memory has high storage capacity than the primary memory.

1.2 IDENTIFICATION OF COMPUTATIONAL PROBLEM:

Identification of computational problem is part of the scientific method, as it comes as the first step in systematic process to identify, evaluate the problem and explore potential solutions.

It consists of two steps.

① → Identifying and acknowledging that there is a problem.

② → Developing a problem statement.

Identification [understanding] gives a foundation for computational thinking which computational thinking is part of a problem solving process that ends with writing a program.

There are 4 steps related to Identification of problem and its solution. They are ① Decomposition ② Pattern recognition ③ Abstraction ④ Algorithm design

Decomposition:

The first step of computational thinking is decomposition. This stage starts by analyzing the problem, stating it precisely and establishing the criteria for the solution.

It starts by breaking the problem down into smaller more familiar components so they can be managed easier.

To develop a better understanding of the problem by identifying all components.

Pattern Recognition:

It identified units in the problem.

It some problems are similar in nature there is good chance that they can be solved using similar or repeated techniques.

It is a key skill to create efficient and effective solutions to given problems.

Abstraction:

The abstraction stage involves the identification of key components of the solutions.

Abstraction allows to consider all the key components prior to the creation of the final solution, while ignoring any unnecessary details.

Algorithm Design:

The final stage under the computational thinking process is algorithm design whereby a detailed step by step of instructions are created which explain how to solve the problem

Transferring a particular problem to a larger class of similar problems.

Problem solving Techniques:

There are three way to represent the logical steps for finding the solution to a given problem.

1. Algorithm
2. Flowchart
3. Pseudo code

Algorithm

It is a step-by-step procedure for solving any problem.

Algorithm is an English-like representation of the logic which is used to solve the problem.

To accomplish a particular task, different algorithms can be written. They differ by their time & space.

The algorithm can be implemented in many different languages by using different methods of programs.

Guidelines for writing Algorithms:

→ An algorithm should be clear, precise and well defined.

→ It should always begin with the word 'start' and end with the word 'stop'

→ Each step should be written in a separate line.

→ Steps should be numbered as step 1, step 2 and so on.

Advantages of Algorithm:

It is a simple to understand step by step solution of the problem.

It is easy to debug.

It is independent of programming Languages.

It is compatible to computers because each step of an algorithm can easily coded into its equivalent high level language.

Example: Algorithm to find the greatest among three numbers.

Step 1: Start

Step 2: Read the three numbers A, B, C

Step 3: Compare A and B. If A is greatest perform Step 4 else perform Step 5.

Step 4: Compare A and C. If A is greatest, output "A is the greatest" else output "C is the greatest".

Step 5: Compare B and C, If B is the greatest. output "B is the greatest" else output "C is the greatest".

Step 6: Stop

MAX

BUILDING BLOCKS OF ALGORITHMS

The algorithm can be constructed from basic building blocks. the building blocks are.

1. Statements
2. State
3. Control flow
4. Functions

Statements / Instructions.

An algorithm is a sequence of instruction to accomplish a task or solve a problem.

The algorithm consists of finite number of statements. It must be in an ordered form.

Statement may consists of 3

- Assignment Statement
- I/O Statement
- Control Statement

Ex

```
Read the value of a & b      //Input Stmt
Calculate C = a + b          //Assignment Stmt
print c                      //output Stmt.
```

States:

An Algorithm starts from the initial state with some input values. As actions are performed, its state changes. It ends with a final state.

Control flow

The order of execution of statements is known as the control flow. There are 3 types.

i) Sequential control flow
ii) Selection control flow
iii) Looping or Iteration control flow.

i) Sequence control flow

A sequence of statements is executed one after another in the same order as they written. only one step executed once. this algorithm performs the steps in a purely sequential order.

Ex: Algorithm to find the sum of two numbers.

Step 1: Start
Step 2: Read two numbers A and B
Step 3: Calculate Sum = A + B
Step 4: Print the Sum value
Step 5: Stop.

ii) Selection control flow:

A condition of the state is tested, and if the condition is true, one statement is executed If the condition is false, an alternative statement is executed.

basic structure:

```
IF condition is TRUE then
        perform some action
Else if condition is FALSE then
        perform some action.
```

Ex: Finding the greatest number.

iii) Iteration (Looping) control flow:

A flow set of statements are repeatedly executed based upon a condition. If a condition evaluates to true, the set of statements (true) is executed again and again.

One or more steps are performed repeatedly until some condition is reached

Algorithm to find the sum of first 100 integers

1. Start
2. Assign sum = 0, i = 0
3. Calculate i = i + 1 and sum = sum + i
4. Check whether i >= 100, if no repeat step 3. Otherwise goto next step
5. Print the value of sum.
6. Stop

Functions:

In some cases, algorithms can become very complex. the variables of an algorithm and dependencies among the variables may be too many. then it is difficult to build algorithm correctly.

In such situations, we break an algorithm into parts, construct each part separately. and then integrate the parts to the complete algorithm.

"A function is a block of organized reusable code that is used to perform a single task of some kind"

Functions avoid repetition of same codes over and over. It helps easy debugging, testing & understanding of the program.

Functions reduce program size & program development time. & high degree of reusability for the problems.

NOTATION

A notation is a system of characters, expressions, graphic or symbols used in problem solving process to represent technical facts to facilitate the best result for a problem.

Example: Pseudo code, Flow chart.

Pseudo code:

→ Pseudo code is a short, readable and formally styled english language used for explaing an algorithm.

→ It does not include details like variable declarations, subroutines. It is short hand way

→ Using Pseudo code It is easier for a programmer or a non programmer to understand the general working of the program.

→ It is not a machine readable.

→ It can not be compiled or executed, there is no standard for the syntax of Pseudocode.

preparing a Pseudocode:

Written using structured English.

Pseudo code uses some keywords to denote programming process. some of them

Input : INPUT, GIET, READ, PROMPT

Output: OUTPUT, PRINT, DISPLAY, SHOW

Processing: COMPUTE, CALCULATE, DETERMINE, ADD, SUBTRACT, MULTIPLY, DIVINE

Initialize: SET and INITIALISE

Incrementing: INCREMENT

The keywords should be Capitalized.

There are three control structures used in Pseudo code,

i) Sequence control structure

ii) Selection " "

iii) Iteration " "

Sequence of control structures.

The statements are executed one after another in the same order as they are written from top to bottom.

EXAMPLE: Find product of any two numbers

READ values of A and B
COMPUTE C by multiplying A with B
PRINT the result C

Selection Control Structure

The condition is tested, and if the condition is true, one set of statements are executed. if the condition is false, an alternative set of statements are executed.

There are two main selection structures.

1) IF-THEN-ELSE statement
2) CASE Statement

The

IF-THEN-ELSE Statement:

```
IF condition THEN
        process1
ELSE
        process2
END IF
```

Ex: Find maximum of any three numbers

```
READ values of A, B, C
IF A is greater than B THEN
        ASSIGN A to MAX
ELSE
        ASSIGN B to MAX
IF MAX is greater than C THEN
        PRINT MAX is greatest
ELSE
        PRINT C is greatest
STOP
```

The case statements.

It used when many numbers of conditions to be checked. In a case statement, dependening on the expression, one of the condition is true. Based on the value, the corresponding statements are executed.

syntax..

```
CASE value 1:
    process 1
CASE value 2:
    process 2
CASE value n:
    process n
END CASE
```

Iterative control statement.

A set of statements are repetitively executed based upon a condition.

There are two iterative control structures. they are WHILE and DO - WHILE.

WHILE condition statements END WHILE	DO statements WHILE condition END DO
WHILE loop the condition is executed at the beging of the Loop	DO WHILE loop the condition is executed at the end of the loop.
Execute the statements while the condition is true	Execute the statements while the condition is true.

Ex: Find sum of first 100 integers

```
INITIALISE sum to zero
INITIALISE i to zero
DO WHILE (i less than 100)
    INCREMENT i by 1
    ADD i to SUM and store in SUM
END DO
PRINT SUM
STOP
```

Advantages of Pseudo code:

A Pseudo code is closer to the programming code. Thus it can be easily converted into the actual program.

Writing Pseudocode is much easier in comparison with drawing a flow chart.

Limitations of Pseudo code:

It is difficult to understand

It becomes hard to maintain consistency.

Flow Chart:

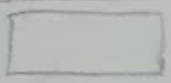
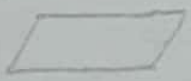
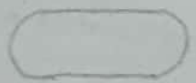
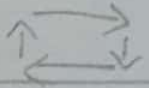
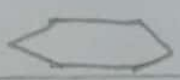
A flow chart is a digramatic representation of the logic for solving a task.

A flow chart is drawn using boxes of different shapes with lines connecting them to show the flow of control.

Flow Chart Symbols:

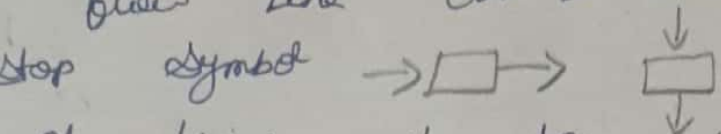
Draws using a different kinds of symbols.

Every symbol used in a specific purpose.

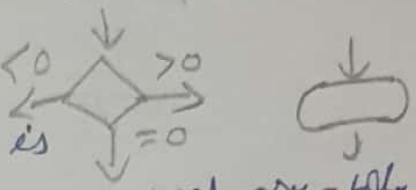
Symbol	Symbol Name	Description.
	process	Operation or action step
	Data	Input / output (I/o) to form a process.
	Terminator	Represent the start & stop of the program
	Decision	Decision making and branching.
	Flow Lines	Indicates the direction of flow.
	connector	Join flow Lines.
	preparation	Set up step process.

Guidelines for preparing a flowchart

There can be only one start and one stop symbol in a flow chart.

Only one flow line can be used with the start and stop symbol 

Direction of flow information from top to bottom or from left to right.

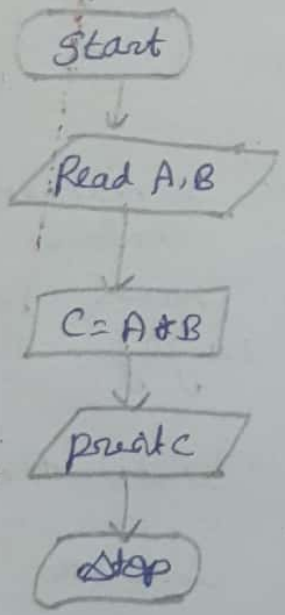
Only one flow line should enter a decision symbol but two or three flow lines one for each possible answer, should leave the decision symbol. 

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

If the flow chart becomes complex it is better to use connector symbols to reduce the number of flow lines.

The flow lines should not cross each other. It has logical start & finish.

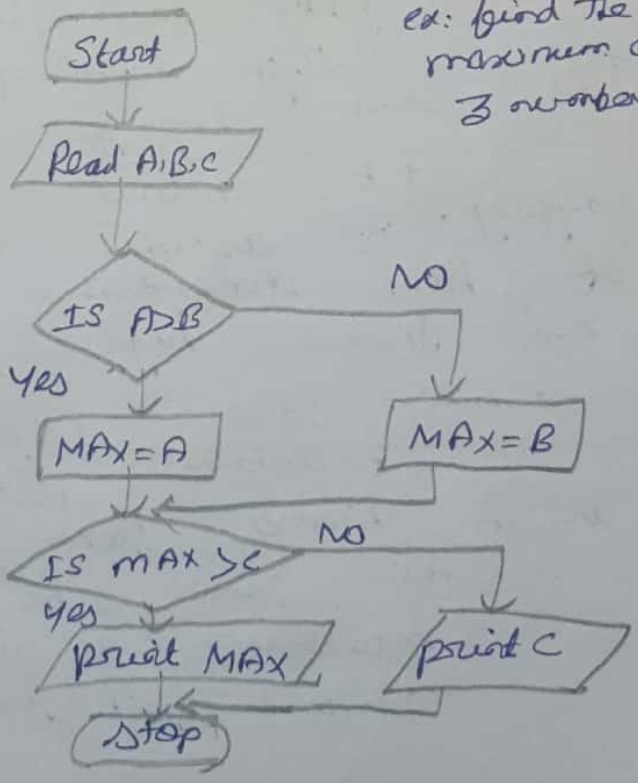
sequence control structure



ex: product any two numbers.

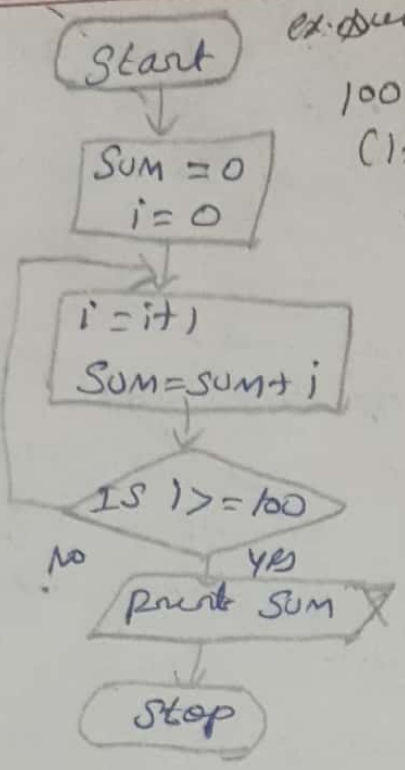
selection control structure

ex: find the maximum of 3 numbers.



Iterative control structure.

ex. Sum of first 100 integers (1+2+....+100)



Advantages of flow chart

→ Easier to understand
→ better communicate the problem solving logic to the users.
→ A problem can be analyzed in a effective way.

Limitations of flow chart

→ In case of large programs, the flowcharts may continue too many pages. difficult to understand
→ lot of time
→ It is a challenging job for changes or modifying.

Programming Language:

It consists of a vocabulary containing a set of grammatical rules intended to convey instructions to a computer.

It has unique set of keywords along with a special syntax to organize the software instructions.

two types of programms Language.

i) Low Level Languages
ii) High Level Language.

Low Level Languages:

It include assembly & machine Language. An assembly Language contains a list of basic instructions and is more harder to read than a high Level Language.

⇒ Machine Level Language.

It contains set of Instructions that are in binary form 0 or 1.

It cannot be easily understood by humans.

maintenance is also very high.

It is a first generation programms Language.

⇒ Assembly Language.

It contains some human-readable commands such as MOV, ADD, SUB. It is easier to write & understand.

The assembler is used to convert the assembly code into machine code.

It is a second generation Programming Language.

High Level Language:

Programmers to write the programs which are independent of a particular type of computer.

It has high level because they are closer to human languages than machine Language.

BASIC, COBOL, FORTRAN, JAVA ... ex.

Algorithmic problem solving:

Computers are used for solving various day to day problems. It is important to mention that computers themselves cannot solve a problem.

Problem solving begins with the precise identification of the problem and ends with a complete working solution in terms of a program or software. Key steps are.

1. Obtain a description of the problem
2. Analyze the problem
3. Develop a high level algorithm.
4. Refine the Algorithm by adding more details
5. Review the algorithm.

Step1: Obtain a description of the problem.

To solve the problem, first we must state the problem clearly. A problem is specified by the given input and desired output.

The goal of the algorithm is to establish the relation between the input and output. Developer must create an algorithm that will solve the user's problem.

The user is responsible for creating a description of the problem, but the user often the weaker part of the process.

A problem description suffer from one or more of the following types of defects.

- the description may rely on unstated assumptions
- the description may be ambiguous.
- the description may be incomplete.
- the description may have internal contradictions

Step2: Analyze the Problem

⇒ It is important to clearly understand a problem before we begin to find the solution for it. by analyzing a problem, we would be able to figure out what are the inputs that our program should accept and the output that it should produce.

Asking the following questions often helps to determine the starting point.

⇒ what data are available?
⇒ where is that data from?
⇒ what are the rules existing for working with the data
what are the relationships existing among the data value?
⇒ when algorithm needs determining the ending point.

Of a solution. Asking these questions describe the characteristics

⇒ what new facts will we have?
⇒ what items would have changed?
⇒ what changes should have been made to them
what terms?
what things will no longer exist?

Step3: Develop a high level algorithm

It is essential to devise a solution before writing a program code for a given problem. we start with a tentative solution plan & keep on refining the algorithm until the algorithm is able to capture all the aspects of desired solution.

For a given problem, more than one algorithm is possible and we have to select the most suitable solution.

It is better to start with a high level algorithm that includes the major part of a solution, but leaves the details until later.

Step 4: Refine the algorithm by adding more detail.

→ A high level algorithm shows the major steps that need to be followed to solve a problem. Our goal is to develop algorithms that will lead to computer programs.

→ For larger, more complex problems, it is common to go through the process several times. Each time more details are added to the previous algorithm for further refinement.

→ This technique of gradually working from a high-level to a detailed algorithm is often called stepwise refinement.

→ Stepwise refinement is a process of developing a detailed algorithm by gradually adding details to a high level algorithm.

Step 5: Review the algorithm.

→ The final step is to review the algorithm. First work through the algorithm step by step to determine whether or not it will solve the original problem. Asking these questions and seeking their answers is good way to develop skills that can be applied to the next problem.

→ For example an algorithm that computes the area of a circle having radius 5.2 meters (formula πr 5.2²) algorithm that compute area of any circle (formula πr²) solve a more general problem.

· Can this algorithm be simplified.

ex: one formula for computing the perimeter of rectangle is length + width + length + width

A simple formula would be 2(length + width)

Once an algorithm is developed, translate it into a computer program in some programming language.

1.9 SIMPLE STRATEGIES FOR DEVELOPING ALGORITHMS:

There are various kind of algorithms development techniques formulated and used for different types of problems.

i) Iteration
ii) Recursion.

Iteration:

Iteration is the process of repeating the same set of statements again and again until the specified condition holds true.

Human find iterative task boring but computers are very good at performing iterative tasks. computers execute the same set of statements again & again by putting them in a loop.

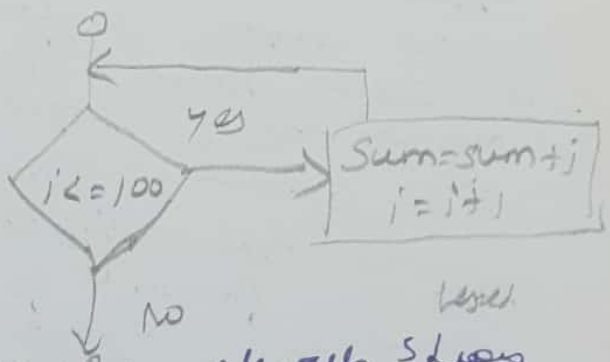
In general loops are classified as
counter - controlled loops
sentinel - controlled loops.

Counter Controlled Loops.

It is a form of looping in which the number of iterations to be performed is known in advance. It use the loop counter, to keep a track of loop iteration. It starts with the initial value of the loop counter and terminates when the final value of the loop counter is reached.

ex: Consider a program to find the sum of given 100 integers.

```
i = 1
Sum = 0
while (i <= 100):
    Sum = Sum + i
    i = i + 1
```



the computer executes the loop body a desired number of times.

flow chart clearly shows the repetition.

Sentinal - controlled Loops.

The execution or termination of the loop depends upon a special value called the sentinal value. It is true the loop body will be executed. otherwise, it will not.

since the number of times a loop will iterate is not known in advance, this type of loop is also known as indefinite repetition loop.

Algorithm: To find the sum of first N Integers.

1. Start
2. Read N
3. Assg Sum=0; i=0
4. Calculate i=i+1 and SUM=SUM+i
5. Check whether i>=N, if not repeat step 4. otherwise goto next step.
6. Print the value of SUM
7. Stop.

II. Recursion:

Recursion is a powerful programming technique that can be used to solve problems that can be expressed in terms of similar problems of smaller size. Ex finding factorial of n n!=nx(n-1)!

In recursive programming, a function calls itself. A function that calls itself, is known as recursive function, and the phenomenon is known recursion

Recursion is classified according to the following criteria

1) whether the function call itself directly. (i.e) direct recursion) or Indirectly (Indirect recursion).

2) whether there is any pending operation on return from a recursive call. If the recursive call is the last operation of a function, the recursion is known as tail recursion

Function is directly recursive if it call itself. Indirect function calls another function.

Ex. Recursive algorithm for finding the factorial of a number.

Step 1: Start
Step 2: Read number n
Step 3: Call factorial (n)
Step 4: Print factorial f
Step 5: Stop

Factorial (n)
Step 1: If n == 1 then return 1
Step 2: Else
f = n * factorial (n-1)
Step 3: Return f.

---

1. ILLUSTRATIVE PROBLEMS.

Finding Minimum Number in a List:

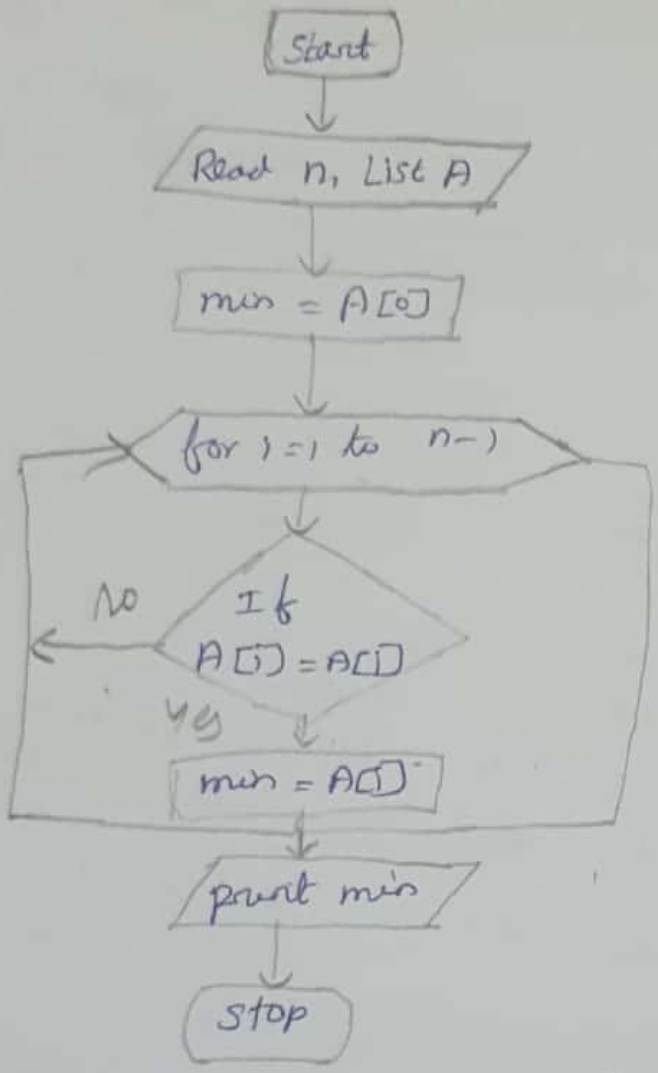
problem statement:
Finding the minimum number in a list is an example of sorting problem.

Algorithm:
1. Assign the value of the list as minimum value
2. Compare this value to the other values starting from second value
3. When a value is smaller than the present minimum value, then it becomes the new minimum.
4. Print minimum value.

Pseudocode:

```
Min = A[0]
i=1
WHILE ( i<=n-1)
   IF (A[i] < Min) THEN
      Min = A[i]
   END IF
   i=i+1
END WHILE
PRINT Min.
```

Flow Chart:



Example:

Consider an array A:

0	1	2	3	4
50	40	5	9	45

Step 1: Assign min = A[0]

0	1	2	3	4
50	40	5	9	45

min = 50

Step 2: Compare A[1] and min

0	1	2	3	4
50	40	5	9	45

Min = 40 40 < 50 now min = 40

Step 3: Compare A[2] and min

0	1	2	3	4
50	40	5	9	45

Min = 5 5 < 40 Now m

Step 4: Compare A[3] and min

0	1	2	3	4
50	40	5	9	45

Min = 5 9 > 5 so again min = 5

Step 5: Compare A[4] and min

0	1	2	3	4
50	40	5	9	45

Min = 5 45 > 5, So again min = 5

2. Inserting a card in a List of Sorted Cards.

Problem Statement

Imagine that you are playing a card game. You are holding the cards in your hand, and the cards are sorted. You are given exactly one new card. You have to put it into the correct place so that the cards you are holding are still sorted.

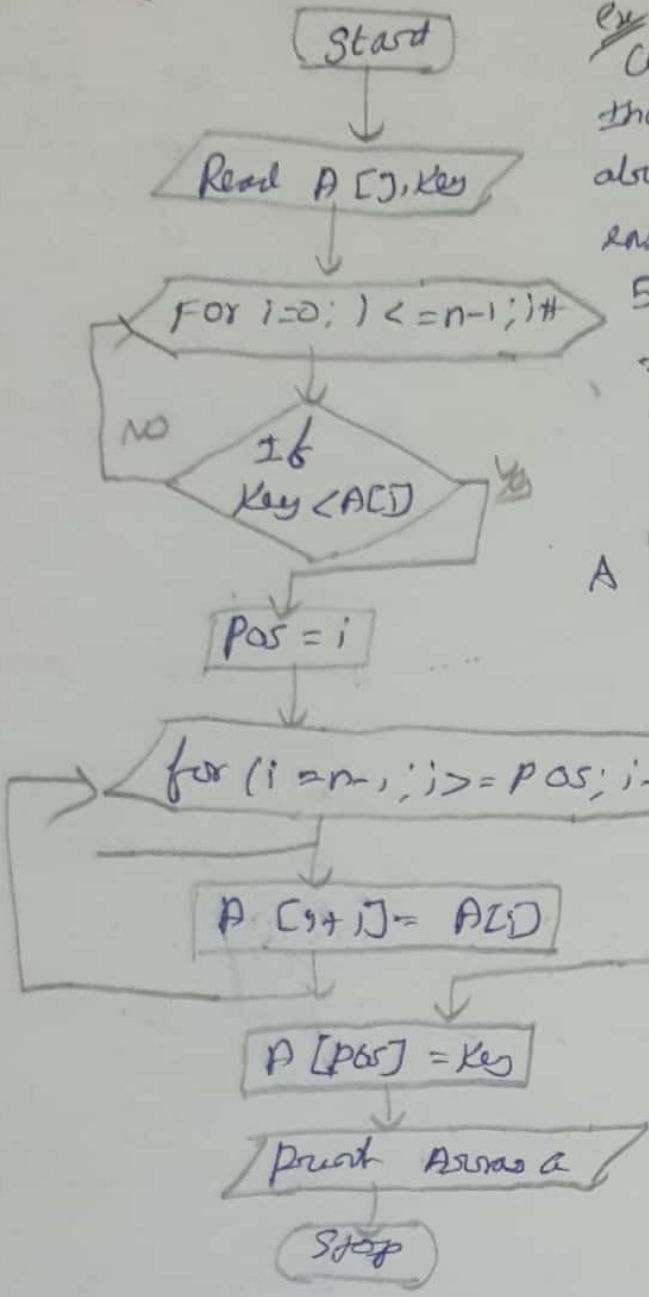
Algorithm:

Step1: Start
Step2: Read all numbers in an array A
Step3: Read Key
Step4: From first position to the end of the array compare key and array element
Step5: If key A[i] then assign i to pos. and turn pos
Step6: From n-1 to pos move the elements one position down.
Step7: Now store key value in pos location
Step8: Stop

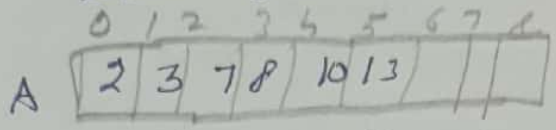
Pseudocode:

```
Void inserting_card (int A[], int n, int Key)
{
  int pos;
  for (i=0; i<=n-1; i++)
  {
    if (Key < A[i])
    {
      pos=i;
      return pos;
      break;
    }
  }
  for (i=n-1; i>=pos; i--)
  {
    A[i+1]= A[i];
  }
}
```

Flow Chart

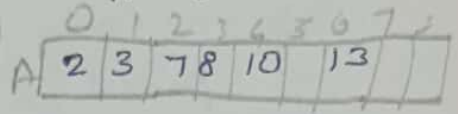


Ex Consider the array from index 0 through index 5 is already sorted & need to insert the key element 5 into sorted sub-array. Then the sub array from index 0 through index 6 is sorted.

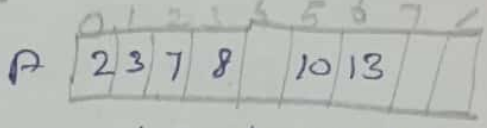


Key = 5, Pos = 2

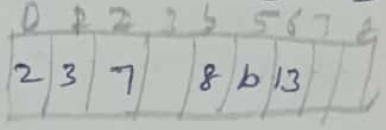
Step 1: i = 5
A[6] = A[5]



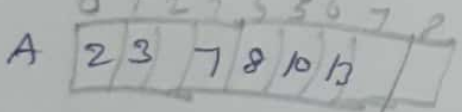
Step 2: i = 4 A[5] = A[4]



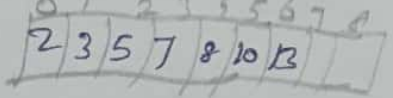
Step 3: i = 3 A[4] = A[3]



Step 4 i = 2 A[3] = A[2]



Step 5: Now insert the key value



the above operation just copies the element one position to the right.

3 Guessing an Integer Number in a Range.

problem statement:

The objective is to randomly generate integer number from 0 to n. Then the player has to guess the number. If the player guesses the number correctly output an appropriate message.

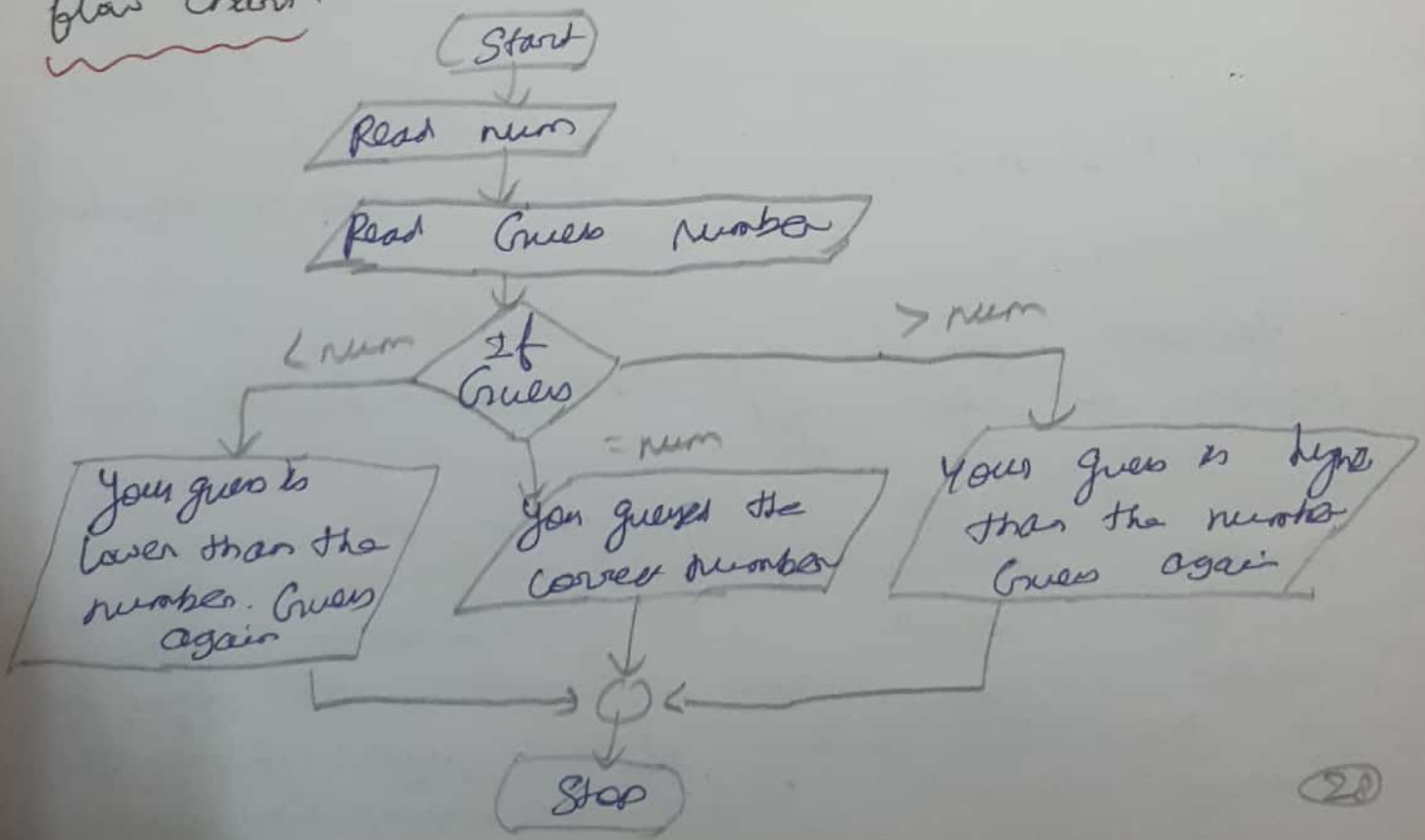
If the guessed number is less than the random number generated, output the message, "Your guess is lower than the number. Guess again". Otherwise output the message "Your guess is higher than the number. Guess again".

Then the player guesses another number. This process is repeated until the player guesses the correct number.

Algorithm:

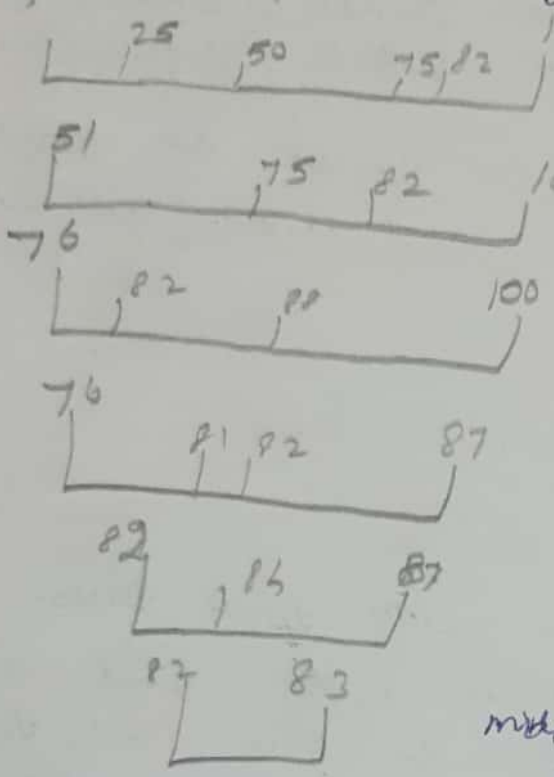
1. Start
2. Generate a random number and read num.
   a. Enter the number to guess
   b. If (guess is equal to num)
      Print "You guess the correct number".
   Otherwise
      If (Guess is less than num)
      Print "your guess is lower than the number. Guess again"
3. Stop.

flow chart.



Example:

Consider to guess a number between 1 and 100 the number selected for guess is 83. now the player wants to guess this number.



First Guess: 50
Answer: Too Low

Second Guess: 75
Answer Too Low:

Third Guess: 88
Answer: Too high

Fourth Guess: 81
Answer: too Low.

Fifth Guess: 84
Answer too high.

Midpoint is 82.5, which rounded to 82

Sixth Guess: 83
Answer: Correct.

Pseudo Code:

```
BinSearch (int Value)
{
    int upperBound, lowerBound, mid;
    upperBound = n-1;
    lowerBound = 0;
    while (lowerBound <= upperBound)
    {
    mid = (upperBound+lowerbound)/2;
    If (A[mid] == value)
        return mid;
    else
        if (value < A[mid])
            upperbound = mid-1;
        else
            lowerBound = mid+1;
    }
}return;
```

4. Tower of Hanoi problem:

problem statement:

Tower of Hanoi is one of the classical problems of computer science the problem statement

→ there are three stands (Stand1, 2 and 3) on which a set of disks, each with a different diameter, are placed.

→ Initially, the disks are stacked on Stand1, in order of size, with the largest disk at the bottom.

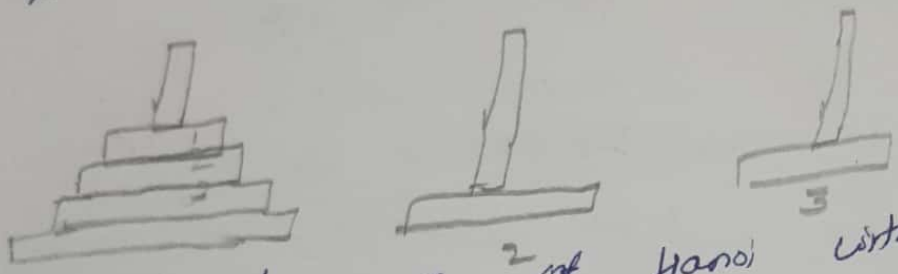


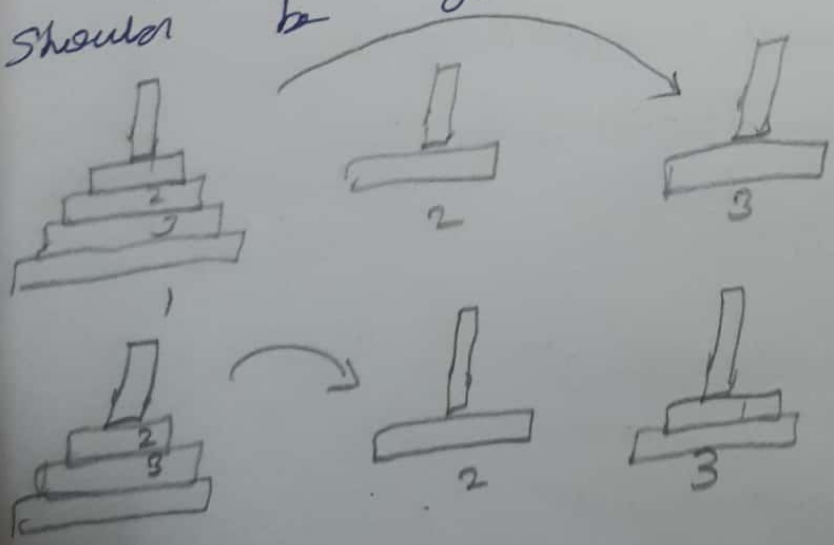
fig: Tower of Hanoi with three Disks.

The "Tower of Hanoi problem" is to find a sequence of disk moves so that all the disks move from Stand-1 to Stand 3, adhering to the following rules:

move only one disk at a time

A large disk cannot be placed on top of a smaller disk.

All disks except the one being moved should be on a stand.



Pseudocode

START
Procedure Hanoi (disk, Source, dest, aux)
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)

END IF

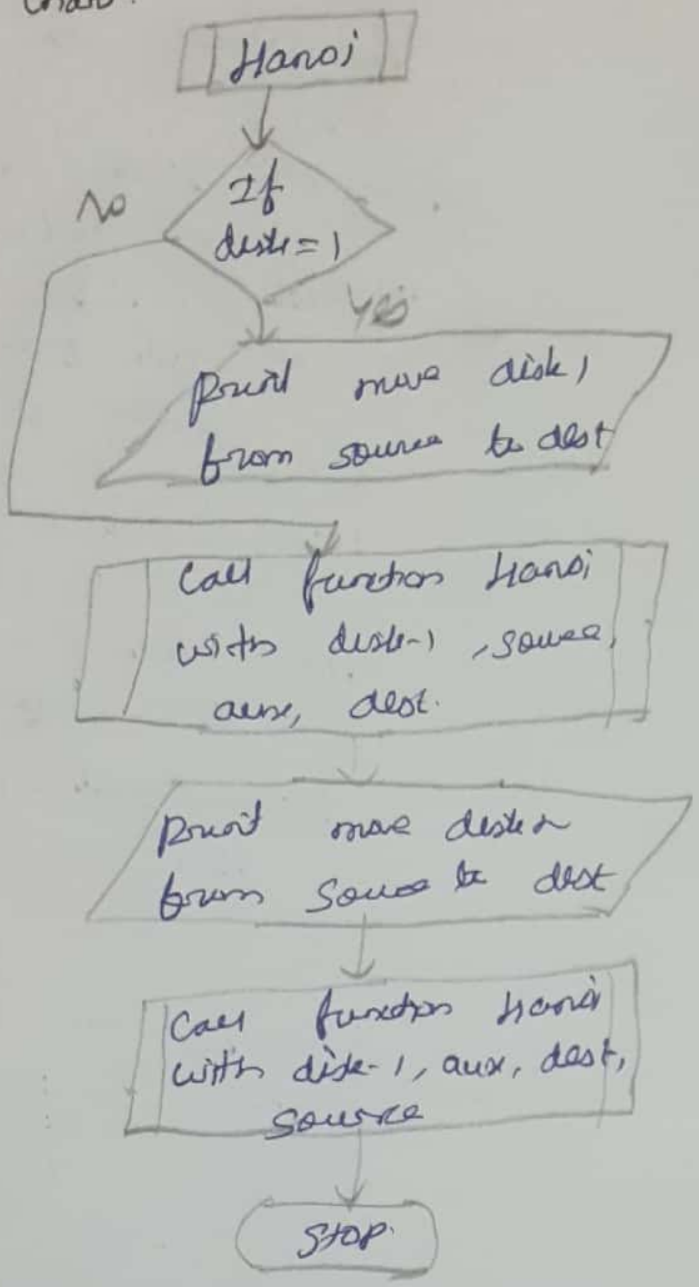
END Procedure
STOP

Generate Steps to Solve the Tower of Hanoi problem with three disks.

The movement of n-1 disks forms the recursive case of recursive solution to move n disks. The base case of a solution involves the movement of only one disk.

$$\text{Tower of Hanoi(disks)} = \begin{cases} \text{move the disk} & \text{If disks} = 1 \\ \text{Tower of Hanoi (disks-1)} & \text{if disks} > 1 \end{cases}$$

Flow Chart.



Recursive Algorithm

Hanoi (disk, Source, dest, aux)

1. If only one disk is there then move disk from source to dest.
2. If more than one disk is there then
   2.1 Recursively call Hanoi (disk-1, Source, aux, dest)
   2.2. move disk from source to dest
   2.3. Recursively call Hanoi (disk-1, aux, dest, Source)

Thank you

UNIT II DATA TYPES, EXPRESSIONS, STATEMENTS

Python interpreter and Interactive mode, debugging Values and types: int, float, boolean, string, and List: variables, expressions, statements, tuple assignment, Precedence of Operators, Comments; Illustrative programs: exchange the values of two variables, circulate the values of n variables, distance between two points.

Introduction:

"Python is a general purpose interpreted, Object oriented and high level programming Language."

It was created by Guido van Rossum during 1985 - 1990. Python got its name from "monty python's flying circus". Python was released in the year 2000.

I Python has two basic modes:

Interpreted mode
Interactive mode.

Interpreter mode is the mode where the scripted and finished .py files are run in the Python interpreter.

Interactive mode is the command line shell which gives immediate feedback for each statement, fed in the active memory. As new lines are fed into the interpreter, the fed program is calculated both in part and as whole.

2.1: Python Interpreter mode:

The Python interpreter is a program that reads and executes Python code, in which Python statements can be stored in a file.

The Python system reads and executes the commands from the file rather than from the console. Such a file is termed as Python program.

Source program → Interpreter → output

```
Python 3.6.0 Shell
File Edit Shell Debug Options Windows Help
Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 07:18:10)
[MSC v.1900 32 bit (Intel)] on win32
Type "Copyright", "credits" or "License()" for more information.
>>>
```

Fig Python Interpreter window.

The first 3 lines contain information about the Interpreter and the operating system it is running on. The version number is 3.6.0 It begins with 3. if the version is Python 3. It begins with 2, if the version in Python 2.

The last line >>> is a prompt that indicates that the interpreter is ready to enter code.

If the user types a line of code and hits enter the interpreter displays the result.

The file name has extension ".py" to execute a script. type the file name along with the path at the prompt.

Ex: >>> python Sum.py
Enter 2 numbers
6
3
The Sum is 9

2. Python Interactive Mode

Python allows the user to work in an interactive mode. It is a way of using the Python interpreter by executing python commands from the command line with no scripts.

This allows the user to type an expression, and immediately the expression is executed and the result is printed.

The '>>>' is the prompt used in the python interactive mode which indicates that the prompt is waiting for the python command and produces the output immediately.

Example..

```
>>> 5+7
12
>>> Print ("Hello world!")
Hello world.)
>>> num =10
>>> num = num/2
>>> Print (num)
5.0
```

Writing Multiline code in Python Command Line.

In this enter key for continuation lines this prompts by default three dots (...) the continuation lines are needed only in the case of procedures, branching loop constructs. when it happens, press a tap of indentation.

Example:

```
>>> flag =1
>>> if flag:
... Print ("WELCOME TO PYTHON")
...
WELCOME TO PYTHON.
>>>
```

2.2: DEBUGGING:

A programma can make mistakes while writing a program. and hence, the program may not execute or may generate wrong output. the process of identifying and removing such mistakes also known as bugs or errors from a program is called debugging.

Errors occuring in programs can be categories.

i) Syntax error
ii) Logical error
iii) Runtime error

Syntax errors:

Python has its own rules that determine its syntax. The interpreter interprets the statements only if it is syntactically correct. If any syntax error is present, the interpreter shows error message(s) and stops the execution there.

Example: Parenthesis must be in pairs, so the expression (20+22) is syntactically correct. (17+15 is not correct.

Logical errors or Semantic errors:

A logical error is a bug in the program that causes it to behave incorrectly. A logical error produces an undesired output but without abrupt termination of the execution of the program. It is sometimes difficult to identify these errors.

Ex: To find the average of two numbers 10 and 12 we write the code as 10+12/2 runs successfully and produces the result 16. Whereas, 16 is not the average of 10 & 12. the correct code to find the average (10 + 12)/2 to give the correct output 11.

Running Error:

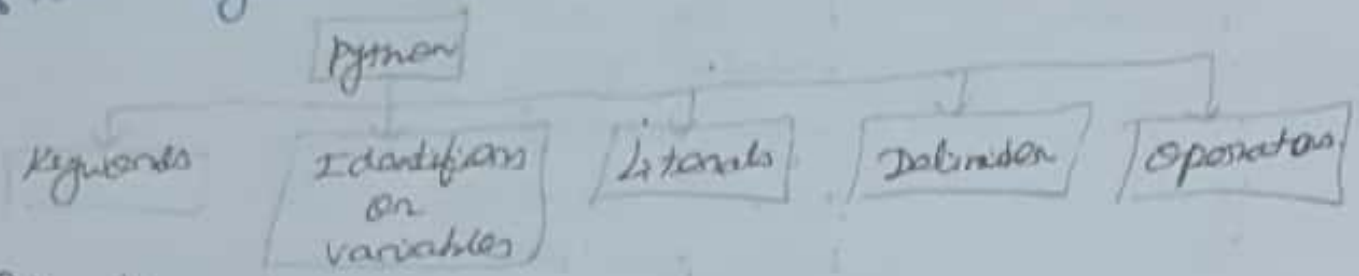
Running error causes abnormal termination of program while it is executing. It when the interpreter is correct syntactically, but it cannot execute it correctly. It do not appear until after the program starts running or executing.

Example.

In a statement having division operation in the program. by mistake, if the denominator entered is zero then it will give a runtime error like "division by zero".

2.3 TOKENS

Python breaks each logical line into a sequence of elementary lexical components known as tokens. A token is the smallest unit of the program. Each token corresponds to a substring of the logical line. The tokens in Python are.



2.4 Keywords:

Keywords are reserved words they have predefined meanings in Python. They cannot be used as ordinary identifiers. Python is case sensitive so keyword must be spelled exactly as they are written. Python 3 has 33 keywords. The first three are grouped together because they all start with uppercase letters.

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

2.5 Identifiers.

A Python identifier is the name given to a variable, function, class, module or other object.

An identifier can begin with an alphabet (A-Z or a-z) or underscore (_) and can include any number of letters, digits or underscore.

Spaces are not allowed.

Python will not accept @, $ and %. as identifiers.

Python is case sensitive language. Thus Hello and hello are different identifier.

2.6 Escape sequences:

Escape sequences are non-Printable characters. It consists of backslash followed by a character. It is used to representing certain white space characters. The following Escape sequence are.

Escape sequence	Meaning
\newline	Backslash and newline ignored
\\	Backslash (\)
\'	single quote (')
\"	Double quote (")
\a	ASCII Bel (BEL)
\b	ASCII Backspace (BS)
\f	ASCII Formfeed (FF)
\n	ASCII Linefeed (LF)
\r	ASCII Carriage Return (CR)
\t	ASCII Horizontal Tab (TAB)
\v	ASCII Vertical Tab (VT)

2.7 LITERALS

A Literals is a sequence of one of more characters that stands for itself. There are two types

i) Numeric Literals ii) String Literals.

2.7.1 Numeric Literals:

Is a Literal containing only the digits 0-9 & an optional sign character (+ or -) and a possible decimal point. Commas are never used in numerical literal.

If a numeric Literal does not contain a decimal point, then it denotes an integer Ex: 10

If a numeric Literal a decimal point then it denotes a floating point (fixed) value Ex: 12.35

Limits & Range in floating point Representation

i) Arithmetic overflow => too large

Ex >>> 1.5e200 * 2.0e20 (infinity)

ii) Arithmetic underflow => too small

Ex >>> 1.0e230 / 1.0e100

2.7.2 String Literals

String represent a sequence of characters surrounded by quotes.

Ex: 'Hello' "Java, Java" "231, Cannot Negan, 621655"

A character Literal is a single character surrounded by single or double quotes. The value of triple quote """ is used for a multiline string literal.

Ex: >>> print ("welcome to Python")
welcome to Python!

2.8 DELIMITERS

are symbols that perform three special role in Python like grouping, Punctuation and assignment / binding of codes to name types.

Delimiters	Classification
( ) [ ] { }	Groupings
- , : ; @	Punctuation
= += -= *= /= //= %= **=	Arithmetic assignment / binding
& = != <= <<= >>=	Bit wise assignment binding.

2.9 Values

Values are the basic units of data, like a number or a string that a program manipulates. There are 3 different types. Ex: 2 an integer, 42.0 floating point "Hello, world!" is a string

2.10 TYPES:

Every value belongs to a specific datatype in Python. built in data types are, Numbers (Integer type, Floating point type, a complex type), String, List, Tuple, Set, Boolean or mapping types

Number data types store numeric values. Python has three number types.

i) Integer number
ii) Floating point number
iii) Complex numbers

2.10.1 Integer type

Int represent signed whole numbers. Represent both positive & negative numbers. Ex: -2147,483,647 & Zero is written as just 0

To write decimal (base 10) first digit not zero ex. 25000

To write octal (base 8) 0 to 7, ex 0o77

hexadecimal (base 16) '0x' or '0X' ex. 0x77

binary (base 2) '0b' or "0B". ex 0b101

2.10.2 Floating point type.

Represents numbers with fractional part. has a decimal point and a fractional part.

Ex. 3.0 or 3.17 or -28.72 stored in 3 parts

i) A sign + or -

ii) A mantissa

iii) An exponent.

2.10.3 Complex type

It is an immutable type that holds a pair of floats, one representing the real part and the other representing the imaginary part of a complex number. complex numbers are written with the real and imaginary parts joined by a + or - sign imaginary part followed by a j. Ex: 5+14j

2.10.4 Boolean type:

It represents special values "True" and 'False' they are represented as 1 and 0 can used numeric expressions as value. ⇒ relational operator.

Ex: 2<3 is True.

2.10.5 String Type:

Sequence of character surrounded by quotes, the character may be alphabets. digits or special characters including spaces. In python string values are enclosed in single quotes, double & triple quotes.

'HELLO', "HELLO", """Hello every one welcome to Python Programming"""

2.10.6 List Type:

List is a sequence data type, sequence of items separated by commas & closed in square brackets [].
The value in a list are called elements or items.

Ex >>> Address = ['131', 'Casmal Nagar', 'Nagercoil', 629004]
>>> print Address
['23', 'Casmal Nagar', 'Nagercoil', '629004].

2.10.7 By Tuples.

A tuple is another sequence data types similar to a list. A tuple consists of a sequence of items separated by commas & items are enclosed in parenthesis().

Ex tuple with string values

>>>T = ('sun', 'mon', 'tue')
>>> print T
('sun', 'mon', 'tue')

tuples with single character

>>>T = ('P', 'Y', 'T', 'H', 'O', 'N')
>>> print T
('P', 'Y', 'T', 'H', 'O', 'N')

2.10.8 Sets:

Set is an unordered collection of items separated by commas & items are enclosed in curly brackets{}.
Set is similar to list. Cannot have duplicate entries

2.10.9 None Type:

It is a special type of single value. No special operation and it is neither False nor 0 (zero)

Ex drink = "available"
food = None
def menu(x):
    if x == drink:
        print(drink)
    else:
        print(food)
menu(drink)
menu(food)

Output

Available
None.

2.11 Variables.

Variable is an Identifier which holds a value. In Programming a value is assigned to a variable. variable can be used to define a name of an identifier. An identifier is a name used to identify a variable function, class, module or other object.

Rules for naming variables.

variable names can be uppercase Letters or lower case Letters.

variable names can be of any Length. they can contain both Letters and numbers, but they can't begin with a number.

No blank spaces are allowed between variable name.

Variable name are case-sensitive.

Ex: Roll_no, GrossPay, a1, Salary.

Creating variable:

The assignment statement (=) assigns a value to a variable. The usual assignment operator is =

Ex: `Variable = expr`

2.12 Expressions:

In Python, most of the lines or statements are written in the form of expressions. expressions are made up of operator & operands.

Ex: A * B + C *, + are operators, A, B & C operands.

type of expressions:

1) Based on the position of operators in an expression. they are.

i) Infix expression: a = b + c

ii) Prefix expression: a = + b c

iii) postfix expression: a = b c +

Based on the type of the result obtained on Evaluating an expression.

i) Arithmetic Expression : +, -, *, /

ii) Relational / Condition Expressions: <, >, <=, >=,

iii) Logical Expression : and or, or not. True or False

iv) Conditional Expression: or relational expression then expression is also called branching (if, if-else) Loops (while) statement. either True or False.

2:13 STATEMENTS:

A statement is an instruction that the Python interpreter can execute. There are two kinds of statements

2:13:1 i) Assignment statement

associates the name to the left of the '=' symbol with the object denoted by the expression to the right of the '=' symbol.

ex >>> Name = 'Jovita' >>> Name 'Jovita'

>>> Age = 9 >>> Age 9

ii) simultaneous Assignment Statement

Python permits any number of variables to appear on the Left side, separated by commas. the same number of expressions must be appear on the right side again separated by commas.

2:13:2 print statement.

Each values separated by commas. Each value is converted into a string (by implicitly invoking the str function) and then printed.

```
>>> name = input("Enter your name:")
Enter your name: Stephy
>>> print ("Hello", name, "!")
Hello Stephy.
```

2:14 TUPLE

In Python, a tuple may be defined as a finite ordered list of numbers or strings. A tuple is similar to a LIST and it contains immutable sequence of values separated by commas. (max(), min() value stored.

Ex >>> max (5,8,9) 9

Creating a tuple

i) Single element, a comma is to be included at the end >>> t1 = 'a',

ii) the built-in function tuple() with no arguments creates an empty tuple >>> t = tuple() >>> t ()

iii) If the argument is a sequence (string) >>> t = tuple ('Jesus') >>> t ('J', 'e', 'S', 'u', 'S')

2:15 Tuple Assignment

is an assignment with a sequence on the right side & a tuple of variables on the left. the right side is evaluated and then its elements are assigned to the variables on the left.

>>> T1 = (10, 20, 30) >>> print T1 (10, 20, 30)

2:16 Precedence of operators (order of operators)

Evaluation of expression is based on Precedence of operators.

- Parantheses have the highest precedence and can be used to force an expression to evaluate in order Ex 5 * (9-3) = 30

* Exponentiation has the next highest precedence

Ex 1 + 2 * 3 = 9, not 27, 2 * 3 * 2 = 18 not 36

* Multiplication and Division have higher precedence than Addition & subtraction

Ex 2 * 3 - 1 = 5 not 4 6 + 4/2 = 8 not 5

* Operators with the same precedence are evaluated from Left to right

2:17 Comments.

Contains information for persons reading the program. It easily readable and understandable by the programmer than programs who are developing.

In Python comments starts with # (hash sign) Everything following the # till the end of that line is treated as a comment & interpreter.

Ex #swap.py
#Swapping the values of a two variable program
written by G. Sujatha, April 2018.

2:18 Mutable and immutable Types:

Sometimes we may require changing or updating the values of certain variables used in a program.

Mutable (changeable) type:

Variables whose values can be changed after they are created and assigned are called mutables or an object whose state or value can be changed in place is said to be mutable type.
ex List

Immutable (unchangeable) type:

Variable whose values cannot be changed after they are created and assigned are called immutable type. Integers, Float, Boolean, Complex, String, tuples & sets are immutable data type.

2:19 Indentation:

Whitespace at the beginning of the line is called identation. These whitespaces or the indentation are very important in python.

program: Illustration of identification (voting.py)

```
age = int (input ("Enter age:"))
if (age>18)
    print ('Eligible for voting')
else
    print ('Not Eligible for voting'):
```

Python not use curly braces {...}

2:20 INPUT FUNCTION

2:20:1 input() Function

The purpose of input() function is to read input from the standard input (the keyboard by default) using this function, we can extract integer or numeric values. Syntax: <variable> = input ([prompt])

Here..

The prompt is an optional parameter.

The prompt is an expression that asks to prompt the user for input & the results of input() function is returned a numeric value. & assigns to a variable.

Result is prompt on the screen.

2:20: 2 raw_input()

The purpose of raw_input() function is to read input from the standard input stream (the keyboard, by default). using this function, we can extract string of character (both alphanumeric and special).

Syntax: <variable> = raw_input ([prompt])

Here: This function reads a line from input (the keyboard) and returns a string stripping a newline.

It always returns a string.

Ex: name = raw_input ("Enter name")

Address = raw_input ("Enter Address")

illustration of raw_input function.

Num1 = int (raw_input ("Enter first number"))

Num2 = int (raw_input ("Enter second number"))

print "The sum of ", Num1, "and", Num2, "is", Num1+Num2

Output:

Enter first number: 24

enter second Number: 67

The sum of 24 and 67 is 91

ILLUSTRATIVE PROGRAMS

1. Exchange the values of two variables

```
def exchange(x,y):                    # function Definition
    x,y = y,x                         # swapping using tuple assignment
    print("After exchange of x,y")
    print("x = ", x)
    print("y = ", y)
x = input("Enter value of x")         # main Function
y = input("Enter value of y")
print("Before exchange of x,y")
print("x =" x)
print("y-", y)                        # Function call
exchange(x,y)
x = input("Enter value of x:")
y = input("Enter value of y:")
print("Before exchange of x,y")
print("x =", x)
print("y =", y)
temp = x
x = y
y = temp
print("After exchange of x,y")
print("x =", x)
print("y =", y)
```

output

```
Enter value of x: 67
Enter value of y: 56
Before exchange of x,y
x = 67
y = 56
After exchange of x,y
x = 56
y = 67
```

2. Circulate the values of n variables cutreau using slicing technique.

```
def circulate(A,N):
    for i in range(1, N+1):
        B = A.pop(0)
        A.append(B)
        print("circulation",i,"=",A)
    return
```

```
def circulate(A,N):
    for i in range(1, N+1):
        B = A[i:] + A[:i]
        print("circulation",i,"=",B)
    return
A = [91,92,93,94,95]
N = int(input("Enter n:"))
circulate(A,N)
```

output

```
Enter n:5
circulation 1 = [92,93,94,95,91]
circulation 2 = [93,94,95,91,92]
circulation 3 = [94,95,91,92,93]
circulation 4 = [95,91,92,93,94]
circulation 5 = [91,92,93,94,95]
```

3. Finding distance between two points.

```
import math
x1 = int(input("Enter a x1:"))
y1 = int(input("Enter a y1:"))
x2 = int(input("Enter a x2:"))
y2 = int(input("Enter a y2:"))
distance = math.sqrt((x2-x1)**2) + ((y2-y1)**2))
print("Distance =", distance)
```

output

```
Enter a x1:3
Enter a y1:2
Enter a x2:7
Enter a y2:8
Distance = 7.211102550927978
```

Thank you

UNIT-III

CONTROL FLOW, FUNCTIONS, STRINGS

Conditionals : Boolean values and operators, conditional (if), alternative (if-else), Chained conditional (if-else-if); iteration: state, while, for, break, Continue, Pass; Functional functions: return values, Parameters, local and global scope, function Composition, recursion; strings: string slices, immutability string functions and methods, string module; lists as arrays: illustrative Programs: Square root, gcd exponentiation, sum an array of numbers, Linear search, binary search.

3:1 BOOLEAN VALUES

A boolean expression is an expression that is either true or false.

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

the most common way to produce a Boolean value is with a relational operator.

the relational operators in python are,

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

There is no =< or => operators in python.

= is an assignment operator == is relational operator.

ex: >>> 10==10 True >>8==6 False.

3:2 OPERATORS.

An operator is a symbol that represents an operation that may be performed on one are more operands.

Operands: the value that the operator operates on is called the operand.

Ex - x=a+b, a, b, x are operands, =, + are operators

3.3 CLASSIFICATION OF OPERATORS

3.3.1. Unary operators.

Unary operators operate on only one operand. Its usually precede their single operand.

the unary plus + and unary minus − operators are used to provide sign to the operands ex +6, −9

the bitwise inverse operator changes every bit i.e 0 becomes 1 and 1 becomes 0.

Since the way of negative numbers are stored in the computer. it changes a positive value to negative and a negative value to positive.

3.3.2 Binary operators.

the binary operators in Python are grouped into

1) Arithmetic operators 2) Relational operators
3) Logical operators 4) bitwise operators
5) Assignment operators 6) special operators
7) Identity operators 8) Membership operators.

Arithmetic operators:

are used to perform mathematical operations like addition, substraction, multiplication etc..

Operator	Meaning	Example
+	Add two operands	x+y print('x+y=',x+y)
−	Subtract right operand from the left	x-y print('x-y=',x-y)
*	multiply two operands	x*y
/	Divide left operand by the right one (always results into float)	x/y
%	Modulus: remainder of the division of left operand by the right	x%y (remainder of x/y)
//	Floor division or Truncating division: division that results into whole number adjusted to the left in the number line	x//y
**	Exponent: Left operand raised to the power of right	x**y (x to the power y)

2. Relational operators / comparison operators.

The operators used to do comparision are called relational operators. always returns a Boolean value (True/False). in python non zero value is treated as true & zero value is treated as false.

Operator	Meaning	Example
>	Greater than - True if left operand is greater than the right	x>y print('x>y' is x>y)
<	Less than true if left operand is less than the right	x<y
==	Equal to - true if both operands are equal x==y	x==y
!=	not equal to - true if operands are not equal	x!=y
>=	Greater than or equal to -	x>=y
<=		x<=y

3. Logical Operators

Logically relate the two expressions. 'and' 'or' and 'not' operators. It operate according to the truth table. Used with if & while keywords.

Operator	Meaning	Example.	x = true, y = False
and	True if both true	x and y	print('x and y is' x and y) False
or	True if either one is	x or y	print('x or y is' x or y) True
not	true if operand is false	not x	print ('not x is', not x) False

4. Bit-wise operators.

Decimal numbers are natural to humans whereas binary numbers are native to computer. work with bit of binary numbers. It operates bits by bit, 2 is 010 this '11', & bitwise AND, | bitwise OR, ~ NOT, ^ XOR >> bitwise right shift. << bitwise left shift.

5. Assignment Operators

are used to assign values to a variable.

a = 5 is a simple assignment operator. a+=15 then a=a+15

=5, +=5, -=5, *=5, /=5, %=5, //=5, **=5, &=5, |=5, ^=5, >>=5, <<=5

X=5 X=X+5 X=X-5 X=X*5 X=X/5 X=X%5 X=X//5 X=X**5

6. Special Operators.

i) Identity Operators, ii) Membership Operators

a) Identity operators: are used to determine whether the value of a variable is of a certain type or not.

IS and is not are identity operators in Python they are used to check whether two values of a variables are located on the same part of the memory or not. Two variables that are equal do not imply that they are identical.

Operator	Meaning	Example
is	True if the operands are identical	X is true
is not	True if the operands are not identical	X is not true.

b) Membership operators: are used to test whether a value or variable is found in a sequence (string, List, tuple, set & dictionary) In and not in are the membership operators in Python. In a dictionary, they test for the presence of key not value

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

x = 'Hello'
y = {1:'a',}
print('H' in x)
print('A' in y)
print('hello' not in x)
True, True

3.4: CONDITIONAL EXECUTION

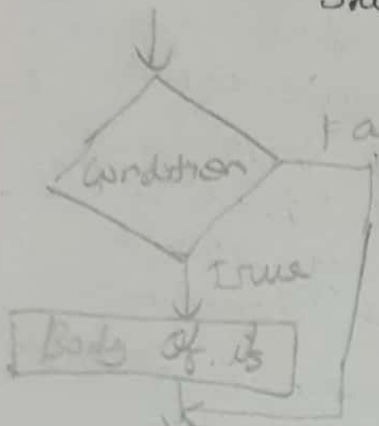
To write useful programs, it is needed to check the condition and change the behaviour of program they allow executing statements selectively based on certain decisions. Such type of decision control statements are known as selection control statements or conditional branching statements, they are

i) Conditional Execution (if)
ii) Alternative Execution (if-else)
iii) Chained conditional Execution (if-elif-else)
iv) Nested Conditionals.

3.4.1. Conditional Execution (if)

The if statement is the simplest form of decision control statement that is frequently used in decision making. The if statement condition is true, otherwise the action is skipped.

Syntax
```
if condition :
    Statements(s)
```



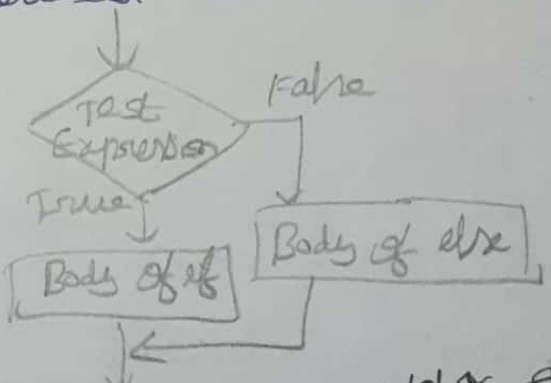

```
age = 16
if age <= 21:
    print("you are a minor.")
```

→ : the colon is required at the end of the condition
→ the body of the if statement is indicated by the indentation

3.4.2 Alternative Execution (if-else)

The second form of the if statement is "alternative execution" If-else statement evaluates the condition and execute the true statement block only when the condition is true. If the condition is False, false statement block is executed. It uses separate blocks.

Syntax
```
if condition:
    Body of if
else:
    Body of else
```



ex/ Programs if checks if the number is odd or even oddeven.py

```
num = int(input("Enter a number:"))
if num % 2 == 0
    print("number is even")
else:
    print("number is odd")
```

Output
```
enter a number: 8
number is even
```

```
Enter a number 9
number is odd.
```

3.4.3 Chained Conditional Execution (if-elif-else)

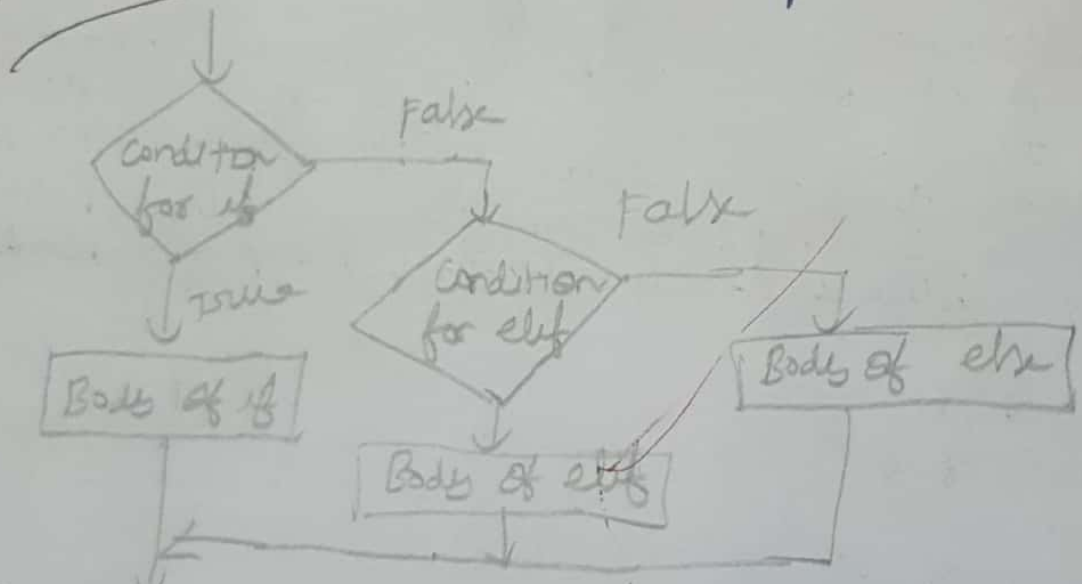
Sometimes there are more than two possibilities and need more than two branches. elif statement allows checking multiple expressions for truth value and executing a block of code as soon as one condition evaluates to true.

Syntax

```
if condition:
    Body of if
elif condition:
    Body of elif
else
    Body of else
```

```
average = input ("Enter the average mark!!")
if average >80:
    print "Grade is A"
elif average >70:
    print "Grade is B"
elif average >60:
    print "Grade is C"
else:
    print "Grade is D"
```

flow Chart



3.4.4 Nested Conditionals.

→ There may be a situation to check for another condition resolves to true. In such a situation the nested if construct can be used.

→ One conditional can also be nested within another.

→ Any number of these statements can be nested inside one another.

→ Indentation is the only way to figure out the Level of nesting.

Syntax

```
if condition 1:
    Statement (s)
if condition 2:
    Statement (s)
. . .
. . .
else
    Statement (s)
elif condition n:
    Statement (s)
else :
    Statements
```

program to compare two numbers (compare.py)

```
x = int(input ("Enter x : "))
y = int (input ("Enter y : "))
if x == y:
    print ('x and y are equal')
else :
    if x < y:
        print ('x is less than y')
    else:
        print ('x is greater than y')
```

enter x : 9
enter y : 4
x is greater than y

3.5 ITERATION or repetition structure

Python allows a block of statements to be repeated as many times as long as the processor could support this generally called Looping.

It causes one or more statements on a block to be executed repeatedly.

An iteration structure allows the programmer to perform an action which is to be repeated until the given condition is true.

3.5.1 State

An iteration table is a way to visualize the states of the iteration.

The columns of the iteration table are the state variables involved in the iteration and the rows are the values of the state variables on each invocation of the iteration.

Ex: $L1 = [1, 2, 3, 4]$ the iteration table for reverse [L]

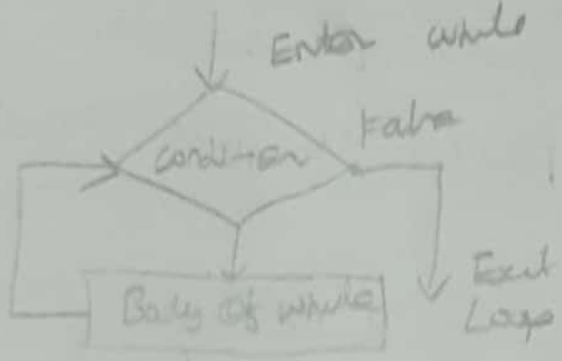
List	Result
[1, 2, 3, 4]	[]
[2, 3, 4]	[1]
[3, 4]	[2, 1]
[4]	[3, 2, 1]
[]	[4, 3, 2, 1]

List we are reversing
result we are building

3.5.2 while loop

The while loop in python repeat one or more statements as long as the particular condition is true. It is used if the number of times to iterate is not known before.

Syntax:
```
while condition:
    Body of while
```



ex program to add natural numbers upto n (sum of 1 to n)

```
# sum = 1+2+3+...+n
n = int(input("Enter n:"))
sum = 0
i = 1
while i<=n:
    sum = sum + i
    i = i+1
print("The sum is", sum)
```

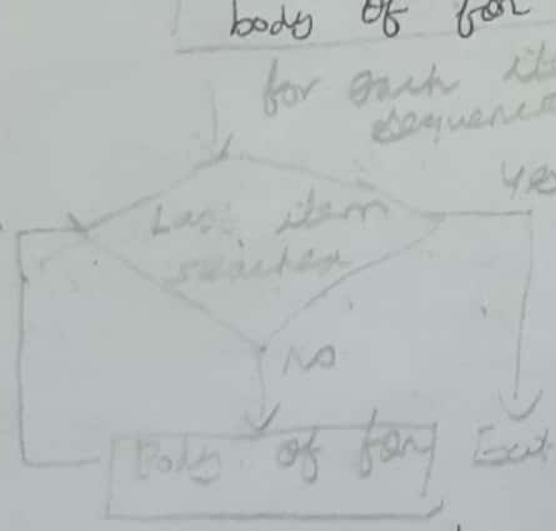
enter n = 10
the sum = 55

3.5.3 for loop

The for statement is used to iterate over a range of values or a sequence. iterating over a sequence is called traversal.

Here value can be numeric, string, list, or tuple.

Syntax:
```
for val is sequence:
    body of for
```



ex/ sum of all numbers stored in a list

```
# List of numbers
numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]
sum = 0
for val in numbers:
    sum = sum + val
print("The sum is", sum)
```

output: the sum is 48

Using range() function

⇒ Python has built-in function range(), which is most often used in for loops. It is used to create a list containing a sequence of integers from the given start value up to stop value

⇒ arguments must be integers.

Syntax:

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

Start & Step are optional, Start is the starting number of range. If the start argument is omitted, it defaults to 0. Stop is the last number of range where the range will end.

ex// for i in range (5,10):
print (i)

output
5
6
7
8
9

3.5.4 break

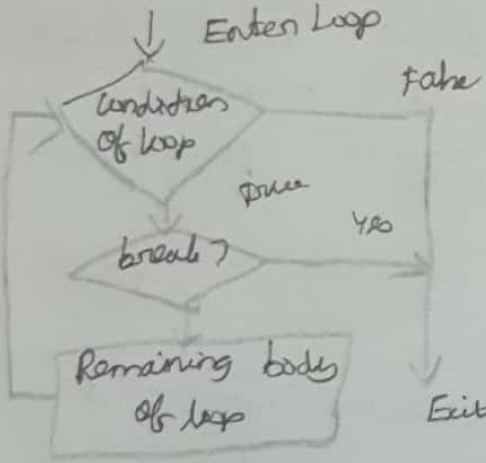
Loop iterates over a block of code until test expression is false, but sometimes we wish to terminate the current iteration or even the whole loop without checking the test expression

If break statement is inside a nested loop (loop inside another loop) 'break', will terminate the innermost loop

Syntax: break

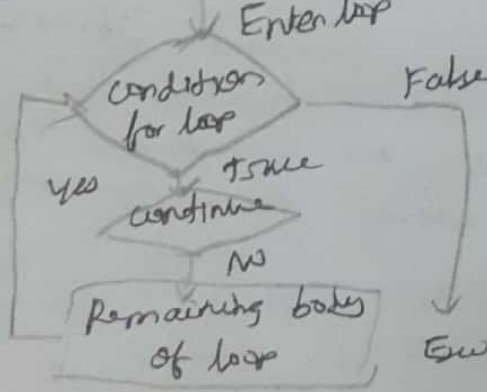
ex
num = 10
while num > 0:
print ('current value:' num
num = num - 1
if num == 6
break

output
current value: 10
current value: 9
current value: 8
current value 7



3.5.5 continue

The continue statement is used to skip the remaining part of the statement in the current loop and start with the next iteration of the loop. that means it will help to skip a part of the loop. Continue



ex/ for val in 'computer':
if val == 'p':
continue
print (val)
print ("The end")

output
C
O
m
p
U
e
r
The end.

3.5.6 Pass

In some cases, loop or a function is not implemented currently but would be wanted in the future. They cannot have an empty body because the interpreter would generate an error message. Pass statement can be used in 'if' clause as well as within loop construct, when you do not want any statement or commands within the block to be executed syntax

```
pass
```

3.6 FUNCTIONS

In programming the use of function is one of the way to achive modularity and reusability. function can be defined can be defined as a named group of instructions that accomplish a specified task when it is invoked. a function can be called repeatedly from different places of the program without writing all the code of the function everytime.

A function can be called from inside another function by simply writing the name of the function and passing the required parameters.

Advantages of Function.

- Increases readability
- Reduces code Length
- Increases reusability
- work can be easily divided among team members and completed in Parallel.

Types of functions.

- built-in function
- user defined function

built in function

Python has a very extensive standard Library It is a collection of many built in functions that can be called in the program as and when required, thus saving programmer's time of creating these commonly used functions every time.

Built-in-Functions

Input or output	Datatype conversion	mathematical Function	Other Functions
input()	bool()	abs() absolute	import_()
print()	chan()	divmod()	range()
	dict()	pow().	type()
	int()	sum()	len()
	float()		
	set()		

3.6.2: User defined Functions:

In addition to the standard library functions we can define our own functions while writing the program. Such a functions are called user defined functions. the user can understand the internal working of the function and can be changed and modified.

3.7: ELEMENTS OF USER DEFINED FUNCTIONS

It contains two elements.

3.7.1: Function definition:

A function begins with def (short for define). A function definition specifies the name of a new function and the sequences of statements that run when the function is called.

Syntax

```
def function_name(parameters): // function header
    statements(s)              // function body.
```

Elements of function definition

- function header
- Function body

Function header

A first line of a function definition is the function header.

Starts with the keyword def, followed by function name. followed by a comma-separated list of identifiers called formal parameters or simply parameters.

function header is ended with a colon

Function body:

function body follows the function header. It contains one or more valid Python statements or function instructions. The statements must be indented relative to the function header. The statements outside the function indentation are not considered as part of the function.

Example:
```
def great(name):
    print("Hello," + name + ". Good morning!")
```

3.7.2 Function Call:

Is a statement that calls a function. It contains of the function name followed by an argument list of parenthesis.

Once function is defined, it can be called from another function, program or even the Python prompt.

3.8 FRUITFUL FUNCTIONS

Functions that return values are called as fruitful functions. In many programming language a function that doesn't return any value is called a procedure.

3.8.1 Return Statement.

The calling function generates a return value, which is usually assigned to a variable or used as a part of an expression.

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

Syntax: `return [expression]`

3.8.2 Return None: (return nothing)

If there is no expression in the statement or the return statement itself, then the function will return the None object.

3.8.3 Return values:

The return value may or may not be assigned to another variable in the caller. It immediately exits that function. Therefore any code written after the return statement is never executed.

3.8.4. Return fruitful return:

In a fruitful function the return statement includes an expression. Fruitful return means "Return immediately from this function and use the following expression as a return value.

3.9 PARAMETERS AND ARGUMENTS

Parameters:
A name used inside a function to refer to the value passed as an argument. Parameters are specified within the pair of parenthesis in the function definition and are separated by commas

Arguments:
An argument is a value passed to the function during the function call which is received in corresponding parameter defined in function header

```
                    Parameters
def sum (a,b):      # Function Definition
    s = a+b
    print ("s = ", s)
x = 10
y = 10
sum (x,y)           # Function Call
    |               Arguments
```

Arguments are used to call a function and there are four main types of defining function arguments they are.

Required Arguments
Keyword Arguments
Default Arguments
Variable Length Arguments

3.9.1 Required Arguments

The arguments are passed to a function in correct positional order. The number of arguments in the function call should exactly matches with the number of arguments specified in the function definition.

Ex
```
def area (length, breadth):
    A = length * breadth
    Print ("Area = ", A)
area (4, 5)
```
Output : 20

Ex
```
def printstring (num, str):
    Print (num, str)
    return
printstring (5, "Star")
```
output 5 Star

3.9.2 Keyword arguments.

are used when a function is called with a long parameter list where most of the parameters have default values.

Python also allows to skip arguments according the order of arguments can be changed.

the values are not assigned to arguments according to their position but based on their names. but all the keyword arguments should match the parameters in the function definition.

Ex
```
def namelist (name, degree, dept):
    print ("Name :" Name)
    print ("Degree :" degree)
    print ("Department :" dept)
namelist (name = "Abi", degree = "B.E", dept = "CSE")
```
Output:
Name: Abi
Degree: BE
Department : CSE

3.9.3 Default Arguments

Python allows users to specify function arguments that can have defaults values. this means that a function can be called with fewer arguments.

The default value to an argument is provided by using the assignment operator (=). User can specify a default value for one or more arguments.

The default arguments must be written after the non-default arguments. This is non non-default argument can not follow default arguments

```
def display (name, dept = "CSE", rollno = 15):
    print ("Name:", name)
    print ("Department:", dept)
    print ("Roll_No:", rollno)
display ("Arun")
```

output

```
Name: Arun
Department: CSE
Roll_No: 15
```

3.9.4 Variable Length Arguments

In some situation, it's not known in advance how many arguments will be passed to a function.

In such cases, Python allows programmers to make function calls with arbitrary number of arguments. function definition uses (*) asterisk before parameter name

ex

```
def display (name, *friends):
    print ("Name:", name)
    print ("Friends:", friends)
```

output

```
display ("Arun", "Bala", "Cindia", "Dinesh", "Elsa", "Freeda")
Name: Arun
Friends: ('Bala', 'Cindia', 'Dinesh', 'Elsa', 'Freeda')
```

3.10 LAMDA FUNCTIONS OR ANONYMOUS FUNCTIONS

In Python, anonymous function is a function that is defined without a name. In python normal functions are defined using the `def` keyword. anonymous functions are defined using the `lambda` keyword.

lambda [arguments] : expression

Lambda function is mostly used for creating small and one time anonymous function.

mainly used in combination with the functions like filter(), map() and reduce()

Lambda function can take any number of arguments and must return one value in the form of an expression. variables in its parameter list.

Ex
```
sum = lambda x, y : x+y
print ("The sum is", sum(30,40))
print ("The sum is:", sum(-30,40))
```
output
The sum is: 70
The sum is: 10

lambda 7.0: x=0 can be written as def sum(x,y):
return sum

3:)) SCOPE

Scope of variable refers to the part of the program, where it is visible. variable in a program may not be accessible at all locations in that program. this depends on where a variable is declared

there are two types of scope of variable
Local scope - declared with in a function
Global scope - declared main program.

Local variable.

A variable defined in a function body has a local scope. It can be created by defining a variable inside a function definition. the same names used outside the function. i.e variable names are local to the function. only exists the function is executed

Ex
```
x= 50
def add():
    x=30
    print(value of x inside add is:'x')
    x= x+10
    Print (value of x inside add is:'x)
Print (value of x is :'x)
add()
Print after function call value x in main is:'x)
```

Global
```
x=10
y=30
def sum():
    z=x+y
    print(z)
sum()
```

```
def sum():
    x=y=30
    print(x)
```

output
value of x is: 50
value of x inside add:30
value of x inside add:40
After function call value of x in main:50

3.11.2 Global scope:

A variable defined outside a function body has a global scope. It can be created by defining a variable outside of any function / block.

It is accessible anywhere in the program. Any modification to global variable is permanent and visible to all the functions written in the file.

Eg:
```
x=50          (here x is the Global variable)
def add(x):
    x+=10     (here x is the local variable)
    print('Inside add x is', x)
print('Main : Value of x is' x)
add(x)
```
Output: Main : Value of x is 50
Inside test x is 60.

3.12 FUNCTION COMPOSITION

The value returned by a function may be used as an argument for another function in a nested manner. this is called composition.

Composition is the ability to take small building blocks (variables, expressions and statements) and compose them.

Eg: `x = math.sin(degrees/360.0 * 2 * math.pi)`

Here the argument of a function can be any kind of expression, including arithmetic operators and function calls.

3.13 RECURSION

A function that calls itself is recursive.

The process of executing it is called recursion.

Recursion can be used to solve the problems that can be expressed in terms of similar problems of smaller size.

Advantage of Recursion:

A complex task can be broken down into simpler sub problems using recursion.

Sequence generation is easier with recursion rather than using some nested iteration

Ex: Finding factorial of a number: $n! = n \times (n-1)$

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

output: Enter a number: 4
The factorial of 5 is 24

4*fact(3) ⇒ 4*3*fact(2)
4*3*2*fact(1) ⇒
4*3*2*1
4*3*2 ⇒ 4*6

3.13.1 Stack Diagrams for recursive functions

It represents the state of a program during a function call. It helps to interpret a recursive function.

Every time a function gets called, python creates a frame to contain the function's local variables and parameters.

More than one frame on the stack at the same time.

Ex: program to calculate GCD using recursive function. Greatest common divisor

$$GCD(a,b) = \begin{cases} b, & \text{if } b \text{ divides } a \\ GCD(b, a \bmod b) & \text{otherwise} \end{cases}$$

```
def GCD(x,y):
    rem = x%y
    if (rem ==0):
        return y
    else:
        return GCD(y,rem)
n=int(input("Enter the first number:"))
m = int(input("Enter the second number:"))
print("The GCD of numbers is", GCD(n,m))
```

Output

Enter first number: 55
Enter second number: 5
The GCD of number is: 5

the Fibonacci series:

$$FIB(n) = \begin{cases} 1, & \text{if } n <= 2 \\ FIB(n-1) + FIB(n-2), & \text{otherwise} \end{cases}$$

```
def fibonacci (n):
  if (n<2):
     return 1
  return (Fibonacci (n-1) + Fibonacci (n-2))
n = int (input ("Enter the number of terms: "))
for i in range (n):
    print ("Fibonacci (" i ") = ", Fibonacci (i))
```

Output:

```
Enter the number of terms: 5
Fibonacci (0) = 1
Fibonacci (1) = 1
Fibonacci (2) = 2
Fibonacci (3) = 3
Fibonacci (4) = 5
```

3.13.2 Infinite recursion:

If a recursion never reaches a base case, it goes on making recursive calls forever and the program never terminates. This is known as infinite recursion and it is generally not a good idea.

```
def recurse():
    recurse()
```

3.14 STRINGS:

String is a sequence which is made up of one or more characters. Here the character can be letter, digit, whitespace or any other symbol.

In python strings can be created using single quotes, double quotes and triple quotes. Using triple quotes, strings can span several lines without using the escape characters.

```
>>> str1 = 'python is simple & easy language'
>>> str2 = "python is free & open source software"
>>> str3 = '''python is a high level language'''
```

3.14.1 Accessing String Elements.

Each individual character in a string can be accessed using a technique call indexing.

The index specifies the character to be accessed in the string and is written in square brackets ([])

If we give index value of the range then we get an Index error. the value must be integer (positive, zero or negative)

3.14.2 Traversing a string.

traversing a string means accessing all the elements of the string one after the other by using the index. A string can be traversed using: for loop or while loop.

Traversal with a for loop

→ On Execution of the for loop, the characters in the string are printed till the end of the string is not reached.

ex
```
>>> bird = 'parrot'
>>> for letter in bird:
        print (letter)
```

output
```
p
a
r
r
o
t
```

Traversal with a While loop

The len() function calculates the length of the string. on entering the while loop. the interpreter checks the condition.

ex
```
>>> bird = 'parrot'
>>> i = 0
>>> while i < len (bird):
        letter = bird [i]
        print (letter)
        i = i + 1
```

output
```
p
a
r
r
o
t
```

the loop traverse the string and displays each letter on a line by itself.

3:15 STRING SLICES

Accessing some part of a string or substring is known as slicing. This can be done by specifying an index range. Subsets of strings can be taken using the slice operator with the indices in square brackets separated by a colon ([:] and [::])

Ex
```
>>> a = 'Python programming'
>>> a[0:5]
'Pytho'
>>> a[:5]
'Pytho'
>>> a[5:]
'n programming'
>>> a[5:10]
'n pro'
>>> a[:]
'Python programming'
```

Ex
```
>>> a = 'Python programming'
>>> a[0:10:2]
'Pto n'
>>> a[0:10:3]
'Ph o'
```

3:16 STRINGS are Immutable.

A string is an immutable data type. It means that the contents of the string cannot be changed after it has been created, but a new string can be created from the variables on the original string.

Ex
```
>>> word = 'read'
>>> word = 'b' + word[:2] + 'a' + word[2:]
>>> word
'bread'
```

Ex
```
String = "python"
new_string = string.replace('p','P')
Print ["string = ", string]
Print ("new string =", new_string)
```

Output
```
String = python
newString = Python.
```

3:17 STRING FUNCTIONS And methods

Strings provide methods to perform a variety of useful operations. A method is similar to a function. It takes arguments and returns a value. Some of the functions are listed in the table.

S.capitalize()	Capitalizes first letter of S
S.capwords()	Capitalizes first letter of each word in S
S.count(sub)	Count number of occurrences of sub in S
S.index(sub)	Find first index of sub in S, or raise value Error if not found
S.rfind(sub)	Same as find, but last index
S.rindex(sub)	Same as index, but last index
S.lower()	Convert S to lower case
S.split()	Return a list of words in S
S.join(lst)	Join a list of words into a single string with S as separator
S.strip()	Strip leading/trailing white space from S
S.upper()	Converts S to upper string
S.replace(old,new)	Replace all instances of old with new in string.

3:17.1 UPPER

The method upper takes a string and returns a new string with all uppercase letters. The method call is called an invocation. So it is invoking 'upper' on 'word'.

2. Lower:

The method 'lower' takes a string and returns a new string with all lowercase letters

Upper

```
>>> word = 'Python'
>>> newword = word.upper()
>>> newword
'PYTHON'
```

Lower

```
>>> word = 'PYTHON'
>>> newword = word.lower()
>>> newword
'python'
```

3.17.3 Capitalize.

the capitalize function capitalizes the first character of s

3.17.4 Split.

the split function strip leading or trailing white space from a string.

3.17.5 Find.

the find method can find substrings in a string.

Ex Capitalize

```
>>>s = 'python'
>>>s.capitalize()
'Python'
```

Split

```
>>>s='python program'
>>>s.split()
['python', 'program']
```

Find.

```
>>> word='python programming'
>>>newword=word.find('pro')
>>>newword
7
```

3.17.6 The in operator

the word in a boolean operator that takes two strings and returns. 'True' if the first string appears as a substring in the second.

3.17.7 String concatenation.

concatenation means to join. python allows us to join two strings using concatenation operator plus which is denoted by symbol +

3.17.8 Repetition.

Python allows us to repeat the given string using repetition operator which is denoted by symbol *.

Ex In

```
>>> 'ra' in 'parrot'
False
>>> 'ro' in 'parrot'
True
```

String concatenation

```
>>>first='problem'
>>>second='solving'
>>>first+second
'problemsolving'
```

Repetition.

```
>>>a='Hello'
>>>a*5
'HelloHelloHelloHelloHello'
```

3:17:9 String Length

the len function returns the number of characters in a string.

Ex

```
>>>s='problem solving'
>>>len(s)
15
```

3:17:10 String Comparison

The relational operators work on strings. The relational operators are used to compare two strings.

Ex: if 'python' == 'python':
    Print ("Both string are equal")
Both string are equal.

3:18 STRING MODULE

The string module provides additional tools to manipulate strings. Some methods available in the standard data structure are not available in the string module (Ex: isalpha)

3:19 Lists as arrays.

Creating fixed size list is similar to creating arrays in other programming languages. The typical python solution is to use repetition operator and a list containing the value none.

Ex: >>> a = [None] * 5 [None, None, None, None, None]
>>> a

Ex: >>> data = [0] * 5.

3:20 Illustrative programs.

1. Square root:

num = input ("Enter a number:")
number = float (num)
Square root = number ** 0.5
Print ("Square Root of %0.2f is %0.2f" %
(number, Squareroot))

output

Enter a number: 10
Square Root of 10.00 is 3.16

2. Write a program to calculate GCD using recursive functions.

```
def GCD (x,y):
    rem = x % y
    if (rem == 0):
        return y
    else:
        return GCD (y, rem)
n = int (input ("Enter the first number :"))
m = int (input ("Enter the second number :"))
print ("the GCD of numbers is ", GCD (n,m))
```

Output

```
Enter first numbe : 50
Enter second numbe : 5
the GCD of numbers is =5
```

3. Exponentiation of a number.

```
import math
num = int ( input ("Enter a number :"))
ex = math. exp (num)
print ("Exponentiation =", ex)
```

Output

```
Enter a number : 50
Exponentiation = 5.184705528587072e+21
```

4. Sum of arrays of numbers.

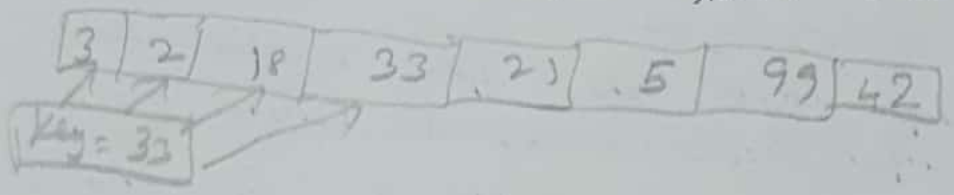
```
A = [0, 0, 0, 0, 0, 0, 0, 0]
Sum = 0
n = int (input ("Enter a number :"))
print ("Enter a numbers")
for i in range (0, n):
    A[i] = int (input ())
for i in range (0, n):
    Sum = Sum + A[i]
print ("Sum = ", Sum)
```

Output

```
Enter a number : 5
Enter n numbers
1
2
3
4
5
Sum = 15
```

5 Searching a Key Value in an array (Linear Search)

Linear search is the simplest searching technique. The search begins at one end of the list and searches for the required element one by one until the element is found or till the end of the list is reached.




```
A = [0,0,0,0,0,0,0,0,0,0]
found = 0
n = int (input("Enter a number: "))
print ("Enter n numbers")
for i in range [0,n]:
  A[i] = int (input())
Key = int (input ("Enter a key to be searched: "))
  for i in range [0,n]:
  if key == A[i]:
   print ("Key found")
     found += 1
  else:
     continue
  if found == 0:
   print ("Key not found")
```

Output

```
Enter a number: 8
Enter n number:
8
4
7
5
6
3
9
2
```

```
Enter a key to be
  searched: 9
Key found
Enter a number: 8
Enter n numbers.
8
4
7
5
6
8
9
2
Enter a key to be searched:
Key not found.
```

6. Binary Search:

binary search requires the list to be stored in ascending order.

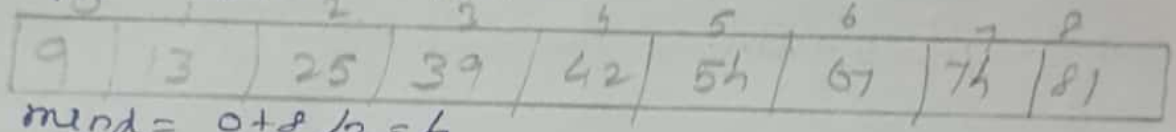
It begins by comparing the element that is present at the middle of the List.

If there is a match then the search ends and the location of the middle element is returned.

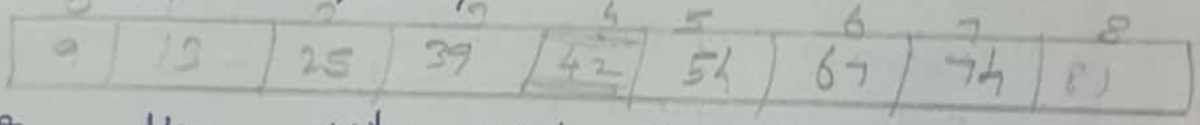
If there is mismatch with the middle element and the search element is less than the middle element then first part of the List is searched otherwise second part of the list is searched.

This process continues until the search element is equal to the middle element or the list contains only one element that is not equal to the search element.

Example



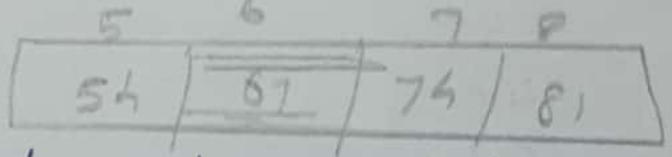
Key = 67 mid = 0+8/2 = 4



Compare Key with mid element

67 # 42 and 67 > 42. do search 2nd list

$mid = \frac{5+8}{2} = 6$



Compare key with mid element (6th element)

67 = 67. do search completed

the key element 67 is found 6th location

Program:

```
def binary search (A, item):
    first = 0
    last = Len(A) - 1
    found = False
    while first <= last and not found
        mid point = (first + last)//2
```

```
if A[midpoint] == item:
    found = True
else:
    if item < A[midpoint]:
        last = midpoint - 1
    else:
        first = midpoint + 1
    if found == True:
        print ("Key found")
x = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
n = int (input("Enter a number:"))
print ("Enter number in stored order")
for i in range (0, n):
    x[i] = int (input ())
key = int (input ("Enter a key to be searched:"))
print (binarySearch (x, key))
```

Output

```
Enter a number: 6
Enter number in sorted order
23
34
45
56
67
78
Enter a key to be searched: 67
Key found.
```

Thank you

Unit IV LISTS, TUPLES, DICTIONARIES:

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

List:

A List is a sequence of values. they can be any type. the value in a list are called elements or items.

4.1.1 Creating a List:

there are several ways to create a new list.

i) the simplest way is enclose the elements in square brackets ([and]).

ex → A list of four integers: [10, 20, 30, 40]
→ A list of three strings: ['Jovita', 'Jevita', 'Varsha']
→ A list of different types: ['march', 2018, 2.6]

ii) A list that contains no elements is called an empty list. It can be created with empty brackets, []. ex empty = []

iii) A list of that is an element of another list is called nested list.

ex ['Dell', 2.0, [50, 100]] a list contains another list.

4.1.2 Assign list values to variables.

list values can be assigned to variables.

ex
```
>>>icreams = ['vanilla', 'strawberry', 'mango']
>>>numbers = [5, 10, 6]
>>> name = ['Dansha']
>>> print (icreams, numbers, name)
['vanilla', 'strawberry', 'mango'] [5, 10, 6] ['Dansha']
```

4.1.3 Accessing List elements:

Accessing the elements of a list is same as for accessing the characters of a string that is using the bracket operator. the expression inside the brackets specifies the index. the indices start at 0.

eg >>> fruits = ['vanilla', 'strawberry', 'mango']
>>> print(fruits[0])
'vanilla'
>>> print(fruits[2])
'mango'

Negative indexing:

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

example
>>> birds = ['peacock', 'dove', 'duck', 'cuckoo']
>>> print(birds[-1])
cuckoo
>>> print(birds[-2])
duck.

4.2 List operations:

4.2.1 Concatenation operation

Concatenation means joining two operands by linking them end to end. In list concatenation, + operator concatenate two lists with each other and produce a third list.

eg >>> a = [1,2,3]
>>> b = [4,5,6]
>>> c = a+b
>>> c
[1,2,3,4,5,6]

eg >>> [1]*5
[1, 1, 1, 1, 1]
>>> [1,2,3]*4
[1,2,3, 1,2,3, 1,2,3, 1,2,3]

4.2.2 Repeat operation:

List can be replicated or repeated or repeatedly concatenated with the asterisk operator "*", the * operator repeats a list a given number of times.

4.3 List slices:

→ An individual element of a list is called a slice. Selecting a slice is similar to selecting an element (s) of a list.

- Subsets of lists can be taken using the slice operator with two indexes in square brackets separated by colon [:] and [m:n].

→ Index from Left 0 1 2 3
List elements pen pencil Rubber Scale
Index from right end -4 -3 -2 -1

ex >>> books = ['pencil', 'Pen', 'duck', 'cuckoo']
>>> print (books[2:4])
['duck', 'cuckoo']

4.4 List methods:

Python provides methods that operate on list.

① append() Add an element to the end of the list
ex >>> t = ['a', 'b', 'c'] >>> t.append('d')
>>> t ['a', 'b', 'c', 'd']

② extend() Add all elements of a list to another list.
ex >>> t1 = ['a', 'b', 'c'] >>> t2 = ['d', 'e']
>>> t1.extend(t2) >>> t1 ['a', 'b', 'c', 'd', 'e']
>>> t2 ['d', 'e']

③ sort() sort method arranges the elements of the list in ascending order.
ex >>> t = ['d', 'c', 'e', 'b', 'a'] >>> t.sort()
>>> t ['a', 'b', 'c', 'd', 'e']

④ Insert() Insert an item at the defined index inserts an element/object into a list at the index position. This method does not return any value but inserts the given element at the given index position.

Syntax

List.insert(index, obj)

index is the index where the object obj need to be inserted.

obj is the object to be inserted into the given list.

Ex >>> name = ['sudhan', 'Anto']
>>> name.insert(1, 'jovita')
print(name)
['sudhan', 'jovita', 'Anto']

⑤ count(x) Returns the count number of items passed as an argument

Ex >>> a = ['a', 'p', 'p', 'l', 'e'] >>> print(a.count('p'))2

⑥ Len() sutrun the number of elements in a list.

Ex >>> Num = [25, 50, 60, 70, 80] >>> print Len(Num)5

⑦ reverse():

reverse() method reverse the order of item in a list this method does not return any value but reverses the given object from the list.

Syntax List.reverse()

Ex >>> Name = ['vasana', 'jovita', 'sudhan'] >>>Name.reverse()
>>> print(Name) ['sudhan', 'jovita', 'vasana']

⑧ max : max() function returns the maximum value from a list.

syntax: max(list)

⑨ min : min(list) the min() function returns the minimum value from a list.

Ex >>> mark = [76, 80, 85, 92]
>>> print('minimum mark:', min(mark))
minimum mark: 76.

⑩ POP() Removes and returns an element at the given index.

⑪ copy() Returns a shallow copy of the list

4.5 List loop (Traversing A List)

① Traversing a list means process or go through each element of a list sequentially. when the list elements are processed within a loop. the loop variable or a separate counter is used as an index into the list which prints each counter positon elements till the end - 1. this pattern of computation is called a list traversal.

syntax: for <List_Item> in <List>:
    Statement to process <List_Item>

here. List_Item is individual element in the list.

example >>> for icecreams in icecreams.
    print (icecream)

output Vanilla
Strawberries
mango

② for loop works well to read the element of the list. but to write or update the elements.

Syntax. for <Index> in range (len (List)):
    Statement to process < List [Index]>

Index is the positional element of a list.

ex >>> icecreams = ['vanilla', 'strawberry', 'mango']
>>> for i in range (len (icecreams)): output
print (ice cream [i]) Vanilla
Strawberry
mango.

③. A for loop over an empty list never runs the body.

for x in []:
    print ('This never happens'.)

④ the nested list will be considered as a single element. the length of this list is four: >>> L = ['Dell', 2.0, [50, 100]]
>>> print (len (L))

⑤

4.6 MUTABILITY

The lists are mutable (changable). when the braket operator appears on the left side of an assignment.

eg
```
>>> numbers = [5, 10, 6]
>>> numbers[1] = 5
>>> numbers
[5 5 6]
```

4.7 LIST Membership

in and not in are the membership operators in Python. they are used to test whether a value or variable is found in a sequence or not.

4.7.1 in operator

The in Operator tests whether an element is a member of a list or not. If the element is a membe in the list. then it will produce True otherwise False

4.7.2 not in operator.

the not in operator evaluates to true if it does not find the elemen in the list and otherwise false.

eg
```
>>> icecreams = ['vanila', 'Strawbeers', 'mango']
>>> 'Strawberas' in icreams
True
>>> 'chocolate' not in icreams
True.
```

4.8 ALIASING

Aliasing is a circumstance where whe two or more variables refer to the same object. if 'a' refers to an object and assign 'b=a' then both variables refer to the same objec.

the association of a variable with an object is called refrence

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

→ If the aliased object is mutable, changes made with one alias affect the other.

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

a → [1,2,3] ← b

alias.py
```
list = ['A', 'B', 'C', 'D']
temp = list
print("list =", list)
print("Temp =", temp)
temp.append('E')
print("After Aliasing")
print("list =", list)
print("Temp =", temp)
```

output
```
list = ['A', 'B', 'C', 'D']
temp = ['A', 'B', 'C', 'D']
After Aliasing
list = ['A', 'B', 'C', 'D', 'E']
temp = ['A', 'B', 'C', 'D', 'E']
```

4.8.1 Aliasing with mutable objects.

It is safer to avoid aliasing when working with mutable objects. Although this behaviour can be useful, it is error-prone

ex
```
>>> t = [1,2,3]
>>> t X = t
>>> t[0,2,3]
>>> t [1,2,3]
>>> X[0] = 55
>>> X [55, 2, 3]
>>> t [55, 2, 3]
```

ex: immutable
```
a = 'engineering'
b = 'engineering'
```

It almost never makes a difference whether 'a' and 'b' refer to the same string or not.

4.8.2 Aliasing with immutable objects.

for immutable objects like strings, aliasing is not as much of a problem.

4.9 CLONING LISTS

→ Cloning a list is to make a copy of the list itself, not just the reference.

→ The easiest way to clone a list is to use the slice operator.

→ Taking any slice of a list creates a new list. In the case, the slice consists of the whole list.

→ The changes made in the cloned list will not be reflected back in the original list.

Illustration of cloning a list: (cloning.py)

```
list = ['A', 'B', 'C', 'D']
temp = list[:]
print("list =", list)
print("Temp =", temp)
print("changes in the cloned list will not affect orginal list")
temp.append('E')
print("list =", list)
print("Temp =", temp)
```

output

```
list = ['A', 'B', 'C', 'D']
temp = ['A', 'B', 'C', 'D']
changes in the cloned list will never affect orginal list.
list = ['A', 'B', 'C', 'D']
Temp = ['A', 'B', 'C', 'D', 'E']
```

4.10: LIST PARAMETERS

⇒ Passing a list as an argument actually passes a reference to the list. not a copy of the list. If the function modifies the list, the caller sees the change.

example

```
def delete_head(t):
    del t[0]
```

delete_head function removes the first element from a list

```
>>> letters = ['a', 'b', 'c']
>>> delete_head(letters)
>>> letters
['b', 'c']
```

Example of append:

```
>>> t1 = [1, 2]
>>> t2 = t1.append(3)
>>> t1
[1, 2, 3]
>>> t2
None
```

Example of + operator

```
>>> t3 = t1 + [4]
>>> t1
[1, 2, 3]
>>> t3
[1, 2, 3, 4]
```

* the tail function returns all but the first element of a list.

⇒

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

the function leaves the orginal list unmodified.

```
>>> letters = ['a', 'b', 'c']
>>> rest = tail(letters)
>>> rest
['b', 'c']
```

4.11 MAP, FILTER, AND REDUCE

4.11.1 Map

Map function applied onto each of the elements in a sequence and creates another sequence.

4.11.2 Filter

Filter operation is to select some of the elements from list & returns a sublist.

The only-upper function takes a list of strings and returns a list that contains only the uppercase strings.

```
def only_upper(t):
    res = []
    for s in t:
        if s.isupper():
            res.append(s)
    return res
```

```
def capitalize_all(t):
    res = []
    for s in t:
        res.append(s.capitalize())
    return res
```

'res' is initialized empty list.

4.11.3 Reduce

An operation that combines a sequence of elements into a single value is called reduce.

```
>>> t = [1,2,3]
>>> sum(t)
6
```

4.12 Deleting Elements

There are several ways.

1. pop: To know the index of the element deleted, the pop function is used.

2. del: If the index of element to be deleted is not provided, it deletes and returns the last element. If the removed value is not needed, then the del operator is used.

3. remove: To remove the element (but not the index).

4. del with slice: To remove more than one element the del with a slice index is used.

```
>>> t = ['a','b','c']
>>> x = t.pop(1)
>>> t
['a','c']
>>> x
'b'
```

```
>>> t = ['a','b','c']
>>> del t[1]
>>> t
['a','c']
```

```
>>> b = ['a','b','c']
>>> t.remove('b')
>>> t
['a','c']
```

```
>>> t = ['a','b','c','d','e','f']
>>> del t[1:5]
>>> t
['a','f']
```

4.13 MATRIX Using List

A list inside another list is called nested list or a matrix. In python a matrix is often represented with a list of lists (nested list)

The matrix could be created with

M = [[5, 4, 7], [11, 3, 6], [8, 17, 9]]

So M[0][0] is 5, M[0] M[1] is 4 and so on we can say M contains three rows as [5, 4, 7], [11, 3, 6] & [8, 17, 9]. each row contains three value each.

loop structure using for loop

```
M=[[5,4,7], [11,3,6], [8,17,9]]
for i in range(3):
    for j in range(3):
        print(M[i][j], "\t")
print("\n")
```

output

```
5   4   7
11  3   6
8   17  9
```

4.14 TUPLES

- A tuple is a sequence of values, which can be of any type and they are indexed by integers.
- Tuples are an immutable sequence of elements that is once a tuple is defined, it cannot be altered.
- Tuples & lists are essentially the same.
- A tuple is a comma-separated list of values

```
>>> t = 'a', 'b', 'c', 'd', 'e'
```

but it is not necessary to enclose tuples in parentheses.

```
>>> t = ('a', 'b', 'c', 'd', 'e')
```

4.14.1. Creation of tuples

- To create a tuple with a single element, include a final comma.

ex

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

```
>>> t2 = ('a')
>>> type(t2)
<class 'str'>
```

t2 is a string.

(2) A tuple can be created using the built-in function tuple. It can create a empty tuple with no argument. ex >>>t = tuple() >>> t ()

3. A tuple built-in function can be used to create a tuple with sequence of arguments.

ex >>> t = tuple ('computer')
>>> t
('c', 'o', 'm', 'p', 'u', 't', 'e', 'r')

4.4.2 Operators on tuple.

1. Bracket operator.
The bracket operator indexes an element.

>>> t = ('c', 'o', 'm', 'p', 'u', 't', 'e', 'r')
>>> t [0]
'c'

2. Slice operator.
The slice operator selects a range of elements.

>>> t[0:4]
('c', 'o', 'm', 'p')

tuples are immutable, so the elements in the tuples cannot be modified. >>> t[0] = 'C'
type error: Object doesn't support item assignment.

⇒ but tuples can be replaced with another tuple.

>>> t = ('C',) + t[0:]
>>> t
('C', 'c', 'o', 'm', 'p', 'u', 't', 'e', 'r')

3. Concatenation operator.
tuples can be concatenated or joined them using + or concatenation operator.

>>> T1 = ('pen', 'pencil')
>>> T2 = ('Eraser')
>>> T1 += T2
>>> print (T1)
>>> print (T2)

('pen', 'pencil', 'Eraser')
(Eraser.

Relational Operators.

It works with tuples and other sequences. Python starts by comparing the first element from each sequence. If they are equal. it goes on to the next elements, and so on.

Ex >>> (5,8,2) < (5,10,6) True

>>> (3,2,5000) < (3,8,5)

4.12.3 Looping through Tuple Elements.

Tuple elements can be traversed using control statements. The for loop is best to traverse tuple elements. The loop prints tuple elements either in separate line or same line

```
>>> weekday = ['Sunday', 'MONDAY', 'Tuesday', 'Wednesday',
'Thursday', 'Friday', 'Saturday']
>>> for wd in weekday:
    Print(wd)
Sunday Monday Tuesday Wednesday Thursday Friday Saturday
```

4.15 Tuple Assignment.

It is useful feature in python. It allows a tuple of variable on the left side of the assignment operator to be assigned respective value from a tuple on the right side.

```
>>>T1 = (10,20,30)
>>>T2 = (100,200,300,400)
>>>T1,T2 = T2,T1
>>>print T1    (100,200,300,400)
>>>print T2    (10,20,30)
```

Ex >>> a,b = min-max ('abcd')

```
>>> a
'a'
>>> b
'd'
```

```
>>>a,b = 1,2,3
Value Error:
```

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

```
>>>[x,y] = [3,4]
>>> x
3
>>> y
4
```

Ex-4 to split an email address into a user name and a domain.

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

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

the return values from split is a list with two elements. the 1st element is assigned to uname the 2nd to domain

4.16: Tuple As Return Values:

Function can always return only a single value but by making that value a tuple, a function can return more than one value.

```
>>> t = divmod(13,4)
>>> t
(3,1)
```

→ quotient
→ Remainder

```
>>> Q,R = divmod(13,4)
>>> Q
3
>>> R
```

A function can return a tuple, the built in function min_max() is used to find the largest & smallest element of a sequence and returns a tuple of two values

```
def min_max(t):
    return min(t), max(t)
```

ex

```
>>> min_max([6,3,7,12])
(3,12)
>>> min_max('abcd')
('a','d')
```

4.17 Tuple operations.

① Len() method returns the number of items in a tuple

② cmp() is used to check whether the given tuples are same or not. If both are same, it will return 'zero' otherwise return 1 or -1. If first tuple is big, then it will return 1, otherwise return -1

```
>>>T2 = (100, 200, 300, 400, 500)
>>> len(T2)
5
>>> max(T)
500
>>> min(T)
100
>>>T = (100, 200, 300, 100, 100)
>>> print(T.count(100)
3.
```

```
>>>T1 (10,20,30)
>>>T2 = (100,200,300)
>>>T3 = (10,20,30)
>>> cmp(T1,T2)
-1
>> cmp(T1,T3)
0
>>>> cmp(T2,T1)
1
```

(3) max: the max method returns its larger item in the tuple

(4) min: the min method returns its smaller item in tuple.

(5) count: It count how many times of an object occurs in a tuple.

4.18 Variable-Length Arguments tuples.

Function can take a variable number of arguments. A parameter name that begins with * gathers arguments into a tuple.

```
def printall(*args):
    print(args)
```

⇒ Printall function takes any number of arguments and prints them.

⇒ the gather parameter can have any name but args is conventional.

eg >>> printall(1, 2.0, '3') (1, 2.0, '3')

⇒ The complement of gather is scatter. It has sequence of value to a function multiple arguments the * operator is used.

eg. divmod takes exactly two arguments.

```
>>>t (7,3)
>>> divmod(t)
type error
```

```
>>>% divmod(*t)
(2, 1)
```

4.19 Lists and Tuples

→ Zip is a built in function that takes two or more sequences and returns a List of tuples where each tuple contains one element from each sequence.

→ the name of the function refers to a zipper, where Joins and interlaces two rows of teeth.

Ex
```
>>> s = 'abc'
>>> t = [0, 1, 2]
>>> zip(s, t)
<zip object at 0x7f7d0a9e7c8>
```

→ the result is a zip object that knows how to iterate through the pairs.

the most common use of zip is in a for loop.

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

A zip object is a kind of iterator
zip is used to make a list.

```
>>> list(zip(s, t))
[('a', 0), ('b', 1), ('c', 2)]
```

→ if the sequence are not the same Length, the result has the Length of the shorter one.

```
>>> list(zip('Anna', 'Ben'))
[('A', 'B'), ('n', 'e'), ('n', 'n')]
```

Difference between List and tuples.

→ Lists are enclosed in brackets ([]) and their elements and size can be changed, while tuples are enclosed in parenthesis (()) and cannot be updated.

→ Tuple can be thought of as read only List. Like list to initialize a tuple, one can enclose the values in parenthesis and separate them commas

4.20 Dictionaries and Tuples.

Dictionaries have a method called items that returns a sequence of tuples. where each tuple is a key value pair.

```
>>>d = {'a':0, 'b':1, 'c':2}
>>> t = d.items()
>>> t
dict_items([('c', 2), ('a', 0), ('b', 1)])
>>> for key, value in d.items():
...     print(key, value)
...
c 2
a 0
b 1
```

4.21. Dictionaries:

→ the data type dictionary fall under mapping. It is a mapping between a set of keys and a set of values. The key value pair is called an item.

→ A key is separated from its value by colon(:) and consecutive items separated by commas.

→ The entire items in a dictionary are enclosed in curly brackets ({}).

Syntax: `Dictionary_name = {key_1: value_1, key_2: value_2, key_3: value_3}`

→ If there are many key and values in dictionaries then write just one key-value pair on a line to make the code easy to read & understand.

→ Keys in the dictionary must be unique & immutable datatype. A Dictionary can be of any type.

→ Dictionaries are not sequences, rather they are mappings

→ The location that an element is stored in & retrieved from the dictionary depends solely on its key value.

→ The specific location that a value is stored is determined by a particular method of converting key values into index values called hashing

Ex:
```
>>> daily_temp = {'Sun': 68.8, 'mon': 70.2, 'tue': 67.2,
                  'wed': 71.8, 'thu': 73.2, 'fri': 75.6, 'sat': 74.3}
>>> print(daily_temp)
{'Sun': 68.8, 'mon': 70.2, 'tue': 67.2, 'wed': 71.8,
 'thu': 73.2, 'fri': 75.6, 'sat': 74.3}
```

4.21.1 Creating a dictionary

Method 1: Dictionary using dict() function

the dict() function is used to create a new dictionary with no items. Here 'dict' is the name of a built-in function.

eg >>> mach = dict() >>> mach {}

Method 2: Creating empty dictionary using {}

Dictionaries can be created using an empty string. to add an item to the dictionary use square brackets ([]) with keys for accessing & initializing dictionary values.

Method 3: Dictionary using literal notation

It uses literal notation with key-value pairs

Syntax

directory_name = {Key: value, Key, Value ... - Key N: value N}

⇒ the keys of the dictionary may be either integers, or one character strings the corresponding value may be integer code points. one character string or the special value None.

eg >>> weekdays = {"Sunday": 1, "Monday": 2, "T": 3, "W": 4, "T": 5, "F": 6, "S": 7}

print (weekdays)

{'Sunday': 1, 'M': 2, 'T': 3, 'W': 4, 'T': 5, 'F': 6, 'S': 7}

or empty {}

>>> weekdays = {}
>>> weekdays[0] = 'Sunday'
>>> weekdays[1] = 'Mon'
>>> weekdays[2] = 'Tuesday'
>>> print weekdays
{0: 'Sun', 1: 'Mon', 2: 'Tuesday'}

4.21.2 Inserting - Adding items into dictionary

to add items to the dictionary, the square brackets are used.

4.21.3: Accessing Dictionary.

Dictionaries use keys instead of index to access values. key can be used either inside square brackets or with the get() method.

>>> cse = {'name': 'Jason', 'age': 19}
>>> print (cse['name'])
Jason
>>> print (cse.get('age'))
19

Add items

>>> cse = {}
>>> print (cse)
{}
>>> cse['name'] = 'Anto'
>>> cse['age'] = 18
>>> print (cse)
{'name': 'Anto', 'age': 18}

Acm dictionary

4.21 Printing Dictionary

A key value pair with a colon between the key and value can be printed using 'print' command.

```
>>> print(cse)
{'name': 'Anto', 'age': 18}
>>> for x in cse:
    print(x, ":", cse[x])
```

name: Anto
Age: 18

4.22 operations and methods in Dictionary

Operation	Results:
dict()	creates a new, empty dictionary
dict(s)	create a new dictionary with key value pairs associated values from sequence. ex: fruit_price = dict(fruits_data) (pass tuple/list) [['apples": 6], ['banana': 12]]
len(d)	Length (num of key/value pairs) of dictionary d.
d[key] = value	adds the associated value for key to value, then to either add a new key/value pair, or replace the value of an existing key/value pair.
del d[key]	Remove key and associated value from dictionary d.
key in d	True if key exists in dictionary d, or returns false.

① Len function
this function returns the total number of key value pair in the dictionary. ex >>> len(cse) 3

② in operator
the in operator tells whether a key is present in the dictionary ex >>> 'name' in cse True
>>> 'Antony' in cse False.

③ values function
It tells whether a value is present in the dictionary
ex
>>> val = cse.values() >>> 'Antony' in val True

④ sorted function
It is used to sort the keys of a dictionary
ex >>> Students = {'Hasna': 'B+', 'Jovita': 'O+', 'Anto': 'B-', 'Ben': 'B+'}
>>> for key in sorted(Students):
print("{} : {}".format(Students[key], key))
B- Anto B+ Ben B+ Hasna O+ Jovita.

⑤ keys() function
this function returns the total number of key values pairs in the dictionary ex >>> print(cse.keys())
dict_keys(['name', 'age', 'dept'])

23 Advanced List processing : List comprehensions.

→ List comprehension is an elegant & concise way to create new list from an existing list in Python.
→ It can be characterized by a process the process is described by a series of keywords

[expr for val in list if expr]

→ val is a variable name. List is an existing sequence.
→ the optional if part requires an expression that evaluates true or false. only those elements that evaluate to true are examined.
→ list comprehension combines the aspects of both a filter and a map. It uses the body of function.
→ the list comprehension is an easy to understand and easy to debug way to write the body of the function.

Ex: >>> def list_of_squares(a):
    return [x*x for x in a]
>>> list_of_squares([6,4,9]) [36, 16, 81]

ex: range function allows the generation of sequence of integers in fixed increments

>>> L = [x for x in range(1,9)]
>>> print(L) [1, 4, 9, 16, 25, 36, 49, 64]

ex: square the integers in y and print the square

>>> y = [1, 2, 'a', 3, 4.0]
>>> print([x*x for x in y if type(x) == int]) [1, 4, 9]

ex: selects the positive elements from the list, 'numbers'

>>> numbers = [-3, -2, -1, 0, 1, 2, 3]
>>> [x for x in numbers if x >= 0] [0, 1, 2, 3]

Ex: the tuple vowels is used for generating a list containing only the vowels in string E.

>>> vowels = ('a', 'e', 'i', 'o', 'u') >>> t = 'hello'
>>> [ch for ch in t if ch in vowels] ['e', 'o']

ex:
>>> a = [1, 2, 3, 4, 5]
>>> [a[i] for i in range(0, len(a)) if i % 2 == 0]
[1, 3, 5]

Python 3 -m pip install numpy

24 Dictionary as a collection of counters

There are several ways to count how many times each letter appears in the string. some of them are

① Create 26 variables one for each letter of the alphabet then traverse the string and for each character, increment the corresponding counter, probably using a chained conditional.

② Create a list with 26 elements. Then convert each character to a number as an index into the list, and increment the appropriate counter

③ Create a dictionary with characters as keys and counters as the corresponding values, when the character is seen at first time, add an item to the dictionary else increment the value of an existing item

Example

```
>>> def histogram(s):
    d = dict()
    for c in s:
        if c not in d:
            d[c]=1
        else:
            d[c]+=1
    return d
```

the dictionary implementation uses its function histogram, which is a statistical term for a collection of counters (or frequencies)

eg
```
>>> h = histogram('parrot')
>>> h
{'p':1, 'a':1, 'r':2, 'o':1, 't':1}
```

25 Looping and Dictionaries

If the 'for' statement is used in a dictionary, it traverses the keys of the dictionary.

eg print_hist prints each Key and the corresponding values

```
def print_hist(h):
    for c in h:
        print(h[c])
```

```
>>> h = histogram('parrot')
>>> print_hist(h)
a 1
p 1
r 2
t 1
o 1
```

here, its keys are not in particular order.

to traverse the keys in sorted order, use its built in function 'sorted'

eg
```
>>> for key in sorted(h):
...     print(key, h[key])
```

```
a 1
o 1
p 1
r 2
t 1
```

illustration of histogram.

```
def histogram(items):
    for n in items:
        output = ''
        times = n
        while (times > 0):
            output += '*'
            times = times - 1
        print(output)
histogram([6,13,6,25,1,8])
```

output

******
****
******
****
****

Reverse lookup

⇒ Given a dictionary 'd' and a key 'k', it is easy to find the corresponding value v = d[k]. This operation is called a lookup.

⇒ To find value 'v' in dictionary 'd' there are two possibilities There might be more than one key that maps to the value 'v'. Depending on the application, we can select one or a list that contains all of them.

⇒ There is no matching between 'v' and 'k' exists in dictionary 'd'. This is called reverse lookup.

⇒ A function that takes a value, returns the first key that maps to that value

```
def reverse_lookup(d, v):
    for k in d:
        if d[k] == v:
            return k
    raise LookupError()
```

```
>>> h = histogram('parrot')
>>> key = reverse_lookup(h, 2)
>>> key
'r'
```

ILLUSTRATIVE PROGRAMS

1. **Selection sort**: select the smallest element in the list. When the element is found it is swapped with the first element in the list.

e.g. 56, 91, 35, 72, 48, 68

unorder list	56	91	35	72	48	68
After pass 1	56	91	35	72	48	68
After pass 2	35	91	56	72	48	68
After pass 3	35	48	56	72	91	68
After pass 4	35	48	56	72	91	68
After pass 5	35	48	56	68	91	72
Sorted list	35	48	56	68	72	91

program:

```
def selectionSort(A):
    for i in range(len(A)-1, 0-1):
        max=0
```

```
for j in range (1, i+1):
    if A[j] > A[max]:
        max = j
temp = A[i]
A[i] = A[max]
A[max] = temp
X = [0,0,0,0,0,0,0,0,0,0]
n = int (input ("Enter list size:"))
print ("Enter numbers:")
for i in range (0,n):
    X[i] = int (input())
selection sort (X)  function call
print (X)
```

output

```
Enter list size: 6
Enter numbers
4
8
2
9
5
7
[0,0,0,0,2,4,5,7,8,9]
```

2. Insertion sort

⇒ insertion sort technique is the simplest sorting technique. Each successive element in the list to be sorted and inserted into its proper place with respect to the other already sorted elements.

⇒ It start with the second element and put it's correct place so that the first second elements of the list are in order. this process continue until all the elements have been sorted.

⇒ Insertion sort performs n-1 passes.

for pass P=1 through n-1 insertion sort ensures that the elements in position 0 through pass sorted order.

⇒ In pass P we move the element in position P to the left until its correct place is found.

example Sort the numbers 34, 8, 64, 51, 32, 21.

	[0]	[1]	[2]	[3]	[4]	[5]	Position moved
Original	34	8	64	51	32	21	
After P=1	8	34	64	51	32	21	1
After P=2	8	34	64	51	32	21	0
After P=3	8	34	51	64	32	21	1
After P=4	8	34	34	51	64	21	3
After P=5	8	34	32	34	51	64	4

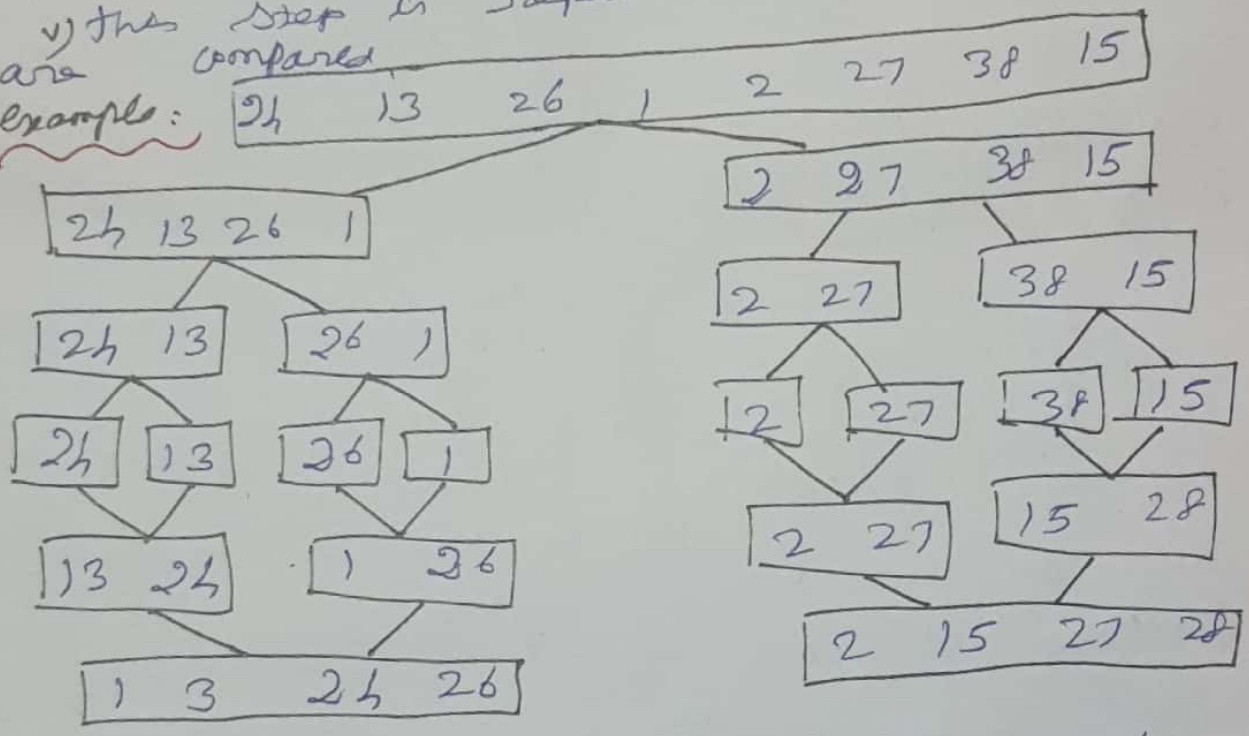
3. merge sort

merge Sort uses divide and conquer method. the problem is divided into smaller problem and solved recursively. Finally it merges two sorted list.

Basic Algorithm:

i) Divide the given array of elements into two half
ii) Recursively sort the first half & second half of array elements.
iii) Take two split arrays A and B. an output array C
iv) The first element of A array B array are compared. then the smaller element is stored in the output array C, the corresponding pointer is incremented.
v) this step is repeated until all the elements are compared.

example:

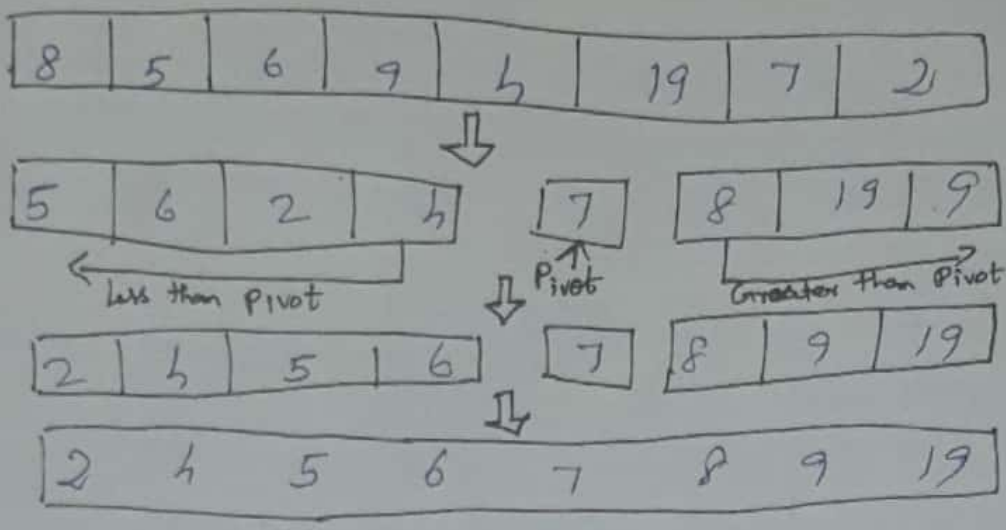


Quick sort

Quick sort uses divide and conquer method.

basic Algorithm.

① If the number of elements in S is 0 or 1, then return
② Pick any element v in S. this is called pivot
③ Partion S - {v} (the remaining element in S) into two disjoint groups: Return {quick sort (S1) followed by v followed by quicksort (S2)}



Retail Bill Preparation

```
tax = 0.18
Rate = { "pencil": 10, "pen": 2.0, "scale": 7, "A4 paper": 16.5,
"writingPad": 15 }
Print ("Retail Bill calculator \n")
Print ("Enter the quality of ordered item : \n")
Pencil = int (input ["pencil:"])
pen = int (input ("pen:"))
scale = int (input ("scale:"))
A4 paper = int (input ("A4 paper :"))
writingpad = int (input ("writingpad :"))
Cost = Pencil * Rate ["pencil"] + Pen * Rate ["pen"] + Scale * Rate
   ["scale"] + A4 paper * Rate ["A4 Paper"] + writingpad *
                                Rate ["writingPad"]
Bill = cost + cost * tax
Print ("Please pay Rs. %6" % Bill)
```

Outputs

```
Retail Bill Calculator
Enter the quality of ordered item
Pencil: 5
pen : 7
scale : 4
A4 paper : 3
Writing pad: 2
Please pay RS: 876.760000
```

thank you.

UNIT V FILES, MODULES, PACKAGES

Files and Exceptions: text files, reading and writing files formal operation, command line arguments errors and exceptions, handling exceptions, modules, packages. Illustrative programs: word count, copy file, Voter's age validation, marks range validation (0-100).

5:1 Files:

A File is a collection of data stored on a secondary storage device like hard disk. they can be easily retrieved when required. Python supports two types of files. they are Text file & binary files

5.1.1 Text Files:

→ A text file is a stream of characters that can be sequentially processed by a computer in forward direction.

→ The text files can process characters: they can read or write data only one character at a time.

→ In Python, a text stream is treated as a special kind of file.

→ Each line of data ends with a newline character. Each file ends with a special character called the End-of-File (EOF) marker.

5.1.2 Binary Files:

→ A binary file is a file which may contain any type of data, encoded in binary form for computer storage and processing purpose.

→ binary file does not require any special processing of the data and each byte of data is transferred to or from the disk unprocessed.

→ the binary files can be processed sequentially or randomly depending on the needs of the application.

5.2 Opening a file

⇒ The contents of a file can be read by opening the file in read mode.

→ Python has a built-in function open() to open a file. The function returns a file object, also called a handle, as it is used to read or modify the file accordingly.

⇒ two arguments that are mainly needed by the open() function are: file name or file path and mode in which the file is opened.

Syntax: `file_object = open(file name [, access_mode])`

file_name : File name containing the name of the file which we want to access

access_mode : The value of access_mode specifies the mode in which we want to open the file. i.e.: read, write, append, etc.

Mode	Description
'r'	open a file for reading (default)
'w'	Open a file for writing. It creates a new file if it does not exist or truncates the file if it exists
'x'	Open a file for exclusive creation if the file already exists or the operation fails.
'a'	Open for appending file at the end of the file without truncating it creates a new if it does not exist.
't'	Open in text mode (default)
'b'	Open in binary mode
'+'	Open file for updating (reading and writing)

Python File modes.

⇒ the binary mode returns bytes and this is the mode to be used when dealing with non-text files like image or exe files.

```
>>> f = open("test.txt")  # equivalent to 'r' or 'rt'
>>> f = open("test.txt", 'w')  # write in text mode
>>> f = open("img.bmp", 'r+b')  # read and write in binary mode.
```

program: illustration of opening a file

```
f1 = open("sample.txt", "wb")
print("Name of the file:", f1.name)
print("opening mode:" f1.mode)
f2 = open("Sample1.txt", "r")
line = f2.readline()
print("The contents of sample1 textare:", line)
f2.close()
```

output: Name of the file: Sample.txt
opening mode: wb

5.3 Closing A File

⇒ Closing a file will free up the resources that were tied with the file and is done using the close() method.

⇒ Python has a garbage collector to clean up unreferenced objects. but we must not rely on it to close the file.

Syntax: fileObject.close()

example:

```
>>> f = open("test.txt", encoding = 'utf-8')
>>> f.close()
```

5.4 Writing to a file

⇒ The write() method is used to write a string to an already opened file.

⇒ to write into a file, it is needed to open a file in write 'w', append 'a' or exclusive creation 'x' mode.

⇒ the 'w' mode will overwrite into the file if it already exists. so all previous data are erased.

⇒ Writing a string or sequence of bytes (for binary files) is done using write() method. this method returns the number of characters written to the file

Syntax: fileObject.write(string)

Program illustration of writing and reading a file.

```
f1 = open("sample2.txt", "w")
str = "Problem solving and python programming"
f1.write(str)
f1.close()
f2 = open("sample2.txt", "r")
line = f2.read()
print("the contents of sample2 text are:", line)
f2.close()
```

output: the contents of sample2 text are: Problem solving and Python programming

5.4.1 writelines() method

is used to write a list of strings.

ex:
```
file = open("file1.txt", "w")
lines = ["Hello students", "welcome", "Enjoy learning Python"]
file.writelines(lines)
file.close()
print("writing to file...")
```

output: writing to file...

5.4.2 Append() Method

Is used to append file. In order to append a new line to a existing file. open the file in append mode. by either 'a' or 'a+' as the acces mode. the definition of the acces modes are.

Append only ('a'): Open the file for writing. the file is created if it does not exist. the handle is positioned at the end of the file. the data being written will be inserted at the end after the existing data.

Append and Read ('a+'): open the file for reading and writing. the file is created if it does not exist. the handle is positioned at the end of the file. the data being written will be inserted at the end. after the existing data.

file object.read ([size])

5.5 Reading From a File

5.5.1. Reading a file using read(size) method.

To read the content of a file. its must be opened in read mode. It used to read size number of data. If size parameter is not specified. its reads and returns up to the end of the file.

Syntax fileobject.read([size])

It starts reading from the begning of the file until the size given. If no size is passed. it ends up reading until the end of the file.

ex

```
>>> f = open ("test.txt", "r", encoding = 'utf-8')
>>> f.read(4)        #read first 4 data
'A 10'
>>> f.read(4)        #read next 4 data
'une'
>>> f.read()         #read the file till end of file.
'y of \n a thousand miles begins \n\n with a single step\n'
>>>f.read()          further reading returns empty string
```

5.5.2 Reading a file using for loop

A file can be read line by line using a 'for loop'. this is both efficient and fast. the line in file itself has a newLine character '\n'.

```
>>> f = open ("test.txt", 'rn', encoding = 'utf-8')
>>> for line in f:
        print (line, end = '')
```

A journey of a thousand mile begins with a single step

5.5.3 Reading a file using readline() method.

is used to read individual lines of a file. this method reads a file till the newline, including the newline character.

5.6 Renaming and Deleting Files:

Python os module has various methods that can be used to perform file-processing operations. such as renaming and Deleting files.

5.6.1 the rename() method.

it takes two arguments, the current filename and the new filename.

Syntax os.rename (current_file_name, new_file_name)

Ex To rename an existing file test1.txt to test2.txt.

```
import os
os.rename ("test1.txt", "test2.txt")
```

5.6.2 the remove() Method.

is used to delete files the method take a filename as an argument and deletes that file.

Syntax: os.remove (file_name)

Ex import os
os.remove ("test2.txt").

5.7 Format Operator.

the argument of write should to be a string. To put other values in a file, they have to be converted to strings.

Method : 1:

The str() method is used to convert other type of data to string type.

String → "%.s"
Integer → "%.d"
float = "%.f"

Ex: >>> with open ("strexample.txt", 'w', encoding = 'utf-8') as f:
x=52
f.write (str(x))
2

Method 2: Format operator is an operator (%) that takes a format string and a tuple as input and generates a string that includes the elements of the tuple. formatted as specified by formatting string.

Ex >>> Passmarks = 15
>>> '%d' % passmarks
'15'
Here 15 is the string

>>> name = "Python"
>>> Print ("welcome %.s", %name)
welcome Python

Syntax [key][flags][width][.precision][length type] conversion type

Key mapping key, consisting of parenthesized sequence of characters

Flags: conversion flags, which affect the result of some conversion type.

width: minimum field width. If specified as '*' (asterisk) the actual width is read from the next element of the tuple in values and the object to convert comes after the minimum field width optional precision.

precision. given as a '.' (dot) followed by precision. If specified as '*' (an asterisk), the actual width is read from the element of the tuple in values, and the value to convert comes after the precision

Length type: Length modifier

Conversion type. conversion type.

Ex Print ("The number is %d" %123) the number is :123
Print ("the float number is %f" %123-568789) the float number is:
Print ("%.05d" %12) 00012
Print ("%.8f" %-12.236) -12.236

5.7.1 Formatting Using . format function.

→ Python supports format function which provides better performance than the format operator %

→ All the formatting done by % operator is also incorporated by the . format function. It is very similar to the operator. However, the conversion types are specified in curly braces.

Syntax: <format expression>. format (v1, v2, v3 ... vn) (or)

<format expression>. format values.

where v1 to vn are variables that are formatted using the expression

Values is a tuple with exactly the number of items specified by the format string

Formatting with alignment:

The operators <, ^, > and = are used for alignment when aligning a certain width to the numbers.

type	Meaning.
<	Left align to the remaining space
^	center align to the remaining space
>	Right align to the remaining space
=	Force the sign (+) (-) to the leftmost position.

Program illustration of number formatting using . format function

```
Print ("the number is : {d}", format (123))        the number is 123
Print ("the float number is : {:f}", format (123.456789))   the float number is 123.456789
print ("bin : {0:b}, oct : {0:o}, hex : {0:x}", format (12))   bin 1100, oct 14, hex c
Print ("{:5d}". format (12))                        12
print ("{:^10.3f}". format (12.2356))              12.235
print ("{:<05d}". format (12))                     12000
print ("{:=8.3f}". format (-12.2356))              -12.235
```

5.8 Command Line Arguments

→ Python provides a getopt module that helps to Parse command-line options and arguments.

→ $ Python test.py arg1 arg2 arg3

the Python sys module provides access to any command line arguments via the sys.argv. this serves two purpose

→ sys.argv is list of command line arguments

→ Len(sys.argv) is the number of command line arguments.

Ex Script test.py

```
#!/usr/bin/python
import sys
print 'Number of arguments:' Len(sys.argv), 'arguments'.
print 'Argument List:', str(sys.argv)
```

To run $python test.py arg1 arg2 arg3

output number of arguments : 4 arguments

Argument List: ['test.py', 'arg1', 'arg2', 'arg3']

5.9 Errors and Exceptions:

Python raises exceptions when it encounters errors ex. divided by zero

5.9.1 Syntax error:

error caused by not following the proper structure (Syntax) of the language is called Syntax error or parsing error. Ex >>> if a < 3 SyntaxError: invalid Syntax

5.9.2 Logic error

→ Error caused due to wrong algorithm or logic to solve a particular program.

→ Logic error depends all type of errors in which the program executes but gives incorrect results

→ Logic error leads to a run time error that causes the program to terminate abruptly. this type of run time error is known as exceptions

ex. Divide by zero

5.9.3 Exceptions.

Even if a Statement or expression is syntactically correct it may cause an error when an attempt is made to execute. Errors detected during execution are called Exceptions.

eg FileNotFound Error, ZeroDivisionError

5.9.4 Python built_in Exceptions.

Illegal operations can raise exceptions. There are lots of built-in exceptions in python that are raised when corresponding errors occurs.

eg syntax error, SystemError, ValueError, typeerror

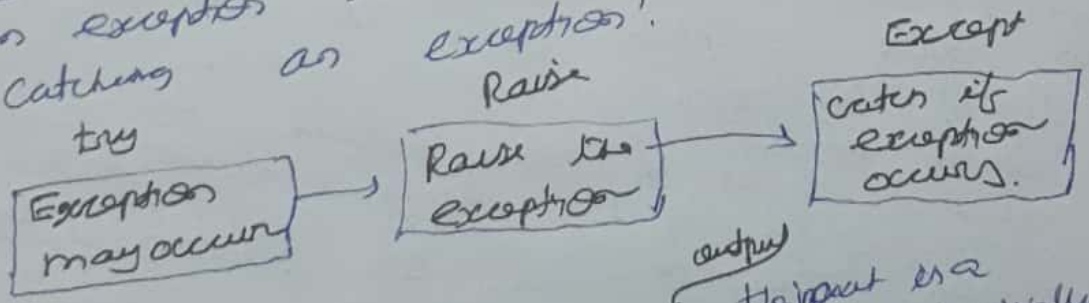
5:10 Handling Exceptions.

When exceptions occurs, it makes the current process to stop and passes it to the calling process until it is handled. If not handled, the program will crash.

If never handled, an error message is split out & the program comes to a sudden, unexpected halt.

5:10.1 Catching Exceptions in python

In python exceptions can be handled using 'try' statement. It starts by executing the try clause If an exception occurs, it jumps out of the try clause & runs the except clause. handling an exception with a try statement is called 'catching an exception'.



Syntax

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

```
import sys
N = ['a', 0, 2]
for x in N:
    try:
        print("The input is", x)
        r = 1/int(x)
        break
    except:
        print("oops!", sys.exc_info()[0], "occured.")
        print("next input.")
        print()
print("The reciprocal of", x, "is", r)
```

output

```
The input is a
oops! <class 'ValueError'> occured.
next input.
The input is 0
oops! <class 'ZeroDivisionError'> occured.
Next input
The input is 2
The reciprocal of 2 is 0.5
```

5.10.2 Raising Exception

the exception can be deliberately raised using the raise keyword.

syntax raise [Exception [, args [, traceback]]]

```
try:
    num = 100
    print(num)
    raise ValueError
except:
    print("Exception occured.
          Program Terminating.")
```

output
100
Exception occured. program terminating

```
try:
    k = 5//0  # raises divide by zero exc
    print(k)
except ZeroDivisionError:
    print("can't divide by zero")
finally:
    print("This always executed")
```

output
can't divide by zero
this is always executed.

10: 5.3 The Finally Block

The try block has an optional block called finally. It is also a part of exception handling. It always executed before leaving the try block.

5:11 Modules.

→ modules are pre-written pieces of code that are used to perform common tasks like generating random number, performing mathematical operations etc.

→ A module is a file with .py extension that has definitions of all functions and variables that we use in other programs.

→ It used to break down large programs into small manageable & organized files. It promote reusability code

→ The user can define their own most used function in a module and import it, instead of copying their definition to different programs.

5.11.1 Creating a module.

The modules can be created as we want. Every Python program is a module that is every file saved as .py extension is a module

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

The module should be placed in same directory as that of the program in which it is imported.

```
>>> import example
>>> example.add(8,9.5)
17.5
```

output

```
from math import pi
print("the value of pi is", pi)
```

output
The value of pi is 3.141592...

5.11.2 the from... import modules

A module may contain definitions for many variables and functions. when importing a module, we can use any variable or functions defined in that module. to use only selected variables or functions that we can use the from... import statement.

Eg: from math import pi, sqrt

from dyp import * import all the identifiers defined in the dyp module. use the from dyp import * statement

5.12 Datetime Module

Python display the date in yyyy-mm-dd format. It handle the extraction & formatting of date and time variables. objects of date time type represent a date & time in some time zone.

Calender date value are represented with the date class. All the attributes are accessed through the dot(.) operator with the date object.

Syntax: import datetime date = datetime.date.today()

Eg: import datetime
idate = datetime.date.today()
print 'Today is':, tdate

5.13 MATH Module

Python provides many useful mathematical functions in a special math library. Python has a math module that provides most of familiar mathematical functions. to access one of the functions we have to specify the name of the module and the name of the function separated by a dot(.) operator. This format is called dot(.) notation

method 1: import math

The statement imports the entire math module into the current applications.

>>> import math >>> print math.sqrt(100) 10.0

Method 2: from math import sqrt

The statement imports only one method called sqrt() into the current application. To use the sqrt() method we can avoid dot(.)

>>> from math import sqrt
>>> print sqrt(100). 10.0

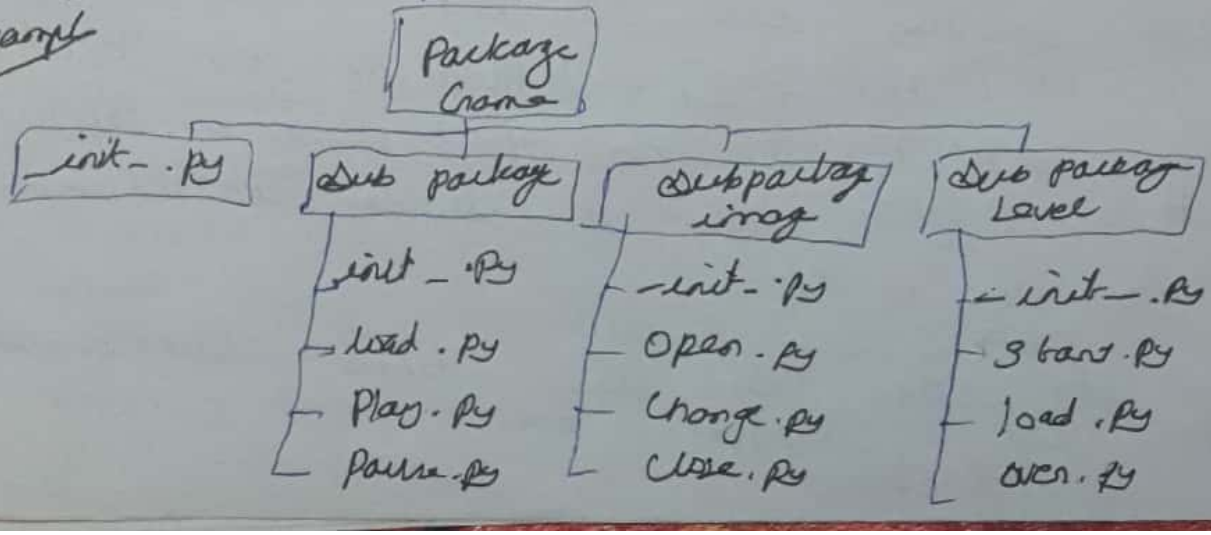
Method 3: from math import sqrt, pi
The statement imports two methods sqrt and the value of constant pi into the current application.

Method 4: from math import *
imports all the methods from math module into the current application.

5.14 Packages:
A Package is a way of collecting related modules together within a single tree like hierarchy.
It has user-organized hierarchy of directories for easier access.
A directory can contain sub-directories & files.
Where a python package can have sub packages & modules.
A directory must contain a file named __init__.py in order for python to consider it as a package.
the file can be left empty by generally, the initialization code is placed for first package in the file.

5.14.1 Creating a Package.
Create a directory & name it with package name.
Keep sub directories (sub packages) & modules in it
Create __init__.py file in the directory.
The __init__.py file can be left empty but generally place the initialization code with import statements to import resources from a newly created package

Example



5.14.2 importing module from a package.

⇒ The modules are imported from packages using the dot(.) operator. To import the Start module in

[import Game.Level.start]

⇒ If the module contains a function named difficulty(), one has to full name to reference it.

[Game.Level.start - difficulty(2)]

⇒ If this construct seems lengthy, import the module without the package prefix.

[From Game.Level import start]

using the full namespace avoids confusion & prevents two same identifier names from colliding.

5.15 DATABASES

★ A database is a file that is organized for storing data. Many databases should be created,

★ Many databases are organized like a dictionary in the sense that they map from key to value

★ The biggest difference between a database & Dictionary is that the database is on disk (or other permanent storage). So it persists after program ends

★ The module dbm provide an interface for creating & updating database files. Opening a database is similar to opening other files.

Example >>> import dbm
>>> db = dbm.open('captions', 'c')

>>> db['Jack.Png'] = 'Photo of John Jack'

when accessing one of the items, dbm reads file

>>> db['Jack.Png'] b'Photo of John Jack'

the result is a bytes object which is why begins with b. A bytes object is similar to a string in many ways.

for key in db:
    print(key, db[key])

As with other files, the database should be closed when its operations are done.

>>> db.close()

5.16 Picking

★ The write methods of the file object in Python accepts only string datatype. any other datatype has to be converted to string before writing in the file

★ Picking converts any kind of complex data to OS and IS (byte streams). this process can be refered to as picking. Serialization, flattening or marshalling.

★ Pickling refers to the process of converting an object in memory to a byte stream that can be stored on disk.

★ The resulting byte stream can also be converted back into python objects by a process known as unpickling

★ When dealing with binary, the write operation is known as dumping. and read operation is known as loading.

Pickling can be done with following datatypes.

i) Booleans ii) Integers iii) Floats iv) Complex numbers v) (normal & unicode) strings vi) Tuples vii) Lists viii) Sets ix) Dictionaries.

Dump and load:

Pickling and unpickling involves file IO. It uses the file writing / reading routines.

the pickle.dump() is method for saving the data out to the designated pickle file.

similarly pickle.load() method is used to read pickled file

Syntax To write: pickle.dump(<datatype variables>, <filename>)

To read: pickle.load(<filename>)

To unpickle this dictionary

```
import pickle
Pickle_off = open("Emp.pickle", "rb")
emp = pickle.load(Pickle_off)
print(emp)
```

Output: {1:'A', 2:'B', 3:'C', 4:'D', 5:'E'}

Illustrative problems.

1. Counting number of words in a string.

```
string = input ("Enter any string:")
word_length = len (string.split())
print("number of words =", word_length, "\n")
```

output

Enter any string: problem solving & programming with python.
Number of words = 6

2. Copying file.

```
from shutil import copyfile
source file = input ("Enter source file name:")
destination file = input ("Enter destination filename:")
copyfile (sourcefile, destination file)
print ("File copied successfully!")
c = open (destinationfile, "r")
print (c.read())
c.close()
print()
print()
```

output

Enter source file name: file1.txt
Enter destination file name: file2.txt
file copied successfully!
Sunflower
Jasmine
Roses.

3. Voters Age validation:

```
import datetime
year_of_birth = int (input ("In which year you took birth:"))
current_year = datetime.datetime.now().year
current_age = current_year - year_of_birth
print ("your current age is:", current_age)
if (current_age >= 18):
    print ("you are eligible to vote")
else
    print ("you are not eligible to vote")
```

output

In which year you took birth: 2015
your current age is 6
you are not eligible to vote.

4. Marks range validation (0-100)

```
Mark = int (input ("Enter the Mark:"))
if Mark < 0 or Mark > 100:
    print ("The value is out of range, try again")
else:
    print ("The mark is in the range")
```

output

Enter the Mark: 98
The mark is in the range
Enter the Mark: 160
The value is out of range, try again thankyou.

(1) Greatest among three numbers:

Algorithm:

Step1 Start

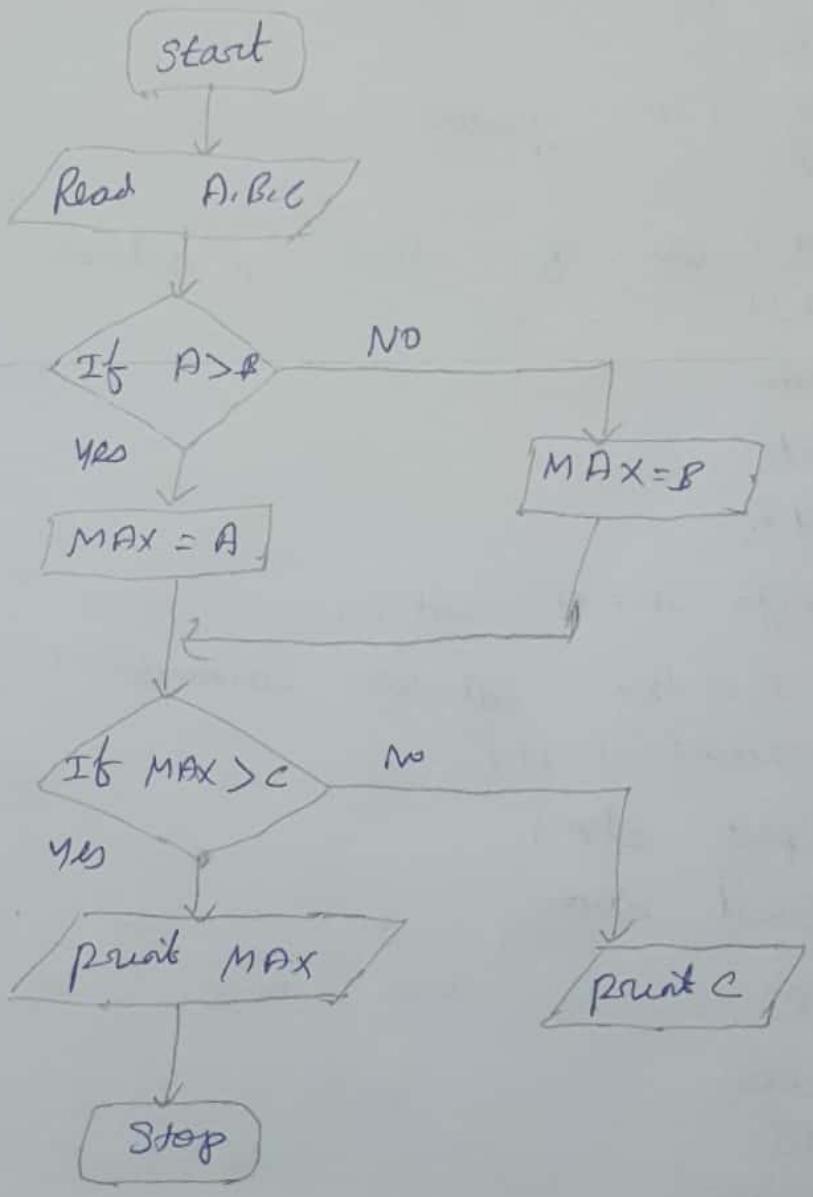
Step2: Read the three numbers A, B, C

Step3: Compare A and B. If A is the greatest, store A in MAX, else store B in MAX

Step4: Compare MAX and C. If MAX is the greatest, output "MAX is the greatest" else output "C is the greatest".

Step 5: Stop

Flow chart:



Pseudo code:

START
READ the Values of A, B, C
If A is greater than B THEN
ASSIGN A to MAX
ELSE
ASSIGN B to MAX
if MAX is greater than C THEN
PRINT MAX is greatest
ELSE
PRINT C is greatest
STOP

(2) Finding sum of first N natural numbers (1+2+3+ ... .. N)

Algorithm:

1. Start
2. Read N
3. Assign Sum=0 and i=1.
4. If i<N then Calculate Sum=Sum+i.
5. Increment i=i+1.
6. Repeat Step 4.
7. Print Sum.
8. Stop.

Pseudo code.

START
READN
ASSIGN Sum=0 and i=1

DO WHILE i<N
CALCULATE SUM = SUM + i
INCREMENT i by 1
END DO
PRINT SUM
STOP

(3) Checking Leap year or not

Algorithm:

1. Start
2. Read the year
3. If (year % 4) = 0 then
   print "year Leap year"
4. Else print "year is not Leap year".
5. Stop

Pseudo code:

START
READ year
If (year % 4) = 0 THEN
PRINT "year is Leap year"
ELSE
PRINT "year is not leap year"
STOP.

(4) write an algorithm to accept two numbers. compute the sum and print the result.

1. Start
2. Read two numbers a and b
3. Calculate sum = a + b
4. Print sum
5. Stop

(5) Checking odd or even numbers.

Algorithm:

1. Start
2. Read n
3. Calculate remainder = n % 2.
4. If remainder is 0 then print even number otherwise print odd number
5. Stop.

Pseudo code:

```
START
READ n
CALCULATE r=n%2
If r=0 THEN
    PRINT even number
ELSE
    PRINT odd number
STOP.
```

Flow chart