GE3151 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:

A computer is an electronic device that can be programmed to accept data (input), process it and generate results (output).

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

Generations of a Computer:

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

Each 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 computers 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 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 uses 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 operat

Machines were prone to frequent malfunctioning

Example:

UNIVAC - Universal Automatic Computer

ENIAC - Electronic Numerical Integrator & Calculator

EDVAC - Electronic Discrete 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.

Input was through punch cards and output was through printouts.

Software Technology:

→ Assembly language

This language uses ADD for Addition and SUB for subtraction for coding of the instructions.

→ The Computation time was in microseconds

→ Transistors are smaller in size compared to vaccum tubes, thus the size of the computer was also reduced.

Disadvantages:

Generated lot of heat but less than 1st generation computers.

Less maintanance 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 computer.

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 extensively for programming, instead of machine language and Assembly language.

→ The Computation time was in Nanoseconds.

→ the size of those computers were small Compared to 2nd generation Computers

Advantages:

This generation computers used less power and generated less heat.

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 memory, resulting in fast random access to memory.

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

The linking of computers is another key development and computers were linked to form networks that lead to emergence of Internet.

Software Technology:

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

High level programming languages are used for writing of programs.

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

Computing characteristics:

The computation time is in picoseconds.

Physical Appearance:

Smaller than computers of previous generation.

Advantage:

In this computers are portable & more reliable

They generate much losses heat and require less maintanence.

Fifth Generation (Present and Next):

Artificial Intelligence:

Fifth generation computers use super large scale Integrated (SCSI) chips that are able to store millions of components on a single chip

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 Com software. Hardware is the physical component of a computer like motherboard, memory devices, monitor, keyboard etc.,

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

The Computer system hardware comprises of three main components:

Input / Output Unit

Central Processing Unit (CPU)

Memory Unit.

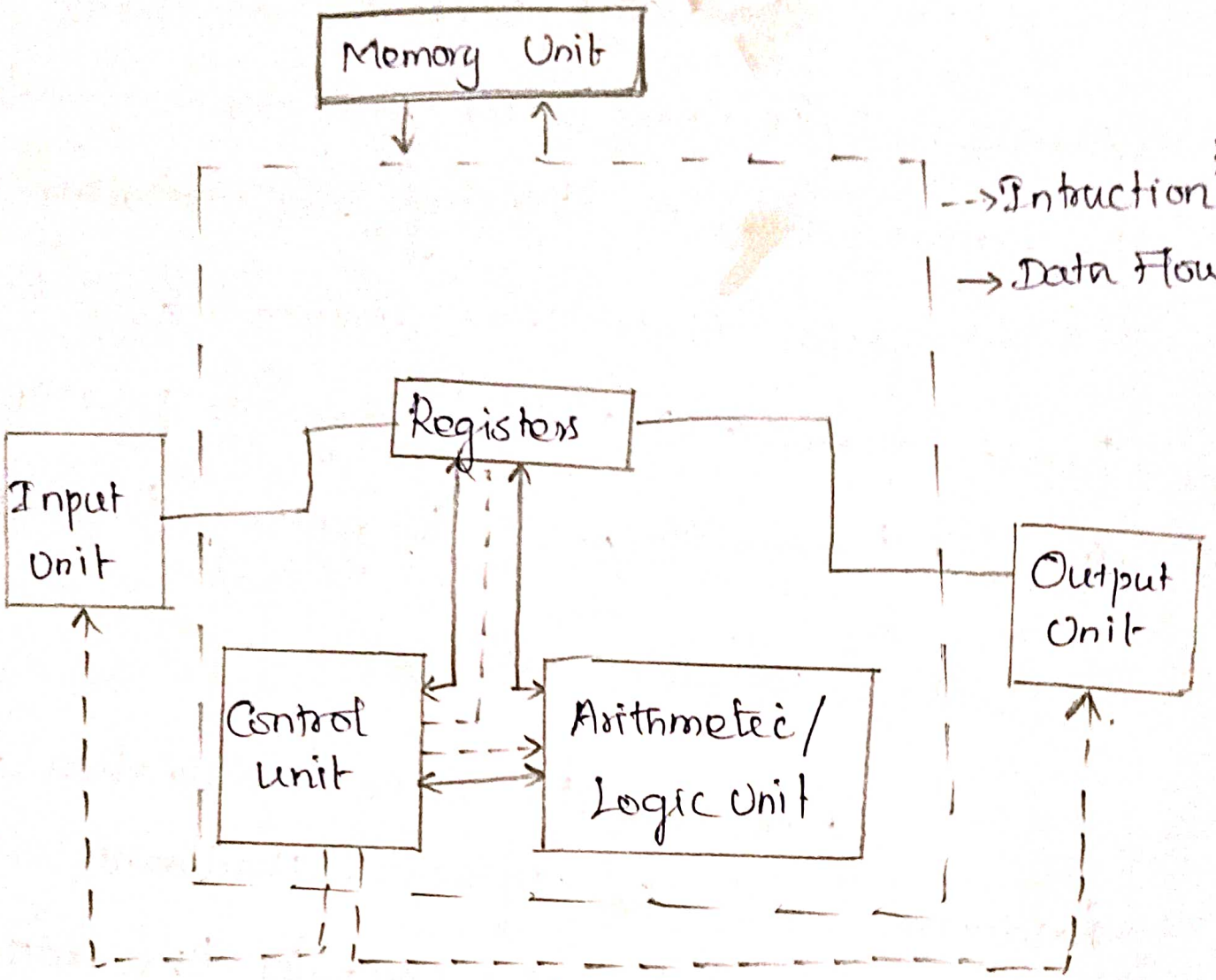



Fig: The Computer System Interaction.

1. 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.

2. Central Processing Unit (CPU)

CPU or the processor is often called the Brain of Computer.

CPU Consists of

(i) Arithmetic Logic Unit (ALU)

ii) Control Unit (CU)

iii) Set of Registers.

(i) Arithmetic Logic Unit

ALU Consist of Arithmetic unit and Logic unit

Arithmetic unit performs arithmetic operations

→ Addition, Subtraction, Multiplication and division.

The Logic unit performs logic operations

→ Comparision of Numbers, letters and characters.

(ii) Control Unit

CU coordinates the input and output devices of a computer.

The control unit organizes the processing of data and instructions.

(iii) Register:

Registers store data, Instructions, addresses and intermediate results of processing so registers are often called CPU's Working Memory.

3. Memory Unit:

The Memory unit consist of cache memory, primary memory and secondary memory.

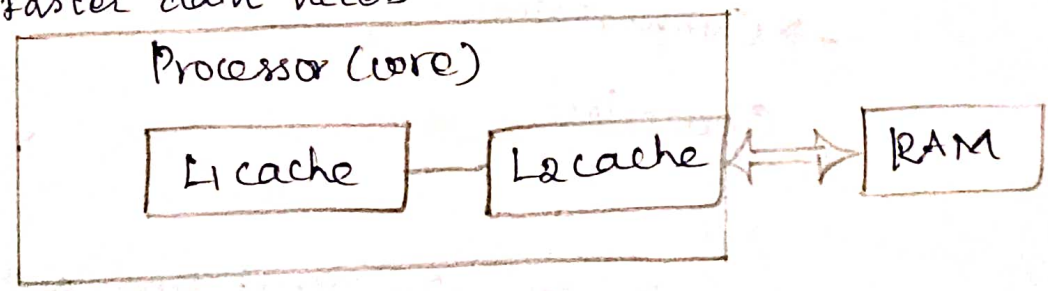
Memory unit stores the data and instructions, intermediate results and output temporarily during processing of data. This memory called Main Memory / Primary Memory.

Example: RAM
ROM.

(i) Cache Memory :→ High speed memory

→ cache memory is placed between RAM and CPU

→ faster data access




ii) Primary Memory → is the main memory of computer store data temporarily

iii) RAM → it provides temporary storage for data and instructions.

→ it is volatile

→ It stores data when computer is on

→ limited storage capacity, due to high cost.

iii) ROM → it provides permanent storage

→ it is not volatile

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

Secondary Memory:

It stores data and instructions permanently

It provides back up storage for data & instruction

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 serves as the first step in systematic process to identify, evaluate the problem and explore potential solutions.

It consist two steps

1. Identifying and acknowledging that there is a problem
2. Developing a problem identification statements

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.

Pattern Recognition:

The second step is pattern recognition are identified within the problem.

Abstraction:

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

Algorithm Design:

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

Decision problem:

A decision problem is a computational problem where the answer for every instance is either yes or no.

Search problem:

In search problem, the answers can be arbitrary strings.

Example : Algorithms to find the greatest among three numbers.

Algorithm

Step 1 : Start

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

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

Step 4 : Compare A and C. If A is the 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.

Properties of an Algorithm:

1. Finiteness - An algorithm must be terminated after a finite number of steps.
2. Definiteness - Each step of an algorithm must be precisely defined.
3. Input - Initial data supplied by READ instruction initial value using the SET instruction.

## 1.3 PROBLEM SOLVING TECHNIQUE

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

1. Algorithm
2. Flowchart
3. Pseudocode

## 1.4 ALGORITHMS :-

Algorithm is a step by step procedure for solving any problem.

Algorithm is an ordered sequence of finite, well defined, unambiguous instructions for completing a task.

Guidelines for Writing Algorithm:

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 seperate line

Steps should be numbered at step1, step 2 and so on.,

4. Output - After executing all the steps of the algorithm at least one output must be obtained.

5. Effectiveness - The operations to be performed in the algorithm can be carried out manually in finite intervals of time.

Advantages of Algorithm:

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

It is easy to debug.

It is independant of programming languages.

1.5 Building Blocks of ALgorithms:

The algorithm can be constructed from basic building blocks.

The building blocks are

Statements

State

Control flow

Functions

Statements :

An algorithm is a sequence of instructions to accomplish a task or solve a problem. It consist finite number of statements.

State :

Computational processes in the real-world have state. As a process evolves, the state changes.

In an algorithm the state of a process can be represented by a set of variables.

State is a basic and important abstraction.

Control Flow :

The statement to be executed next may depend on the state of the process. Thus, the order in which the statements are executed may differ from the order in which they are written in the algorithm.

This order of execution of statements is known as the control flow.

Three control flow statements

i) Sequence control flow
ii) Selection Control flow
iii) Iteration Control flow.

Functions :-

A function is a block of organized, reusable code that is used to perform a similar task of some kind.

Functions avoid the repitition of some codes over and over. It helps easy debugging, testing and understanding of the program.

Functions reduce program size and the program development time.

Benefits

→ Reduction in line of code
→ Code reduce
→ Better readability
→ Information hiding
→ Easy to debug and test

Example

Algorithm of two numbers using function

Main function()
Step 1: Start
Step 2: Call the function add()
Step 3: Stop
Sub function add()
Step 1: Function Start
step 2: Get a, b Values
Step 3: add c=a+b
Step 4: Print C Step 5: Return.

## 1.6 Notations

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

Example : Pseudocode, Flowchart

### Pseudocode

Pseudocode is short, readable and formally styled English language used for explaining an algorithm. Natural language with

### Preparing a Pseudocode

Pseudocode is written using structured English

Pseudocode uses some keywords to denote programming process.

Input : INPUT, GET, READ and PROMPT

Output : OUTPUT, PRINT, DISPLAY and SHOW

Processing : COMPUTE, CALCULATE, DETERMINE, ADD, SUBTRACT, MULTIPLY and DIVIDE

Initialize : SET and INITIALISE

Incrementing : INCREMENT.

The keywords should be capitalized

There are 3 control structures used in Pseudocode

a. Sequence Control structure

b. Selection control structure

c. Iteration control structure

Sequence Control structure:

In sequence control structure, the statements are executed one after another in the same order as they are written from top to bottom. The statements in sequence control structure are executed exactly one.

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 structures:

In 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.

Two main selection 1. IF-THEN-ELSE statement

2. CASE statement

## IF-THEN-ELSE Statement

If the condition is true, THEN part is executed. Otherwise, the ELSE part is executed.

### Syntax

```
IF Condition THEN
        process 1
ELSE
        process 2
END IF
```

## The Case Statement:

The case statement is used, when many numbers of conditions to be checked.

### Syntax

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

## Iterative Control Structure :

In iterative control flow, a set of statements are repetitively executed based upon a condition. If a condition evaluates to true, the set of statements (true block) is executed again and again. As soon as the condition becomes false the repetition stops. This is also known as looping statement or iteration statement.

There are 2 iterative control statements

WHILE
DO-WHILE

**Syntax**

```
WHILE condition
    statements
END WHILE
```

In WHILE loop, the condition is executed at the beginning of the loop. If the condition is false then the loop will not be executed.

**Syntax**

```
DO statements
WHILE condition
END DO
```

In DO-WHILE loop, the condition is executed at the end of the loop, so the loop is executed atleast once irrespective of the loop.

Advantages of Pseudocode:

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

Limitations of Pseudocode:

It is difficult to understand and manipulate pseudocode as compared to algorithm and flowchart.

## FLOWCHART:

A flowchart is a diagrammatic representation of the logic for solving a task.

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

Flowchart symbols:

A flow chart is drawn using different kinds of symbols. Every simple used in a flowchart is for a specific purpose.

| Symbol | Symbol Name | Description |
|---|---|---|
|  | Flow lines | Used to connect symbols |
|  | Terminal | Used to start, pause or halt in the program logic |
|  | Input/ Output | Represents the information entering or leaving the system |
|  | Processing | Represents arithmetic and logical instructions |
|  | Decision | Represents a decision to be made |
|  | Connector | Used to join different flow lines |
|  | Sub function | used to call function |

Rules for drawing flowchart

1. The flowchart should be clear, neat and easy to follow.
2. The flowchart must have a logical start and finish.
3. Only one flow line should come out from a process symbol.
4. only one flow line is used with a terminal symbol.
5. Intersection of flow lines should be avoided.

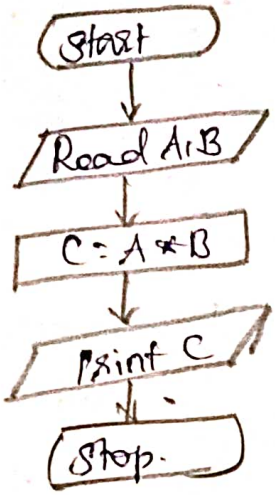
Advantages of flowchart

1. Communication- Flowcharts are better way of Communicating the logic of a system to all concerned.
2. Effective analysis
3. Paper documentation
4. Efficient coding
5. Proper debugging
6. Efficient program Maintanence

## Sequence control structure

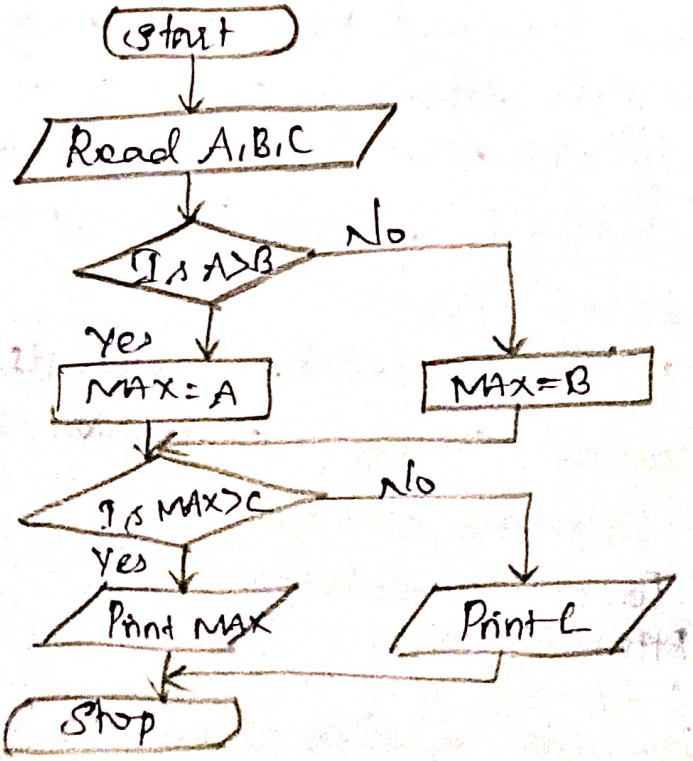
In sequential control structure, a sequence of statements is executed one after another in the same order as they are written. The instructions in sequence control flow are executed exactly once.

Ex.



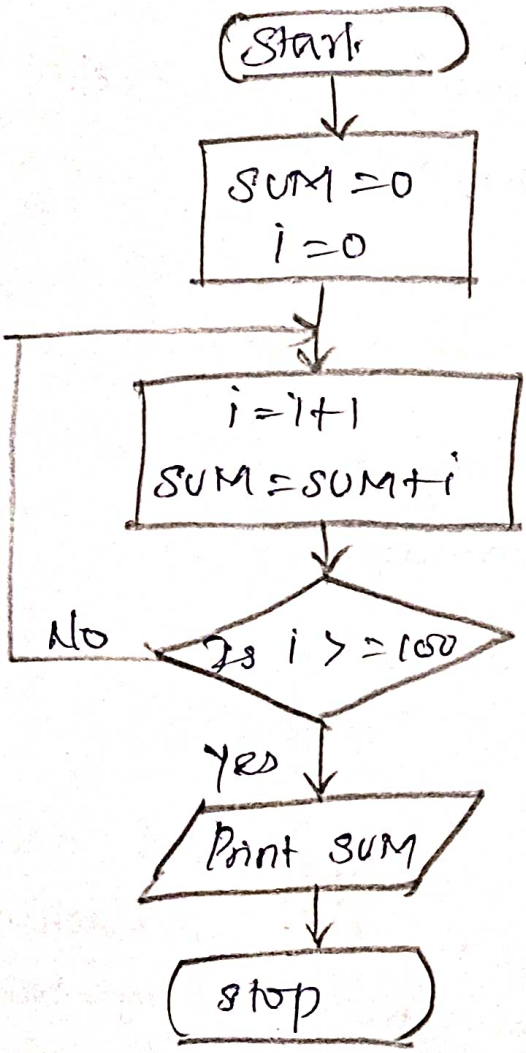

## Selection Control Structure :

In selection control, 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.



# Iterative Control Structure

In iteration control flow, a set of statements are repetitively executed based upon a condition. If a condition evaluates to true, the set of statements is executed again and again.



# PROGRAMMING LANGUAGES:

A progeamming language is a set of symbols and the rules for instructing a compuler to perform specific tasks.

The user has to communicate with the computer using language which it can understand.

The programmers have to follow all the specified rules before writing program

## Types of progeamming language:

Machine level language
Assembly Language
High level language.

## Machine language:

The computer can understand only machine language which uses 0's and 1's. In machine language the different instructions are formed by taking different combinations of 0's and 1's.

Advantages:

→ Translation Free: Machine language is the only language which the computer understands. For executing any program written in any programming language, the conversation to machine language is necessary. The program written in machine language can be executed directly on compuler. In this case any conversation process is not required.

High speed: The machine language program is translation free. since the conversation time is saved, the execution of machine language program is extremely fast.

Disadvantages:

Hard to find errors

Time Consuming process

Machine dependent: According to architecture used, the compuler differs from each other. So Machine language differ from Computer to Computer. So a program developed for a particular type of computer may not run on other type of computer.

Assembly Language:

To overcome the issue in programming language and make the programming process easier, an assembly language is developed which is logically equivalent to Machine language but it is easier for people to read, write and understand.

Ex: ADA a, b

Assembler → is the program which translates assembly language instruction in to a machine language.

Advantages:

Easy to understand and use

It is easy to locate and correct errors.

Disadvantages:

Machine dependent - The assembly language program which can be executed on the machine depends on the architecture of that Computer.

Hard to learn - It is machine dependent, so the programmer should have the hardware knowledge to create applications using assembly language.

Less efficient:

Execution time of assembly language program is more than machine language program. Because assembler is needed to convert from assembly lang. to machine language

## High level language

It contains English words and symbols. The specified rules are to be followed while writing program in high level language.

### Translating high level language to Machine language:

The programs that translate high level language in to Machine language are called interpreter or compiler.

**Compiler :-** A compiler is a program which translates the source code written in high level language into object code which is in machine language program. Compiler reads the whole program written in high level language and translates it to machine language. If any error is found it display error mosage on the screen.

**Interpreter:**

Interpreter translates the high level language in line by line manner. The interpreter translates a high level language statement in a source program to a machine code and executes it immediately before translating the next statement.

**Advantages:**

Readability – High level language is closer to natural language so they are easier to learn and understand.

**Machine Independent :-**

The advantage of being portable between machines

**Easy debugging -**

Easy to find and correct error in high level language.

Disadvantages:

Less efficient - The translation process increases the execution time of the process. Program in high level language require more memory and take more execution time to execute.

Categories of programming language:

1. Interpreted programming language
2. Functional programming language
3. Compiled programming language
4. Procedural programming language
5. Scripting programming language
6. Markup programming language
7. Concurrent programming language
8. Object oriented programming language.

## ALGORITHMIC PROBLEM SOLVING

Computer in solving a problem depends on how correctly and precisely we define the problem, design a solution (algorithm) and implement the solution (program) using a programming language.

Thus the problem solving is the process of identifying a problem, developing an algorithm for the identified problem and finally implementing the algorithm to develop a computer program.

When problem are simple and easy we can easily find the solution.

But a complex problem requires a methodical approach to find the right solution.

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 required for solving a problem using a computer

Step 1: Obtain a description of the problem

Step 2: Analyze the problem

Step 3: Develop a high level algorithm

Step 4: Refine the algorithm by adding more details

Step 5: Review the algorithm.

Obtain a description of the problem:

The algorithmic problem solving process starts by obtaining a description of the problem. This step is much more difficult that it appears. In the software development process, the developer must create an algorithm that will solve the problem.

Client → refers to a person who wants a final a solution to a problem

Developer → refers to a person who finds a way to solve the problem.

Analyze the problem:

To read and analyze the problem statement carefully in order to list the principal components of the problem and decide the core functionalities that our solution should have.

By analysing a problem, we would be able to figure out what are the inputs that our program should accept and the outputs that it should produce.

Step 3: Develop a high level algorithm:

It is essential to device a solution before writing a program code for a given problem. The solution is represented in natural language and is called an algorithm. We start with a tentative solution plan and keep on refining the algorithm until the algorithm is able to capture all the aspects of the desired 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. For larger, more complex problems, it is common to go through this 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

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 a good way to develop skills that can be applied to the next problem.

For example, an algorithm that computes the area of circle having radius 5.2 meters (formula $\pi * 5.22$) solves a very specific problem, but an algorithm that computes the area of any circle (formula $\pi \times r2$) solves a more general problem.

→ can this algorithm be simplified?

For example, one formula for computing the perimeter of a rectangle is

length + width + length + width

A simpler formula would be:

(length + width)

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

SIMPLE STARTERGIES FOR DEVELOPING ALGORITHMS

There are various kind of algorithm development techniques

1. Iteration formulated & used for different types of problem.

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

Computers execute the same set of statements again and again by putting them in a loop.

In general, loops are classified as

1. Counter - controlled loops
2. Sentinel - controlled loops.

Counter - Controlled Loops:

Counter controlled loops are so named because they use a control variable, known as the loop counter, to keep a track of loop iterations.

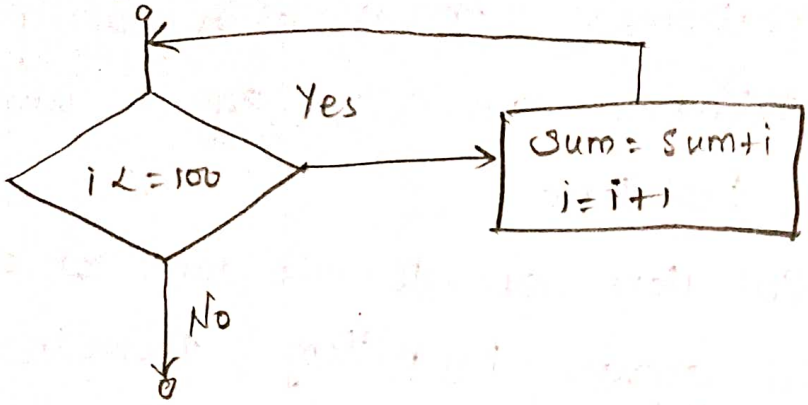
The counter controlled loop starts with the initial value of the loop counter and terminates when the final value of the loop counter is reached.

Since the counter controlled loops iterate a fixed number of times, which is known in advance, they are also known as definite repetition loops.

Example Consider a program segment to find the sum of first 100 integers

```
i=1
sum=0
while(i<=100);
```

The flowchart in Figure illustrates the flow of control in the while structure that corresponds to the preceding while structure.




Algorithm : To find the sum of first N integers

1. Start
2. Read N
3. Assign sum = 0, i = 0
4. Calculate i = i+1 and sum = sum + i
5. Check whether i >= N, if not repeat step 4, otherwise go to next step.
6. Print the value of sum
7. Stop.

Recursion:

Recursion is a powerful programming technique that can be used to solve the problems that can be expressed in terms of similar problems of smaller size.

Example

Consider a problem to find the factorial of a number n. The problem of finding the factorial of n can be expressed in terms of a similar problem of smaller size as $n! = n \times (n-1)!$.

Recursion is classified according to the following criteria:

1. Whether the function calls itself directly or indirectly (i.e, 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.

Example: Recursive algorithm for finding the factorial of a num

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.

TOWER OF HANOI

Tower of Hanoi is one of the classical problems of computer science. The problem states that

1. There are three stands (Stand 1, 2, and 3) on which a set of disks, each with a different diameter, are placed
2. Initially, the disks are stacked on stand 1, in order of size, with the largest disk at the bottom.

The initial structure of Tower of Hanoi with 3 disks

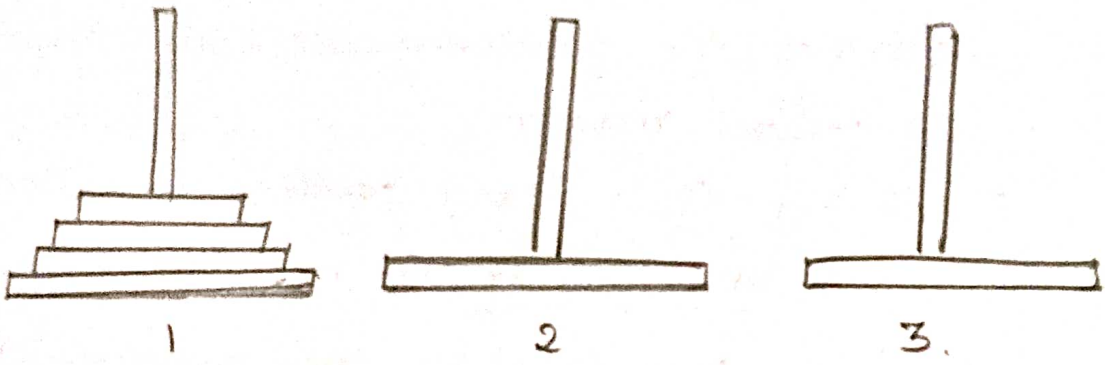



Fig: Tower of Hanoi with three disks.

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

1. More only one disk at a time
2. A larger disk cannot placed on top of a smaller disk
3. All disks except the one being moved should be on a stand.

The recurrence relation for solving the Tower of Hanoi problem can be written as:

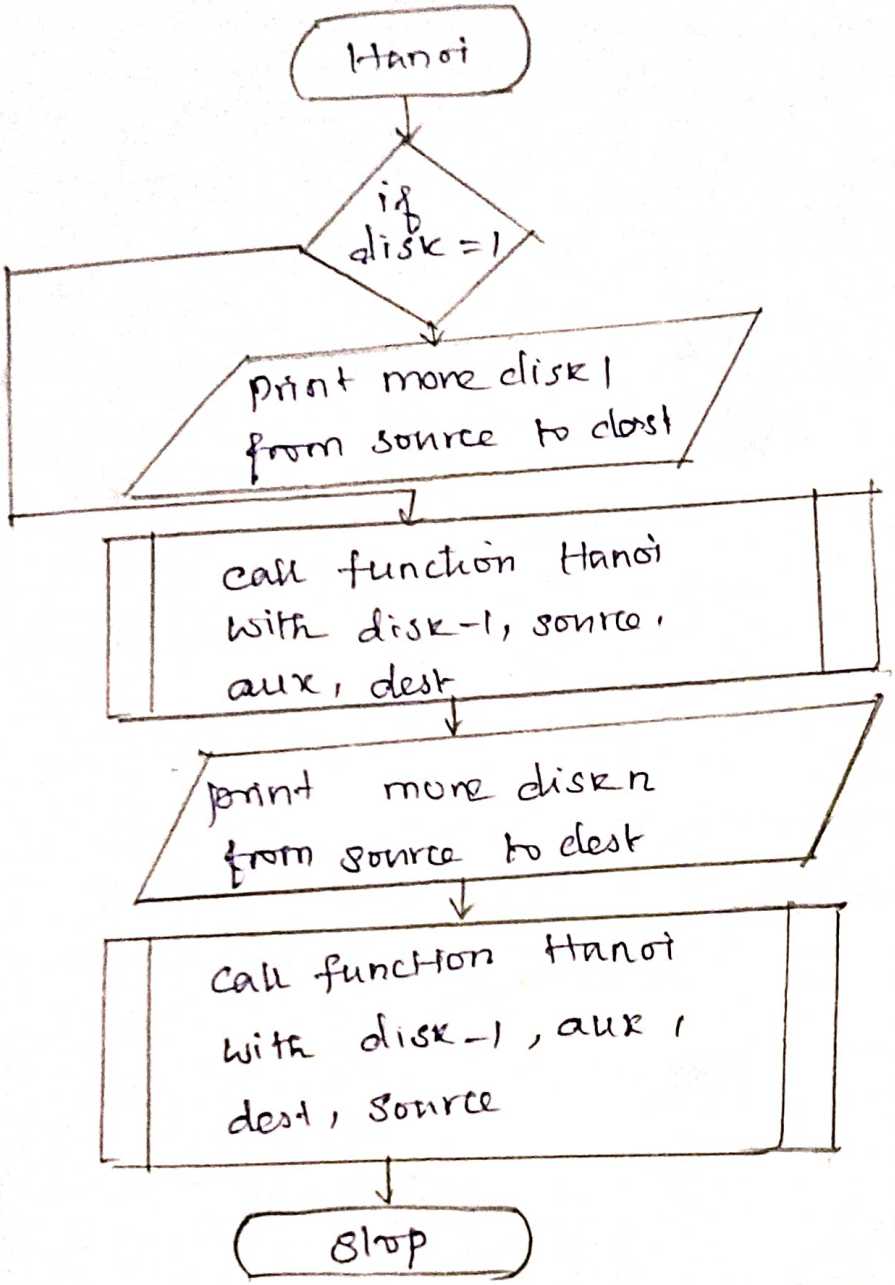
$$\text{Tower of Hanoi(disks)} = \begin{cases} \text{move the disk} & \text{if disks} = 1 \\ \text{Tower of Hanoi(disk-1)} & \text{if disks} > 1 \end{cases}$$

Recursive algorithm (Tower of Hanoi)

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 source to dest
   - 2.3 Recursively call Hanoi (disk-1, aux, dest, source)

Flow chart




Pseudocode:

```
START
Procedure Hanoi (disk, source, dest, aux)
    IF disk == 1. THEN
        move disk from source to dest
    ELSE m
        Hanoi (disk-1, source, aux, dest)
        move disk from source to dest
        Hanoi (disk-1, aux, dest, source)
    END IF
END Procedure
STOP
```

# Illustrative Problems

1. Finding Minimum Number in a List

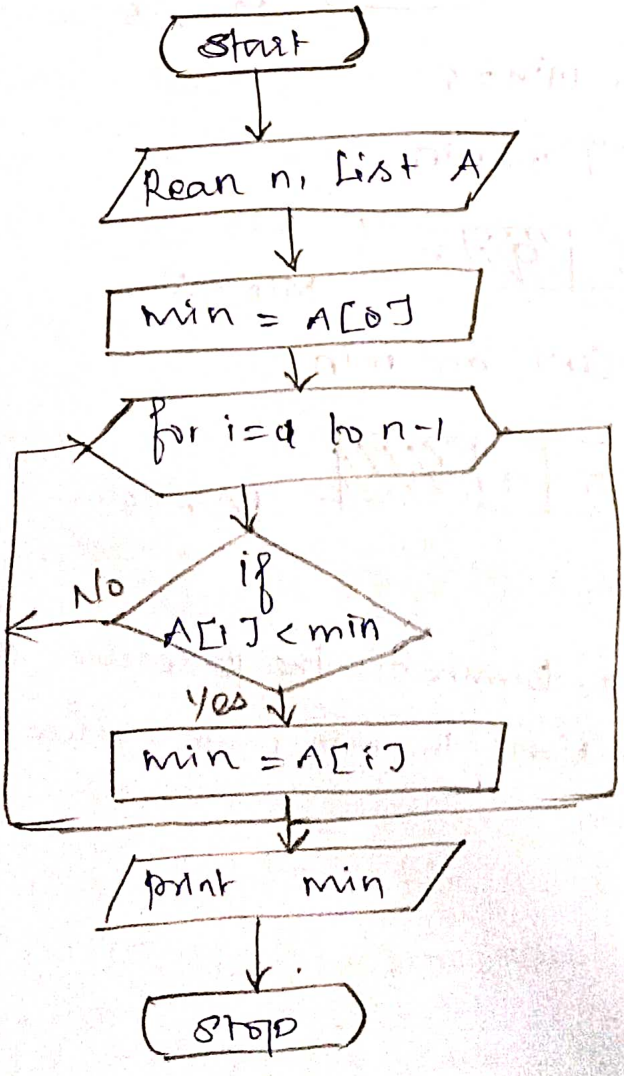
Problem Statement:

Finding the minimum number in a list is an example of selection problem.

Algorithm:

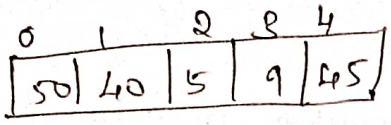
1. Assign the first value of the list as minimum value.
2. Compare this value to the other values starting from second value.
3. When a value is smaller than the present minimum value, then it becomes the new minimum.
4. Print the minimum value.

Flow Chart:

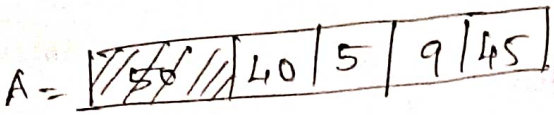


## Example

Consider an array




Step 1 : Assign min = A[0]

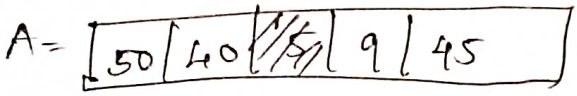



Min = 50

Step 2 : Compare A[1] and min


40 < 50. Now Min = 40

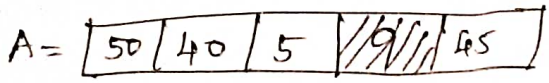
Step 3 : Compare A[2] and min




Min = 5

5 < 40 Now min = 5

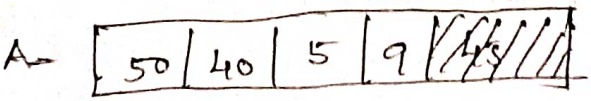
Step 4 : Compare A[3] and min




Min = 5

9 > 5.

Step 5 : Compare A[4] and min




Min = 5

45 > 5 . min = 5

The end of the list, terminate the procedure of finding minimum value. Now the minimum value is 5.

```
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.
```

2. Inserting a Card in a List of Sorted Cards:-

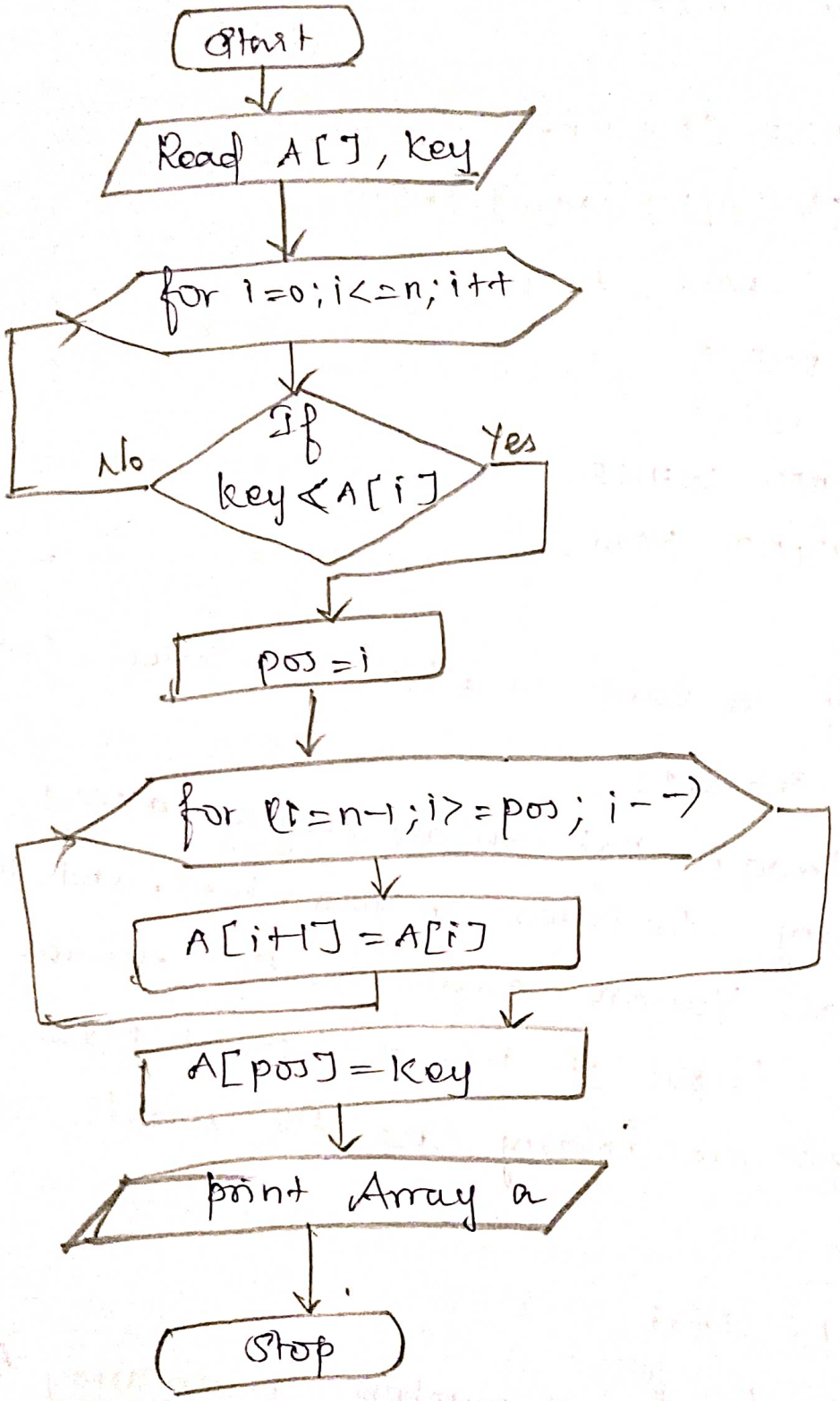
Problem Statement:

Imagine that you are playing a card game. You are holding the cards in your hand, and these 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

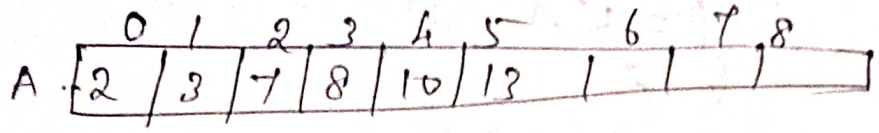
Step 1 : Start
Step 2 : Read all numbers in an array A.
Step 3 : Read key.
Step 4 : From first position to the end of the array Compare key and array element.
Step 5 : If key < A[i] then assign i to pos, and return pos.
Step 6 : from n-1 to pos move the elements one position down
Step 7 : Now store key value in pos location.
Step 8 : Stop.

# Flow chart


Start
Read A[], key
for i=0; i<=n; i++
If
key < A[i]
No
Yes
pos = i
for (i=n-1; i>=pos; i--)
A[i+1] = A[i]
A[pos] = key
print Array a
Stop

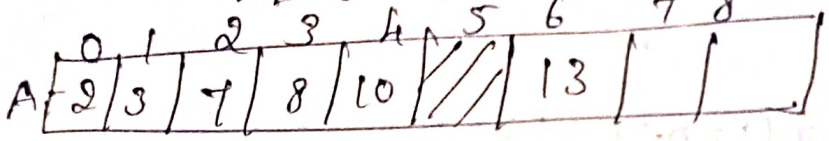
## Example

Consider the array from index 0 through index 5 is already sorted, and need to insert the key element 5 into this sorted sub-array, so that the sub array from index 0 through index 6 is sorted.

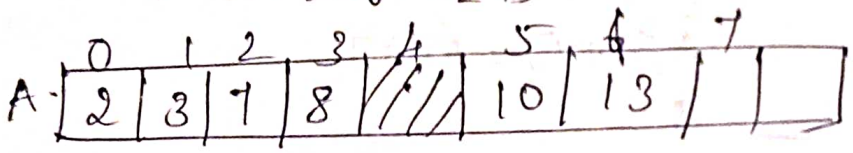



key = 5, pos = 2

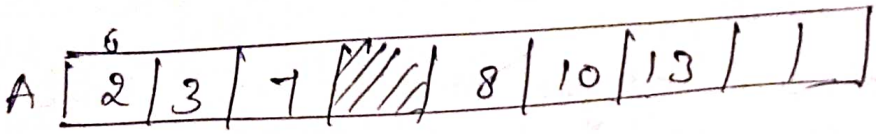
Step 1: i=5 A[6] = A[5]



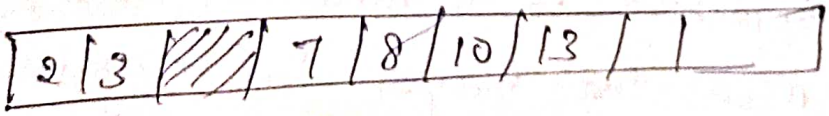

Step 2: i=4 A[5] = A[4]



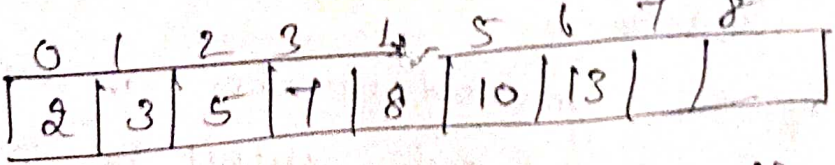

Step 3: i=3 A[4] = A[3]




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




Step 5: Now insert the key value.




This slide operation first copies the element one position to the right.

Pseudocode

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

3. Guessing an Integer Number in a Range:

Problem Statement-

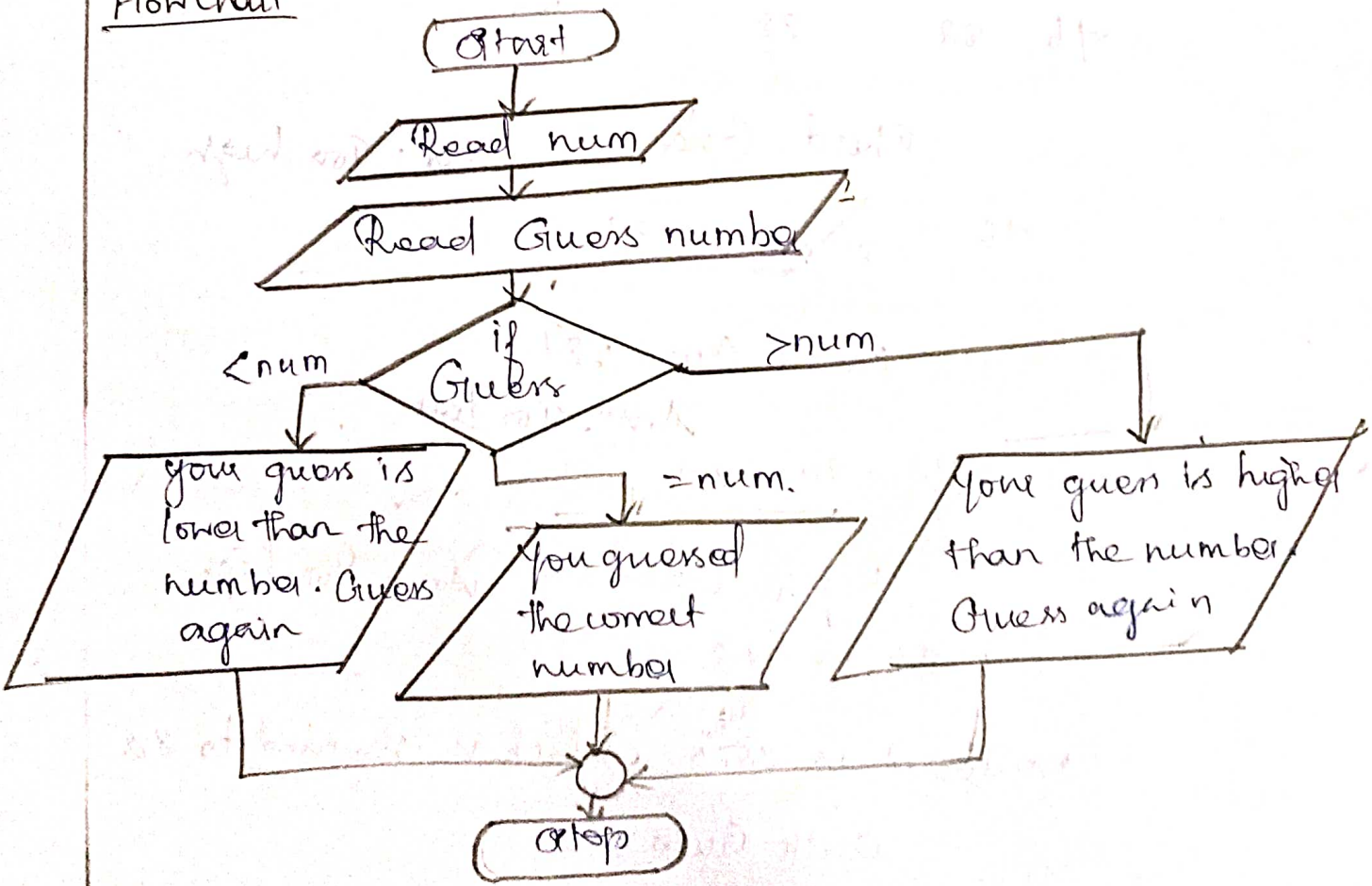
The objective is to randomly generate integer number from 0 to n. Then the player has to guess the number. If the player guesses the number correctly output an appropriate message.

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".

## Algorithm

1. Start
2. Generate a random numbers 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

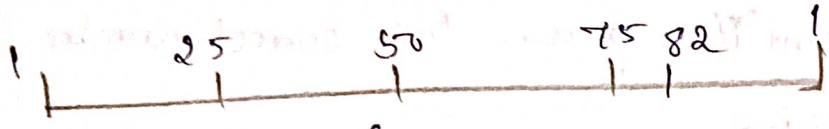
## Flowchart



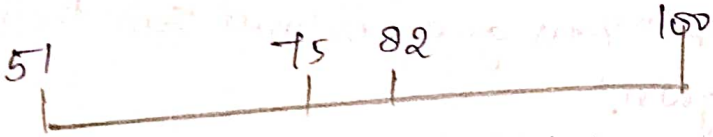
Example

Consider to guess a number between 1 and 100. The number selected for guess is 82. Now the player wants to guess this number.

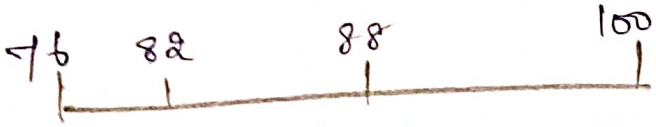
Guessing Game - secret number is 82.



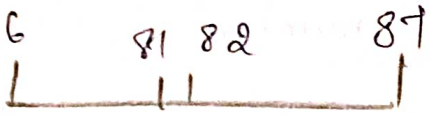

First Guess - 50 Ans: Too low




Second Guess: 75 Ans: Too low

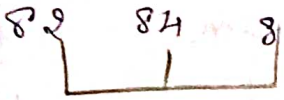



Third Guess: 88 Ans: Too high




Fourth Guess: 81

Ans: Too low




Fifth Guess: 84 Ans: Too high


Midpoint is 82.5. Which is rounded to 82.

Sixth Guess: 82

Answer: Correct.

UNIT - 2 [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 value of two variables, circulate the values n variables, distance between two points.

## DATA TYPES, EXPRESSIONS, STATEMENTS

### Python Interpreter and Interactive Mode

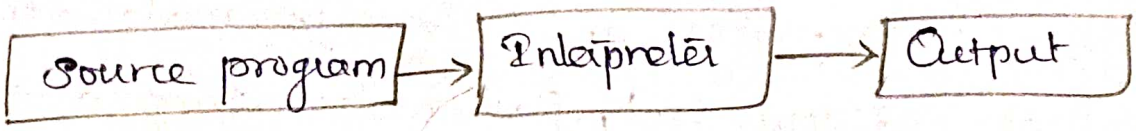
Python has two basic modes: interpreter and Interactive mode.

Interpreter mode - The interpreter mode is the mode where the scripted and finished '.py' files are run in the python interpreter.

Interactive mode - The interactive mode is the command line shell which gives immediate feedback for each statement, feed in the activity memory. As new lines are fed into the interpreter, the fed program is evaluated both in part and in whole.

Python Interpreter mode:

The python interpreter mode is the 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.




Python 3.6.0 Shell

File Edit Shell Debug Options Window Help

Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016 07:08:10) [MSC v.1900 32 bit Intel] on win32

Type "copyright", "credits" or "license()" for more information

>>>

Fig: Python Interpreter Window.

The first three 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 begin with 2, if the version is Python 2.

The last time >>> 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-

```
>>> python sum.py
Enter 2 numbers
6
3
The sum is 9.
```

# 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 script.

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 (...).

Example

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

## 2. DEBUGGING

A programmer 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 categorized as:

i) Syntax errors
ii) Logical errors
iii) Runtime errors

Syntax errors:

The interpreter interprets the statements only if it is syntactically (as per the rules of Python) correct. If any syntax error is present (the interpreter shows error message(s)) and stops the execution there.

Such errors need to be removed before the execution of the program.

Logical 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.

The only evidence to the existence of logical errors is the wrong output.

Runtime Error –

A runtime error causes abnormal termination of program while it is executing.

Runtime error is when the statement is correct syntactically, but the interpreter cannot execute it. Runtime errors do not appear until after the program starts running or executing

VALUES:

3. Values :-

Values are the basic units of data, like a number or a string that a program manipulates.

Examples :

2, 42.0 and 'Hello World!"

2 is an integer, 42.0 is a floating point number, "Hello world" is a string.

TYPES :-

Every value belongs to a specific data type in Python. Data type identifies the type of data values a variable can hold and the operations that can be performed on that data.

Datatype is a category of values.

Data types are

* Numbers
  - → Integer type (int)
  - → Floating point (float) type
  - → Complex type
* String List
* Tuple
* Set
* Boolean
* None types

Python has 3 number type

Integer numbers. Floating point numbers,
Complex numbers.

Integer Type:

An integer type (int) represents signed whole numbers
An integer represents both positive and negative numbers.

An integer in decimal (base 10

ex → 25000

An integer in octal base 8)

ex → 0o177

An integer in hexadecimal (base 16)

ex → 0x77

An integer in binary (base 2)

ex → 0b0101

Floating Point Type:

A floating point (float) type represents numbers with fractional part. It has a decimal part & fractional part. Ex: 3.0 or 3.17 or −28.72

Comm It is stored with three parts

A sign + or −

A mantissa

An exponent

Complex Type:

The complex data type 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.

A sign + or −

Ex: 5 + 14j

```
>>> z=5+14j
>>> z.real
5.0
>>> z.imag
14.0
>>> print(z)
(5+14j)
>>> print((2+3j)*(4+5j))
(-7+22j)
```

## Boolean Type

A Boolean type represents special values "True" and "False". They are represented as 1 and 0 and can be used in numeric expression as value.

Ex. 2<3 is True.

boottype.py

```
x=(1==True)
y=(1==False)
a=True+4
b=False+10
print("x is",x)
print("y is",y)
print("a:",a)
print("b:",b)
```

output

```
x is True
y is True
a:5
b:10
```

In Python, True represents the value as 1 and False as 0

## String Type:

A string type represents sequence of characters surrounded by qotes. Those characters may be alphabets digits or special characters including spaces.

In Python, string values are enclosed in single quotes, double quotes & triple quotes.

Ex.
Using Single quotes: 'HELLO'
Using Double quotes: "HELLO"
using Triple Quotes: """Hello Every One. Wolcome to Python Programming""".

## List Type:

List is a sequence of data type.

List is a sequence of items seperated by commas and the items are enclosed in square brackets [].

Ex: >>> Address=['231', 'Carmel Nagar', 'Nagercoil', '629004']
['231', 'Carmel Nagar', 'Nagercoil', '629004']
>>> print Address

## Tuples

A tuple is another sequence data type similar to list. A tuple consist of a sequence of items seperated by commas and items are enclosed in parenthesis ().

List are enclosed in brackets [] & their elements & size can be changed.

Tuple are enclosed parenthesis () & cannot be changed.

Example

Tuple with string values

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

Tuple with single character

```
>>>T = ('p','y','T','H','O','N')
>>> print T
('p', 'y', 'T', 'H', 'O', 'N')
```

## Sets

Set is an unordered collection of items seperated by commas and items are enclosed in curly brackets {}.

example

```
>>> x = set ("PYTHON PROGRAMMING")
>>> print(x)
{'p', 'I', 'M', 'y', 'G', 'N', 'R', 'H', 'O', 'A', 'T', ' '}
>>> type(x)
<class 'set'>
```

None type

None is a special data type with a single value. The None data type means nonexistent, not known or empty.

Example

```
>>> x = None
>>> x
>>> print x
None.
```

## VARIABLES :-

A variable is an identifier, which holds a value. In a programming, a value is assigned to a variable.

A 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.

Ex: Roll_no, GrossPay, a1, Salary.

### Creating variables

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

### Syntax

```
variable = expr
```

→ Variable is an identifier & 'expr' is an expression.

## Expressions :-

In Python, most of the lines or statements are written in the form of expressions. Expressions are made up of operators and operands. Expressions are evaluated according to operators.

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

### Example

```
A * B + C
```

Where *, + are operator;

A, B and C are operands.

Types of expressions:

(i) Based on the position of operators in an expression

Three types

→ Infix expression: The operator is placed in b/w the operands

→ Ex: a = b+c

→ Prefix expression: The operator is placed before the operands : a = +bc

→ Postfix expression : The operator is placed after the operands : a = bc+

(ii) Based on the type of the result obtained on evaluating an expression.

This type of expressions is mathematical operations like +, −, *, / etc.,

ARITHMETIC EXPRESSION

Ex Arithmetic operation (arithmetic.py)

```
num 1 = 20
num 2 = 30
Sum = num 1 + num 2
print 'the value of Sum is:', Sum
```

Output

The value of sum is: 50

num 1, num 2 → operands

+ → operator

2. RELATIONAL / CONDITIONAL EXPRESSION:

This expression compares two statements using relational operators like >, <, >=, <=, etc.,

Relational Expression (relational.py)

```
a = int (input ('Enter first number'))
b = int (input ('Enter first number'))
Flag = (a > b)
print ('Is %d greater than %d: %s' %(a, b, Flag))
```

o/p
Enter first num 20
Enter first num 31
Is 20 greater than 31: False

3. Logical Expression:

The Logical expression uses the logical operators like and or, or not. The logical expression also produces a Boolean result like either True or False

logical expression (logical.py)

```
a=20
b=30
c=23
d=21
Flag=((a>b) and (c>d))
print(" The logical expression:((a>b) and (c>d))returns:",Flag)
```

Output

The logical expression:((a>b) and (c>d))returns. False.

4. Conditional Expressions:

The conditional expression is also called relational expression. This expression is used with branching (if, if-else, etc.,) and looping (while) statement. A conditional ~~statement~~ expression always produces two results either True or False depending upon the cond.

Conditional expression( conditional.py)

```
Age=input("Enter your age")
if (Age>=18)
print "you can vote."
else:
   print "You can't vote."
```

Output

Enter your age:25

You can vote.

# TUPLE

In Python, the tuple may be defined as a finite, static list of numbers or string. A tuple is similar to a list and it contains immutable sequence of values seperated by commas.

Example

```
>>> t = ('a','b','c','d','e')
>>> max(5,8,9)
9
>>>min(9,99,999)
9
```

## Creating a tuple

i) To create a tuple with a single element, a comma is to be included at the end.

```
>>> t1 = 'a'
```

ii) The built-in function tuple with no argument creates an empty tuple.

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

iii) If the argument is a sequence (string, list or tuple), the result is a tuple with the elements of the sequence

```
>>>t = tuple('Jovita')
>>>t
('J','o','v','i','t','a')
```

## Operations on tuple

The bracket operator indexes an element

```
>>>t=('a','e','i','o','u')
>>>t[0]
'a'
>>> t[3]
'o'
```

## PRECEDENCE OF OPERATORS (ORDER OF OPERATORS)

Evaluation of expression is based on precedence of operators. When an expression contains different kinds of operators, precedence determines which operator should be applied first. Higher precedence operator is evaluated before the lower precedence operator.

For mathematical operators, Python follows mathematical convention.

→ Parenthesis have the highest precedence and can be used to force an expression to evaluate in the order.

Example: $5 * (9-3) = 30$

Since expression in parenthesis are evaluated first.

→ Exponentiation has the next highest precedence

Example: $1 + 2 ** 3 = 9$, not 27 and

$2 * 3 ** 2 = 18$, not 36.

→ Multiplication and Division have higher precedence than Addition and Subtraction

Example: $2 * 3 - 1 = 5$, not 4 and

$6 + 4 / 2 = 8$, not 5

→ Operators with the same precedence are evaluated from left to right.

## COMMENTS

A comment statement contains information for persons reading the program.

Comments make the program easily readable and understandable by the programmer and non-programmer who are seeing the code.

In Python, a comment starts with # (hash sign). Everything following the # till the end of that line is treated as a comment and the interpreter simply ignores it while executing the statement. They have no effect on the program results.

Example

```
# swap.py
# Swapping the values of a two variable program
#written by G. Sugitha, April 2015.
```

MUTABLE AND IMMUTABLE TYPES:

Sometimes we may require changing or updating the values of certain variables used in program. However, for certain data types, Python does not allow us to change the values (once a variable of that type has been created and assigned values.

1. Mutable type or
2. Immutable type

Mutable Type: (Changeable)

Variables whose values can be changed after they are created and assigned called mutable or an object whose state or value can be changed in place is said to be mutable type. Lists and Dictionaries are mutable, that means user can del / add / edit any value inside the list and dictionaries.

## Immutable type (Unchangeable)

Variables whose values cannot be changed after they are created and assigned are called immutable or an object whose state or value cannot be changed in place is said to be immutable type. When an attempt is made to update the value of an immutable variable, the old value is destroyed and a new variable is created by the same name in memory. Integers, Float, Boolean, Complex, Strings, Tuples and Sets are immutable datatypes.

## Input function

→ 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]).

→ The prompt is an optional parameter

→ The prompt is an expression that serves to prompt the user for input. This is almost always a string literal which is enclosed within quotes (single or double) with parenthesis.

→ The result of input() function is returned a numeric value, and assigned to a variable.

Example

```
>>> num1 = input("Enter first number:")
Enter first number: 20
>>> num2 = input("Enter second number:")
Enter second number: 40
>>> print "Sum is:" num1+num2
Sum is: 60
```

2. raw_input():

The purpose of raw_input() function to read input from the standard input stream (the keyboard, by default).

Syntax

```
<variable> = raw_input(['prompt'])
```

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
```

## Illustrative Problems

1. Exchange the values of two variables

```
x=input ("Enter value of x:")
y=input ("Enter value of y:")
print ("Before exchange of x,y").
print ("x=", x)
print ("y=", y)
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
x= 67
Y=56
After exchange of x,y
x=56
Y=67
```

2. Circulate the values of n variable

```
def circulate(A,N):
  for i in range(1,N+1):
  B=A[1:]+A[0: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.sqr(((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 = 7.211102 55
```

# UNIT - 3

## CONTROL FLOW, FUNCTIONS, STRINGS

### 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 => operator in Python

= is an assignment operator & == is a relational operator.

Example

```
>>> 10 == 10
True
>>> 8 == 6
False.
```

### Operators:

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

Operands - The value that the operator operates on is called the operand

Ex: x = a + b where a, b, x are operands & + an operator

CLASSIFICATION OF OPERATORS:

1. UNARY OPERATORS - The unary operators on only one operand. Unary operator usually precede their single operand.

| Operator | Meaning |
|---|---|
| + | Unary plus |
| − | Unary Minus |
| ~ | Bitwise Inverse |

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 it is the way that negative numbers are stored in the computer, it changes a positive value to negative, and a negative value to positive.

Example

```
>>> ~4
-5
>>> ~-7
6.
```

Binary Operators:-

The binary operators operate on two operands. The binary operators in Python.

1. Arithmetic operators
2. Relational operators
3. Logical operators
4. Bitwise operators
5. Assignment operators
6. Special operators.

## 1. ARITHMETIC OPERATORS

Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication & division.

Arithmetic operators (arithmetic.py)

```
x = 25
y = 5
print('x+y =', x+y)
print('x-y =', x-y)
print('x*y =', x*y)
print('x/y =', x/y)
print('x//y =', x//y)
print('x**y =', x**y)
```

Output

```
x+y = 30
x-y = 20
x*y = 125
x/y = 5.0
x//y = 5
x**y = 9765625
```

## 2. Relational Operators:

The operators used to do comparision are called relational operators. These operators always result in a Boolean value (True or False). The relational relational operators are also called comparision operators.

relational operators (relational.py)

```
x=10
y=12
print('x>y is', x>y)
print('x<y is', x<y)
print('x==y is', x==y)
print('x!=y is' x!=y)
print('x>=y is', x>=y)
print('x<=y is', x<=y)
```

Output

```
x>y is False
x<y is True
x==y is False
x!=y is True
x>=y is False
x<=y is True.
```

3. Logical Operators:

The logical operators are used to logically relate the sub expressions. Logical operators are 'and', 'or' and 'not' operators. The logical operators operate according to the truth tables. These are most often used with if and while keywords.

Logical operator in Python.

| Operator | Meaning | Example |
|---|---|---|
| and | True if both the operands are true | x and y |
| or | True if either of the operands is true | x or y |
| not | True if operand is false | not x. |

Logical operators. (logical.py)

```
x=True
y=False
print('x and y is', x and y)
print('x or y is', x or y)
print('not x is', not x)
```

Output

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

4. Bitwise Operators :-

Decimal numbers are natural to humans whereas binary numbers are native to computers. Bitwise operators work with bits of a binary number. It operates bit by bit.

For ex - 2 is 010 in binary & 7 is 111

5. Assignment Operator

Assignment operators are used to assign values to variables, a=15 is a simple assignment operator that assigns the value 15 on the right to the variable 'a' on the left.

There are vaious Compound operators in Python, a+=15 that add to the variable and later assigns the same. It is equivalent to a=a+15.

Assignment operators in Python:

| Operator | Meaning | Equivalent to |
|---|---|---|
| = | x=5 | x=5 |
| += | x+=5 | x = x+5 |
| -= | x-=5 | x = x-5 |
| *= | x*=5 | x = x*5 |
| /= | x/=5 | x = x/5 |

6. Special Operators

Python language offers some special type of operators. They are

a) Identify Operators
b) Membership Operators.

Identify Operators:

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

is and is not are the identify operators in Python.

Identify Operator in Python:

| Operator | Meaning | Example |
|---|---|---|
| is | True if the operands are identical | x is True |
| is not | True if the operands are not identified | x is not True. |

Example. Identify operators (identify.py)

```
x1 = 5
y1 = 5
print(x1 is not y1)
print(x1 is y1)
```

Output

```
False
True
```

→(a) Memembership Operator:

Membership operators are used to test whether a value or variable is found in a sequence (string, list, tuple, set and dictionary).

→ in and not in are membership operator in Python.

Membership operators in Python.

| 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. |

Program : Membership operator (membership.py)

```
x = "Hello World"
y = {1:'a', 2:'b'}
print ('H' in x)
print ('hello' not in x)
```

Output

```
True
True.
```

## 4. CONDITIONAL EXECUTION

Conditional statements allow executing statements selectively based on certain decisions.

Such type of decision control statement are known as selection control statements or conditional branching statements.

Python language supports different types of conditional branching statements.

They are

- Conditional Execution (if)
- Alternative Execution (if-else)
- Chained Conditional Execution (if-elif-else)
- Nested conditionals.

## 1. Conditional Execution (if)

→ The if statement is the simplest form of decision control statement that is frequently used in decision making.

→ The if selection structure performs an indicated action only when the condition is true.; Otherwise the action is skipped.

Syntax:

```
if condition:
    statement(s)
```

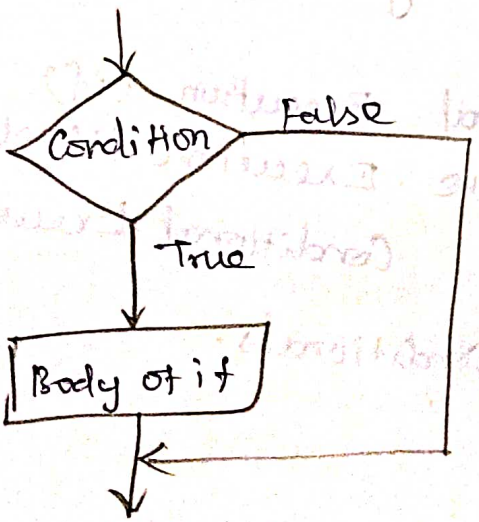
→ The colon (:) → end of the condition

→ if → body of this statement is indicated by the indentation.

Example

```
age = 16
if age <= 21
    print "You are a Minor."
```

Flowchart:




Operation of if statement.

## 2. Alternative Execution (if-else)

The if-else statement evaluates the condition and executes the true statement block only when the condition is True.

If the condition is False, false statement block is executed. Indentation is used to seperate the blocks.

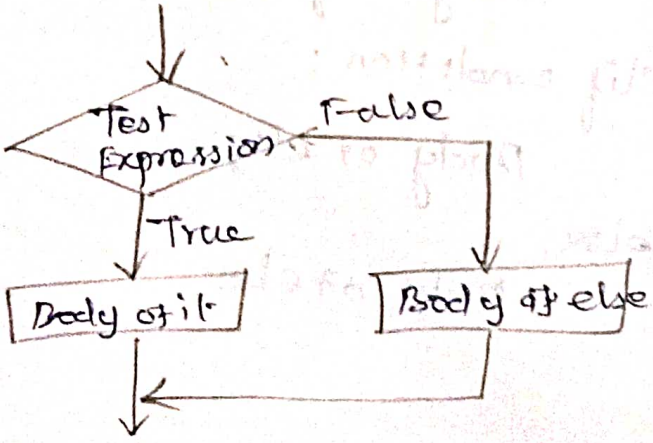
In Python program, use the if..else statement together with the conditional operators and logical operators.

Syntax:

```
If condition
    Body of if
else:
    Body of else
```

The if else statement evaluates the condition and will execute body of if only when the condition is True. If the condition is false, body of else is executed. Indentation is used to seperate the blocks.

Flowchart:




Operation of if-else statement

Program - Program 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. Chained Conditional Execution (if - elif - else)

The elif statement allows checking multiple expressions for truth value and executing a block of code as soon as one of the conditions evaluates to true.

Syntax

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

Example :

```
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"
```

Flowchart :

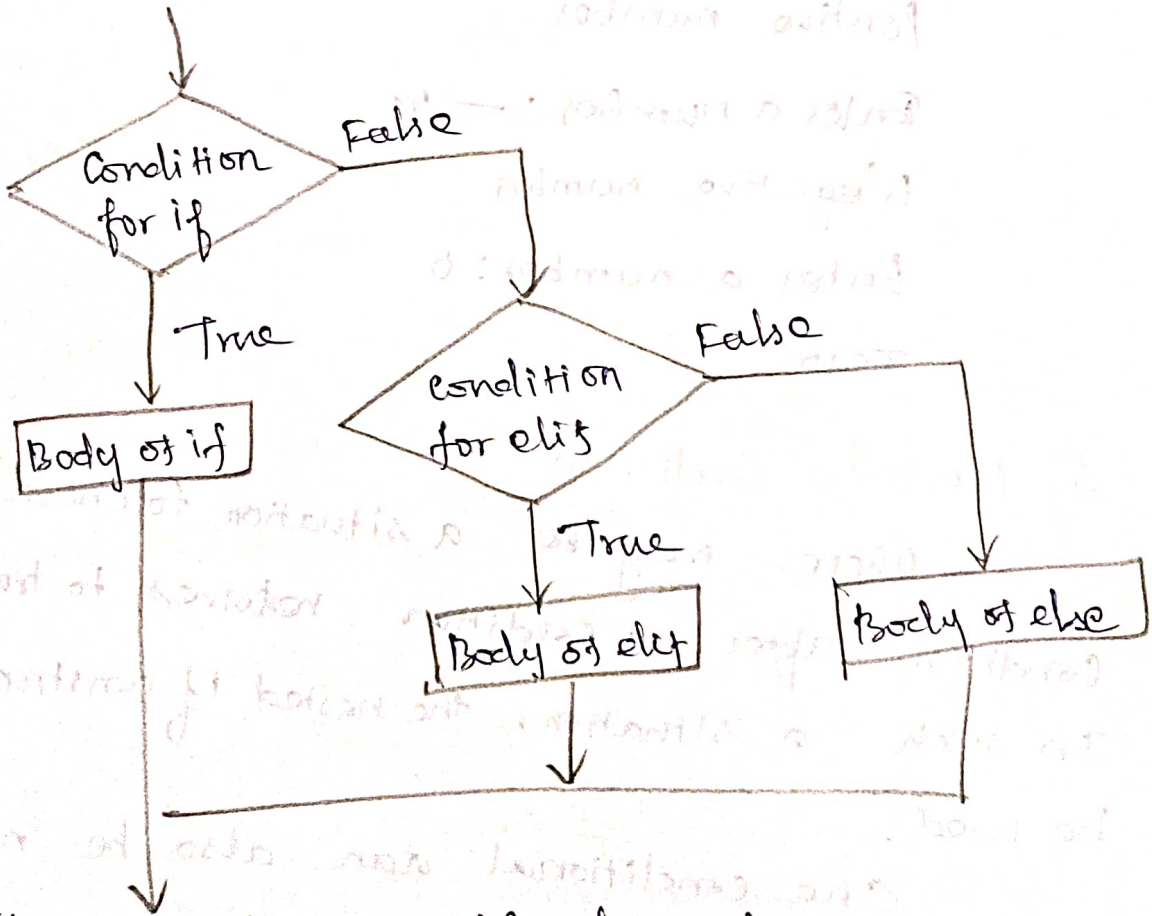



Fig : Operation of if..elif...else statement

Program to check if the number is positive or negative or zero.

```
num = int(input("Enter a number: "))
if num>0:
    print("Positive number")
elif num==0:
    print("zero")
else:
    print("Negative number")
```

Output:

```
Enter a Number : 25
Positive number
Enter a number : -45
Negative number
Enter a number: 0
zero.
```

4. Nested Conditions:

There may be a situation to check for another condition after a 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.

5. Iteration

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

Looping is also called as repetition structure or iteration. An iteration structure allows the programmer to perform an action which is to be repeated until the given condition is true.

## 5.1 State

An iteration involves states variables which keep track of the state of each iteration variable for each pass through the iteration.

An Iteration table is a way to visualize the states of the iterations.

Given a list L1 = [1, 2, 3, 4] the iteration table for reverse [L1] would be

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

Syntax

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

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')
```

Output

Enter x: 56
Enter y: 98
x is less than y

Enter x: 23
Enter y: 23
x and y are equal

Enter x: 94
Enter y: 47
x is greater than y.

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:

The for loop is executed for each of the items in the range. These values can be numeric, string, list or tuple.

Syntax

```
for val in sequence:
    body of for
```

Flowchart:

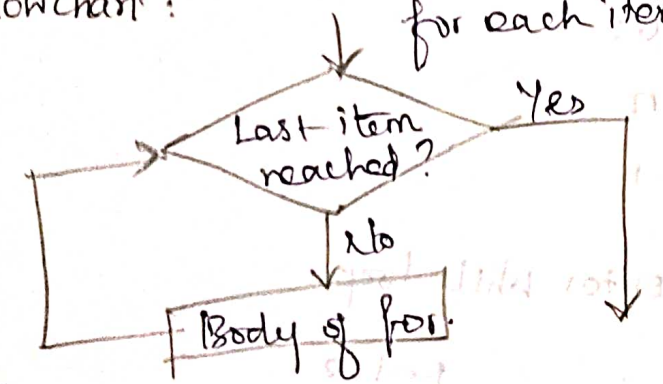



Fig: Operation of for loop.

Program to find the sum of all numbers stored in a list

(listsum.py)

```
# 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
```

## 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 items to iterate is not known before.

Syntax

```
while Condition
    Body of while
```

Example

```
n=5
while n!=0:
    print n
    n=n-1
```

Flowchart

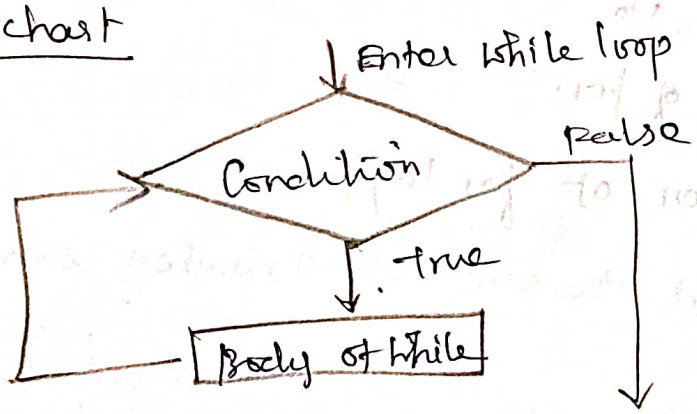



Fig: Operation of while loop.

Program: Add natural number upto n (sum.py)

```
#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)
```

Output

```
Enter n:10
The sum is 55
```

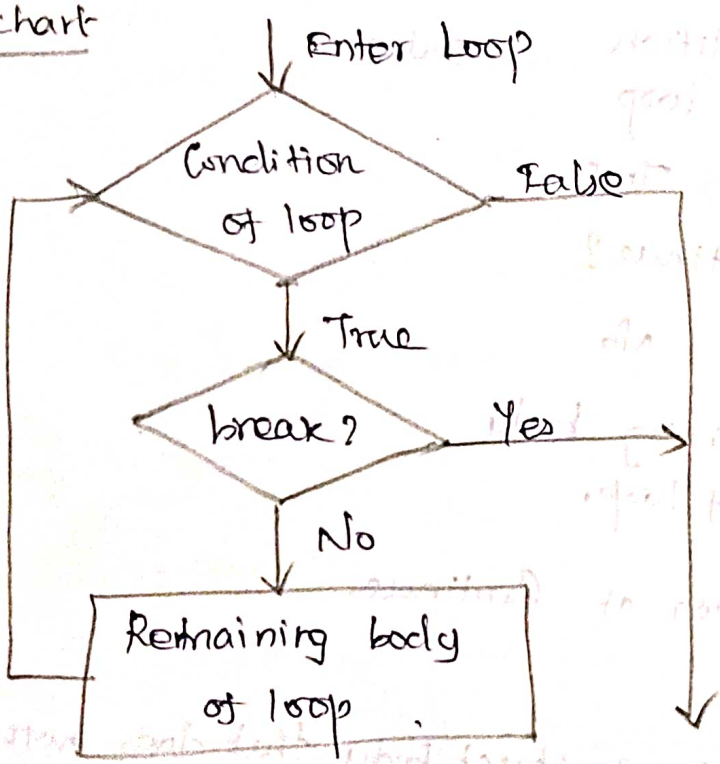
## 5.4. Break

The break statement terminates the execution of loop containing it and the control of the program goes to the statement immediately after the body of the loop

Syntax

```
break
```

Flowchart




```
num = 10
while num>0:
print "current value:"
num=num-1
if num == 5
    break
```

o/p
current value: 10
9
8
7
6

Fig: Operation of Break.

Program – illustration of break statement inside loop (break2.p

```
for val in "computer"
if val == "p"
    break
print(val)
print("The end")
```

output

c
o
m
p
u
The end.

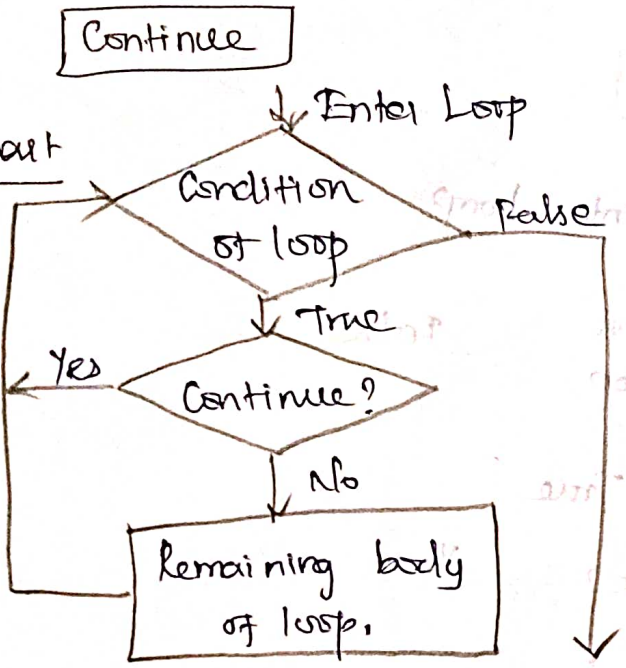
## 5. Continue

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

### Syntax

Continue

### Flowchart




```
for val in "computer":
    if val == "p":
        continue
    print(val)
print("The end")
```

```
o/p
Computer
The end.
```

Fig: Operation of Continue.

## 5.6 Pass

→ It is used to construct body that does nothing

→ In Python programming, pass is null statement. It does nothing when pass is executed. It results into 'no operation' (NOP).

→ 'Pass' statement can be used in 'if' clause as well as within loop construct. When you do not want any statements or commands within that block to be executed.

Syntax

```
pass
```

```
for val in "computer":
    if val == 'p':
        pass
    else
        print(val)
```

```
o/p
Computer
```

# 6. FUNCTIONS

In programming, the use of function is one of the ways to achieve modularity and reusability. Function can be defined as a named group of instruction that accomplish a specific task when it is invoked.

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

Advantages of Function:

Increases readability

Reduce code length

Increase reusability

Types of function:

1. Built-in-functions
2. User defined functions.

```
a= int(input("Enter a number:"))
b=a*a
print("the square of ",
      a,"is",b)
```

## Built-in-functions:

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 those commonly used functions every time.

Built-in-functions are the ready made functions in Python that are frequently used in programs but cannot modify them.

## Built-in-Functions

| Input or Output | Datatype conversions | Mathematical Functions |
|---|---|---|
| input() | bool() | abs() |
| print() | chr() | divmod() |
| | dict() | pow() |
| | int() | sum() |
| | float() | |
| | set() | |

User defined Functions:-

In addition to the standard library functions we can define our own functions while writing the program. Such functions are called user defined functions.

ELEMENTS OF USER DEFINED FUNCTIONS

In Python, the user defined functions contain two elements:

1. Function definition
2. Function call.

Function Definition:

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

Syntax:

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

Elements of function definition:

1. Function header
2. Function body.

## FRUITFUL FUNCTIONS

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

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

Syntax

```
return [expression]
```

2. Return None.

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

3. Return Values:

The return value may or may not be assigned to another variable in the caller.

```
# areacircle.py
def area(r):
    a = math.pi * r * r
    return a
import math
r = int(input("Enter a number"))
x = area(r)
print("area =", x)
```

Ex: r = num: 2
area: 12.5

4. Return fruitful return

In a fruitful function the return statements includes an expression.

Return immediately from this function and use the following expressions as a return value.

PARAMETERS AND ARGUMENTS:

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

Argument:

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

```
def sum(a, b)  # def fn def
    s = a + b
    print ("s =", s)

x = 10
y = 15
sum(x, y)      # fn call
```

Parameters

Arguments.

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

Output

Value of x is: 50
Value of x inside add is: 30
Value of x inside add is: 40
After function call Value of x in main is: 50.

## 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.

Program

```
x = 50                          → x is Global var
def add(x):
    x += 10                     → here x is the Local var
    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.

## FUNCTION COMPOSITION:

a = fn2(x)
b = fn1(a)

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 (var, exp, stmt) and compose them.

# RECURSION :

A function that calls itself is recursive ; the process of executing it is called recursion.

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

Recursive fn to find the factorial of a number

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

n=4
4 is 24

Advantages of recursion :

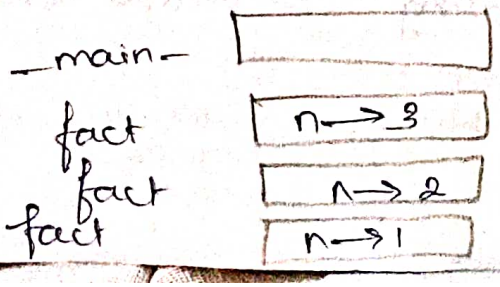
1. A complex task can be broken down into simpler sub problems using recursion.
2. Sequence generation is easier with recursion rather than using some nested iteration.

Stack diagrams for recursive function :

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

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

Example 1 Stack diagram for fact called with n=4




4!
4×3!
4×3×2!
4×3×2×1!

## Example 2 Greatest Common Divisor:

The greatest common divisor of two numbers (integers) is the largest integer that divides both the numbers. We can find GCD of two numbers recursively by using the Euclid's algorithm

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

## Infinite recursion:

If a recursion never reaches a base case, it goes on making recursive calls forever and the program never terminates

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

## STRINGS:

String is a sequence which is made up of one or more characters. Here the character can be a 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 line without using the escape character.

```
>>>str1 = 'Py is simple & easy lang.'
>>>str2 = "Pythi is free & open source s/w"
>>>str3 = '''Py is a high level language'''
>>>print(str1)
>>print(str2)
```

## Accessing string Elements:

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

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

The index of the first character (from left) in the string is 0 and the last character is n-1 where n is the length of the string.

Positive index helps in accessing the string from the beginning.

Negative index helps in accessing the string from the end.

From left to right, the first character of a string has the index 0.

From right and to left, the index of first character of a string is -1.

| index from left | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Character | C | o | m | p | u | t | e | r |
| index from right | -8 | -7 | -6 | -5 | -4 | -3 | -2 | -1 |

```
>>> fruit = 'banana'
>>> fruit[5]
'n'
>>> fruit[-5]
'a'
>>> fruit[7]
IndexError: string index out of range
```

## 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 loop

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

Example

```
>>> 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. If the condition is true, it enters the loop.

STRING SLICES

Accessing some part of a string or substring is known as slicing. This can be done by specifying an index range.

Subset of strings can be taken using the slice operator with two indices in square brackets separated by a colon ([] and [m:n])

The operator [m:n] returns the part of the string starting from index n (inclusive and ending at m (exclusive)

In other words, str [n:m] returns all the characters starting from str[n] till str[m-1].

```
>>> a : 'Python Programming'
>>a[0:5]    5:0
'Pytho'     'n Programming'
```

## STRINGS ARE IMMUTABLE

A String is an immutable datatype. It means that the contents of the string cannot be changed after it has been created.

Example

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

## STRING FUNCTIONS AND METHODS :

String provide methods to perform a variety of useful operations. A method is similar to a function. It takes arguments and returns a value.

1. Upper

The method upper takes a string and returns a new string with all upperclass letters.

Example

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

2. Lower

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

Example

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

## 3. Capitalize

The 'capitalize' function capitalize the first characters of s.

Example

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

## 4. Split

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

Example

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

## 5. Find

The 'find' method can find substrings in a string.

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

## 6. The in operator

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

Example

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

```
>>> def in_both(word1, word2):
    for letter in word1:
        if letter in word2:
            print(letter)
>>> in_both('java', 'javita')
```

## 7. String Concatenation

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

Example

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

## 8. Repetition

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

Example

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

## 9. String Length

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

Example

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

10. String Comparison

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

Example

```
>>> if 'python' == 'python'
        print ('Both strings are equal')
Both strings are equal.
```

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 (eg - .isalpha).

Example

```
>>> import string
>>> string.digits
'0123456789'
>>> string.ascii_letters
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLM
NOPQRSTUVWXYZ'
>>> string.ascii_lowercase
'abcdefghijklmnopqrstuvwxyz'
```

List as arrays - Creating fixed size list is similar to creating array in other programming language. The typical Python solution is to use the repetition operator and a list containing the value None.

Example

```
>>> a = [None] * 5
>>> a
[None, None, None, None, None]
```

## Illustrative Programs

1. Finding Square root a number

```
num = input("Enter a number :")
number = float(num)
squareroot = number ** 0.5
print("Square Root of %0.2f is %0.2f" %(num, 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 number : 50
Enter second number : 5
The GCD of numbers = 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 = 5.1847
```

4. Sum an array of numbers

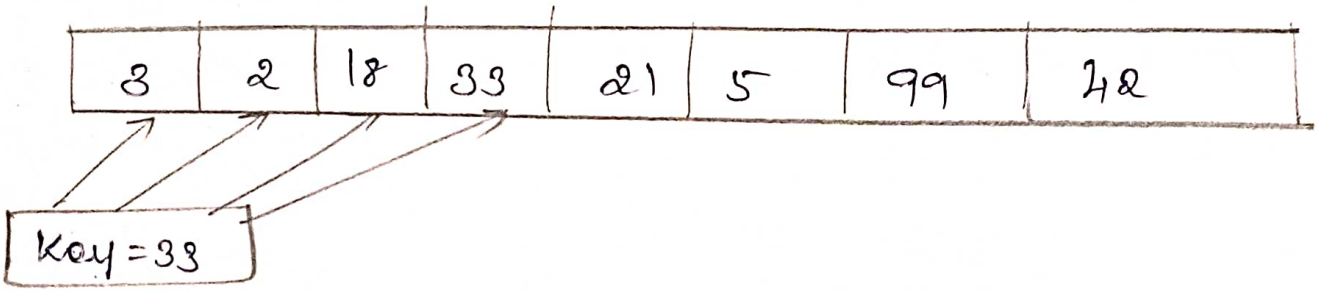
```
A = [0,0,0,0,0,0,0,0,0,0]
sum = 0
n = int(input("Enter a number:"))
print("Enter n numbers")
for i in range(0,n):
    A[i] = int(input())
for i in range(0,n):
    sum = sum + A[i]
print("sum =", sum)
```

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. This 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.




Algorithm:

Linear search is implemented using following steps.

Step 1: Read the search element from the user.

Step 2: Compare, the search element with the first element in the list.

Step 3: If both are matching, then display "Key found!!" and terminate the function.

Step 4: If both are not matching, then compare search element with the next element in the list.

Step 5. Repeat steps 3 and 4 until the search element is compared with the last element in the list.

Step 6: If the last element in the list is also doesn't match, then display "Key not found!!!" and terminates the function.

# UNIT-4

## List, Tuples, Dictionaries

### 1. Lists:

A list is a sequence of values. They can be of any type. The values in a list are called elements.

**Creating a list:**

There are several ways to create a new list.

1. The simplest way is enclose the elements in square bracket ([ and ])

**Example:**

A list of four integers : [10, 20, 30, 40]

A list of three strings : ['Jovita', 'Jesvita', 'varsha']

A list of different type : ['March', '2017', '2.6']

2. A list that contains no elements is called an empty list. It can be created with empty brackets [].

**Example**

```
empty = []
```

3. A list that is an element of another list is called nested list.

**Example**

```
['Dell', 2.0, [50, 100])
```

**Assign list values to variables:**

The list values can be assigned to variables

**Example:**

```
>>> icecreams = ['vanilla', 'strawberry', 'mango']
>>> numbers = [5, 10, 6]
>>> name = ['Denisha']
>>> print (icecreams, numbers, name)
['vanilla', 'strawberry', 'mango'] [5, 10, 6] ['Denisha']
```

## 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.

Example:

```
>>> icecreams =['vanilla','strawberry','mango']
>>> icecreams[0]
'vanilla'
>>> icecreams[1]
'strawberry'
>>> icecreams[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=['parrot','Dove','duck','cuckoo']
>>> print(birds[-1])
Cuckoo
>>> print(birds[-2])
duck
>>> print(birds[-3])
Dove.
```

## 2. LIST OPERATORS

### 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.

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

### 2.2 Repeat Operation:

Lists can be replicated or repeated concatenated with the asterisk operator "*". The * operator repeats a list in given number of times.

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

## 3. LIST SLICES:

Subset of lists can be taken using the slice operator with two indices in square brackets seperated by a colon[] and[m:n]). A range of items in a list can be accessed using the slicing operator.

Example

```
>>> birds=['parrot', 'Dove', 'duck', 'cuckoo']
>> print (birds[2:4])
['duck', 'cuckoo']
```

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

```
>>> print (birds[:3])
['parrot', 'dove', 'duck']
```

If the second index is omitted, the slice goes to the end

```
>>> print (birds[2:])
['duck', 'cuckoo']
```

## 4. LIST METHODS:

Python provides methods that operate on lists are

### 1. append()

Add an element to the end of the list.

Example

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

### 2. Extend

The extend methods takes a list as an argument and appends all the elements. This method does not return any value but adds the content to the existing list.

Example

```
>>> t1 = ['a', 'b', 'c']
>>> t2 = ['d', 'e']
>>> t1.extend(t2)
>>> t1
['a', 'b', 'c', 'd', 'e']
>>> t2
['d', 'e']
```

This example leaves t2 unmodified.

3. sort

The sort method arranges the elements of the list in ascending order.

Example

```
>>> t=['d','c','e','b','a']
>>> t.sort()
>>> t ['a','b','c','d','e']
```

4. insert

The insert() method 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)

Example

```
>>>Name=['Suthan','Jonita','Varsha','Denisha']
>>> Name.insert(2,'Jesrita')
>>> print(Name)
['Suthan','Jonita','Jesrita','Varsha','Denisha']
```

5. count(x)

The count method returns the number of times x appears in the list.

Example

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

6. len

The len() function returns the number of elements in a list

Example

```
>>>Num=[23,54,34,44,35,66,27,88,69,54]
>>> print len(Num)
10
```

7. reverse

The reverse() method reverses objects of list in place. This method does not return any value but reverses the given object from the list.

Syntax

```
list.reverse()
```

Example

```
>>> Name=['varsha', 'jovita', 'ebisha', 'donald', 'suthan']
>>> Name.reverse()
>>> print(Name)
['suthan', 'donald', 'ebisha', 'jovita', 'varsha']
```

8. max()

The max() function returns the maximum value from a list

Syntax

```
max(list)
```

Example

```
>>> Mark=[76,87,68,85,77]
>>> print "Maximum mark:", max(Mark)
Maximum mark: 87
```

9. min

The min() function returns the minimum value from a list

Syntax

```
min(list)
```

Example

```
>>> Mark=[76,87,68,85,77]
>>> print' Minimum mark:', min(Mark)
Minimum mark: 68
```

## 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 seperate counter is used as an index into the list which points each counter position 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 elements in the list.

Example

```
>>> for icecreams in icecreams:
        print(icecreams)
```

Output

```
Vanilla
Strawberry
mango.
```

## 6. MUTABILITY:

The list are mutable (changeable). When the bracket operator appears on the left side of an assignment, it identifies the element of the list that will be assigned.

Example

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

Illustration of list mutability. (changeable)

```
birds = ['parrot', 'pigeon', 'dove', 'owl', 'penguin']
print ("Birds =", birds)
birds[1] = 'duck'
birds [3] = 'hen'
print ("After changing pigeon to duck and owl to hen")
print ("birds=", birds)
```

Output

```
Birds = ['parrot', 'pigeon', 'dove', 'owl', 'penguin']
After changing pigeon to duck and owl to hen
Birds = ['parrot', 'duck', 'dove', 'hen', 'penguin']
```

7. List Membership:

in and not are the membership operators in Python.

in Operator:

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

example

```
>>> icecreams = ['vanilla', 'strawberry', 'mango']
>>> 'strawberry' in icecreams
True
>>> 'chocolate' in icecreams
False.
```

strawberry not in icecreams
False
chocolate not in icecreams
True.

② not in operator.

## not in Operator

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

Example

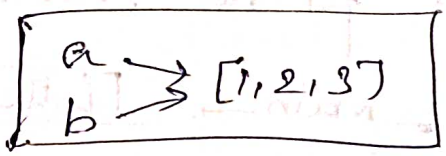
```
>>> icecreams=['vanilla', 'strawberry', 'mango']
>>> 'strawberry' not in icecreams
False
>>> 'chocolate' not in icecreams
True.
```

## 8. ALISING.

Alising is a circumstance ~~of~~ where two or more variables refer to the same object.

Example

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




## 9. Cloning lists:

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

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

Example

```
>>>x = ['a', 'b', 'c']
>>> y = x[:]
>>>print x
['a', 'b', 'c']
>>> print y
['a', 'b', 'c']
```

```
>>> y[0] = 'r'
['r', 'b', 'c']
```

10. List parameters:

Passing a list as an argument actually passes a reference to the list, not a copy of the list.

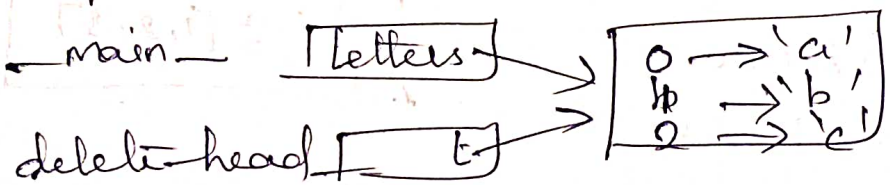
Example

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

Here, delete_head function removes the first element from a list

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

the parameter 't' and the variable letters are aliases for the same object.




11. MAP, FILTER, AND REDUCE

1. Map

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

Example

The capitalize_all function takes a list of strings and returns a new list that contains capitalized strings.

12. Deleting Elements:

There are several ways to delete elements from a list.

1. pop → To remove the index of the element deleted, the pop function is used.

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

2. del → If the index of element to be deleted is not provided, it deletes and returns the last element.

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

3. remove → To know the element to be removed, the remove Method is used.

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

The return value from remove is None.

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

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

## 1.3 Matrix using list

A list inside another list is called nested list or matrix. In Python a matrix is often represented with the list of lists.

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

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.
```

## DICTIONARY:

The datatype 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 seperated from its value by a colon(:) Consecutive items are seperated by commas. The entire items in a dictionary are enclosed in curly bracket ({})

Syntax

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

## Creating a dictionary:

**Method 1 - Dictionary using dict() function**

The dict() function is used to create a new dictionary with no items.

**Example**

```
>>> mech = dict()
>>> mech
{}
```

**Method 2 - Creating empty dictionary using {}**

Dictionary can be created using an empty string use square bracket ([])

**Example**

```
>>> weekdays = {}
>>> weekdays[0] = 'Sunday'
>>> weekdays[1] = 'Monday'
>>> print weekdays {0:'Sunday', 1:'Monday'}
```

**Method 3 - Dictionary using literal notation**

Dictionary can be created using literal notation with key value pairs.

**Syntax**

```
dictionary_name = {key : value, key , value, ... key N : value N}
```

## Operators and Methods in dictionary:

dict() → Creates a new, empty dictionary

dict(s) → Creates a new dict with key values

len(d) → Length of dictionary d (no. of key/value pairs)

d[key] = value → add a new key or replace the value of an existing key.

del d[key] → Remove key & associated value from dict d

key in d → True if key exists in dictionary d. else return False

## Advanced list processing : List Comprehensions :

List Comprehension is an elegant and concise way to create new list from an existing list in Python. List Comprehension can be characterised by a process. This process is described by a series of keywords

[expr for var in list if expr]

## Dictionary as a collection of Counters :

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

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 histogram function creates an empty dictionary.

## Looping and Dictionaries :

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

## 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

1. 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.
2. There is no match between 'v', and 'k' and the value 'v' is not 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()
```

This function is an example of the search pattern.

Example of a successful reverse lookup

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

# Illustrative Programs:

## 1. Selection Sort

The selection sort select the smallest element in the list. When the element is found, it is swapped with the first element in the list.

Then the second smallest element in the list is then searched. When the element is found, it is swapped with the second element in the list.

This process of selection and exchange continues, until all the elements in the list have been sorted in ascending order. 50, 40, 30, 80, 60, 70

Example: 56, 91, 35, 72, 48, 68

| | | | | | | |
|---|---|---|---|---|---|---|
| Unsorted 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] |
| After pass 6 | 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]
n = int(input("Enter list size:"))
print('Enter numbers')
for i in range(0, n):
    X[i] = int(input())
Selection Sort(X)
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 techniques is the simplest sorting technique. Each successive element in the list to be sorted and inserted into its pooper place with respect to the other already sorted element.

Insertion sort performs of n-1 passes.

For pass P=1 through n-1, insertion sort ensures that the element in position 0 through P are in sorted order.

| | [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 | 32 | 34 | 51 | 64 | 21 | 3 |
| After P=5 | 8 | 21 | 32 | 34 | 51 | 64 | 4 |

Program

```
def insertionSort(A):
    for i in range(1, len(A)):
        currentvalue = A[i]
        position = i
        while position>0 and A[position-1]>currentvalue
            A[position] = A[position-1]
            position = position-1
        A[position] = currentvalue
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())
insertionSort(x)
print x
```

Output

```
Enter list size:6
Enter numbers
4
8
5
7
2
1
[0,0,0,0,2,4,5,7,8,9]
```

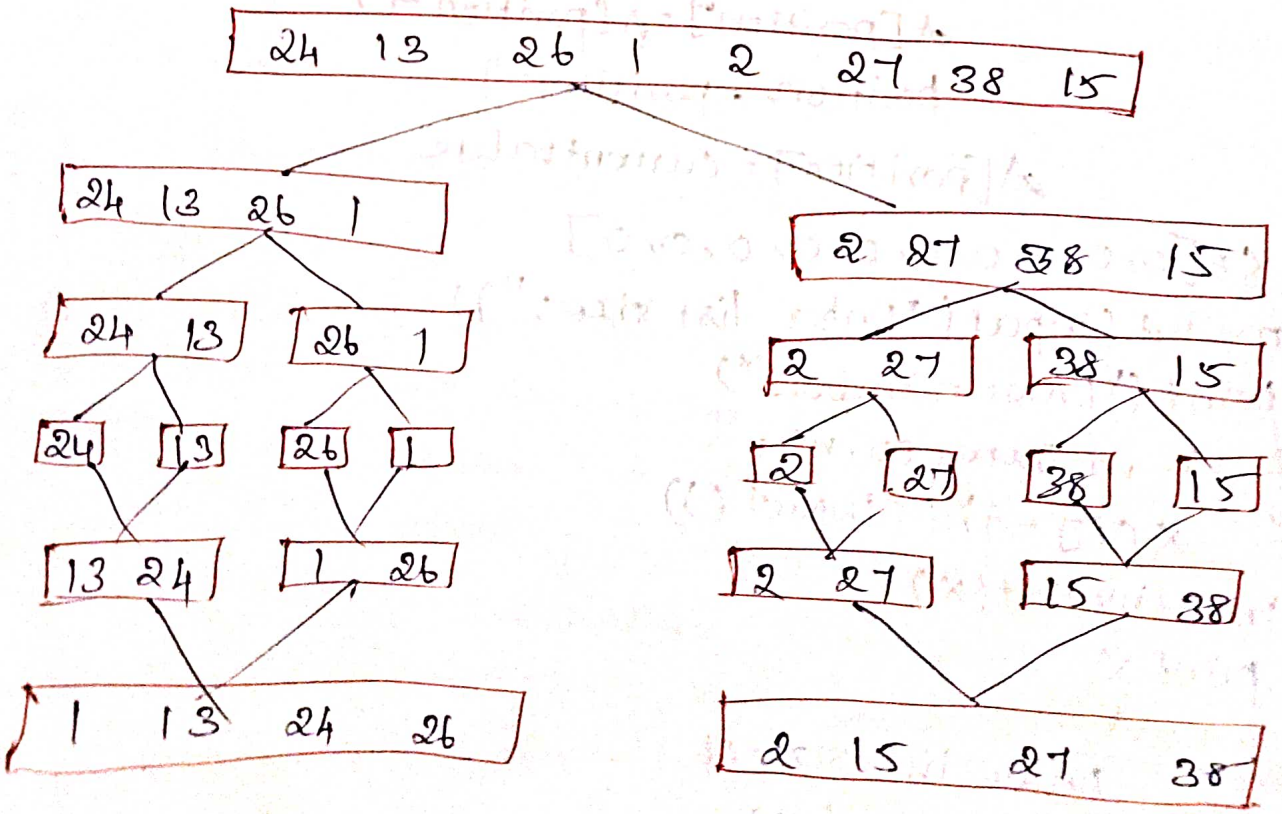
## 4. Merge sort

Merge sort uses devide and conquer method. The problem is divided into smaller problem and solved recursively. Finally it merges two sorted lists.

### Basic algorithm:

1. Divide the given array of elements into two half.
2. Recursively sort the first half and second half of array elements.
3. Take two input arrays A and B, an output array C
4. The first element of array C, the corresponding pointer in incremented
5. This step is repeated until all the elements are Compared.

Example. 24 13 26 1 2 27 38 15



Program

```
def mergeSort(A):
    if len(A)>1:
        mid = len(A)//2
        lefthalf = A[:mid]
        righthalf = A[mid:]
        mergeSort(lefthalf)
        mergeSort(righthalf)
        i=0
        j=0
        K=0
        while i<len(lefthalf) and j<len(righthalf):
            if lefthalf[i] <righthalf[j]:
                A[K]= lefthalf[i]
                i=i+1
            else:
                A[K]=righthalf[j]
                j=j+1
                K=K+1
        while i<len(lefthalf):
            A[K]= lefthalf[i]
            i=i+1
            K=K+1
        while j<len(righthalf):
            A[K]=righthalf[j]
            j=j+1
            K=K+1
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())
mergeSort(x) print(X)
```

Output

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

4. Histogram:

```
def histogram(items):
    for n in items:
        output = ''
        times = n
        while (times > 0):
            output += '*'
            times = times - 1
        print(output)
histogram([5,3,8,4,6,10])
```

Output

```
*****
***
********
****
******
*******
```

5 Student Mark sheet:

```
mark = [ ]
tot = 0
students = []
grade = [ ]
num1 = int(input("Enter number of students:"))
for num in range(num1):
    x = input("Enter name of students:")
    students.append(x)
    print("Enter marks obtained in 5 subjects:")
    for i in range(5):
        mark.append(input())
    for i in range(5):
        tot = tot + int(mark[i])
    avg = tot/5
    print("Name =", x)
    if avg >= 91 and avg <= 100:
        print("Grade = A1")
elif avg >= 81 and avg < 91:
    print("Grade = A2")
elif avg >= 71 and avg < 81:
    print("Grade = B1")
elif avg >= 61 and avg < 71:
    print("Grade = B2")
elif avg >= 51 and avg < 61:
    print("Grade = C1")
elif avg >= 41 and avg < 51:
    print("Grade = C2")
elif avg > 33 and avg < 41:
    print("Grade = E1")
else:
    print("Grade = E2")
```

Output

Enter number of students : 3
Enter name of students: Arun
Enter marks obtained in 5 subjects
98
97
96
91
95
Name = Arun
Grade : A1

6. Retail Bill preparation:

```
tax=0.18
Rate={"Pencil":10,"Pen":20,"Scale":7,"A4 paper":165}
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:"))
A4paper=int(input("A4paper:"))
cost = Pencil * Rate["Pencil"]+Pen * Rate["Pen"]+
    Scale * Rate["scale"]+A4paper * Rate["A4paper"]
Bill = cost+cost * tax
print("Please pay Rs. %f"%Bill)
```

Output:

Retail Bill Calculator
Enter the quality of ordered item:
Pencil : 5
Pen : 7
Scale : 1
A4 paper: 3
Please Pay Rs: 876.740

# UNIT-5

## FILES, MODULES, PACKAGES

### FILE

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 files and Binary files.

1. Text file:

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

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

2. Binary files:

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

### 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. This function returns a file object, also called a handle, as it as used to read or modify the file accordingly.

Syntax

file_object = open (file_name [, access_mode])

file_name → file name contains a string type value containing the name of the file which we want to access.

file access_mode → The value of access mode specifies the mode in which we want to open the file.

ie., read, write, append

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 object.. But we must not rely on it to close the file.

Syntax

fileobject.close()

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.

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)
```

(1) Writelines() Method

The writelines() method is used to write a list of strings.

Example

```
file=open("file1.txt","w")
lines=["Hello students,","Welcome to the world of Python",
       "Enjoy learning Python"]
file.writelines(lines)
file.close()
print("Writting to file...")
```

Output

```
Writing to file....
```

2. Append() Method:

The append() method is used to append files. In order to append a new line to the existing file, open the file in append mode, by using either 'a' or 'a+' as the access mode

(1) Append Only ('a')
(2) Append and Read ('a+')

Reading from a file:

1. Reading a file using read(size) method

To read the content of a file, it must be opened in read mode.

Syntax:

fileobject.read([size])

It starts reading from the beginning of the file until the size given.

Example

```
>>> f = open("test.txt", 'r', encoding = 'utf-8')
>>> f.read()
```

2. Reading a file using for loop

A file can be read line by line using a 'for loop'. This is both fast and efficient.

```
>>> for line in f
```

(3). Reading a file using readline() method

The readline() method is used to read individual lines of a file. This method reads a file till the newline, including the newline character.

```
>>> f = open("test.txt", 'r', encoding = 'utf-8')
>>> f.readline()
```

## 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.

### 1. The rename() Method

The rename() method takes two arguments, the current filename and the new filename.

Syntax

```
os.rename(current_file_name, new_file_name)
```

Example

To rename an existing file test1.txt to test2.txt

```
import os
os.rename("test1.txt", "test2.txt")
```

### 2. The remove() method

The remove() method is used to delete files. The method takes a filename as an argument and deletes that file

Syntax

```
os.remove(file_name)
```

Example

To delete an existing file test2.txt

```
import os
os.remove("test2.txt")
```

Attributes

## 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

### Example

```
>>> with open("strexample.txt", 'w', encoding='utf-8') as f:
        x=52
        f.write(str(x))
```

### 2

The syntax of format expression is.

[key] [flags] [width] [.precision] [length type] Conversation type

Where,

Key – Mapping key, consisting of a parenthesized sequence of characters.

Flags – Conversation flags which affect the result of some conversation type.

Width – Minimum field width. If specified as an '*', the actual width is read from the next element of the tuple

Precision – Precision, '.' followed by the precision. If specified as '*', the actual width is read from the next element of the tuple in values

and the value to convert comes after the precision.

Length type – Length Modifies

Conversation type – Conversation type

Formatting using .format function

Python supports format function which provides better peformance than the format operator %.

Syntax

<format expression> .format(v1, v2, v3 . . . . vn)

or

<format expression>. format values.

Formatting with alignment

The operators <, >, = ^ and are used for alignment when assigned a certain width to the numbers.

8. Command line argument.

Python provides a getopt module that helps to parse command-line options and arguments

$ python test.py arg1 arg 2 arg 3

9. Errors and Exceptions:

Python raises exceptions when it encounters errors. For example – divided by zero

1. Syntax error

```
>>> if a<3
SyntaxError: invalid syntax
```

Error caused by, not following the proper structure (syntax) of the language is called syntax error or passing error.

2. Logic error

Error caused due to wrong algorithm or logic to solve a particular program

Example

Divide by zero.

3. Exceptions

Even if a statement or expression is syntactically correct it may cause an error when an attempt is made to execute it.

Example

```
>>> 1/0
Traceback (most recent call last):
File "<pyshell#14>", line 1, in <module> 1/0
ZeroDivisionError: division by zero.
```

4. Python Built-in Exception

The local() built-in function will return a dictionary of built-in exception functions and attributes.

10. Handling Exception

When exceptions occur, 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.

Catching Exception in Python:

In Python, exceptions can be handled using a 'try' statement. It starts by executing the try clause.

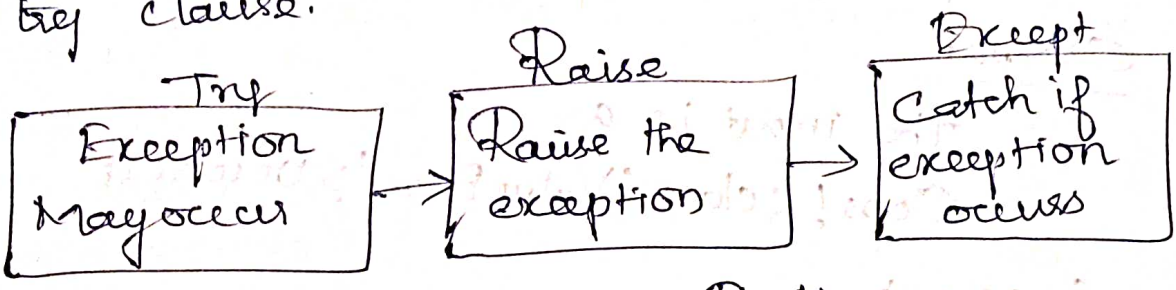



Fig: Exception Handling in Python

Syntax

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

→ First, the try block statements between the try and except are executed.

→ If no exception occurs, the except block is skipped

→ If an exception occurs, during execution of any statement in the try block then, the rest of the try block is skipped.

Example Finding reciprocal of x

```
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
```

In this program, the loop will be executed until the input is an integer that has a valid reciprocal.

Raising Exception

The exception can be deliberately raised using the raise keyword.

Syntax

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

Program

```
try:
    num=100
    print(num)
    raise ValueError
except:
    print("Exception Occured. Program Terminating.")
```

Output

```
100
Exception Occured. Program Terminating
```

## The finally Block

The try block has an optional block called finally. It is also a part of exception handling.

Program

```
try:
    k=5/0
    print(k)
except ZeroDivisionError:
    print("Can't divide by zero")
finally:
    print('This is always executed')
```

Output

```
Can't divide by zero
This is always executed
```

## 11. Modules

Modules are pre-written piece of code that are used to perform common tasks like generating random numbers, performing mathematical operation.

A module is a file with .py extension that has definitions of all functions and variables that would use in other programs

## 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.

### Example

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

## The form. import modules

A module may contain definition for many variables and functions.

### Example

```
>>> import example
>>> example.add(8,9.5)
17.5
```

## 12. Datetime Module

Python datetime module handle the extraction and formatting of date and time variables. Python display the date in yyyy-mm-dd format.

Syntax

```
import datetime.date = datetime.date.today()
```

Example

```
import datetime
tdate = datetime.date.today()
print "Today is:", tdate
```

13. Math Module:

To access one of the functions we have to specify the name of the module and the name of the function seperated by a dot (also known as period). This format is called dot (.) notation.

Python provides many useful mathematical functions in a special math library.

Method-1

The statement imports the entire math module methods into the current application.

Example

```
>>> import math
>>> print math.sqr(100)
10.0
```

Method 2. from math import sqrt

The statements import only one method called sqrt() into the current application.

Example

```
>>> from math import sqrt
>>> print sqrt(100) 10.0
```

Method 3

4. Packages:

A package is a way of collecting related modules together within a single tree like hierarchy.

A directory must contain a file named __init__.py in order for Python to consider it as a package.

1. Creating a package

1. Create a directory and name it with a package name
2. Keep sub directories (sub packages) and modules in it
3. 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.

Importing module from a package

The modules are imported from packages using the dot (.) operator. To import the start module

```
import Game.Level.start
```

15 Databases:

A database is a file that is organized for storing data. Many databases are organized like a dictionary in the sense that they map from keys to values.

Example

```
>>>import dbm
>>>db = dbm.open('captions', 'c')
```

16. Pickling:

Pickling refers to the process of converting an object in memory to a byte stream that can be stored on disk. The write methods of the file object in Python accepts only string datatypes. Pickling converts any kind of complex data to 0s and 1s (byte streams). This process can be referred to as pickling, serialization, flattening.

Pickling can be done with following datatypes.

- Booleans
- Integers
- Floats
- Complex numbers
- Strings (normal and unicode)
- Tuples
- Lists
- Sets, Dictionaries.

## Dump and load

Pickling and unpickling involves file IO. It uses the file writing/reading routines. The pickle.dump() is the method for saving the data out to the designated pickle file.

### Syntax

To write: pickle.dump(<datatype variable>, <filename>)
To read: pickle.load(<filename>)

### Illustrative Programs:

1. Counting number of words in string

```
string = input("Enter any string: ")
word_length = len(string.split())
print("Number of words =", word_length, "\n")
```

### Output

```
Enter any string: problem solving and
                  programming with python.
```

2. Copying file:

```
from shutil import copyfile
sourcefile = input("Enter source file name: ")
destinationfile = input("Enter destination file name: ")
copyfile(sourcefile, destinationfile)
print("File Copied Successfully.")
c = open(destinationfile, "r")
print(c.read())
c.close()
print()
print()
```

Output

```
Enter source filename : file1.txt
Enter destination filename: file2.txt
File copied successfully!
sunflower
Jasmine
Rose
```

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 16
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.
```