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 of sorted cards, guess an integer number in a range, Towers of Hanoi.

1.1. Fundamentals of Computing:

Computers are seen everywhere around us, in all spheres of life, in the field of education, research, travel and tourism, weather forecasting, social networks, e-commerce etc.

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

Data: Data is defined as an un-processed collection of raw facts, suitable for communication, interpretation or processing.

Information: Information is a collection of facts that is processed to give meaningful, ordered or structured information.

1.1.1. Components of a Computer:

The computer system hardware comprises of three main components:

1. Input / output (I/O) unit.
2. Central Processing Unit (CPU) &,
3. Memory Unit.

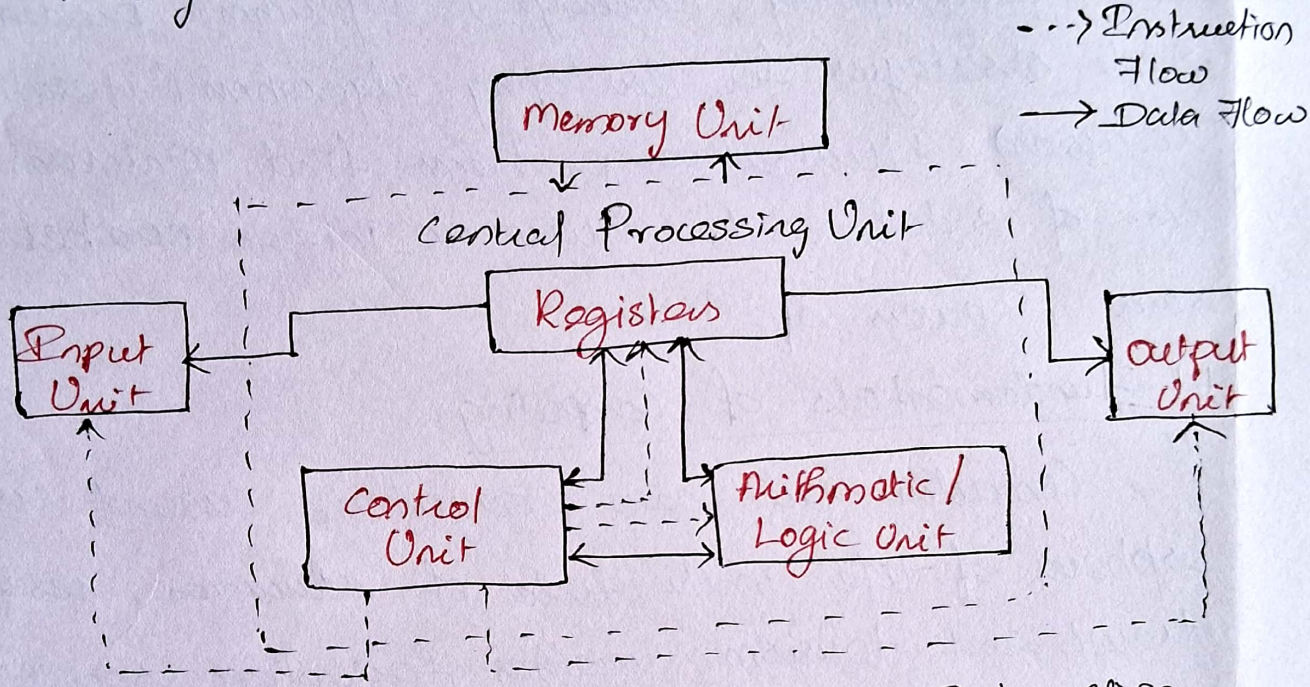


Fig) The computer system Interaction

(1) Input / output Unit:

⇒ The input / output unit consists of the Input unit and the output unit.

⇒ The user interacts with the computer via the I/O unit

The input unit ⇒ accepts data from users.
⇒ It converts the data into form that is, understandable by the computer.
⇒ The input is provided to the computer

The output unit ⇒ provides the processed data.
⇒ Provides output in a form.
⇒ Some of the output devices are monitor and printer

(2) Central Processing Unit (CPU)

⇒ CPU or the processor is often called Brain of Computer
⇒ CPU controls, coordinates & supervises the operation of the computer
⇒ CPU consists of
(i) Arithmetic Logic Unit (ALU)
(ii) Control Unit (CU) &
(iii) Set of Registers.

Arithmetic Logic Unit:

⇒ ALU consists of two units. (ie) arithmetic & logic unit
⇒ The Arithmetic unit performs arithmetic operations on the data that is made available to it.
⇒ The logic unit performs logic operations.
⇒ ALU performs arithmetic and logic operations & uses registers to hold the data that is being processed.

Registers:

⇒ Registers are high-speed storage areas within the CPU, but have the least storage capacity.
⇒ Registers are not referenced by their address, but are directly accessed and manipulated by the CPU.
⇒ Registers store data, instructions, addresses and intermediate results of processing.

Some of the important registers in CPU are as follows.
ACC - Accumulator stores the result of arithmetic & logic
IR - Instruction Register contains current instruction.
PC - Program Counter contains the address of next instructions to be processed.

MAR - Memory Address Register contains the address of next location the memory to be accessed.

MBR - Memory buffer Register. temporarily Stored data

DR - Data Register stores the operands.

(iii) Control Unit (CU)

⇒ The control unit organizes the processing of data and instructions.

⇒ The CU acts as a supervisor, controls and coordinates the activity of the other unit of computer.

⇒ CU coordinates the input and output devices of a computer.

⇒ It directs the computer to carry out Stored progra

③ Memory Unit:

The memory unit consists of cache memory, Primary memory and secondary memory.

⇒ Memory unit stores the data and instructions, intermediate results and output during processing of data.

(i) Cache memory: The data and instructions that are required during the processing of data are brought from the secondary storage devices and stores in RAM.

(ii) Primary memory: is the main memory of computer. It is used to store data and instructions during the processing.

(iii) Secondary Memory: stores data and instructions permanently. The information can be stored in secondary memory for a long time & generally permanent in nature unless erased by the user.

1.2. IDENTIFICATION OF COMPUTATIONAL PROBLEMS.

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

⇒ Computational problem identification consist of two steps:

(i) Identifying and acknowledge that there is problem.

(ii) Developing a problem identification statement.

⇒ Computational problem Identification, require a certain mode of approach or way of thinking. This approach is often called computational thinking and is similar to the scientific method with predictions

⇒ Understanding computational thinking will give foundation for solving problems.

Decomposition:

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

⇒ A computational thinking approach to a solution often start by breaking the problem down into smaller more familiar component so they can be managed easier.

Pattern Recognition:

⇒ The second step is pattern recognition whereby similarities and trends are identified within the problem

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

⇒ This is a key component for making efficient solutions.

Abstraction:

⇒ The abstraction stage involves the identification of key component of the solution.

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

Algorithm Design:

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

⇒ This process uses inductive thinking and is needed for transfering a particular problem to a larger class of similar problems. this step is also sometimes called algorithmic thinking.

1.3. NEED FOR LOGICAL ANALYSIS AND THINKING?

Computer can perform variety of tasks like receiving data, processing it and producing useful results. It cannot perform on its own. A computer needs to be instructed to perform a task.

1.4 PROBLEM SOLVING TECHNIQUES:

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

1. Algorithm. 2. Flowchart, 3. Pseudocode.

1.5. Algorithms:

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

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

⇒ The algorithm can be implemented in many different language by using different methods and programs.

1.5.1. Guidelines for writing Algorithms:

(i) An alogorithm should be clear, precise and well defined.

(ii) It should always begin with the word 'Start' and end with the word 'Stop'.

(iii) Each step should be written in a seperate line

(iv) Steps should be numbered as Step 1, Step 2 & so on

1.5.2. Properties of an algorithm:

(i) Fitness: An alogorithm must be terminated after a finite number of steps.

(ii) Definiteness: Each step of an algorithm must be precisely defined.

(iii) Input.

(iv) Output:

(v) Effectiveness.

1.6. BUILDING BLOCKS OF ALGORITHMS:

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

* Statements.
* States
* Control flow
* Function.

1.6.1. Statement / Instruction:

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

⇒ An instruction describes an action.

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

⇒ The time taken to execute all the statements of the algorithm should be finite and within reasonable limit.

1.6.2 State:

⇒ computational process 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 variable.

⇒ As a values of the variables are changed, the state changes.

⇒ State is a basic and important abstraction. An algorithm starts from the initial value state with some input values.

1.6.3. Control Flow:

⇒ An algorithm is a sequence of statement. However, after executing a statement, the next statement to be executed need not be the next statement in the algorithm.

⇒ The statement to be executed next may depend on the state of the process.

⇒ There are three important control flow statements to alter the control flow depending on the state. They are, (i) Sequence Control Flow.
(ii) Selection control flow.
(iii) Iteration control flow.

(i) Sequence control flow:

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

Example: Algorithm to find the sum of two numbers.

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

(ii) Selection Control Flow:

In selection control flow, a condition of the state is tested, and if the condition true, one statement is executed. If the condition is false, an alternative statement is executed.

Example: Algorithm to find greatest among three no's.

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 greatest, Print "A' is the greatest", else Print "C is the greatest".
Step 5: Compare B and C. If B is greatest Print "B is the greatest" else print "C is greatest".
Step 6: Stop.

(iii) Iteration control flow:

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 are executed again and again.

Example: Algorithm to find the sum of first 100 integers.

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

1.6.4 Functions:

In some cases, algorithm can become very complex. The variables of an algorithm and dependencies

among the variables may be too many. then it is difficult to build algorithms correctly.

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

⇒ Any complex problem will become simpler if the problem is broken smaller and the smaller problems are solved.

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

1.7. NOTATIONS:

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

1.7.1. Pseudocode:

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

⇒ Pseudocode does not include details like variable declarations, subroutines.

⇒ Using pseudocode, it is easier for a Programmer or a non-Programmer to understand the general working of the Program. because it is not based on any Programming language.

Preparing a Pseudocode:

1. Pseudocode is written using structured English.
2. Pseudocode does not include details like declarations, subroutines etc.
3. Using Pseudocode, it is easier for a programmer or a non-programmer.
4. Input: INPUT, GET, READ & PROMPT.
   Output: OUTPUT, PRINT, DISPLAY & SHOW.
   Processing: COMPUTE, CALCULATE, DETERMINE, ADD, SUBTRACT, MULTIPLY, DIVIDE.
   Initialize: SET, & INITIALIZE
   Incrementing: INCREMENT.
5. The keyword should be capitalized.
6. There are three control structures used in pseudocode, They are.

(a) Sequence control structure.

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

(b) Selection control Structure:

1. IF-THEN-ELSE statement.
2. CASE statement.

(c) Iteration control structure:

```
WHILE condition
    statements
END WHILE.
```

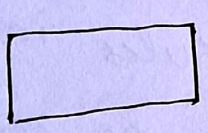
1.7.2. Flowchart:

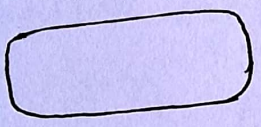
⇒ A flowchart is a diagrammatic representation of the logic for solving a task.

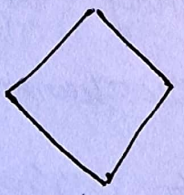
⇒ A flowchart is drawn using blocks of different shapes with lines connecting them to show the flow of control.

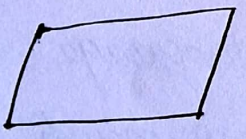
Flowchart symbols:

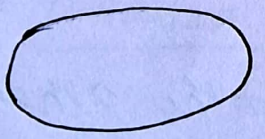
A flowchart is drawn using different kinds of symbols. Every symbol used in a flowchart is for a sepecific purpose.


- Process - Operation or action step.

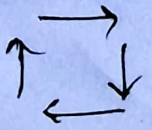

- Alternate Process - Alternate to normal proces


- Decision - Decesion or branch.

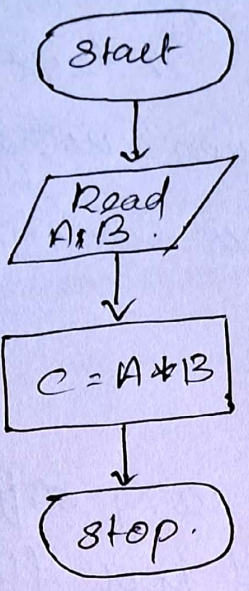

- Data - Input / output to or from a proces.


- Terminales - start or stop point


- Connector - Join flow lines.


- Flow lines - Indicate the direction of flows.

Example: Multiply two given numbers.



1.7.3. Programming Language:

A programming language consist of vocabulary containing a set of grammatical rules intended to convey instructions to a computer or computing device to perform specific tasks.

There are two types of programming languages. They are low-level and high-level programming language.

(i) Low level languages include assembly and machine languages. An assembly language contains a list of basic instructions and is much harder to read than a high-level languages.

(ii) High-level languages, on the other hand, are designed to be easy to read and understood, allowing programmers to write source codes naturally, using logical words and symbols.

1.8. Algorithmic Problem Solving:

Keys steps required for solving a problem using a computer are,

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.

(1) Obtain a description of the problem:

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

⇒ A problem description suffers from one or more of the following types of defects,

(i) the description may rely on unstated assumptions
(ii) the description may be ambiguous.
(iii) the description may be incomplete.
(iv) the description may have internal contradictions

(2) Analyze the problem:

⇒ It is important to clearly understand a problem before we begin to find the solution for it.

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

(i) What are the data available?

(ii) Where is that data from?

(iii) What are the rules exist for working with the data.

(iv) What are the relationships exist among the data values?

Step 3: Develop a high-level language:

⇒ 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.

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

Step 4: Refine the algorithm by adding more detail

A high-level algorithm shows the major steps that need to be followed to solve a problem.

Our goal is to develop algorithms that will lead to compiler program.

Step 5: Review the algorithm:

The final step is to review the algorithm First walk through the algorithm step by step to determine whether or not, it will solve the original problem.

1.9. SIMPLE STRATEGIES FOR DEVELOPING ALGORITHM:

There are various kinds of algorithm development techniques formulated and used for different types of problems.

1.9.1. Iteration:

In general, loops are classified as:

(i) Counter-controlled loops.

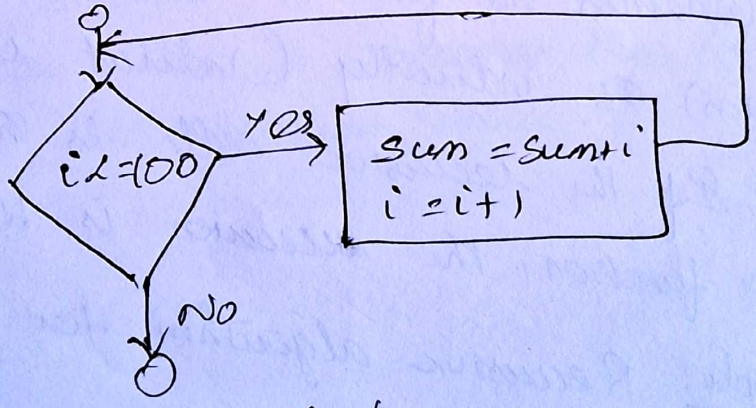
(ii) Sentinel controlled loops.

1.9.1.1. Counter controlled loops:

Counter controlled looping is a form of looping in which the number of iterations to be performed is known in advance.

Counter-controlled loops are so named because they use a control variable, known as loop counter.

Example: consider a progeem segment to fond the sum of first 100 integers.



1.9.1.2. Sentinel-controlled loops:

In Sentinel-controlled looping, the number of times the iteration is to be performed is not known beforehand.

Algorithm: To find sum of first N integers.

1. Start
2. Read N
3. Assign sum=0, i=0,
4. Calculate i=i+1 & 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.

1.9.2. 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.

Recursion is classified according to the following criteria:

1) Whether the function calls itself directly (direct recursion) or indirectly (indirect recursion).

2) If the recursive call is the last operation of a function, the recursion is known as tail recursion

Example: Recursive algorithm for finding factorial of a number.

Step 1: Start.
Step 2: Read number n.
Step 3: call factorial (N)

Step 4: Print factorial f.

Step 5: Stop.

Factorial (n)

Step 1: if n == 1 then return 1

Step 2: Else

f = n * factorial (n-1)

Step 3: Return f.

1.10. Finding Minimum number in a list:

Problem Statement:- Finding minimum number in a list is an example of selection problem.

Algorithm:

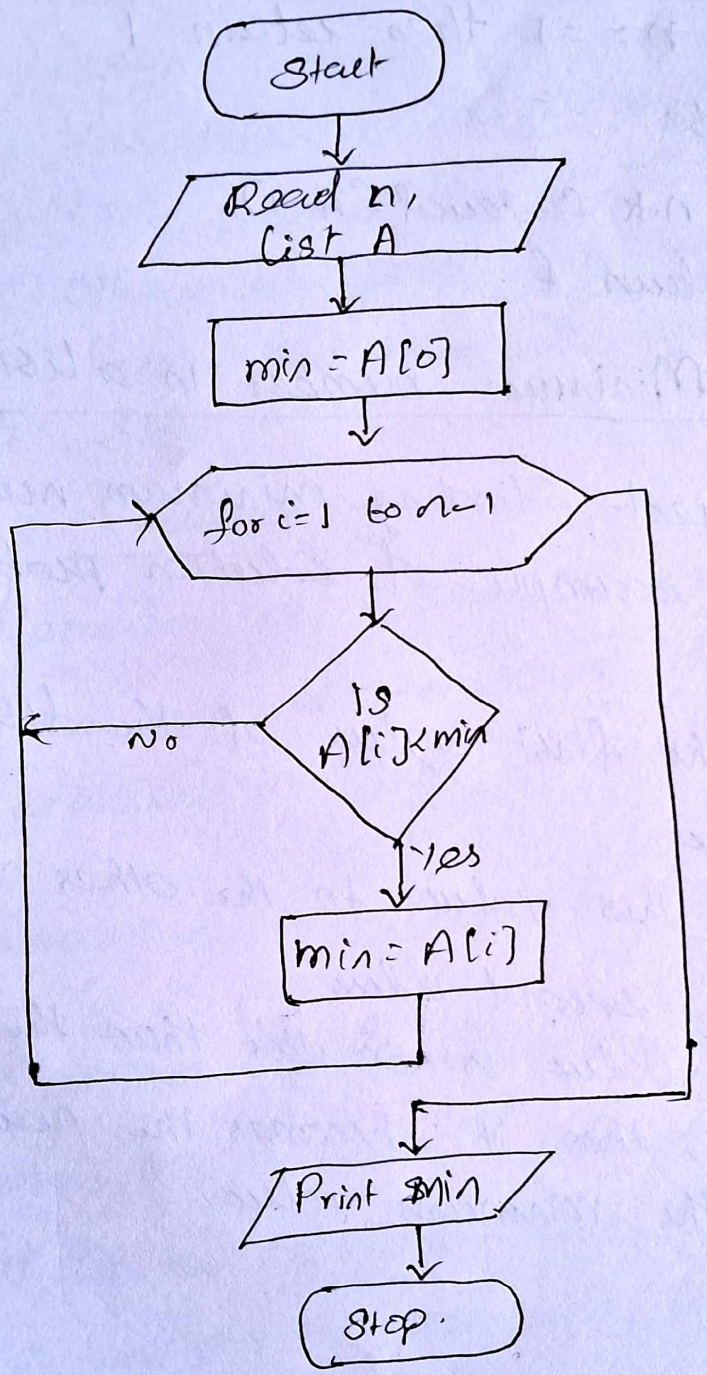
1. Assign the first value of the list as minimum value.
2. Compare this value to the other values starting from second value.
3. When a value is smaller than the present minimum value, then it becomes the new minimum
4. Print the minimum value.

Pseudocode:

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

END WHILE
PRINT Min.

Flow chart:



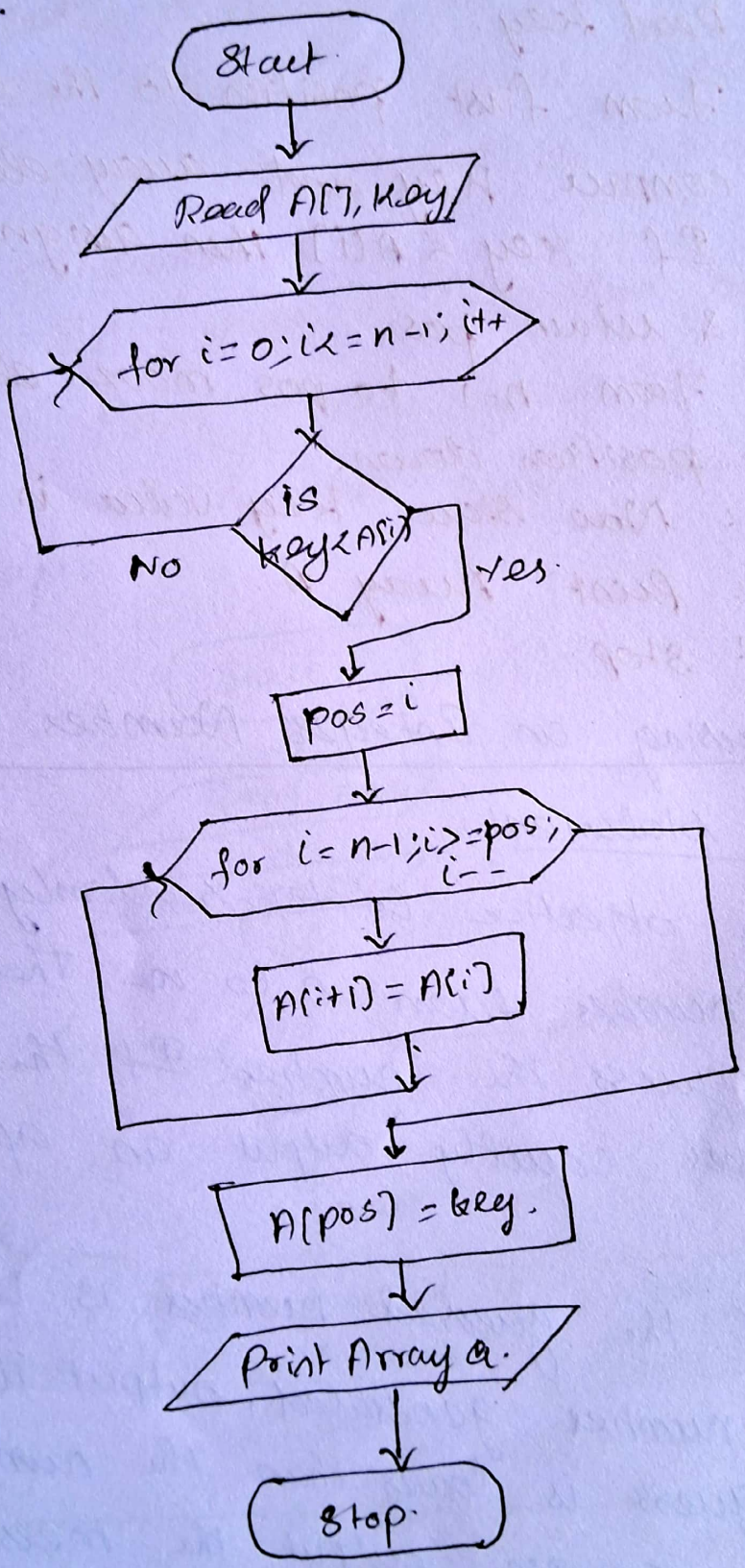
1.11. Inserting a card in a list of sorted cards.

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 cards you are holding still sorted.

Flowchart:



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 array compare key and array element.
Step 5: If key < A[i] then assign i to pos, & return pos.
Step 6: From n-1 to pos move element one position down.
Step 7: Now store key value in pos location.
Step 8: Print Array A.
Step 9: Stop.

1.12. Guessing an Integer Number in a Range:

Problem statement:

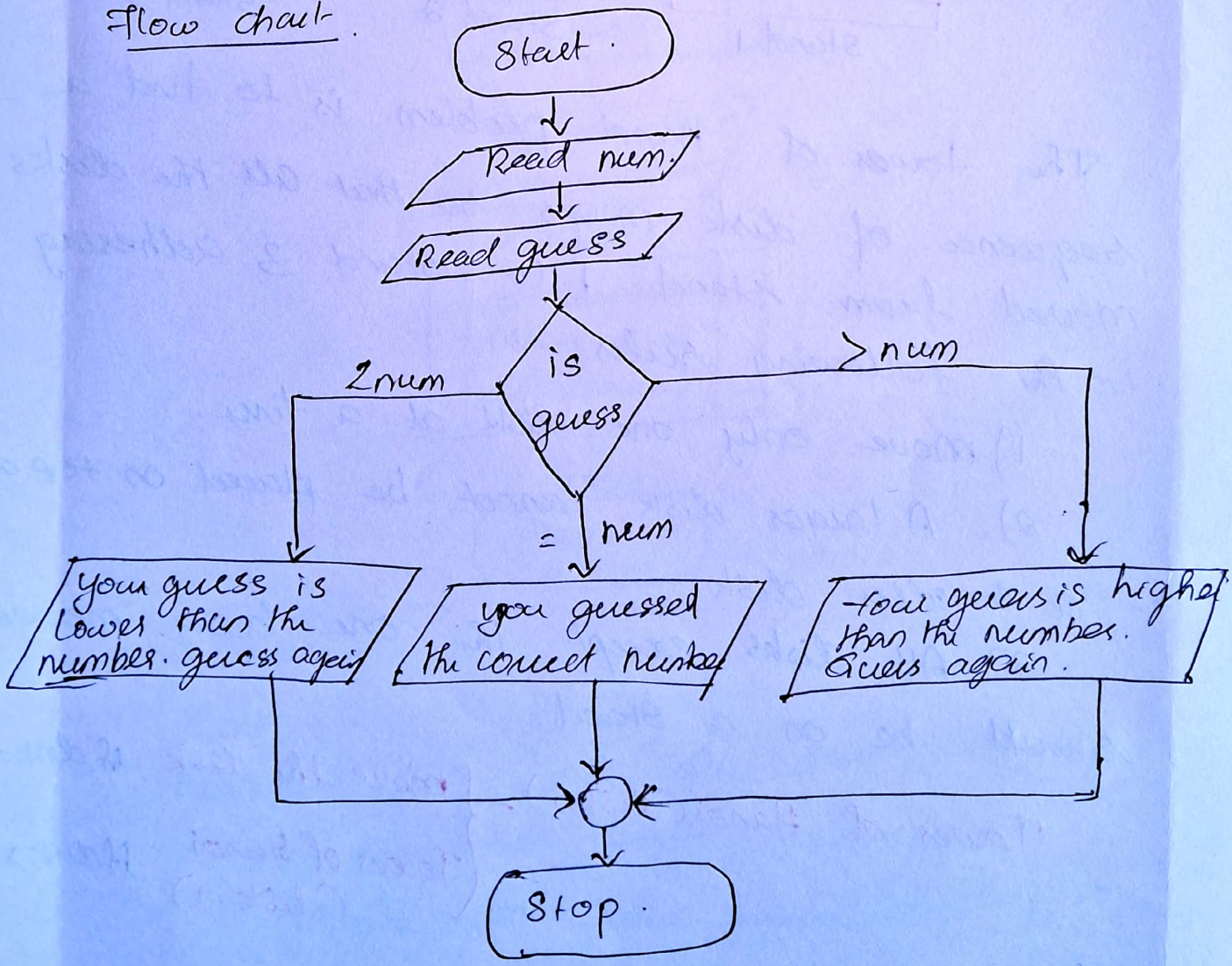
The objective is to randomly generate integer number from 0 to 9. Then the player has to guess the number. If the player guesses the number correctly output an appropriate message.

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 and read num.
   a. Enter a number to guess.
   b. If (guess is equal to num)
      Print "You guess the correct number"
      otherwise.
      If (guess is less than num)
      Print "Your guess is lower than the number" guess again".
3. Stop.

Flow chart.

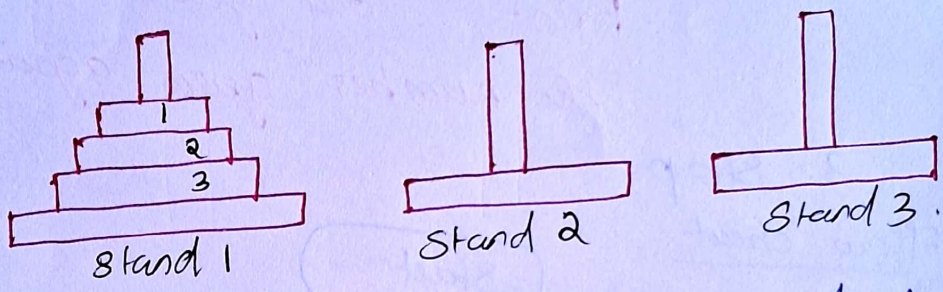


1.13. Tower of Hanoi:

Tower of Hanoi is one of the classical problem 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 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:

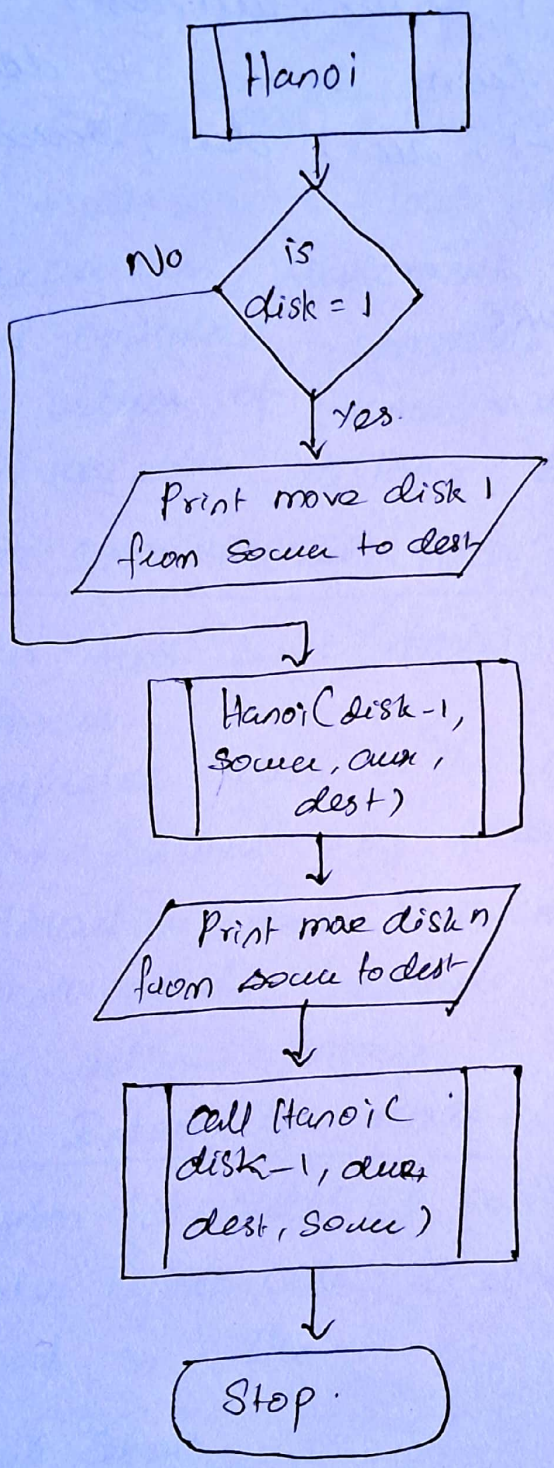
1.) move only one disk at a time.

2) A larger disk cannot be placed on top of a smaller disk.

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

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

Flow chart:



Algorithm:

Start

Procedure Hanoi (disk, source, dest, aux)

If disk == 1 THEN

move disk from source to dest.

ELSE

Hanoi (disk-1, source, aux, dest)

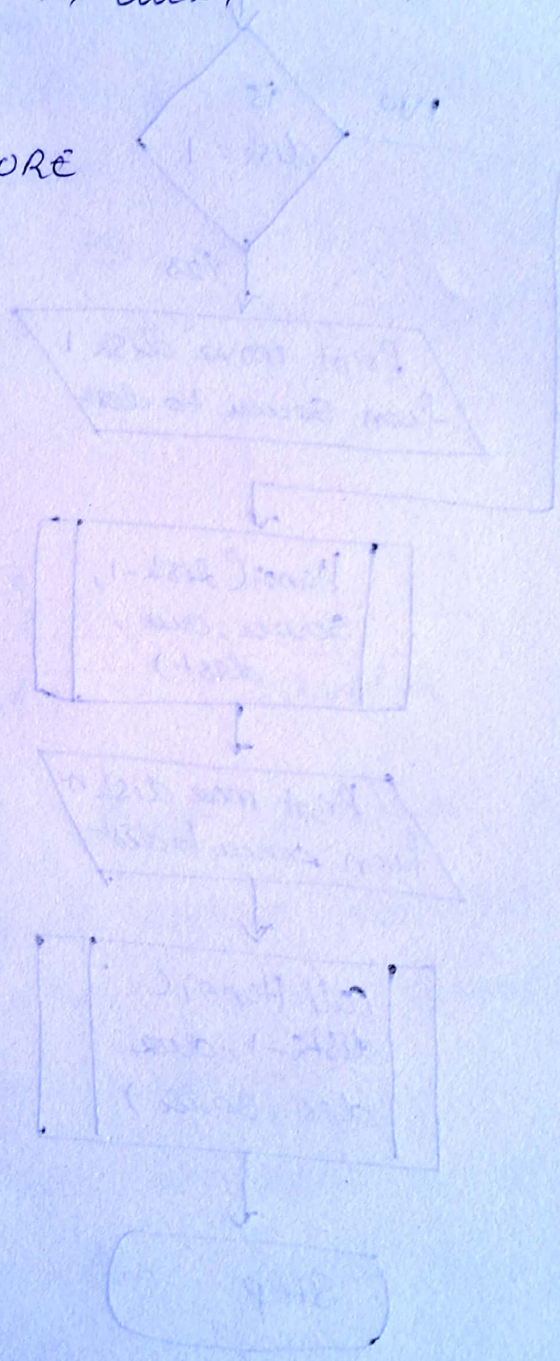
move disk from source to dest

Hanoi (disk-1, aux, dest, source)

END IF

END PROCEDURE

STOP



UNIT - 2.

DATA TYPES, EXPRESSIONS, STATEMENTS.

Python interpreter and interactive mode, debuggings; values and types; int, float, boolean, string and list; variables, expressions, statement, tuple assignment precedence of operators, comment; illustrative programs exchange the values of two variables, circulate the values of n variables, distance between two points.

2.1. PYTHON INTERPRETER AND INTERACTIVE MODE:

Python has two basic modes; interpreter and interactive modes.

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

⇒ The interactive mode is a command line shell which gives immediate feedback for each statement, fed in the active memory.

2.1.1. Python Interpreter mode:

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

Python 3.6.0 Shell.

⇒ Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 07:18:10) [MSC v.1900 32 bit (Intel)] on win32.

⇒ 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 begin with 3 if the version is Python 3.

To execute a script, Type the file name along with the path at the prompt. For example, if the name of the file is sum-py, we type.

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

2.1.2. Python Interactive mode!

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

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:

To write multiline code in Python interactive mode, hit Enter key for continuation lines, this prompts by default three dots (...)

Example:

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

2.2. DEBUGGING:

A programmer can make mistakes while writing a program, and hence, the program may not execute or may generate wrong output.

Errors occurring in programs can be categorized as,

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

2.2.1. Syntax errors:

Python has its own rules that determine its syntax. The interpreter interprets the statements only if it is syntactically correct. If the syntax error is present, the interpreter shows error messages and stop execution.

Example: Parenthesis must be in pairs, so the expression (20+22) is syntactically correct, whereas (7+35 is not due to absence of right parenthesis.

2.2.2. 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. Since the program interprets successfully even when logical errors are present in it, is sometimes difficult to identify these errors.

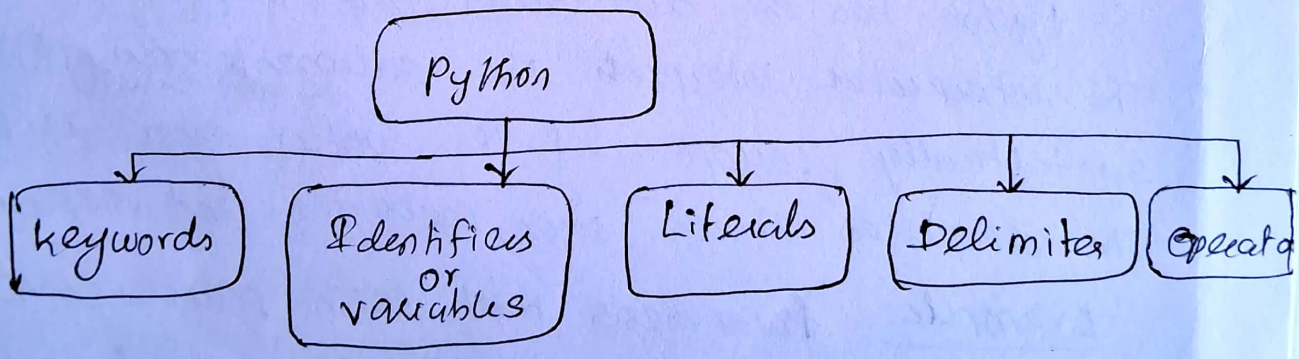
Example: To find the average of two numbers 10 and 12 and we write the code as 10+12/2 it would run successfully and produce the result 16. Surely 16 is not the average of 10 and 12.

2.2.3. Runtime errors:

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.

2.3. TOKENS:

Python breaks each logical line into a sequence of elementary lexical component known as tokens.



2.4. KEYWORDS

Keywords are reserved words they have predefined meanings in python. They cannot be used as ordinary identifiers. Python is case sensitive so keywords must be spelled exactly as they are ly written. Python 3 has 33 keywords.

eg: False, True, and, as, assert, break, class, continue, def, del, elif, if, else, for, is, return, while, not, or, from, finally

2.5. IDENTIFIERS:

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

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

⇒ Spaces are not allowed.

⇒ Python will not accept @, $, and %. as identifi

⇒ Python is a case-sensitive language.

2.6. ESCAPE SEQUENCES:

Escape sequences are non-printable characters. It consist of backslash followed by a characters.

\newline	- backslash & newline ignored.
\\	- Backslash.
\'	- Single quote.
\"	- double quote.
\a	- ASCII Bell (BEL)
\b	- ASCII Backspace (BS)
\f	- ASCII formfeed (FF)
\n	- ASCII Linefeed (LF)
\r	- ASCII Carriage Return
\t	- ASCII Horizontal Tab.
\v	- ASCII Vertical Tab.

2.7. LITERALS:

A literals is a sequence of one or more characters that stands for itself. There are two important types of literals. They are,

1. Numeric Literals.
2. String literals.

2.7.1. Numeric Literals:

- A numeric literal is a literal containing only the digit 0-9. an optinal sign character (+ or -) and posible decimal point.
- If a numeric literal does not contain a decimal point, then it denotes an integer (int) value.
- If a numeric literal contains a decimal point, then it denotes a floating-point (float) value

Arithmetic Overflow: Arithmetic overflow occurs when a calculated result is too large in magnitude to be represented,

```
>>> 1.5e200 * 2.0e210.
inf.
```

This result in the special value inf rather than the arithmetically correct result 3.0e410.

Arithmetic underflow:

Arithmetic underflow occurs when a calculated result is too small in magnitude to be represented

Example:

```
>>> 1.0e2300 / 1.0e1000
0.0.
```

2.7.2. String Literals:

String literals or strings represent a sequence of characters surrounded by quotes.

Example: 'Hello' 'Jovita, Jesuita',

"231, Gumel Nagar, 629004".

2.8. DELIMITERS:

Delimiters are symbols that perform these special roles in Python like grouping, punctuation and assignment binding of objects to names.

( ) [ ] { } - Grouping.

. , : ; @ Punctuation.

= += -= *= /= Arithmetic assignment.

&= != ^= |= Bit wise assignment.

2.9. VALUES:

values are the basic unit of data, like a number or a string that a program manipulates.

Example: 2, 42.0 and "Hello world!".

These values belong to different types: 2 is an integer, 42.0 is a floating point number and "Hello World" is a string.

2.10. 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.

⇒ The built in data types are Numbers (Integer type, floating point type, and Complex type), String, List, Tuple, Set, Boolean and None types.

⇒ Number data types store numeric values. Python has three numbers types: integers, number, floating point numbers and Complex numbers.

2.10.1. Integer Type:

An integer type (int) represents signed whole numbers. An integer represent both positive and negative numbers. The range is at least -2,147,483,648 to 2,147,483,647.

⇒ A zero is written as just 0

⇒ To write an integer in decimal (base 10) the first digit must not be zero. eg 25000

⇒ To write an integer in octal (base 8) precede it with "0o" and used the digit 0 to 7, plus A, B, C, D, E, F. eg: 0o177

⇒ To write an integer in hexadecimal (base 16) Precede it with '0x' or "0X" eg: 0x77

2.10.2. Floating Point Type:

A floating point type represents numbers with a fractional part. A floating point number has a decimal point and a fractional point.

eg: 3.0 (or) 3.17 (or) -38.72.

2.10.3. 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.

eg: 5+4j.

```
>>> z = 5+4j
>>> z.real.
    5.0
>>> z.img.
    4.0
```

2.10.4. Boolean type:

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

eg: 2<3 is True.

```
x = (1 == True)
y = (1 == False)
a = True+4.
b = False+10.
Print ("x is", x)   => x is True.
Print ("y is", y)   => y is False
Print ("a:", a)     => a: 5
Print ("b:", b)     => b: 10.
```

eg: >>> a = None.

>>> x

>>> print x

None.

2.11. VARIABLES:

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

Rules for naming variables:

* Variables names can be uppercase letter or lower case letters.
* Variable names can be at any length.
* They can contain both letters and numbers, but they can't begin with a number.
* The underscore (_) character can appear in a name. It is often used in name with multiple words.
* No blank spaces are allowed between variable names.
* A variable name cannot be any one of the keywords or special characters.
* Variable names are case-sensitive.

Eg: Rool_no, GrossPay, a1, Salary.

Creating Variables:

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

```
variable = expr.
```

eg:

```
>>> n=10.
>>> name = "Jovita".
>>> PI = 3.14
>>> 7wonders = 'tajmahal'.
```

SyntaxError: Invalid Syntax.

2.12. EXPRESSIONS:

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

eg: `A*B+C`

where *, + are operators; A, B and C are operands.

Types of expressions:

(i) Infix expressions: The operator is placed in between the operands. eg: a = b+c

(ii) Prefix expressions: The operator is placed before the operands. a = +bc.

(1) Arithmetic Expressions:

This type of expressions just do mathematical operations like +, -, *, / etc.

eg: num1 = 20
num2 = 30
Sum = num1 + num2
Print 'The value of Sum is:', Sum.

The value of Sum is: 50.

(2) Relational / Conditional Expression:

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

eg: a = input ('Enter first number')
b = input ('Enter second number')
Flag = (a>b)
Print 'Is %i greater than %i: %s' %(a, b, Flag)

output: Enter first number 20
Enter second number 31
Is 20 greater than 31: False

(3) Logical Expressions:

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

eg:
```
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)
```

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

(4) Conditional Expression:

The conditional expression is also called relational expression. This expression is used with branching (if, if-else, etc). and looping (while) statement.

eg:
```
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.
```

2.13. STATEMENTS:

A statement is an instruction that the Python interpreter can execute. There are two kinds of statements: assignment (=) and print.

(1) Assignment Statement:

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

eg: >>> Name = 'Jovita'

>>> Age = 9.

>>> Name.

'Jovita'.

>>> Age + 2

11.

(2) Print Statement:

The print statement takes a series of values seperated by commas. Each value is converted into a string and then printed.

eg: >>> name = input("Enter you name:")

Enter you name: Sugitha.

>>> Print ('Hello', name, '!')

Hello Sugitha!

2.14. TUPLE:

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

The values can be of any type, and they are indexed by integers.

eg: >>> t = ('a', 'b', 'c', 'd', 2)
>>> max(5,8,9)
9
>>> min(9, 99, 999)
9

Creating a type:

(i) To create a tuple with a single element,

>>> 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 element of sequence.

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

2.15. TUPLE ASSIGNMENT:

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

eg: >>> T1 = (10, 20, 30)
>>> T2 = (100, 200, 300, 400)

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

2.16. PRECEDENCE OF OPERATORS

Evaluation of expression is based on precedence of operators. When an expressions contains different kinds of operators, precedence determines which operator should be applied first.

**	- Exponentiation.
~ + -	complement, unary plus, & minus.
* / % //	- Multiply, divide, modulus & floor division
+ -	Addition & subtraction.
>> <<	Right & left bitwise - Shift
&	Bitwise and.
^ |	Bitwise exclusive OR
<= >= <>	comparision operators
<> == !=	Equality operators
is, is not	Identity operators
in, not in	Membership operators
not, or, and	Logical operators

2.17. COMMENTS:

* A comment statement contains information for person reading the program.
* Comments make the program easily readable and understandable by the programmer and non-programmer who are seeing the code.
* Comment statements are non-executable statements so they are ignored during program execution.
* In python, a comment starts with #

Example:

```
# Swap.py
# Swapping the values of a two variable program.
# Written by G.Sugitha, April 2018.
```

2.18 MUTABLE AND IMMUTABLE TYPES:

Sometimes we may require changing or updating the values of certain variables used in a program. Python data types can be classified into two types. They are (i) Mutable type (or)
(ii) Immutable type.

(i) **Mutable type:**
variable whose values can be changed after they are created and assigned are called mutable or an object whose state or value can be changed in placed is said to be mutable type.

(ii) **Immutable Type:** variable whose values cannot be changed after they are created and assigned are called immutable state.

2.19. INDENTATION:

Whitespace at the begining of the line is called identation. These whitespaces (or) the identation are very important in python. In a python program, the leading whitespaces including spaces and tabs at the begining of the logical line determines the identation level of that logical line.

eg: voting.py.

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

2.20. INPUT FONCTION:

(1) input() function.

The purpose of inputer function is to read input from the standard input. Using this function, we can extract integer or numeric values.

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

eg: >>> num1 = input("Enter first number:")
Enter first number: 20

(2) raw_input()

The purpose of raw input() function is read input from standard input stream.

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

eg: Name = raw_input("Enter name:")

UNIT-3

CONTROL FLOW, FUNCTIONS, STRING

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

3.1. BOOLEAN VALUES:

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

⇒ True, and False are special values that belong 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.
```

eg:

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

3.2. 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.

Eg: $x = a + b$; where x, a, b are operands, $=, +$ are operators.

3.3. CLASSIFICATIONS OF OPERATORS:

(i) **Unary Operators:** The unary operator operates on only one operand.

+	Unary plus.
−	Unary minus
~	Bitwise inverse.

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

(ii) **Binary Operators:** The binary operator operates on two operands. They are grouped into.

1. Arithmetic operators
2. Relational operator
3. Logical operator
4. Bitwise operator
5. Assignment operator
6. special operators
   a. identity operator
   b. Membership operator

1. Arithmetic Operators:

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

+	Add two operands	⇒ x+y.
-	subtraction.	⇒ x-y.
*	Multiplication.	⇒ x*y.
/	Division.	⇒ x/y.
%	Modules.	⇒ x%y.
//	Floor Division.	⇒ x//y.

(2) Relational Operators:

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

>	- Greater than	⇒ x>y.
<	- Less than.	⇒ x<y.
==	- Equal to	⇒ x==y.
!=	- Not equal to	⇒ x>=y.
>=	- Greater than or equal to	⇒ x<=y.
<=	- Less than or equal to	

(3) Logical operators:

The logical operators are used to logically relate the sub expressions. Logical operators are 'and', 'or' and 'not' operators.

and	- True if both operands are true. x and y.
or	- True if either of operands true x or y.
not	- True if operand is false x not y.

(4) Bitwise Operators:

Decimal numbers are natural to humans whereas binary numbers are native to computers

& - Bitwise AND - x & y = 0 (0000 0000)

| - Bitwise OR. - x | y = 14 (0000 1110)

~ - Bitwise NOT - ~x = -11 (1111 0101)

^ - Bitwise EOR x ^ y = 14 (1111 0101)

(5) Assignment Operators:

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

x = 5 ⇒ x = 5

x += 5 ⇒ x = 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:

(a) Identity Operators:

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

⇒ is and is not are identity operators.

(b) Membership Operators:

Membership operators are used to test whether a value or variable is found in sequence

⇒ in and not in are membership operators in python.

3.4. CONDITIONAL OPERATOR:

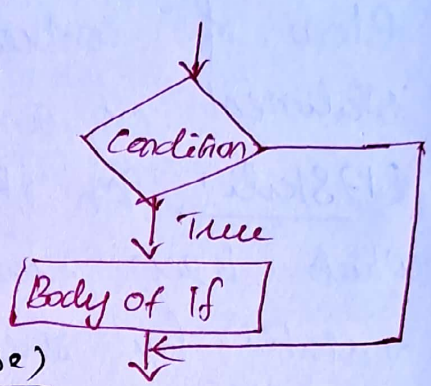
To write useful program, it is needed to check the conditions and change the behaviour of the program.

(i) Conditional Execution (if)

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

Syntax:

```
if condition:
    Statement(s)
```

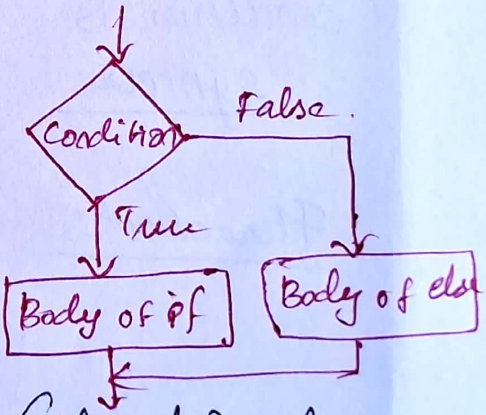


(ii) Alternative Execution (if-else)

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

Syntax:

```
if condition:
    Body of if
else:
    Body of else.
```



(iii) Chained Conditional Execution (if-elif-else)

The elif condition allows checking multiple expression for true value and executing a block of code

Syntax:

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

3.5. Iteration:

Python allow a block of statement to be repeated as many times as long as the processor could support.

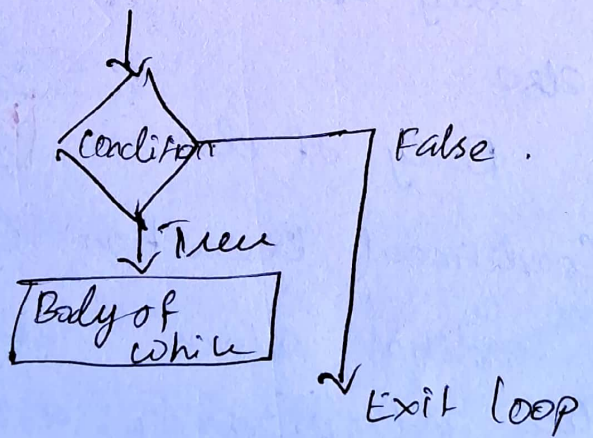
⇒ A looping statiment also influences the flow of control because it causes one or more statiment or a block to be executed repeatly

(1) Stati: An iteration involves stati variables which keep track of the stati of each iteration variable for each pass through the iteration.

(2) While loop: The while loop in Python repeat one or more statiment as long as the particular condition is true.

Syntax: while condition:
Body of while.

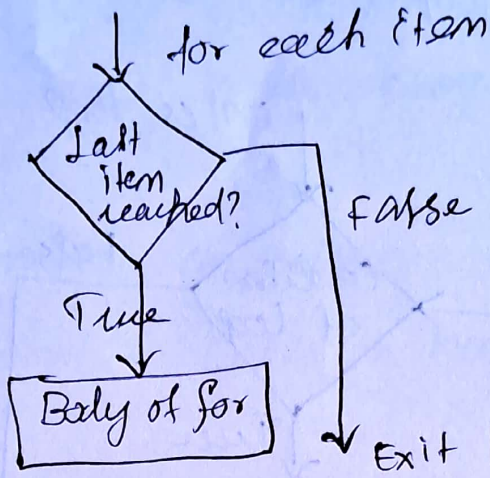
Flowchart:



(3) For loop: The for statiment is used to iterate over a range of values or a sequence The for loop is executed for each of items in the range.

Syntax: for val in sequence:
    body of for.

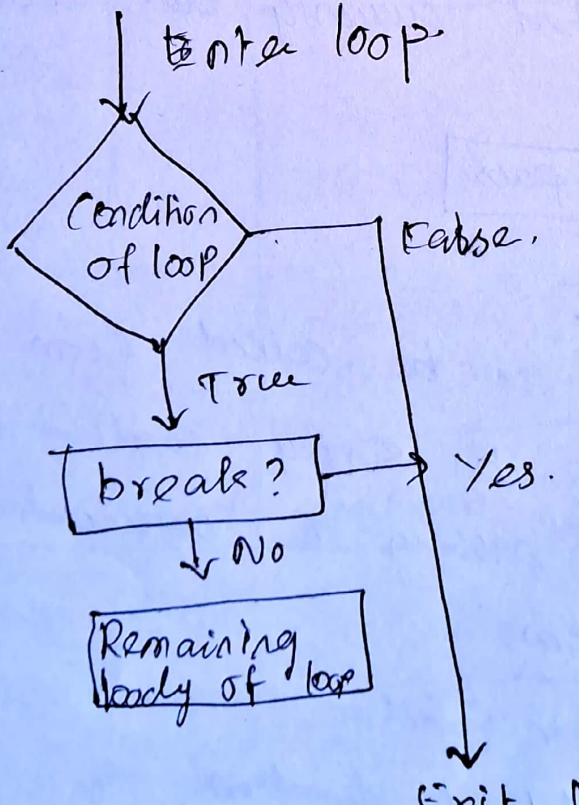
Flowchart:



(4) break: Loop iterates over a block of code until test expression is false, but sometimes we wish to terminate the current iteration or even the while loop without checking test expression.

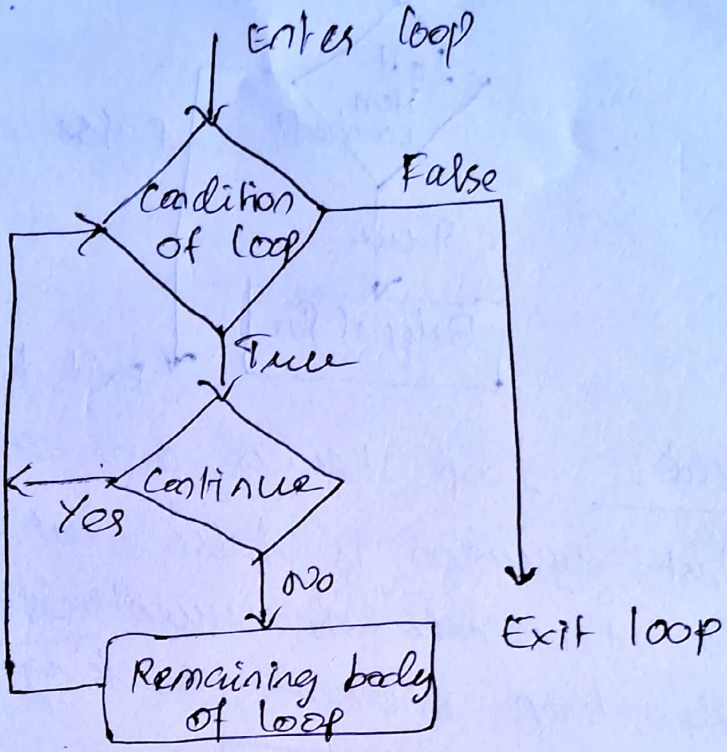
Syntax:

```
break.
```



(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.

Flow Chart:



(6) Pass: In some cases, a loop for a function is not implemented currently but, would be wanted in the future.

Syntax: `pass.`

3.6. FUNCTIONS:

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

Types of function:

(i) Built-in function.

(ii) User-defined function.

(1) Built-in function:

Python has a very extensive Standard library. It is a collection of many built in functions that can be called in the program as and when required

eg: Program to calculate square of number.

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

input(), int(), print() } - built-in functions.

Input or output-
input()
print()

Data type conversions
bool()
chr()
dict()
int()
float()
set()

Mathematical Function.
abs()
divmod()
pow()
sum()

(2) User-defined functions:

We can define our own functions while coding the program. Such functions are called user defined functions.

3.7. ELEMENTS OF USER DEFINED FUNCTIONS:

1. Function declaration.
2. Function call.

(i) Function declaration:

A function definition begins with def. A function definition specific the name of a new function and the sequence of statements that run when the function is called.

Syntax: def function_name (parameters):
    statement(s)

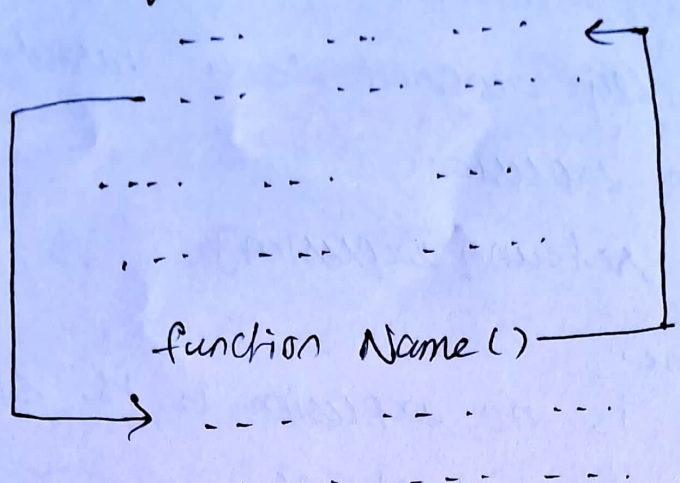
Element of function definition:

(1) Function Header: The first line definition is the function header.
A function header starts with the keyword def, followed by the function name.

(2) Function Body: The function body follows the function header.
The function body contains one or more valid python statements, or the function's instruction

(ii) Function calls:

A function call is a statement that calls a function. It consist of the function name followed by an argument list in parenthesis.



eg: swap.py.

```
def swap(a,b)
    a,b = b,a
    print ("After swap:")
    print ("First no =", a)
    print ("Second no:" b)
swap(a,b).
```

Output

```
enter the first number :67.
enter the second number :98.
(98, 67)
```

3.8. FRUITFUL FUNCTIONS:

Function that return values are called as fruitful functions.

The return statement is followed by an expression which is evaluated. Its result is returned to the caller as the "fruit" of calling function.

(1) Return Statement:

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

Syntax : return [expression].

(2) Return None:

If there is no expression in the statement or the return statement itself.

```
>>> greet ('Ramesh')
    Hello, Ramesh. Good morning.
None.
```

(3) Return values:

The return value may or may not assigned to another variable in the caller. Once the value is returned from the function.

eg:

```
def area(r):
    a = math.pi * r * 2.
    return a.
import math.
r = int(input ('Enter a number:'))
x = area(r)
print ("area =", x).
```

Output:

```
Enter a number : 2.
area = 12.566370614359172
```

(h) Return fruitful return:

In a fruitful function the return statement includes an expression. Fruitful return means "Return immediately from this function".

eg: def area (r):
    return math.pi*r**2

import math.
r = int (input ("Enter a number:"))
a = area (r)
print ("area =", a).

Output:
Enter a number: 2
area = 12.566370614359172

3.9. PARAMETERS AND ARGUMENTS:

parameter: A name used inside a function to refer to the value passed as the arguments.

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

```
def sum(a,b)          # parameter
                      # Function definition.
    s = a+b.
    print ("S =", s)

x = 10
y = 15
sum(x,y)              # Function call.
      # Arguments.
```

3.10. LAMDA FUNCTIONS OR ANONYMOUS FOUCTION

In python, anonymous function is a function that is defined without a name.

⇒ Lamda function is mostly used for creating small and one-time anonymous function.

⇒ Lamda functions are mainly used in combination with the functions like filter(), map()

syntax: lamda [arguments] : expression.

eg: sum = lamda x,y : x+y.
print ("The sum is :", sum(30,40))

The sum is: 70

3.11. SCOPE:

Scope of variable refers to the part of the program, where it is visible. variables in a program may not be accessible at all location in program.

(1) Local Scope: A variable defined in a function body has a local scope.

```
x = 50
def add():
    x = 30          # local variable scope.
    print ("value of x inside :", x)
    x = x + 100
    print ("value of x inside :", x)
```

local scope

(2) Global Scope:

A variable defined outside a function body has a global scope.

```
x = 50          # global variable.
def add(x):
    x+=10
    print('inside add x is':x)
```

Global Scope

3.12. FUNCTION COMPOSITION:

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

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

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

3.13. RECURSION:

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

eg:
```
def fact(x):
    if x==1:
        return 1
    else
        return (x * fact(x-1))
```

```
n= int(input('Enter number'))
print("The factorial of", n, "is", fact(n))
```

Output:

Enter number 4.
The factorial of 4 is 24.

3.14. STRINGS

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

eg:
```
>>> str1 = 'Python is a simple language'
>>> str2 = "Python is free & open source"
>>> str3 = """Python is high level language"""
```

Accessing String Elements:

Index from left:	0	1	2	3	4	5	6	7
characters :	C	o	m	p	u	t	e	r
Index from right:	-8	-7	-6	-5	-4	-3	-2	-1

A string can be accessed by the elements one at a time with the brackets.

```
>>> fruit = 'banana'
>>> fruit[0]
'b'
>>> fruit[4]
'n'
```

3.15 STRING SLICES:

Accessing some part of a String or Substring is known as slicing.

subset of Strings can be taken using the slice operator with two indices in square brackets seperated by colon.

```
>>> a = "Python Progeamming"
>>> a[0:5]
'Pytho'
>>> a[:5]
'Pytho'
>>> a[5:]
'n Progeamming.
>>> a[:]
'Python Progeamming'
```

3.16. STRINGS ARE IMMUTABLE:

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

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

3.17. STRING FUNCTIONS AND METHODS

String provide methods to perform a variety of useful operations. A method is similar to function

S.capitalize()	- capitalize first char.
S.capwords()	- capitalize first letter.
S.count(sub)	- count numbers.
S.find(sub)	- Find first index.
S.index(sub)	- Find first index of sub.
S.rfind(sub)	- Same as find, but last index.

(1) Upper: A method upper takes a string and returns a new string with all uppercase

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

(2) Lower: The method lower takes a string and returns a new string with all lowercase

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

3.18. STRING MODULE:

The string module provides additional tools to manipulate strings. Some methods available in the standard data structures are not available in the string module.

```
>>> import string
>>> string.digits
'0123456789'
>>> string.ascii_letters
'abcdefghijklmnopqrstuvwxyz'
>>> string.octdigits
'01234567'
>>> string.hexdigits
'0123456789abcdefABCDEF'
>>> string.punctuation
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
>>> string.whitespace
'\t\n\r\x0b\x0c'
```

3.19. LISTS AS ARRAYS:

Creating fixed sizes list is similar to creating array in other programming languages.

eg: >>> a = [None] * 5

>>> a

[None, None, None, None, None]

⇒ In some applications the array will hold number data, and zero is more appropriate initial value

>>> data = [0] * 5

eg: >>> a = 'none'

>>> data = [a] * 5

>>> for i in range(0, 5):

>>> data[i] = [0] * 5

>>> data

[[0,0,0,0,0], [0,0,0,0,0], [0,0,0,0,0] [0,0,0,0,0], [0,0,0,0,0]]

>>> data[2][2] = 7

>>> data.

[[0,0,0,0,0], [0,0,0,0,0], [0,0,7,0,0] [0,0,0,0,0] [0,0,0,0,0]]

UNIT - IV

LIST, TUPLES, DICTIONARIES

SYLLABUS: Lists: list operations, list slices, list methods, list loop, mutability, aliasing, cloning lists, list parameters; Tuples: tuple assignment, tuple as return values; Dictionaries: operations and methods; advanced list processing - list comprehension; Illustrative programs: simple sorting, histogram, student mark statement, Retail bill prepration.

4.1. LISTS:

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

4.1.1. Creating a list:

There are several ways to create a new list.

1. The simplest ways is enclose the elements in square brackets ([and]).

eg: A list of four integers : [10, 20, 30, 40]
A list of three strings : ['Jovita', 'Jesvita', 'Jas']
A list of different type : ['March', '2017', 26]

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

empty = [].

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

['Dell', 2.0, [50, 100]].

4.1.2. Assign list values to variables:

The list values can be assigned to variables.

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

4.1.3. Accessing list elements:

Accessing the elements of list is same as for accessing the characters of a string that is using the bracket operator.

eg: >>> icecreams = '['vanilla', 'strawberry', 'mango'].

```
>>> icecreams [0]
'vanilla.
>>> 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 & so on.

```
>>> birds = ['parrot', 'Dove', 'duck', 'cuckoo'].
>>> print (birds[-1])
cuckoo.
>>> print (birds[-2])
Dove.
```

4.2. LIST OPERATIONS:

4.2.1. Concatenation operation:

Concatenation means joining two operands by linking them end-to-end. In list concatenation, '+' operator concatenate two list 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].
```

4.2.2. Repeat Operation:

List can be replicated or repeated or repeatly concatenated with the asterisk operator '*'.

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

4.3. LISTS SLICES!

An individual element of a list is called a slice. Selecting a slice is similar to selecting an element of a list.

```
>>> birds = ['parrot', 'Dove', 'duck', 'cuckoo'].
>>> print (birds [2:4])
['duck', 'cuckoo'].
>>> print ( birds[:3])
['parrot', 'Dove', 'duck'].
>>> print ( birds[2:])
['duck', 'cuckoo'].
```

4.4. LIST METHODS!

append() - Add an element to end of list.
extend() - Add all elements of a list to another list
insert() - Insert an element at defind index.
remove() - Removes and item from list.
pop() - Removes and return an element
clear() - Removes all items in list.

index() - Return index of first matched item.
count() - Returns count number of items
sort() - Sort items in a list in ascending order
reverse() - Reverse the order of items in list.
copy() - Returns a shallow copy of list.

4.4.1. append!

The append method adds a new element to the end of a list.

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

4.4.2. extend!

The extend method takes a list as argument and appends all the element.

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

4.4.3. sort!

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

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

4.4.4. insert:

The insert() method insert an element/object into a list at the index position.

syntax: `List.insert(index, obj)`

```
>>> Name = ['sathan', 'Jovita', 'Varsha',]
>>> Name.insert(2, 'Jesvita')
>>> print (Name)
['sathan', 'Jovita', 'Jesvita', 'Varsha'].
```

4.4.5. count(x):

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

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

4.4.6. len:

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

```
>>> Num = [23, 54, 34, 44, 35, 66, 27, 88, 89, 54]
>>> print len(Num)
10.
```

4.4.7. reverse.

The reverse() method reverse objects of list in place. `List.reverse()`

4.4.8. Max:

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

max(list)

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

4.4.9. min:

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

min(list)

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

4.5. LIST LOOP (TRAVERSING A LIST):

(1) Traversing a list means, process or go through each element of a list sequentially. When the list elements are processed within a loop, the loop variable or a seperate counter is used as an index into the list which prints each counter position elements till the end-1.

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

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

output:

vanilla.
strawberry.
mango.

Q. The 'for loop' works well to read the elements of the list.

for <Index> in range (len (List)):
statements to process <List[index]>

eg: >>> icecreams = ['vanilla', 'strawberry', 'mango']
>>> for i in range (len (icecream)):
print (icecream[i])

output:

vanilla.
Strawberry.
mango.

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

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

4) The nested list will be considered as a single element.

>>> L = ['bell', 2.0, [50, 100]]
>>> print (len(L))

3.

4.6. MUTABILITY:

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

eg: >>> numbers = [5, 10, 6].
>>> numbers [1] = 5
>>> numbers
[5, 5, 6].

4.7. LIST MEMBERSHIP:

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

4.7.1. in Operator:

The in Operator test whether an element is a member of a list or not.

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

4.7.2. not in Operator:

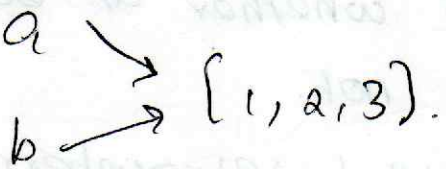
The not in operator evaluates to true if it does not find the element in the list and otherwise false.

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

4.8. ALIASING!

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

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



4.8.1. Aliasing with mutable objects:

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

```
>>> t = [1, 2, 3]
>>> x = t
>>> x [1, 2, 3]
>>> t [1, 2, 3]
```

```
>>> a[0] = 55
>>> a  ⇒ [55, 2, 3].
>>> t  ⇒ [55, 2, 3].
```

4.8.2. Aliasing with immutable Objects:

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

```
a = 'engineering'.
b = 'engineering'.
```

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

4.9. CLONING LISTS:

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

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

⇒ Taking any slice of a list creates a new list.

eg:
```
>>> x = ['a', 'b', 'c']
>>> y = x[:]
>>> print(x)
['a', 'b', 'c'].
```

```
>>> y[0] = 'r'
>>> print(y)
['r', 'b', 'c']
>>> print(x)
['a', 'b', 'c'].
```

4.10. LIST PARAMETERS:

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

Example:

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

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

4.11. MAP, FILTER AND REDUCE:

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

eg:

```
def capitalize_all(t):
    res = []
```

```
for s in t:
    if s.isupper():
        res.append(s)
    return res.
```

isupper is a string method that return True if the string contains only upper case letters.

4.11.3. Reduce:

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

eg:
```
def add_all(t):
    total = 0
    for x in t:
        total += x
    return total.
```

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

4.12. DELETING ELEMENTS:

There are several ways to delete elements from a list.

(i) pop To know 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 return 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'].
```

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'].
```

4.13. MATRIX USING LIST:

A list inside another list is called nested list or a matrix.

the matrix could be created with.

```
>>> m = [[5, 3, 7], [11, 3, 6], [8, 17, 9]]
>>> M[0][1]
```

4.14. TUPLES:

⇒ A tuple is a sequence of values, which can be of any type and they are indexed by integer.

⇒ Tuples are an immutable sequence of elements. That is once a tuple is defined, it cannot be deleted.

⇒ A tuple is a comma-seperated list of values. >>> t = 'a', 'b', 'c', 'd', 'e'.

not neccessary to enclose tuples in parenthesis

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

4.14.1. Creation of tuples:

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

```
>>> t1 = 'a',
>>> type (t1)
<class 'tuple'>
>>> t2 = ('a')
>>> type (t2)
<class str>
```

2. A tuple can be created using the buildin function tuple.

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

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

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

4.14.2. Operators on tuple:

(1) Bracket Operator:

The bracket operator indexes an element.

```
>>> t = ('c','o','m','p','u','t','e','r')
>>> t[0]
'c'
>> t[3]
'p'
```

2. Slice Operator:

The slice operator selects a range of elements.

eg: `>>> t[0:4]`

```
('c','o','m','p')
>>> t[0] = 'c'
```

Type Error: Object doesn't support item assignment.

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

3. Concatenation Operator:

Tuples can be concatenated or joned them using + or concatenation operator.

eg:
```
>>> T1 = ('Pen', 'Pencil', 'Rubber')
>>> T2 = ('Eraser', 'Refil')
>>> T1 += T2.
>>> print (T1)
('Pen', 'Pencil', 'Rubber', 'Eraser', 'Refil')
```

4. Relational Operators:

The relational operators work with tuples and other sequences.

```
>>> (5, 8, 2) < (5, 10, 6)
True.
>>> (3, 2, 500000) < (3, 8, 5)
True.
```

4.14.3. Looping through Tuple Elements:

Tuple element can be traversed using loop control statements.

```
>>> weekdays = ('sunday', 'Monday', 'Tuesday',
                'Wednesday', 'Thursday')
for wd in weekdays:
    print (wd)
```

Sunday
Monday
Tuesday.
Wednesday.
Thursday.

4.15. TUPLE ASSIGNMENT:

Assignment of tuple is a useful feature in Python. It allows a tuple of variables on the left side of the assignment operator to be assigned respective values from a tuple on the right side.

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

4.16. TUPLES AS RETURN VALUES:

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

eg: `>>> t = divmod(13, 4)`

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

⇒ To store the elements seperately, use tuple assignment,

```
>>> Q,R = divmod(13,4)
>>> Q
3
>>> R
1
```

4.17 TUPLE OPERATIONS:

(1) len: The len() method returns the number of items in a tuple.

```
>>> Ta = (100, 200, 300, 400, 500)
>>> len(Ta)
5
```

(2) cmp: The cmp() method is used to check whether the given tuples are same or not.

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

(3) max! The max method returns th largest item in the tuple.

```
>>> T = (100, 200, 300, 400, 500)
>>> max(T)
500.
```

(4) min! the min method returns th smallest item in the tuple.

```
>>> T = (100, 200, 300, 400, 500)
>>> min(T)
100
```

(5) count! The count method count how many items of an object occurs in a tuple.

eg:
```
>>> T = (100, 200, 300, 400, 100, 200, 100)
>>> print (T.count(100))
3.
```

4.18. VARIABLE-LENGTH ARGUMENT TUPLES!

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

```
def printall (*args):
    print(args)
>>> printall(1, 2.0, '3')
(1, 2.0, '3')
```

4.21. DICTIONARIES:

⇒ The data type dictionary fall under mapping. It is a mapping between a set of keys and a set of values.

⇒ Items in dictionaries are unordered, so we may not get back the data in the same order in which we had entered the data initially in the dictionary.

Dictionary-name = { key-1 : value-1, key-2 : value-2
key-3 : value-3 }.

eg:

```
>>> daily-temp = { 'sun' : 68.8 , 'mon' : 70.2 ,
'tue' : 67.2 , 'wed' : 7-18 , 'thur' :
73.2 , 'fri' : 75.6 , 'sat' : 74.0 }
```

4.21.1. Creating a Dictionary:

(1) Dictionary using dict() function:

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

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

(2) Creating empty dictionary using { }.

Dictionary can be created using an empty String. To add an item to the dictionary, use square brackets with keys for accessing values.

(3) Dictionary using literal notation:

Dictionary can be created using literal notation with key-value pair.

directory-name = {key: value, key, value, ...keyN, valueN}.

(4) Adding items into dictionary:

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

```
eg: >>> cse = {}.
>>> print(cse)
{}.
>>> cse['name'] = 'Antony'.
>>> cse['age'] = 18
>>> cse['height'] = 26
>>> print(cse).
{'name': 'Antony', 'age': 18, 'height': 26}.
```

4.21.3. Accessing Dictionary:

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

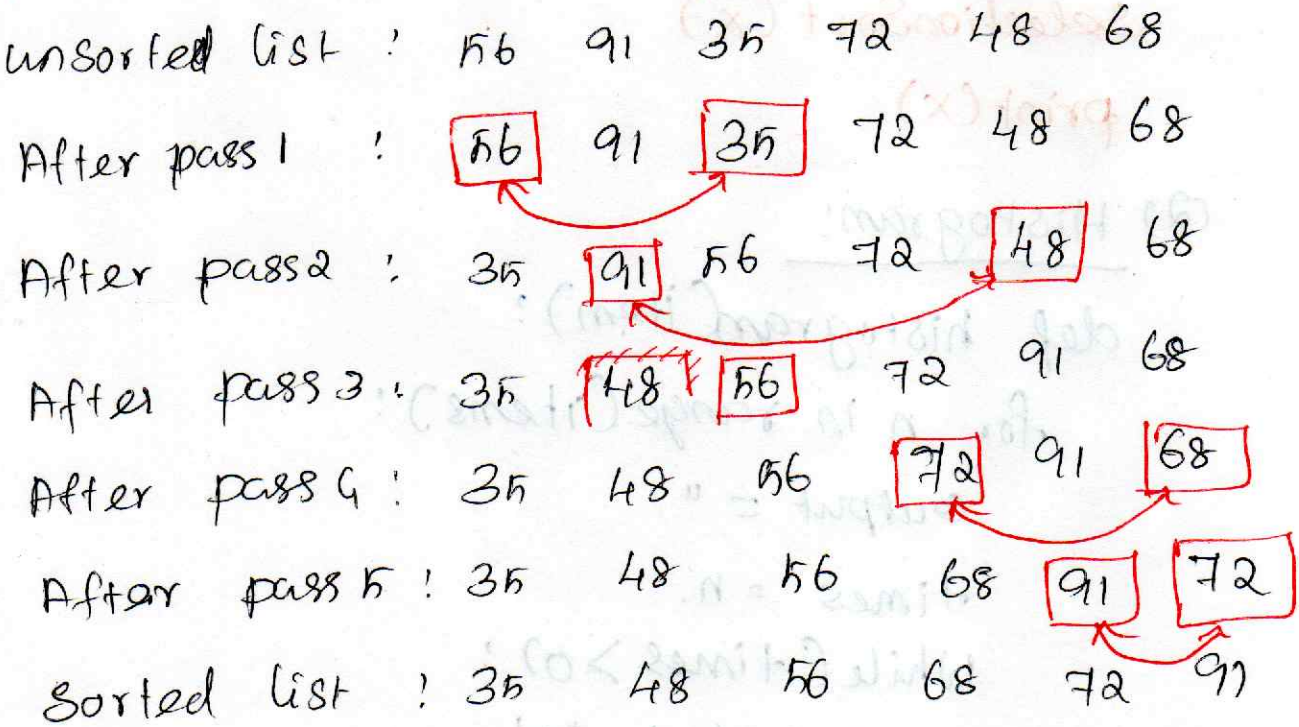
```
>>> cse = {'name': 'Jason', 'age': 19}.
>>> print(cse['name'])
Jason.
>>> print(cse.get('age'))  ⇒ 19.
```

Illustrative Programs:

1. Simple Sorting:

(i) **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. This process of selection, and exchange continues, until all the elements in the list have been sorted in ascending order.

eg: 56, 91, 35, 72, 48, 68.



Program:

```
def selectionSort(A):
    for i in range(len(A)-1, 0, -1):
        max=0
        for j in range(1, i, +1):
            if A[j] > A[max]:
```

```
        max = j
    temp = A[i]
    A[i] = A[max]
    A[max] = temp
    X = [0,0,0,0,0,0,0,0,0,0,0]
    n = int (input ("Enter list size:"))
    print ('Enter number')
    for i in range (0,n):
        X[i] = int (input ())
    selectionSort (X)
    print(X)
```

(2) Histogram:

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

(3) Student Mark Statement

```
mark = []
tot = 0
student = []
grade = []
num1 = int(input("Enter number of students:"))
for num in range(num1):
    x = input("Enter name of student")
    student.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")
    else:
        print("Grade = E")
```

(4) Retail Bill Preparation:

```
tax = 0.18
Rate = {"Pencil":10, "Pen":20, "Scale":7}
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:"))
cost = Pencil * Rate["Pencil"] + Pen * Rate["Pen"]
        + Scale * Rate["Scale"]
Bill = cost + cost * tax
print ("Please Pay Rs. %.f", % Bill).
```

UNIT - V

FILES, MODULES, PACKAGES.

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

5.1. 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.

5.1.1. Text Files:

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

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

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

5.1.2. Binary files:

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

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

5.2. OPENING A FILE:

⇒ The content 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 is 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 name of the file which we want to access.
* access_mode: The value of access_mode specifies the mode in which we want to open the file.
*

Mode	Description.
'r'	- open a file for reading.
'w'	- open a file for writing. It creates a new file if it does not exist or truncates the file.
'x'	- open a file for exclusive creation if the file already exist.
'a'	- open for appending file at the end of the file without truncating it
't'	- open in text mode.
'b'	- open in binary mode.
'+'	- open file for updating

```
>>> f = open("test.txt")
>>> f = open("test.txt", 'w')
>>> f = open("img.bmp", 'r+b')
>>> f = open("test.txt", mode='r',
                encoding='utf-8')
```

5.3. CLOSING A FILE:

⇒ Closing a file will free up the resources that were tied with the fil done using the close() method.

Syntax:

fileobject.close()

eg: >>> f = open("test.txt", encoding="utf-8")

>>> f.close()

5.4. WRITING TO A FILE:

⇒ The write() method is used to write a string to an already opened file.

⇒ To write into a file, it is needed to open a file in write 'w', append 'a' or exclusive creation 'x' mode.

Syntax: fileobject.write(string)

eg: >>> with open("test.txt",'w', encoding="utf-8") as f:

>>> f.write("A journey of \n")

f.write("a thousand miles begins\n\n")

f.write("with a single step\n")

4
1
21

5.4.1. Writelines() Method:

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

eg: file = open("file1.txt", 'w')

```
file.writelines(lines)
file.close()
print("writing to file")
```

5.4.2. Append() method:

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

eg:
```
file1 = open("myfile.txt", "w")
L = ["This is Delhi\n", "This is Paris\n",
     "This is London"]
file1.writelines(L)
file1.close()
```

5.5. READING FROM A FILE:

(1) Reading a file using read(size) method:

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

Syntax: fileobject.read([size])

eg:
```
>>> f = open("test.txt", 'r', encoding='utf8')
>>> f.read(4)
'A jo'
>>> f.read(4)
'urne'
>>> f.read()
```

5.6. READING AND DELETING FILES:

Python OS module has various methods that can be used to perform file-processing operations such as renaming and deleting files.

5.6.1. The rename() function.

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

Syntax: os.rename (current-file-name, new-file-name)

5.6.2. The remove() Method:

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

Syntax: os.remove (file-name)

5.7. FORMAT OPERATOR:

The argument of write should be a string. To put other values in a file, they have to be converted to strings.

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

the syntax of format expression is.

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

key: Mapping key, consisting of paranthesized sequence of characters.

flags: Conversion flags, which effect the result of some conversion types.

width: Minimum field width.

Precision: Precision, given as a '.' followed by the precision.

Length type: Length modifier.

Conversion type: Conversion type of expression.

6.8. COMMAND LINE ARGUMENTS:

Python provides a getopt module that helps to passe command-line options and argments.

$ python test.py arg1 arg2 arg3.

⇒ sys.argv - is the list of command-line command.

⇒ len(sys.argv) is the number of command line arguments.

```
import sys.
print ('Number of arguments :', len(sys.argv),
                                 'arguments')
print ('Argument List :', str(sys.aegv)
```

To run:

```
$ python test.py arg1 arg2 arg3.
```

5.9. ERRORS AND EXCEPTIONS:

(1) Syntax error:

- Error caused by, not following the proper structure of the language is called syntax error or parsing error.

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

(2) Logic errors:

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

⇒ Logic error specifies all type of errors in which the program executes but gives incorrect results.

eg: Divide by zero.

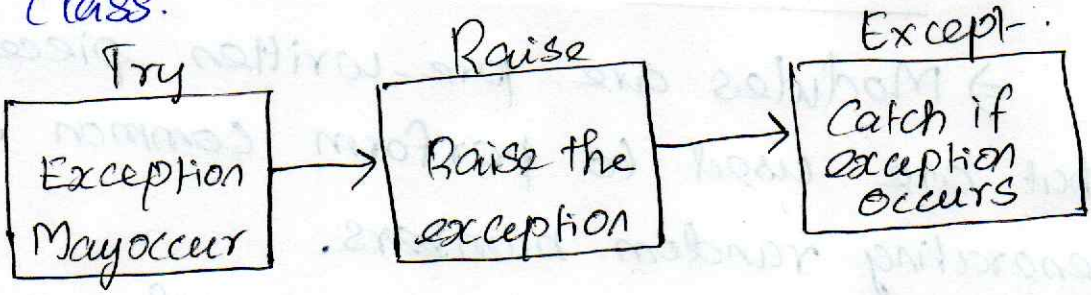
5.10. HANDLING EXCEPTIONS

⇒ When exceptions occur, it makes the current process to stop and passes it to the calling process until it is handled.

⇒ For example, if function A calls function B which in turn calls function C, an exception occur in function C. If it is not handled in C, the exception passes to B and then to A.

5.10.1. Catching Exceptions in Python:

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



Syntax: try:

    Statements
except ExceptionName:
    Statements

5.10.2. Raising Exception:

The exception can be deliberately rasied using the raised keyword.

Syntax: raise [Exception [, args [, traceback]]]

5.10.3. The Finally Block:

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

Pgm:

```
try:
    k = 5/0  # raised divided by zero
    print (k)
except ZeroDivisionError:
    print ("cant divide by zero")
finally:
    print ("This is always executed")
```

5.11. MODULES:

⇒ Modules are pre-written pieces of code that are used to perform common tasks like generating random numbers.

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

5.11.1. Creating a modules:

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

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

5.11.2. The from... Import modules:

⇒ A module may contain definition for many variables and functions.

⇒ To use only selected variables for functions then we can use the from.. import.

eg: >>> import example.
>>> example.add(8,9,5)
17.5

>>> from math import pi
>>> print("The value of pi is", + pi)

5.12. DATETIME MODULE:

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

Syntax: import datetimedate = datetime.date.today()

eg: import datetime
tdate = datetime.date.today()
print 'Today is:', tdate

Today is 2023-11-12

5.13. MATH MODULE:

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

⇒ Python has a math module that provides most of the familiar mathematical functions.

Method 1: import math.

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

Method 2: from math import sqrt.

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

Method 3: from math import *

```
>>> from math import *
>>> print sqrt(100), pi, floor(10.91)
10.0 3.14159265 10
```

5.14. PACKAGES:

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

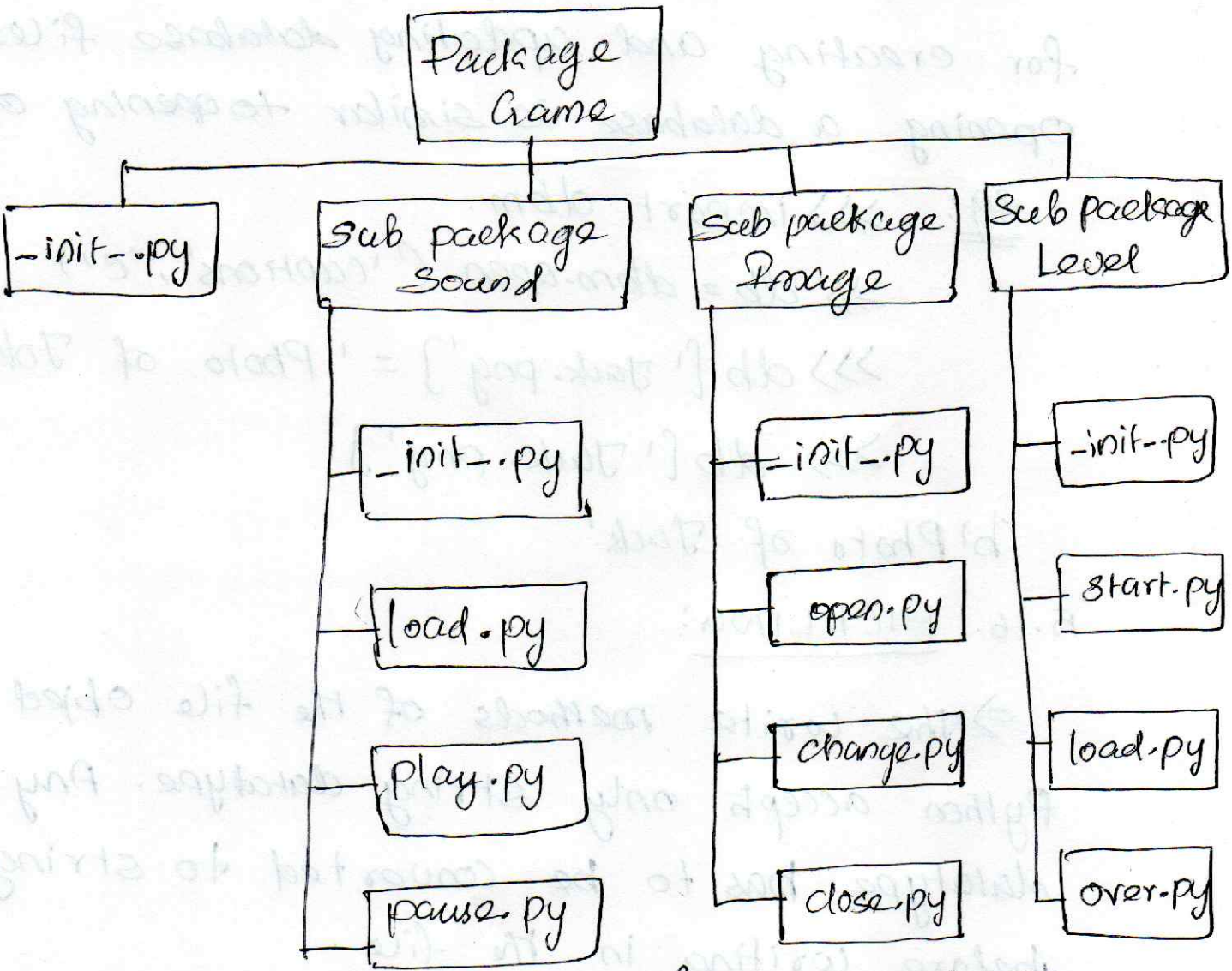
⇒ A directory can contain sub-directories and files, where as a python package can have sub-packages and modules.

5.14.1. Creating a package:

(i) Create a directory and name it with a package name.

(ii) Keep sub directories (sub packages) and modules in it.

(iii) Create -init-.py file in the directory.



5.14.2. Importing module from a package:

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

import Game.Level.start.

5.15. 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.

⇒ The module dbm provides an interface for creating and updating database files. Opening a database is similar to opening other file

eg: >>> import dbm.

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

>>> db ['Jack.png'] = 'Photo of John'

>>> db ['Jack.png']

b'Photo of Jack'

5.16. PICKLING:

⇒ The write methods of the file object in Python accepts only string datatype. Any other datatype has to be converted to string before writing in the file.

⇒ Pickling converts any kind of complex data to 0s and 1s. This process can be referred to as pickling, serialization or marshalling.

pickling can be done with following datatypes,

* Booleans.
* Integers,
* Floats,
* Complex numbers.
* Strings.
* Tuples,
* Lists
* Sets, and
* Dictionaries.

Dump and load:

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

Syntax:

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

To read : pickle.load (<filename>)

eg: import pickle

emp = {1:"A", 2:"B", 3:"C", 4:"D", 5:"E"}.

pickling_on = open ("Emp.pickle", "wb")

```
pickle.dump (emp, pickling_on)
pickling_on.close()
```

To unpickle this dictionary

```
import pickle
pickle_off = open("Empty.pickle", "rb")
emp = pickle.load(pickle_off)
print(emp)
```

O/p:

{1:'A', 2:'B', 3:'C', 4:'D', 5:'E'}.

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 number of string: problem solving and python programming.

Number of words = 5

(2) Copying file:

```
from shutil import copyfile
sourcefile = input("Enter source filename:")
```

```
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 file name : file1.txt
Enter destination file name : file2.txt
File copied successfully!
Sunflower
Jasmine
Roses.
```

(3.) Voters Age Validation:

```
import datetime
Year_of_birth = int(input("In which year you
                    took birth:-"))
Current_year = datetime.datetime.now().year
current_age = current.year - Year_of_birth.
print("Your current age is", current_age)
```

```
if (current_age >= 18):
    print ("you are eligible to vote")
else:
    print ("you are not eligible to vote")
```

Output:

```
In which year you took birth :- 2015
your current age is 8
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 : 100
The value is out of range, try again.
```