KINGS
ENGINEERING COLLEGE
An Autonomous Institution
Affiliated to Anna University, Chennai

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

REGULATIONS 24

II YEAR/III SEM

CS241201 -PROBLEM SOLVING AND PYTHON PROGRAMMING

CS241201 PROBLEM SOLVING AND PYTHON PROGRAMMING L T P C
3 0 0 3

COURSE OBJECTIVES:

- To read and write simple Python programs.
- To develop Python programs with conditionals and loops.
- To define Python functions and call them.
- To use Python data structures –- lists, tuples, dictionaries.
- To do matrix operation using Numpy and DataFrame

UNIT I DATA TYPES, EXPRESSIONS, STATEMENTS 9

Python interpreter and interactive mode, debugging; values and types: int, float, boolean, string and list; variables,expressions, statements, tuple assignment, precedence of operators, comments; Illustrative programs: exchange the values of two variables.

UNIT II CONTROL FLOW, FUNCTIONS, STRINGS 9

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

UNIT III LISTS, TUPLES, DICTIONARIES 9

Lists: list operations, list slices, list methods, list loop, mutability, aliasing, cloning lists, list parameters; Tuples: tuple assignment, tuple as return value; Dictionaries: operations and methods; advanced list processing – list comprehension; Illustrative programs: Students marks statement.

UNIT IV FILES, MODULES, PACKAGES 9

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.

UNIT V NUMPY AND DATAFRAME 9

NumPy-Creating a NumPy – The shape and Reshaping of NumPy Array- Index and Slicing of NumPy Array-Maths with NumPy Arrays- basic arithmetic operations on NumPy array. Pandas Series –Data Frame –Selection and Indexing-Missing data-Merging, Joining, Concatenation- Group By-Apply Function-Sorting-File Read and Write Support

TOTAL: 45 PERIODS

COURSE OUTCOMES:

Upon successful completion of the course, students should be able to:

CO1: Develop and execute simple Python programs.

CO2: Write simple Python programs using conditionals and loops for solving problems.

CO3: Represent compound data using Python lists, tuples, dictionaries etc.

CO4: Read and write data from/to files in Python programs.

CO5: Write simple Python programs using NumPy and Pandas

TEXT BOOKS:

1. Allen B. Downey, "Think Python: How to Think like a Computer Scientist", 2nd Edition, O'Reilly Publishers, 2016.

2. Karl Beecher, "Computational Thinking: A Beginner's Guide to Problem Solving and Programming", 1st Edition, BCS Learning & Development Limited, 2017.

REFERENCES:

1. Paul Deitel and Harvey Deitel, "Python for Programmers", Pearson Education, 1st Edition, 2021.

2. G Venkatesh and Madhavan Mukund, "Computational Thinking: A Primer for Programmers and Data Scientists", 1st Edition, Notion Press, 2021.

3. John V Guttag, "Introduction to Computation and Programming Using Python: With Applications to Computational Modeling and Understanding Data", Third Edition, MIT Press, 2021.

4. Eric Matthes, "Python Crash Course, A Hands - on Project Based Introduction to Programming", 2nd Edition, No Starch Press, 2019.

5. Martin C. Brown, "Python: The Complete Reference", 4th Edition, Mc-Graw Hill, 2018.

CO's-PO's & PSO's MAPPING

CO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1	3	3	3	3	2	-	-	-	-	-	2	2	3	3	-
CO2	3	3	3	3	2	-	-	-	-	-	2	2	3	-	-
CO3	3	3	3	3	2	-	-	-	-	-	2	-	3	-	-
CO4	2	2	-	2	2	-	-	-	-	-	1	-	3	-	-
CO5	1	2	-	-	2	-	-	-	-	-	1	-	2	-	-
AVG	**2**	**2**	**3**	**3**	**2**	**-**	**-**	**-**	**-**	**-**	**1**	**-**	**2**	**3**	**-**

low, 2 - medium, 3 - high, '-' - no correlation

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

FUNDAMENTALS OF COMPUTER

INTRODUCTION TO COMPUTER

- The word "computer" is comes from the word "TO COMPUTE" means to calculate.
- A computer is normally considered to be a calculation device which can perform thearithmetic operations very speedily.
- A computer can store, process & retrieve data as and when we desired.

DEFINITION

- A computer is an electronic device which takes input from the user, processes it and gives the output as per user's requirement.
- So the main tasks performed by the computer are:
    - Input
    - Process
    - Output

CHARACTERISTICS OF AN COMPUTER

Automatic:

- Computers are automatic machines because it works by itself withouthuman intervention.
- Once it started on a job they carry on until the job is finished.
- Computer cannot start themselves.
- They can works from the instructions which are stored inside the system in theform of programs.

Accuracy:

- The accuracy of a computer is very high.
- The degree of accuracy depends upon its design.
- Errors can occur by the computer. But these are due to human weakness or due to incorrect data, but not due to the technological weakness.

Speed:

- Computer is a very fact device. It can perform the amount of work in few seconds for which a human can take an entire year.
- While talking about computer speed we do not talk in terms of seconds and milliseconds but in microseconds.
- A powerful computer is capable of performing several billion (109) simple arithmetic operations per second.

Diligence:

- Unlike human beings, a computer is free from monotony, tiredness & lack of concentration.
- It can continuously work for hours without creating any error & without grumbling.
- If you give ten million calculations to performed, it will perform with exactly the same accuracy & speed as the first one.

Versatility:

- It is one of the most wonderful features about the computer.
- One moment it is preparing the results of a particular examination, the next moment it is busy with preparing electricity bills and in between it may be helping an office secretary to trace an important letter in seconds.

Power of remembering:

- Computer can store and recall any amount of data because of its high storage capacity of its storage devices.
- Every piece of information can be retained as long as desired by the user and can be recalled as and when required.
- Even after several years, if the information recalled, it will be as accurate as on the day when it was filled to the computers.

No I.Q.:

- A computer is not a magical device; it processes no intelligence of its own.
- Its I.Q. is zero.
- It has to be told what to do & in what sequence.
- It cannot take its own decision.

No Fallings:

- A Computer has no feelings because they are machines.
- Computer goes exactly the way which we have given the instructions.

BLOCK DIAGRAM OF COMPUTER

A simple computer system comprises the basic components like

- Input Devices
- CPU(Central Processing Unit)
- Output Devices

Input Devices:

- The devices which are used to enter data in the computer systems are known as input devices.
- Keyboard, mouse, scanner, mike, light pen etc are example of input devices.

Function of Input Devices:

- Accept the data from the outside worlds.
- Convert that data into computer coded information.
- Supply this data to CPU for further processing.

Output Devices:

- The devices which display the result generated by the computer are known as output devices.
- Monitor, printer, plotter, speaker etc are the example of output devices.

Functions of Output Devices

- Accepts the result from the CPU.
- Converts that result into human readable form.

Memory Unit:

- The data & instruction have to store inside the computer before the actual processing start.
- Same way the result of the computer must be stored before passed to the output devices. This tasks performed by memory unit.

Functions of Memory Unit

- Stores data & instruction received from input devices.
- Stores the intermediate results generated by CPU.
- Stores the final result generated by CPU.

Arithmetic & Logical Unit:

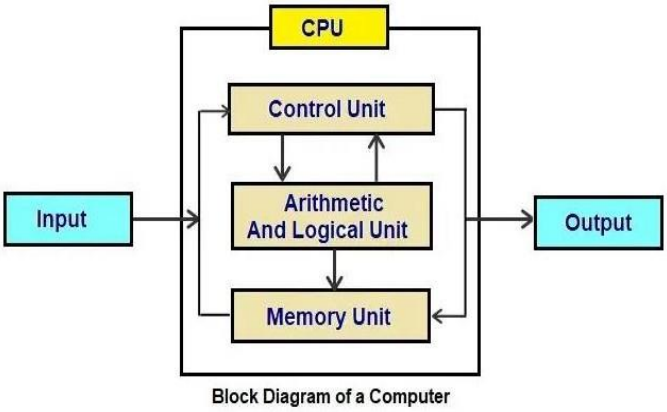
- The ALU is the place where actual data & instruction are processed.
- All the calculations are performed & all comparisons are made in ALU.
- Performs all arithmetic & logical operations.
- An arithmetic operation contains basic operations like addition, subtraction, multiplication, division.
- Logical operations contains comparison such as less than, greater than, lessthan equal to, greater than equal to, equal to, not equal to.

Control Unit:

- It controls the movement of data and program instructions into and out of the CPU, and to control the operations of the ALU.
- In sort, its main function is to manage all the activities within the computer system.
- Controls the internal parts as well as the external parts related with thecomputer**.**

Central Processing Unit:

- The Unit where all the processing is done is called as Central Processing Unit.
- It contains many other units under it.
- Main of them are:- Control Unit And ALU (Arithmetic & Logic Unit)



Block Diagram of a Computer

Input Device

- Keyboard

Pointing Devices

- Mouse
- Track Ball
- Joystick
- Light Pen
- Touchscreen

Output Devices

- **Monitor**

IDENTIFICATION OF COMPUTATIONAL PROBLEMS

WHAT IS A COMPUTATIONAL THINKING?

Computational Thinking (CT) involves a set of problem-solving skills and techniques thatsoftware engineers use to write programs that underlie the computer applications you use such assearch, email, and maps.

There are many different techniques today that software engineers use for CT such as:

Decomposition: Breaking a task or problem into steps or parts.

Pattern Recognition: Make predictions and models to test.

Patten Generalization and Abstraction: Discover the law, or principles that cause thesepatterns.

Algorithm Design: Develop the instructions to solve similar problems and repeat the process.

DECOMPOSITION

Part of being a computer scientist is breaking down a big problem into the smaller problems that make it up. If you can break down a big problem into smaller problems, then you can give them to a computer to solve.

Example, the equation to work out the roots of a quadratic equation:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

On first look, it might appear a little scary, but if we decompose it, we should stand a better chance of solving it:

1. B^2
2. $4ac$
3. $b^2 - 4ac$
4. $\sqrt{b^2 - 4ac}$
5. $-b \pm \sqrt{b^2 - 4ac}$
6. $2a$
7. $x = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$
8. Repeat for $-b - \sqrt{b^2 - 4ac}$

PATTERN RECOGNITION

Often breaking down a problem into its components is a little harder than taking apart an algorithm, we are often given a set of raw data and then are asked to find the pattern behind it: 1, 4, 7, 10, 13, 16, 19, 22, 25, …

This is pretty easy with number sets, the above pattern $n=n+3$. But pattern recognition might also involve recognizing shapes, sounds or images. If your camera highlights faces when you point it at some friends, then it is recognizing the pattern of a face in a picture.

The face was automatically detected by pattern recognition

PATTERN GENERALISATION AND ABSTRACTION

Once we have recognized patterns, we need to put it in its simplest terms so that it can be used whenever we need to use it. For example, if you were studying the patterns of how people speak, we might notice that all proper English sentences have a subject and a predicate.

ALGORITHM DESIGN

Once we have our patterns and abstractions, we can start to write the steps that a computer can use to solve the problem. We do this by creating **Algorithms**. Algorithms are not computer code, but are independent instructions that could be turned into compute code. We often write these independent instructions as **pseudo code**.

TYPES OF COMPUTATIONAL PROBLEM

Decision problem

A decision problem is a computational problem where the answer for every instance is either yes or no. An example of a decision problem is primality testing:

"Given a positive integer *n*, determine if *n* is prime."

For example, primarily testing can be represented as the infinite set

$L = \{2, 3, 5, 7, 11, ...\}$

Search problem

In a search problem, the answers can be arbitrary strings. For example, factoring is a searchproblem where the instances are positive integers and the solutions are collections of primes.

A search problem is represented as a relation consisting of all the instance-solution pairs,called a *search relation*. For example, factoring can be represented as the relation

$R = \{(4, 2), (6, 2), (6, 3), (8, 2), (9, 3), (10, 2), (10, 5)...\}$

which consist of all pairs of numbers (n, p), where p is a nontrivial prime factor of n.

Counting problem

A counting problem asks for the number of solutions to a given search problem. Forexample, a counting problem associated with factoring is

"Given a positive integer *n*, count the number of nontrivial prime factors of *n*."

A counting problem can be represented by a function f from $\{0, 1\}^*$ to the non-negative integers. For a search relation R, the counting problem associated to R is the function

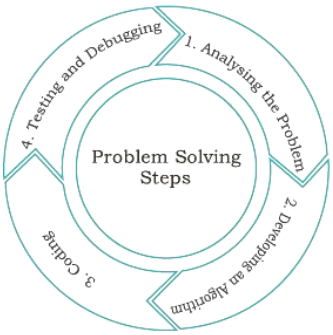
$f_R(x) = |\{y: R(x, y)\}|$.

Optimization problem

An optimization problem asks for finding a "best possible" solution among the set of all possible solutions to a search problem. One example is the maximum independent set problem:

"Given a graph *G*, find an independent set of *G* of maximum size."Optimization problems can be represented by their search relations.

STEPS FOR PROBLEM SOLVING



Analyzing the problem:

It is important to clearly understand a problem before we begin to find the solution for it. If we are not clear as to what is to be solved, we may end up developing a program which may not solve our purpose.

Thus, we need to read and analyze the problem statement carefully in order to listthe principal components of the problem and decide the core functionalities that our solution should have. By analyzing a problem, we would be able to figure out what are the inputsthat our program should accept and the outputs that it should produce.

Developing an 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 can imagine an algorithm like a very well-written recipe for a dish, with clearly defined steps that, if followed, one will end up preparing the dish. 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. For a given problem, more than one algorithm is possible and we have to select the most suitable solution. The algorithm is discussed.

Coding

After finalizing the algorithm, we need to convert the algorithm into the format which can be understood by the computer to generate the desired solution. Different high level programming languages can be used for writing a program. It is equally important to record the details of the coding procedures followed and document the solution. This is helpful when revisiting the programs at a later stage.

Testing and Debugging

The program created should be tested on various parameters. The program should meet the requirements of the user. It must respond within the expected time. It should generate correct output for all possible inputs. In the presence of syntactical errors, no output will be obtained. In case the output generated is incorrect, then the program should be checked for logical errors, if any.

Software industry follows standardized testing methods like unit or component testing, integration testing, system testing, and acceptance testing while developing complex applications. This is to ensure that the software meets all the business and technical requirements and works as expected. The errors or defects found in the testing phases are debugged or rectified and the program is again tested. This continues till all the errors are removed from the program. Once the software application has been developed, tested and delivered to the user, still problems in terms of functioning can comeup and need to be resolved from time to time. The maintenance of the solution, thus, involves fixing the problems faced by the user.

ALGORITHM

(AU QP - What is an algorithm? Summarise the characteristics of a good algorithm.(8) (May 2019))

Definition

- An algorithm is a finite sequence of instructions, or step-by-step procedure for solving a problem.
- An algorithm is a procedure for solving a problem in terms of the actions to be executed and the order in which those actions are to be executed. An algorithm is merely the sequence of steps taken to solve a problem. The steps are normally "sequence," "selection," "iteration," and a case-type statement.

Characteristics of an Algorithm

Well-ordered **-** The steps are in a clear order.
Unambiguous **-** The operations described are understood by a computing agent without further simplification.
Effectively computable **-** The computing agent can actually carry out the operation.

Approaches to Algorithm Design

Top Down

The top down approach to design is starting by examining the full problem or whole picture first. Then divide the problem into smaller components or parts. Then each component is tested separately, and they are combined together to obtain the final solution.

Bottom Up

The bottom up approach to algorithm design starts with the smallest units or parts of a problem. This approach solves the smallest units first and then gradually builds out the next layer or solution. Using this method ensures that the smallest unit has been successfully tested and therefore, when you start solving or implementing the next sub- solution that it will work due to the previous layers working successfully.

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 separate line.
- Steps should be numbered as Step 1, Step 2, and so on.

Properties of an Algorithm

- ***Finiteness*****:** An algorithm must be terminated after a finite number of steps.
- ***Definiteness*****:** Each step of an algorithm must be precisely defined.
- ***Input*****:** The data must be present before any operations can be performed on it. The initial data is supplied by a READ or INPUT Instruction.

  Example: READ A, B

- ***Output*:** After executing all the steps of the algorithm at least one output must be obtained. The WRITE or PRINT statement is used to print messages and variables.
- ***Effectiveness*:** The operations to be performed in the algorithm can be carried out manually in finite intervals of time.

Advantages of Algorithm

- Simple step by step solution of the problem.
- It is easy to understand and debug.
- It is independent of programming languages.
- It can be easily coded into its equivalent high level language.

BUILDING BLOCKS OF ALGORITHMS

Algorithm

An algorithm is a sequence of simple steps that can be followed to solve a problem. These steps must be organized in a logical and clear manner.

Algorithms can be designed using the below basic methods:

- Statements
- State
- Control Flow
  - Sequence
  - Selection or Conditionals
  - Repetition or Loop
- Functions

I. Statements

A **statement** is a command given to the computer that instructs the computer to take a specific action, such as display to the screen, or collect input etc. A computer program is made up of a series of statements. A statement in algorithm may be,

- Description of problem
- Set up
- Parameters
- Execution
- Conclusion

1. Description of problem: To find the sum of two numbers.
2. Set up: Two numbers required for addition and one variable for storing the result.
3. Parameters:
   1. Read 1st number
   2. Read 2nd number
4. Execution: Calculate the sum of the two numbers read (let us say two numbers are ‘a’ and ’b’.
   Result=a+b
4. Conclusion: The desired result is sum.

II. State

State is information your program manipulates to accomplish some task. It is data or information that gets changed or manipulated throughout the runtime of a program.

III. Control Flow Statements

Building Block	Common name	Example
Sequence	Action	Product of two numbers
Selection	Decision	Odd or Even
Iteration	Repetition or Loop	Display first 10 Natural numbers

Sequence:

Sequential control means that the steps of an algorithm are carried out in a sequential manner, where each step is executed exactly once.

Example 1: Conversion of Fahrenheit degrees into Celsius.
An algorithm to solve this problem would be:

Step 1: Start
Step 2: Read Temperature in Fahrenheit
Step 3: Calculate Celsius=(5/9)*(Farenheit-32)
Step 4: Print Celsius
Step 5: Stop

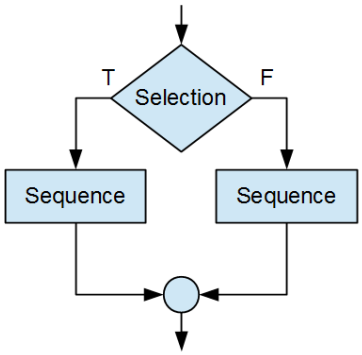
Example 2: Product of two numbers

Step 1: Start
Step 2: Read value for x, y
Step 3: Calculate Prod=x*y
Step 4: Print Prod
Step 5: Stop

Selection or Condition Control:

In selection control only one of a number of alternative steps is executed based on the condition. Let's take an example to find the biggest number from the two numbers.

Step 1: Start
Step 2: Read first number 'a'
Step 3: Read second number 'b'
Step 4: IF(a>b)
print 'A is Big'
Step 5: ELSE
Print 'B is Big'
Step 6: Stop



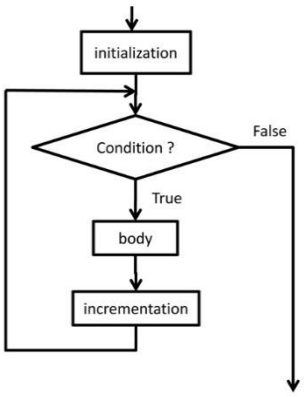
This is the simplest case of selection control. In fact, the third step is only taken when the condition contained in the brackets is true, if it is true then it prints 'A is Big' Otherwise, it prints 'B is Big'

Repetition or Control Flow:

In repetition one or more steps are performed repeatedly. This logic is used for producing loops in a program logic, when one or more instructions may be executed several times or depending on some conditions.

Example: Display First 10 Natural Numbers

Step 1: Start
Step 2: Set i=1
Step 3: Repeat Step 4 & 5 until i<=10
Step 4: Print i
Step 5: Calculate i=i+1
Step 6: Stop



IV. Functions

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

Other Terms: methods, sub-routines, procedure

Display 1 to 10 using Function:
Step 1: Start
Step 2: Call Display(10)
Step 3: Stop

Display(n):
Step 1: Set i=1
Step 2: Repeat step 3 and 4 until IF i<=n
Step 3: Print i
Step 4: i=i+1 [End of Loop]
Step 5: Stop

NOTATION

I. Pseudocode:

- It is a short, readable and formally-styled English language used for explaining algorithm.
- It is also called as Program Design Language.
- It is a short-hand way of describing programming statements.
- It consist of short English Phrases called Keywords.

Rules for writing pseudo code:

1. Write one statement per line.
2. Capitalize initial keywords
3. Use Indent to show hierarchy in selection and iteration statements.
4. Keywords must be used.
5. Do not include

Pseudocode Keyword (OR) Notations:

Input: READ, OBTAIN, GET
Output: PRINT, DISPLAY, SHOW
Compute: COMPUTE, CALCULATE, DETERMINE
Initialize: SET, INIT
Condition: IF, IF-ELSE, ELSE-IF, END-IF
Iteration: WHILE, LOOP
Function: DEFINE, CALL
Add one: INCREMENT

Example: Sum of 2 Numbers:

```
BEGIN
READ A, B
COMPUTE Sum = A + B
PRINT Sum
END
```

Advantages of Pseudo Code:

- Since it is independent of any language, it can be used by most programmers.
- It is easy to develop a program from pseudo code than with a flowchart.
- It is easy to translate pseudo code into a programming language.
- Pseudo code is compact and does not tend to run over many pages.

Disadvantages of Pseudo Code:

- It does not provide visual representation of the program's logic.
- There are no accepted standards for writing pseudo codes.
- It cannot be compiled nor executed.

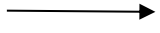
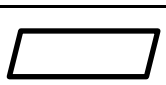
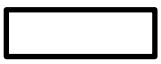
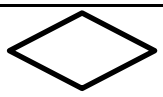
Example 2: #Leap Year or Not

```
READ year
IF ((year%4==0) && (year % 100!= 0)) || (year%400 == 0))  THEN
        Display "Leap year"
ELSE
        Display "Not Leap year"
END IF
END
```

II. Flowchart

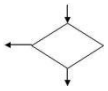
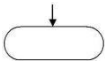
- Flowchart is a Graphical or diagrammatic representation of the logic for solving a task.
- Used to design complex task for better understanding of its visual form.

Symbols Used In Flowchart:

Symbol	Purpose	Description
	Flow line	Used to indicate the flow of logic by connectingsymbols.
	Terminal(Stop / Start)	Used to represent start and end of flowchart.
	Input / Output	Used for input and output operation.
	Processing	Used for arithmetic operations and data-manipulations.
	Decision	Used to represent the operation in which there aretwo alternatives, true and false.
	On-page Connector	Used to join different flow line.

	Off-page Connector	Used to connect flowchart portion on differentpage.
	Function / Predefined Process	Used to represent a group of statements performing one processing task.

Rules for drawing Flow Chart:

- The flowchart should be clear, neat and easy to follow.
- The flowchart must have a logical start and finish.
- Only one flow line should come out from a process symbol. 
- Only one flow line should enter a decision symbol. However, two or three flow lines may leave the decision symbol. 
- Only one flow line is used with a terminal symbol. 
- Within standard symbols, write briefly and precisely.
- Intersection of flow lines should be avoided.
- It is useful to test the validity of the flowchart with normal/unusual testdata.

Advantages of Flowchart:

- It is the most efficient way of communicating the logic of system.
- It acts like a guide for blueprint during program design.
- It also helps in debugging process.
- Using flowchart we can easily analyze the programs.
- Flowcharts are good for documentation.

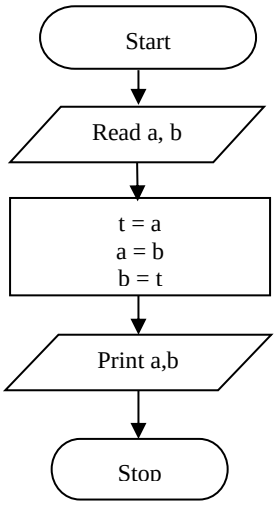
Disadvantages of Flowchart:

- Flowcharts are difficult to draw for large and complex programs.
- It does not contain the proper amount of details.
- Flowcharts are very difficult to modify**.**
- Costly

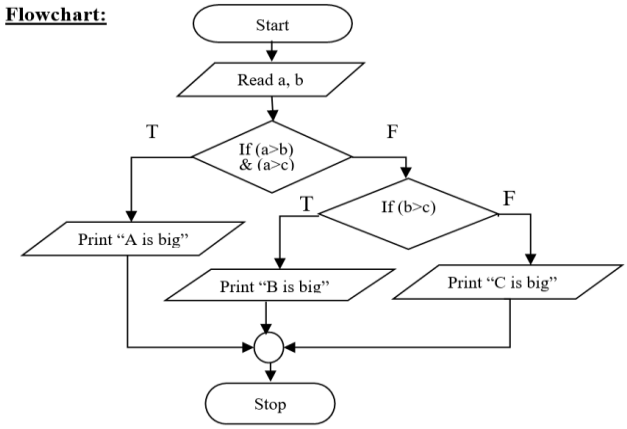
Reasons for using flowcharts as a problem solving tool:

- Makes Logic Clear
- Better Communication
- Effective Analysis
- Effective Synthesis
- Effective coding
- Appropriate documentation
- Systematic debugging
- Systematic Testing
- Efficient program maintenance

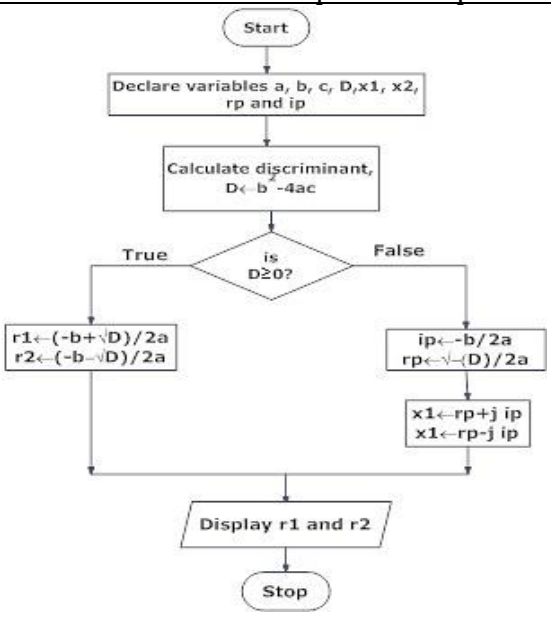
Example 1: Draw a flowchart to swap two numbers entered by user.



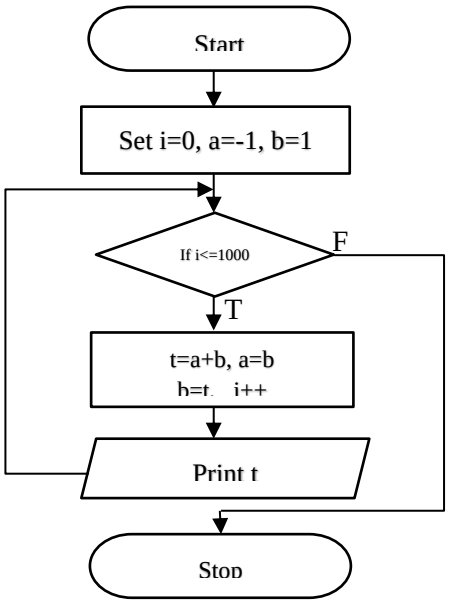
Example 2: Draw flowchart to find the largest among three different numbers entered by user.



Example 3: Draw a flowchart to find all the roots of a quadratic equation ax2+bx+c=0



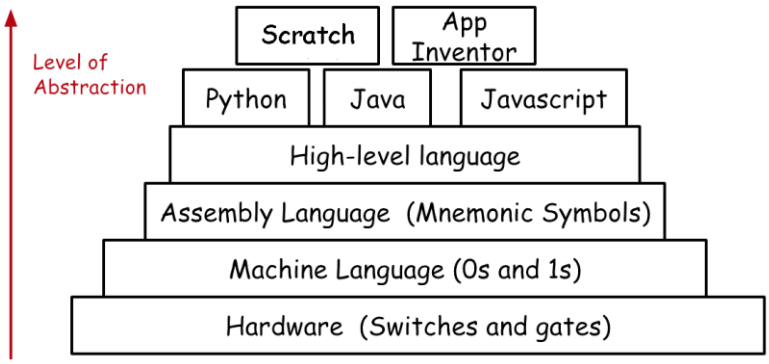
Example 4: Draw a flowchart to find the Fibonacci series till term ≤ 1000



III. Programming Language: *(AU QP - What is a programming language? What are its types? Explain them in detail with their advantages and disadvantages. (8) (Nov / Dec 2019)*

Definition of programming language

- A programming language is a set of commands, instructions, and other syntax that are used to create a software program.
- We generally write a computer program using a high-level language.
- A high-level language is one which is understandable by us humans.
- It contains words and phrases from the English (or other) language.
- But a computer does not understand high-level language.
- It only understands program written in 0's and 1's in binary, called the machine code.
- A program written in high-level language is called a source code.
- We need to convert the source code into machine code and this is accomplished by compilers and interpreters.



Hierarchy Level of Languages

Low Level Language:

- A low-level language is a programming language that deals with a computer's hardware components and constraints. It has no (or only a minute level of) abstraction in reference to a computer and works to manage a computer's operational semantics.
- A low-level language may also be referred to as a computer's native language.
- Machine language and assembly language are popular examples of low-level languages.

High Level Language:

- A high-level language (HLL) is a programming language such as C, FORTRAN, or Pascal that enables a programmer to write programs that are more or less independent of a particular type of computer.
- Such languages are considered high-level because they are closer to human languages and further from machine languages.

What is a program?

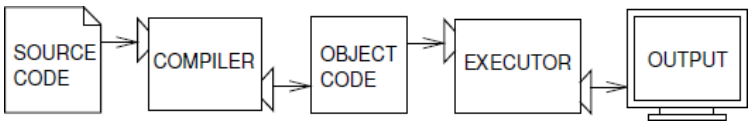
- A program is a sequence of instructions that specifies how to perform a computation.
- The computation might be something mathematical, such as solving a system of equations or finding the roots of a polynomial, but it can also be a symbolic computation, such as searching and replacing text in a document or (strangely enough) compiling a program.

Instruction

- An instruction is an order given to a computer processor by a computer program.
- An instruction is a segment of code that contains steps that need to be executed by the computer processor. Few basic instructions appear in just about every language
    - **input**: Get data from the keyboard, a file, or some other device.
    - **output**: Display data on the screen or send data to a file or other device.
    - **math or calculation**: Perform basic mathematical operations like addition and multiplication.
    - **conditional execution**: Check for certain conditions and execute the appropriate code.
    - **repetition:** Perform some action repeatedly, usually with some variation.

Compiler:

- A compiler or an interpreter is a program that converts program written in high- level language into machine code understood by the computer.



Interpreter:

The difference between an interpreter and a compiler is given below:

Interpreter	Compiler
Translates program one statement at a time.	Scans the entire program and translates it as a whole into machine code.
It takes less amount of time to analyze the source code but the overall execution time is slower.	It takes large amount of time to analyze the source code but the overall execution time is comparatively faster.
No intermediate object code is generated, hence are memory efficient.	Generates intermediate object code which further requires linking, hence requires more memory.
Continues translating the program until the first error is met, in which case it stops. Hence debugging is easy.	It generates the error message only after scanning the whole program. Hence debugging is comparatively hard.
Programming language like Python, Ruby use interpreters.	Programming language like C, C++ use compilers.

ALGORITHMIC PROBLEM SOLVING

Problem Solving

- The process of working through details of a problem to reach a solution. Problem solving may include mathematical or systematic operations.
- The problem solving process starts with the problem specification and ends with a program.
- The steps to do in the problem solving process may be: problem definition, problem analysis, algorithm development, coding, program testing and debugging, and documentation.
- The stages of analysis, design, programming, implementation, and operation of an information system forms the life cycle of the system.

Introduction to Problem Solving in Every Day life:

- People make decision every day to solve problems that affect their lives. If bad decision is made, time and resource are wasted, so it is important that people should know how to take right decisions.

There are six steps to follow to ensure the best decisions. These are given below

1. **Identify the Problem**:
   - You identify the problem before you start solving it. If you don't know what the problem is, you cannot solve it.
2. **Understand the Problem**:
   - You must understand what is involved in the problem before you continue towards the solution. You cannot solve the problem if you do not know the subject.
3. **Identify alternative ways to solve the problem**:
   - Find out different alternatives for solving problem. Alternative solutions must be acceptable ones.
4. **Select the best way to solve the problem from the list of alternative solutions**:
   - In this step you identify and evaluate the pros and cons of each possible solution before selecting the best one for doing this, you should have selection criteria for the evaluation which will serve guidelines for evaluating each solution.
5. **List instructions that enable you to solve the problem using the selected solution**:
   - List step by step instruction from knowledge base for solving the problem using the selected solution. No instruction can be used unless the individual or the machine can understand it.

6. Evaluate the solution:

- Evaluation of solution means to check its result to see if it is correct, and to see if it satisfies the needs of the person with the problem. If the result is unsatisfactory, restart the process.
- If any of these six steps are not completed well, the results may be less than desired.

Testing and Debugging

- Testing means running the program, executing all its instructions/functions, and testingthe logic by entering sample data to check the output.

Debugging is the process of finding and correcting program code mistakes. It finds syntax errors, runtime errors, logic errors(or so called bugs).

Problem solving tools are essentially tools and techniques for

- Gathering information
- Analyzing causes and problems
- Identifying alternative solutions
- Prioritizing importance
- Generating new ideas and solutions

SIMPLE STRATEGIES FOR DEVELOPING ALGORITHMS

(AU QP - Identify simple strategies for developing an algorithm. (Jan-2019))

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

1. Iteration

Iteration is a process of repetition of a set of operations or instructions for a specific number of times until a particular condition is satisfied. In a computer program, one form of iteration is a loop, which repeats code until a certain condition is met. Another is a recursive function, which may repeatedly execute itself as part of its operation.

There are two types of Loops:

1. Entry Controlled Loop
2. Exit Controlled Loop

A Loop must include:

1. Initial value for Counter Variable
2. Condition
3. Body of the Loop to iterate
4. Incrementing/Decrementing the Counter Variable

Entry Controlled Loop:

Entry controlled loop is a loop in which the test condition is checked first, and then loop body will be executed. In entry controlled loop, if the test condition is false, loop body will not be executed. It is also called as Counter Controlled Loop.

Example:

Algorithm to find sum of odd numbers between 1-100

Step 1: Start
Step 2: Set i=1, Sum=0
Step 3: Repeat Steps 4&5 until i<=100
Step 4: Sum=Sum+i
Step 5: i=i+2
Step 6: Print Sum
Step 7: Stop

Exit Controlled Loop:

An exit control loop checks the condition for exit and if given condition for exit evaluate to true, control will exit from the loop body or else control will enter again into the loop. An example of exit controlled loop is Do While Loop. ***But do-while loop is not supported by python***.

Example:

Algorithm to find sum of Even numbers between 1-100

Step 1: Start
Step 2: Set i=2, Sum=0
Step 3: Sum=Sum+i
Step 4: i=i+2
Step 5: Repeat Steps 3&4 until i<=100
Step 6: Print Sum
Step 7: Stop

Recursion: *(AU QP - What is recursive function? What are its advantages and disadvantages? Compare it with iterative function. (6) (Nov / Dec 2019)*

Recursion is the process of calling a function by itself again and again. A function which calls itself again and again until some condition is satisfied is called Recursive function or Recursion. It is similar to looping.

In order to write a recursive program, the user must satisfy the following.

- The problem must be analyzed and written in recursive form.
- The problem must have the stopping condition.

Example 1: Factorial of a number

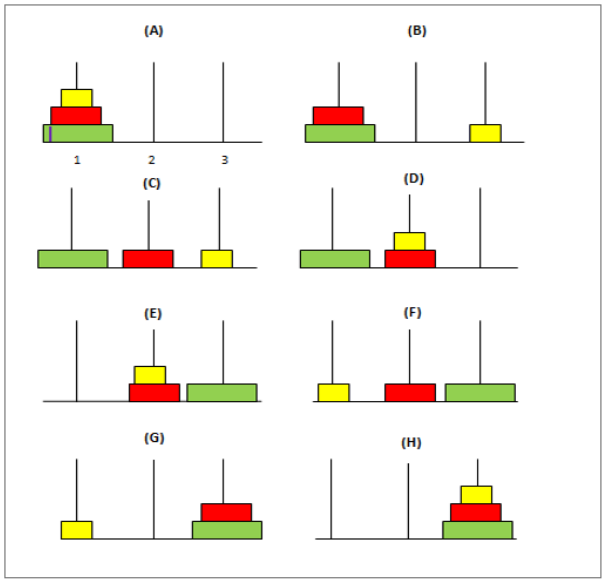
Step 1: Start
Step 2: Input number as n
Step 3: Call **Factorial(n)**
Step 4: Print Factorial
Step 5: Stop

Factorial(n)
Step 1: Set f=1
Step 2: IF n==1
return 1
Step 3: Else
return(n*factorial(n-1)
Step 4: Stop

Example 2: Tower of Hanoi

The Tower of Hanoi is a children's playing game, played with three poles and a number of different sized disks, each disk has a hole in center, allowing it to be stacked around any of the poles.

- Initially the disks are stacked on the left most pole in the order of decreasing size, i.e., the largest on the bottom and the smallest on the top.
- The objective of this game is to transfer the disks from the left most pole to the right most pole, without ever placing a larger disk on the top of the smaller disk.
- Only one disk may be moved at a time and each disk must always be played around one of the poles.



This can be achieved through the following recursive algorithm:

i. Move the top n-1 disks from the left pole to the center pole.
ii. Move the n^{th} disk(largest) to the right pole.
iii. Move the n-1 disks on the center pole to the right pole.

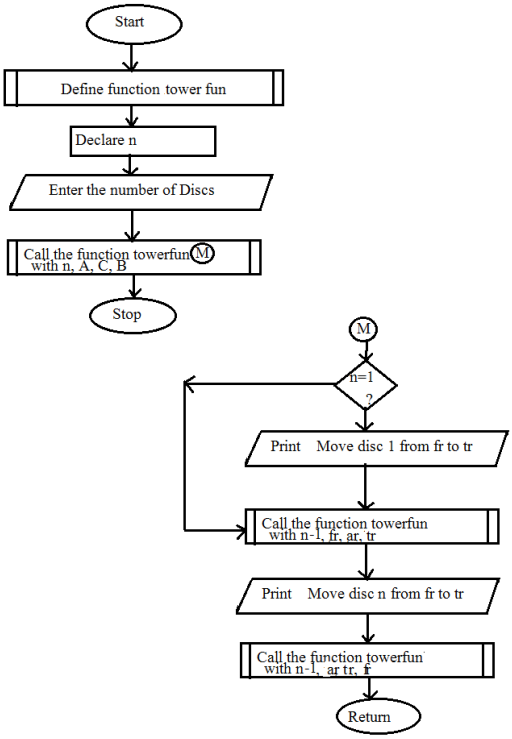
Algorithm:(Main program)

Step 1 : Start the program
Step 2 : Read n as number of disks
Step 3 : Call the function Hanoi (n, source, dest, aux)
Step 4 : Stop

Hanoi (n, source, dest, aux)

Step 1 : if n==1,
    Step 1.1 : move disk from source to dest
Step 2 : else
    Step 2.1 : Hanoi(n-1, source, aux, dest)
    Step 2.2 : move disk from source to dest
    Step 2.3 : Hanoi(n-1, aux, dest, source)
Step 3 : return

Flowchart:



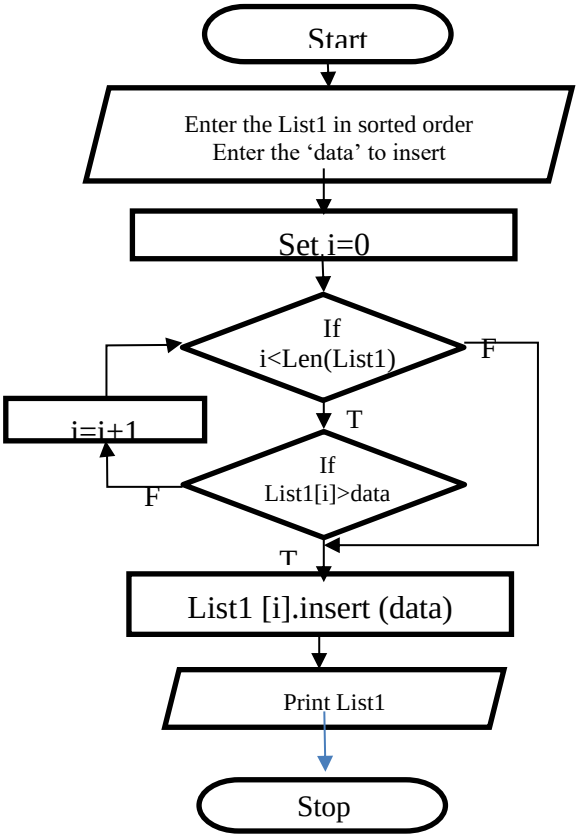
Illustrative Problems

1. Inserting a card in a list of sorted cards Problem Statement: (AU QP – Jan'2019)

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 itinto the correct place so that the cards you are holding are still sorted.

Algorithm:

Step 1: Start
Step 2: Enter the 'List1' in sorted order
Step 3: Enter the 'data' to insert
Step 4: Set i=0
Step 5: Repeat Step 6 & 7 until (i<len(List1)
Step 6: if List1[i]> data
Step 7: List1[i].insert(data)
Step 8: Calculate i=i+1
Step 9: Print List1
Step10: Stop



2. Guessing an integer number in a range.

PROBLEM STATEMENT:

Imagine you are trying to guess a number in a range. 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 correctly, output an appropriate message.

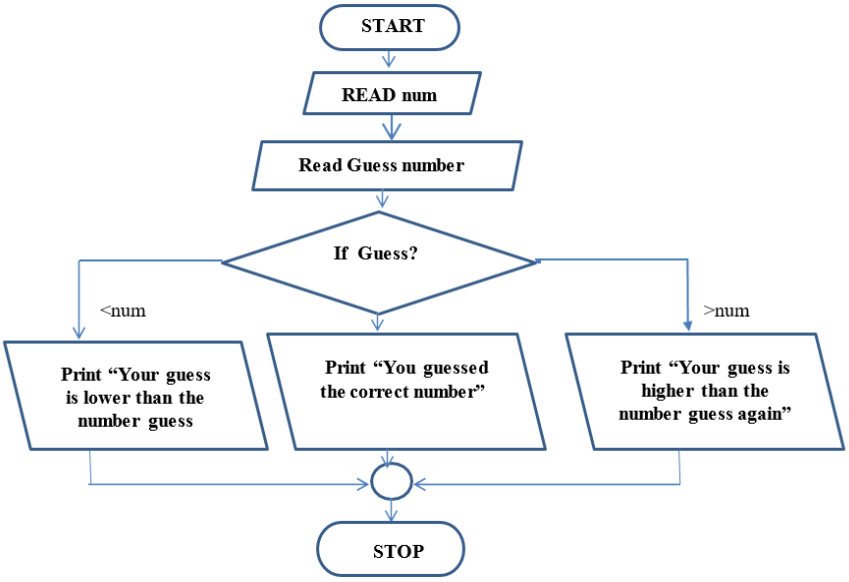
Algorithm**:**

Step 1: Start
Step 2: Generate a random number and call it num.
Step 3: Repeat the following steps until the player has guessed the correct number.(i)Enter the number to guess.
(ii) if (guess is equal to num)
Print "You guessed the correct number"
Otherwise
if(guess is less than num)
print " Your guess is lower than the number. Guess again!"otherwise
print "Your guess is higher than the number. Guess again!"Step 4: End

Pseudocode:

START
READ Guess_num,input_num
IF guess_num>input_num THEN
WRITE “Your guess is higher than the input_number!”ELSE
IF guess_num<input_num THEN
WRITE “Your guess is lower than the input_number!”ELSE
WRITE “You guessed the correct number!”

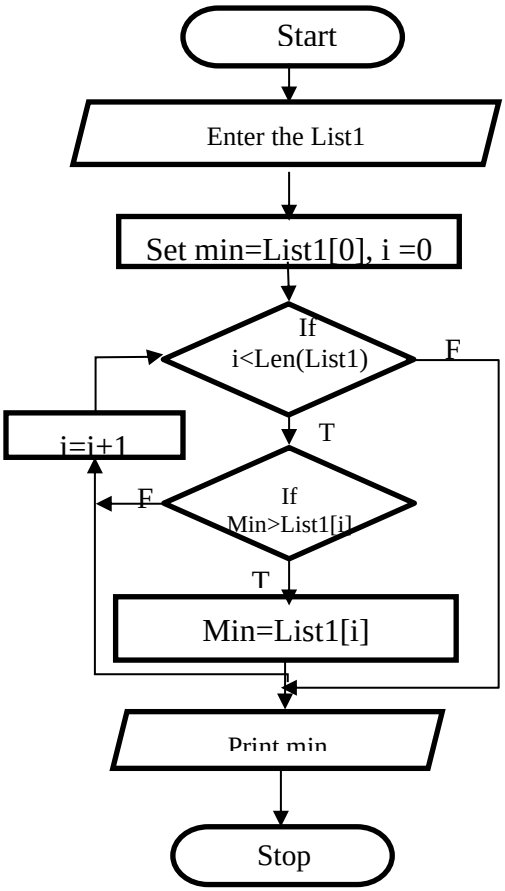
Flowchart



3. Finding Minimum From a List:

Algorithm:
Step 1: Start
Step 2: Read value for ‘List1’
Step 3: Set min=List1[0], i=0
Step 4: Repeat Step 5 & 6 until if(i<Len(List1))
Step 5: if(min>List1[i])
Set min=List1[i]
Step 6: Calculate i=i+1
Step 7: Print min
Step 8: Stop

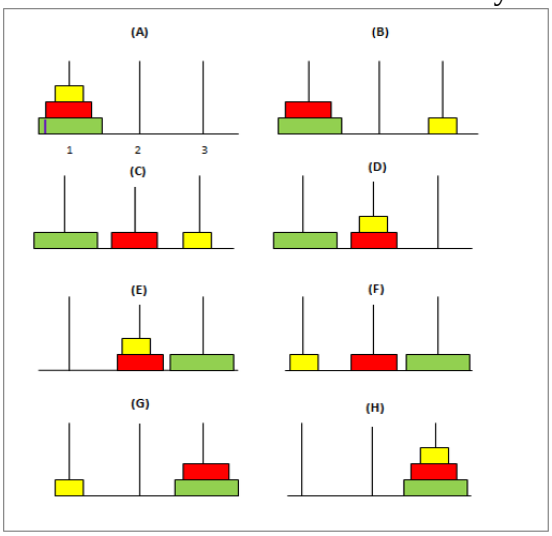
Flowchart



4. Tower of Hanoi: *(AU QP - Outline the Towers of Hanoi problem. Suggest a solution to the Towers of Hanoi problem with relevant diagrams. (16) (Jan-2018))*

The Tower of Hanoi is a children's playing game, played with three poles and a number of different sized disks, each disk has a hole in center, allowing it to be stacked around any of the poles.

- Initially the disks are stacked on the left most pole in the order of decreasing size, i.e., the largest on the bottom and the smallest on the top.
- The objective of this game is to transfer the disks from the left most pole to the right most pole, without ever placing a larger disk on the top of the smaller disk.
- Only one disk may be moved at a time and each disk must always be played around one of the poles.



This can be achieved through the following recursive algorithm:

iv. Move the top n-1 disks from the left pole to the center pole.
v. Move the n^{th} disk(largest) to the right pole.
vi. Move the n-1 disks on the center pole to the right pole.

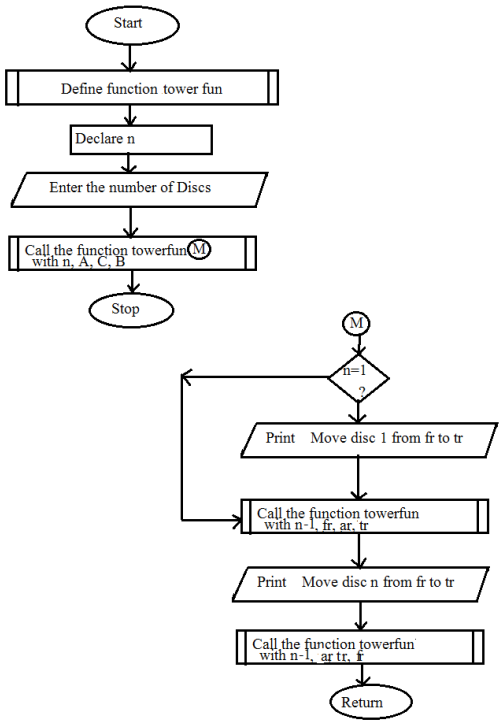
Algorithm:

Step 1 : Start the program
Step 2 : Read n as number of disks
Step 3 : Call the function Hanoi (n, source, dest, aux)
Step 4 : Stop

Hanoi (n, source, dest, aux)

Step 1 : if n==1,
    Step 1.1 : move disk from source to dest
Step 2 : else
    Step 2.1 : Hanoi(n-1, source, aux, dest)
    Step 2.2 : move disk from source to dest
    Step 2.3 : Hanoi(n-1, aux, dest, source)
Step 3 : return

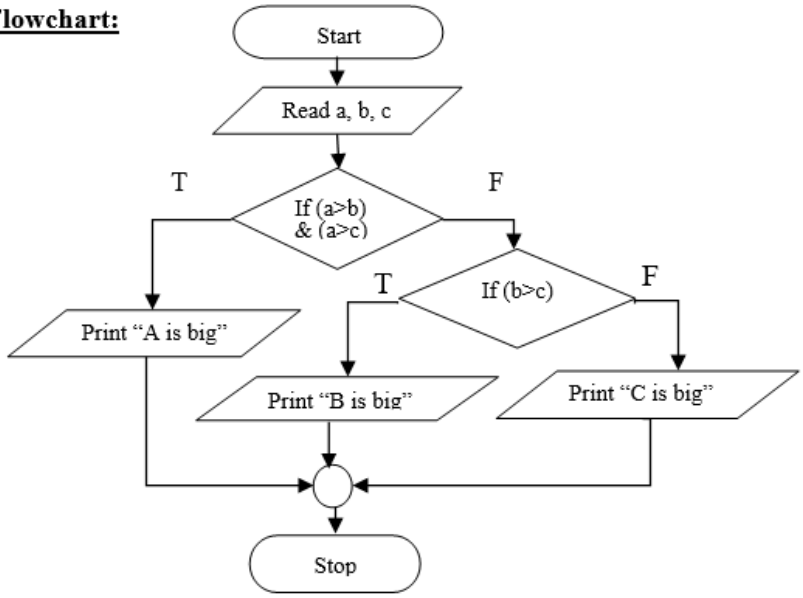
Flowchart:



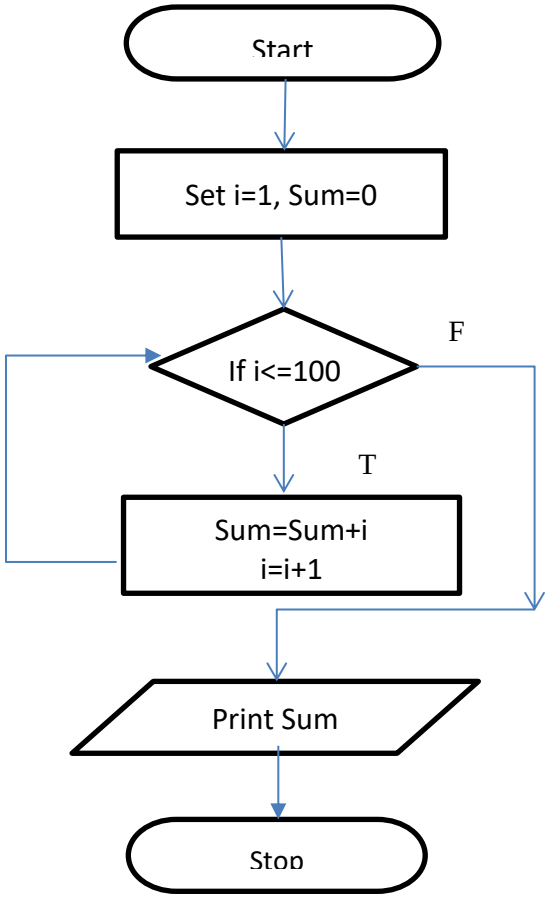
Anna University Solved Problems:

1) Draw a flow chart to accept three distinct numbers, find the greatest and print the result. (8) (Jan-2018)

Flowchart:



2) Draw a flow chart to find the sum of the series 1+2+3+……+100. (8)(Jan-2018)



Outline the algorithm for displaying the first n odd numbers. (May 2019)

Step 1: Start
Step 2: Read value for 'n'
Step 3: Set i=1
Step 4: Repeat Step 5 & 6 until if(i<=n)
Step 5: Print i
Step 6: Calculate i=i+2
Step 7: Stop

The algorithm to find all the prime numbers less than or equal to a given integer n: (10) (Nov / Dec 2019)

Step 1: Start
Step 2: Read value for 'n'
Step 3: Set i=2
Step 4: Repeat Step 5 to 10 until if(i<=n)
Step 5: Set j=2
Step 6: Repeat Step 7 & 8 until if(j<i)
Step 7: if(i%j==0)
    break
Step 8: Calculate j=j+1
Step 9: if(j==i)
    Print i
Step 10: Calculate i=i+1
Step 11: Stop

PART- A (2 Marks)

1. What is an algorithm?(Jan-2018)

Algorithm is an ordered sequence of finite, well defined, unambiguous instructions for completing a task. It is an English-like representation of the logic which is used to solve the problem. It is a step-by-step procedure for solving a task or a problem. The steps must be ordered, unambiguous and finite in number.

2. Write an algorithm to find minimum of three numbers. ALGORITHM : Find Minimum of three numbers

Step 1: Start
Step 2: Read the three numbers A, B, C
Step 3: Compare A,B and A,C. If A is minimum, perform step 4 else perform step 5.
Step 4:Compare B and C. If B is minimum, output "B is minimum" else output "C is minimum".
Step 5: Stop

3. List the building blocks of algorithm.

The building blocks of an algorithm are

- ✓ Statements
- ✓ Sequence
- ✓ Selection or Conditional
- ✓ Repetition or Control flow
- ✓ Functions

An action is one or more instructions that the computer performs in sequential order (from first to last). A decision is making a choice among several actions. A loop is one or more instructions that the computer performs repeatedly.

4. Define statement. List its types.

The instructions in Python, or indeed in any high-level language, are designed as components for algorithmic problem solving, rather than as one-to-one translations of the underlying machine language instruction set of the computer. Three types of high-level programming language statements. Input/output statements make up one type of statement. An input statement collects a specific value from the user for a variable within the program. An output statement writes a message or the value of a program variable to the user's screen.

5. Write the pseudocode to calculate the sum and product of two numbers and display it.

INITIALIZE variables sum, product, number1, number2 of type real
PRINT "Input two numbers"
READ number1, number2
COMPUTE sum = number1 + number2
PRINT "The sum is", sum
COMPUTE product = number1 * number2
PRINT "The Product is", product
END program

6. How does flow of control work?

Control flow (or flow of control) is the order in which individual statements, instructions or function calls of an imperative program are executed or evaluated. A control flow statement is a statement in which execution results in a choice being made as to which of two or more paths to follow.

7. Write the algorithm to calculate the average of three numbers and display it.

Step 1: Start
Step 2: Read values of X,Y,Z
Step 3: S = X+Y+Z
Step 4: A = S/3
Step 5: Write value of A
Step 6: Stop

8. Give the rules for writing Pseudocode.

- ✓ Write one statement per line.
- ✓ Capitalize initial keywords.
- ✓ Indent to show hierarchy.
- ✓ End multiline structure.
- ✓ Keep statements language independent.

9. What is a function?

Functions are named sequence of statements that accomplish a specific task. Functions usually "take in" data, process it, and "return" a result. Once a function is written, it can be used over and over and over again. Functions can be "called" from the inside of other functions.

10. Give the difference between flowchart and pseudocode.

Flowchart	Pseudocode
• A flowchart is a diagram showing an overview of the problem. • It is a pictorial representation of howthe program will work, and it follows a standard format. • It uses different kinds of shapes to signify different processes involved inthe problem	• Pseudocode is a means of expressing the stepwise instructions for solving a problem without worrying about the syntax of a particular programming language. • Unlike a flowchart, it uses a written format which requires no absolute rulesfor writing. • It can be written in ordinary English, andwe can use some keywords in it too.

11. Define a 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.
- The purpose of drawing a flowchart is to make the logic of the program clearer in a visual form.

12. Give an example of iteration.

```
a = 0
for i in range(4):      # i takes the value 0,1,2,3 a = a + i
print(a)                # number 6 is printed (0+0;0+1;1+2;3+3)
```

13. Write down the rules for preparing a flowchart.

While drawing a flowchart, some rules need to be followed:

- A flowchart should have a start and end.
- The direction of flow in a flowchart must be from top to bottom and left to right.
- The relevant symbols must be used while drawing a flowchart.

14. List the categories of Programming languages.

Programming Languages are divided into the following categories:

- Interpreted
- Functional
- Compiled
- Procedural
- Scripting
- Markup
- Logic-Based
- Concurrent
- Object-Oriented Programming Languages.

15. Mention the characteristics of algorithm.

- Algorithm should be precise and unambiguous.
- Instruction in an algorithm should not be repeated infinitely.
- Ensure that the algorithm will ultimately terminate.
- Algorithm should be written in sequence.
- Algorithm should be written in normal English.
- Desired result should be obtained only after the algorithm terminates

16. List out the simple strategies to develop an algorithm. Algorithm development process consists of five major steps.

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 detail.
Step 5: Review the algorithm.

17. Compare machine language, assembly language and high-level language.

Machine Language	Assembly Language	High-Level Language
• The language of 0s and 1s is called as machine language. • The machine language is system independent because there are different set of binary instruction for different types of computer systems	• It is low level programming language in which the sequence of 0s and 1s are replaced by mnemonic (ni-monic) codes. • Typical instruction for addition and subtraction	• High level languages are English like statements and programs . • Written in these languages are needed to be translated into machine language before to their execution using a system software compiler.

18. Describe algorithmic problem solving.

Algorithmic Problem Solving deals with the implementation and application of algorithms to a variety of problems. When solving a problem, choosing the right approach is often the key to arriving at the best solution.

19. Give the difference between recursion and iteration.

Recursion	Iteration
Function calls itself until the base condition is reached.	Repetition of process until the condition fails.
In recursive function, base case (terminate condition) and recursive case are specified.	Iterative approach involves four steps: initialization, condition, execution and updation.
Recursion is slower than iteration due to overhead of maintaining stack.	Iteration is faster.
Recursion takes more memory than iteration due to overhead of maintaining stack.	Iteration takes less memory.

20. What are advantages and disadvantages of recursion?

Advantages	Disadvantages
Recursive functions make the code look clean and elegant.	Sometimes the logic behind recursion is hard to follow through.
A complex task can be broken down into simpler sub-problems using recursion.	Recursive calls are expensive (inefficient) as they take up a lot of memory and time.
Sequence generation is easier with recursion than using some nested iteration.	Recursive functions are hard to debug.

21. Write an algorithm to accept two numbers, compute the sum and print the result.(Jan-2018)

Step 1: Read the two numbers a and b.
Step 2: Calculate sum = a+b
Step 3: Display the sum

22. Write an algorithm to find the sum of digits of a number and display it.

Step 1:Start
Step 2: Read value of N
Step 3: Sum = 0
Step 4: While (N != 0)
Rem = N % 10
Sum = Sum + Rem N = N / 10
Step 5: Print Sum
Step 6: Stop

23. Write an algorithm to find the square and cube and display it.

Step 1: Start
Step 2: Read value of N
Step 3: S =N*N
Step 4: C =S*N
Step 5: Write values of S,C
Step 6: Stop

24. Explain Tower of Hanoi.

The Tower of Hanoi is a mathematical game. It consists of three rods and a number of disks of different sizes, which can slide onto any rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a conical shape.
The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

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

With 3 disks, the puzzle can be solved in 7 moves. The minimal number of moves required to solve a Tower of Hanoi puzzle is $2^n - 1$, where n is the number of disks.

25. What is recursion?

Recursion is a method of solving problems that involves breaking a problem down into smaller and smaller subproblems until you get to a small enough problem that it can be solved trivially. Usually recursion involves a function calling itself. While it may not seem like much on the surface, recursion allows us to write elegant solutions to problems that may otherwise be very difficult to program.
Example:

```
def calc_factorial(x):
    if x == 1:
    return 1
    else:
    return (x * calc_factorial(x-1))
    num = 4
    print("The factorial of", num, "is", calc_factorial(num))
```

26. Write an algorithm to find minimum in a list. (Jan-2019)

ALGORITHM : To find minimum in a list

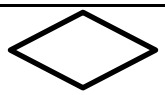
Step 1: Start
Step 2: Read the list
Step 3: Assume the first element as minimum
Step 4: Compare every element with minimum. If the value is less than minimum, reassign that value as minimum.
Step 5: Print the value of minimum.
Step 6: Stop

27. Distinguish between algorithm and program. (Jan-2019)

Algorithm	Program
Algorithm is the approach / idea to solve someproblem.	A program is a set of instructions for the computerto follow.
It does not have a specific syntax like any of theprogramming languages	It is exact code written for problem following all therules (syntax) of the programming language.
It cannot be executed on a computer.	It can be executed on a computer

28. List the Symbols used in drawing the flowcart. (May 2019)

Flowcharts are usually drawn using some standard symbols

Symbol	Purpose	Description
	Flow line	Used to indicate the flow of logic by connectingsymbols.
	Terminal(Stop / Start)	Used to represent start and end of flowchart.
	Input / Output	Used for input and output operation.
	Processing	Used for arithmetic operations and data-manipulations.
	Decision	Used to represent the operation in which there aretwo alternatives, true and false.
	On-page Connector	Used to join different flow line.
	Off-page Connector	Used to connect flowchart portion on differentpage.
	Function / Predefined Process	Used to represent a group of statements performing one processing task.

29. Give the python code to find the minimum among the list of 10 numbers. (May 2019)

```
numList = []
n=int(raw_input('Enter The Number of Elements in List :'))
for i in range(0, n):
        x = raw_input('Enter the Element %d :' %(i+1))
        numList.append(x)
maxNum = numList[0]
for i in numList:
        if i > maxNum:
                maxNum = i
print('Maximum Element of the Given List is %d' %(int(maxNum)))
```

30. How will you analysis the efficiency of an algorithm? (Nov / Dec 2019)

Time efficiency, indicating how fast the algorithm runs. Space efficiency, indicating how much extra memory it uses

31. How do algorithm, flowchart and pseudo code use for problem solving? (Nov / Dec 2019)

Algorithm: A process or set of rules to be followed in calculations or other problem – solving operations, especially by a computer

Flowchart: Flowchart is diagrammatic representation of the algorithm.

Pseudo code: In Pseudo code normal English language is translated into the programming languages to be worked on.

All are tools for problem solving independent of programming language. Difference is only in the way of representing the solution.

UNIT 2

DATA TYPE, EXPRESSION, STATEMENT

Python interpreter and interactive mode,debugging; values and types: int, float, boolean, strings, and lists; variables, expressions, statements, tuple assignment, precedence of operators, comments; Illustrative programs: exchange the values of two variables, circulate the values of n variables, distance between two points.

2.1 PYTHON INTERPRETER

2.1.1 What is Python?

Python is a very popular general purpose, interpreted, object oriented, and high level programming language.

Python is dynamically-typed, and garbage collected programming language. It is developed by ”Guido van rossum” during 1985-1990.

2.1.2 Characteristics of Python:

- It supports functional and structured programming method as well as oop**.**
- It can be used as a scripting language or can be compiled to byte code for building large application.
- It provides very high level dynamic data types and supports dynamic type checking.
- It support automatic garbage collection.
- It can be easily integrated with c, c++, CORBA, and Java.

2.1.3 Features of python:

- Easy to learn
- Easy to maintain
- Interactive mode
- Extensible
- Portable
- Free and open source
- Embeddable
- Object oriented
- Scalable

2.1.4 Application of python:

- Web Development
- Machine Learning and Artificial Intelligence
- Data Science and Data Visualization
- Web Scrapping Application
- Audio and Video Application

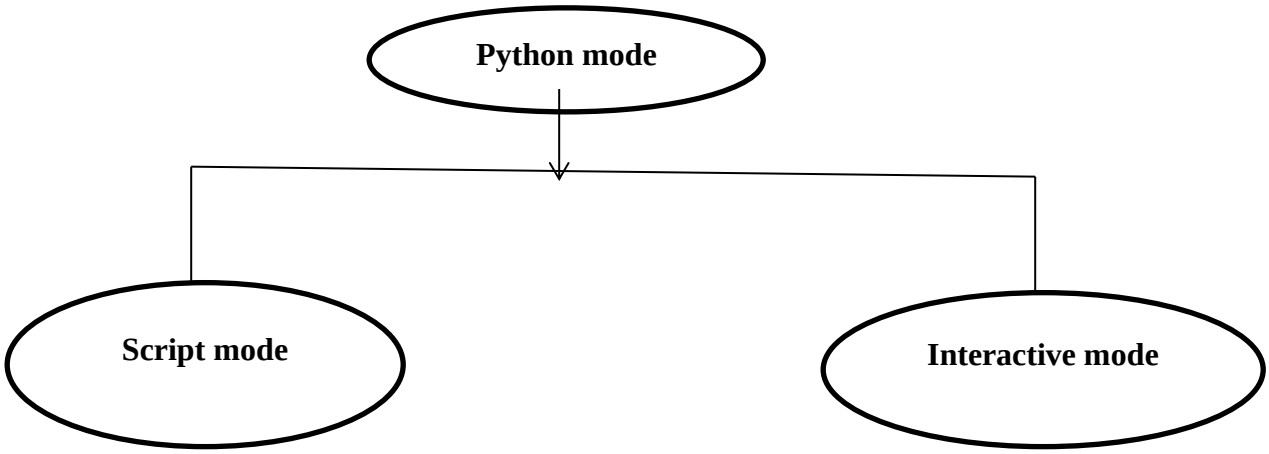
2.1.5 Python Interpreter:

- The python interpreter is a program that reads and executes python code. Even though most of today's Linux and mac have python pre-installed in it, the version might be out of date. So, it always a good idea to install the most current version.
- After installation, the python interpreter lives in the installed directory.By default it is

  "/usr/local/bin/python x.x" –Linux/unix

  "c:\python x.x"- windows

  Where 'x' denotes the version number.
- Now there are two type ways to start python



2.1.5.1 Interactive mode:

Typing python in the command line will invoke the interpreter in interactive mode. When it starts, you should see output like this:

"python 2.7.13(v2.7.13: A06454B1AFA1, DEC 17 2016, 20:42:59)

[MS [v.1500 32 bit (INTEL) on WIN 32

Type "COPYRIGHTS" , "CREDITS" OR "LICENSE()" For more INFORMATION]"

The first 3 line contains information about the interpreter and the operating system its running on, so it might be different for you. But you should check that the version number, which is 2.7.13 in the example, begins with 2, which indicates that you are running python 2. If it begins with 3, you are running python 3.

The last line is a prompt indicate that the interpreter is ready for you to enter code. If you type a line of code and hit enter, the interpreter display the results.

For Example:

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

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

2.1.5.2 Script mode:

This mode is used to execute python program written in a file. Such a file called a Script. Script can be saved for future use. Python scripts have the extension .py, meaning the filename end with .py.

For Example:

Helloworld.py

To execute this file in script mode we simply write python helloworld.py at the command prompt.

2.1.5.3 IDLE (Integrated Development and Learning Environment) :

IDLE is a graphical user interface (GUI) that can be installed along with the Python programming language and is available from the official website.

We can also use other commercial or free IDLE according to our preference .pyscript IDLE is one of the source IDLE.

For Example:

Now that we have python up and running, we can continue on to write our first python program.

Type the following code in any text editor or an IDLE and save it as **"helloworld.py"**

```
Print("helloworld!")
```

Now at the command window, go to location of the file: you can use the **cd** command to change directory.

To run the script, type **python helloworld.py** in the command windows. We should be able to see the output as follow:

```
helloworld!
```

If you are using pyscripter, there is a green arrow button on top. Press that button or press Ctrl+F9 on your keyboard to run the program.

2.1.6 Debugging

Programming is error-prone, for unusual reasons. Programming errors are called bugs and the process of tracking them down is called Debugging.

Three kinds of errors can occurs in a program,

- Syntax Error
- Run Time Error
- Semantic Error

2.1.6.1 Syntax Error:

Python can only execute a program if the syntax is correct; Otherwise the interpreter display an error message. Syntax refers to the structure of a program and the rules about that structure.

For Example:

Parentheses have to come in matching pair, so (2+2) is legal, but 2) is a syntax error.

Python is not so forgiving. If there is a single syntax error anywhere in your program, python will display an error message and quit, and you will not be able to run your program.

2.1.6.2 Run time Error:

The Second type of error is a runtime error, so called because the error does not appear until after the program has started Running. These errors are also called exceptional because they usually indicate that something exceptional has happened.

For Example:

Division by zero
Performing an operation on incompatible types.

2.1.6.3 Semantics Error:

The third type of error is the semantic error. If there is a semantic error in your program, it will run successfully in the sense that the computer will not generate any error message, but it will not do the right things. It will do something else. Specifically it will not do what you told it to do.

The problem is that the program you wrote is not the program you wanted to write. The meaning of the program semantics is wrong. Identifying semantics errors can be tricky because it requires you to work backward by looking at the output of the program and trying to figure out what it is doing.

For Example:

```
n= int(input("enter a number:"))
sum=0
sum+n=sum
```

In the above example, we can see a syntax error but actually it is a semantic error, because the statement:

Sum+n=sum doesn't make any sense. This is because an expression cannot come on the left side of an assignment according to python language rules.

2.2 VALUES AND TYPES

A value is one of the basic things a program works with, like a letter or a number. Some example values are,

5 is an integer

83.0 is an floating point number

"Helloworld" is an string

For Example:

```
>>> type(5)
  <class 'int'>

>>> type(83.0)
   <class 'float'>

>>>type('helloworld!')
      <class 'str'>
```

What about values like '5','83.0'? They look like numbers, but they are in quotation marks like strings.

```
>>> type('5')
 <class'str'>

>>> type('83.0')
 <class 'str'>
```

They are string.

2.2.1 Standard data types

Data types:

Every value in python has a data types. Since everything is an object in python programming, data types are actually classes and variable are instances (object) of these classes. Python has variable standard data types that are used to define operation possible on them and the storage method for each of them.

Python has five standard data types

- Numbers.
- Strings.
- List.
- Tuple.
- Dictionary.

2.2.1.1 Numbers:

Numbers data types stores numeric values. Number object are created when you assign a value to them.

Integer-> 1, 3, 42…..

Example:

```
>>>2+3
   5
>>> 3-5
   -2
>>> 3*4/2**2
    3.0
```

Floats ->3.0, 31e12, -6e-4…..

```
>>> 2.5/2
   1.25
```

Complex number -> 3+2j, -4-2j, 4.2+6.3j…..

```
>>> (3+2j)+(3+4j)
    (6+6j)
```

Boolean -> True, False.

```
>>> a=True
>>> not a
    False
```

2.2.1.2 Strings:

Strings in python are identified as a contiguous set of character represented in the quotation marks “”,’’. Strings are immutable.

Python allows for either pairs of single or double quotes. Subset of strings can be taken using the slice operator **([] and[:])** with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.

\+ -> string concatenation.

* -> repetition operator.

For Example:

```
Str=’python programming’
Print(str)
Print(str[0])
Print(str[-1])
Print(str[2:5])
Print(str[2:])
Print(str*2)
Print(str+”course”)
```

Output:

```
python programming
p
g
tho
thon  programming
python programming python programming
python programming course.
```

2.2.1.3 Lists:

Lists are the most versatile of python’s compound data types. A list contains item separated by commas and enclosed within square brackets [].

To some extent, lists are similar to arrays in c. But all the items belonging to a list can be of different data types.

The values stored in a list can be accessed using the slice operator ([] and[:]) with indexes starting at 0 in the beginning of list and working their way to end -1.

For Example:

```
list=[‘hai’, 123, 1.75, ‘vinu’, 100.25]
```

```
smalllist=[251, 'vinu']
        Print(list)
        Print(list[0])
        Print(list[-1])
        Print(list[2:5])
        Print(list[2:])
        Print(smalllist*2)
        Print(list+smalllist)
```

OUTPUT:

```
['hai', 123, 1.75, 'vinu', 100.25]
 hai
100.25
[123, 1.75]
[251, 'vinu', 251, 'vinu']
['hai', 123, 1.75, 'vinu', 100.25, 251, 'vinu']
```

2.2.1.4 Data type Conversion:

Sometimes, you may need to perform conversions between the built in types. To convert between types, you may simply use the type name as a function.

There are several built in function to perform conversion from one data type to another.

Function	Description
int(x[,base])	convert x to an integer, base specifies the base if x is a string.
long(x[,base])	convert x to an long integer, base specifies the base if x is a string.
float(x)	convert x to an floating point.
complex(real[,img])	creates a complex number.
str(x)	convert object x to a string representation.
repr(x)	convert object x to a expression string .
eval(str)	evaluate a string and returns an objects.
list(s)	convert to a list.
chr(x)	convert an integer to a character.
unichr(x)	convert an integer to a Unicode character.
ord(x)	convert an single character to its integer values.
hex(x)	convert an integer to hexadecimal values..
oct(x)	convert an integer to octal string.

2.3 VARIABLE

A variable is a name that refers to a value. Variable reserved memory locations to stores values. This means that when you create a variable you reserved some space in memory. Based on the data types of a variable, the interpreter allocate memory and decides what can be stored in the reserved memory.

2.3.1 Rules for variable names:

- Name should be meaningful.
- Variable name can be arbitrarily long.
- They can contain both letter and numbers, but they have to begin with a letter.
- It is legal to use uppercase letter, but it is a good idea to begin variable name with a lower case letter.
- Names are case sensitive.
- Variable name must start with letter or an underscore(_).
- Special Character cannot be used except underscore(_).
- White space are not allowed in between a variable name.

For Example:

```
>>> n=17
>>> message ='python programming'
```

2.3.2 Python Identifier:

Identifier is the meaningful variable name given to entities like class, function, variable, etc…..in python. It helps differentiating one entity from another.

2.3.3.1 Rules for writing identifiers:

- Identifier can be combination of letter in lowercase(a to z), or uppercase(A to Z) or digits(0 to 9) or an underscore(_).
  Ex: myclass, var_1.

- An identifier cannot start with a digits.
  Ex: 1variable is invalid
  Variable1 is valid.

- Keyword cannot be used as identifiers.
  Ex: >>> global=1
  Syntax error: invalid syntax.

- We cannot use special symbol like !,@,#,$,%,etc……in our identifiers.
  Ex:>>>a@=0
  Syntax error: invalid syntax.

- Identifier can be of any length.

2.3.3 Variable Assignment:

An assignment statement creates a new variable and gives it a value. We use the assignment (=) to assign values to a variable. Any type of value can be assigned to any valid variable.

For Example:

```
a=5
b= 3.2
c=”hello”
```

2.3.5 Multiple assignment (or) Tuple Assignment:

In python, multiple assignments can be made in a single statement as follows.

For Example:

```
a,b,c= 5,3.2,”hello”
x=y=z=”same”      assign ‘same’ value to all variable.
```

2.3.4 Python Keyword:

Keywords are the reserved word in python. We cannot use a keyword as a variable name, function name or any other identifier. In python, keywords are case sensitive. There are 33 keywords in python 3.3. All the keyword except true, false and none are in lower case and they must be written as it is.

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

2.4 EXPRESSION

An expression is a meaningful combination of operator and operands. In any programming language, an expression is evaluated based on precedence of its operator.

2.4.1 Types of Expression:

- Constant expression
- Arithmetic expression
- Integral expression
- Floating expression
- Relational expression
- Logical expression
- Bitwise expression
- Combinational expression

2.4.1.1 Constant Expression

Expressions with only constant inputs are called constant expressions.

```
E: x=15+1.3
      Print(x)
OUTPUT: 16.3
```

2.4.1.2 Arithmetic Expression

An arithmetic expression is a combination of number, operator, and sometimes parenthesis. The result of expression is also a numeric value.

Operator	Description	Syntax
+	Addition: adds two operands	x + y
–	Subtraction: subtracts two operands	x – y
*	Multiplication: multiplies two operands	x * y
/	Division (float): divides the first operand by the second	x / y
//	Division (floor): divides the first operand by the second	x // y
%	Modulus: returns the remainder when the first operand is divided by the second	x % y
**	Power: Returns first raised to power second	x ** y

Example:

```
x=40
Y=12
Add=x+y
Sub=x-y
Pro=x*y
Div=x/y
Print(add)
Print(sub)
Print(pro)
Print(div)
```

```
Output:
52
28
480
3.3333
```

2.4.1.3 Integral Expression:

These are the kind of expression that produce only integer results after all computation and type conversion.

EX: a=13 OUTPUT: 25
b= 12.0
c=a+int(b)
Print(c)

2.4.1.4 Floating Expression:

These are the kind of expression that produces only floating points results after all computation and type conversion**.**

EX: a= 13 OUTPUT: 2.6
b=5
c=a/b
print(c)

2.4.1.5 Relational Expression :

In these type of expression, arithmetic expression are written on both sides of relational operator (>, <. >=, <=). Those arithmetic expression are evaluated first, and then compared as per relational operator and produce a Boolean output at end.

EX: a=21 OUTPUT: true
b=13
c= 40
d= 37
p=(a+b)>=(c-d)
print(p)

2.4.1.6 Logical Expression:

These are kinds of expression that results in either true or false. It basically one or more condition.

Operator	Description	Syntax
and	Logical AND: True if both the operands are true	x and y
or	Logical OR: True if either of the operands is true	x or y
not	Logical NOT: True if the operand is false	not x

Ex: P=(10==9)
Q=(7>5)
R= P and Q
S= P or Q
T= not P
print(R)
print (S)
print(T)

Output:

False

True

True

2.4.1.7 Bitwise expression:

These are the kind of expression in which computations are performed at bit level.

```
EX: a=12                                    OUTPUT: 3 24
    X=a>>2
    Y=a<<1
   Print(x, y)
```

2.4.1.8 Combinational Expression:

We can also use different types of expression in a single expression, and that will be terminated as combinational expression.

```
Ex: a=16                                    OUTPUT: 22
    b=12
    C= a+(b>>1)
   Print(c)
```

2.5 STATEMENT

A statement is a unit of code that the python interpreter can execute. We have seen two kind of statement: print and assignment.

Instruction that a python interpreter can execute are called statement. For example, a=1 is an assignment statement. If statement, while statement, for statement etc…….

2.5.1 Multi line statement:

In python, end of a statement is marked by a new-line character. But we can make a statement extend over multiple line, with the line continuation character(\).

```
EX: a= 1+2+3+\
       4+5+6+\
```

This is explicit line continuation. In python, line continuation is implied inside parentheses (), bracket [] and brace {}.

```
EX: a = (1+2+3+4+5
         +6+7+8+9)
EX: colors =['red' ,
             'blue' ,
            'green'].
```

We could also put multiple statements in a single line using semicolon.

```
EX: a=1; b=2; c=3.
```

2.5.2 Simple Statements:

Python simple statement is comprised of a single line. The multiline statements created above are also simple statements because they can be written in a single line. Let's look at some important types of simple statements in Python.

1. Python Expression Statement

```
i = int("10")  # expression is evaluated and the result is assigned to the variable.
sum = 1 + 2 + 3  # statement contains an expression to be evaluated first.
```

2. Python Assignment Statement

```
count = 10  # value is assigned to the variable, no expression is evaluated
message = "Hi"
```

3. Python pass Statement

```
def disp():
   pass  # pass statement
```

4. Python del Statement

```
name = "Python"
del name  # del statement
```

5. Python return Statement

```
def disp():
   return 10  # return statement
```

6. Python break Statement

```
numbers = [1, 2, 3]
for num in numbers:
   if num > 2:
      break  # break statement
```

7. Python continue Statement

```
numbers = [1, 2, 3]
for num in numbers:
   if num > 2:
      continue  # continue statement
   print(num)
```

2.5.3. Python Compound Statements:

Python compound statements contain a group of other statements and affect their execution. The compound statement generally spans multiple lines. Let's briefly look into a few compound statements.

1. Python if Statement

```
if 5 < 10:
   print("This will always print")
else:
   print("Unreachable Code")
```

2. Python for Statement

```
for n in (1, 2, 3):
   print(n)
```

3. Python while Statement

```
count = 5
while count > 0:
   print(count)
   count -= 1
```

4. Python try Statement

```
try:
   print("try")
except ValueError as ve:
   print(ve)
```

5. Python Function Definition Statement

A python function definition is an executable statement. Its execution binds the function name in the current local namespace to a function object. The function is executed only when it's called.

```
def disp():
    pass
```

2.5.4. Input and Output Statements:

Output

Python output using print () function. We use the print () function to output data to the standard output device (screen).

Syntax:

```
print(*objects, sep='',end='\n', file=sys.stdout, flush=false)
```

Ex:

```
print(1,2,3,4)
#output: 1 2 3 4
print(1, 2, 3, 4, sep='*')
# output: 1*2*3*4
print(1, 2, 3, 4, sep='#', end='&')
#output: 1#2#3#4&
```

Input:

Python input function takes a single parameter that is a string. This string is often called the prompt because it contains some helpful text prompting the user to enter something.

Syntax : input([prompt])

Ex:

```
name = (input('enter your name:'))
course = (input('enter your course:'))
m1= float (input ('enter your mark in sub1 :'))
m2= float (input ('enter your mark in sub2 :'))
m3= float (input ('enter your mark in sub3 :'))
total = m1+m2+m3
avg= total/3
print('your total mark:', total)
print('your average mark:', avg)
```

Output:

Enter your name: muni

Enter your course: BE

Enter your mark in sub1: 87.5

Enter your mark in sub2: 98.5

Enter your mark in sub3: 79.5

2.6 TUPLE ASSIGNMENT

Python has a very powerful tuple assignment features that allows a tuple of variable on the left of an assignment to be assigned value from a tuple on the right of the assignment.

The left side is a tuple of variable; the right side is a tuple of values. Each value is assigned to its respective variable. The number of variables on the left and the right of the assignment operator must be same.

All the expression on the right side are evaluated before any of the assignment. This features make tuple assignment quite versatile.

For Example:

```
>>> (a, b, c)=(1, 2, 3)
```

 # Program to swap a and b (without using third variable)

```
a=2; b=3
print(a, b)
a,b=b,a
print(a, b)
```

OUTPUT:

(2, 3)

(3, 2)

- **One way to think of tuple assignment is as tuple packing/unpacking:**

In tuple packing, the value on the left are 'packed' together in a tuple:

```
>>> b= ("George",25,"20000") #tuple packing
>>> (name, age, salary) = b #tuple unpacking
>>> name
    'George'
>>> age
     25
```

2.7 PYTHON OPERATORS PRECEDENCE RULE - PEMDAS

Precedence of Arithmetic Operators:

Operator precedence in python follows the PEMDAS rule for arithmetic expressions. The precedence of operators is listed below in a high to low manner.

Firstly, parentheses will be evaluated, then exponentiation and so on.

- **P** – Parentheses
- **E** – Exponentiation
- **M** – Multiplication
- **D** – Division
- **A** – Addition
- **S** – Subtraction

Example:

- Parentheses have the highest precedence and can be used to force an expression to evaluate in the order you want. Since expression in parenthesis are evaluated first,

  2*(3-1) = 4, and (1+1)**(5-2) is 8.

- Exponentiation has the next highest precedence, so

  1+2**3=9 and not 27. And 2*3**2 is 18 and not 36.

- Multiplication and Division have higher precedence than Addition and Subtraction. So,

  2*3-1 is 5 and not 4. And 6+4/2 is 8 not 5

In the case of tie means, if two operators whose precedence is equal appear in the expression, then the associativity rule is followed.

2.8.1 Associativity Rule

All the operators, except exponentiation (**) follow the left to right associativity. It means the evaluation will proceed from left to right, while evaluating the expression.

Example- (43 + 13 - 9 / 3 * 7)(43+13−9/3∗7)

In this case, the precedence of multiplication and division is equal, but further, they will be evaluated according to the left to right associativity.

Let's try to solve this expression by breaking it out and applying the precedence and associativity rule.

- Interpreter encounters the parenthesis (. Hence it will be evaluated first.
- Later there are four operators ++, -−, ** and //.
- Precedence of (/,(/, and *) >∗) > Precedence of (+, -)(+,−).
- This expression will be evaluated from left to right, 9 / 3 * 79/3∗7 = 3 * 73∗7 = 553.
- Now, our expression has become 43+13-2143+13−21. Left to right Associativity rule will be followed again. So, our final value of expression will be, 43+13-2143+13−21 = -2095.

Examples

Below are two examples to illustrate the operator precedence in python. See the explanation to get a good idea of how these things work internally.

2.8.2 Precedence of Operators

```
a = (10 + 12 * 3 % 34 / 8)
b = (4 ^ 2 << 3 + 48 // 24)
print (a)
print (b)
```

Output:
10.25
68

Precedence of Operators

Operator	Description
**	The exponent operator is given priority over all the others used in the expression.
~ + -	The negation, unary plus, and minus.
* / % //	The multiplication, divide, modules, reminder, and floor division.
+ -	Binary plus, and minus
>> <<	Left shift. and right shift
&	Binary and.
^ |	Binary xor, and or
<= < > >=	Comparison operators (less than, less than equal to, greater than, greater then equal to).
<> == !=	Equality operators.
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

2.8.3 Conclusion

- The operator works on operands according to their specific order while evaluating the expression. This hierarchy is called operator precedence in python.
- In case of the same precedence, the left to right associativity is followed. But the exponentiation operator is an exception that follows right to left associativity.

2.8 Comments

Comments in Python are the lines in the code that are ignored by the interpreter during the execution of the program. Comments enhance the readability of the code and help the programmers to understand the code very carefully. There are three types of comments in Python .

- Single line Comments
- Multiline Comments
- Docstring Comments

Advantages of Comments:

- Code Readability
- Explanation of the code or Metadata of the project
- Prevent execution of code
- To include resources

2.9.1 Types of Comments in Python

There are three main kinds of comments in Python. They are:

Single-Line Comments

Python single-line comment starts with the hash tag symbol (#) with no white spaces and lasts till the end of the line. If the comment exceeds one line then put a hash tag on the next line and continue the comment. Python's single-line comments are proved useful for supplying short explanations for variables, function declarations, and expressions. See the following code snippet demonstrating single line comment:

Example:

```
>>>a="Welcome"     #It is a string
```

Multi-Line Comments

Python provides the option for multiline comments.

```
>>>"""This is

        My First Program"""
```

Python Docstring:

Python docstring is the string literals with triple quotes that are appeared right after the function. It is used to associate documentation that has been written with Python modules, functions, classes, and methods. It is added right below the functions, modules, or classes to describe what they do. In Python, the docstring is then made available via the __doc__ attribute.

Example:

```
def multiply(a, b):
        """Multiplies the value of a and b"""
        return a*b
# Print the docstring of multiply function
print(multiply.__doc__)
```

Output:

Multiplies the value of a and b

2.9 ILLUSTRATIVE PROBLEM

1. Exchange the values of two variables:

```
# To take input from the user
# x = input ('Enter value of x: ')
# y = input ('Enter value of y: ')
  x = 5
  y = 10
# create a temporary variable and swap the values
  temp = x
  x = y
  y = temp
  print(x,y)
  print('The value of x after swapping: {}'.format(x))
  print('The value of y after swapping: {}'.format(y))
```

2. Program to find distance between two points

```
Import math p1=[2,4]
p2=[3,6]
distance=math.sqrt(((p2[0]-p1[0])**2)+((p2[1]-p1[1])**2))
print("Distance between two  points: %.2f ", % distance)
```

Output:

Distance between two points: 2.24

3. Circulate the values of n variables

```
def rotate (list, n):
        new=list[n:]+list[:n]
        return new
example=[1,2,3,4,5]
print("Original list:", example)
a=rotate(example,1)
print("List rotated clockwise by 1:",a)
a=rotate(example,2)
print("List rotated clockwise by 2:",a)
a=rotate(example,-2)
print("List rotated counter clockwise by 2:",a)
```

Output:

```
Original list: [1, 2, 3, 4, 5]
List rotated clockwise by 1: [2, 3, 4, 5, 1]
List rotated clockwise by 2: [3, 4, 5, 1, 2]
List rotated counter clockwise by 2: [4, 5, 1, 2, 3]
```

2 Mark Questions with Answers

1. What is meant by interpreter?

An interpreter is a computer program that executes instructions written in a programming language. It can either execute the source code directly or translate the source code in a first step into a more efficient representation and executes this code.

2. How will you invoke the python interactive interpreter?

The Python interpreter can be invoked by typing the command "python" without any parameter followed by the "return" key at the shell prompt.

3. What are the commands that are used to exit from the python interpreter in UNIX and windows?

CTRL+D is used to exit from the python interpreter in UNIX and CTRL+Z is used to exit from the python interpreter in windows.

4. Define a variable and write down the rules for naming a variable.

A name that refers to a value is a variable . Variable names can be arbitrarily long. They can contain both letters and numbers, but they have to begin with a letter. It is legal to use uppercase letters, but it is good to begin variable names with a lowercase letter.

5. Write a snippet to display "Hello World" in python interpreter.

In script mode:
>>>print("Hello World") Hello World
In Interactive Mode:
>>> "Hello World" 'Hello World'

6. List down the basic data types in Python.

- Numbers
- String
- List
- Tuple
- Dictionary

7. Define keyword and enumerate some of the keywords in Python. (Jan-2019)

A keyword is a reserved word that is used by the compiler to parse a program. Keywords cannot be used as variable names. Some of the keywords used in python are:

- and
- del
- from
- not
- while
- is
- continue

8. What do you mean by an operand and an operator? Illustrate your answer with relevant example.

An operator is a symbol that specifies an operation to be performed on the operands. The data items that an operator acts upon are called operands. The operators +, -, *, / and ** perform addition, subtraction, multiplication, division and exponentiation.

Example: 20+32.In this example, 20 and 32 are operands and + is an operator.

9. Explain the concept of floor division.

The operation that divides two numbers and chops off the fraction part is known as floor division.

Example:>>> 5//2= 2

10. Define an expression with example.

An expression is a combination of values, variables, and operators. An expression is evaluated using assignment operator. Example:Y= X + 17

11. Define statement and mention the difference between statement and an expression.

A statement is a unit of code that the Python interpreter can execute. The important difference is that an expression has a value but a statement does not have a value.

12. What is meant by rule of precedence? Give the order of precedence.

The set of rules that govern the order in which expressions involving multiple operators and operands are evaluated is known as rule of precedence. Parentheses have the highest precedence followed by exponentiation. Multiplication and division have the next highest precedence followed by addition and subtraction.

13. Illustrate the use of * and + operators in string with example.

The * operator performs repetition on strings and the + operator performs concatenation on strings.

Example:>>> ‘Hello*3’

Output:HelloHelloHello

>>>’Hello+World’

Output:HelloWorld

14. What is function call?

A function is a named sequence of statements that performs a computation. When you define a function, you specify the name and the sequence of statements. Later, you can “call” the function by name is called function call.

Example:

sum() //sum is the function name

15. What is a local variable?

A variable defined inside a function. A local variable can only be used inside its function.

Example:

deff():

s = "Me too." // local variable print(s)

a = "I hate spam." f()

print (a)

16. Define arguments and parameter.

A value provided to a function when the function is called. This value is assigned to the corresponding parameter in the function. Inside the function, the arguments are assigned to variables called parameters.

17. What do you mean by flow of execution?

- In order to ensure that a function is defined before its first use, you have to know the order in which statements are executed, which is called the flow of execution.
- Execution always begins at the first statement of the program. Statements are executed one at a time, in order from top to bottom.

18. What is the use of parentheses?

Parentheses have the highest precedence and can be used to force an expression to evaluate in the order you want. It also makes an expression easier to read.
Example: 2 + (3*4) * 7

19. What do you meant by an assignment statement?

An assignment statement creates new variables and gives them values:
>>> Message = 'And now for something completely different'
>>> n = 17
This example makes two assignments. The first assigns a string to a new variable namedMessage; the second gives the integer 17 to n.

20. What is tuple? (or) What is a tuple? How literals of type tuples are written? Give example(Jan-2018)

A tuple is a sequence of immutable Python objects. Tuples are sequences, like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets. Creating a tuple is as simple as putting different comma-separated values. Comma-separated values between parentheses can also be used.
Example: tup1 = ('physics', 'chemistry', 1997, 2000); tup2= ();

21. Name the four types of scalar objects Python has. (Jan-2018)

- int
- float
- bool
- None

22. List down the different types of operator.

Python language supports the following types of operators:

- Arithmetic operator
- Relational operator
- Assignment operator
- Logical operator
- Bitwise operator
- Membership operator
- Identity operator

23. What is a global variable?

Global variables are the one that are defined and declared outside a function and we need to use them inside a function.
Example:#This function uses global variable s def f():
print(s) # Global scope
s = "I love India" f()

24. Define function.

A function in Python is defined by a def statement. The general syntax looks like this:

The parameter list consists of zero or more parameters. Parameters are called arguments, if the function is called. The function body consists of indented statements. The function body gets executed every time the function is called. Parameter can be mandatory or optional. The optional parameters (zero or more) must follow the mandatory parameters.

25. What is the purpose of using comment in python program?

Comments indicate information in a program that is meant for other programmers (or anyone reading the source code) and has no effect on the execution of the program. In Python, we use the hash (#) symbol to start writing a comment.

Example: #This is a comment

26. Outline the logic to swap the content of two identifiers without using third variable. (May 2019)

```
a=10 b=20
a,b=b,a
print("After Swapping a="a," b=",b)
```

27. State about Logical operators available in python language with example. (May 2019)

Logical operators are the and, or, not operators.

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 (complements the operand)	not x

28. Compare Interpreter and Compiler (Nov / Dec 2019)

Compiler: A Compiler is a program which translates the source code written in a high level language in to 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 message on the screen.

Interpreter: Interpreter translates the high level language program 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. When an error is found the execution of the program is halted and error message is displayed on the screen

29. Write a python program to circulate the values of n variables (Nov / Dec 2019)

```
no_of_terms = int(input("Enter number of values : "))
list1 = []
for val in range(0,no_of_terms,1):
        ele = int(input("Enter integer : "))
        list1.append(ele)
print("Circulating the elements of list ", list1)
for val in range(0,no_of_terms,1):
        ele = list1.pop(0)
        list1.append(ele)
        print(list1)
```

UNIT III CONTROL FLOW, FUNCTIONS, STRINGS

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

BOOLEAN VALUES

- The Boolean is a data type that has one of two possible values.
  - True
  - False

- The following values are considered false:
  - None
  - False
  - zero of any numeric type, for example, 0, 0L, 0.0, 0j.
  - any empty sequence, for example, '', (), [].
  - any empty mapping, for example, {}.

- All other values are considered as true — so objects of many types are always true.

- We can also check the data type of True or False with the help of 'type' function.

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

- The Relational Operators always results, Boolean value True(1) or False(0).

```
Ex >>> 3==3
                True
   >>>3==5
                False
```

The other relational operators 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
```

OPERATORS IN PYTHON WITH SUITABLE EXAMPLES [Dec/Jan 2018]

- Operators are a special symbol that specifies an operation to be performed on the operands. Operators in programming languages are taken from mathematics.
- An operator may have one or two operands. Operand is input to operator.
- Those operators that work with **only one operand** are called **unary operators**.
- Those who work with **two operands** are called **binary operators**.
- Ex for Binary operator, a+b, a and b are operands,+ operator.
- Ex for Unary operator, -b, here, b is operand, - is operator.

Types of operators:

1. Arithmetic Operators **(+ ,-,*,/,%,//,**)**
2. Comparison (Relational) Operators **(>,<,==,!=,>=,<=)**
3. Logical Operators **(and,or,not)**
4. Assignment Operators **(=,+=,-=,*=,/=,%=,**=)**
5. Bitwise Operators **(&, |, ^, ~)**
6. Membership Operators **(in ,not in)**
7. Identity Operators **(is, is not)**

1. Arithmetic Operators

- Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication etc. (+ ,-,*,/, %, //(***floor division***), **(***exponentiation***)).

Operator	Description	Syntax
+	Addition: adds two operands	x + y
–	Subtraction: subtracts two operands	x – y
*	Multiplication: multiplies two operands	x * y
/	Division (float): divides the first operand by the second	x / y
//	Division (floor): divides the first operand by the second	x // y
%	Modulus: returns the remainder when the first operand is divided by the second	x % y
**	Power: Returns first raised to power second	x ** y

Example:

```
x = 15
y = 4
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)
print(" x ** y = ", x**y)
```

OUTPUT:

x + y = 19
x - y = 11
x * y = 60
x / y = 3.75
x%y = 3
x // y = 3
x ** y = 50625

2. Comparison operators

- Comparison operators are used to compare two or more operands. These are also known as Relational Operators.
- Relational operators are used in decision making process.
- It either returns True or False according to the condition.
- The comparison operators are **>,<,==,!=,>=,<=.**

Operator	Description	Syntax
>	Greater than: True if the left operand is greater than the right	x > y
<	Less than: True if the left operand is less than the right	x < y
==	Equal to: True if both operands are equal	x == y
!=	Not equal to – True if operands are not equal	x != y
>=	Greater than or equal to True if the left operand is greater than or equal to the right	x >= y
<=	Less than or equal to True if the left operand is less than or equal to the right	x <= y
is	x is the same as y	x is y
is not	x is not the same as y	x is not y

For example,

```
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:

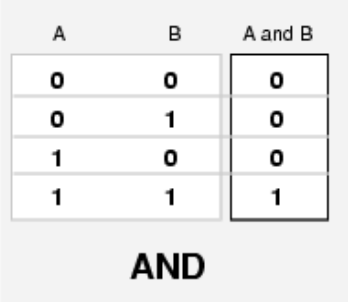
x > y is False
x < y is True
x == y is False
x != y is True
x >= y is False
x <= y is True

3. Logical operators:

- Logical operators are used to combine the results of two or more conditions.
- Logical operators are **and, or, not** operators.
- Python's logical operators mostly use boolean operands to produce boolean results.

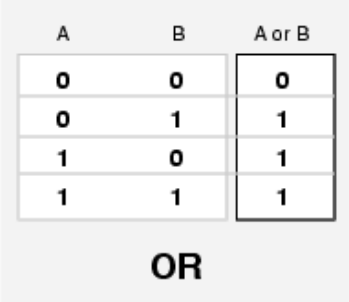
Operator	Description	Syntax
and	Logical AND: True if both the operands are true	x and y
or	Logical OR: True if either of the operands is true	x or y
not	Logical NOT: True if the operand is false	not x

Truth Table:



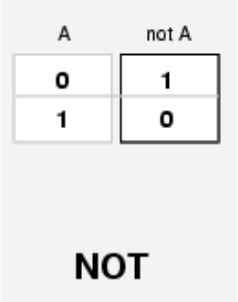
A	B	A and B
0	0	0
0	1	0
1	0	0
1	1	1

AND



A	B	A or B
0	0	0
0	1	1
1	0	1
1	1	1

OR



A	not A
0	1
1	0

NOT

Example:

```
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. Assignment operators

- Assignment operators are used in python to **assign values** to variables.
  (=,+=,-=,*=,/=,%=,**=,>>=,<<=,&=,|=,^=)

Operator	Description	Example
=	Simple Assignment	a=10 b=5 Sum=a+b
+=,-=,*=,/=,%=, **=,>>=,<<=,&=,|=,^=	Compound Assignment	a+=b (a=a+b)

Syntax:

Variable=expression (or) value

Ex.1

```
a = 21
b = 10
c = 0    # Simple Assignment Operator
c += a  #Compound Assignment Operator
print( "Line 1 - Value of c is ", c)
c **= a
print( "Line 2 - Value of c is ", c)
```

Output:

```
Line 1 - Value of c is  21
Line 2 - Value of c is
58425870183859825213811244 21
```

Ex2: value to multiple variables

```
a = b = c = 4
print (a, b, c )
```

Output:

```
4 4 4
```

5. Bitwise operator

- It works on **bits** and performs bit by bit operation. It operates on integers only.
- Decimal numbers are natural to humans. Binary numbers are native to computers. Binary, octal, decimal or hexadecimal symbols are only notations of the same number.
- Bitwise operators work with bits of a binary number. We have binary logical operators and shift operators.
- Bitwise operators are seldom used in higher level languages like Python.

Operator	Description	Syntax
&	Bitwise AND	x & y
|	Bitwise OR	x | y
~	Bitwise NOT	~x
^	Bitwise XOR	x ^ y
>>	Bitwise right shift	x>>y
<<	Bitwise left shift	x<<3

Example:

```
a = 60
b = 13
c = 0
c = a & b
print ("Line 1 - Value of c is ", c)
c = a | b
print ("Line 2 - Value of c is ", c)
c = a ^ b
print ("Line 3 - Value of c is ", c)
c = ~a
print ("Line 4 - Value of c is ", c)
c = a << 2
print ("Line 5 - Value of c is ", c)
c = a >> 2
print ("Line 6 - Value of c is ", c)
```

Output:

```
Line 1 - Value of c is  12
Line 2 - Value of c is  61
Line 3 - Value of c is  49
Line 4 - Value of c is  -61
Line 5 - Value of c is  240
Line 6 - Value of c is  15
```

6. Membership operators:

- **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

Operator	Description	Example
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	x in y, here in results in a 1 if x is a member of sequence y.
not in	Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.	x not in y, here not in results in a 1 if x is not a member of sequence y.

Ex.

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

Output:

True
True
True

7. Identity Operators:

- Identity operators compare the memory locations of two objects**.(is ,is not)**

Operator	Description	Example
Is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	x is y, here **is** results in 1 if id(x) equals id(y).
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	x is not y, here **is not** results in 1 if id(x) is not equal to id(y).

For ex.

```
a = 20
b = 20
if ( a is b ):
    print( "Line 1 - a and b have same identity“)
else:
    print ("Line 1 - a and b do not have same identity")
```

OUTPUT:

Line 1 - a and b have same identity

Operator Precedence

The precedence of the operators is essential to find out since it enables us to know which operator should be evaluated first. The precedence table of the operators in Python is given below.

Operator	Description
**	The exponent operator is given priority over all the others used in the expression.
~ + -	The negation, unary plus, and minus.
* / % //	The multiplication, divide, modules, reminder, and floor division.
+ -	Binary plus, and minus
>> <<	Left shift. and right shift
&	Binary and.
^ |	Binary xor, and or
<= < > >=	Comparison operators (less than, less than equal to, greater than, greater then equal to).
<> == !=	Equality operators.
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

CONDITIONAL OR DECISION MAKING STATEMENTS [Dec/Nov 2018]

- ✓ Conditional Statements are also called as Selection Structure, Decision Making Statements and Control Flow.
- ✓ In a program all the instructions are executed sequentially by default, when no repetitions of some calculations are necessary. In some situation we may have to change the execution order of statements based on condition or to repeat a set of statements until certain conditions are met.
- ✓ Conditional Statements decides the order in which statements or blocks of code are executed at runtime based on a condition.

Conditional Statements:

- A statement that controls the flow of execution depending on some condition.
- In python the keywords if , elif , and if-elif-else are used for **conditional** statements.
- The conditional statements are

a) if statement(conditionals)
b) if –else statement(alternative)
c) if –elif –else(chained conditional)

a) if statement(conditionals)

- ✓ The **if** statement is a decision making statement. It is used to control the flow of execution of the statements and also used to test logically whether the condition is true or false.
- ✓ The program evaluates the test condition and executes the statements only if the test condition is True.
- ✓ If the test condition is False, the statements are not executed.

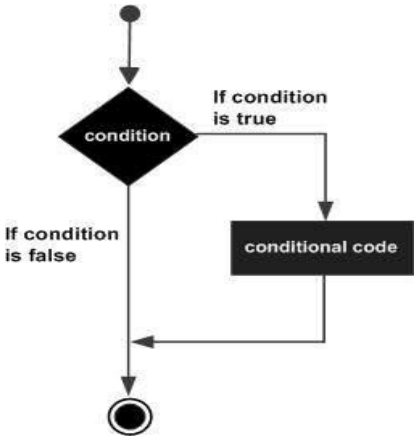
Properties of an if statement:

i. if the condition is true, then the simple or compound condition statements are executed.
ii. If the condition is false, it does not do anything.
iii. The condition is given in parenthesis and must be evaluated as true (non-zero value) or false (zero value).

Syntax:

```
if (expression):
        True Statements
                .
                .
                .
```

Flow Chart:



For ex, program to find given number is positive.

```
x=int(input("enter number"))
if ( x>0):
        print ("x is positive")
```

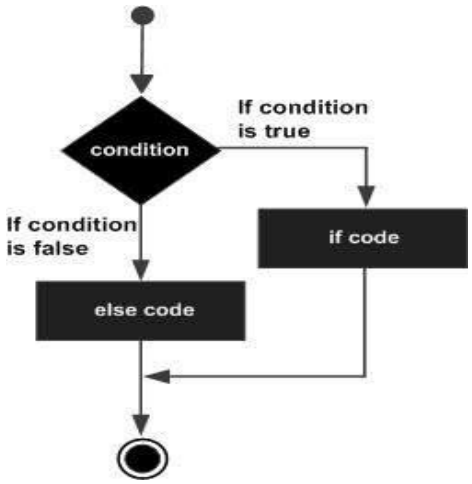
b) if –else statement

- ✓ It evaluates test condition and executes the True statement block only when the test condition is true.
- ✓ If the condition is False, false statements block is executed.
- ✓ Indentation is used to separate the blocks.

Syntax:

```
if(condition):
        True Statements
else:
        False Statments
```

Flow charts:



Ex. program to find the given number is odd or even

```
n = int( input("Enter a No: "))
if (n % 2 == 0):
        print (n," is even")
else:
        print (n," is odd")
```

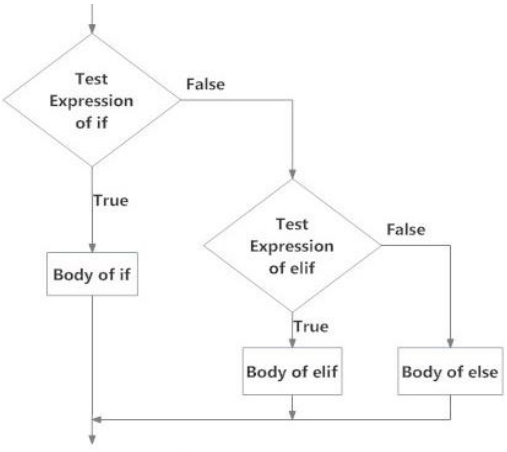
c) if –elif –else(chained conditional) or nested if…else statement:

- ✓ The elif is short form for else if.
- ✓ It is used to check multiple conditions.
- ✓ If the condition1 is false, it will check the condition 2 of the next elif block and so on.
- ✓ If all the conditions are false, then the else statement is executed.

Syntax:

```
if condition 1:
        True statement block for cond 1
elif condition 2:
        True statement block for cond 2
else:
        False statement Block
```

Flow chart:



Ex. Given number is positive, negative or zero.

```
n=int(input("Enter the value of n:"))
if(n>0):
        print("The number is positive")
elif (n<0):
        print("The number is negative")
elif (n==0):
        print("the number is equal to zero")
else:
        print("invalid")
```

OUTPUT:

Enter the value of n: -9
The number is negative

LOOPING OR ITERATING STATEMENTS: [DEC/NOV 2018]

- ✓ The loop is defined as the block of statements which are repeatedly executed for certain number of times until the specified condition is true.
- ✓ The loop consists of 2 parts.
  1.body of the loop
  2.control statement.
- ✓ Any looping statement would include the following **steps**:
  1. Initialization of a condition variable.
  2. Test the control statements
  3. Executing the body of the loop depending on the condition
  4. Updating the condition variable.

The following are the loop structures available in python:

1.while loop
2.for loop

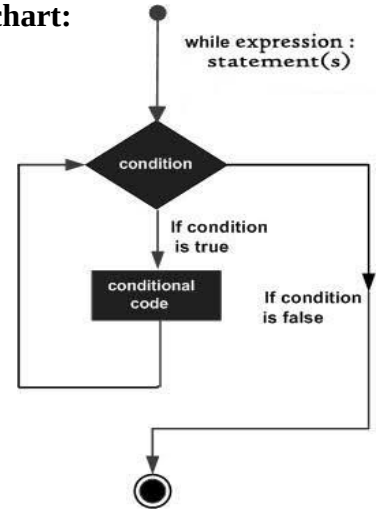
1. while loop

- It is a repetitive control structure, used to execute the statements within the body, until the condition is true.
- The **while** loop is an **entry controlled loop statement**, which means the condition is evaluated first and if it is true, then the body of the loop is executed.
- After executing the body of the loop, the condition is once again evaluated, if it is true, the loop continues until the condition finally becomes false and the control is transferred out of the loop.

Syntax:

```
while (expression):
        statement(s)
```

Flow chart:



Ex.1 Addition of numbers upto 10 by using the while loop.

```
i=0
sum=0
while(i<=10):
        sum=sum+i
        i=i+1
print("sum of numbers upto 10 is",sum)
```

Output:
sum of numbers upto 10 is 55

Using 'else' statement with while loops

while loop executes the block until a condition is satisfied. When the condition becomes false, the statement immediately after the loop is executed.

The else clause is only executed when your while condition becomes false. If you break out of the loop, or if an exception is raised, it won't be executed.

Syntax:

```
while condition:
        # execute these statements
else:
        # execute these statements
```

Example:

```
# Python program to illustrate 'else with while'
count = 0
while (count < 3):
        count = count + 1
        print("Hello Geek")
```

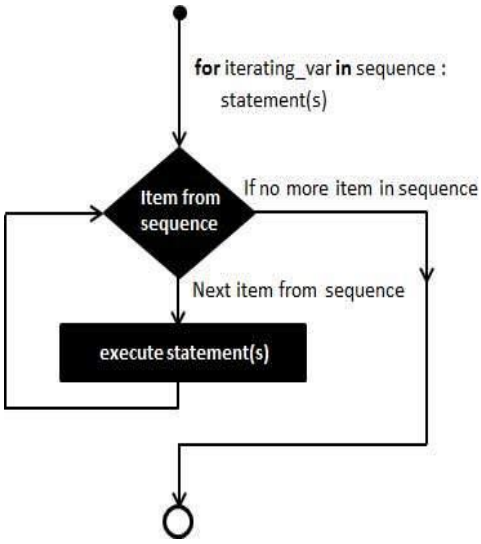
2. for loop:

- ✓ The **for** loop is another **repetitive control structure**, and is used to execute a set of instructions repeatedly, until the condition becomes false.
- ✓ In Python, the for loop is used to **iterate over a sequence** such as a list, string, tuple, other iterable objects such as range.
- ✓ The Initializing Counter Variable, Increment or Decrement and Condition checking is done in **for** statement itself, where as other control structures are not offered all these features in one statement.

Syntax:

```
for  iterating_var in sequence:
        statements(s)
```

Flow chart:



Ex 1. Addition of numbers upto 10 by using the for loop

```
sum=0
for  i  in  range(11):
        sum=sum+i
print("sum of numbers upto  10  is :",sum)
```

output:
sum of numbers upto 10 is: 55

Ex.2 printing characters in a word using for loop

```
for word in "Python":
        print("char in word:",word)
```

Output:

```
char in word: P
char in word: y
char in word: t
char in word: h
char in word: o
char in word: n
```

Ex3. print the pyramid of numbers.

```
for i  in  range(5):
        print("{0:3}  {1:16}".format(i, 10**i))
```

OUTPUT:

0	**1**
1	**10**
2	**100**
3	**1000**
4	**10000**

Using else statement with for loops:

We can also combine else statement with for loop like in while loop. But as there is no condition in for loop based on which the execution will terminate. So the else block will be executed immediately after for block finishes execution.

```
# Python program to illustrate combining else with for
list = ["geeks", "for", "geeks"]
for index in range(len(list)):
   print(list[index])
else:
   print("Inside Else Block")
```

LOOP CONTROL STATEMENTS IN DETAIL? (UNCONDITIONAL STATEMENTS) [Dec/Nov 2018]

- ✓ Loop control statements change execution from its normal sequence.
- ✓ When execution leaves a scope, all automatic objects that were created in that scope are destroyed.
- ✓ Python supports the following control statements.
  1. break statement
  2. continue statement
  3. pass statement

Control statement	Description
break	Terminates the loop statement and transfers execution to the statement immediately following the loop.
continue	Causes the loop to skip the reminder of its body and immediately retest its condition prior to reiterating.
pass	The pass statement in python is used when a statement is required syntactically but you do not want any command or code to execute.

1.break statement:

- ✓ The **break** statement is used to terminate the loop.
- ✓ When the keyword **break** is used inside any python loop, control is automatically transferred to the first statement after the loop. A **break** is usually associated with **if** statement.
- ✓ The **break** statement can be used in both while and for loops.

Syntax:

```
break
```

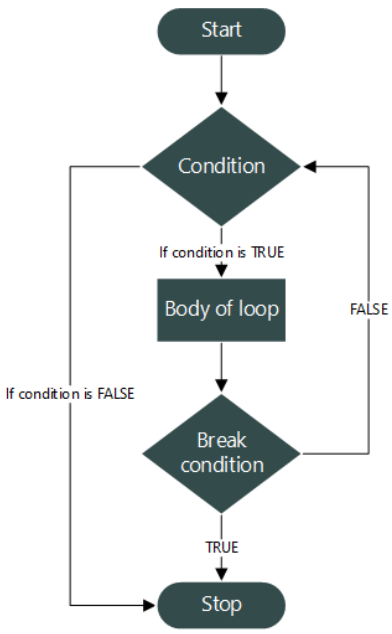
Ex.

```
for i in "Python":
        if i == 'h':
                break
         print ("Current Letter :", letter)
```

output:

Current Letter : P
Current Letter : y
Current Letter : t

Flow chart:



2.continue:

- ✓ In some situations, we want to take the control to the beginning of the loop, by passing the statements inside the loop which have not yet been executed, for this purpose the **continue** is used.
- ✓ When the statement **continue** is encountered inside any python loop control **automatically passes to the beginning of the loop**.
- ✓ The **continue** statement can be used in both while and for loops.

Syntax:

```
continue
```

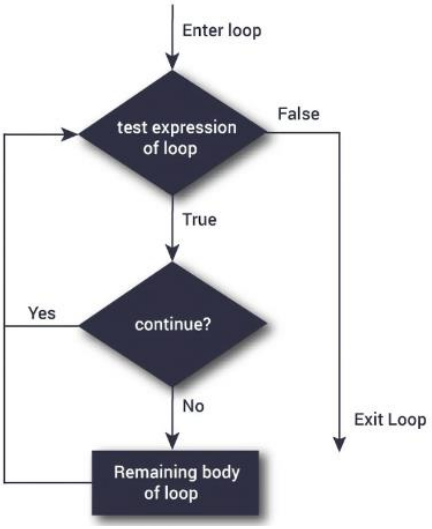
Example:

```
for letter in 'Python':
    if letter == 'h':
        continue
    print("Current Letter :", letter)
```

OUTPUT:

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : o
Current Letter : n
```

Flow chart:



3. pass statement:

- ✓ It is used when a statement is required syntactically but you do not want any command or code to execute.
- ✓ The **pass** statement is a null operation; nothing happens when it executes. We generally use it as a placeholder.

Syntax: pass

- ✓ Suppose we have a loop or a function that is not implemented yet, but we want to implement it in the future.
- ✓ They cannot have an empty body. The interpreter would complain. So, we use the pass statement to construct a body that does nothing.

For ex.

```
for letter in 'Python':
   if letter == 'h':
         pass
         print ('This is pass block' )
   print( 'Current Letter :', letter)
```

OUTPUT:

```
Current Letter: P
Current Letter: y
Current Letter: t
This is pass block
Current Letter: h
Current Letter: o
Current Letter: n
```

FRUITFUL FUNCTIONS

- A function that returns or yields a value to caller function (or calling function) is known 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.

RETURN VALUE:

- Return value means the result of a function.
- Return takes zero, values or an expression. Default value is none.
- If a function is called using an expression, the return value is the value of the expression.
- The value can be returned from a function using the keyword ***return*.**

Syntax:

return[expression _list]

For example,

```
#Area of Circle
def Area_Circle(r):
        return 3.14*r*r

r=int(input("Enter r:"))
print("ans=",Area_Circle(r))
```

function calling

- Sometimes it is useful to have multiple return statements, one in each branch of a conditional:

```
def  big(a,b):
    if(a>b):
        return a
    else:
        return b

print(big(5,8))
```

- In the above ex, since these return statements are in an alternative conditional, only one will be executed. As soon as one is executed, the function terminates without executing any subsequent statements.
- Code that appears after a return statement, or any other place the flow of execution can never reach, is called **dead code**.

PARAMETERS:

- Input values (parameters) are passed from function calling line to function definition block, these inputs are called as arguments or parameters.
- Here is an example of a user-defined function that takes an argument:

```
def print_twice(bruce):
    print (bruce)
    print (bruce)
```

- This function assigns the argument to a parameter named bruce. When the function is called, it prints the value of the parameter (whatever it is) twice.
- This function works with any value that can be printed.

Ex 1 >>> print_twice('Spam') #String
Spam
Spam

Ex 2 >>> print_twice(17) #Integer
17
17

Ex 3 >>> print_twice(math.pi) #Pre-defined value of 'pi'
3.14159265359
3.14159265359

Ex 4 >>> print_twice('Spam '*4) #String 4 times
Spam Spam Spam Spam
Spam Spam Spam Spam

Ex 5 **Use a variable as an argument:** #Variable value
>>> michael = 'Eric, the half a bee'
>>> print_twice(michael)
Eric, the half a bee.
Eric, the half a bee.

The name of the variable we pass as an argument (michael) has nothing to do with the name of the parameter (bruce). It doesn't matter what the value was called back home (in the caller)

Here in print_twice, we call everybody bruce.

LOCAL AND GLOBAL SCOPE (or) SCOPE OF VARIABLES

- All variables in a program may not be accessible at all locations in that program. This depends on where you have declared a variable.
- The scope of a variable determines the portion of the program where you can access a particular identifier.
- There are two basic scopes of variables in Python −
  - Global variables
  - Local variables

1. global variables:

Variables that are defined outside a function have a global scope.

2. local variables:

Variables that are defined inside a function body have a local scope.

Example

```
a=5              #→global variable
def fun():
        x=3      #→local variable
        print ("value of x is:", x)
        print ("value of a is:", a)
fun()
```

Output:

value of x is 3
value of a is 5

CALL-BY-VALUE vs. CALL-BY-REFERENCE

Call by Value: In this parameter passing method, values of actual parameters are copied to function's formal parameters and the two types of parameters are stored in different memory locations. So any changes made inside functions are not reflected in actual parameters of caller.

Call by Reference: Both the actual and formal parameters refer to same locations, so any changes made inside the function are actually reflected in actual parameters of caller.

Ex. for call by value:

```
def changeme(mylist):
        mylist.append([1,2,3,4]);
        print("Values inside the function: ", mylist)
        return
mylist = [10,20,30];
changeme(mylist);
print ("Values outside the function: ", mylist)
```

OUTPUT:
Values inside the function: [10, 20, 30, [1, 2, 3, 4]]
Values outside the function: [10, 20, 30, [1, 2, 3, 4]]

Ex. for call by reference:

```
def changeme(mylist):
        mylist = [1,2,3,4];
        print("Values inside the function: ", mylist)
        return
mylist = [10,20,30];
changeme(mylist);
print ("Values outside the function: ", mylist)
```

OUTPUT:
Values inside the function: [1, 2, 3, 4]
Values outside the function: [10, 20, 30]

FUNCTION COMPOSITION

- Function composition is a way of combining functions such that the result of each function is passed as the argument of the next function.
- For example, the composition of two functions f and g is denoted f(g(x)). x is the argument of g, the result of g is passed as the argument of f and the result of the composition is the result of f.

Example:

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

Output:

```
mainfun(8,8)
160
```

RECURSION

- Function which calls itself is known as Recursion. Recursion is a way of programming or coding a problem, in which a function calls itself one or more times in its body.
- Usually, it is returning the return value of this function call. If a function definition fulfils the condition of recursion, we call this function a recursive function.

Syntax:

```
Function1():
    Function1()
```

the function1() is called themselves continuously, so the function is in recursion manner.

Advantages:

1. The code look clean and elegant.
2. Large problems broken down into small problems

Disadvantages:

1. Logic is hard to follow.
2. It takes more memory.
3. It is hard to debug.

Example 1: Factorial of a number

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

x=int(input("Enter a number"))
print("Facorial is",fact(x))
```

function call

Output:

```
Enter a number   3
Factorial is 6
```

Example 2: Fibonacci series using recursion

```
def fibo(n):
    if(n <= 1):
        return n
    else:
        return(fibo(n-1) + fibo(n-2))

n = int(input("Enter number of terms:"))
print("Fibonacci sequence:")
for i in range(n):
    print(fibo(i))
```

OUTPUT:

```
Enter number of terms:5
Fibonacci sequence:
0
1
1
2
3
```

Example 3 Tower of Hanoi

```
def hanoi(disk,source,dest,aux):
    if disk>0:
        hanoi(disk-1, source, aux, dest)
        print('move disk from', source, 'to', dest)
        hanoi(disk-1, aux, dest, source)

source = 'A'
aux = 'B'
dest = 'C'
disks=int(input('how many disks:'))
hanoi(disk,source, dest, aux)
```

OUTPUT:

```
how many disks:3
move disk from A to C
move disk from A to B
move disk from C to B
move disk from A to C
move disk from B to A
move disk from B to C
move disk from A to C
```

STRINGS

- A string is a sequence of characters, enclosed within single-quotes (‘ ’) or double quotes(“ ”). Strings are **immutable,** which means you can’t change an existing string.
- For ex.

```
>>>fruit=”banana”
>>>print(fruit)
        banana
```

- **String index:** The expression in brackets is called an **index.**
- The index indicates which character in the sequence to be displayed. The index starts at 0.

Example

```
>>>fruit=”banana”
>>>print(fruit[1])
      a

>>>fruit=’banana’
>>>print(fruit[4])
      n
```

- As an index, we can use an expression that contains variables and operators.

```
>>>fruit=’banana’
>>>i=1
>>>fruit[i]
      a
>>>fruit[i+1]
      n
```

- The value of the index has to be an integer.

```
>>>letter[1.5]
```

Type error: string indices must be integers

Reading strings from keyboard:

- The **input**() function reads a line entered on a console by an **input** device such as a **keyboard** and convert it into a string and returns it. You can use this **input** string in your **python** code.

  Ex. str=input(“Enter a string:”)

String slices:

A segment of a string is called a slice. This can be represented using slicing operator ‘:’.

Example 1.

```
s= “Monty Python!”
print (s[0:5])
print(s[6:12]
```

Output:

Monty
Python

- The operator [n:m] returns the part of the string from the “n-eth” character to the “m-eth” character, including the first but excluding the last.
  - print (s[0:5]) #Monty
- If you omit the first index (before the colon), the slice starts at the beginning of the string.
  - print (s[:5]) #Monty
- If you omit the second index, the slice goes to the end of the string.
  - print (s[6:]) #Python

- If the first index is greater than or equal to the second the result is an empty string, represented by two quotation marks.
    - print (s[7:5]) #

- An empty string contains no characters and has length 0, but other than that, it is the same as any other string.

Accessing strings:

- Python does not support a character type; these are treated as strings of length one, thus also considered a substring.
- To access substrings, use the square brackets for slicing along with the index or indices to obtain your substring.
- For example −

```
var1 = 'Hello World!'
var2 = "Python Programming"
print("var1[0]: ", var1[0])
print ("var2[1:5]: ", var2[1:5])
```

OUTPUT:

```
var1[0]  :  H
var2[1:5]:  ytho
```

len( ):

- ✓ len() is a built-in(pre-defined) function that returns number of characters in a string.

```
>>>fruit=’banana’
>>>len(fruit)
    6
```

- ✓ To get the last letter of a string, you might be tempted to try something like this:

```
>>> length = len(fruit)
>>> last = fruit[length-1]
```

- ✓ Since we started counting at zero, the six letters are numbered 0 to 5.

```
>>> print last
    a
```

- ✓ Alternatively, you can use negative indices, which count backward from the end of the string. The expression fruit[-1] yields the last letter, fruit[-2] yields the second to last, and so on.

- ✓ Ex.

```
>>>fruit=’banana’
>>>print(fruit[-1]
    a
>>>print(fruit[-6]
    b
```

Traversal with a for loop

- ✓ A lot of computations involve processing a string one character at a time. Often they start at the beginning, select each character in turn, do something to it, and continue until the end. This pattern of processing is called a traversal.

- ✓ **Ex1:** One way to write a traversal is with a while loop:

```
index = 0
fruit=”banana”
while( index < len(fruit)):
        letter = fruit[index]
        print( letter )
        index = index + 1
```

OUTPUT:

b
a
n
a
n
a

- ✓ This loop traverses the string and displays each letter on a line by itself

- ✓ **Ex2:** Another way to write a traversal is with a for loop

```
fruit=”banana”
for char in fruit:
        print char
```

Immutability:

- ✓ Strings are immutable, that is we cannot change the existing strings.

Ex.

```
>>> msg=”Good Morning”
>>> msg[0]=’g’
```

Type error: ‘str’ object does not support item assignment

- ✓ The reason for the error is that strings are immutable, which means you can’t change an existing string. The best you can do is create a new string that is a variation on the original:

```
>>> greeting = 'Hello, world!'
>>> new_greeting = 'J' + greeting[1:]
>>> print (new_greeting Jello, world!)
```

- ✓ This example concatenates a new first letter onto a slice of greeting. It has no effect on the original string.

String functions & methods:

- ✓ A method is similar to a function—it takes arguments and returns a value.
- ✓ Python has several built-in functions associated with the string data type.
- ✓ These functions helps us to easily modify and manipulate strings.

str.isalnum() - String consists of only alphanumeric characters (no symbols)
str.isalpha() - String consists of only alphabetic characters (no symbols)
str.islower() - String‘s alphabetic characters are all lower case
str.isnumeric()- String consists of only numeric characters
str.isspace() - String consists of only whitespace characters

Ex.

```
Book1 = "PYTHON PROGRAMMING"
book2  = "Python Program Book"
print(book1.islower( ))
print(book1.isupper( ))
print(book2.islower( ))
```

OUTPUT:
False
True
False

join(), split(), and replace() methods

join( ) method:

- ✓ The str.join() method will concatenate two strings, but in a way that passes one string through another.
- ✓ **Syntax:** str.join(Sequence)

Ex.

```
s = "-";
seq = ("a", "b", "c");
print s.join( seq )
```

output: a-b-c

split( ) method:

- ✓ The method **split()** returns a list of all the words in the string

Ex.

```
words = "This is random text"
words2 = words . split(" ")
print(words2)
```

output:
['This', 'is', 'random', 'text']

replace( ) method:

- ✓ The replace() method returns a copy of the string where all occurrences of a substring is replaced with another substring.

Example:

```
song = 'cold, cold heart'
print (song . replace('cold', 'hurt'))
```

output:
hurt, hurt heart

STRING MODULES

- ✓ The string module provides additional tool to manipulate strings.
- ✓ This module contains a number of functions to process standard python strings.
- ✓ In recent version, string build in function are available as string module functions.

```
# String module
import string as str
text="Monty Python's Flying Circus"
print("upper", "=>",str.upper(text))
print( "lower", "=>",str.lower(text))
print("split", "=>",str.split(text))
print("join", "=>", str.join(str.split(text),"+"))
print("replace", "=>",str.replace(text,"Python", "Java"))
print("find", "=>",str.find(text, "Python")),
print("count", "=>",str.count(text, "n"))
```

OUTPUT:

```
upper => MONTY PYTHON'S FLYING CIRCUS
lower => monty python's flying circus
split => ['Monty', "Python's", 'Flying', 'Circus']
join => Monty+Python's+Flying+Circus
replace => Monty Java's Flying Circus
find => 6
count => 3
```

LISTS AS ARRAYS

List:

- A List is a group of values.
- Each and every value in a list is called as elements or items.
- Each element is separated using ,.
- List can also be sliced as [m:n].
-

Example

```
a=[1,2,5,8,9]
b=[“a”,”g”,”r”]
c=[67,”red”,90,”yellow”]
```

Accessing values in Lists:

- [] are used to access values in a list for slicing along with the index or indices to obtain value available at that index.
- Slice[m:n] where m is starting index which is included & n in end index which is excluded.

Example.py

```
l1=['English','Maths',1000,3000]
l2=[10,20,30,40,50]
print(l1[0])
print(l2[2:4])
print(l1[:-2])
print(l2[-6])
```

Output:

```
English
[30, 40]
['English', 'Maths']
Traceback (most recent call last):
 File "python", line 6, in <module>
IndexError: list index out of range
```

Updating list

Single or multiple elements can be updated in a list by giving the slice on the lefthand side of the assignment operator and also add the elements in a list with the append() method.

Example.py

```
s=['Red','green','Blue','yellow','Purple']
print("value at index 2:",s[2])
s[2]='Lavender'
print(s)
s.append('Orange')
print(s)
```

Output:

```
value at index 2: Blue
['Red', 'green', 'Lavender', 'yellow', 'Purple']
['Red', 'green', 'Lavender', 'yellow', 'Purple', 'Orange']
```

Deleting elements in a List

- When index of element is known then, ‘del’ keyword is used to delete.
- When element to delete is known,then remove() method is used.

Example.py

```
s=['Red','green','Blue','yellow','Purple']
print(s)
del s[2]
print(s)
s.remove('yellow')
print(s)
```

Output:

```
['Red', 'green', 'Blue', 'yellow', 'Purple']
['Red', 'green', 'yellow', 'Purple']
['Red', 'green', 'Purple']
```

ILLUSTRATIVE PROGRAMS

1. Square root of a number

```
def newtonsqrt(n):
        approx=0.5*n
        better=0.5*(approx+n/approx)
        while(better!=approx):
                approx=better
                better=0.5*(approx+n/approx)
        return approx
n=int(input("Enter the number:"))
result=newtonsqrt(n)
print("The square root for the given number is:",result)
```

Output:
Enter the number:81
The square root for the given number is: 9.0

2. gcd of a number

```
def gcd(a,b):
        if(b==0):
                return  a
         else:
                return gcd(b,a%b)

a=int(input("Enter first number:"))
b=int(input("Enter second number:"))
 print ("The GCD of the two numbers is:", gcd(a,b))
```

Output:
Enter the first number:15
Enter the second number:5
The GCD of the two numbers is 5

3. Exponentiation

```
import math
x=int(input("Enter X:"))
y=int(input("Enter Y:"))
print("Power:",math.pow(x,y))
```

Output:
Enter X:3
EnterY:2
Power :9

4. linear search

```
n=int(input("Enter the limit:"))
list=[ ]
print("Enter the element one by one:")
for i in range(0,n):
        x=int(input())
       list.append(x)
key=int(input("Which element to search:"))
count=0
for i in range(0,n):
        if(key==list[i]):
                count=count+1
 if(count>0):
        print("Element found")
 else:
        print("Element not found")
```

Output:
Enter the limit:5
Enter the element one by one:
34
12
45
67
32
Which element to search:45
Element found

5. Binary search

```
n=int(input("Enter the limit"))
list=[ ]
print("Enter list elements one by one")
for i in range(0,n):
    x=int(input())
    list.append(x)
print("The list elements are")
print(list)
key=int(input("Which element to search"))
locn=-1
first=0
last=n
while(first<=last):
    mid=(first+last)//2
    if(list[mid]>key):
        last=mid-1
    elif(list[mid]<key):
        first=mid+1
    else:
        first=last+1
if(list[mid]==key):
    locn=mid
if(locn<0):
    print("Element not found")
else:
    print(key, "Found at postion", locn)
```

Output:

```
Enter the limit 4
Enter list elements one by one
10
20
30
40
The list elements are
[10, 20, 30, 40]
Which element to search
40
Element not found
40   Found at position 3
```

6.Sum of array of numbers

```
sum=0
i=0
a=[10,20,30,40]
for i in range(4):
    sum=sum+a[i]
print(sum)
```

Additional Programs:

1.Write a python program to find factorial of a number without recursion and without recursion [Dec /Jan 2018]

without recursion:

```
n=int(input("Enter number:"))
i=1
fact=1
while(i<=n):
        fact=fact*i
        i=i+1
print("Factorial of the number is: ")
print(fact)
```

with recursion:

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

x=int(input("enter a number:"))
print("Facorial is:",fact(x))
```

Output:

```
enter a number : 3
Factorial is: 6
```

2.Write a python program to generate first 'N' Numbers.(0,1,1,2,3,5,8,13,……)
[Dec/Jan 2018]

```
def fibo(n):
    if(n <= 1):
        return n
    else:
        return(fibo(n-1) + fibo(n-2))

n = int(input("Enter number of terms:"))
print("Fibonacci sequence:")
for i in range(n):
    print(fibo(i))
```

Output:

```
Enter number of terms: 6
Fibonacci sequence:
0
1
1
2
3
5
```

2 Marks with Answers

1. Define Boolean Expression with example.

A boolean expression is an expression that is either true or false. The values true and false are called boolean values. The following examples use the operator "==" which compares two operands and produces True if they are equal and False otherwise:

Example :>>> 5 == 6 False

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

2. What are the different types of operators?

- Arithmetic Operator (+, -, *, /, %, **, //)
- Relational operator (== , !=, <>, < , > , <=, >=)
- Assignment Operator (=, += , *= , - =, /=, %= ,**=)
- Logical Operator (AND, OR, NOT)
- Membership Operator (in, not in)
- Bitwise Operator (& (and), | (or) , ^ (binary Xor), ~(binary 1's complement , << (binary left shift), >>
- (binary right shift))
- Identity(is, is not)

3. Explain modulus operator with example.

The modulus operator works on integers and yields the remainder when the first operand is divided by the second. In Python, the modulus operator is a percent sign (%). The syntax is the same as for other operators: Example:

>>> remainder = 7 % 3

>>> print remainder 1

So 7 divided by 3 is 2 with 1 left over.

4. Explain 'for loop' with example.

for loops are traditionally used when you have a block of code which you want to repeat a fixed number of times. he Python for statement iterates over the members of a sequence in order, executing the block each time.

The general form of a for statement is

Syntax:

Example:x = 4

for i in range(0, x): print i

Output:0 1 2 3

5. Explain relational operators.

The == operator is one of the relational operators; the others 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

6. Explain while loop with example.(or)Explain flow of execution of while loop with Example.(Jan 2019)

The statements inside the while loop is executed only if the condition is evaluated to true.

Syntax:

Example:

Program to add natural numbers upto, sum = 1+2+3+...+10 n = 10

```
# initialize sum and counter sum = 0
i = 1
while i <= n:
sum = sum + i
i = i+1        # update counter
# print the sum print("The sum is", sum)
```

7. Explain if-statement and if-else statement with example (or) What are conditional and alternative executions?

If statement:

The simplest form of if statement is:

Syntax:

```
If (condition or expression):
        True Statement
```

Example:

```
if x > 0:
print 'x is positive'
```

The boolean expression after 'if' is called the condition. If it is true, then the indented statement gets executed. If not, nothing happens.

If-else:

A second form of if statement is alternative execution, in which there are two possibilities and the condition determines which one gets executed. The syntax looks like this:

Example:

```
if x%2 == 0:
print 'x is even'
else:
print 'x is odd'
```

If the remainder when x is divided by 2 is 0, then we know that x is even, and the program displays a message to that effect. If the condition is false, the second set of statements is executed. Since the condition must be true or false, exactly one of the alternatives will be executed.

8. What are chained conditionals?

Sometimes there are more than two possibilities and we need more than two branches. One way to express a computation like that is a chained conditional:

Eg:

```
if x < y:
print 'x is less than y' elif x > y:
print 'x is greater than y'
else:
print 'x and y are equal'
```

elif is an abbreviation of "else if." Again, exactly one branch will be executed. There is no limit on the number of elif statements. If there is an else clause, it has to be at the end, but there doesn't have to be one.

9. What is a break statement?

When a break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.

Eg:

```
while True:
line = raw_input('>') if line == 'done': break
print line print'Done!'
```

10. What is a continue statement?

The continue statement works somewhat like a break statement. Instead of forcing termination, it forces the next iteration of the loop to take place, skipping any code in between.

Example:

```
for num in range(2,10):
if num%2==0;
print “Found an even number”, num
continue
print “Found a number”, num
```

11. What is recursion? (or) Define Recursion with an example.(Jan 2019) (May 2019)

The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function. Using recursive algorithm, certain problems can be solved quite easily. Examples of such problems are Towers of Hanoi (TOH), Inorder/Preorder/Postorder Tree Traversals, Depth First Search (DFS) of Graph, etc.

Example:

```
def factorial(n): if n == 1:
return 1 else:
return n * factorial(n-1)
```

12. Compare return value and composition.

Return Value Composition Return gives back or replies to the caller of the function. The return statement causes your function to exit and handback a value to its caller.

Example:

```
def area(radius):
temp = math.pi * radius**2 return temp
```

Calling one function from another is called composition.

Example:

```
def circle_area(xc, yc, xp, yp):
radius = distance(xc, yc, xp, yp) result = area(radius)
return result
```

13. Define string immutability.

Python strings are immutable. ‘a’ is not a string. It is a variable with string value. You can’t mutate the string but can change what value of the variable to a new string.

Program:

```
a = “foo”     Output:
#foofoo
# a now points to foo
b=a
# b now points to the same foo that a points to      #foo
```

It is observed that ‘b’ hasn’t changed even though ‘a’ has changed.

```
a=a+a
# a points to the new string “foofoo”, but b points to the same old “foo”
print a
print b
```

14. Explain about string module.

The string module contains number of useful constants and classes, as well as some deprecated legacy functions that are also available as methods on strings.

Example:

PROGRAM:

```
text = "Monty Python's Flying Circus"
# Perform all the operations and print the results side by side
print("upper", "=>", text.upper(), "\nlower", "=>", text.lower())
print("split", "=>", text.split(), "\njoin", "=>", "+".join(text.split()))
print("replace", "=>", text.replace("Python", "Java"))
print("find", "=>", text.find("Python"), "\nfind", "=>", text.find("Java"))
print("count", "=>", text.count("n"))
```

OUTPUT:

```
upper => MONTY PYTHON'S FLYING CIRCUS
lower => monty python's flying circus
split => ['Monty', "Python's", 'Flying', 'Circus']
join => Monty+Python's+Flying+Circus
replace => Monty Java's Flying Circus
find => 6
find => -1
count => 3
```

15. Explain global and local scope. (or) Comment with an example on the use of local and global variable with the same identifier name. (May 2019)

The scope of a variable refers to the places that you can see or access a variable. If we define a variable on the top of the script or module, the variable is called global variable. The variables that are defined inside a class or function is called local variable.

Example:

```
def my_local():
a=10
print(“This is local variable”)
```

Example:

```
a=10
def my_global():
print(“This is global variable”)
```

16. Compare string and string slices. A string is a sequence of character. Eg: fruit = ‘banana’

String Slices :

A segment of a string is called string slice, selecting a slice is similar to selecting a character.

```
Eg:>>> s ='Monty Python'
>>> print s[0:5] Monty
>>> print s[6:12] Python
```

17. Mention a few string functions.

s.captilize() – Capitalizes first character of string s.count(sub) – Count number of occurrences of sub in string s.lower() – converts a string to lower case

s.split() – returns a list of words in string

18. What are string methods?

A method is similar to a function. It takes arguments and returns a value. But the syntax is different. For example, the method upper takes a string and returns a new string with all uppercase letters: Instead of the function syntax upper(word), it uses the method syntax word.upper()

.>>> word = 'banana'

>>> new_word = word.upper()

>>> print new_word

BANANA

19. Write a Python program to accept two numbers, multiply them and print the result. (Jan-2018)

print("Enter two numbers") val1=int(input()) val2=int(input()) prod=val1*val2

print("The product of the two numbers is:",prod)

20. Write a Python program to accept two numbers, find the greatest and print the result. (Jan-2018)

```
print("Enter two numbers")
val1 = int(input())
val2 = int(input())
# Check which number is larger
if val1 > val2:
   largest = val1
else:
   largest = val2
# Print the largest number
print("Largest of two numbers is:", largest)
```

Sample output:

Enter two numbers

4

7

Largest of two numbers is: 7

21. What is the purpose of pass statement?

Using a pass statement is an explicit way of telling the interpreter to do nothing.

Example:def bar():

pass

If the function bar() is called, it does absolutely nothing.

22. What is range() function?

If you do need to iterate over a sequence of numbers, the built-in function range() comes in handy. It generates arithmetic progressions:

Example:

Prints out the numbers 0,1,2,3,4 for x in range(5):

print(x)

This function does not store all the values in memory, it would be inefficient. So it remembers the start, stop, step size and generates the next number on the go.

23. Define Fruitful Function.

The functions that return values, is called fruitful functions. The first example is area, which returns the area of a circle with the given radius:

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

24. What is dead code?

Code that appears after a return statement, or any other place the flow of execution can never reach, is called

dead code.

25. Explain Logical operators

There are three logical operators: and, or, and not. For example, x > 0 and x < 10 is true only if x is greater than 0 and less than 10. n%2 == 0 or n%3 == 0 is true if either of the conditions is true, that is, if the number is divisible by 2 or 3. Finally, the not operator negates a boolean expression, so not(x > y) is true if x > y is false, that is, if x is less than or equal to y. Non-zero number is said to be true in Boolean expressions.

26. Do loop statements have else clauses? When will it be executed? (Nov / Dec 2019)

Loop statements may have an else clause, it is executed when the loop terminates through exhaustion of the list (with for) or when the condition becomes false (with while), but not when the loop is terminated by a break statement.

27. Write a program to display a set of strings using range( ) function (Nov / Dec 2019)

```
# List of strings
strings = ["apple", "banana", "cherry", "date", "elderberry"]
 # Loop through the indices of the list using range()
for i in range(len(strings)):
      print(strings[i])
```

UNIT IV LISTS, TUPLES, DICTIONARIES

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

LISTS

Like a string, a **list** is a sequence of values. In a string, the values are characters; In a list, they can be of any type. The values in a list are called **elements** or sometimes **items**.

A list need not be homogenous. The elements of a list ***don't have to be the same type***. A list can contain different data type items such as string, integer, float, real and another list. Each and every item has its unique index.

There are several ways to create a new list; the simplest is to enclose the elements in ***square brackets ([ ]):***

```
>>>L1=[10, 20, 30, 40]                  #List of four Integers
>>>L2=['Apple', 'Banana', 'Orange']     #List of three Strings
>>>L3=['spam', 2.0, 5, [10, 20]]        #list with string, a float, an integer,  and another list
>>>L4=[]                                #A list that contains no elements is called an empty list;
```

Mutability in Lists

To access the elements of a list the bracket operator is used. The expression inside the brackets specifies the ***index***. The ***indices starts at 0.***

Any ***integer expression*** can be used as an index. If an index has a ***negative value***, it counts backward from the end of the list.

```
>>> L2=['Apple', 'Banana', 'Orange']
>>> L2[0]
     'Apple'
```

Unlike strings, ***lists are mutable***. We can delete or replace the items or elements in the list.

```
>>> L2[1]= 'Grapes'  #replaces 'Banana'
>>> L2
    ['Apple', 'Grapes', 'Orange']
```

The in operator also works on lists:

```
>>> L2=['Apple', 'Banana', 'Orange']
>>>'Apple' in L2
     True
>>>'Banana' in L2
      False
```

Traversing a List:

The most common way to traverse the elements of a list is with for loop. For loop in python is an iterator that works on any sequence like list, tuple and string.

Example 1: Program to list contents

```
mylist = ['one', 'two', 'three']

for x in mylist:

        print('number' ,x)
```

Output:

```
number one
number two
number three
```

Accessing Values in Lists:

To access values in lists, use the square brackets for slicing along with the index or indices to obtain value available at that index.

Example: Program to access the values in lists.

```
list1 = ['computer', 'maths', 'physics', 'chemistry']
list2 = [1,2,3,4,5]
list3 = ['x', 'y', 'x']
print(list1[0])
print(list2[2:5])
print(list3)
```

Output:

```
computer
[3,4,5]
['x', 'y', 'x']
```

Updating list

Single or multiple elements can be updated in a list by giving the slice on the lefthand side of the assignment operator and also can add the elements in a list with the append() method.

Example.py

```
s=['Red','green','Blue','yellow','Purple']
print("value at index 2:",s[2])
s[2]='Lavender'
print(s)
s.append('Orange')
print(s)
```

Output:

```
value at index 2: Blue
['Red', 'green', 'Lavender', 'yellow', 'Purple']
['Red', 'green', 'Lavender', 'yellow', 'Purple', 'Orange']
```

Deleting elements in a List

- When index of element is known then, 'del' keyword is used to delete.
- When element to delete is known,then remove() method is used.

Example.py

```
s=['Red','green','Blue','yellow','Purple']
print(s)
del s[2]
print(s)
s.remove('yellow')
print(s)
```

Output:

```
['Red', 'green', 'Blue', 'yellow', 'Purple']
['Red', 'green', 'yellow', 'Purple']
['Red', 'green', 'Purple']
```

LIST OPERATIONS

List Operations:

List respond to the + and * operators much like strings; they mean concatenation and repetition here too, expect that the result is a new list, not a string.

Python Expression	Result	Description
len ([1,2,3,4])	4	Length
[1,2,3,4] + [5,6,7,8]	[1,2,3,4,5,6]	Concatenation
['Hi!'] * 2	['Hi!', 'Hi']	Repetition
2 in [1,2,3]	True	Membership
for x in [1,2,3,4,5]: print(x)	1 2 3 4 5	Iteration

The + is a concatenation operator, that concatenates lists:

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

The * is a Repetition operator, that repeats the list for a given number of times:

```
>>> [0] * 4
[0, 0, 0, 0]
>>> [1, 2, 3] * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

The first ex, repeats [0] four times. The second ex, repeats the list [1, 2, 3] three times.

len() is used to find the number of elements in a sequence.

```
>>>list1=[1,2,3,4]
>>>print(len(list1))
    4
```

max () & min(): max(s) returns the largest value in a sequence s and min(s) returns the smallest value in a sequence s

```
>>>list1=[1,2,3,4]
>>>print(max(list1))
        4
>>>print(min(list1))
        1
```

List index:

The index operator[ ] is used to access an item in a list. Index starts from 0. So, a list having 5 elements will have index from 0 to 4.Trying to access an element other that this will raise an **IndexError**. The index must be an integer. We can't use float or other types, this will result into **TypeError**. Nested list are accessed using nested indexing.

Example: Program to access list elements using index.

```
my_list = ['p', 'y', 't', 'h', 'o', 'n']
print(my_list[4])
n_list = ["happy", [2,0,1,5]]
print(n_list)
print(n_list[0][1])
print(n_list[1][3])
```

Output:

```
o
['happy', [2,0,1,5]]
a
5
```

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 : Program to use negative index in lists.

```
my_list = ['p', 'y', 't', 'h', 'o', 'n']
print(my_list[-1])
print(my_list[-3])
print(my_list[-5])
```

Output:

```
n
h
y
```

LIST SLICES

We can access a range of items in a list by using the slicing operator(colon :)

The slice operator also works on lists:

```
>>> t = [‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’]
>>> t =[1:3]
[‘b’, ‘c’]
>>> t[ :4]
[‘a’, ‘b’, ‘c’, ‘d’]
>>> t[3:]
[‘d’, ‘e’, ‘f’]
```

If you omit the first index, the slice starts at the beginning. If you omit the second, the slice goes to the end. So if you omit both, the slice is a copy of the whole list:

```
>>> t[:]
[‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’]
```

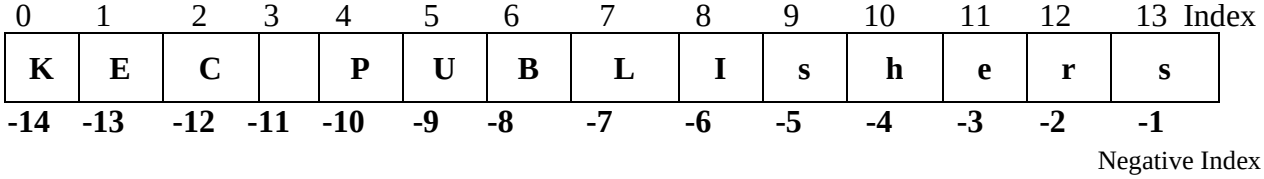
Since lists are mutable, it is often useful to make a copy before performing operations that modify lists.

Example: Program to use slice operator in the list

```
my_list = ['K', 'E', 'C', ' ', 'P', 'u', 'b', 'l', 'i', 's', 'h', 'e', 'r', 's']
print(my_list[0:3])
print(my_list[:-5])
print(my_list[4:])
print(my_list[:])
```

Output:
[‘K’, ‘E’, ‘C’]
[‘K’, ‘E’, ‘C’, ‘ ’, ‘P’, ‘U’, ‘B’, ‘L’, ‘I’]
[‘P’, ‘U’, ‘B’, ‘L’, ‘I’, ‘S’, ‘H’, ‘E’, ‘R’, ‘S’]
[‘K’, ‘E’, ‘C’, ‘ ’, ‘P’, ‘U’, ‘B’, ‘L’, ‘I’, ‘S’, ‘H’, ‘E’, ‘R’, ‘S’]

Slicing can be bets visualized by considering the index to be between the elements as shown below. So if we want to access a range, we need two index that will slice that portion from the list.



LIST METHODS

List method is a method to list all the elements in the set. Python provides methods that operates on lists. They are accessed as **list.method( )**.

- **List methods are,**

1. list.append(elem) – adds a single elements to the end of the list.
2. list.insert(index,elem) – inserts the element at the given index.
3. list.extend(list2) – adds the elements in list2 to the end of the list.
4. List.index(elem) – search for the given element from the start of the list and returns its index.
5. list.remove(elem) – search for the first instance of the given element and removes it.
6. list.sort( ) – sorts the list in place

7. list.reverse( ) – reverses the list in place
8. list.pop(index) – removes and returns the elements at the given index.
9. list.clear( ) – removes all elements from the list
10. list.count( ) – returns the count of number of items passed as an argument.
11. list.copy( ) – returns a shallow copy of the list

Example for list methods:

```
list = [‘apple’ , ‘orange’ , ‘grapes’]
list.append(‘banana’)   #append element at end
list.insert(0, ‘gova’)  #insert element at index 0
list.extend([‘chikoo’ , ‘jackfruit’])  #add elements at end
print list       #[‘gova’ , ‘apple’ , ‘orange’ , ‘grapes’ , ‘banana’ ,    ‘chikoo’ , ‘jackfruit’]
print list.index(‘orange’)  #2
list.remove(‘orange’)  #search and remove the element
list.pop(1)  #removes and returns ‘apple’
print list  #[‘gova’ , ‘grapes’ , ‘banana’ , ‘chikoo’ , ‘jackfruit’]
```

Output:

```
[‘guava’ , ‘apple’ , ‘orange’ , ‘grapes’ , ‘banana’ ,    ‘chikoo’ , ‘jackfruit’]
2
[‘guava’ , ‘grapes’ , ‘banana’ , ‘chikoo’ , ‘jackfruit’]
```

append():
This method appends / adds the pass object (v) to the existing list.

```
list1=[1,2,3,4]
list1.append(99)
print(list1)
Output:
[1, 2, 3, 4, 99]
```

insert():
This method insert the given element at the specified position.

```
list1=[1,2,3,4]
list1.insert(1,100)
print(list1)
Output:
[1, 100, 2, 3, 4]
```

remove():
Removes the element from the list. If there is no element then it displays error.

```
list1=[1,2,3,4]
list1.remove(4)
print(list1)
Output:
[1, 2, 3]
```

extend():
This method appends the contents of the list.

```
list1=[1,2,3,4]
list1.extend([10,20,30])
print(list1)
Output:
[1, 2, 3, 4, 10, 20, 30]
```

sort():
This method sorts the element either alphabetically or numerically.

```
list1=[5,2,10,4]
list1.sort()
print(list1)
Output:
[2, 4, 5, 10]
```

reverse():
This method reverses the element in the list.

```
list1=[5,2,10,4]
list1.reverse()
print(list1)
Output:
[4, 10, 2, 5]
```

count():
This method returns count of how many times elements occur in the list.

```
list1=[5,2,10,4,3,5]
print(list1.count(5))
Output:
2
```

index():
This method returns the index value of an element.

```
list1=[5,2,10,4,3]
print(list1.index(10))
Output:
2
```

pop():
This method removes the element from the list at the specified index. If the index value is not specified it removes the last element from the list.

```
list1=[5,2,10,4,3]
print(list1.pop(4))
print(list1.pop(7))
Output:
3
Traceback (most recent call last):
 File "python", line 3, in <module>
IndexError: pop index out of range
```

del():
Element to be deleted is mentioned using list name and index.

```
list1=[5,2,10,4,3]
del list1[3]
print(list1)
Output:
[5, 2, 10, 3]
```

LIST LOOP

Creating list using range( )

The range( ) functions returns an immutable sequence object of integers between the given start integer to the stop integer.

It generates a list of numbers, which is generally used to iterate over with loops.

Syntax:

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

where,

start – starting number of the sequence
stop – generate numbers up to, but not including this number.
step – integer value which determines the increment between each integer in the sequence

Iterate or Loop a List using 'for':

Example 1: To print numbers using range( ) function

```
for i in range(5):
    print(i, end= "  ")
```

Output:
0 1 2 3 4

Example 2:

```
thislist = ["apple", "banana", "cherry"]
for x in thislist:
    print(x)
```

Iterate or Loop a List using 'while':

We can loop through the list items by using a while loop.

The len() function is used to determine the length of the list, then start at 0 and loop through the list items by referring to their indexes.

After each iteration, the index must be incremented by one.

Example: Loop a list using while loop

```
mylist = range(5)
i=0
while(i<len(mylist)):
    print(i,end= "  ")
    i+=1
```

Output:

0 1 2 3 4

Looping Using List Comprehension

List Comprehension offers the shortest syntax for looping through lists:

Example

A short hand for loop that will print all items in a list:

```
thislist = ["apple", "banana", "cherry"]
[print(x) for x in thislist]
```

ALIASING

An object with more than one references has more than one name, such object is called alias.

```
Example:   a = [1, 2, 3]
           b = a
           b is a
```

program of List Aliasing

```
a=[1,2,3,4]
b=a
print  'b:',b
b.append(5)
print 'a after change in b:',a
```

OUTPUT

```
b: [1,2,3,4]
a after change in b:[1,2,3,4,5]
```

CLONING LISTS:

Cloning is different from aliasing. Aliasing just creates another name for the same list. Cloning creates a new list with same values under another name. Taking any slice of a list creates a new list.

Example:

```
fruits=[“apples”,”pear”,”grapes”,”orange”,”mango”]
fruit_list=fruits
clonefruit=fruit_list[:]
clonefruit.append(“banana”)
fruit_list[1]=”Lemon”
print “Original List:”,fruits
print “Alias List:”,fruit_list
print “clone List:”,clonefruit
```

OUTPUT:

Original List: [“apples”, “Lemon”, “grapes”, “orange”, “mango”]

Alias List: [“apples”, “Lemon”, “grapes”, “orange”, “mango”]

LIST PARAMETERS

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

For ex: delete _head removes the first element from a list:

```
def delete_head(t):
    del t[0]
>>> letters=[‘a’,b’,’c’]
>>>delete_head(letters)
>>>letters
  [‘b’,’c’]
```

The parameter‘t’ and the variable letters are aliases from the same object.

TUPLE

A tuple is a sequence of values. The values can be any type. They are immutable (i.e)objects whose value is unchangeable once they are created.

Example: t = ('a', 'b', 'c',’d’, 'e')

CREATING AND ACCESSING TUPLE:

Tuple can be created by putting different values separated by comma;the parantheses are optional

Example:

```
a=()  #Empty tuple

b=(1,2,3) #Integer tuple

c=(1,”hello”,3.4) # Mixed tuple
```

Simple example for tuples:

```
No's_Tuple=(1,2,3)
print  No's_tuple
Nested _tuple=("string",['list','in','tuple'],('tuple','in',tuple'))
print  Nested _tuple[0]
print  (Nested_tuple[1]
print  Nested _tuple[2]
Mixed_tuple=3,"kg","apples"
print(Mixed_tuple)
Nos,measure,what=Mixed_tuple
print(Nos)
print(measure)
print(what)
```

OUTPUT:

```
(1, 2, 3)
string
['list', 'in', 'tuple']
('tuple', 'in', 'tuple')
(3, 'kg', 'apples')
 3
 kg
apples
```

OPERATIONS IN TUPLE

Two operators + and * is allowed in tuples. '+' concatenates the tuples and '*' repeates the tuple elements a given number of times.

Ex:

```
t1= (1,2,3,4,7,8,9)
t2=('a',)*5
 t3=t1+t2
print "Add operation:",t3
print "Repeat operation:",t2
```

OUTPUT:

```
Add operation: (1, 2, 3, 4, 7, 8, 9, 'a', 'a', 'a', 'a', 'a')
```

SLICING AND INDEXING OF TUPLES:

Slicing of tuples is similar to list.

Ex:

```
a=('physics','chemistry',1997,2000)
b=(1,2,3,4,5)
c=("x","y",(10,4,5))
print "a[0]: ",a[0]
print "c[2]: ",c[2]
print "b[-2]: ",b[-2]
print "b[1:3]: ",b[1:3]
```

OUTPUT:

```
a[0]:  physics
c[2]:  (10, 4, 5)
b[-2]:  4
b[1:3]:  (2, 3)
```

DELETING AND UPDATING TUPLES:

Tuples are immutable; the elements cannot be changed. In lists the elements can be deleted.In tuples elements cannot be deleted. However the entire tuple can be deleted using del.Similarly the elements cannot be modified in a tuple.

In tuple we can replace one tuple with another, but can't modify the elements.

Ex :

```
>>>t = ('a', 'b', 'c', 'd', 'e')
>>>t =('A',)+t[1:]
<<<t
    ('A',  'b' ,  'c',  'd',  'e')
```

TUPLE ASSIGNMENT

Python has a very powerful tuple assignment feature that allows a tuple of variables on the left of an assignment to be assigned values from a tuple on the right of the assignment.

Example: a,b,c=1,2,3

The left side is a tuple of variable; the right side is a tuple of values. Each value is assigned to its respective variable. The number of variables on the left and the right of the assignment operator must be same.

All the expression on the right side are evaluated before any of the assignment. This features make tuple assignment quite versatile.

For Example:

>>> (a, b, c)=(1, 2, 3)

 # Program to swap a and b (without using third variable)

```
a=2; b=3
print(a, b)
a,b=b,a
print(a, b)
```

OUTPUT:

(2, 3)

(3, 2)

- **One way to think of tuple assignment is as tuple packing/unpacking:**

In tuple packing, the value on the left are 'packed' together in a tuple:

```
>>> b= ("George",25,"20000") #tuple packing
>>> (name, age, salary) = b #tuple unpacking
>>> name
    'George'
>>> age
     25
```

TUPLE AS RETURN VALUE

Functions can only return value, but if the value is a tuple, the effect is the same as returning multiple values. For example: If we want to divide two integers and compute the quotient and remainder, it is inefficient to compute x/y and then x%y. It is better to compute them both at the same time.

The built-in-function divmod takes two arguments and return a tuple of two values: the quotient and remainder. You can store the result as a tuple:

```
>>> def divmod(x,y):
        Return(x/y,x%y)

>>> t=divmod(7,3)
>>>t
>>>(2,3)
```

Or use tuple assignment to store the elements separately:

```
>>>quot,rem=divmod(7,3)
>>>quot
    2
>>>rem
    1
```

Here is an example of a function that returns a tuple:

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

max and min are build-in-functions that find the largest and smallest elements of a sequence.min_max computes both and returns a tuple of two values.

DICTIONARIES

DICTIONARY

A dictionary is set of pairs of value with each pair containing a key and an item and is enclosed in curly brackets. It is one of the compound data type like string, list and tuple.

- In dictionary the index can be immutable data type.
- It is a set of zero or more ordered pairs of key and value.
- The value can be any type.
- Each key may occur only once in the dictionary
- No key may be mutable. A key may not be a list or tuple or dictionary and so on.

Example:

```
dict={} #empty dictionary
d1={1:'fruit', 2:'vegetables', 3:'cereals'}
```

Creation and Accessing dictionary

There are two ways to create dictionaries.

1) It is created by specifying the key and value separated by a colon(:) and the elements separated by commas and entire set of elements must be enclosed by curly braces.
2) dict() constructor that uses a sequence to create dictionary.

Example:

```
d1={1:'fruit',2:'vegetables',3:'cereals'}
d2=dict({'name':'aa','age':40})
print(d1[1])
print(d2)
Output:
fruit
{'name': 'aa', 'age': 40}
```

Deletion of elements

The elements in the dictionary can be deleted by using del statements.
The entire dictionary can be deleted by specifying the dictionary variable name.

Example:

```
del d1[1]#remove entry with key '1'
print(d1)
d1.clear()#remove all entries in dict
print(d1)
```

Output:

```
{2: 'vegetables', 3: 'cereals'}
{}
```

Updating elements

In dictionary the keys are always immutable.
The values are mutable that can be modified. New-key value pair can be inserted or deleted from the dictionary.

Example:

```
d3={'name':'abc','dept':'ece'}
d3['name']='xxxx'
print(d3)
```

Output:

```
{'name': 'xxxx', 'dept': 'ece'}
```

Length:

The len function works on dictionaries; it returns the number of key-value pairs:

```
>>> dict1 = {'one': 'uno', 'two': 'dos', 'three': 'tres'}
>>> len(dict1)
    3
```

In Operator:

The **in operator** works on dictionaries; it tells you whether something appears as a key in the dictionary (appearing as a value is not good enough).

```
>>> dict1= {'one': 'uno', 'two': 'dos', 'three': 'tres'}
>>> 'one' in eng2sp
    True
>>> 'uno' in dict1
     False
```

To see whether something appears as a value in a dictionary, we can use **the method values**, which returns the values as a list, and then use the in operator:

```
>>> vals = eng2sp.values()
>>> 'uno' in vals
    True
```

Delete Dictionary Elements

- We can either remove individual dictionary elements or clear the entire contents of a dictionary.
- To explicitly remove an entire dictionary, just use the del statement.

Following is a simple example −

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
del dict['Name']          # remove entry with key 'Name'
dict.clear()              # remove all entries in dict
del dict                  # delete entire dictionary
print ( "dict  ['Age']: " ,   dict ['Age'] )
print ("dict  ['School']: " ,   dict ['School'])
```

Properties of Dictionary Keys:

Here are two important points to remember about dictionary keys −

1) More than one entry per key is not allowed. This means no duplicate key is allowed. When duplicate keys are encountered during assignment, the last assignment wins.

For example −

```
dict = {'Name': 'Zara', 'Age': 7, 'Name': 'Mani'}
print ("dict['Name']: ", dict['Name'])
```

Output:

```
dict['Name']:  Mani
```

2) Keys must be immutable. This means you can use strings, numbers or tuples as dictionary keys but something like ['key'] is not allowed.

Following is a simple example −

```
dict = {['Name']: 'Zara', 'Age': 7}
print ("dict['Name']: ", dict['Name'])
```

When the above code is executed, it produces the following result

```
Traceback (most recent call last):
File "test.py", line 3, in <module>
dict = {['Name']: 'Zara', 'Age': 7}
TypeError: list objects are unhashable
```

Looping and dictionaries:

If you use a dictionary in a for statement, it traverses the keys of the dictionary.

For example,

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

print_hist prints each key and the corresponding value:

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

Output:

```
>>> h = histogram('parrot')
>>> print_hist(h)
a1
p1
r2
t1
o1
```

Again, the keys are in no particular order.

OPERATIONS IN DICTIONARY

1. cmp(dict1, dict2)

- Compares elements of both dict.
- This method returns 0 if both dictionaries are equal, -1 if dict1 < dict2 and 1 if dict1 > dic2.

For ex.

```
dict1 = {'Name': 'Zara', 'Age': 7};
dict2 = {'Name': 'Mahnaz', 'Age': 27};
dict3 = {'Name': 'Abid', 'Age': 27};
dict4 = {'Name': 'Zara', 'Age': 7};
print ("Return Value :",cmp(dict1, dict2))
print ("Return Value :",cmp(dict2, dict3))
print ("Return Value :",cmp(dict1, dict4))
```

OUTPUT:

Return Value : -1
Return Value : 1
Return Value : 0

2. len(dict)

Gives the total length of the dictionary. This would be equal to the number of items in the dictionary.

For ex.

```
dict = {'Name': 'Zara', 'Age': 7};
print ("Length : ", len (dict))
```

OUTPUT:

Length : 2

3. str(dict)

Produces a printable string representation of a dictionary

For ex.

```
dict = {'Name': 'Zara', 'Age': 7}
print ("Equivalent String :",str (dict))
```

OUTPUT:

Equivalent String : {'Age': 7, 'Name': 'Zara'}

4. type(variable)

- Returns the type of the passed variable. If passed variable is dictionary, then it would return a dictionary type.

For ex.

```
dict = {'Name': 'Zara', 'Age': 7};
print ("Variable Type :", type (dict))
```

METHODS IN DICTIONARY

1.dict.clear()

Removes all elements of dictionary dict

Ex.

```
dict = {'Name': 'Zara', 'Age': 7}
print ("Start Len :",len(dict))
dict.clear()
print ("End Len :",len(dict))
```

OUTPUT:

Start Len : 2

End Len : 0

2. dict. copy()

Returns a shallow copy of dictionary *dict*

Ex.

```
dict1 = {'Name': 'Zara', 'Age': 7};
dict2 = dict1.copy()
print ("The copied value:",dict2)
```

3.dict.values()

The method **values()** returns a list of all the values available in a given dictionary.

For ex.

```
dict = {'Name': 'Zara', 'Age': 7}
print ("Values:",dict.values())
```

4.dict.key()

The method **keys()** returns a list of all the available keys in the dictionary

Example:

```
dict = {'Name': 'Zara', 'Age': 7}
print("The keys are:",dict.keys())
```

5. dict.has_key(key)

The method **has_key()** returns true if a given *key* is available in the dictionary, otherwise it returns a false.

Ex.

```
dict = {'Name': 'Zara', 'Age': 7}
print "Value :", dict.has_key('Age')
print "Value :",dict.has_key('Sex')
```

6.dict.item()

The items() method returns a view object that displays a list of a given dictionary's (key, value) tuple pair.The items() method doesn't take any parameters.

For.ex

```
sales = { 'apple': 2, 'orange': 3, 'grapes': 4 }
print(sales.items())
```

7.dict.fromkeys()

The method **fromkeys()** creates a new dictionary with keys from *seq* and *values* set to value.

Syntax:

dict.fromkeys(seq [, value])

For ex.

```
keys = {'a', 'e', 'i', 'o', 'u' }
value = 'vowel'
vowels = dict.fromkeys(keys, value)
print(vowels)
```

8. update()

The method **update()** adds dictionary dict2's key-values pairs in to dict. This function does not return anything.

For ex.

```
dict = {'Name': 'Zara', 'Age': 7}
dict2 = {'Gender': 'female' }
dict.update(dict2)
print ("The updated value :",dict)
```

ADVANCED LIST PROCESSING

LIST COMPREHENSION

Python supports a concept called "list comprehensions".It can be used to construct lists in a very natural,easy way,like a mathematics is used to do.

In python, we can write the expression almost exactly like a mathematics with special cryptic syntax.

For example:

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

The above code creates a list of square numbers from 0 to 9.This list can be created using a simplified expression "X**2 for x in range (10)".This is basically list comprehension.

The list comprehension is achieved using the tools like map, lambda, reduce and filter

MAP:

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

Syntax: result=map(function,sequence)

Example:

```
>>>def increment_by_three(x):
        return x+3
>>>new_list=list(map(increment_by_three,range(10)))
>>>new_list
```

OUTPUT:

[3,4,5,6,7,8,9,10,11,12]

Another example: There is a build in function for calculating length of a string and and i.e len().We can pass this function to map and find the length of every element in the list.

```
>>> names=['Sandip','Lucky','Mahesh','Sachin']
>>>lengths=list(map(len,names))
>>>lengths
    [6,5,6,6]
```

FILTER:

The filter() function selects some of the elements and filters out others .If we use it wit list comprehension,it is almost equivalent to the filter build-in. This function takes two arguments.

Syntax: result=filter(function,sequence)

for example:

```
>>>def even_fun(x):
```

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

In above example,using filter() function the odd numbers are filtered out and only even numbers are displayed.

LAMBDA:

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

Syntax: lamda argument_list:expression

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

Example:
```
>>>sum=lambda x,y:x+y
    >>>sum(1,3)
        4
```

REDUCE:

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

```
Result=reduce(function,sequence)
```

The function reduce() had been dropped from core of python when migratin to python 3 Hence for using the reduce function we have to import functools to be capable of using reduce.\

```
>>>import functools
>>>functools.reduce(lambda x,y:x+y[1,2,3,4])
    10
```

ILLUSTRATIVE PROGRAMS

1.Selection Sort

```
source = [10,90,30,20]
for i in range(len(source)):
    mini = min(source[i:])
    min_index = source[i:].index(mini)
    source[i + min_index] = source[i]
    source[i] = mini
print(source)
```

2.Insertion Sort

```
def insertionSort(alist):
   for index in range(1,len(alist)):
      currentvalue = alist[index]
      position = index
      while position>0 and alist[position-
1]>currentvalue:
           alist[position]=alist[position-1]
           position = position-1
       alist[position]=currentvalue

alist=[12,34,67,30,82]
insertionSort(alist)
print(alist)
```

3.Histogram

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

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

4. Merge Sort

```
def mergeSort(alist):
    print("Splitting ",alist)
    if len(alist)>1:
        mid = len(alist)//2
        lefthalf = alist[:mid]
        righthalf = alist[mid:]
        mergeSort(lefthalf)
        mergeSort(righthalf)
        i=0
        j=0
        k=0
        while (i < len(lefthalf)) and (j < len(righthalf)):
            if lefthalf[i] < righthalf[j]:
                alist[k]=lefthalf[i]
                i=i+1
            else:
                alist[k]=righthalf[j]
                j=j+1
            k=k+1
        while i < len(lefthalf):
            alist[k]=lefthalf[i]
            i=i+1
            k=k+1
        while j < len(righthalf):
            alist[k]=righthalf[j]
            j=j+1
            k=k+1
    print("Merging ",alist)
n = input("Enter the size of the list: ")
alist = [30,50,10,70]
mergeSort(alist)
print(alist)
```

5.Sum of array of numbers

```
sum=0
i=0
a=[10,20,30,40]
for i in range(4):
        sum=sum+a[i]
print(sum)
```

PART- A (2 Marks)

1. What is a list?(Jan-2018)

A list is an ordered set of values, where each value is identified by an index. The values that make up a list are called its elements. Lists are similar to strings, which are ordered sets of characters, except that the elements of a list can have any type.

2. Relate String and List? (Jan 2018)(Jan 2019) String:

String is a sequence of characters and it is represented within double quotes or single quotes. Strings are immutable.

Example: s=”hello”

List:

A list is an ordered set of values, where each value is identified by an index. The values that make up a list are called its elements. Lists are similar to strings, which are ordered sets of characters, except that the elements of a list can have any type and it is mutable.

Example:

b= [’a’, ’b’, ’c’, ’d’, 1, 3]

3. Solve a)[0] * 4 and b) [1, 2, 3] * 3.

```
>>> [0] * 4
[0, 0, 0, 0]
>>> [1, 2, 3] * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

4. Let list = [’a’, ’b’, ’c’, ’d’, ’e’, ’f’]. Find a) list[1:3] b) t[:4] c) t[3:] .

```
>>> list = [’a’, ’b’, ’c’, ’d’, ’e’, ’f’]
>>> list[1:3]
[’b’, ’c’]
>>> list[:4]
[’a’, ’b’, ’c’, ’d’]
>>> list[3:]
[’d’, ’e’, ’f’]
```

5. Mention any 5 list methods.

- append()
- extend ()
- sort()
- pop()
- index()
- insert
- remove()

6. State the difference between lists and dictionary.

Lists	Dictionary
• List is a mutable type meaning that it can be modified. • List can store a sequence of objects in a certain order. • **Example:** list1=[1,’a’,’apple’]	• Dictionary is immutable and is a key value store. • Dictionary is not ordered and it requires that the keys are hashable. • **Example:** dict1={‘a’:1, ‘b’:2}

7. What is List mutability in Python? Give an example.

Python represents all its data as objects. Some of these objects like **lists** and dictionaries are **mutable**, i.e., their content can be changed without changing their identity. Other objects like integers, floats, strings and tuples are objects that cannot be changed.

Example:

```
>>> numbers = [17, 123]
>>> numbers[1] = 5
>>> print numbers [17, 5]
```

8. What is aliasing in list? Give an example.

An object with more than one reference has more than one name, then the object is said to be aliased.

Example: If *a* refers to an object and we assign b = a, then both variables refer to the same object:

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

9. Define cloning in list.

In order to modify a list and also keep a copy of the original, it is required to make a copy of the list itself, not just the reference. This process is sometimes called cloning, to avoid the ambiguity of the word "copy".

Example:

```
def Cloning(li1): li_copy = li1[:] return li_copy
# Driver Code
li1 = [4, 8, 2, 10, 15, 18]
li2 = Cloning(li1) print("Original List:", li1) print("After Cloning:", li2)
```

Output:

```
Original List: [4, 8, 2, 10, 15, 18]
After Cloning: [4, 8, 2, 10, 15, 18]
```

10. Explain List parameters with an example.

Passing a list as an argument actually passes a reference to the list, not a copy of the list. For example, the function head takes a list as an argument and returns the first element:

Example:

```
def head(list):
return list[0]
```

Here's how it is used:

```
>>> numbers = [1, 2, 3]
>>> head(numbers)
>>> 1
```

11. Write a program in Python returns a list that contains all but the first element of the given list.

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

Here's how tail is used:

```
>>> numbers = [1, 2, 3]
>>> rest = tail(numbers)
>>> print rest [2, 3]
```

12. Write a program in Python to delete first element from a list.

def deleteHead(list): del list[0]

Here's how deleteHead is used:

>>> numbers = [1, 2, 3]

>>> deleteHead(numbers)

>>> print numbers [2, 3]

13. What is the benefit of using tuple assignment in Python?

It is often useful to swap the values of two variables. With conventional assignments a temporary variable would be used.

For example, to swap a and b:

>>> temp = a

>>> a = b

>>> b = temp

This solution is cumbersome; tuple assignment is more elegant:

>>> a, b = b, a

14. Define key-value pairs.

The elements of a dictionary appear in a comma-separated list. Each entry contains an index and a value separated by a colon. In a dictionary, the indices are called keys, so the elements are called key-value pairs.

15. Define dictionary with an example.

A dictionary is an associative array (also known as hashes). Any key of the dictionary is associated (or mapped) to a value. The values of a dictionary can be any Python data type. So dictionaries are unordered key-value-pairs.

Example:

>>> eng2sp = {} # empty dictionary

>>> eng2sp['one'] = 'uno'

>>> eng2sp['two'] = 'dos'

16. How to return tuples as values?

A function can only return one value, but if the value is a tuple, the effect is the same as returning multiple values. For example, if you want to divide two integers and compute the quotient and remainder, it is inefficient to compute x/y and then x%y. It is better to compute them both at the same time.

>>> t = divmod(7, 3)

>>> print t (2, 1)

17. List two dictionary operations.

- Del -removes key-value pairs from a dictionary
- Len - returns the number of key-value pairs

18. Define dictionary methods with an example.

A method is similar to a function. It takes arguments and returns a value but the syntax is different. For example, the keys method takes a dictionary and returns a list of the keys that appear, but instead of the **function syntax** keys(dictionary_name), **method syntax** dictionary_name.keys() is used.

Example:>>> eng2sp.keys() ['one', 'three', 'two']

19. Define List Comprehension.

List comprehensions apply an arbitrary expression to items in an iterable rather than applying function. It provides a compact way of mapping a list into another list by applying a function to each of the elements of the list.

20. Write a Python program to swap two variables.

```
x = 5
y = 10
x,y=y,x
print('The value of x after swapping: {}'.format(x)) print('The value of y after swapping: {}'.format(y))
```

21. Write the syntax for list comprehension.

The list comprehension starts with a '**[' and ']'**, to help you remember that the result is going to be a list. The basic syntax is[expression for item in list if conditional].

Example:

```
new_list = [] for i in old_list:
if filter(i): new_list.append(expressions(i))
```

22. How list differs from tuple. (Jan-2018)

List	Tuple
• List is a mutable type meaning that it can be modified. • **Syntax:** list=[] • **Example:** list1=[1,'a']	• Tuple is an immutable type meaning that it cannot be modified. • **Syntax:** tuple=() • **Example:** tuple1=(1,'a')

23. How to slice a list in Python. (Jan-2018)

The values stored in a list can be accessed using slicing operator, the colon (:) with indexes starting at 0 in the beginning of the list and end with -1.

Example:

```
>>> list = ['a', 'b', 'c', 'd', 'e', 'f']
>>> list[1:3]
['b', 'c']
```

24. Write python program for swapping two numbers using tuple assignment?

```
a=10 b=20
a,b=b,a
print("After swapping a=%d,b=%d"%(a,b))
```

25. What is list loop?

In Python lists are considered a type of iterable . An iterable is a data type that can return its elements separately, i.e., one at a time.

Syntax:

```
for <item> in <iterable>:
    <body>
```

Example:

```
>>>names = ["Uma","Utta","Ursula","Eunice","Unix"]
>>>for name in names:
print("Hi "+ name +"!")
```

26. What is mapping?

A list as a relationship between indices and elements. This relationship is called a **mapping**; each index "maps to" one of the elements.The in operator also works on lists.

```
>>> cheeses = ['Cheddar', 'Edam', 'Gouda']
>>> 'Edam' in cheeses True
>>> 'Brie' in cheeses False
```

27. Give a function that can take a value and return the first key mapping to that value in a dictionary.(Jan 2019)

```
a={'aa':2, "bb":4}
print(a.keys()[0])
```

28. How to create a list in python? Illustrate the use of negative indexing of list with example. (May 2019)

List Creation:

```
days = ['mon', 2]
days=[] days[0]='mon' days[1]=2
```

Negative Indexing:

Example:

```
>>> print(days[-1])
```

Output: 2

29. Demonstrate with simple code to draw the histogram in python. (May 2019)

```
def histogram( items ):
    for n in items:
        output = '' times = n
        while( times > 0 ):
            output += '*' times = times - 1 print(output)
histogram([2, 3, 6, 5])
```

Output:

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

30. How will you update list items? Give one example. (Nov / Dec 2019)

Using the indexing operator (square brackets) on the left side of an assignment, we can update one of the list items

```
fruit = ["banana","apple","cherry"] print(fruit)
['banana', 'apple', 'cherry'] fruit[0] = "pear"
fruit[-1] = "orange" print(fruit)
['pear', 'apple', 'orange']
```

31. Can function return tuples? If yes Give examples. (Nov / Dec 2019)

Function can return tuples as return values.

```
def circle_Info(r):
#Return circumference and area and tuple c = 2 * 3.14 * r
a = 3.14 * r * r return(c,a)
print(circle_Info(10))
```

Output

(62.800000000000004, 314.0)

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).

FILES AND EXECPTION

Definition:

Files is a named location on disk to store related information, settings, or commands in secondary storage device like magnetic disks, magnetic tapes and optical disks.

Types of Files:

There are two types of files:

Text files

Binary files

Text files:

Text files are sequence of lines or sequence of character in text format. Each line is terminated with a special character, called EOL or end of line character.

Binary files:

Binary files is any type of files other than a text files.

File Operation:

1. Open()
2. read()
3. write()
4. close()

Opening a file:

This function creates a file object, which would be utilized to call other support methods associates with it.

Syntax : file_object= open('filename' , 'mode')

Here,

Filename: The file name argument is a string value that contains the name of the file that you want to access.

Mode: The access mode determines the mode in which the files have to be opened, (i.e)., read, write, append etc.

Modes	Description
r	Open a file for reading only. The file pointer is placed at the beginning of the file. This is default mode.
rb	Open a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is default mode.
r+	Open a file for both read and write. The file pointer is placed at the beginning of the file.
rb+	Open a file for both read and write in binary format. The file pointer is placed at the beginning of the file
w	Open a file for writing only. Overwrite the file if exists. If the files does not exists, creates a new file for writing.
wb	Open a file for writing only in binary format. Overwrite the file if exists. If the files does not exists, creates a new file for writing.
w+	Open a file for both read and write. Overwrite the file if exists. If the files does not exists, creates a new file for writing and reading.
wb+	Open a file for both read and write in binary format. Overwrite the file if exists. If the files does not exists, creates a new file for writing and reading.
a	Open a file for appending. This file pointer is placed at the end of the file.
ab	Open a file for appending only in binary format. This file pointer is placed at the end of the file, if exists. That is, the file is in the appending mode. If the files does not exists, creates a new file for writing.
a+	Open a file for appending and reading. This file pointer is placed at the end of the file, if exists.
ab+	Open a file for appending and reading in binary format. This file pointer is placed at the end of the file, if exists. It creates a new file for reading and writing.

Example: fn=open('D:/ex.txt','r')

Writing into a file:

write() is used to write a string to an already opened files. To write into a file, it is needed to open a file in write 'w', append 'a' or exclusive 'x' mode.

```
Syntax :
filevariable.write("filename","mode")
```

```
Example:
    f=open('D:/ex.txt','w')
    f.write("welcome\n")
    f.write("thank you")
    f.close()
```

New file is created in the name "test.txt" and content is written

Welcome

Thank you

Use of with statement:

- ✓ Because every call on function open should have a corresponding call on method close, python provides a **with statement** that automatically closes a file when the end of the block is reached.

Syntax:

with open (filename, mode) as variable :
block

Example:

```
with open ("file.txt", "r") as file:
    contents=file.read( )
Print(contents)
```

Methods of writing a file:

1. Write() – writes a single line into the specified file.
2. Writelines()- writes multiple line into specified file.

Writelines():

The writelines method put multiple data into the file.

The writelines() method writes any string to an open file.

Syntax: filevariable.writelines(string)

Example.py:

```
f=open('D:/ex.txt','w')
str='this is my book\n I found it here'
f.writelines(str)
f.close()
```

Output: This is my book I found it here

Reading a file:

This method helps to just view the content of given input files. To read the content of a file, one must open the file in reading mode.

Syntax: filevariable=open('filename','r')

Methods used in reading a file.

1. read(size)
2. Readline()
3. Readlines()

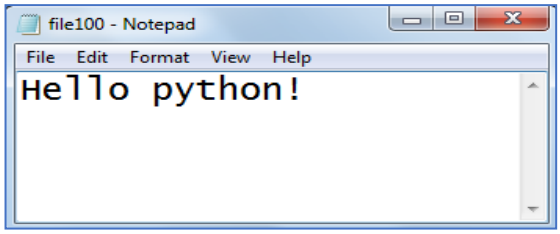
file.read(size)

This read() specifies the size of data to be read from input files.
If size not mentioned, it reads the entire files and cursor waits in last position of files.

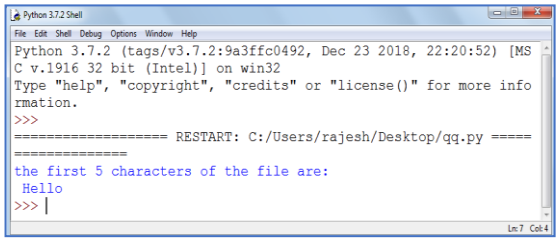
Ex.

```
f=open("file100.txt", "r")
print("The first 5 characters of the file are:\n",  f.read(5))
f.close()
```

The content of the file100 is:



OUTPUT:



Readline():

This method is used to read a single line from the input file, till the new line character occur. It doesn't takes any argument.

Syntax: f.readline()

ex.

```
f=open("file.txt","r")
print("reading content of the file by realine() method:\n")
print(f.readline())
f.close()
```

Readlines():

This method is used to read and display all the line from the input file.

Syntax:
f.readlines()

ex.

```
f=open("file.txt","r")
print("reading content of the file by realines() method:\n")
print(f.readlines())
f.close()
```

File object attributes:

name: Return the name of the file. It is a read-only attribute and may not be present on all file-like objects. If the file object was created using the open() function, the file's name is returned. Otherwise, some string indicates the source of the file object is returned.

encoding: It returns the encoding this file uses, such as UTF-8. This attribute is read-only. When Unicode strings are written to a file, they will be converted to byte strings using this encoding. It may also be None. In that case, the file uses the system default encoding for converting Unicode strings.

mode: Returns the file access mode used while opening a file.

closed: Returns True if a file is closed. It is a Boolean value indicating the current state of the file object.

newline: Files opened in universal newline read mode keep track of the newlines encountered while reading the file. The values are '\r', '\n', '\r\n', None (no newlines read yet), or a tuple containing all the newline types seen. For files not opened in universal newline read mode, the value of this attribute will be None.

Closing a file:

The close() methods of a file object flushes any unwritten information and closes the file object, after which no more writing can be done.

Syntax:
filevariable.close()

Example.py:

```
Open('input.txt','wb')
Print("name of the file:",f.name)
f.close()
```

Output:
Name of the file: input.txt

Example: Python program to implement all file read operation:

```
fn=open('D:/ex.txt','r')
print(fn.read())
fn.seek(0)
print(fn.read(4))
print(fn.tell())
fn.seek(0)
print(fn.readline())
fn.seek(0)
print(fn.readlines())
```

Output:
Welcome to the world of robotics
Welc
4
Welcome to the world of robotics
Welcome to the world of robotics and automation

ex.txt:

Welcome to the world of robotics and automation

FILE METHODS

Methods	Description
File.close()	Close the file. A closed file cannot be read or write anymore.
File.flush()	Flush the internal buffer, like stdio's fflush. This may be a no-op on some file like objects.
File.fileno()	Return the integer file description that is used by underlying implementation to request I/O operation from os.
File.isatty()	Return True if the file is connected to a tty(-like) devices, else False.
File.read([size])	Read at most size bytes from files (less if the read hits EOF before obtaining size bytes)
File.readline([size])	Read one entries line from the file. A trailing newline character is kept in the string.
File.readlines([sizehint])	Read until EOF using readline() and return a list containing the lines. If the optional sizehint of argument is present, instead of reading up to EOF, whole line totaling approximately sizehint bytes are read.
File.seek(offset[,whence])	Set the files current position.
File.tell()	Return the files current position
File.truncate([size])	Truncates the files size. If the optional size argument is present, the file is truncated to that size.
File.write(str)	Writes a string to the files. There is no return value.
File.writelines(sequence)	Writes a sequence of string to a file. The sequences can be any iterable. Object producing string, typically a list of strings.
File.readable()	Return true if file stream can be read from.
File.seekable()	Return true if file stream supports random access.
File.writable(0	Return true if file stream can be written to.

Tell() and seek():

Tell() method display the current position of cursor from the input files.

Seek() takes an argument and moves the cursor to the specified position which is mentioned as argument.

Syntax: print(f.tell())
print(f.seek())

Example:

```
fn=open('D:/ex.txt','r')
print(fn.read())
fn.seek(0)
print(fn.read(4))
print(fn.tell())
fn.seek(0)
print(fn.readline())
fn.seek(0)
print(fn.readlines())
```

Output:
Welcome to the world of robotics
Welc
4
Welcome to the world of robotics
Welcome to the world of robotics and automation

ex.txt:

Welcome to the world of robotics and automation

FORMAT OPERATOR

String formatting is the process of infusing things in the strings dynamically and presenting the strings.

There are four different ways to perform string formatting.

1. Formatting with % operator
2. Formatting with format() string methods.
3. Formatting with string literals, called f-strings.
4. String template class

1. Formatting with % operator:

It's the oldest method of string formatting. Here we use the modulo % operator. The modulo % is also known as the **"string formatting operator"**.

'%s' is used to inject strings.
'%d' is used to integer
'%f' for floating point values.
'%b' for binary format.

Example:

```
Print('joe stood up and %s to the crowd.'%'spoke')
Print('there are %d dogs.'%4)
Print('floating point numbers:%.2f' %(13.144))
```

Output:

```
Joe stood up and spoke to the crowd
There are 4 dogs
Floating point numbers:13.14
```

2. Formatting with format() string method:

Format() method was introduced with Python3 for handling complex string formatting more efficiently. Formatters work by putting in one or more replacement fields and placeholders defined by a pair of curly braces {} into a string and calling the str.format().

```
Syntax: 'String here {} then also
{}'.format('something1','something2')
```

Example:

```
print('We all are {}.'.format('equal'))
```

Output: We all are equal.

The **format()** method has many advantages over the placeholder method:

- We can insert object by using index-based position:

  Example: print('{2} {1} {0}'.format('directions','the', 'Read'))
  Output: Read the directions.

- We can insert objects by using assigned keywords:

  Example: print('a: {a}, b: {b}, c: {c}'.format(a = 1, b = 'Two',c = 12.3))
  Output: a: 1, b: Two, c: 12.3

- We can reuse the inserted objects to avoid duplication:

  Example: print ('The first {p} was alright, but the {p} {p} was tough.'.format(p = 'second'))
  Output: The first second was alright, but the second second was tough.

Float precision with the format() method:

Syntax: {[index]:[width][.precision][type]}

The type can be used with format codes:

- **'d'** for integers
- **'f'** for floating-point numbers
- **'b'** for binary numbers
- **'o'** for octal numbers
- **'x'** for octal hexadecimal numbers
- **'s'** for string
- **'e'** for floating-point in an exponent format

Example:

```
print('The value of pi is: %1.5f' %3.141592)        # vs.
print('The value of pi is: {0:1.5f}'.format(3.141592))
```

Output:

```
The value of pi is: 3.14159
The value of pi is: 3.14159
```

COMMAND LINE ARGUMENT

The arguments that are given after the name of the program in the command line shell of the operating system are known as **Command Line Arguments**. Python provides various ways of dealing with these types of arguments. The three most common are:

1. Using sys.argv
2. Using argparse module

1.Using sys.argv:

The sys module provides functions and variables used to manipulate different parts of the Python runtime environment. This module provides access to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter.

One such variable is sys.argv which is a simple list structure. It's main purpose are:

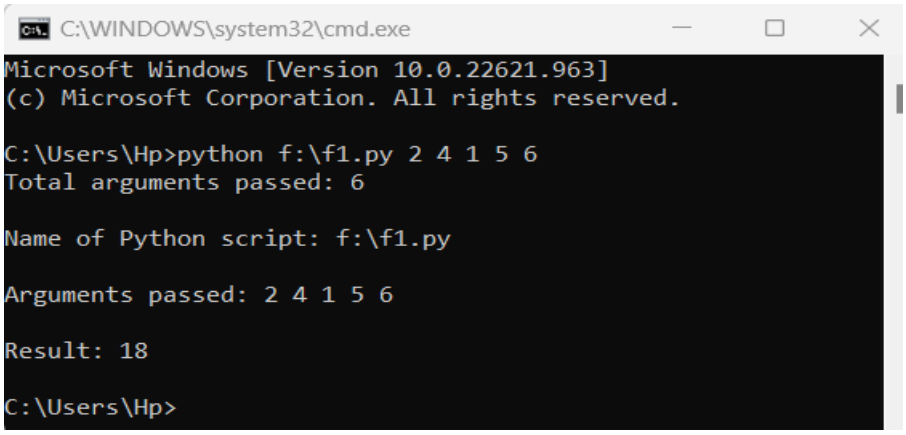
- It is a list of command line arguments.
- len (sys.argv) provides the number of command line arguments.
- sys.argv[0] is the name of the current Python script.

Example**:** Let's suppose there is a Python script for adding two numbers and the numbers are passed as command-line arguments.

```
# Python program to demonstrate command line arguments
import sys

# total arguments
n = len (sys.argv)
print ("Total arguments passed:", n)

# Arguments passed
print("\nName of Python script:", sys.argv[0])
print("\nArguments passed:", end = " ")
for i in range(1, n):
    print(sys.argv[i], end = " ")

# Addition of numbers using argparse module
```

```
Sum = 0
for i in range(1, n):
        Sum += int(sys.argv[i])
print("\n\nResult:", Sum)
```

Output:



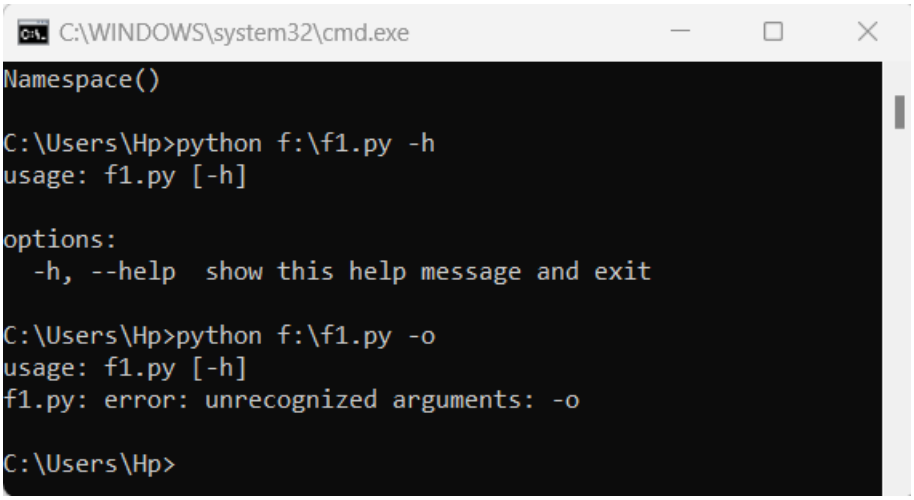
2. Using argparse module:

Using argparse module is a better option than the above two options as it provides a lot of options such as positional arguments, default value for arguments, help message, specifying data type of argument etc.

Example:

```
import argparse
# Initialize parser
parser = argparse.ArgumentParser()
parser.parse_args()
```

Output:



ERRORS AND EXCEPTION

Exception:

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

When a Python code comes across a condition it can't handle, it raises an exception. An object in Python that describes an error is called an exception.

They are mainly two types of errors:

1. Syntax errors :

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

(i) Putting a keyword at wrong place.

(ii) Misspelling the keyword

(iii) Incorrect Indentation

(iv) Forgetting symbols such as comma, colon, brackets, quotes

(v) Empty blocks

2. Runtime errors:

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

Eg of runtime error are:

1) Trying to access a file which does not exist.

2) Using an identifier which is not defined.

3) Performing operations of incompatible type elements

4) Division by Zero

Such errors are handled using exception handling.

S.No.	Name of the Exception	Description of the Exception
1	**Exception**	All exceptions of Python have a base class.
2	**StopIteration**	If the next() method returns null for an iterator, this exception is raised.
3	**SystemExit**	The sys.exit() procedure raises this value.
4	**StandardError**	Excluding the StopIteration and SystemExit, this is the base class for all Python built-in exceptions.

5	**ArithmeticError**	All mathematical computation errors belong to this base class.
6	**OverflowError**	This exception is raised when a computation surpasses the numeric data type's maximum limit.
7	**FloatingPointError**	If a floating-point operation fails, this exception is raised.
8	**ZeroDivisionError**	For all numeric data types, its value is raised whenever a number is attempted to be divided by zero.
9	**AssertionError**	If the Assert statement fails, this exception is raised.
10	**AttributeError**	This exception is raised if a variable reference or assigning a value fails.
11	**EOFError**	When the endpoint of the file is approached, and the interpreter didn't get any input value by raw_input() or input() functions, this exception is raised.
12	**ImportError**	This exception is raised if using the import keyword to import a module fails.
13	**KeyboardInterrupt**	If the user interrupts the execution of a program, generally by hitting Ctrl+C, this exception is raised.
14	**LookupError**	LookupErrorBase is the base class for all search errors.
15	**IndexError**	This exception is raised when the index attempted to be accessed is not found.
16	**KeyError**	When the given key is not found in the dictionary to be found in, this exception is raised.
17	**NameError**	This exception is raised when a variable isn't located in either local or global namespace.
18	**UnboundLocalError**	This exception is raised when we try to access a local variable inside a function, and the variable has not been assigned any value.
19	**EnvironmentError**	All exceptions that arise beyond the Python environment have this base class.
20	**IOError**	If an input or output action fails, like when using the print command or the open() function to access a file that does not exist, this exception is raised.
22	**SyntaxError**	This exception is raised whenever a syntax error occurs in our program.
23	**IndentationError**	This exception was raised when we made an improper indentation.

24	**SystemExit**	This exception is raised when the sys.exit() method is used to terminate the Python interpreter. The parser exits if the situation is not addressed within the code.
25	**TypeError**	This exception is raised whenever a data type-incompatible action or function is tried to be executed.
26	**ValueError**	This exception is raised if the parameters for a built-in method for a particular data type are of the correct type but have been given the wrong values.
27	**RuntimeError**	This exception is raised when an error that occurred during the program's execution cannot be classified.
28	**NotImplementedError**	If an abstract function that the user must define in an inherited class is not defined, this exception is raised.

Handling an exception

When it raises an exception, it must either handle the exception immediately else it terminates and quits. The exception handling mechanism uses the try….except…..else blocks.

try: includes suspicious code.

After the try: block, include an **except:** statement, followed by a block of code which handles the problem as elegantly as possible.

Syntax

Here is simple syntax of *try....except...else* blocks −

```
try:
   write the suspicious code here;
   ......................
except ExceptionI:
   If there is ExceptionI, then execute this block.
except ExceptionII:
   If there is ExceptionII, then execute this block.
   ......................
else:
   If there is no exception then execute this block.
```

Note:

1. A single try statement can have multiple except statements. This is useful when the try block contains statements that may throw different types of exceptions.
2. You can also provide a general except clause, which handles any exception.
3. After the except clause(s), you can include an else-clause. The code in the else-block executes if the code in the try: block does not raise an exception.
4. The else-block is a good place for code that does not need the try: block's protection.

Example 1:

```
try:
   fh = open("testfile", "w")
   fh.write("This is my test file for exception handling!!")
except IOError:
```

```
  print "Error: can\'t find file or read data"
else:
  print "Written content in the file successfully"
fh.close()
```
Output:
Written content in the file successfully

Example 2:
```
try:
  fh = open("testfile", "r")
  fh.write("This is my test file for exception handling!!")
except IOError:
  print "Error: can\'t find file or read data"
else:
  print "Written content in the file successfully"
```
Output:
Error: can't find file or read data

The except Clause with No specific Exceptions:

You can also use the except statement with no exceptions defined as follows –

Syntax:
```
try:
  You do your operations here;
  ......................
except:
  If there is any exception, then execute this block.
  ......................
else:
  If there is no exception then execute this block.
```

Example:
```
try:
   n=int(input("Enter some number"))
except
      print("you have entered wrong data")
else:
       print("You have entered:",n)
```
Output:
```
Enter some number    a
You have entered a wrong data
Enter some number   10
You have entered :10
```

Note: This kind of a **try-except** statement catches all the exceptions that occur.

The except Clause with Multiple Exceptions:

You can also use the same except statement to handle multiple exceptions as follows −

Syntax:
```
try:
  You do your operations here;
  ......................
```

```
except(Exception1[, Exception2[,...ExceptionN]]]):
   If there is any exception from the given exception list,
   then execute this block.
   ......................
else:
   If there is no exception then execute this block.
```

Example:

```
try:
    a=int(input("Enter the value of a"))
    b=int(input("Enter the value of b"))
    c=a/b
except value error:
            print ("You have entered wrong data")
except Zerodivision error:
            print ("Divide by zero error!!!")
else:
            print ("The result",c)
```

Output:

```
Enter the value of a:10
Enter the value of b:a
You have entered wrong data
```

The try-finally Clause:

You can use a **finally:** block along with a **try:** block. The finally block is a place to put any code that must execute, whether the try-block raised an exception or not.

The syntax of the try-finally statement is this −

Syntax:

```
try:
   You do your operations here;
   ......................
   Due to any exception, this may be skipped.
finally:
   This would always be executed.
   ......................
```

Example:

```
try:
   fh = open("testfile", "w")
   fh.write("This is my test file for exception handling!!")
finally:
   print "Error: can\'t find file or read data"
```

Output:

```
Error: can't find file or read data
```

Raising an Exceptions:

You can raise exceptions in several ways by using the raise statement. The general syntax for the **raise** statement is as follows.

Syntax

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

Here, *Exception* is the type of exception (for example, NameError) and *argument* is a value for the exception argument. The argument is optional; if not supplied, the exception argument is None.

The final argument, traceback, is also optional (and rarely used in practice), and if present, is the traceback object used for the exception.

Example:

```
def functionName( level ):
  if level < 1:
    raise "Invalid level!", level
    # The code below to this would not be executed
    # if we raise the exception
```

User-Defined Exceptions:

Python also allows you to create your own exceptions by deriving classes from the standard built-in exceptions.Here is an example related to RuntimeError. Here, a class is created that is subclassed from RuntimeError. This is useful when you need to display more specific information when an exception is caught.

In the try block, the user-defined exception is raised and caught in the except block. The variable e is used to create an instance of the class Networkerror.

```
class Networkerror(RuntimeError):
  def __init__(self, arg):
    self.args = arg
```

So once you defined above class, you can raise the exception as follows −

```
try:
  raise Networkerror("Bad hostname")
except Networkerror,e:
  print e.args
```

MODULES

A Python module is a file containing Python definitions and statements. A module can define functions, classes, and variables. A module can also include runnable code. Grouping related code into a module makes the code easier to understand and use.

Example:

```
def add(x, y):
        return (x+y)
def subtract(x, y):
        return (x-y)
```

Import Module in Python

We can import the functions, and classes defined in a module to another module using the **import statement** in some other Python source file.

Syntax

```
import module
```

Example:

```
# importing  module calc.py
import calc
print(calc.add(10, 2))
```

Output: 12

The from-import Statement in Python

Python's from statement lets you import specific attributes from a module without importing the module as a whole.

Example:

```
from math import sqrt, factorial
print(sqrt(16))
print(factorial(6))
```

Output:

```
4.0
720
```

Import all Names :

The * symbol used with the from import statement is used to import all the names from a module to a current namespace.

Syntax:

```
from module_name
import *
```

Example:

```
from math import *
print(sqrt(16))
print(factorial(6))
```

Output: 4.0
720

Renaming the Python module:

We can rename the module while importing it using the keyword.

Syntax: Import Module_name as Alias_name
Example: # importing sqrt() and factorial from the

```
# module math
import math as mt
print(mt.sqrt(16))
print(mt.factorial(6))
```

Python built-in modules

There are several built-in modules in Python, which you can import whenever you like.

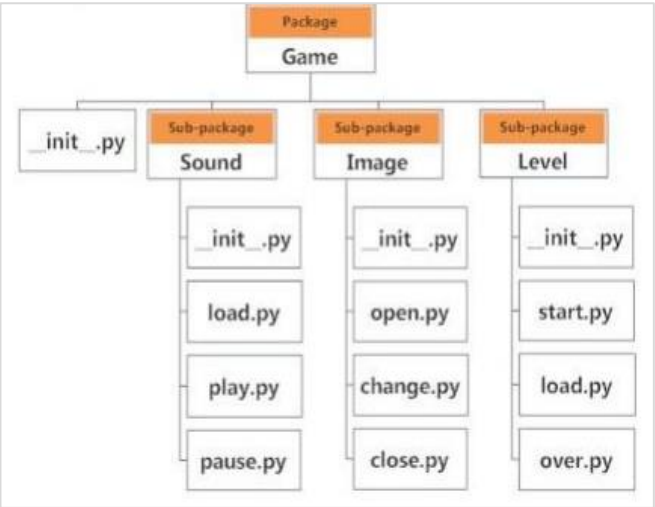
```
Example:
import math
print(math.sqrt(25))
print(math.pi)
print(math.degrees(2))
print(math.radians(60))
print(math.sin(2))
print(math.cos(0.5))
print(math.tan(0.23))
print(math.factorial(4))
```

```
import random
print(random.randint(0, 5))   # printing random integer between 0 and 5
print(random.random())        # print random floating point number between 0 and 1
print(random.random() * 100)          # random number between 0 and 100
List = [1, 4, True, 800, "python", 27, "hello"]
print(random.choice(List))
```

```
import datetime
from datetime import date
import time
# Returns the number of seconds since the
# Unix Epoch, January 1st 1970
print(time.time())
# Converts a number of seconds to a date object
print(date.protestantisme(454554))
```

Packages

A **package** is a collection of modules. A Python package can have sub-packages and modules. A directory must contain a file named __init__.py in order for Python to consider it as a package. This file can be left empty but we generally place the initialization code for that package in this file.



Importing module from a package:

We can import modules from packages using the dot (.) operator.

For example, if want to import the start module in the above example, it is done as follows. import Game.Level.start. Now if this module contains a function named select_difficulty(), we must use the full name to reference it.

Game.Level.start.select_difficulty(2)

If this construct seems lengthy, we can import the module without the package prefix as follows.

from Game.Level import start

We can now call the function simply as follows.

start.select_difficulty(2)

Yet another way of importing just the required function (or class or variable) form a module within a package would be as follows.

From Game.Level.start

import select_difficulty

Now we can directly call this function.

select_difficulty(2)

Although easier, this method is not recommended. Using the full namespace avoids confusion and prevents two same identifier names from colliding.

While importing packages, Python looks in the list of directories defined in sys.path, similar as for module search path.

ILLUSTRATIVE PROGRAM

1. Word Count

```
import sys
file=open("/Python27/note.txt","r+")
wordcount={}
for word in file.read().split():
if word not in wordcount:
wordcount[word] = 1
else:
wordcount[word] += 1
file.close();
print ("%-30s %s " %('Words in the File' , 'Count'))
for key in wordcount.keys():
print ("%-30s %d " %(key , wordcount[key]))
```

2.copy files:

```
def copyFile(oldFile, newFile):
f1 = open(oldFile, "r")
f2 = open(newFile, "w")
while True:
text = f1.read(50)
if text == "":
break
f2.write(text)
f1.close()
f2.close()
return
```

2 Mark Questions and Answers

1.What is a text file?

A text file is a file that contains printable characters and whitespace, organized in to lines separated by newline characters.

2.Write a python program that writes "Hello world" into a file.

```
f =open("ex88.txt",'w')
f.write("hello world")
f.close()
```

3.Write a python program that counts the number of words in a file.

```
f=open("test.txt","r")
content =f.readline(20)
words =content.split()
print(words)
```

4.What are the two arguments taken by the open() function?

The open function takes two arguments : name of the file and the mode of operation.

Example:

```
f = open("test.dat","w")
```

5.What is a file object?

A file object allows us to use, access and manipulate all the user accessible files. It maintains the state about the file it has opened.

Example:

```
f = open("test.dat","w") // f is the file object.
```

6.What information is displayed if we print a file object in the given program?

```
f= open("test.txt","w")
print f
```

The name of the file, mode and the location of the object will be displayed.

7.What is an exception?

Whenever a runtime error occurs, it creates an exception. The program stops execution and prints an error message.

Example:

#Dividing by zero creates an exception: print 55/0
ZeroDivisionError: integer division or modulo

8.What are the two parts in an error message?

The error message has two parts: the type of error before the colon, and specification about the error after the colon.

Example:

>>> 10 * (1/0)
Traceback (most recent call last):
File "<stdin>", line 1, in ?
ZeroDivisionError: integer division or modulo by zero

9.What are the error messages that are displayed for the following exceptions?

1. Accessing a non-existent list item
2. Accessing a key that isn't in the dictionary
3. Trying to open a non-existent file
4. IndexError: list index out of range
5. KeyError: what
6. IOError: [Errno 2] No such file or directory: 'filename'

10.How do you handle the exception inside a program when you try to open a non-existent file?

filename = raw_input('Enter a file name: ') try:
f = open (filename, "r") except IOError:
print 'There is no file named', filename

11. How does try and execute work?

The try statement executes the statements in the first block. If no exception occurs, then except statement is ignored. If an exception of type IOError occurs, it executes the statements in the except branch and then continues.

Example:

```
try:
   print "Hello World" except:
   print "This is an error message!"
```

12.What is the function of raise statement? What are its two arguments?

The raise statement is used to raise an exception when the program detects an error. It takes two arguments: the exception type and specific information about the error.

13.What is a pickle?

Pickling saves an object to a file for later retrieval. The pickle module helps to translate almost any type of object to a string suitable for storage in a database and then translate the strings back in to objects.

14.What is the use of the format operator?

The format operator % takes a format string and a tuple of expressions and yields a string that includes the expressions, formatted according to the format string.

Example:

>>> nBananas = 27
>>> "We have %d bananas." % nBananas 'We have 27 bananas.'

15.What are the two methods used in pickling?

The two methods used in pickling are,

1. pickle.dump()
2. pickle.load().

To store a data structure, dump method is used and to load the data structures that are dumped, load method is used.

16.What are modules?(or) Write a note on modular design (Jan-2018)

- Modules are files containing Python definitions and statements (ex: name.py)
- Modules can contain executable statements along with function definitions.
- Each modules has its own private symbol table used as the global symbol table all functions in the module.
- Modules can import other modules.

17.What is a package?

Packages are namespaces that contain multiple packages and modules themselves. They are simply directories.

Example:

```
from Game.Level.start
import select_difficulty
```

18.Write a Python script to display the current date and time. (Jan-2018)

```
import datetime
print("date and time", datetime.datetime.now())
```

19.What is the special file that each package in Python must contain?

Each package in Python must contain a special file called init .py. init .py can be an empty file but it is often used to perform setup needed for the package(import things, load things into path, etc).

Example :

```
package/
init .py file.py file2.py file3.py subpackage/
init .py submodule1.py submodule2.py
```

20.How do you use command line arguments to give input to the program? (or) What is command line argument? (May 2019) (Nov / Dec 2019)

Python sys module provides access to any command-line arguments via sys.argv. sys.argv is the list of command-line arguments. len(sys.argv) is the number of command-line arguments.

Example:

```
import sys
program_name = sys.argv[0]
arguments = sys.argv[1:]
count = len(arguments)
```

21.What are the different file operations?

In Python, a file operation takes place in the following order.

1. Open a file
2. Read or write (perform operation)
3. Close the file

22.What are the different file modes?

- 'r' - Open a file for reading. (default)
- 'w - Open a file for writing. Creates a new file if it does not exist or truncates the file if it exists.
- 'x' - Open a file for exclusive creation. If the file already exists, the operation fails.
- 'a' - Open for appending at the end of the file without truncating it. Creates a new file if it does not exist.
- 't' - Open in text mode. (default)
- 'b' - Open in binary mode.
- '+' - Open a file for updating (reading and writing)

23.How to view all the built-in exception in python.

The built-in exceptions using the local() built-in functions as follows.

Syntax: >>> locals()['__builtins__']

This will return us a dictionary of built-in exceptions, functions and attributes.

24.What do you mean IndexError?

IndexError is raised when index of a sequence is out of range.

Example:

```
>>> l=[1,2,3,4,5]
>>> print l[6]
Traceback (most recent call last):
File "<pyshell#16>", line 1, in <module> print l[6]
IndexError: list index out of range
```

25.Find the syntax error in the code given:

while True print('Hello World') **(Jan 2019)**

In the above given program, colon is missing after the condition. The right way to write the above program is,

```
while True:
    print('Hello World')
```

26.Categorize the different types errors arises during programming. Interpret the following python code

```
>>>import os
>>>cwd = os.getcwd()
>>>print cwd
```

(May 2019)

Basic types of errors:

1. Syntax Error: Raised by the parser when a syntax error is encountered. Semantic Error:

2. Semantic Error: Raised by the parser when there is logical error in the program.

Here in the above given program, Syntax error occurs in the third line (print cwd) SyntaxError: Missing parentheses in call to 'print'.

28.Write method to rename and delete files (Nov/Dec 2019)

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

Reg. No. : | | | | | | | | | | | | |

Question Paper Code : 21184

B.E./B.Tech. DEGREE EXAMINATIONS, NOVEMBER/DECEMBER 2023.

First Semester

Civil Engineering

GE 3151 – PROBLEM SOLVING AND PYTHON PROGRAMMING

(Common to All Branches)

(Also common to all branches for B.E (Part-Time) Regulations - 2023)

(Regulations 2021)

Time : Three hours Maximum : 100 marks

Answer ALL questions.

PART A — (10 × 2 = 20 marks)

1. What is Algorithm?

2. What are the simple strategies for developing algorithms?

3. List the various single valued data types in Python.

4. What is an indentation in python? Give an example.

5. Write a for loop that prints numbers from 0 to 57 using the range function in python.

6. What is the fruitful function in python?

7. What are List Comprehension? Mention its advantages.

8. What is a tuple in python? Give an example.

9. Write the significance of format operator.

10. How exceptions are handled in python?

PART B — (5 × 16 = 80 marks)

11. (a) (i) What are the building blocks of algorithm, explain in detail. (6)

(ii) Discuss the various process to the smallest value in the list a = [18, 52, 23, 41, 32] and write a simple python program for the same. (10)

Or

(b) (i) Explain the logic of the Tower of Hanoi puzzle and write a simple python program for the same and mention the time complexity. (12)

(ii) Define the computational problem and how these problems are classified. (4)

12. (a) (i) Explain the python interpreter and interactive mode in detail. (12)

(ii) What operator has the highest precedence in Python? (4)

Or

(b) (i) Write a python program to calculate the distance between two points. (8)

(ii) Explain the different Boolean and bitwise operator types in Python? (8)

13. (a) (i) Illustrate the different types of control flow statements in Python with flowcharts. (12)

(ii) Explain any two string formats available in Python. (4)

Or

(b) (i) Discuss the binary search algorithm with time complexity and write a python to implement the same using the recursive method. (12)

(ii) Why are strings in Python immutable? (4)

14. (a) (i) Discuss the differences and applications of List, Tuple, and Dictionary in Python. (12)

(ii) Explain any cloning list technique in python. (4)

Or

(b) (i) Discuss Python dictionaries and list some of their methods. (8)

(ii) Write a simple sorting python program to sort different data types. (8)

15. (a) (i) Explain in detail python files, their types, functions, and operations that can be performed on files with examples. (12)

(ii) Differentiate between Python Modules and Packages. (4)

Or

(b) (i) Write a simple python program to count the number of words in the sentence using the split method and list other count methods. (12)

(ii) What are command line arguments in python? (4)

Reg. No. : | | | | | | | | | | | | |

Question Paper Code : 51224

B.E./B.Tech. DEGREE EXAMINATIONS, APRIL/MAY 2024.

First Semester

Civil Engineering

GE 3151 – PROBLEM SOLVING AND PYTHON PROGRAMMING

(Common to all Branches)

(Also Common to PTGE 3151 – Problem Solving and Python Programming for B.E. (Part-Time) First Semester – All Branches – Regulations – 2023)

(Regulations 2021)

Time : Three hours | Maximum : 100 marks

Answer ALL questions.

PART A — (10 × 2 = 20 marks)

1. Differentiate Algorithm and Pseudo code.
2. Write any two disadvantages of flowchart?
3. What is the difference between interactive mode and script mode?
4. Mention the features of lists in python.
5. What is 'len' function? Give example for how it is used on strings.
6. How to split strings and what function is used to perform that operation?
7. What is range( ) function and how it is used in lists?
8. What are the advantages of 'Tuple' over 'List'?
9. What is module and package in Python?
10. List few common exception types.

PART B — (5 × 16 = 80 marks)

11. (a) (i) Explain the steps involved in program development cycle. (8)

(ii) Write the algorithm, pseudocode and draw the flowchart for the following :

(1) Guess an integer number in a range. (4)

(2) Towers of Hanoi. (4)

Or

(b) (i) Explain the design structures in pseudo code. (8)

(ii) Write the algorithm, pseudocode and draw the flowchart for the following :

(1) To find the sum of square root of any three numbers. (4)

(2) To find the sum of first 100 integers. (4)

12. (a) Write the following python programs

(i) Test whether a given year is leap year or not (8)

(ii) To convert Celsius to Fahrenheit. (8)

Or

(b) Write the following python programs

(i) To find whether a given number is Armstrong number or not (8)

(ii) To print Fibonacci series. (8)

13. (a) (i) Explain call by value and call by reference in python. (8)

(ii) How to perform a user input in Python? Explain with example. (8)

Or

(b) (i) Briefly explain about function prototypes. (8)

(ii) Write a program to check whether entered string is palindrome or not. (8)

14. (a) What are the basic list operations that can be performed in Python? Explain each operation with its syntax and example. (16)

Or

(b) What is Dictionary? Explain Python dictionaries in detail discussing its operations and methods. (16)

15. (a) (i) Write a program to enter a number in Python and print its octal and hexadecimal equivalent. (8)

(ii) Explain in detail about namespaces and scoping. (8)

Or

(b) Explain in detail about Python Files, its types, functions and operations that can be performed on files with examples. (16)

———————

Reg. No. : | | | | | | | | | | | | |

Question Paper Code : 30198

B.E./B.Tech. DEGREE EXAMINATIONS, APRIL/MAY 2023.

First Semester

GE 3151 – PROBLEM SOLVING AND PYTHON PROGRAMMING

(Common to ALL Branches)

(Regulations 2021)

Time : Three hours Maximum : 100 marks

Answer ALL questions.

PART A — (10 × 2 = 20 marks)

1. State the qualities of a good algorithm.
2. Enumerate the strategies involved in developing algorithm.
3. What are the rules for writing an identifier?
4. Mention the types of arguments in python.
5. Compare return value and composition.
6. Define string immutability.
7. Define cloning in list.
8. What is the benefit of using tuple assignment in python?
9. How do you delete a file in Python?
10. How do you use command line arguments to give input to the program?

PART B — (5 × 16 = 80 marks)

11. (a) (i) Briefly describe iteration and recursion. Illustrate with an example. (8)

(ii) Write an algorithm and give the flowchart to find the net salary of an employee. (8)

Or

(b) (i) Compare machine language, assembly language and high-level language. (8)

(ii) Write the pseudocode for the following : (8)

(1) Find maximum in a list

(2) Towers of Hanoi.

12. (a) (i) List down the rules for naming the variable with examples. (8)

(ii) Write a Python program to swap two variables. (8)

Or

(b) (i) What is precedence? List out the order of precedence and demonstrate in detail with example. (8)

(ii) Write a Python program to check whether a given year is a leap year or not. (8)

13. (a) (i) Explain string module. (8)

(ii) Write a program to find the sum and average of array of numbers. (8)

Or

(b) Write a program to perform search operation which sequentially checks each element of the list until a match is found or the whole list has been searched? (16)

14. (a) Explain in detail about lists, list operations and list slices. (16)

Or

(b) Write a program to generate electricity bill based upon the no. of units consumed. Refer the table below for unit and price details. (16)

No. of units consumed	Price (Domestic)	Price (Commercial)
1. Upto 200 units	200 (Base price)	500
2. Units >200 and <500	1100	1700
3. Units >500 <1000	4500	7000

Note : Program should read information from the user. Whether it is for domestic or commercial purpose.

15. (a) (i) Write a function that copies a file reading and writing upto 50 characters at a time. (8)

(ii) Write a python program to count number of lines, words and characters in a text file. (8)

Or

(b) Write a Python program to implement stack operations using modules. (16)

Reg. No. : | | | | | | | | | | | | |

Question Paper Code : 30509

B.E./B.Tech. DEGREE EXAMINATIONS, NOVEMBER/DECEMBER 2022.

First Semester

GE 3151 – PROBLEM SOLVING AND PYTHON PROGRAMMING

(Common to: All Branches)

(Regulations 2021)

Time : Three hours Maximum : 100 marks

Answer ALL questions.

PART A — (10 × 2 = 20 marks)

1. Write an algorithm to find the sum of first 'N' natural numbers.

2. Distinguish between top down and bottom up approaches to design an algorithm.

3. What is slicing operator in python? Give an example.

4. Python variables do not have specific types. Justify this statement using an example.

5. Write a python program to add two matrices.

6. Write a simple function to multiply two numbers in python.

7. Write a program to create a clone of the list: list 1 = [1,2,3,4,5,6].

8. How are the values of tuples accessed? Illustrate with an example.

9. Define relative path and absolute path with respect to files.

10. What are exceptions?

PART B — (5 × 16 = 80 marks)

11. (a) (i) Explain the different building blocks of algorithms with their notations. (10)

(ii) Write an algorithm to find an element in the given set of numbers. (6)

Or

(b) (i) What is meant by recursion? Write a recursive algorithm to solve Towers of Hanoi problem. (8)

(ii) Draw a flowchart to check if the given word is palindrome. (8)

12. (a) (i) Write a program to find the roots of a quadratic equation given the coefficients a, b, c. (8)

(ii) Describe the shift and logical operators used in python with examples. (8)

Or

(b) (i) Elaborate on membership, identity and bitwise operators of python with suitable examples. (12)

(ii) Write a program to print the digit at one's place of a number. (4)

13. (a) (i) Describe the conditional branching statements of python with examples. (8)

(ii) Write the syntax of while loop and use the same to classify if a given number is prime or not. (8)

Or

(b) (i) Describe parameter passing in functions using examples. (8)

(ii) Discuss about the scope and lifetime of variables considering functions. (8)

14. (a) (i) Describe the addition and deletion operation in a list data structure with examples. (8)

(ii) Write a program that has a nested list to store topper details and display the details. (8)

Or

(b) (i) Discuss the basic tuple operations with examples. (8)

(ii) Write a program to swap two values using tuple assignment. (8)

15. (a) (i) Explain opening and closing of files in python using examples. (8)

(ii) Write a program to display the contents of a file by performing split operation whenever a comma is encountered in a file. (8)

Or

(b) (i) Explain the use of packages and modules in python with examples. (8)

(ii) Write a program to handle the division by zero exception. (8)
