KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed to be University)

(Established Under Section 3 of UGC Act 1956)

Eachanari (po), Coimbatore-21

Semester – III

18ITU304B PROGRAMMING IN PHYTHON 3H – 3C

Instruction Hours / week: L: 3 T: 0 P: 0 **Marks:** Internal : **40** External : **60** Total: **100**

End Semester Exam : 3 Hours

Course Objectives

- To Learn Syntax and Semantics and create Functions in Python.
- To Handle Strings and Files in Python.
- To Understand Lists, Dictionaries in Python.
- To Implement Object Oriented Programming concepts in Python
- To Build GUI applications

Course Outcomes (COs)

Upon completion of the course, students will be able to

1. Develop algorithmic solutions to simple computational problems
2. Read, write, execute by hand simple Python programs.
3. Structure simple Python programs for solving problems.
4. Decompose a Python program into functions.
5. Represent compound data using Python lists, tuples, dictionaries.
6. Read and write data from/to files in Python Programs.

Unit I

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

Unit II

Data, Expressions, Statements : Python interpreter and interactive mode; values and types: int, float, boolean, string, and list; variables, expressions, statements, tuple assignment, precedence of operators, comments; modules and functions, function definition and use, flow of execution, parameters and arguments; Illustrative programs: exchange the values of two variables, circulate the values of n variables, distance between two points.

Unit III

Control Flow, Functions: 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.

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: selection sort, insertion sort, merge sort, histogram.

Unit V

Files, Modules, Packages: Files and exception: text files, reading and writing files, format operator; command line arguments, errors and exceptions, handling exceptions, modules, packages; Illustrative programs: word count, copy file.

Suggested Readings

1. Allen B. Downey, ``Think Python: How to Think Like a Computer Scientist‘‘, 2nd edition, Updated for Python 3, Shroff/O‘Reilly Publishers, 2016 (http://greenteapress.com/wp/thinkpython/)
2. Guido van Rossum and Fred L. Drake Jr, ―An Introduction to Python – Revised and updated for Python 3.2, Network Theory Ltd., 2011.
3. John V Guttag, ―Introduction to Computation and Programming Using Python‘‘, Revised and expanded Edition, MIT Press , 2013
4. Robert Sedgewick, Kevin Wayne, Robert Dondero, ―Introduction to Programming in Python: An Inter-disciplinary Approach, Pearson India Education Services Pvt. Ltd., 2016.
5. Timothy A. Budd, ―Exploring Python‖, Mc-Graw Hill Education (India) Private Ltd.,, 2015.
6. Kenneth A. Lambert, ―Fundamentals of Python: First Programs‖, CENGAGE Learning, 2012.
7. Charles Dierbach, ―Introduction to Computer Science using Python: A Computational ProblemSolving Focus, Wiley India Edition, 2013.
8. Paul Gries, Jennifer Campbell and Jason Montojo, ―Practical Programming: An Introduction to Computer Science using Python 3‖, Second edition, Pragmatic Programmers, LLC, 2013.

ESE Pattern	
Part – A (Online)	20 * 1 = 20
Part – B	5 * 2 = 10
Part – C (Either or)	5 * 6 = 30
Total	60 marks

Faculty **HOD**

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed to be University)
(Established Under Section 3 of UGC Act 1956)
Eachanari (po), Coimbatore-21

DEPARTMENT OF CS, CA & IT

LECTURE PLAN

SUBJECT NAME: PYTHON PROGRAMMING

SUBJECT CODE: 18ITU303 **SEMESTER: III**

BATCH: 2018-2021 **CLASS: II B.Sc.IT**

STAFF: Dr.D.SHANMUGA PRIYAA

S.No	Lecture Duration (Hr)	Topics	Support Materials
		UNIT -I	
1.	1	Algorithms, building blocks of algorithms	S1: 65
2.	1	Notation (Pseudo code), Notation (flow chart, programming language)	S3: 66-70
3.	1	Algorithmic problem solving	S3: 71-72
4.	1	Simple strategies for developing algorithms (iteration, recursion)	S3: 44
5.	1	Illustrative programs: find minimum in a list, insert a card in a list of sorted cards	W1
6.	1	Guess an integer number in a range, Towers of Hanoi	W2
7.	1	Recapitulation and Discussion of important questions	
	1	**Total No. of Periods allotted for Unit – I**	**7**
		UNIT-II	
1.	1	DATA, EXPRESSIONS, STATEMENTS Python interpreter and interactive mode, Values and types: int, float, boolean, string and list	S1: 13-14, 20-22
2.	1	Variables, expressions, statements, tuple assignment Precedence of operators, comments	S1: 8-9, 15-19
3.	1	Modules and functions	S1: 177-181
4.	1	Function definition and use, flow of execution	S1: 77-81
5.	1	Parameters and arguments	S1: 82
6.	1	Illustrative programs: exchange the values of two variables, circulate the values of n variables, distance	W3

		between two points.	
7.	1	Recapitulation and Discussion of important questions	
		Total No. of Hours allotted for Unit – II	**7**
		UNIT-III	
1.	1	Conditionals: Boolean values and operators Conditional (if), alternative (if-else), chained conditional (if-elif-else)	S1: 39-48
2.	1	Iteration: state, while, for, break, continue, pass	S1: 49-62
3.	1	Fruitful functions: return values	S1: 83, 90-91
4.	1	Parameters, local and global scope, Function composition, recursion	S1: 85,105, 95
5.	1	Strings: string slices, immutability	S1: 106, 109-115
6.	1	String functions and methods, String module; Lists as arrays	S1: 114, 120-122
7.	1	Recapitulation and Discussion of important questions	
		Total No. of Hours allotted for Unit – III	**7**
		UNIT-IV	
1.	1	Lists: list operations, list slices, List methods	S1: 133-136, 142-143
2.	1	List loop, list mutability	S1: 68, 137
3.	1	List aliasing, cloning lists, list parameters	S1 139-141
4.	1	Tuples: tuple assignment, tuple as return value	S1: 129-131
5.	1	Dictionaries: operations and methods, Advanced list processing - list comprehension	S1: 143,151-154
6.	1	Illustrative programs: Insertion sort, Mergesort, histogram	W4
7.	1	Recapitulation and Discussion of important questions	W4
		Total No. of Hours allotted for Unit – IV	**7**
		UNIT-V	
1.	1	Text files, reading and writing files, Format operator; command line arguments	S1: 167-170
2.	1	Errors and exceptions, Handling exceptions	S1: 255-258
3.	1	Modules, Packages	S1:171-18,183
4.	1	Illustrative programs: word count, copy file	W5
5.	1	Recapitulation and Discussion of important questions	
6.	1	Discussion of previous ESE Question papers	
7.	1	Discussion of previous ESE Question papers	
8.	1	Discussion of previous ESE Question papers	
		Total No. of Hours allotted for Unit – V	**8**

Total No. of Hours: 36

Suggested Readings

S1. Allen B. Downey, ``Think Python: How to Think Like a Computer Scientist‘‘, 2nd edition, Updated for Python 3, Shroff/O‘Reilly Publishers, 2016 (http://greenteapress.com/wp/think-python/)

S2. Guido van Rossum and Fred L. Drake Jr ,— An Introduction to Python - Revised and updated for Python 3.2, Network Theory Ltd., 2011.

S3. John V Guttag, “Introduction to Computation and Programming Using Python”, Revised and expanded Edition, MIT Press , 2013

S4. Robert Sedgewick, Kevin Wayne, Robert Dondero ―Introduction to Programming in Python: An Inter-disciplinary Approach, Pearson India Education Services Pvt. Ltd., 2016.

S5. Timothy A. Budd, ―Exploring Python, Mc-Graw Hill Education (India) Private Ltd., 2015.

S6. Kenneth A. Lambert, ―Fundamentals of Python: First Programs, CENGAGE Learning, 2012.

S7. Charles Dierbach, ―Introduction to Computer Science using Python: A Computational Problem- Solving Focus, Wiley India Edition, 2013.

S8. Paul Gries, Jennifer Campbell and Jason Montojo, ―Practical Programming: An Introduction to Computer Science using Python 3, Second edition, Pragmatic Programmers, LLC, 2013.

Websites

W1. https://www.w3schools.com/python/

W2. https://www.tutorialspoint.com/python/

W3. https://docs.python.org/3/tutorial/

W4. https://www.javatpoint.com/python-tutorial

W5. https://www.programiz.com/python-programming/tutorial

SYLLABUS

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

1. PROBLEM SOLVING

Problem solving is the systematic approach to define the problem and creating number of solutions.

The problem solving process starts with the problem specifications and ends with a Correct program.

1.1 PROBLEM SOLVING TECHNIQUES

Problem solving technique is a set of techniques that helps in providing logic for solving a problem.

Problem Solving Techniques:

Problem solving can be expressed in the form of

1. Algorithms.
2. Flowcharts.
3. Pseudo codes.
4. programs

1.2. ALGORITHM

It is defined as a sequence of instructions that describe a method for solving a problem. In other words it is a step by step procedure for solving a problem.

Properties of Algorithms

- Should be written in simple English
- Each and every instruction should be precise and unambiguous.
- Instructions in an algorithm should not be repeated infinitely.
- Algorithm should conclude after a finite number of steps.
- Should have an end point
- Derived results should be obtained only after the algorithm terminates.

Qualities of a good algorithm

The following are the primary factors that are often used to judge the quality of the algorithms.

Time – To execute a program, the computer system takes some amount of time. The lesser is the time required, the better is the algorithm.

Memory – To execute a program, computer system takes some amount of memory space. The lesser is the memory required, the better is the algorithm.

Accuracy – Multiple algorithms may provide suitable or correct solutions to a given problem, some of these may provide more accurate results than others, and such algorithms may be suitable.

Example:

Example

Write an algorithm to print „Good Morning"

Step 1: Start

Step 2: Print "Good Morning"

Step 3: Stop

2. BUILDING BLOCKS OF ALGORITHMS (statements, state, control flow, functions)

Algorithms can be constructed from basic building blocks namely, sequence, selection and iteration.

2.1. Statements:

Statement is a single action in a computer.

In a computer statements might include some of the following actions

- input data-information given to the program
- process data-perform operation on a given input
- output data-processed result

2.2. State:

Transition from one process to another process under specified condition with in a time is called state.

2.3. Control flow:

The process of executing the individual statements in a given order is called control flow.

The control can be executed in three ways

1. sequence
2. selection
3. iteration

Sequence:

All the instructions are executed one after another is called sequence execution.

Example:

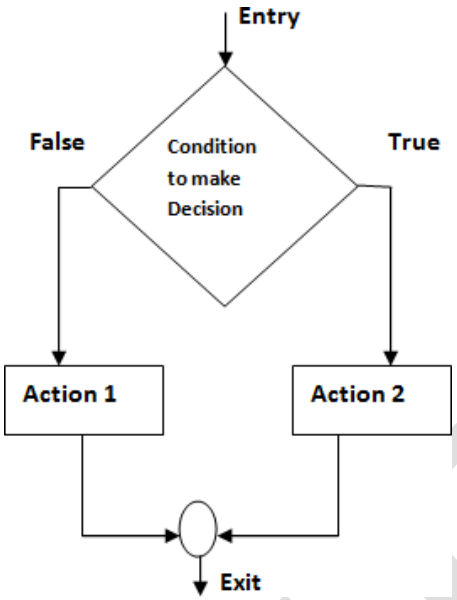
Add two numbers:

Step 1: Start
Step 2: get a,b
Step 3: calculate c=a+b
Step 4: Display c
Step 5: Stop

Selection:

A selection statement causes the program control to be transferred to a specific part of the program based upon the condition.

If the conditional test is true, one part of the program will be executed, otherwise it will execute the other part of the program.



Example

Write an algorithm to check whether he is eligible to vote?

Step 1: Start
Step 2: Get age
Step 3: if age >= 18 print "Eligible to vote"
Step 4: else print "Not eligible to vote"
Step 6: Stop

Iteration:

In some programs, certain set of statements are executed again and again based upon conditional test. i.e. executed more than one time. This type of execution is called looping or iteration.

Example

Write an algorithm to print all natural numbers up to n

Step 1: Start
Step 2: get n value.
Step 3: initialize i=1
Step 4: if (i<=n) go to step 5 else go to step 7
Step 5: Print i value and increment i value by 1
Step 6: go to step 4
Step 7: Stop

2.4. Functions:

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

Benefits of Using Functions

- Reduction in line of code
- code reuse
- Better readability
- Information hiding
- Easy to debug and test
- Improved maintainability

Example:

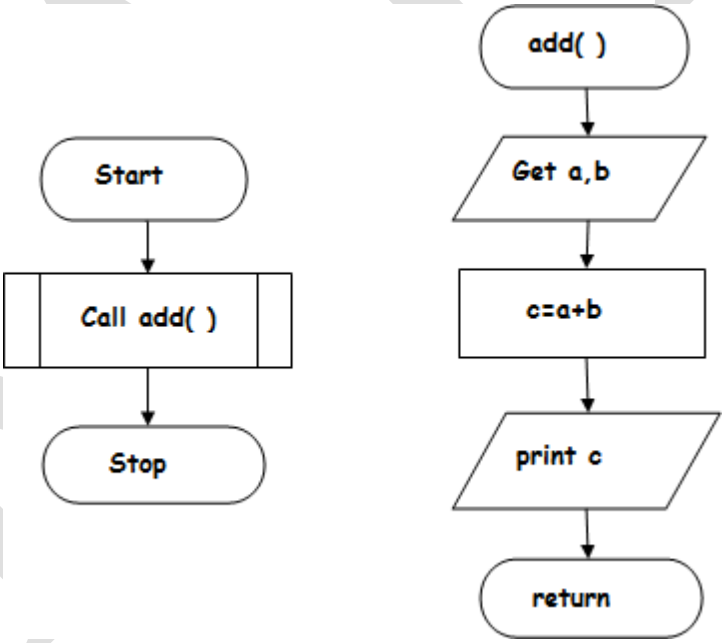
Algorithm for addition of two numbers using function

Main function()

Step 1: Start
Step 2: Call the function add()
Step 3: Stop

sub function add()

Step 1: Function start
Step 2: Get a, b Values
Step 3: add c=a+b
Step 4: Print c
Step 5: Return



3.NOTATIONS

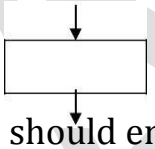
3.1.FLOW CHART

Flow chart is defined as graphical representation of the logic for problem solving. The purpose of flowchart is making the logic of the program clear in a visual representation.

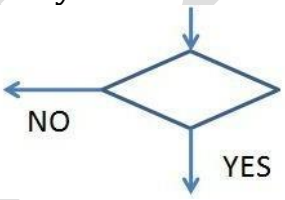
Symbol	Symbol Name	Description
	Flow Lines	Used to connect symbols
	Terminal	Used to start, pause or halt in the program logic
	Input/output	Represents the information entering or leaving the system
	Processing	Represents arithmetic and logical instructions
	Decision	Represents a decision to be made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

Rules for drawing a flowchart

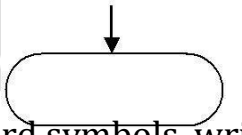
1. The flowchart should be clear, neat and easy to follow.
2. The flowchart must have a logical start and finish.
3. Only one flow line should come out from a process symbol.



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



5. Only one flow line is used with a terminal symbol.



6. Within standard symbols, write briefly and precisely.
7. Intersection of flow lines should be avoided.

Advantages of flowchart:

1. **Communication: -** Flowcharts are better way of communicating the logic of a system to all concerned.
2. **Effective analysis: -** With the help of flowchart, problem can be analyzed in more effective way.

3. **Proper documentation: -** Program flowcharts serve as a good program documentation, which is needed for various purposes.
4. **Efficient Coding: -** The flowcharts act as a guide or blueprint during the systems analysis and program development phase.
5. **Proper Debugging: -** The flowchart helps in debugging process.
6. **Efficient Program Maintenance: -** The maintenance of operating program becomes easy with the help of flowchart. It helps the programmer to put efforts more efficiently on that part.

Disadvantages of flow chart:

1. **Complex logic: -** Sometimes, the program logic is quite complicated. In that case, flowchart becomes complex and clumsy.
2. **Alterations and Modifications: -** If alterations are required the flowchart may require re-drawing completely.
3. **Reproduction: -** As the flowchart symbols cannot be typed, reproduction of flowchart becomes a problem.
4. **Cost: F**or large application the time and cost of flowchart drawing becomes costly.

3.2.PSEUDO CODE:

- Pseudo code consists of short, readable and formally styled English languages used for explain an algorithm.
- It does not include details like variable declaration, subroutines.
- It is easier to understand for the programmer or non programmer to understand the general working of the program, because it is not based on any programming language.
- It gives us the sketch of the program before actual coding.
- It is not a machine readable
- Pseudo code can't be compiled and executed.
- There is no standard syntax for pseudo code.

Guidelines for writing pseudo code:

- Write one statement per line
- Capitalize initial keyword
- Indent to hierarchy
- End multiline structure
- Keep statements language independent

Common keywords used in pseudocode

The following gives common keywords used in pseudocodes.

1. **//:** This keyword used to represent a comment.
2. **BEGIN,END:** Begin is the first statement and end is the last statement.
3. **INPUT, GET, READ**: The keyword is used to inputting data.
4. **COMPUTE, CALCULATE**: used for calculation of the result of the given expression.
5. **ADD, SUBTRACT, INITIALIZE** used for addition, subtraction and initialization**.**
6. **OUTPUT, PRINT, DISPLAY**: It is used to display the output of the program.
7. **IF, ELSE, ENDIF**: used to make decision.
8. **WHILE, ENDWHILE**: used for iterative statements.
9. **FOR, ENDFOR**: Another iterative incremented/decremented tested automatically.

Syntax for if else:	**Example: Greates of two numbers**

IF (condition)THEN
statement
...
ELSE
statement
...
ENDIF
	BEGIN
	READ a,b
	IF (a>b) THEN
	DISPLAY a is greater
	ELSE
	DISPLAY b is greater
	END IF
	END

Syntax for For:	**Example: Print n natural numbers**

FOR(*start-value* to *end-value)* DO
statement
...
ENDFOR
	BEGIN
	GET n
	INITIALIZE i=1
	FOR (i<=n) DO
	PRINT i
	i=i+1
	ENDFOR
	END

Syntax for While:	**Example: Print n natural numbers**

WHILE (condition) DO
statement
...
ENDWHILE
	BEGIN
	GET n
	INITIALIZE i=1
	WHILE(i<=n) DO
	PRINT i
	i=i+1
	ENDWHILE
	END

Advantages:

- Pseudo is independent of any language; it can be used by most programmers.
- It is easy to translate pseudo code into a programming language.
- It can be easily modified as compared to flowchart.
- Converting a pseudo code to programming language is very easy as compared with converting a flowchart to programming language.

Disadvantages:

- 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.
- For a beginner, It is more difficult to follow the logic or write pseudo code as compared to flowchart.

Example:

Addition of two numbers:

BEGIN
GET a,b
ADD c=a+b
PRINT c
END

Algorithm	Flowchart	Pseudo code
An algorithm is a sequence of instructions used to solve a problem	It is a graphical representation of algorithm	It is a language representation of algorithm.
User needs knowledge to write algorithm.	not need knowledge of program to draw or understand flowchart	Not need knowledge of program language to understand or write a pseudo code.

3.3.PROGRAMMING LANGUAGE

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

Types of programming language

1. Machine language
2. Assembly language
3. High level language

Machine language:

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

Advantages:

Translation free:

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

High speed

The machine language program is translation free. Since the conversion time is saved, the execution of machine language program is extremely fast.

Disadvantage:

- It is hard to find errors in a program written in the machine language.
- Writhing program in machine language is a time consuming process.

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

Assembly language:

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

- Assembly language is symbolic representation of machine language. Assembly languages are symbolic programming language that uses symbolic notation to represent machine language instructions. They are called low level language because they are so closely related to the machines.
  Ex: ADD a, b

Assembler:

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

Advantage:

- Easy to understand and use.
- It is easy to locate and correct errors.

Disadvantage

Machine dependent

The assembly language program which can be executed on the machine depends on the architecture of that computer.

Hard to learn

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

Less efficient

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

High level language

High level language contains English words and symbols. The specified rules are to be followed while writing program in high level language. The interpreter or compilers are used for converting these programs in to machine readable form.

Translating high level language to machine language

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

Compiler:

A compiler is a program which translates the source code written in 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.

Advantages

Readability

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

Machine independent

High level language program have the advantage of being portable between machines.

Easy debugging

Easy to find and correct error in high level language

Disadvantages

Less efficient

The translation process increases the execution time of the program. Programs in high level language require more memory and take more execution time to execute.

They are divided into following categories:

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

Interpreted programming languages:

An interpreted language is a programming language for which most of its implementation executes instructions directly, without previously compiling a program into machine language instructions. The interpreter executes the program directly translating each statement into a sequence of one or more subroutines already compiled into machine code.

Examples:

Pascal
Python

Functional programming language:

Functional programming language defines every computation as a mathematical evaluation. They focus on the programming languages are bound to mathematical calculations

Examples:

Clean
Haskell

Compiled Programming language:

A compiled programming is a programming language whose implementation are typically compilers and not interpreters.

It will produce a machine code from source code.

Examples:

C
C++
C#
JAVA

Procedural programming language:

Procedural (imperative) programming implies specifying the steps that the programs should take to reach to an intended state.

A procedure is a group of statements that can be referred through a procedure call. Procedures help in the reuse of code. Procedural programming makes the programs structured and easily traceable for program flow.

Examples:

Hyper talk
MATLAB

Scripting language:

Scripting language are programming languages that control an application. Scripts can execute independent of any other application. They are mostly embedded in the application that they control and are used to automate frequently executed tasks like communicating with external program.

Examples:

Apple script
VB script

Markup languages:

A markup language is an artificial language that uses annotations to text that define hoe the text is to be displayed.

Examples:

HTML
XML

Concurrent programming language:

Concurrent programming is a computer programming technique that provides for the execution of operation concurrently, either with in a single computer or across a number of systems.

Examples:

Joule
Limbo

Object oriented programming language:

Object oriented programming is a programming paradigm based on the concept of objects which may contain data in the form of procedures often known as methods.

Examples:
Lava
Moto

4. ALGORITHMIC PROBLEM SOLVING:

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

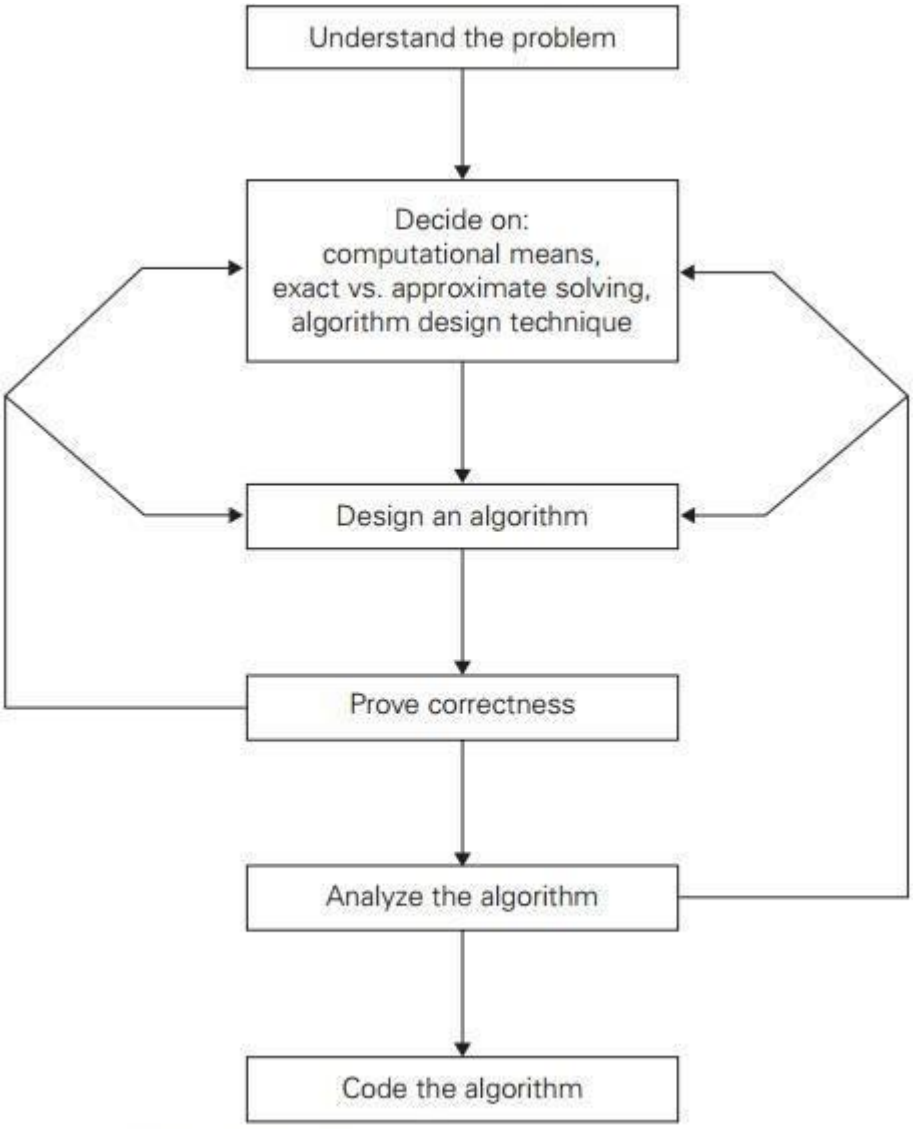


FIGURE 1.2 Algorithm design and analysis process.

Understanding the Problem

- It is the process of finding the input of the problem that the algorithm solves.
- It is very important to specify exactly the set of inputs the algorithm needs to handle.
- A correct algorithm is not one that works most of the time, but one that works correctly for *all* legitimate inputs.

Ascertaining the Capabilities of the Computational Device

- If the instructions are executed one after another, it is called sequential algorithm.

- If the instructions are executed concurrently, it is called parallel algorithm.

Choosing between Exact and Approximate Problem Solving

- The next principal decision is to choose between solving the problem exactly or solving it approximately.
- Based on this, the algorithms are classified as exact *algorithm* and *approximation algorithm.*

Deciding a data structure:

- Data structure plays a vital role in designing and analysis the algorithms.
- Some of the algorithm design techniques also depend on the structuring data specifying a problem's instance
- Algorithm+ Data structure=programs.

Algorithm Design Techniques

- An ***algorithm design technique*** (or "strategy" or "paradigm") is a general approach to solving problems algorithmically that is applicable to a variety of problems from different areas of computing.
- Learning these techniques is of utmost importance for the following reasons.
- First, they provide guidance for designing algorithms for new problems,
- Second, algorithms are the cornerstone of computer science

Methods of Specifying an Algorithm

- ***Pseudocode*** is a mixture of a natural language and programming language-like constructs. Pseudocode is usually more precise than natural language, and its usage often yields more succinct algorithm descriptions.

- In the earlier days of computing, the dominant vehicle for specifying algorithms was a ***flowchart***, a method of expressing an algorithm by a collection of connected geometric shapes containing descriptions of the algorithm's steps.

- **Programming language** can be fed into an electronic computer directly. Instead, it needs to be converted into a computer program written in a particular computer language. We can look at such a program as yet another way of specifying the algorithm, although it is preferable to consider it as the algorithm's implementation.

Proving an Algorithm's Correctness

- Once an algorithm has been specified, you have to prove its ***correctness***. That is, you have to prove that the algorithm yields a required result for every legitimate input in a finite amount of time.
- A common technique for proving correctness is to use mathematical induction because an algorithm's iterations provide a natural sequence of steps needed for such proofs.
- It might be worth mentioning that although tracing the algorithm's performance for a few specific inputs can be a very worthwhile activity, it cannot prove the algorithm's correctness conclusively. But in order to show that an algorithm is incorrect, you need just one instance of its input for which the algorithm fails.

Analysing an Algorithm

1. ***Efficiency***.

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

2. ***simplicity***.

- An algorithm should be precisely defined and investigated with mathematical expressions.
- Simpler algorithms are easier to understand and easier to program.
- Simple algorithms usually contain fewer bugs.

Coding an Algorithm

- Most algorithms are destined to be ultimately implemented as computer programs. Programming an algorithm presents both a peril and an opportunity.
- A working program provides an additional opportunity in allowing an empirical analysis of the underlying algorithm. Such an analysis is based on timing the program on several inputs and then analysing the results obtained.

5. SIMPLE STRATEGIES FOR DEVELOPING ALGORITHMS:

1. iterations
2. Recursions

5.1. Iterations:

A sequence of statements is executed until a specified condition is true is called iterations.

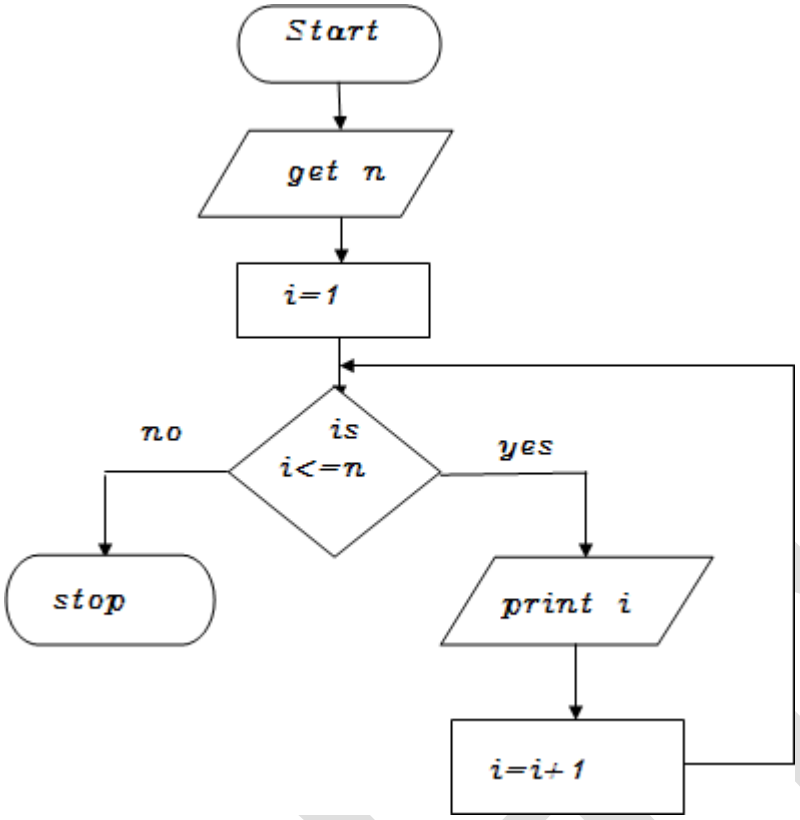
1. for loop
2. While loop

Syntax for For:	**Example: Print n natural numbers**

FOR(*start-value* to *end-value)* DO
  statement
...
ENDFOR
	BEGIN
	GET n
	INITIALIZE i=1
	FOR (i<=n) DO
	  PRINT i
	  i=i+1
	ENDFOR
	END

Syntax for While:	**Example: Print n natural numbers**

WHILE (condition) DO
  statement
  ...
ENDWHILE
	BEGIN
	GET n
	INITIALIZE i=1
	WHILE(i<=n) DO
	  PRINT i
	  i=i+1
	ENDWHILE
	END



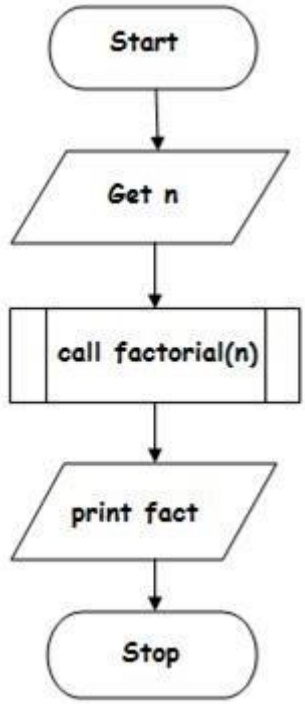
5.2. Recursions:

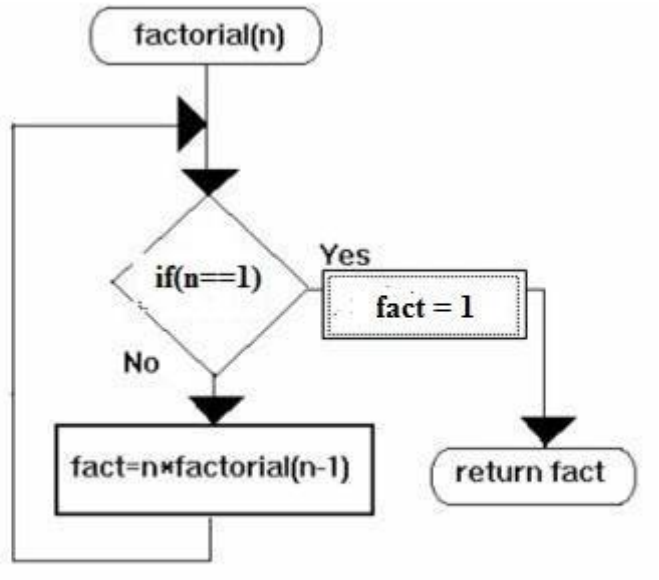
- A function that calls itself is known as recursion.
- Recursion is a process by which a function calls itself repeatedly until some specified condition has been satisfied.

Algorithm for factorial of n numbers using recursion:

Main function:
Step1: Start
Step2: Get n
Step3: call factorial(n)
Step4: print fact
Step5: Stop

Sub function factorial(n):
Step1: if(n==1) then fact=1 return fact
Step2: else fact=n*factorial(n-1) and return fact





Pseudo code for factorial using recursion:

Main function:

```
BEGIN
GET n
CALL factorial(n)
PRINT fact
BIN
```

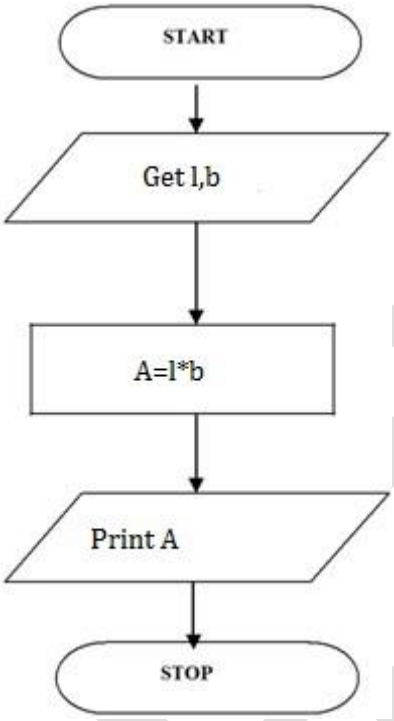
Sub function factorial(n):

```
IF(n==1) THEN
      fact=1
      RETURN fact
ELSE
     RETURN fact=n*factorial(n-1)
```

More examples:

Write an algorithm to find area of a rectangle

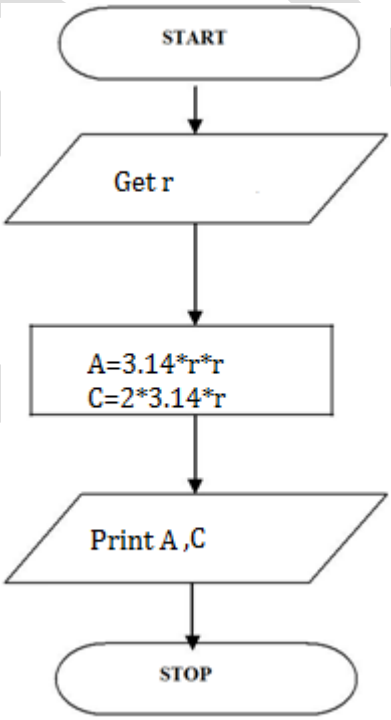
Step 1: Start
Step 2: get l,b values
Step 3: Calculate A=l*b
Step 4: Display A
Step 5: Stop



BEGIN
READ l,b
CALCULATE A=l*b
DISPLAY A
END

Write an algorithm for Calculating area and circumference of circle

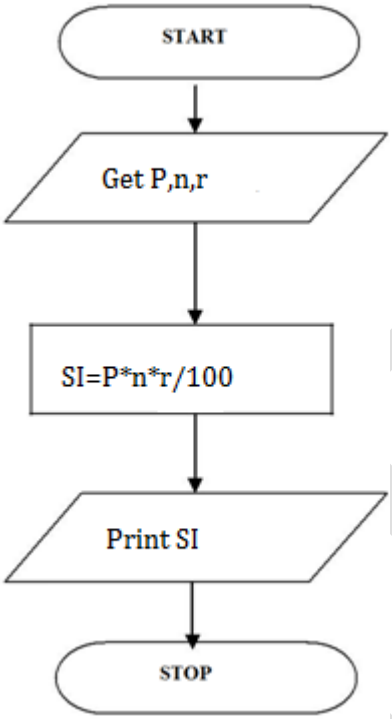
Step 1: Start
Step 2: get r value
Step 3: Calculate A=3.14*r*r
Step 4: Calculate C=2.3.14*r
Step 5: Display A,C
Step 6: Stop



BEGIN
READ r
CALCULATE A and C
A=3.14*r*r
C=2*3.14*r
DISPLAY A
END

Write an algorithm for Calculating simple interest

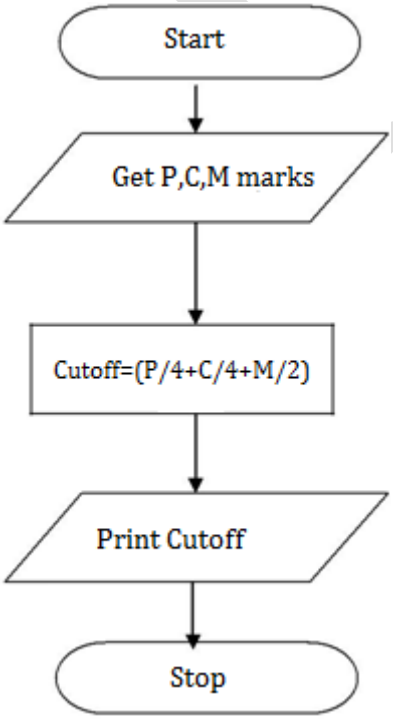
Step 1: Start
Step 2: get P, n, r value
Step3: Calculate
SI=(p*n*r)/100
Step 4: Display S
Step 5: Stop



BEGIN
READ P, n, r
CALCULATE S
SI=(p*n*r)/100
DISPLAY SI
END

Write an algorithm for Calculating engineering cutoff

Step 1: Start
Step2: get P,C,M value
Step3:calculate
Cutoff= (P/4+C/4+M/2)
Step 4: Display Cutoff
Step 5: Stop

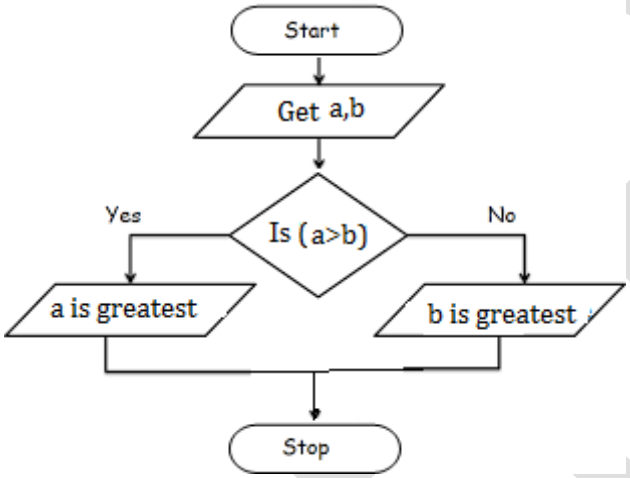


BEGIN
READ P,C,M
CALCULATE
Cutoff= (P/4+C/4+M/2)
DISPLAY Cutoff
END

To check greatest of two numbers

Step 1: Start
Step 2: get a,b value
Step 3: check if(a>b) print a is greater
Step 4: else b is greater
Step 5: Stop

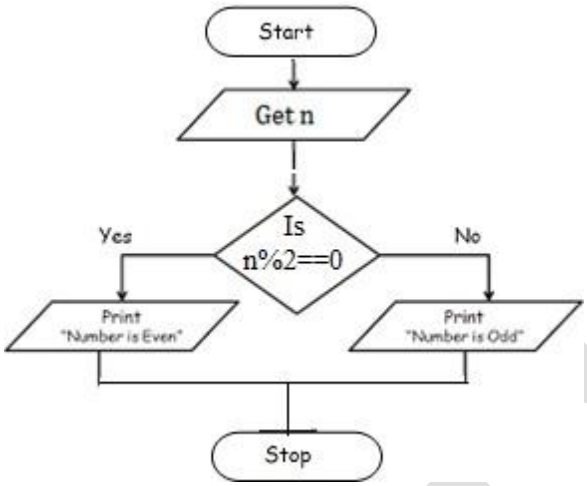
BEGIN
READ a,b
IF (a>b) THEN
DISPLAY a is greater
ELSE
DISPLAY b is greater
END IF
END



To check leap year or not

Step 1: Start
Step 2: get y
Step 3: if(y%4==0) print leap year
Step 4: else print not leap year
Step 5: Stop

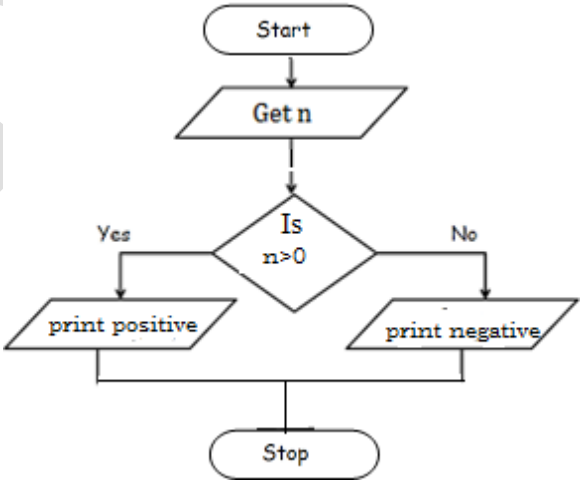
BEGIN
READ y
IF (y%4==0) THEN
DISPLAY leap year
ELSE
DISPLAY not leap year
END IF
END



To check positive or negative number

Step 1: Start
Step 2: get num
Step 3: check if(num>0) print a is positive
Step 4: else num is negative
Step 5: Stop

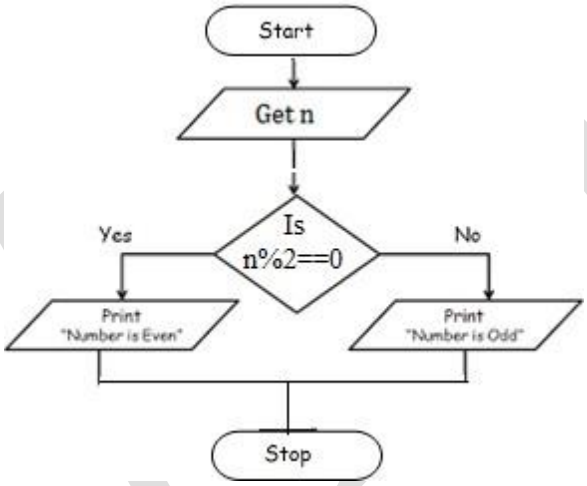
```
BEGIN
READ num
IF (num>0) THEN
DISPLAY num is positive
ELSE
DISPLAY num is negative
END IF
END
```



To check odd or even number

Step 1: Start
Step 2: get num
Step 3: check if(num%2==0) print num is even
Step 4: else num is odd
Step 5: Stop

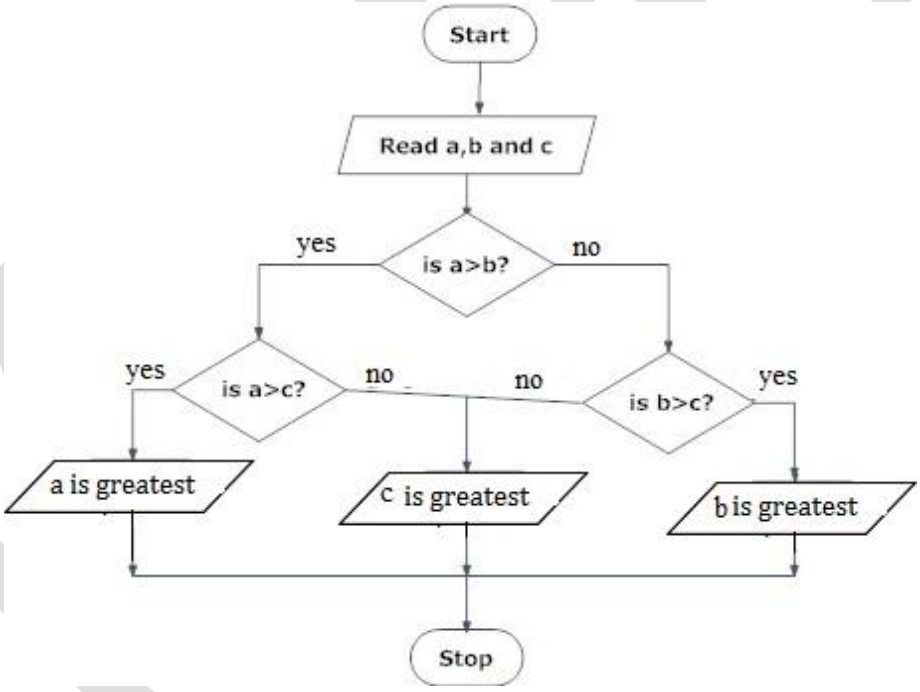
BEGIN
READ num
IF (num%2==0) THEN
DISPLAY num is even
ELSE
DISPLAY num is odd
END IF
END



To check greatest of three numbers

Step1: Start
Step2: Get A, B, C
Step3: if(A>B) goto Step4 else goto step5
Step4: If(A>C) print A else print C
Step5: If(B>C) print B else print C
Step6: Stop

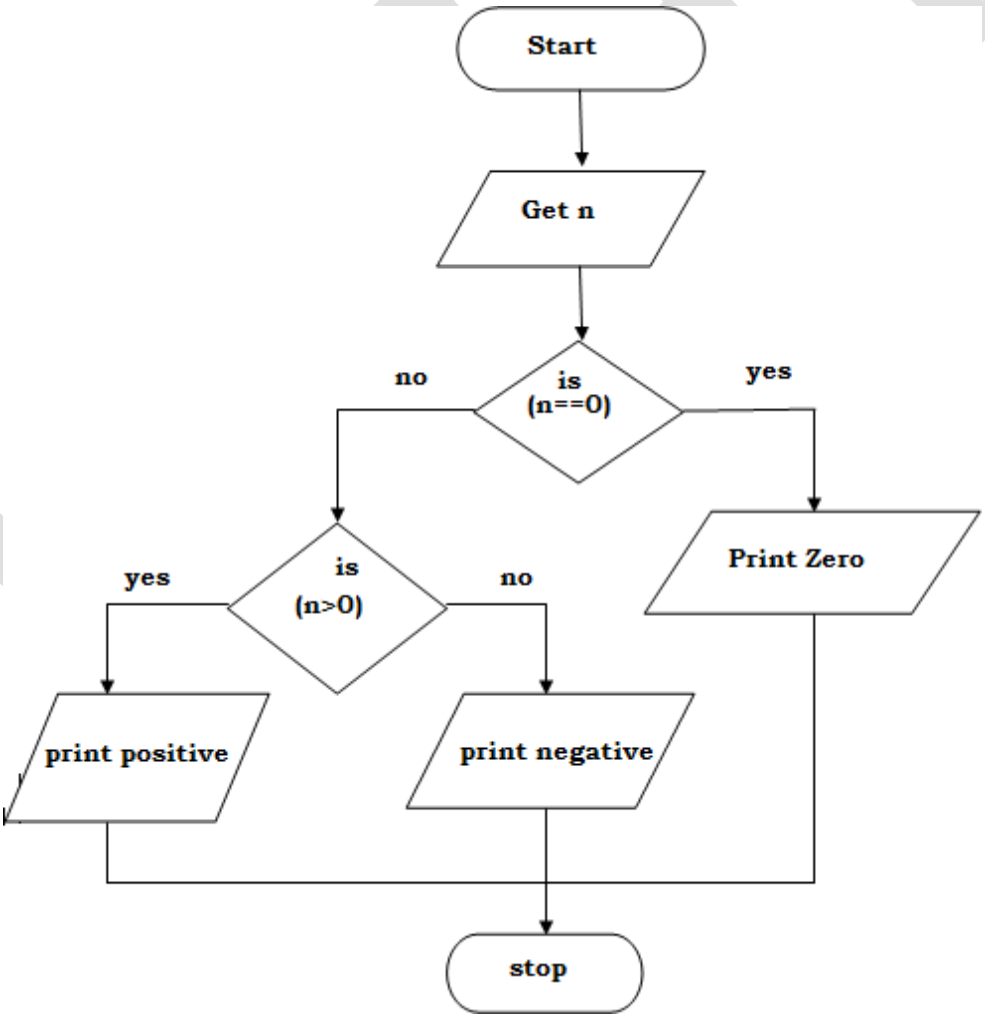
```
BEGIN
READ a, b, c
IF (a>b) THEN
    IF(a>c) THEN
       DISPLAY a is greater
     ELSE
       DISPLAY c is greater
     END IF
ELSE
    IF(b>c) THEN
       DISPLAY b is greater
    ELSE
       DISPLAY c is greater
    END IF
END IF
END
```



Write an algorithm to check whether given number is +ve, -ve or zero.

Step 1: Start
Step 2: Get n value.
Step 3: if (n ==0) print “Given number is Zero” Else goto step4
Step 4: if (n > 0) then Print “Given number is +ve”
Step 5: else Print “Given number is -ve”
Step 6: Stop

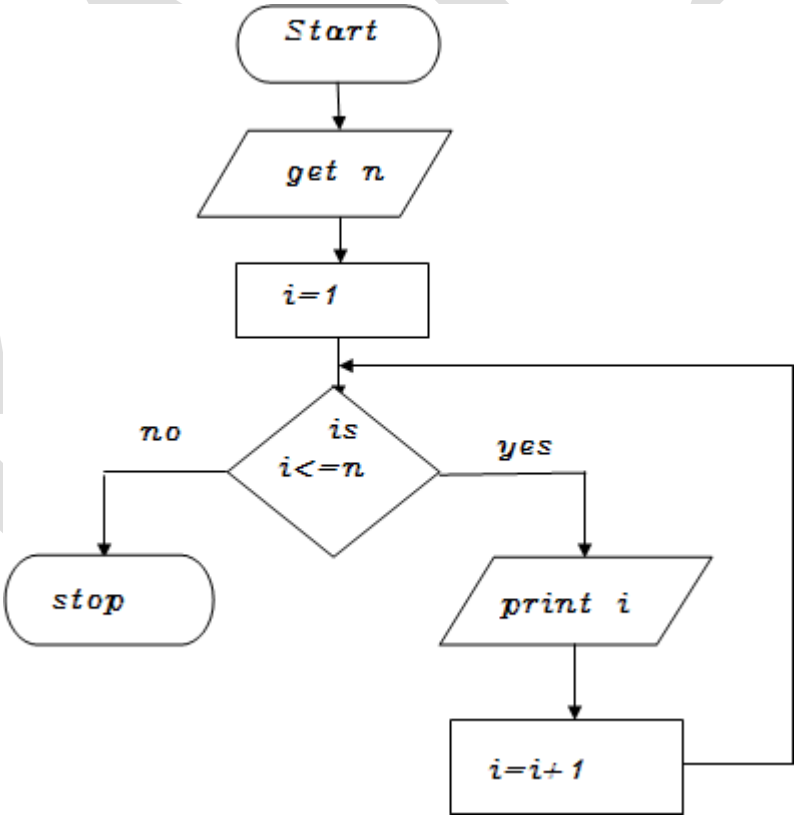
```
BEGIN
GET n
IF(n==0) THEN
   DISPLAY " n is zero"
ELSE
   IF(n>0) THEN
        DISPLAY "n is positive"
   ELSE
        DISPLAY "n is positive"
    END IF
END IF
END
```



Write an algorithm to print all natural numbers up to n

Step 1: Start
Step 2: get n value.
Step 3: initialize i=1
Step 4: if (i<=n) go to step 5 else go to step 8
Step 5: Print i value
step 6 : increment i value by 1
Step 7: go to step 4
Step 8: Stop

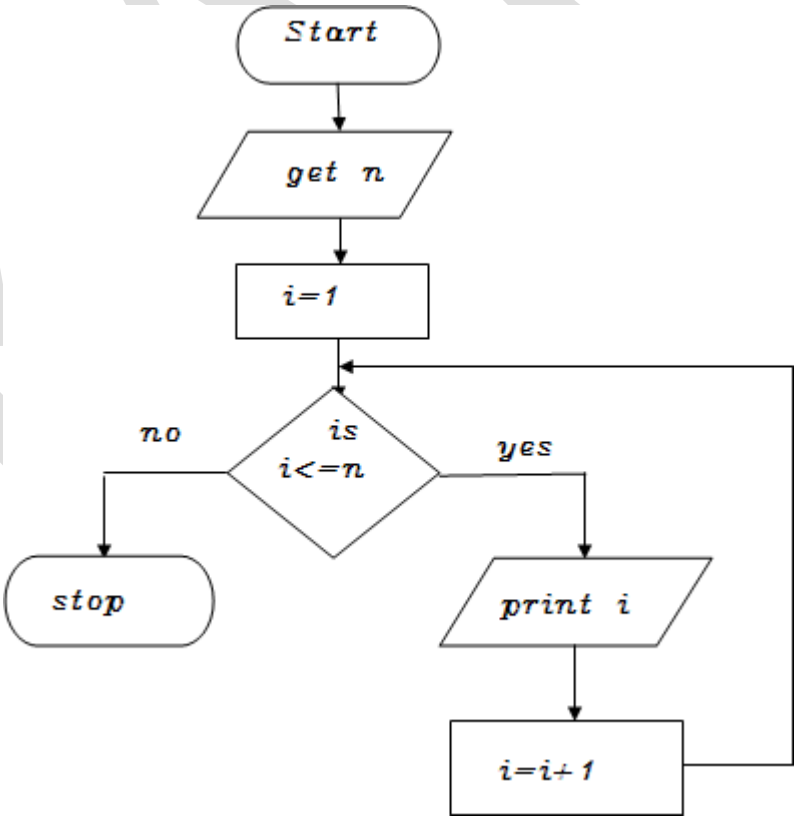
```
BEGIN
GET n
INITIALIZE i=1
WHILE(i<=n) DO
    PRINT i
    i=i+1
ENDWHILE
END
```



Write an algorithm to print n odd numbers

Step 1: start
step 2: get n value
step 3: set initial value i=1
step 4: check if(i<=n) goto step 5 else goto step 8
step 5: print i value
step 6: increment i value by 2
step 7: goto step 4
step 8: stop

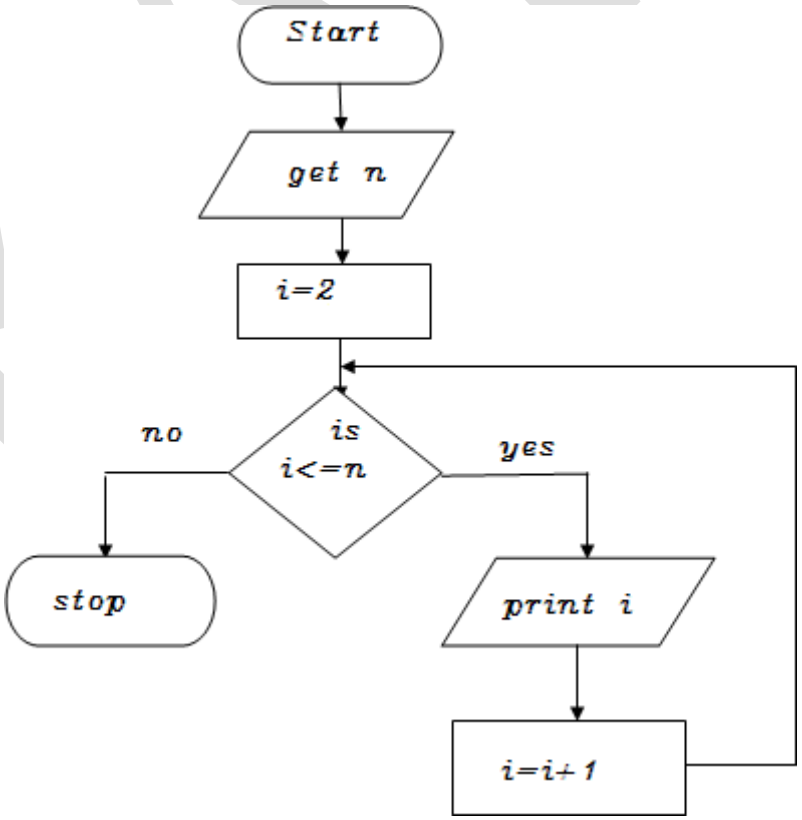
```
BEGIN
GET n
INITIALIZE i=1
WHILE(i<=n) DO
    PRINT i
    i=i+2
ENDWHILE
END
```



Write an algorithm to print n even numbers

Step 1: start
step 2: get n value
step 3: set initial value i=2
step 4: check if(i<=n) goto step 5 else goto step8
step 5: print i value
step 6: increment i value by 2
step 7: goto step 4
step 8: stop

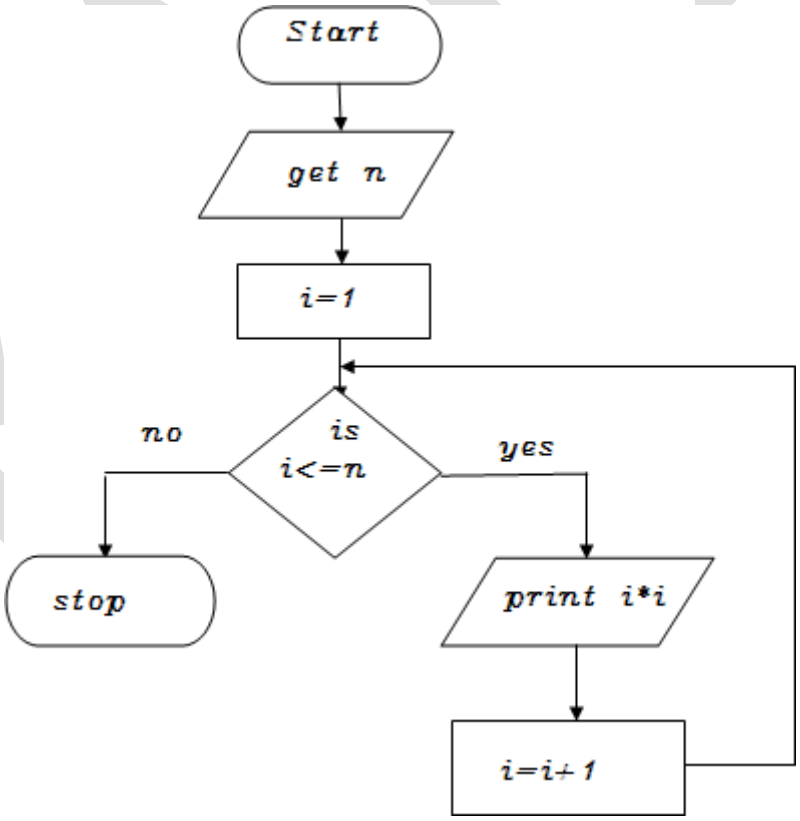
```
BEGIN
GET n
INITIALIZE i=2
WHILE(i<=n) DO
    PRINT i
    i=i+2
ENDWHILE
END
```



Write an algorithm to print squares of a number

Step 1: start
step 2: get n value
step 3: set initial value i=1
step 4: check i value if(i<=n) goto step 5 else goto step8
step 5: print i*i value
step 6: increment i value by 1
step 7: goto step 4
step 8: stop

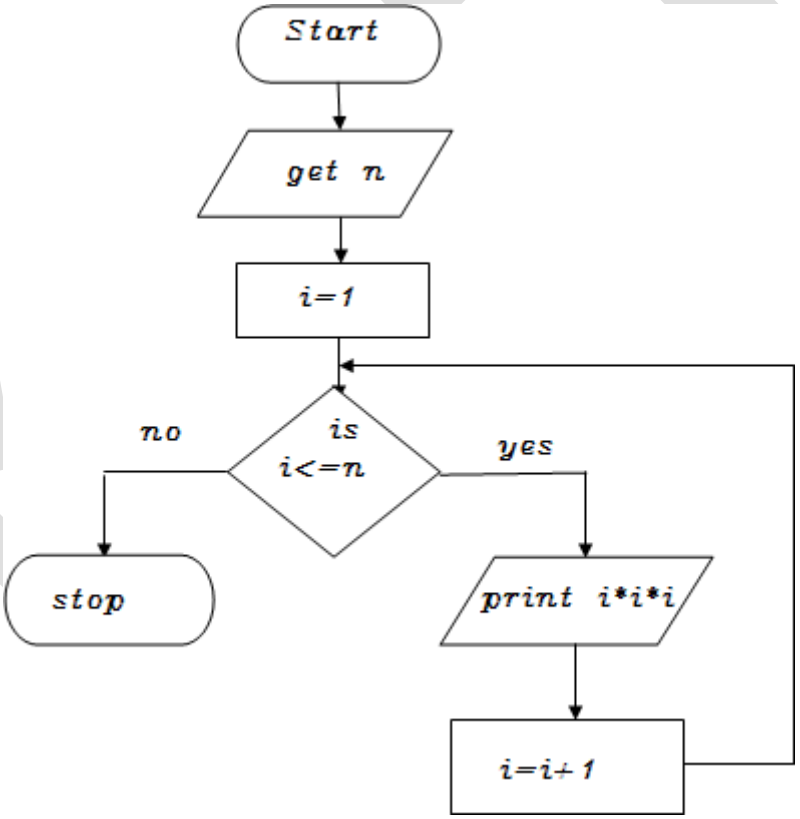
```
BEGIN
GET n
INITIALIZE i=1
WHILE(i<=n) DO
    PRINT i*i
    i=i+2
ENDWHILE
END
```



Write an algorithm to print to print cubes of a number

Step 1: start
step 2: get n value
step 3: set initial value i=1
step 4: check i value if(i<=n) goto step 5 else goto step8
step 5: print i*i *i value
step 6: increment i value by 1
step 7: goto step 4
step 8: stop

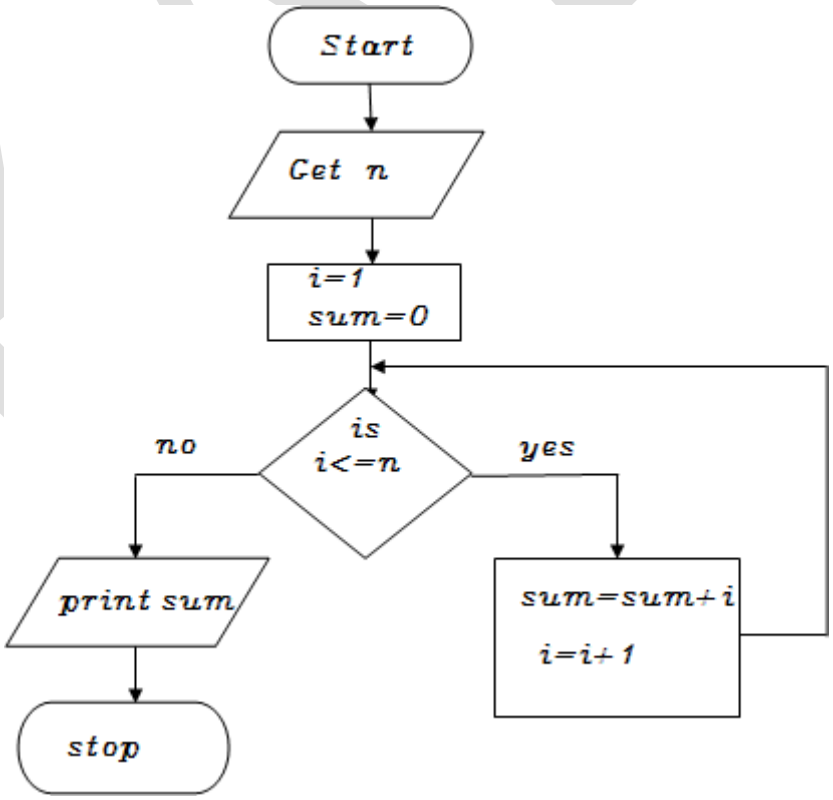
```
BEGIN
GET n
INITIALIZE i=1
WHILE(i<=n) DO
    PRINT i*i*i
    i=i+2
ENDWHILE
END
```



Write an algorithm to find sum of a given number

Step 1: start
step 2: get n value
step 3: set initial value i=1, sum=0
Step 4: check i value if(i<=n) goto step 5 else goto step8
step 5: calculate sum=sum+i
step 6: increment i value by 1
step 7: goto step 4
step 8: print sum value
step 9: stop

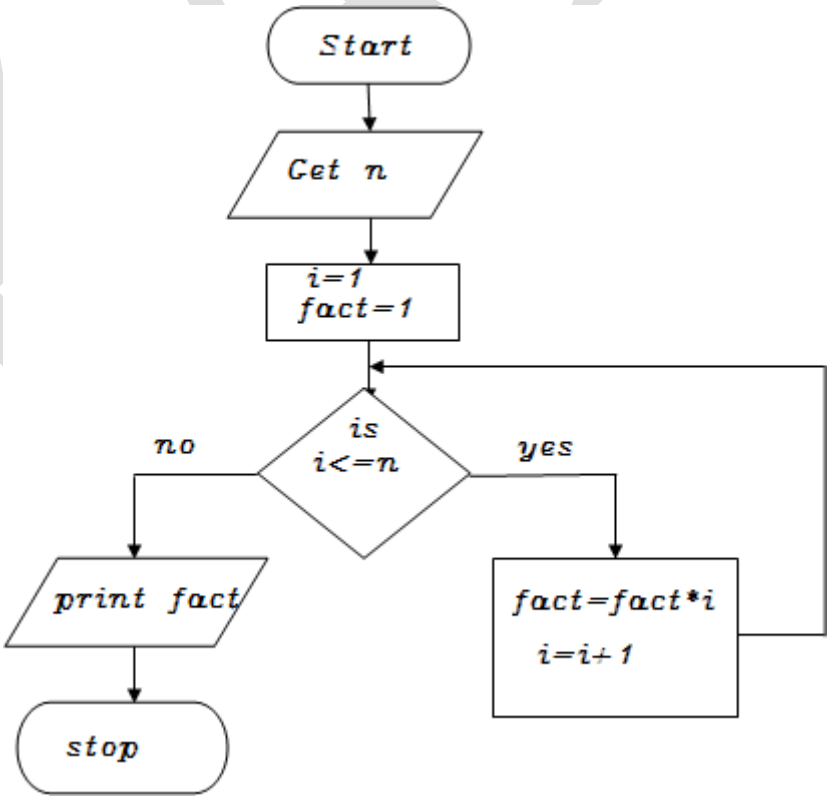
BEGIN
GET n
INITIALIZE i=1,sum=0
WHILE(i<=n) DO
    sum=sum+i
    i=i+1
ENDWHILE
PRINT sum
END



Write an algorithm to find factorial of a given number

Step 1: start
step 2: get n value
step 3: set initial value i=1, fact=1
Step 4: check i value if(i<=n) goto step 5 else goto step8
step 5: calculate fact=fact*i
step 6: increment i value by 1
step 7: goto step 4
step 8: print fact value
step 9: stop

```
BEGIN
GET n
INITIALIZE i=1,fact=1
WHILE(i<=n) DO
    fact=fact*i
    i=i+1
ENDWHILE
PRINT fact
END
```



Basic python programs:

Addition of two numbers	**Output**
a=eval(input("enter first no")) b=eval(input("enter second no")) c=a+b print("the sum is ",c)	enter first no 5 enter second no 6 the sum is 11
Area of rectangle	**Output**
l=eval(input("enter the length of rectangle")) b=eval(input("enter the breath of rectangle")) a=l*b print(a)	enter the length of rectangle 5 enter the breath of rectangle 6 30
Area & circumference of circle	**output**
r=eval(input("enter the radius of circle")) a=3.14*r*r c=2*3.14*r print("the area of circle",a) print("the circumference of circle",c)	enter the radius of circle4 the area of circle 50.24 the circumference of circle 25.12
Calculate simple interest	**Output**
p=eval(input("enter principle amount")) n=eval(input("enter no of years")) r=eval(input("enter rate of interest")) si=p*n*r/100 print("simple interest is",si)	enter principle amount 5000 enter no of years 4 enter rate of interest6 simple interest is 1200.0
Calculate engineering cutoff	**Output**
p=eval(input("enter physics marks")) c=eval(input("enter chemistry marks")) m=eval(input("enter maths marks")) cutoff=(p/4+c/4+m/2) print("cutoff =",cutoff)	enter physics marks 100 enter chemistry marks 99 enter maths marks 96 cutoff = 97.75
Check voting eligibility	**output**

age=eval(input("enter ur age"))
If(age>=18):
  print("eligible for voting")
else:
  print("not eligible for voting")
	Enter ur age
	19
	Eligible for voting

Find greatest of three numbers	output

a=eval(input("enter the value of a"))
b=eval(input("enter the value of b"))
c=eval(input("enter the value of c"))
if(a>b):
  if(a>c):
  print("the greatest no is",a)
  else:
  print("the greatest no is",c)
else:
  if(b>c):
  print("the greatest no is",b)
  else:
  print("the greatest no is",c)
	enter the value of a 9
	enter the value of a 1
	enter the value of a 8
	the greatest no is 9

Programs on for loop

Print n natural numbers	Output
for i in range(1,5,1): print(i)	1 2 3 4
Print n odd numbers	**Output**
for i in range(1,10,2): print(i)	1 3 5 7 9
Print n even numbers	**Output**
for i in range(2,10,2): print(i)	2 4 6 8
Print squares of numbers	**Output**
for i in range(1,5,1): print(i*i)	1 4 9 16
Print squares of numbers	**Output**
for i in range(1,5,1): print(i*i*i)	1 8 27 64

Programs on while loop

Print n natural numbers	**Output**

i=1
while(i<=5):
  print(i)
  i=i+1
	1
	2
	3
	4
	5

Print n odd numbers	**Output**

i=2
while(i<=10):
  print(i)
  i=i+2
	2
	4
	6
	8
	10

Print n even numbers	**Output**

i=1
while(i<=10):
  print(i)
  i=i+2
	1
	3
	5
	7
	9

Print n squares of numbers	**Output**

i=1
while(i<=5):
  print(i*i)
  i=i+1
	1
	4
	9
	16
	25

Print n cubes numbers	**Output**

i=1
while(i<=3):
  print(i*i*i)
  i=i+1
	1
	8
	27

find sum of n numbers	**Output**

i=1
sum=0
while(i<=10):
  sum=sum+i
  i=i+1
print(sum)
	55

factorial of n numbers/product of n numbers	**Output**
i=1 product=1 while(i<=10): product=product*i i=i+1 print(product)	3628800
sum of n numbers	**Output**
def add(): a=eval(input(“enter a value”)) b=eval(input(“enter b value”)) c=a+b print(“the sum is”,c) add()	enter a value 6 enter b value 4 the sum is 10
area of rectangle using function	**Output**
def area(): l=eval(input(“enter the length of rectangle”)) b=eval(input(“enter the breath of rectangle”)) a=l*b print(“the area of rectangle is”,a) area()	enter the length of rectangle 20 enter the breath of rectangle 5 the area of rectangle is 100
swap two values of variables	**Output**
def swap(): a=eval(input("enter a value")) b=eval(input("enter b value")) c=a a=b b=c print("a=",a,"b=",b) swap()	enter a value3 enter b value5 a= 5 b= 3

check the no divisible by 5 or not	**Output**

def div():
  n=eval(input("enter n value"))
  if(n%5==0):
  print("the number is divisible by 5")
  else:
  print("the number not divisible by 5")
div()
	enter n value10
	the number is divisible by 5

find reminder and quotient of given no	**Output**

def reminder():
  a=eval(input("enter a"))
  b=eval(input("enter b"))
  R=a%b
  print("the reminder is",R)
def quotient():
  a=eval(input("enter a"))
  b=eval(input("enter b"))
  Q=a/b
  print("the reminder is",Q)
reminder()
quotient()
	enter a 6
	enter b 3
	the reminder is 0
	enter a 8
	enter b 4
	the reminder is 2.0

convert the temperature	**Output**

def ctof():
  c=eval(input("enter temperature in centigrade"))
  f=(1.8*c)+32
  print("the temperature in Fahrenheit is",f)
def ftoc():
  f=eval(input("enter temp in Fahrenheit"))
  c=(f-32)/1.8
  print("the temperature in centigrade is",c)
ctof()
ftoc()
	enter temperature in centigrade 37
	the temperature in Fahrenheit is 98.6
	enter temp in Fahrenheit 100
	the temperature in centigrade is 37.77

program for basic calculator	Output

def add():
a=eval(input("enter a value"))
b=eval(input("enter b value"))
c=a+b
print("the sum is",c)
def sub():
a=eval(input("enter a value"))
b=eval(input("enter b value"))
c=a-b
print("the diff is",c)
def mul():
a=eval(input("enter a value"))
b=eval(input("enter b value"))
c=a*b
print("the mul is",c)
def div():
a=eval(input("enter a value"))
b=eval(input("enter b value"))
c=a/b
print("the div is",c)
add()
sub()
mul()
div()
	enter a value 10
	enter b value 10
	the sum is 20
	enter a value 10
	enter b value 10
	the diff is 0
	enter a value 10
	enter b value 10
	the mul is 100
	enter a value 10
	enter b value 10
	the div is 1

Possible Questions

Part B (2 Marks)

1. What is mean by problem solving?
2. List down the problem solving techniques?
3. Define algorithm?
4. What are the properties of algorithm?
5. List down the equalities of good algorithm?
6. Define statements?
7. Define state?
8. What is called control flow?
9. What is called sequence execution?
10. Define iteration?
11. What is mean by flow chart?
12. List down the basic symbols for drawing flowchart?

13. List down the rules for drawing the flowchart?
14. What are the advantages of flowchart?
15. What are the disadvantages of flowchart?
16. Define pseudo code?
17. List down the keywords used in writing pseudo code?
18. Mention the advantages of using pseudo code?
19. Mention the disadvantages of using pseudo code?
20. What are the ways available to represent algorithm?
21. Differentiate flowchart and pseudo code?
22. Differentiate algorithm and pseudo code?
23. What is programming language?
24. Mention the types of programming language?
25. What is mean by machine level language?
26. What are the advantages and disadvantages of machine level language?
27. What is high level programming language and mention its advantages?
28. What are the steps in algorithmic problem solving?
29. Write the algorithm for any example?
30. Draw the flow chart for any example?
31. Write pseudo code for any example?

Part C (6 Marks)

1. Explain in detail about problem solving techniques?
2. Explain in detail about building blocks of algorithm?
3. Discuss the symbols and rules for drawing flowchart with the example?
4. Explain in detail about programming language?
5. Discuss briefly about algorithmic problem solving?
6. Write algorithm, pseudo code and flow chart for any example?
7. Explain in detail about simple strategies for developing algorithms?

DEPARTMENT OF CS, CA & IT

UNIT - I : (Objective Type Multiple choice Questions each Question carries one Mark)

PROGRAMMING IN PYTHON (18ITU304B)

PART - A (Online Examination)

Questions	Opt1	opt2	opt3	opt4	KEY
Last step in process of problem solving is to	design a solution	define a problem	practicing the sol	organizing the data	practicing the solution
Second step in problem solving process is to	design a solution	define a problem	practicing the sol	organizing the data	design a solution
Thing to keep in mind while solving a problem is	input data	output data	stored data	all of above	all of above
First step in process of problem solving is to	design a solution	define a problem	practicing the sol	organizing the data	define a problem
Error in a program is called	bug	debug	virus	noise	bug
Error which occurs when program tried to read from	execution error me	built in messages	user-defined mes	half messages	execution error messag
_____ is the process of formulating a probler	problem solving:	Recover	Format	Retrieve	problem solving
_____ is a set of instructions that specifies a	Process	Program	Syntax	Error	Program
_____ is an error in a program that makes it do s	Syntax error	Semantic error	Run time error	Logical Error	Semantic error
_____ is an error in a program that makes	Syntax error	Semantic error	Run time error	Logical Error	Syntax error
_____ is an error that does not occur until	Syntax error	Semantic error	Run time error	Logical Error	Run time error
_____ is an another name for Run time error	Syntax error	Semantic error	Exception	Semantics	Exception
Which translate a program written in a high-level la	Compile	Interpret	Script	bug	Compile
Which execute a program in a high-level language l	Compile	Interpret	Script	bug	Interpret
_____ is a program in a high-level language bef	executable	source code	object code	program	source code
_____ is the output of the compiler after it tran	Coding	source code	object code	program	object code
_____ is the structure of a program	Syntax	Source	Semantics	Algorithm	Syntax
What is a property of a program that can run on m	executable	source code	Coding	Portability	Portability
_____ is another name for object code that is	executable	source code	Coding	program	executable
_____ is the meaning of a program	Syntax	Source	Semantics	Algorithm	Semantics
_____ is to examine a program and analyze th	Syntax	Source	Semantics	parse	parse
_____ is the process of finding and removing ar	bug	debugging	virus	noise	debugging

es

KARPAGAM ACADEMY OF HIGHER EDUCATION
CLASS: II B.Sc IT COURSE NAME: Programming in Python
COURSE CODE: 18ITU304B UNIT: II (Data, Expressions and Statements) BATCH-2019-2022

SYLLABUS

Python interpreter and interactive mode; values and types: int, float, boolean, string, and list; variables, expressions, statements, tuple assignment, precedence of operators, comments; Modules and functions, function definition and use, flow of execution, parameters and arguments; Illustrative programs: exchange the values of two variables, circulate the values of n variables, distance between two points.

1. INTRODUCTION TO PYTHON:

Python is a general-purpose interpreted, interactive, object-oriented, and high- level programming language.

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

- **Python is interpreted**: Python is processed at runtime by the interpreter. You do not need to compile your program before executing it.
- **Python is Interactive**: You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented**: Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Python is a Beginner's Language:** Python is a great language for the beginner-level programmers and supports the development of a wide range of applications.

1.1. Python Features:

- **Easy-to-learn:** Python is clearly defined and easily readable. The structure of the program is very simple. It uses few keywords.
- **Easy-to-maintain:** Python's source code is fairly easy-to-maintain.
- **Portable:** Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Interpreted:** Python is processed at runtime by the interpreter. So, there is no need to compile a program before executing it. You can simply run the program.
- **Extensible:** Programmers can embed python within their C,C++,Java script ,ActiveX, etc.
- **Free and Open Source:** Anyone can freely distribute it, read the source code, and edit it.
- **High Level Language:** When writing programs, programmers concentrate on solutions of the current problem, no need to worry about the low level details.
- **Scalable:** Python provides a better structure and support for large programs than shell scripting.

1.2. Applications:

- Bit Torrent file sharing
- Google search engine, Youtube
- Intel, Cisco, HP, IBM
- i–Robot
- NASA
- Facebook, Drop box

1.3. Python interpreter:

Interpreter: To execute a program in a high-level language by translating it one line at a time.
Compiler: To translate a program written in a high-level language into a low-level language all at once, in preparation for later execution.

Compiler	Interpreter
Compiler Takes **Entire** program as input	Interpreter Takes **Single** instruction as input
Intermediate Object Code is **Generated**	**No** Intermediate Object Code is **Generated**
Conditional Control Statements are Executes **faster**	Conditional Control Statements are Executes **slower**
Memory Requirement is **More**(Since Object Code is Generated)	**Memory Requirement** is **Less**
Program need not be **compiled** every time	Every time higher level program is converted into lower level program
Errors are displayed after **entire program** is checked	**Errors** are displayed for **every instruction** interpreted (if any)
Example : C Compiler	**Example** : PYTHON

1.4 MODES OF PYTHON INTERPRETER:

Python Interpreter is a program that reads and executes Python code. It uses 2 modes of Execution.

1. Interactive mode
2. Script mode

Interactive mode:

- Interactive Mode, as the name suggests, allows us to interact with OS.
- When we type Python statement, **interpreter displays the result(s) immediately.**

Advantages:

- Python, in interactive mode, is good enough to learn, experiment or explore.
- Working in interactive mode is convenient for beginners and for testing small pieces of code.

Drawback:

- We cannot save the statements and have to retype all the statements once again to re-run them.

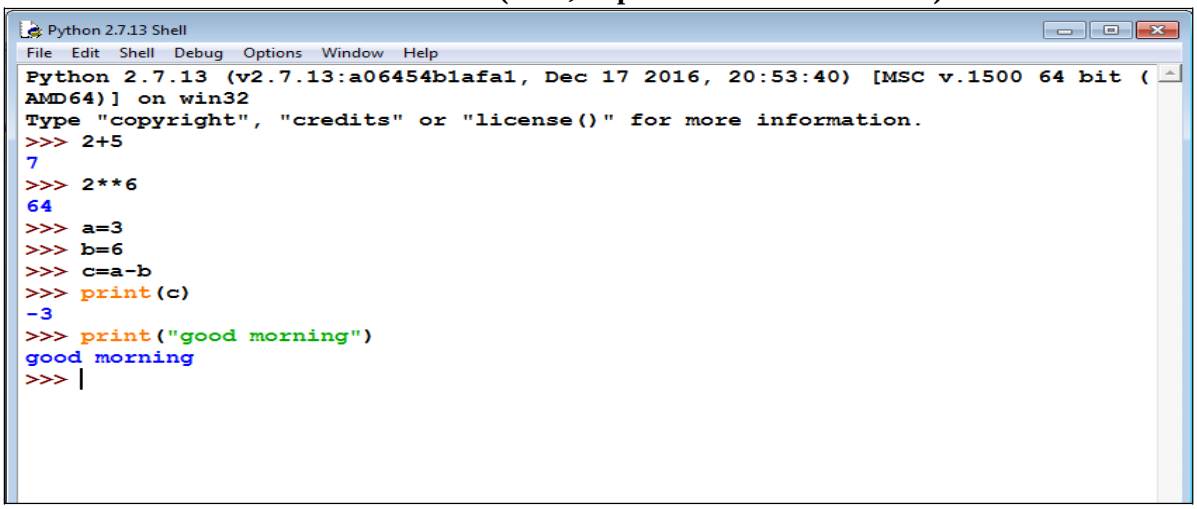
In interactive mode, you type Python programs and the interpreter displays the result:

```
>>> 1 + 1
2
```

The chevron, >>>, is the prompt the interpreter uses to indicate that it is ready for you to enter code. If you type 1 + 1, the interpreter replies 2.

```
>>> print ('Hello, World!')
Hello, World!
```

This is an example of a print statement. It displays a result on the screen. In this case, the result is the words.



Script mode:

- In script mode, we type python program in a file and then use interpreter to execute the content of the file.
- Scripts can be saved to disk for future use. **Python scripts have the extension .py**, meaning that the filename ends with **.py**
- Save the code with **filename.py** and run the interpreter in script mode to execute the script.

Example1:

```
print(1)
x = 2
print(x)
```

Output:

```
>>>1
  2
```

Interactive mode	Script mode
A way of using the Python interpreter by typing commands and expressions at the prompt.	A way of using the Python interpreter to read and execute statements in a script.
Cant save and edit the code	Can save and edit the code
If we want to experiment with the code, we can use interactive mode.	If we are very clear about the code, we can use script mode.
we cannot save the statements for further use and we have to retype all the statements to re-run them.	we can save the statements for further use and we no need to retype all the statements to re-run them.
We can see the results immediately.	We cant see the code immediately.

Integrated Development Learning Environment (IDLE):

- Is a **graphical user interface** which is completely written in Python.
- It is bundled with the default implementation of the python language and also comes with optional part of the Python packaging.

Features of IDLE:

- Multi-window text editor with syntax highlighting.
- Auto completion with **smart indentation**.
- **Python shell** to display output with syntax highlighting.

2.VALUES AND DATA TYPES

Value:

Value can be any letter ,number or string.

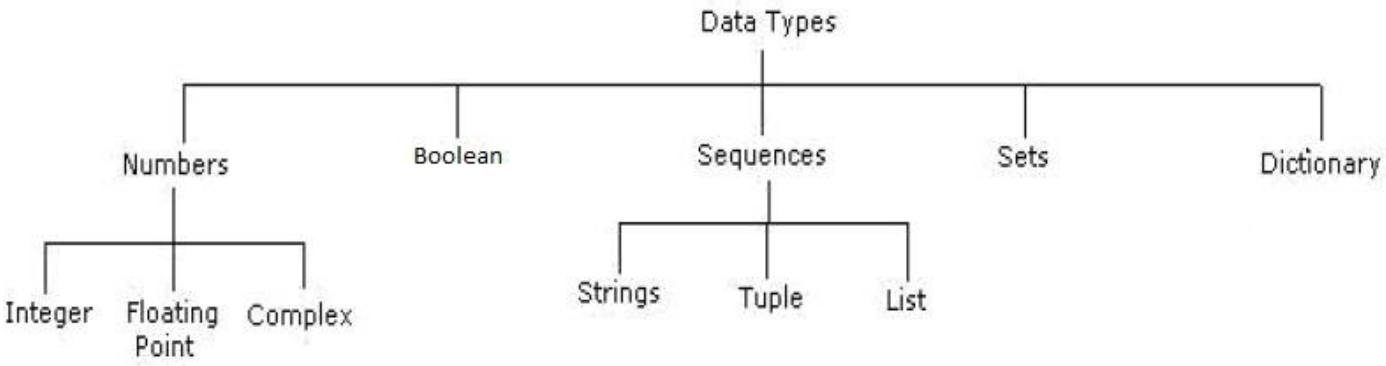
Eg, Values are 2, 42.0, and 'Hello, World!'. (These values belong to different datatypes.)

Data type:

Every value in Python has a data type.

It is a set of values, and the allowable operations on those values.

Python has four standard data types:



2.1 Numbers:

- Number data type stores **Numerical Values.**
- This data type is immutable [i.e. values/items cannot be changed].
- Python supports integers, floating point numbers and complex numbers. They are defined as,

Integers	Long	Float	Complex
- They are often called just integers or **int**. - They are positive or negative whole numbers with no decimal point.	-They are long integers. -They can also be represented in octal and hexadecimal representation.	-They are written with a decimal point dividing the integer and the fractional parts.	-They are of the form **a + bj**, where a and b are floats and j represents the square root of -1 (which is an imaginary number). -The real part of the number is a, and the imaginary part is b.
Eg, 56	**Eg, 5692431L**	**Eg, 56.778**	**Eg, square root of -1 is a complex number**

2.2 Sequence:

- A sequence is an **ordered collection of items**, indexed by positive integers.
- It is a combination **of mutable** (value can be changed) **and immutable** (values cannot be changed) data types.
- There are three types of sequence data type available in Python, they are
  1. **Strings**
  2. **Lists**
  3. **Tuples**

2.2.1 Strings:

- A String in Python consists of a series or sequence of characters - letters, numbers, and special characters.
- Strings are marked by quotes:
  - single quotes (' ') Eg, 'This a string in single quotes'
  - double quotes (" ") Eg, "'This a string in double quotes'"
  - triple quotes(""" """) Eg, This is a paragraph. It is made up of multiple lines and sentences."""
- Individual character in a string is accessed using a subscript (index).
- Characters can be accessed using indexing and slicing operations

Strings are immutable i.e. the contents of the string cannot be changed after it is created.

Indexing:

String A	H	E	L	L	O
Positive Index	0	1	2	3	4
Negative Index	-5	-4	-3	-2	-1

- Positive indexing helps in accessing the string from the beginning
- Negative subscript helps in accessing the string from the end.
- Subscript 0 or –ve n(where n is length of the string) displays the first element.
  Example: A[0] or A[-5] will display "H"
- Subscript 1 or –ve (n-1) displays the second element.
  Example: A[1] or A[-4] will display "E"

Operations on string:

i. Indexing
ii. Slicing
iii. Concatenation
iv. Repetitions
v. Member ship

Creating a string	>>> s="good morning"	Creating the list with elements of different data types.

Indexing	>>> **print(s[2])**
	o
	>>> **print(s[6])**
	O
		❖ Accessing the item in the position 0
		❖ Accessing the item in the position 2

Slicing(ending position -1)
Slice operator is used to extract part of a data
type
	>>> **print(s[2:])**
	od morning
	>>> **print(s[:4])**
	Good
		- Displaying items from 2nd till last.
		- Displaying items from 1st position till 3rd .

Concatenation	>>>**print(s+"friends")** good morningfriends	-Adding and printing the characters of two strings.
Repetition	**>>>print(s*2)** good morninggood morning	Creates new strings, concatenating multiple copies of the same string
in, not in (membership operator)	>>> s="good morning" **>>>"m" in s True** **>>> "a" not in s** **True**	Using membership operators to check a particular character is in string or not. Returns true if present.

2.2.2 Lists

- ❖ List is an ordered sequence of items. Values in the list are called elements / items.
- ❖ It can be written as a list of comma-separated items (values) between **square brackets[ ]**.
- ❖ Items in the lists can be of different data types.

Operations on list:

Indexing Slicing
Concatenation
Repetitions
Updation, Insertion, Deletion

Creating a list	>>>**list1=["python", 7.79, 101, "hello"]** >>>**list2=["god",6.78,9]**	Creating the list with elements of differentdata types.

Indexing	>>>**print(list1[0])** python >>> **list1[2]** 101	❖ Accessing the item in the position 0 ❖ Accessing the item in the position 2
Slicing(ending position -1) Slice operator is used to extract part of a string, or some part of a list Python	>>> **print(list1[1:3])** [7.79, 101] >>>**print(list1[1:])** [7.79, 101, 'hello']	- Displaying items from 1st till 2nd. - Displaying items from 1st position till last.
Concatenation	>>>**print(list1+list2)** ['python', 7.79, 101, 'hello', 'god', 6.78, 9]	-Adding and printing the items of two lists.
Repetition	>>> **list2*3** **['god', 6.78, 9, 'god', 6.78, 9, 'god', 6.78, 9]**	Creates new strings, concatenating multiple copies of the same string
Updating the list	>>> **list1[2]=45** >>>print(list1) [‘python’, 7.79, 45, ‘hello’]	Updating the list using index value

Inserting an element	>>> **list1.insert(2,"program")**
	>>> print(list1)
	['python', 7.79, 'program', 45, 'hello']
		Inserting an element in 2nd position

Removing an element	>>> **list1.remove(45)**
	>>> print(list1)
	['python', 7.79, 'program', 'hello']
		Removing an element by giving the element directly

2.2.4 Tuple:

- ❖ A tuple is same as list, except that the set of elements is **enclosed in parentheses** instead of square brackets.
- ❖ **A tuple is an immutable list.** i.e. once a tuple has been created, you can't add elements to a tuple or remove elements from the tuple.
- ❖ Benefit of Tuple:
- ❖ Tuples are faster than lists.
- ❖ If the user wants to protect the data from accidental changes, tuple can be used.
- ❖ Tuples can be used as keys in dictionaries, while lists can't.

Basic Operations:

Creating a tuple	>>>**t=("python", 7.79, 101, "hello")**	Creating the tuple with elements of different data types.
Indexing	>>>**print(t[0])** python >>> **t[2]** 101	❖ Accessing the item in the position 0 ❖ Accessing the item in the position 2
Slicing(ending position -1)	>>>**print(t[1:3])** (7.79, 101)	❖ Displaying items from 1st till 2nd.
Concatenation	>>> **t+("ram", 67)** ('python', 7.79, 101, 'hello', 'ram', 67)	❖ Adding tuple elements at the end of another tuple elements
Repetition	>>>**print(t*2)** ('python', 7.79, 101, 'hello', 'python', 7.79, 101, 'hello')	❖ Creates new strings, concatenating multiple copies of the same string

Altering the tuple data type leads to error. Following error occurs when user tries to do.

```
>>> t[0]="a"
Trace back (most recent call last):
  File "<stdin>", line 1, in <module>
Type Error: 'tuple' object does not support item assignment
```

2.3 Mapping

-This data type is unordered and mutable.

-Dictionaries fall under Mappings.

2.3.1 Dictionaries:

- ❖ Lists are ordered sets of objects, whereas **dictionaries are unordered sets**.
- ❖ Dictionary is created by using **curly brackets**. i,e. {}
- ❖ Dictionaries **are accessed via keys** and not via their position.
- ❖ 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(**The association of a key and a value is called a key- value pair)

Dictionaries don't support the sequence operation of the sequence data types like strings, tuples and lists.

Creating a dictionary	**>>> food = {"ham":"yes", "egg" : "yes", "rate":450 }** >>>print(food) {'rate': 450, 'egg': 'yes', 'ham': 'yes'}	Creating the dictionary with elements of different data types.
Indexing	>>>> **print(food["rate"])** 450	Accessing the item with keys.
Slicing(ending position -1)	>>>**print(t[1:3])** (7.79, 101)	Displaying items from 1st till 2nd.

If you try to access a key which doesn't exist, you will get an error message:

```
>>> words = {"house" : "Haus", "cat":"Katze"}
>>> words["car"]
Traceback (most recent call last): File
"<stdin>", line 1, in <module> KeyError:
'car'
```

Data type	Compile time	Run time
int	a=10	a=int(input("enter a"))
float	a=10.5	a=float(input("enter a"))
string	a="panimalar"	a=input("enter a string")
list	a=[20,30,40,50]	a=list(input("enter a list"))
tuple	a=(20,30,40,50)	a=tuple(input("enter a tuple"))

3.Variables,Keywords Expressions, Statements, Comments, Docstring ,Lines And Indentation, Quotation In Python, Tuple Assignment:

3.1VARIABLES:

- A variable allows us to store a value by assigning it to a name, which can be used later.
- Named memory locations to store values.
- Programmers generally choose names for their variables that are meaningful.
- It can be of any length. No space is allowed.
- We don't need to declare a variable before using it. In Python, we simply assign a value to a variable and it will exist.

Assigning value to variable:

Value should be given on the right side of assignment operator(=) and variable on left side.

```
>>>counter =45
print(counter)
```

Assigning a single value to several variables simultaneously:

```
>>> a=b=c=100
```

Assigning multiple values to multiple variables:

```
>>> a,b,c=2,4,"ram"
```

3.2KEYWORDS:

- Keywords are the reserved words in Python.
- We cannot use a keyword as variable name, function name or any other identifier.
- They are used to define the syntax and structure of the Python language.
- Keywords are case sensitive.

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*	

3.3 IDENTIFIERS:

Identifier is the name given to entities like class, functions, variables etc. in Python.

- Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore (_).
- all are valid example.
- An identifier cannot start with a digit.
- Keywords cannot be used as identifiers.
- Cannot use special symbols like !, @, #, $, % etc. in our identifier.
- Identifier can be of any length.

Example:

Names like myClass, var_1, and **this_is_a_long_variabl**e

Valid declarations	Invalid declarations
Num Num Num1 _NUM NUM_temp2 IF Else	Number 1 num 1 addition of program 1Num Num.no if else

3.4 STATEMENTS AND EXPRESSIONS:

3.4.1 Statements:

-Instructions that a Python interpreter can executes are called statements.

-A statement is a unit of code like creating a variable or displaying a value.

```
>>> n = 17
>>> print(n)
```

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

3.4.2 Expressions:

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

- A value all by itself is considered an expression, and also a variable.
- So the following are all legal expressions:

```
>>> 42
42
>>> a=2
>>> a+3+2 7
>>> z=("hi"+"friend")
>>> print(z) hifriend
```

3.5 INPUT AND OUTPUT

INPUT: Input is data entered by user (end user) in the program. In python, **input () function** is available for input.

Syntax for input() is:

```
variable = input ("data")
```

Example:

```
>>> x=input("enter the name:")
enter the name: george
>>>y=int(input("enter the number"))
enter the number 3
```

#python accepts string as default data type. conversion is required for type.

OUTPUT: Output can be displayed to the user using Print statement .

Syntax:

```
print (expression/constant/variable)
```

Example:

```
>>> print ("Hello")
Hello
```

3.6 COMMENTS:

- A **hash sign (#)** is the beginning of a comment.
- Anything written after # in a line is ignored by interpreter.

Eg:percentage = (minute * 100) / 60 **# calculating percentage of an hour**

- Python **does not have multiple-line commenting feature.** You have to comment each line individually as follows :

Example:

```
# This is a comment.
# This is a comment, too. # I
said that already.
```

3.7 DOCSTRING:

- Docstring is short for documentation string.
- It is a string that occurs as the first statement in a module, function, class, or method definition. We must write what a function/class does in the docstring.
- **Triple quotes** are used while writing docstrings.

Syntax:

functionname__doc.__ Example:

```
def double(num):
    """Function to double the value"""
    return 2*num
>>> print(double.__doc__)
Function to double the value
```

3.8 LINES AND INDENTATION:

- Most of the programming languages like C, C++, Java use braces { } to define a block of code. But, python uses indentation.
- Blocks of code are denoted by line indentation.
- It is a space given to the block of codes for class and function definitions or flow control.

Example:

```
a=3
b=1
if a>b:
  print("a is greater")
else:
  print("b is greater")
```

3.9 QUOTATION IN PYTHON:

Python accepts single ('), double (") and triple ('" or """) quotes to denote string literals.
Anything that is represented using quotations are considered as string.

- single quotes (' ') Eg, 'This a string in single quotes'
- double quotes (" ") Eg, '"This a string in double quotes"'
- triple quotes(""" """) Eg, This is a paragraph. It is made up of multiple lines and sentences."""

3.10 TUPLE ASSIGNMENT

- An assignment to all of the elements in a tuple using a single assignment statement.
- 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.
- The left side is a tuple of variables; the right side is a tuple of values.
- Each value is assigned to its respective variable.
- All the expressions on the right side are evaluated before any of the assignments. This feature makes tuple assignment quite versatile.
- Naturally, the number of variables on the left and the number of values on the right have to be the same.

```
>>> (a, b, c, d) = (1, 2, 3)
ValueError: need more than 3 values to unpack
```

Example:

-It is useful to swap the values of two variables. With **conventional assignment statements**, we have to use a temporary variable. For example, to swap a and b:

Swap two numbers	Output:
a=2;b=3 print(a,b) temp = a a = b b = temp print(a,b)	(2, 3) (3, 2) >>>

-Tuple assignment solves this problem neatly:

```
(a, b) = (b, a)
```

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

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

```
>>> b = ("George", 25, "20000")    # tuple packing
```

-In tuple unpacking, **the values in a tuple on the right are 'unpacked' into the variables/names on the right:**

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

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

Example:

-To split an email address in to user name and a domain

```
>>> mailid='god@abc.org'
>>> name,domain=mailid.split('@')
>>> print name
god
>>> print (domain) abc.org
```

4.OPERATORS:

- Operators are the constructs which can manipulate the value of operands.
- Consider the **expression *4 + 5 = 9.*** Here, **4 and 5 are called operands** and **+ is called operator**
- Types of Operators:

-Python language supports the following types of operators

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

4.1 Arithmetic operators:

They are used to perform **mathematical operations** like addition, subtraction, multiplication etc.

Assume, a=10 and b=5

Operator	Description	Example
+ Addition	Adds values on either side of the operator.	a + b = 30
- Subtraction	Subtracts right hand operand from left hand operand.	a – b = -10

* Multiplication	Multiplies values on either side of the operator	a * b = 200
/ Division	Divides left hand operand by right hand operand	b / a = 2
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 0
** Exponent	Performs exponential (power) calculation on operators	a**b =10 to the power 20
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed	5//2=2

Examples

```
a=10
b=5 print("a+b=",a+b)
print("a-b=",a-b)
print("a*b=",a*b)
print("a/b=",a/b)
print("a%b=",a%b)
print("a//b=",a//b)
print("a**b=",a**b)
```

Output:

```
a+b= 15
a-b= 5
a*b= 50
a/b= 2.0
a%b= 0
a//b= 2
a**b= 100000
```

4.2 Comparison (Relational) Operators:

- Comparison operators are used to compare values.
- It either returns True or False according to the condition. **Assume, a=10 and b=5**

Operator	Description	Example
==	If the values of two operands are equal, then the condition	(a == b) is
	becomes true.	not true.
!=	If values of two operands are not equal, then condition becomes true.	(a!=b) is true
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.

<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

Example

```
a=10
b=5 print("a>b=>",a>b)
print("a>b=>",a<b)
print("a==b=>",a==b)
print("a!=b=>",a!=b)
print("a>=b=>",a<=b)
print("a>=b=>",a>=b)
```

Output:
```
a>b=>
True a>b=>
False a==b=>
False a!=b=>
True a>=b=>
False a>=b=>
True
```

4.3 Assignment Operators:

-Assignment operators are used in Python to assign values to variables.

Operator	Description	Example
=	Assigns values from right side operands to left side operand	c = a + b assigns value of a + b into c
+= Add AND	It adds right operand to the left operand and assign the result to left operand	c += a is equivalent to c = c + a
-= Subtract AND	It subtracts right operand from the left operand and assign the result to left operand	c -= a is equivalent to c = c - a
*= Multiply AND	It multiplies right operand with the left operand and assign the result to left operand	c *= a is equivalent to c = c * a

/= Divide AND	It divides left operand with the right operand and assign the result to left operand	c /= a is equivalent to c = c / ac /= a is equivalent to c = c / a
%= Modulus AND	It takes modulus using two operands and assign the result to left operand	c %= a is equivalent to c = c % a
**= Exponent AND	Performs exponential (power) calculation on operators and assign value to the left operand	c **= a is equivalent to c = c ** a
//= Floor Division	It performs floor division on operators and assign value to the left operand	c //= a is equivalent to c = c // a

Example

```
a = 21
b = 10
c = 0
c = a + b
print("Line 1 - Value of c is ", c) c +=
a
print("Line 2 - Value of c is ", c) c *=
a
print("Line 3 - Value of c is ", c) c /=
a
print("Line 4 - Value of c is ", c) c = 2
c %= a
print("Line 5 - Value of c is ", c) c
**= a
print("Line 6 - Value of c is ", c) c //=
a
print("Line 7 - Value of c is ", c)
```

Output

```
Line 1 - Value of c is 31 Line 2 -
Value of c is 52 Line 3 - Value of
c is 1092 Line 4 - Value of c is
52.0 Line 5 - Value of c is  2
Line 6 - Value of c is 2097152 Line 7 -
Value of c is  99864
```

4.4 Logical Operators:

-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

Example a = True b = False
print('a and b is',a and b)
print('a or b is',a or b) print('not a is',not a)

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

4.5 Bitwise Operators:

- A **bitwise operation** operates on one or more **bit** patterns at the level of individual bits

Example: Let x = 10 (0000 1010 in binary) and
y = 4 (0000 0100 in binary)

Operator	Meaning	Example
&	Bitwise AND	x& y = 0 (0000 0000)
|	Bitwise OR	x | y = 14 (0000 1110)
~	Bitwise NOT	~x = -11 (1111 0101)
^	Bitwise XOR	x ^ y = 14 (0000 1110)
>>	Bitwise right shift	x>> 2 = 2 (0000 0010)
<<	Bitwise left shift	x<< 2 = 40 (0010 1000)

Example

```
a = 60            # 60 = 0011 1100
b = 13            # 13 = 0000 1101
c = 0
c = a & b;        # 12 = 0000 1100
print "Line 1 - Value of c is ", c c = a
| b;              # 61 = 0011 1101
print "Line 2 - Value of c is ", c c = a
^ b;              # 49 = 0011 0001
print "Line 3 - Value of c is ", c
c = ~a;           # -61 = 1100 0011
print "Line 4 - Value of c is ", c
c = a << 2;       # 240 = 1111 0000
print "Line 5 - Value of c is ", c
c = a >> 2;       # 15 = 0000 1111
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

4.6 Membership Operators:

- Evaluates to find a value or a variable is in the specified sequence of string, list, tuple, dictionary or not.
- Let, **x=[5,3,6,4,1].** To check particular item in list or not**, in and not in** operators are used.

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

Example:

```
x=[5,3,6,4,1]
>>> 5 in x
True
>>> 5 not in x
False
```

4.7 Identity Operators:

- They are used to check if two values (or variables) are located on the same part of the memory.

Operator	Meaning	Example
is	True if the operands are identical (refer to the same object)	x is True
is not	True if the operands are not identical (do not refer to the same object)	x is not True

Example

```
x = 5
y = 5
x2 = 'Hello' y2
= 'Hello'
print(x1 is not y1)
print(x2 is y2)
```

Output

False True

5. OPERATOR PRECEDENCE:

When an expression contains **more than one operator, the order of evaluation** depends on the order of operations.

Operator	Description
**	Exponentiation (raise to the power)
~ + -	Complement, unary plus and minus (method names for the last two are +@ and -@)
* / % //	Multiply, divide, modulo and floor division
+ -	Addition and subtraction
>> <<	Right and left bitwise shift
&	Bitwise 'AND'
^ |	Bitwise exclusive `OR' and regular `OR'
<= < > >=	Comparison operators
<> == !=	Equality operators
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators

in not in	Membership operators
not or and	Logical operators

-For mathematical operators, Python follows mathematical convention.
-The acronym **PEMDAS** (Parentheses, Exponentiation, Multiplication, Division, Addition, Subtraction**)** is a useful way to remember the rules:

- Parentheses have the highest precedence and can be used to force an expression to evaluate in the order you want. Since expressions in parentheses are evaluated first, 2 * (3-1)is 4, and (1+1)**(5-2) is 8.
- You can also use parentheses to make an expression easier to read, as in (minute * 100) / 60, even if it doesn't change the result.
- Exponentiation has the next highest precedence, so 1 + 2**3 is 9, not 27, and 2 *3**2 is 18, not 36.
- Multiplication and Division have higher precedence than Addition and Subtraction. So 2*3-1 is 5, not 4, and 6+4/2 is 8, not 5.
- Operators with the same precedence are evaluated from left to right (except exponentiation).

Example:

a=9-12/3+3*2-1 a=? a=9-4+3*2-1 a=9-4+6-1 a=5+6-1 a=11-1 **a=10**	A=2*3+4%5-3/2+6 A=6+4%5-3/2+6 A=6+4-3/2+6 A=6+4-1+6 A=10-1+6 A=9+6 **A=15**	find m=? m=-43||8&&0||-2 m=-43||0||-2 m=1||-2 **m=1**
a=2,b=12,c=1 d=a<b>c d=2<12>1 d=1>1 **d=0**	a=2,b=12,c=1 d=a<b>c-1 d=2<12>1-1 d=2<12>0 d=1>0 **d=1**	a=2*3+4%5-3//2+6 a=6+4-1+6 a=10-1+6 a**=15**

6.Functions, Function Definition And Use, Function call, Flow Of Execution, Function Prototypes, Parameters And Arguments, Return statement, Argumentstypes,Modules

6.1 FUNCTIONS:

- **Function is a sub program which consists of set of instructions used to perform a specific task. A large program is divided into basic building blocks called function.**

Need For Function:

- When the program is too complex and large they are divided into parts. Each part is separately coded and combined into single program. Each subprogram is called as function.
- Debugging, Testing and maintenance becomes easy when the program is divided into subprograms.
- Functions are used to avoid rewriting same code again and again in a program.
- Function provides code re-usability
- The length of the program is reduced.

Types of function:

Functions can be classified into two categories:

i) user defined function
ii) Built in function

i) Built in functions

- Built in functions are the functions that are already created and stored in python.
- These built in functions are always available for usage and accessed by a programmer. It cannot be modified.

Built in function	Description
>>>**max(3,4)** 4	# returns largest element
>>>**min(3,4)** 3	# returns smallest element
>>>**len("hello")** 5	#returns length of an object
>>>**range(2,8,1)** [2, 3, 4, 5, 6, 7]	#returns range of given values
>>>**round(7.8)** 8.0	#returns rounded integer of the given number
>>>**chr(5)** \x05'	#returns a character (a string) from an integer
>>>**float(5)** 5.0	#returns float number from string or integer
>>>**int(5.0)** 5	# returns integer from string or float
>>>**pow(3,5)** 243	#returns power of given number
>>>**type(5.6)** <type 'float'>	#returns data type of object to which it belongs
>>>**t=tuple([4,6.0,7])** (4, 6.0, 7)	# to create tuple of items from list
>>>**print("good morning")** Good morning	# displays the given object

>>>**input("enter name: ")** enter name : George	# reads and returns the given string

ii) User Defined Functions:

- ❖ User defined functions are the functions that programmers create for their requirement and use.
- ❖ These functions can then be **combined to form module** which can be used in other programs by importing them.
- ❖ Advantages of user defined functions:
  - Programmers working on large project can divide the workload by making different functions.
  - If repeated code occurs in a program, function can be used to include those codes and execute when needed by calling that function.

6.2 Function definition: (Sub program)

- ❖ def keyword is used to define a function.
- ❖ Give the function name after def keyword followed by parentheses in which arguments are given.
- ❖ End with colon (:)
- ❖ Inside the function add the program statements to be executed
- ❖ End with or without return statement

Syntax:

```
def fun_name(Parameter1,Parameter2…Parameter n): statement1
        statement2…
        statement n return[expression]
```

Example:

```
def my_add(a,b):
        c=a+b
        return c
```

6.3Function Calling: (Main Function)

- ➢ Once we have defined a function, we can call it from another function, program or even the Python prompt.
- ➢ To call a function we **simply type the function name with appropriate arguments.**

Example:

```
x=5
y=4
my_add(x,y)
```

6.4 Flow of Execution:

- ❖ The order in which statements are executed 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.
- Function definitions do not alter the flow of execution of the program, but remember that statements inside the function are not executed until the function is called.
- Function calls are like a bypass in the flow of execution. Instead of going to the next statement, the flow jumps to the first line of the called function, executes all the statements there, and then comes back to pick up where it left off.

Note: When you read a program, don't read from top to bottom. Instead, follow the flow of execution. This means that you will read the **def** statements as you are scanning from top to bottom, but you should skip the statements of the function definition until you reach a point where that function is called.

6.5 Function Prototypes:

i. Function without arguments and without return type
ii. Function with arguments and without return type
iii. Function without arguments and with return type
iv. Function with arguments and with return type

i) Function without arguments and without return type
- In this type no argument is passed through the function call and no output is return to main function
- The sub function will read the input values perform the operation and print the result in the same block

ii) Function with arguments and without return type
- Arguments are passed through the function call but output is not return to the main function

iii) Function without arguments and with return type
- In this type no argument is passed through the function call but output is return to the main function.

iv) Function with arguments and with return type
- In this type arguments are passed through the function call and output is return to the main function

Without Return Type	
Without argument	**With argument**

def add():
  a=int(input("enter a"))
  b=int(input("enter b"))
  c=a+b
  print(c)
add()
	def add(a,b):
	  c=a+b
	  print(c)
	a=int(input("enter a"))
	b=int(input("enter b")) add(a,b)

OUTPUT: enter a 5 enter b 10 15	OUTPUT: enter a 5 enter b 10 15

With return type	
Without argument	**With argument**
def add(): a=int(input("enter a")) b=int(input("enter b")) c=a+b return c c=add() print(c)	def add(a,b): c=a+b return c a=int(input("enter a")) b=int(input("enter b")) c=add(a,b) print(c)
OUTPUT: enter a 5 enter b 10 15	OUTPUT: enter a 5 enter b 10 15

6.6 Parameters and Arguments:

Parameters:

- Parameters are the value(s) provided in the parenthesis when we write function header.
- These are the values required by function to work.
- If there is more than one value required, all of them will be listed in parameter list separated by **comma.**
- Example: def my_add(a,b):

Arguments :

- Arguments are the value(s) provided in function call/invoke statement.
- List of arguments should be supplied in same way as parameters are listed.
- Bounding of parameters to arguments is done 1:1, and so there should be same number and type of arguments as mentioned in parameter list.
- Example: my_add(x,y)

6.7 RETURN STATEMENT:

- The **return statement is used to exit a function** and go back to the place from where it was called.
- If the return statement has no arguments, then it will not return any values. But exits from function.

Syntax:
return[expression]

Example:
def my_add(a,b): c=a+b
return c x=5
y=4 print(my_add(x,y))
Output:
9

6.8 ARGUMENTS TYPES:

1. Required Arguments
2. Keyword Arguments
3. Default Arguments
4. Variable length Arguments

❖ **Required Arguments:** The number of arguments in the function call should match exactly with the function definition.

```
def my_details( name, age ): print("Name: ",
  name)
  print("Age ", age)
  return
my_details("george",56)
```

Output:

```
Name: george
Age  56
```

❖ **Keyword Arguments:**

Python interpreter is able to use the keywords provided to match the values with parameters even though if they are arranged in out of order.

```
def my_details( name, age ):
  print("Name: ", name)
  print("Age ", age)
  return
my_details(age=56,name="george")
```

Output:

```
Name: george
Age  56
```

❖ **Default Arguments:**

Assumes a default value if a value is not provided in the function call for that argument. def my_details(

```
name, age=40 ):
   print("Name: ", name)
   print("Age ", age) return
my_details(name="george")
```

Output:

```
Name: george
Age  40
```

❖ **Variable length Arguments**

If we want to specify more arguments than specified while defining the function, variable length arguments

are used. It is denoted by * symbol before parameter.

```
def my_details(*name ):
    print(*name)
my_details("rajan","rahul","micheal", ärjun")
```

Output:

```
rajan rahul micheal ärjun
```

6.9 MODULES:

- **A module is a file containing Python definitions ,functions, statements and instructions.**
- Standard library of Python is extended as modules.
- **To use these modules in a program**, **programmer needs to import the module.**
- Once we import a module, we can reference or use to any of its functions or variables in our code.
  - oThere is large number of standard modules also available in python. oStandard modules can be imported the same way as we import our user- defined modules.
  - oEvery module contains many function.
  - o*To access one of the function , you have to specify the name of the module and the name of the function separated by dot .* **This format is called dot notation.**

Syntax:

```
import module_name
module_name.function_name(variable)
```

Importing Builtin Module:	**Importing User Defined Module:**
import math x=math.sqrt(25) print(x)	import cal x=cal.add(5,4) print(x)

There are **four ways to import a module** in our program, they are

Import: It is simplest and most common way to use modules in our code.
Example:
`import math`
`x=math.pi`
`print("The value of pi is", x)`
Output: The value of pi is 3.141592653589793
	from import : It is used to get a specific function in the code instead of complete file.
	Example:
	`from math import pi`
	`x=pi`
	`print("The value of pi is", x)`
	Output: The value of pi is 3.141592653589793

import with renaming:
We can import a module by renaming the module as our wish.
Example:
`import math as m`
`x=m.pi`
`print("The value of pi is", x)`
Output: The value of pi is 3.141592653589793
	import all:
	We can import all names(definitions) form a module using *
	Example:
	`from math import *`
	`x=pi`
	`print("The value of pi is", x)`
	Output: The value of pi is 3.141592653589793

Built-in python modules are,

1.math – mathematical functions:

some of the functions in math module is,

- math.ceil(x) - Return the ceiling of *x*, the smallest integer greater han or equal to *x*
- math.floor(x) - Return the floor of *x*, the largest integer less than or equal to *x*.
- math.factorial(x) -Return *x* factorial. math.gcd(x,y)- Return the greatest common divisor of the integers *a* and *b*
- math.sqrt(x)- Return the square root of *x* math.log(x)-
- return the natural logarithm of *x* math.log10(x) – returns
- the base-10 logarithms math.log2(x) - Return the base-2
- logarithm of *x*. math.sin(x) – returns sin of x radians
- math.cos(x)- returns cosine of x radians math.tan(x)-
- returns tangent of x radians
- math.pi - The mathematical constant π = 3.141592
- math.e – returns The mathematical constant e = 2.718281

2 **.random**-Generate pseudo-random numbers

- random.**randrange**(*stop*) random.**randrange**(*start*, *stop*[, *step*]) random.**uniform**(*a*, *b*)
- -Return a random floating point number

ILLUSTRATIVE PROGRAMS

Program for SWAPPING(Exchanging)of values	**Output**
a = int(input("Enter a value ")) b = int(input("Enter b value ")) c = a a = b b = c print("a=",a,"b=",b,)	Enter a value 5 Enter b value 8 a=8 b=5
Program to find distance between two points	**Output**
import math x1=int(input("enter x1")) y1=int(input("enter y1")) x2=int(input("enter x2")) y2=int(input("enter y2")) distance =math.sqrt((x2-x1)**2)+((y2- y1)**2) print(distance)	enter x1 7 enter y1 6 enter x2 5 enter y2 7 2.5
Program to circulate n numbers	**Output:**
a=list(input("enter the list")) print(a) for i in range(1,len(a),1): print(a[i:]+a[:i])	enter the list '1234' ['1', '2', '3', '4'] ['2', '3', '4', '1'] ['3', '4', '1', '2'] ['4', '1', '2', '3']

Possible Questions

Part A: (2 marks)

1. What is interpreter?
2. What are the two modes of python?
3. List the features of python.
4. List the applications of python
5. List the difference between interactive and script mode
6. What is value in python?
7. What is identifier? and list the rules to name identifier.
8. What is keyword?
9. How to get data types in compile time and runtime?
10. What is indexing and types of indexing?

11. List out the operations on strings.
12. Explain slicing?
13. Explain below operations with the example
    (i)Concatenation (ii)Repetition
14. Give the difference between list and tuple
15. Differentiate Membership and Identity operators.
16. Compose the importance of indentation in python.
17. Evaluate the expression and find the result (a+b)*c/d
    a+b*c/d
18. Write a python program to print 'n' numbers.
19. Define function and its uses
20. Give the various data types in Python
21. Assess a program to assign and access variables.
22. Select and assign how an input operation was done in python.
23. Discover the difference between logical and bitwise operator.
24. Give the reserved words in Python.
25. Give the operator precedence in python.
26. Define the scope and lifetime of a variable in python.
27. Point out the uses of default arguments in python
28. Generalize the uses of python module.
29. Demonstrate how a function calls another function. Justify your answer.
30. List the syntax for function call with and without arguments.
31. Define recursive function.
32. What are the two parts of function definition? give the syntax.
33. Point out the difference between recursive and iterative technique.
34. Give the syntax for variable length arguments.

Part B (6 marks)

1. Explain in detail about various data types in Python with an example?
2. Explain the different types of operators in python with an example.
3. Discuss the need and importance of function in python.
4. Explain in details about function prototypes in python.
5. Discuss about the various type of arguments in python.
6. Explain the flow of execution in user defined function with example.
7. Illustrate a program to display different data types using variables and literal constants.
8. Show how an input and output function is performed in python with an example.
9. Explain in detail about the various operators in python with suitable examples.
10. Discuss the difference between tuples and list
11. Discuss the various operation that can be performed on a tuple and Lists (minimum 5)with an example program
12. What is membership and identity operators.
13. Write a program to perform addition, subtraction, multiplication, integer division, floor division and modulo division on two integer and float.

14. Write a program to convert degree Fahrenheit to Celsius
15. Discuss the need and importance of function in python.
16. Illustrate a program to exchange the value of two variables with temporary variables
17. Briefly discuss in detail about function prototyping in python. With suitable example program
18. Analyze the difference between local and global variables.
19. Explain with an example program to circulate the values of n variables
20. Analyze with a program to find out the distance between two points using python.
21. Do the Case study and perform the following operation in tuples i) Maxima minima iii)sum of two tuples iv) duplicate a tuple v)slicing operator vi) obtaining a list from a tuple vii) Compare two tuples viii)printing two tuples of different data types
22. Write a program to find out the square root of two numbers.

Questions	Option1	Option2	Option3	Option4	Answer
What is answer of this	7	1	0	5	1
What is the output of this	27	9	3	1	3
What datatype is the object below ?	list	dictionary	array	tuple	list
output of the following? print("Hello {name1} and	Hello foo and bin	Hello {name1} and {name2}	Error	Hello	Hello foo and bin
What is the output of the following? print("Hello {name1} and	The sum of 2 and 10 is 12	The sum of 10 and a is 14	The sum of 10 and a is c	Error	The sum of 2 and 10 is 12
What is the result of round(0.5) – round(-0.5)?	1	2	0	3	2
What is the maximum possible length of an identifier?	31 characters	63 characters	79 characters	None	None
All keywords in Python are in	lower case	UPPER CASE	Capitalized	Both lower case and UPPER CASE	Both lower case and UPPER CASE
Which of the following is an invalid statement?	abc = 1,000,000	a b c = 1000 2000 3000	a,b,c = 1000, 2000, 3000	a_b_c = 1,000,000	a b c = 1000 2000 3000
Which of these in not a core datatype?	Lists	Dictionary	Tuples	Class	Class
Select the reserved keyword in python	else	import	raise	All the options	All the options

Which of the following symbols are used for comments in Python?	//	"	/**/	#	#
Which keyword is used to define methods in Python?	function	def	method	All of these	def
Which of the following is correct way to declare string variable in Python?	fruit = 'banana'	fruit = "banana"	fruit = banana	fruit = (banana)	fruit = 'banana'
Which predefined Python function is used to find length of string?	length()	len()	strlen()	stringlength()	len()
Python allows string slicing. What is the output of below code: s='cppbuzz chicago' print(s[3:5])	pbuzz	buzzc	bu	buz	bu
How do you insert COMMENTS in Python code?	/*…*/	//	#	!	#

What is a correct syntax to output "Hello World" in Python?	echo("Hello World")	echo "Hello World"	p("Hello World")	print("Hello World")	print("Hello World")
Which one is NOT a legal variable name?	_myvar	Myvar	my_var	my-var	my_var
How do you create a variable with the numeric value 5?	x=5	x=int(5)	Both x=5 and x=int(5) are correct	x=val(5)	Both x=5 and x=int(5) are correct
What is the correct file extension for Python files?	.pt	.py	.pyth	.pyt	.py
How do you create a variable with the floating number 2.8?	x=2.8	x=foat(2.8)	Both x=2.8 and x=float(2.8) are correct	x=val(2.8)	Both x=2.8 and x=float(2.8) are correct
What is the correct syntax to output the type of a variable or object in Python?	print(typeOf(x))	print(typeof(x))	print(type(x))	print(typeof x))	print(type(x))
What is the correct way to create a function in Python?	def myFunction():	create myFunction():	function myFunction():	func myFunction():	def myFunction():
What is a correct syntax to return the first character in a string?	x=sub("Hello",0,1)	x="Hello"[0]	x="Hello".sub(0,1)		

Select the reserved keyword in python	else	import	raise	all	all
Which of the following is correct way to declare string variable in Python?	fruit = 'banana'	fruit = "banana"	fruit =banana	fruit =(banana)	fruit = 'banana'

SYLABUS

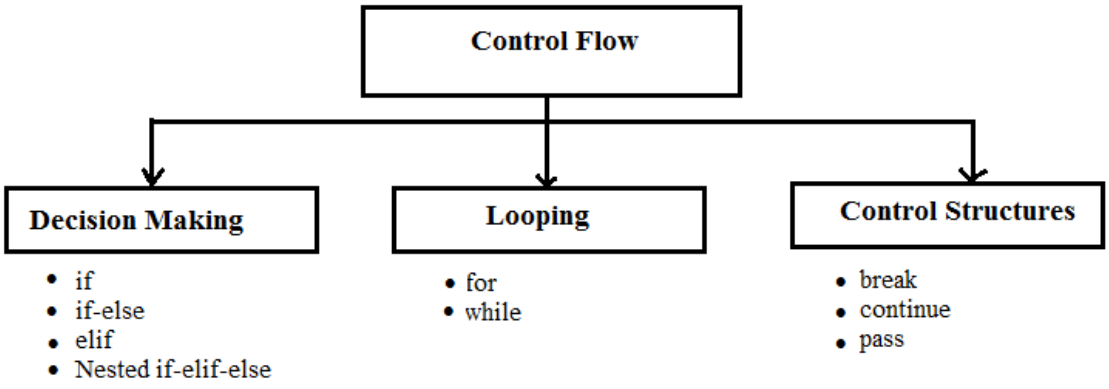
Conditionals: Boolean values and operators, conditional (if), alternative (if-else), chained conditional (if-elif-else); Iteration: state, while, for, break, continue, pass; Fruitful functions: return values, parameters, local and global scope, function composition, recursion; Strings: string slices, 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.

CONTROL FLOW

Control flow

- Control flow is the order in which individual statements are executed.
- ***Control*** means controlling
- ***Flow*** means way or sequence of program execution

Types of control flow



DECISION MAKING (Conditional or Selection or Branching)

- Decision structures evaluate multiple expressions, which produce TRUE or FALSE as the outcome
- Python programming language provides the following types of decision-making statements.
    1. If-statement
    2. If-else statement
    3. If-elif-else

4. Nested if-elif-else

Statement	Description
if statements	An if statement consists of a Boolean expression followed by one or more statements.
if...else statements	An if statement can be followed by an optional else statement, which executes when the boolean expression is FALSE.
nested if statements	You can use one if or else if statement inside another if or else if statement(s).

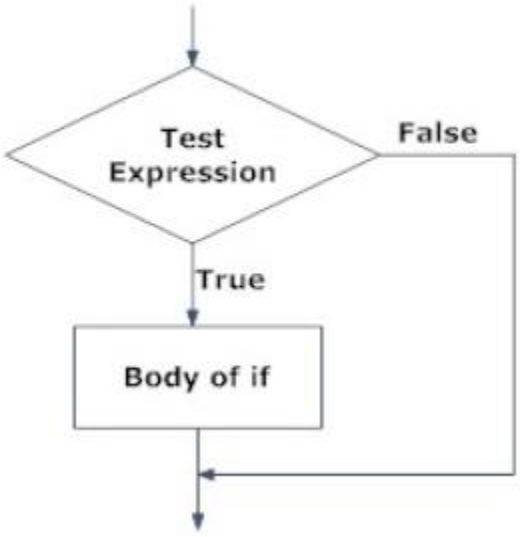
1. Conditional if Statement

- If condition (expression) is **TRUE**, then the block of statement(s) inside the 'if' statement is executed.
- If condition is **FALSE**, the statement is not executed.

Syntax

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

Flow chart



Example

```
a = 200
b = 100
if a > b:
    print("a is greater")
```

Output:

a is greater

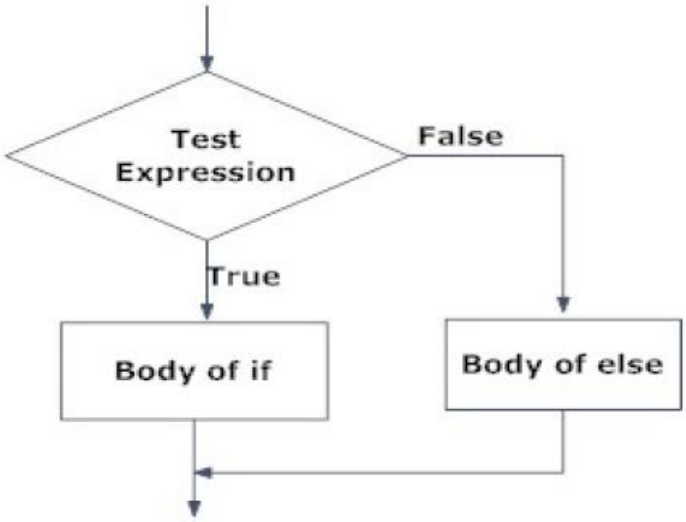
2. Alternative if-else Statement

- If condition is **TRUE**, then the block of statement(s) inside the 'if' statement is executed.
- If condition is **FALSE**, the statement(s) inside else block is executed.

Syntax

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

Flow chart



Example

```
a = 200
b = 100
if a > b:
    print("a is greater")
else:
    print("b is greater")
```

Output:

a is greater

3. Chained conditional if-elif-else Statement

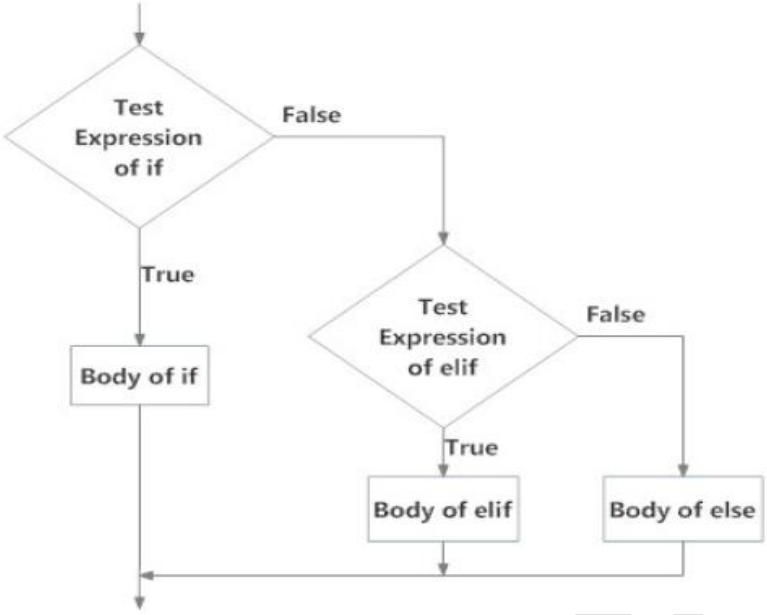
- An elif(else if) statement can be used to check ***multiple expressions***
- An if-elif-else structure first checks 'if condition'
- If condition is **TRUE**, execute the statements in the 'if ' block
- If condition is **FALSE**, it tests the condition in the elif block
- If elif statement is true, execute the statements in the elif block
- otherwise control passes to the else block

Syntax

```
if condition:

Statement(s)
elif condition:
```

Flow chart



Example

```
n = 50
if n > 50:
    print("Passed")
elif n == 50:
    print("Passed")
else:
    print("Failed")
```

Output:

Passed

4. Nested if-elif-else Statement

- Used to check another condition after the first condition has been evaluated as true.

Syntax

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

Example

```
n = 5
if  n > =0:
    if  n ==0:
        print("zero")
    else:
        print("Positive number")
else:
    print("Negative Number")
```

Output:

Positive number

LOOPING (Iteration)

- Looping executes sequence of statement again and again until specific condition satisfies.
- Python programming language provides the following types of loops

1. for loop
2. While loop

Loop Type	Description
while loop	Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.
for loop	Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.

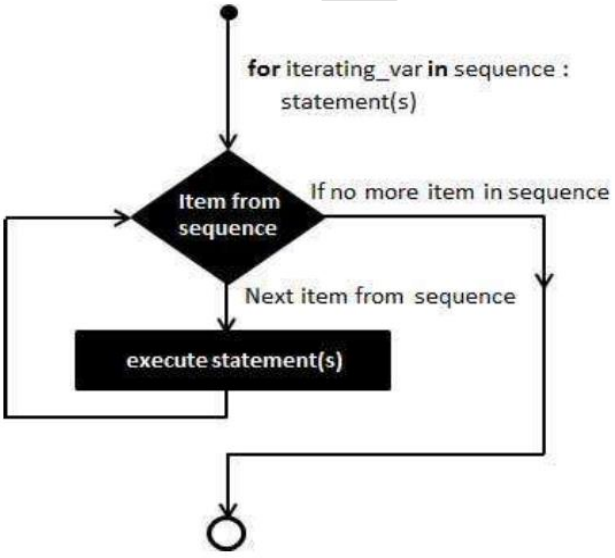
1. For Loops

- The 'for loop' repeats a given block of codes by specified number of times

Syntax

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

Flow chart



Example

```
fruits = ['Apple', 'Orange', 'Mango', 'Cherry']
for  i  in  fruits:
     print(i)
```

Output:

```
Apple
Orange
Mango
Cherry
```

range() function

- Generate a sequence of numbers using range() function.
- **range(10)** will generate numbers from 0 to 9 (10 numbers).
- **Syntax**

range(start, end, step)

- The function list() is used to force this function to output all the items

Example

```
>>>range(5)
Output:
    [0, 1, 2, 3, 4]
>>>range(3, 10)
>>>list(range(3,10))
Output:
    [3, 4, 5, 6, 7, 8, 9]
>>>range(4, 10, 2)
>>>list(range(4, 10, 2))
Output:
    [4, 6, 8]
```

for loop with else

- For loop can have an optional else block.
- The else part is executed if the items in the sequence used in for loop exhausts.

Example

```
numbers = [10, 99, 3]
for  i  in  numbers:
    if  i % 2 == 0:
        print(i, "is an even number")
    else:
        print(i, "is an odd numbers")
Output:
  10 is an even number
  99 is an odd number
   3 is an odd number
```

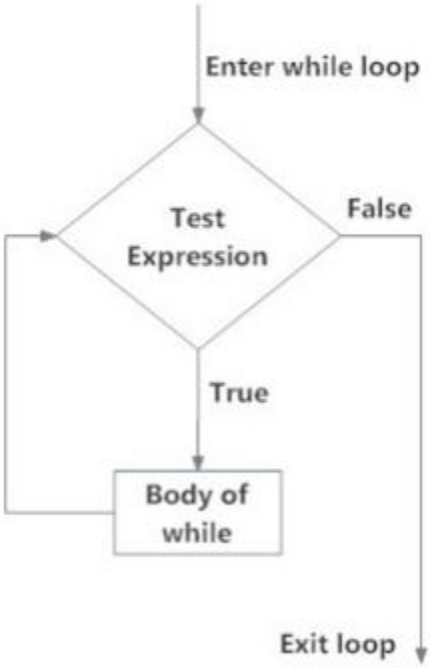
2. While Loop

- It is also called as ***Entry-Controlled Loop.***
- The while loop in Python is used to iterate over a block of code as long as the test expression (condition) is true.

Syntax

```
while  test_expression:
    Body of while
```

Flow chart



Example

```
Count = 1
While (count < =3):
    print("Python Programming")
    Count = count + 1
```

Output:

```
Python Programming
Python Programming
Python Programming
```

While loop with else

- While loop can have an optional else block.
- The else part is executed when the condition becomes false.

Example

```
Count = 1
While (count < =3):
    print("Python Programming")
    Count = count  + 1
else:
    print("Exit")
```

Output:

```
Python Programming
Python Programming
Python Programming
Exit
```

NESTED LOOPS

- Placing of one loop inside the body of another loop is called ***nested loop.***

Nested for loop
Syntax:
```
for  iterating_var in sequence:
   for  iterating_var in sequence:
      statements(s)
  statements(s)
```

Nested while loop
Syntax:
```
while  test_expression:
      while  test_expression:
          statements(s)
   statements(s)
```

CONTROL STATEMENTS *(8 marks)*

Control Statements

- It controls the flow of program execution to get desire result.
- Python supports the following three control statements, they are
    1. break
    2. continue
    3. pass

1. break statement

- It ***terminates*** the loop statement and transfers execution to the statement immediately following the loop.

Syntax:

```
break
```

Example
```
for  val  in "computer":
  if  val == "t":
     break
  print(val)
```
Output:
```
>>> c o  m  p  u
```

2. Continue statement

- It is used to ***skip*** the rest of the code inside a loop for the current iteration only.
- Loop does not terminate but continues on with the next iteration.

Syntax:

```
continue
```

Example
```
for  val  in "computer":
  if  val == "t":
     continue
  print(val)
```
Output:
```
>>> c o  m  p  u  e r
```

3. Pass statement

- It is a ***null*** statement.
- It does nothing when pass is executed
- It is used when a statement is required syntactically but do not want any command or code to execute.

Syntax:

```
pass
```

Example

```
for val in "computer":
  if val == "t":
     pass
  print(val)
```

Output:

```
>>> c o m p u e r
```

FRUITFUL FUNCTION *(16 or 8 marks)*

- **RETURN VALUES (***Refer UNIT-2 Notes, page No.30*)
- **PARAMETERS** (*Refer UNIT-2 Notes, page No.31-35*)
- **LOCAL AND GLOBAL SCOPE** (*Refer UNIT-2 Notes, page No.35-36*)
- **FUNCTION COMPOSITION** (*Refer UNIT-2 Notes, page No.36*)
- **RECURSION** (*Refer UNIT-2 Notes, page No.37-38*)

Fruitful function:

A function that returns a value is called fruitful function.

Example:

```
Root=sqrt(25)
```

Example:

```
def add():
a=10
b=20
c=a+b
return c
c=add()
print(c)
```

Void Function

A function that perform action but don't return any value.

Example:

```
print("Hello")
```

Example:

```
def add():
a=10
b=20
c=a+b
print(c)
add()
```

Return values:

return keywords are used to return the values from the function.

example:

return a – return 1 variable
return a,b– return 2 variables
return a,b,c– return 3 variables
return a+b– return expression
return 8– return value

PARAMETERS / ARGUMENTS:

- Parameters are the variables which used in the function definition. Parameters are inputs to functions. Parameter receives the input from the function call.
- It is possible to define more than one parameter in the function definition.

Types of parameters/Arguments:

1. Required/Positional parameters
2. Keyword parameters
3. Default parameters
4. Variable length parameters

Required/ Positional Parameter:

The number of parameter in the function definition should match exactly with number of arguments in the function call.

Example Output:

```
def student( name, roll ):
print(name,roll)
student(“George”,98)
```

George 98

Keyword parameter:

When we call a function with some values, these values get assigned to the parameter according to their position. When we call functions in keyword parameter, the order of the arguments can be changed.

Example Output:

```
def student(name,roll,mark):
print(name,roll,mark)
student(90,102,"bala")
```

Default parameter:

Python allows function parameter to have default values; if the function is called without the argument, the argument gets its default value in function definition.

Example Output:

```
def student( name, age=17):
print (name, age)
student( "kumar"):
student( "ajay"):
```

Kumar 17
Ajay 17

Variable length parameter

- Sometimes, we do not know in advance the number of arguments that will be passed into a function.
- Python allows us to handle this kind of situation through function calls with number of arguments.
- In the function definition we use an asterisk (*) before the parameter name to denote this is variable length of parameter.

Example

```
def student( name,*mark):
print(name,mark)
student ("bala",102,90)
```

Output: bala (102 ,90)

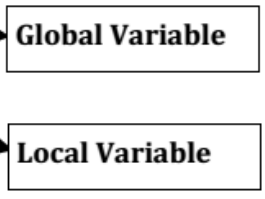
Local and Global Scope

Global Scope

- The *scope* of a variable refers to the places that you can see or access a variable.
- A variable with global scope can be used anywhere in the program.
- It can be created by defining a variable outside the function.

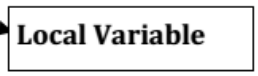
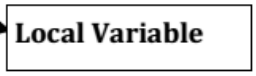
Example	output

a=50
def add():
  b=20
  c=a+b
  print©
def sub():
  b=30
  c=a-b
  print©
print(a)
	
	70
	
	20
	
	50



Local Scope A variable with local scope can be used only within the function .

Example	output

def add():
  b=20
  c=a+b
  print©
def sub():
  b=30
  c=a-b
  print©
print(a)
print(b)
	
	
	70
	
	20
	 
	
	error
	error

Function Composition:

❖ Function Composition is the ability to call one function from within another function

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

❖ In other words the output of one function is given as the input of another function is known as function composition.

Example:	**Output:**
math.sqrt(math.log(10))	

def add(a,b):
  c=a+b
  return c
def mul(c,d):
  e=c*d
  return e
c=add(10,20)
e=mul(c,30)
print(e)
	900

find sum and average using function composition	**output**

def sum(a,b):
  sum=a+b
  return sum
def avg(sum):
  avg=sum/2
  return avg
a=eval(input("enter a:"))
b=eval(input("enter b:"))
sum=sum(a,b)
avg=avg(sum)
	enter a:4
	enter b:8
	the avg is 6.0

print("the avg is",avg)	

Recursion

A function calling itself till it reaches the base value - stop point of function call.

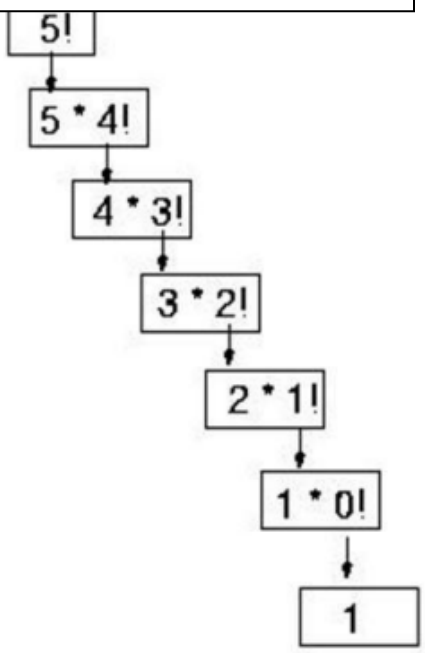
Example: factorial of a given number using recursion

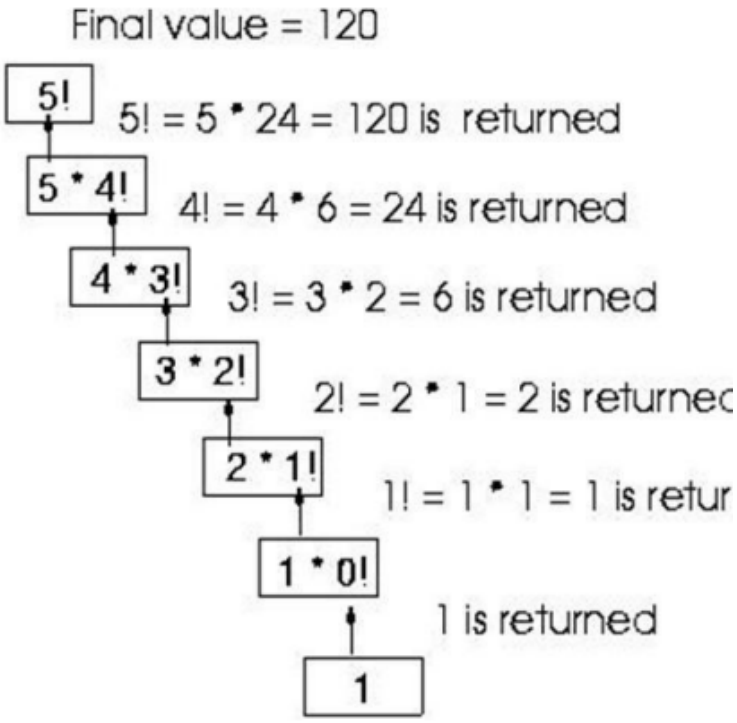
Factorial of n	Output

def fact(n):
  if(n==1):
  return 1
  else:
  return n*fact(n-1)
 	
n=eval(input("enter no. to find fact:"))
fact=fact(n)
print("Fact is",fact)
	enter no. to find fact:5
	Fact is 120

Explanation

STRINGS ***(16 or 8 marks)***





Examples:

1. sum of n numbers using recursion
2. exponential of a number using recursion

Sum of n numbers	**Output**

def sum(n):
  if(n==1):
  return 1
  else:
  return n*sum(n-1)
 	
n=eval(input("enter no. to find sum:"))
sum=sum(n)
print("Fact is",sum)
	enter no. to find sum:10
	Fact is 55

STRING

"String is the collection of characters, numbers, special characters or a combination of these types represented within the quotation marks"

- Strings can be enclosed within single(‘), double (“) and triple quotes (“ “ “)

Example:

‘C’- **string with single character**
‘@gmail.com’ - **string with special character**
“ “ - **Empty string**

INITIALIZING THE STRING VARIABLE

- Strings can be initialized by using an assignment statement(=)

Syntax

Variable_name = ‘initial_string

Example:
Name= “Ravi”
Year= “2017”

ACCESSING THE STRING VARIABLE

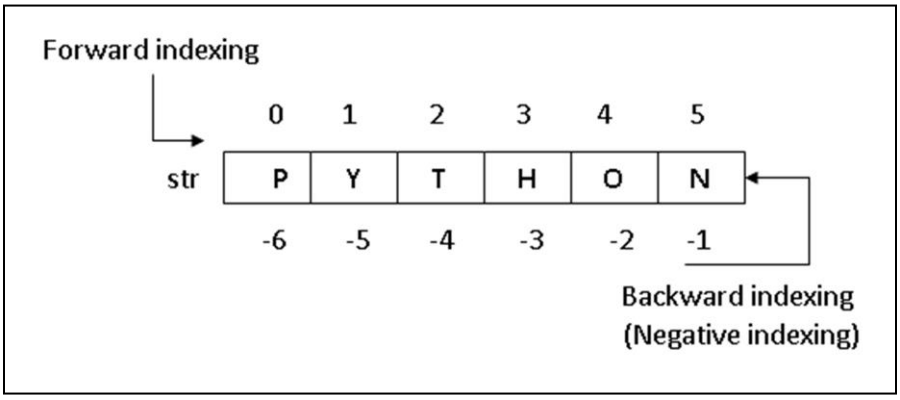
- String can be accessed by using the square brackets or index operator [].
- **Forward indexing:** Index starts from 0
- **Negative(Backward) indexing:** Index starts from -1,-2,-3 and so on, from left side.

Syntax

string_variable [index_number]

Example

```
>>>str=”PHYTHON”
>>>str[0]
        P       #output
>>>str[5]
        N       #output
>>>str[-2]
        O       #output
```



SLICING STRING

- A segment of a string is called a slice
- The slicing operator [:] is used to access a range of characters in a string
- The colon inside the square brackets is used to separate two indices from each other.
- **Syntax**

string_variable [start : end-1]

Example

```
>>>a= "Python Programming"
>>>a[0 : 5]
    'Pytho'          #output
>>>a[:5]
    'Pytho'          #output
>>> a[5:]
    'n Programming'  #output
>>> a[5:10]
     'n Pro'     #output
>>> a[:]
      'Python Programming'     #output
```

STRING ARE IMMUTABLE

- Strings are ***immutable***
- Which means that ***cannot change or modify*** any elements of a string.
- Solution to this problem is to generate a new string rather than change the old string.

Example

```
>>>a = 'hello python'
>>>a[0] = 'p'
```

Output:

TypeError: 'str' object does not support item assignment

STRING TRAVERSAL

- ***Traversal*** is a process in which can access all the elements of the string one by one using some conditional statements such as 'for' loop, 'while' loop

Traversal using 'for' loop

Example

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

Output:

P a r r o t

Traversal using 'while' loop

Example

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

Output:

P a r r o t

STRING OPERATION

- There are many operations that can be performed with string which makes it one of the most used datatypes in Python.
- The various string operations are,
    1. String concatenation
    2. Repeating a string (replication operator)
    3. In operator (string Membership test)
    4. String Comparison

1. String concatenation

- A number of strings can be combined to form a single string is called ***string concatenation.***
- The (+) operator is to join the strings.

Syntax

"string_variable 1" + "string_variable 2"+…..+ "string_variable n"

Example

```
>>>a = 'Hello'
>>>b = 'World'
>>>c = a + b
>>>print(c)
```

Output

Hello world

2. Repeating a string(Replication Operator)

- Group of strings can be repeated by using the (*) operator.

- The (*) operator is the string replication operator.

Syntax

(“string_variable” * n)

Where, **string_variable** is the variable name
***** is the string replication operator
n is the number of times the string to be repeated.

Example
print(“Python” * 3)
Output
PythonPythonPython

3. **The ‘ in’ operator**
   - Can test if a sub string exists within a string or not, using the keyword ‘**in’**.

Example
>>> 'a' in 'program'
True
>>> 'at' not in 'battle'
False

4. **String Comparison**
   - The relational operators are used to compare two strings.

Example
>>> if ‘python’ == ‘python’
Print(“Both strings are equal”)
Output
Both strings are equal

ESCAPE CHARACTERS (SEQUENCE)

- The backslash character (\) is used to escape characters.
- It converts difficult –to-type characters into a string.

Example
>>> print "I am 6'2\" tall."
Output
I am 6'2\" tall

List of Escape characters

Escape character	Description of Action
\a	Alert sound. A beep is generated by the computer on execution
\b	Backspace.
\f	Form feed.
\n	New line. Shifts the cursor to new line. Words on the right of "\n" go to next line
\r	Carriage return. Positions the cursor to the beginning of current line.
\t	Horizontal tab, it moves the cursor by a number of spaces or to next tab stop
\v	Vertical tab.
\\	Backslash. Displays a black slash character (\).
\'	Displays a single-quote character(').
\"	Displays a double-quote character(").
\?	Displays question mark (?).
\0	Null character. Marks the end of string.

PYTHON STRING FORMATTING

- The **format() method** is an extremely convenient way to format text.
- Other supported symbols and functionality are listed in the following table

Option	Meaning
'<'	The field will be left-aligned within the available space
'>'	The field will be right-aligned within the available space
'0'	If the width field is preceded by a zero ('0') character, sign-aware zero-padding for numeric types will be enabled.
','	This option signals the use of a comma for a thousand separators
'='	Forces the padding to be placed after the sign but before the digits.
'^'	Forces the field to be centered within the available space

'+'	Indicates that the sign should be used for both positive as well as negative numbers.
'-'	Indicates that the sign should be only for negative numbers.
space	Indicates that a leading space should be used on positive numbers and a minus sign on negative numbers.

Syntax

```
format (value, format_specifier)
```

Where, **value** is the string to be displayed

format_specifier is the combination of formatting options.

Example

```
>>> print(format('Python Programming','<20'))  # Left Justified
>>> print(format('Python Programming','<60'))  # Right Justified
>>> print(format('Python Programming','^50'))  # Center
```

Output

```
Python Programming
                                                Python Programming
                                    Python Programming
```

STRING FUNCTIONS & METHODS *(16 or 8 marks)*

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

String function in python

s.capitalize()	Capitalizes first character of s
s.capwords( )	Capitalizes first letter of each word in s
s.count(sub)	Count number of occurrences of sub in s
s.find(sub)	Find first index of sub in s
s.index(sub)	Find first index of sub in s
s.rfind(sub)	Same as find, but last index
s.rindex(sub)	Same as index, but last index
s.lower( )	Convert s to lowercase
s.split( )	Return a list of words in s
s.joint(1st)	Join a list of words into a single string with s as separator
s.strip( )	Strip trailing white space from s
s.upper( )	Convert s to upper string

s.replace(old,new)	Replace all instances of old with new in string

String Length()

- The length function returns the number of characters in a string

```
Example
>>>a = 'Hello'
>>> len(a)
 5
```

String upper()

- The upper function converts the letters in the string into uppercase letters.

```
Example
>>>a = 'python programming'
>>> a.upper()
Output
PYTHON PROGRAMMING
```

String lower()

- The lower function converts the letters in the string into lowercase letters.

```
Example
>>>a = 'PYTHON PROGRAMMING'
>>> a.lower()
Output
python programming
```

Capitalize()

- The 'capitalize' function capitalizes the first character of the string.

```
Example
>>>a = 'python'
>>> a.captialize()
Output
Python
```

Split()

- The 'split' function trailing white space from a string.

Example
>>>a = 'Python Programming'
>>> a.split()
Output
['python','programming']

Find()

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

Example
>>>a = 'python programming'
>>> a.find(pro)
Output
7

Count()

- The 'count' method returns the total number of overlapping of given substrings.

Example
>>>a = 'python programming python programs'
>>> a.count(python)
Output
2

Isalnum()

- This method returns the Boolean value ***true*** if there is one character atleast and if all the characters in the given string are **alphanumeric.**
- It return false otherwise

Example
>>>a = 'apple grapes orange3'
>>>b = 'orange3'
>>>print(a.isalnum())
>>>print(b.isalnum())
Output
False
True

Isalpha()

- This method returns the Boolean value ***true*** if there is one character atleast and if all the characters in the given string are **alphabetic.**
- It return false otherwise

```
Example
>>>a = 'orange3'
>>>b = 'orange'
>>>print(a.isalpha())
>>>print(b.isalpha())
Output
False
True
```

Isdigit()

- This method returns the Boolean value ***true*** if there is one character atleast and if all the characters in the given string are **numeric.**
- It return false otherwise

```
Example
>>>a = 'orange3'
>>>b = '123456'
>>>print(a.isdigit())
>>>print(b.isdigit())
Output
False
True
```

Islower()

- This method returns the Boolean value ***true*** if all the cased characters present in the string are **lowercase.**
- It return false otherwise

```
Example
>>>a = 'orange'
>>>b = 'APPLE'
>>>print(a.islower())
>>>print(b.islower())
Output
True
False
```

Isupper()

- This method returns the Boolean value ***true*** if all the characters present in the string are **uppercase.**
- It return false otherwise

Example

```
>>>a = ‘orange’
>>>b = ‘APPLE’
>>>print(a.isupper())
>>>print(b.isupper())
```

Output

```
False
True
```

STRING MODULES *(8 & 2marks)*

- The string module provides additional tools to manipulate strings.

Examples

```
import string
string . brackets = “ [ ] { } ( ) < > “
print(string .brackets)
print(string .digits)
print(string .hexdigits)
print(string .octdigits)
print(string .punctuation)
print(string .whitespace)
```

Output

```
[ ] { } ( ) < >

0123456789

0123456789abcdefABCDEF

01234567

!”#$%’()*+,-./:;<=>?@[\]^_’{|}~

‘\011\012\013\014\015’
```

LISTS AS ARRAYS (*2 marks)*

- Arrays and lists are both used in Python to store data, but they don't serve exactly the same purposes.
- ***Array*** is the collection of same data types.
- ***List*** is the collection of element of any data types enclosed in [ ] bracket
- They both can be used to store any data type (real numbers, strings, etc)
- They both can be indexed and iterated.
- The main difference between a list and an array is the functions that perform to them.

Array

Example:

```
>>>x = array([3, 6, 9, 12])
>>>x/3.0
>>>print(x)
```

Output:

array([1, 2, 3, 4])

List

Example:

```
>>>y = [3, 6, 9, 12]
>>>y/3.0
>>>print(y)
```

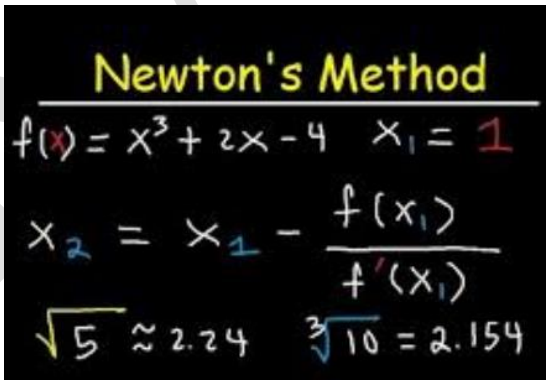
Output:

Error

IIUSTRATIVE PROGRAMS *(16 or 8 marks)*

1. Square root of a number

Description



Program

```
A=int(input("Enter a Number:"))
x=a
for i in range(3):
```

Output:

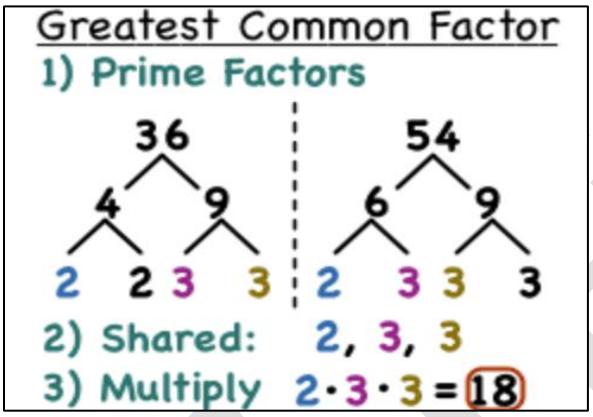
Enter a Number: 5

The square root of a Number: 2.24

```
    x- =(x*x-a)/(2.0 * x)
print("The square root of a Number:", x)
```

2. Greatest Common Divisor (GCD)

Description



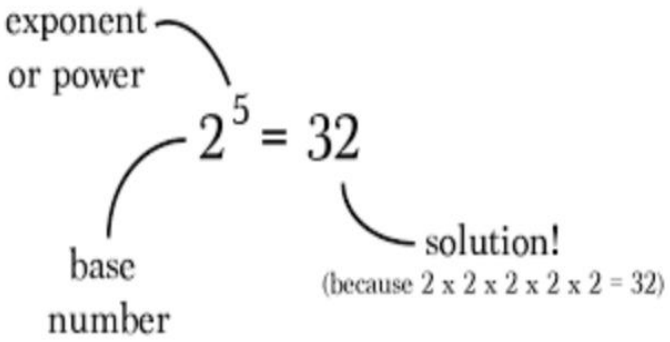
Program

```
n1=int(input("Enter a number:"))
n2=int(input("Enter another number:"))
rem = n1 % n2
while rem!=0:
    n1=n2
    n2=rem
    rem= n1 % n2
print("GCD of given numbers is:",n2)
```

Output:
Enter a Number: 36
Enter another number: 54
GCD of given numbers is: 18

3. Exponentiation of a number

Description



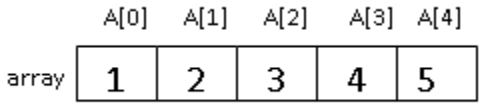
Program

```
n=int(input("Enter a number:"))
e=int(input("Enter exponent:"))
r= n
for i in range(1,e):
  r= n * r
print("Exponentiation is:", r)
```

Output:
Enter a Number: 2
Enter exponent: 5
Exponentiation is: 32

4. Sum an array of numbers

Description



Sum=A[0] + A[1] + A[2] + A[3] + A [4]

= 1 + 2 + 3 + 4 + 5

= 15

Program

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

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

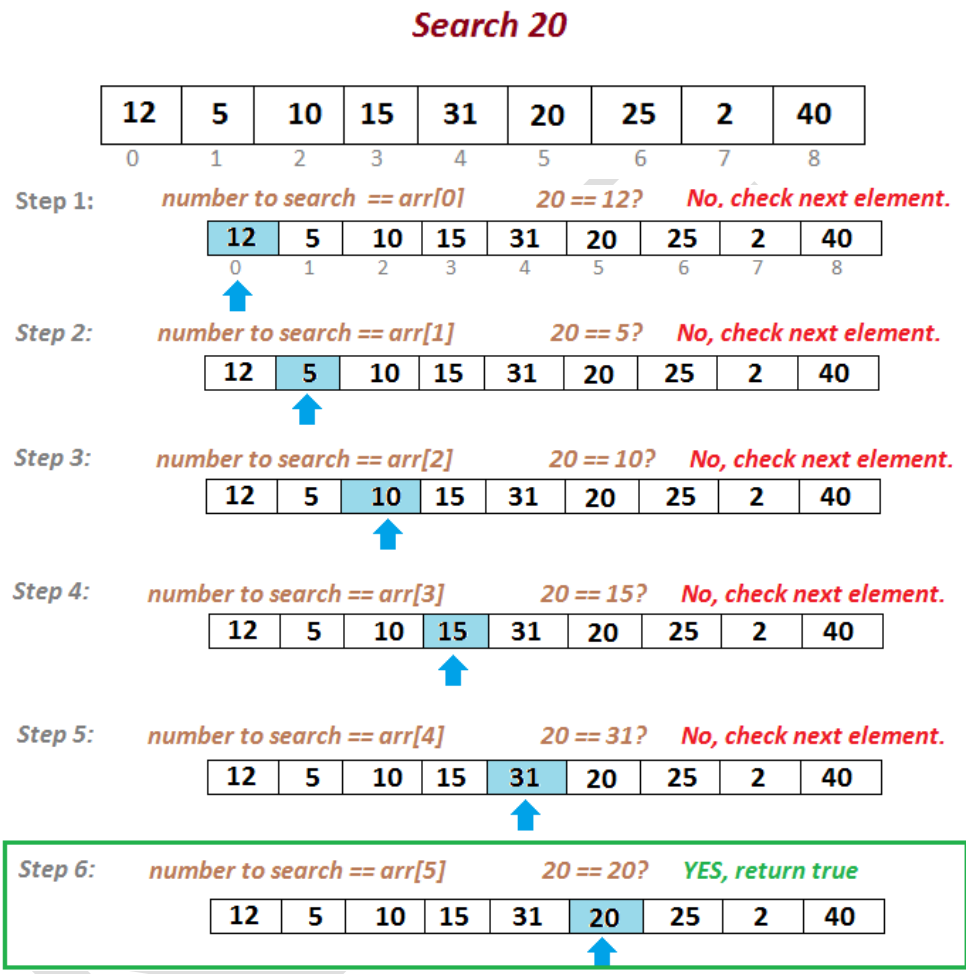
5. Linear Search

Description

- Linear search, also called as **sequential search**
- Very simple method used for searching an array for a particular value.

- It works by comparing the value to be searched with every element of the array one by one in a sequence until a match is found.
- Linear search is mostly used to search an unordered list of elements (array in which data elements are not sorted).

Example



Program

```
A=[0,0,0,0,0,0,0,0,0,0]
found=0
n=int(input("enter a number:"))
print('enter n numbers')
for i in range(0,n):
    A[i]=int(input())
key=int(input("enter a key to be searched:"))
for i in range(0,n):
    if key==A[i]:
        print('key found')
```

Output:

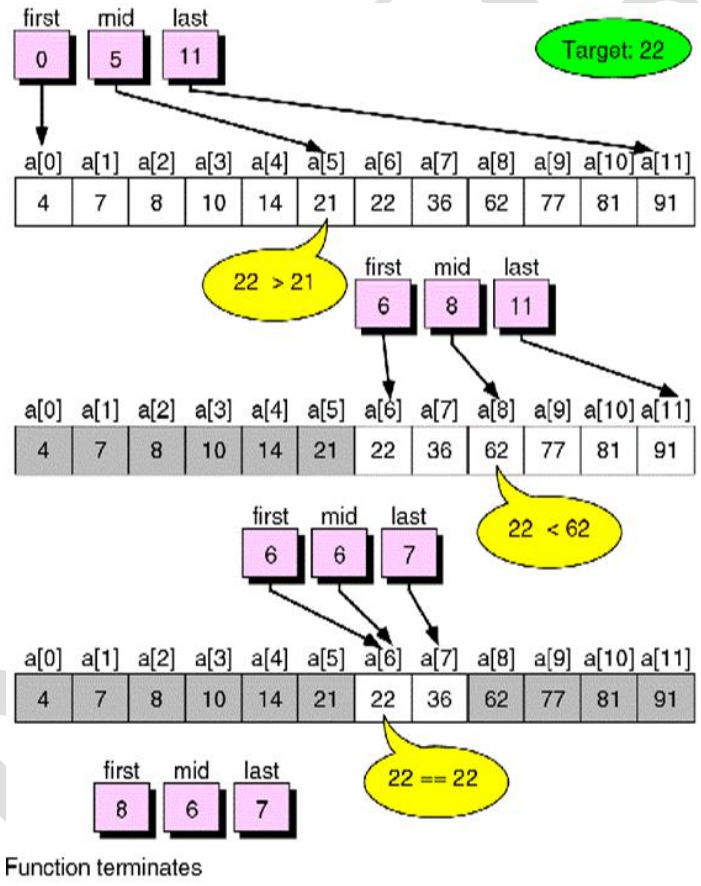
```
Enter a number: 8
Enter n numbers
8 4 7 5 6 3 9 2
Enter a key to be searched: 9
Key found
>>>
Enter a number: 8
Enter n numbers
8 4 7 5 6 3 9 2
Enter a key to be searched: 11
Key not found
```

```
            found+=1
        else:
            continue
    if found==0:
    print('key not found')
```

6. Binary Search

Description

- Binary search is a searching algorithm that works efficiently with a sorted list
- It divides the list into two halves, by taking the middle element



Program

```
def binarySearch(alist,item):
    first=0
    last=len(alist)-1
    found=False
    while first<=last and not found:
```

Output:

```
>>>
False
True
```

```
        midpoint=(first + last)//2
        if alist[midpoint] == item:
            found = True
        else:
            first = midpoint + 1
        return found
testlist= [0,1,2,8,13,17,19,32,42]
print(binarySearch(testlist,3))
print(binarySearch(testlist,13))
```

Possible Questions

Part-B (2 Marks)

1. Define flow control (decision making) ***Pg no:1***
2. List the types of python's decision making statements ***Pg no:1***
3. List the types of conditional(selection or branching) statement** ***Pg no:2***
4. Write syntax for if (conditional)statement in python ** ***Pg no:2***
5. Draw the flowchart for if statement ***Pg no:2***
6. Write syntax for if-else(alternative) statement in python ** ***Pg no:3***
7. Write syntax for if-elif-else(chained conditional) statement in python ***Pg no:3***
8. Name the types of Boolean operators ***Ans: True,False***
9. Define loops in python ** ***Pg no:4***
10. Define the types of looping constructs ** ***Pg no:4***
11. Define 'for' loop with else case ** ***Pg no:6***
12. Write syntax for 'for' loop in python ** ***Pg no:5***
13. Write syntax for 'while' loop in python ** ***Pg no:6***
14. Define nested loops with example. ***Pg no:7***
15. What is the purpose of break statement in python** ***Pg no:8***
16. What is the purpose of continue statement in python** ***Pg no:8***
17. Difference between continue and pass statement. ***Pg no:8-9***
18. Define pass statement in python** ***Pg no:9***
19. Define string with example ** ***Pg no:9***
20. Write syntax to initialize string variable in python with example** ***Pg no:10***
21. How to access string variable in python **(or)** How to access characters of string ***Pg no:10***
22. What is the output of print str if str='Hello World!' ***Ans: Hello World!***
23. What is the output of print str[0] if str='Hello World!' ***Ans: H***

24. Define string indexing in python ***Pg no:10***
25. What is the use of slicing operators** ***Pg no:10***
26. What is meant by string immutability** ***Pg no:11***
27. Define string concatenation** ***Pg no:12***
28. What is replication operator ***Pg no:12***
29. Define escape sequence. List any three. ***Pg no:13-14***
30. What is meant by string formatting? ***Pg no:14***
31. List any four string build-in function** ***Pg no:15***
32. Define string module with example ***Pg no:19***
33. List any four string module ***Pg no:19***
34. Define array with example ***Pg no:20***
35. Write program to check the number is positive or negative or zero ***Pg no:153(book)***
36. Write program to find sum of first N natural numbers using while loop ***Pg no:153(book)***
37. Write program to find factorial of a number ***Pg no:173(book)***
38. Write program to count the number of vowels ***Pg no:180(book)***
39. Write program to remove duplicates from a list ***Pg no:279(book)***
40. Write program that accept a word from the user and reverse it ***Pg no:178(book)***

Part-C (6 marka\s)

1. Explain **build-in string(pre-defined) function** with example in detail

   (or)

   Explain string **methods** with example in detail ******* ***(16m)***
2. Define methods in a string with an example program using atleast five methods.
3. Explain in detail **conditional (selection) statement** with example **** ***(16m)***
4. Explain in detail **iterative (looping) statement** with example ******* ***(16m)***
5. Explain the types of looping construct in detail
6. explain break and continue statement with the help of for loop in an example
7. Explain various **string modules** with example in detail ******* ***(16m)***
8. Write python program to find **Binary search** with example***** ***(8m)***
9. Write python program to find **linear search** with example***** ***(8m)***
10. Write python program to find **GCD** of two numbers with example***** ***(8m)***
11. Write python program to find **square root of a number using newton's method** with example***** ***(8m)***
12. Write python program to find **sum an array of numbers** with example** ***(8m)***

IMPORTANT PROGRAMS

Program to find

1. Fibonacci series ***(Refer book Pg.no:174)***
2. odd or even ***(Refer book Pg.no:153)***
3. Sum of 'n' natural numbers ***(Refer book Pg.no:163)***
4. Count no.of vowels in a string ***(Refer book Pg.no:180)***
5. Largest of three number ***(Refer book Pg.no:156)***
6. Leap year or not ***(Refer book Pg.no:154)***
7. Factorial of a number***(Refer book Pg.no:173)***
8. Reverse a string ***(Refer book Pg.no:178)***
9. Simple interest ***(Refer book Pg.no:94)***
10. Roots of Quadratic equation ***(Refer book Pg.no:155)***
11. Prime or not ***(Refer book Pg.no:171)***
12. Palindrome number or not ***(Refer book Pg.no:167)***

Questions	Option1	Option2	Option3	Option4	Answers
Which predefined Python function is used to find length of string?	length()	len()	strlen()	stringlength()	len()
Syntax of constructor in Python?	def __init__()	def _init_()	_init_()	All of these	def __init__()
How to find the last element of list in Python? Assume `bikes` is the name	bikes[0]	bikes[-1]	bikes[lpos]	bikes[:-1]	bikes[-1]
If a='cpp', b='buzz' then which of the following operation would show	a+b	a+"+b	a+""+b	All the options	All the options
If a='cpp', b='buzz' then what is the output of: c = a-b print(c)	cpp-buzz	cppbuzz	TypeError: unsupporte	None	TypeError:
a = 8.6 b = 2 print a//b	4.3	4.0	4	compilation error	4.0
The format function, when applied on a string returns :	list	bool	int	str	str
print(chr(ord('b')+1))	b	syntax error	c	b+1	c
Find out output of following code snippet >>>str="cppbuzz"	uzz	cpp	buzz	cppb	cpp
What is the data type of X in X = [12.12, 13, 'cppbuzz']	Tuple	Array	List	Dictionary	List
What is the output of the expression? round(4.5676,2)?	4.5	4.6	4.57	4.56	4.57
What is the output of the function shown below?	f	oxF	oXf	oxf	oxf
What is the output of the following piece of code?	1	4	A	Invalid syntax for	A
What is the output of the code shown below? t=32.00	[0]	0	[0.00]	Error	Error
What is the output of the following? print([i.lower() for i in "HELLO"])	[‘h’, ‘e’, ‘l’, ‘l’, ‘o’]	‘hello’	[‘hello’]	hello	[‘h’, ‘e’, ‘l’, ‘l’,
The format function, when applied on a string returns :	list	bool	int	str	str
print(chr(ord('b')+1))	b	syntax error	c	b+1	c
Which of the following operators is used to get accurate result (i.e	//	%	/	\	//
What is the associativity of Operators with the same precedence?	Depends on operators	Left to Right	Right to Left	Depends on Python	Left to Right
Which of the following has more precedance?	+	/	-	()	()

Which of the following variable is invalid?	_str	_ _str	_str_	None	None
Select the command to Find and print Data types using the Type command.	type(name)	print(type(name))	print(name)	type()	print(type(name))
what is the output for: name="Hello World" print(type(name))	Hello World	hello World	<class 'str'>	str	<class 'str'>
What data type is the object below ? L = [1, 23, 'hello', 1]	List	Dictionary	Tuple	Array	List
Which of the following function convert a string to a float in python?	int(x [,base])	long(x [,base])	float(x)	str(x)	float(x)
What is called when a function is defined inside a class?	Module	Class	Another Function	Method	Method
What will be the output of the following code : print type(type(int))	type 'int'	type 'type'	Error	0	type 'type'
Which of the following is the use of id() function in python?	Id returns the identity	Every object	returns the object	All of the mentioned	Id returns the
time.time() returns _____	the current time	the current time in	the current time in	the current time in	the current time in
Suppose list1 is [3, 4, 5, 20, 5, 25, 1, 3], what is list1 after list1.pop(1)?	[3, 4, 5, 20, 5, 25, 1, 3]	[1, 3, 3, 4, 5, 5, 20,	[3, 5, 20, 5, 25, 1, 3]	[1, 3, 4, 5, 20, 5, 25]	[3, 5, 20, 5, 25, 1, 3]
Which of the following keyword is a valid placeholder for body of the	break	continue	body	pass	pass
Let a = [1,2,3,4,5] then which of the following is correct ?	print(a[:]) => [1,2,3,4]	print(a[0:]) =>	print(a[:100]) =>	print(a[-1:]) =>	print(a[:100]) =>
What is the need of if __name__ == "__main__": somemethod()	Create new module	Define generators	Run python	Create new objects	Run python
In python which is the correct method to load a module ?	include math	import math	#include< math.h>	using math	import math
Which of The Following Statements Is True?	By default, the	The __init__()	The __str__()	The __new__()	All options
Which of The Following Statements Can Be Used To Check, Whether An	obj.isinstance(A)	A.isinstance(obj)	isinstance(obj, A)	isinstance(A, obj)	isinstance(obj, A)
Which of the following data types is not supported in python ?	String	Numbers	Slice	List	Slice
Which of the following commands will create a list?	list1 = list()	list1 = []	list1 = list([1, 2,	all of the mentioned	all of the mentioned
What is the output when we execute list("hello")?	['h', 'e', 'l', 'l', 'o']	['hello']	['llo']	['olleh']	['h', 'e', 'l', 'l',
Suppose listExample is ['h','e','l','l','o'], what is	5	4	3	6	5
Suppose list1 is [2, 33, 222, 14, 25], What is list1[:-1] ?	[2, 33, 222, 14]	Error	25	[25, 14, 222, 33, 2]	[2, 33, 222, 14]

Which of the following is a Python tuple?	[1, 2, 3]	(1, 2, 3)	{1, 2, 3}	{}	(1, 2, 3)
Suppose t = (1, 2, 4, 3), which of the following is incorrect?	print(t[3])	t[3] = 45	print(max(t))	print(len(t))	t[3] = 45
How many elements are in m? m = [[x, y] for x in range(0, 4) for y in	8	12	16	32	16
What is the data type of (1)?	Tuple	Integer	List	Both tuple and integer	Integer
If a=(1,2,3,4), a[1:-1] is	Error, tuple slicing	[2,3]	(2,3,4)	(2,3)	(2,3)
What is the type of each element in sys.argv?	set	list1 = []	tuple	string	string
What is the length of sys.argv?	number of arguments	number of arguments + 1	number of arguments – 1	number of arguments +2	number of arguments + 1
How many keyword arguments can be passed to a function in a single	zero	one	zero or more	one or more	zero or more
Suppose list1 = [0.5 * x for x in range(0, 4)], list1 is :	[0, 1, 2, 3]	[0, 1, 2, 3, 4]	[0.0, 0.5, 1.0, 1.5]	[0.0, 0.5, 1.0, 1.5,	[0.0, 0.5, 1.0, 1.5]
The **if...elif...else** executes only one block of code among several blocks.	TRUE	FALSE	It depends on expression used		There is no elif statement in python
In Python, for and while loop can have optional else statement?	only for loop can have optional else statements	only while loop can have optional else statements	Both for and while can have optional else statements		Loops cannot have optional else statement in python

SYLLABUS

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: selection sort, insertion sort, merge sort, histogram.

LISTS

LIST

"A list is an ordered sequence of values of any data types (string, float, integer, etc.)"

- Values in the list are called elements or items.
- The elements in the list are mutable and indexed / ordered.
- The elements in the list are enclosed in **square bracket [ ]**separated by comma**.**

CREATING A LIST

Syntax

```
List_name = [item1, item2, item3, item4……., item n]
```

Example

***# Empty list*:** A list that contains no elements is called an empty list

```
my_list = []
```

 # list of integers

```
my_list = [1, 2, 3]
```

 # list with mixed datatypes

```
my_list = [1, "Hello", 3.4]
```

Nested List

- A list is an element of another list is called ***nested list.***

Example

[“Dell, 2.0 ,[50, 100]]

ACCESSING LIST ELEMENTS

- List elements can be accessed by using the square brackets or index operator [].
- **Syntax**

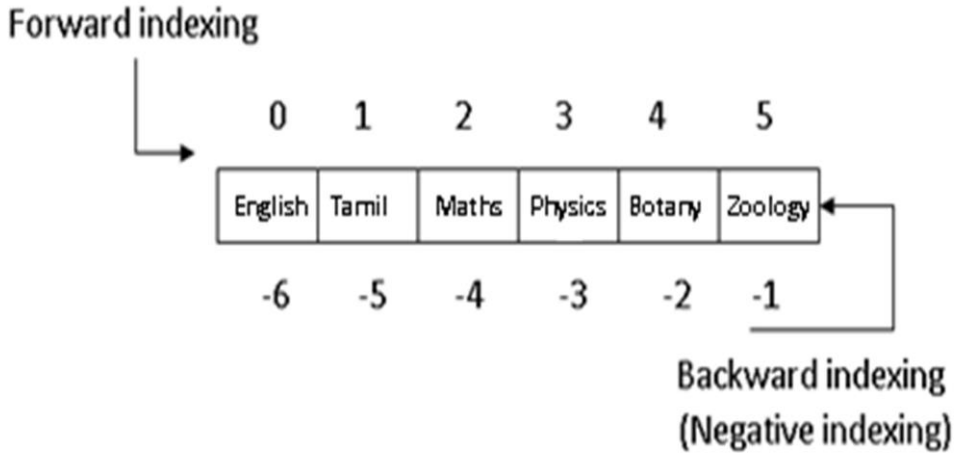
List_variable [index_number]

Indexing

1. **Forward indexing:** Index starts from 0 to n-1.

Example

```
>>>subject= [“English”, “Tamil”, “Maths”, “Physics”, “Botany”, “Zoology”]
>>>subject [0]
‘English’#output
>>>subject [2]
     ‘Maths’      #output
>>>subject [5]
     ‘Zoology’    #output
```



2. **Negative (backward) indexing:** The index of -1 refers to the last item, -2 to the second last item and so on.

Example

```
>>>subject= [“English”, “Tamil”, “Maths”, “Physics”, “Botany”, “Zoology”]
>>>subject [-1]
‘Zoology’#output
>>>subject [-3]
‘Physics’ #output
>>>subject [-6]
‘English’ #output
```

SLICING LISTS

- A segment of a list is called a slice
- The slicing operator [:] is used to access a range of elements in a list.
- The colon inside the square brackets is used to separate two indices from each other.
- **Syntax**

```
List_variable [start : end-1]
```

Example

```
>>>a= [‘parrot’,’Dove’,’duck’,’cuckoo’]
>>>a[2 :4]
        [‘duck’, ’cuckoo’]#output
>>>a[:3]
        [‘parrot’, ’Dove’, ’duck’] #output
>>>a[2:]
        [‘duck’, ’cuckoo’]#output
>>>a[:]
        [‘parrot’, ’Dove’, ’duck’,’cuckoo’] #output
```

LISTS ARE MUTABLE

- Lists are ***mutable***
- Which means ***can change or modify*** any elements of a list.

Example

```
>>>a = [‘parrot’, ’Dove’, ’duck’, ’cuckoo’]
>>>a [3] = “crow”
```

Output:

```
[‘parrot’, ’Dove’, ’duck’, ’crow’, ’cuckoo’]
```

ALIASING

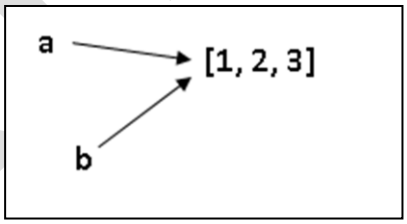
- Aliasing is a circumstance where two or more variables refer to the same object.

Example

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

Output:

```
True
```



- Here, the same list has two different names, ‘a’ and ‘b’, so it is aliased.
- Changes made with one alias affect the other.

CLONING LISTS

- Cloning list is to make a copy of the list itself.
- The easiest way to clone a list is to use the slice operator

Example

```
>>>a = [1, 2, 3]
>>>b = a [:]
>>>print(b)
        [1, 2, 3]
>>>print(a)
        [1, 2, 3]
```

LIST LOOP (TRAVERSING A LIST)

- The most common way to traverse the elements of a list is using a 'for loop'.

Traversal using 'for' loop

Example

```
>>>bird = ['p', 'a', 'r',' r', 'o', 't']
>>>for letter in bird:
        print(letter)
```

Output:

P a r r o t

LIST PARAMETERS

- List can be passed as an argument in function
- If function modifies the list, the caller can see the changes

Example

```
defdelete_head(t):          #function definition
del t[0]
>>>list=['a', 'b' ,'c']
>>>delete_head(list)        #function call
>>>list
['b', 'c']
```

LIST OPERATION*(8 marks)*

- There are many operations that can be performed with list.
- The various list operations are,

1. List concatenation
2. Repeating a list (replication operator)
3. 'in' operator (List Membership test)

1. List concatenation

- A number of lists can be combined to form a single list is called ***list concatenation.***
- The (+) operator is to join the list.

Syntax

"list_variable 1" + "list_variable 2"+…..+"list_variable n"

Example

```
>>>a = ['x', 'y', 'z']
>>>b = [5, 10, 15]
>>>c = a + b
>>>print(c)
```

Output

['x', 'y', 'z', 5, 10, 15]

2. Repeating a list(Replication Operator)

- Group of list can be repeated by using the (*) operator.
- The (*) operator is the list replication operator.

Syntax

("list_variable" * n)

Where, **list_variable** is the variable name

***** is the list replication operator

n is the number of times the list to be repeated.

Example

```
>>>a = [ 1, 2, 3 ]
>>>print (a* 3)
```

Output

[1, 2, 3, 1, 2, 3, 1, 2, 3]

3. The 'in'operator (List Membership test)

- Can test if an element is exists within a list or not, using the keyword '**in'**.

Example

```
>>>a = [ 1, 2, 3, 4, 5 ]
>>>5 in a
True
>>>10 in a
False
```

BUILD-IN LIST METHODS*(16 or 8 marks)*

- A list provides methods to perform a variety of useful operations.

List Method in python

append()	Add an element to the end of the list
extend( )	Add all elements of a list to the another list
insert( )	Insert an item at the defined index
remove( )	Removes an item from the list
pop( )	Removes and returns an element at the given index
clear( )	Removes all item in the list
index( )	Returns the index of the first matched item.
count( )	Returns the count number of items passed as an argument
sort()	Sort items in a list in ascending order
reverse( )	Reverse the order of items in the list
copy( )	Returns a shallow copy of the list

Adding elements to the end of the list

Append( )

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

Syntax

list_name.append(object)

Example

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

Extend( )

- The extend method takes a list as an argument and appends all of the elements.

Syntax

list_name.extend(seq)

Example

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

Inserting items to a list

insert ( )

- The insert method can be used to insert an item on a desired position.

Syntax

list_name.insert (index, object)

Example

```
>>>t = [‘a’, ‘b’, ‘c’ ]
>>>t.insert(2,‘d’)
>>>t
[‘a’, ‘b’, ’d’, ‘c’ ]
```

Changing items of a list

- The assignment (=) and indexing [ ] operators are used to change an item or range of items on the list.

Example

```
>>>t = [1, 2, 3, 4]
>>>t[0] = 5
>>>t [5, 2, 3, 4]
```

Sort( )

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

Syntax

list_name.sort( )

Example

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

reverse ( )

- The reverse method reverses the elements of the list, in place

Syntax

list_name.reverse( )

Example

```
>>>t = [9, 8, 7, 6, 5]
>>>t.reverse( )
>>>t [5, 6, 7, 8, 9]
```

Count (x)

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

Syntax

list_name.count(object)

Example

```
>>>t = [‘a’, ‘p’, ‘p’, ‘l’, ‘e’]
>>>print(a.count(‘p’)
    2
```

Removing or deleting items from a list

remove( )

- The remove method removes the ***specified item*** from the list.

Syntax

list_name.remove(object)

Example

```
>>>t = [‘a’, ‘p’, ‘p’, ‘l’, ‘e’]
>>>t.remove(‘l’)
>>>t
[‘a’, ‘p’, ‘p’, ‘e’]
```

pop( )

- The pop method removes the item ***associated with the given index***.

Syntax

list_name.pop(index)

Example

```
>>>t = ['a', 'p', 'p', 'l', 'e']
>>>t.pop(2)
>>>t
['a', 'p', 'l', 'e']
```

clear( )

- The clear method is used to ***empty a list***

Syntax

list_name.clear()

Example

```
>>>t = ['a', 'p', 'p', 'l', 'e']
>>>t.clear( )
>>>t
   [ ]
```

Deleting items using the keyword 'del'

- The keyword ***del***can be used to delete one or more items on a list or the entire list itself.

Syntax

del list_name[ ]

Example

```
>>>t = ['a', 'p', 'p', 'l', 'e']
>>>del t[0]
>>>t
['p', 'p', 'l', 'e']
```

BUILD-IN FUNCTIONS WITH LIST*(16 or 8 marks)*

- Python's built-in functions can be used with list to obtain needed value and execute various tasks.

len()

- The length function returns the number of items on a list.

Example

```
>>>a = [0, 5, 10, 15, 20, 25]
>>>len(a)
6
```

Max()

- The Max( ) function returns item with ***maximum*** value from the list

Example
>>>a = [0, 5, 10, 15, 20, 25]
>>>max(a)
25

Min( )

- The Min( ) function returns item with ***minimum*** value from the list

Example
>>>a = [0, 5, 10, 15, 20, 25]
>>>min(a)
0

sum( )

- The sum( ) function returns the ***sum*** of all items on a list

Example
>>>a = [0, 5, 10, 15, 20, 25]
>>>sum(a)
75

sorted( )

- The sorted function returns a ***sorted list*** in ascending order.

Example
>>>a = [1, 9, 15, 11, 3, 7]
>>>sorted(a)
[1, 3, 7, 9, 11, 15]

list( )

- Convert othersequence like tuple, string ordictionary into a list.

Example
>>>atuple = (123, 'java', 'python')
>>>alist = list(atuple)
[123, 'java', 'python']

TUPLES (*16 & 8 marks)*

TUPLES

"A tuple is an immutable, ordered sequence of values of any data types (string, float, integer, etc.)"

- Values in the tuples are called elements or items.
- The elements in the tuples are immutable and indexed / ordered.
- The elements in the list are enclosed in **parentheses( )**separated by comma**.**

CREATING A LIST

Syntax

Tuple_name = (item1, item2, item3, item4……., item n)

Example

***# Empty list*:** A list that contains no elements is called an empty list

my_tuple = ( )

 # list of integers

my_tuple = (1, 2, 3)

 # list with mixed datatypes

my_tuple = (1, "Hello", 3.4)

ACCESSING TUPLE ELEMENTS

- Tuple elements can be accessed by using the square brackets or index operator **[].**
- **Syntax**

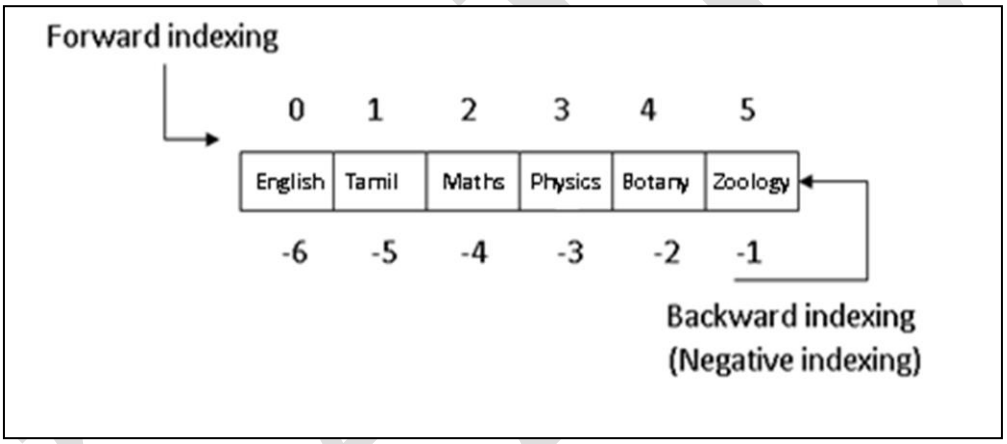
Tuple_variable [index_number]

Indexing

1. **Forward indexing:** Index starts from 0 to n-1.

Example

```
>>>subject= (“English”, “Tamil”, “Maths”, “Physics”, “Botany”, “Zoology”)
>>>subject [0]
‘English’#output
>>>subject [2]
    ‘Maths’      #output
>>>subject [5]
    ‘Zoology’    #output
```



2. **Negative (backward) indexing:** The index of -1 refers to the last item, -2 to the second last item and so on.

Example

```
>>>subject= (“English”, “Tamil”, “Maths”, “Physics”, “Botany”, “Zoology”)
>>>subject [-1]
‘Zoology’#output
>>>subject [-3]
‘Physics’      #output
>>>subject [-6]
‘English’    #output
```

SLICING TUPLES

- A segment of a tuple is called a slice

- The slicing operator [:] is used to access a range of elements in a tuple
- The colon inside the square brackets is used to separate two indices from each other.
- **Syntax**

Tuple_variable [start : end-1]

Example

```
>>>a= (‘parrot’, ’Dove’, ’duck’, ’cuckoo’)
>>>a [2 : 4]
        (‘duck’, ’cuckoo’)#output
>>>a [:3]
        (‘parrot’, ’Dove’, ’duck’)#output
>>>a [2:]
        (‘duck’, ’cuckoo’)#output
>>>a [:]
        (‘parrot’, ’Dove’, ’duck’, ’cuckoo’)#output
```

TUPLES ARE IMMUTABLE

- Tuples are ***immutable***
- Which means ***cannot change or modify*** any elements of a tuple.

Example

```
>>>a = (‘parrot’, ’Dove’, ’duck’, ’cuckoo’)
>>>a [3] = “crow”
```

Output:

TypeError: object doesn’t support item assignment

TUPLE ASSIGNMENT*(2 or 6 marks)*

- A programmer can assign multiple variables in one statement by using tuple assignment.
- Tuple assignment is an assignment with a sequence on the right side and a tuple of variables on the left.
- The right side is evaluated and then its elements are assigned to the variables on the left

Example

```
>>>a, b = 1, 2
>>>a
1
>>>b
2
```

- Where a, b is tuple of expressions
  1, 2 is tuple of variables
- Each value is assigned to its respective variables.
- The number of variables on the left and the number of values on the right have to be same.

Example

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

TUPLE AS RETURN VALUES

- Tuples can also be returned by the function as return values.
- A function can return more than one value using tuple.

Example

```
#function definition
>>>def div_mod(a,b):
quotient = a / b
remainder = a % b
        return    quotient, remainder
#function call
>>>x, y = 10, 3
>>>t = div_mod (x, y)
>>>print(t)
    (3, 1)#output
```

- In this example both the quotient and remainder can be computed at the same time.
- Two values will be returned, i.e., quotient and remainder by using the tuple as the return value of the function.

VARIABLE-LENGTH ARGUMENTS TUPLES

- Variable number of arguments can also be passed to a function.
- A variable name that is preceded by an asterisk (*) collects the arguments into a tuple

Example

```
#function definition
>>>def greeting (*t):
        i=0
        while i <len(t):
            print (“Hai”,t[i])
            i = i + 1
#function call
>>>greeting ("Anu","Kavin","Ram","Krish")
Output
HaiAnu
HaiKavin
Hai Ram
HaiKrish
```

UPDATING TUPLES

- Tuples are ***immutable***, which means cannot update or change the values of tuple elements.
- Can generate a new tuple rather than change the old tuples.

Example

```
>>>a = (‘parrot’, ’Dove’, ’duck’, ’cuckoo’)
>>>a [3] = “crow”
```

Output:

TypeError: object doesn’t support item assignment

DELETEING TUPLE ELEMENTS

- Removing **individual** tuple **elements*is not possible.***
- To remove an entire tuple, use the keyword **del.**

Syntax

del tuple_name

Example

```
>>>tup= (‘parrot’, ’Dove’, ’duck’, ’cuckoo’)
>>>tup
        (‘parrot’, ’Dove’, ’duck’, ’cuckoo’)
>>>deltup
>>>tup
```

Output:

NameError: name 'tup' is not defined (***meanstup is deleted***)

TUPLE OPERATION*(8 marks)*

- There are many operations that can be performed with tuple.
- The various tuple operations are,
  1. Tuple concatenation
  2. Repeating a tuple (replication operator)
  3. ‘in’ operator (Tuple Membership test)
  4. Iteration through a tuple

1. Tuple concatenation

- A number of tuple can be combined to form a single tuple is called ***tuple concatenation.***
- The (+) operator is to join the tuple.

Syntax

“tuple_variable 1” + “tuple _variable 2”+…..+“tuple _variable n”

Example
>>>a = ('x', 'y', 'z')
>>>b = (5, 10, 15)
>>>c = a + b
>>>print(c)
Output
('x', 'y', 'z', 5, 10, 15)

2. Repeating a list(Replication Operator)

- Group of tuple can be repeated by using the (*) operator.
- The (*) operator is the tuple replication operator.

Example
>>>a = (1, 2, 3)
>>>print (a* 3)
Output
(1, 2, 3, 1, 2, 3, 1, 2, 3)

Syntax

("tuple_variable" * n)

Where, **tuple_variable** is the variable name
***** is the list replication operator
n is the number of times the list to be repeated.

3. The 'in' operator (Tuple Membership test)

- Can test if an element is exists within a tuple or not, using the keyword '**in'**.

Example
>>>a = (1, 2, 3, 4, 5)
>>>5 in a
True
>>>10 in a
False

4. Iterating through a Tuple

- The most common way to traverse the elements of a tuple is using a 'for loop'.

Traversal using 'for' loop
Example
>>>bird = ('p', 'a', 'r',' r', 'o', 't')
>>>for letter in bird:
print(letter)
Output:
P a r r o t

BUILD-IN TUPLE METHODS*(8 marks)*

- In python count( ) and index( ) are the two methods that works with tuples.

Tuple Method in python

count( )	Returns the count number of items passed as an argument
index( )	Returns the index of the first matched item.

Count (x)

- The count method returns the **number** of times ***x***appears in the tuple.

Syntax

tuple_name.count(object)

Example

```
>>>t = ('a', 'p', 'p', 'l', 'e')
>>>print(a.count('p')
    2
```

index (x)

- The index method returns the index of the first matched item.in the tuple.

Syntax

tuple_name.index(object)

Example

```
>>>t = ('a', 'p', 'p', 'l', 'e')
>>>print(a.index('l')
3
```

BUILD-IN FUNCTIONS WITH TUPLES*(16 or 8 marks)*

- Python's built-in functions can be used with tuple to obtain needed value and execute various tasks.

len( )

- The length function returns the number of items on a tuple.

Example
>>>a=(0, 5, 10, 15, 20, 25)
>>>len(a)
6

Max( )

- The Max( ) function returns item with ***maximum*** value from thetuple.

Example
>>>a = (0, 5, 10, 15, 20, 25)
>>>max(a)
25

Min( )

- The Min( ) function returns item with ***minimum*** value from the tuple.

Example
>>>a = (0, 5, 10, 15, 20, 25)
>>>min(a)
0

sum( )

- The sum( ) function returns the ***sum*** of all items on a tuple

Example
>>>a = (0, 5, 10, 15, 20, 25)
>>>sum(a)
75

sorted( )

- The sorted function returns a ***sorted list*** in ascending order.

Example
>>>a = (1, 9, 15, 11, 3, 7)
>>>sorted(a)
(1, 3, 7, 9, 11, 15)

tuple( )

- Convert othersequence like list, string, or dictionary into a tuples

i) Converting string to tuple

Example
>>>str= "python"
>>>tuple(str)
Output:
('p', 'y', 't', 'h', 'o', 'n')

ii) Converting list to tuple

Example
>>>list= ['red', 'green', 'blue', 'yellow']
>>>tuple(list)
Output:
('red', 'green', 'blue', 'yellow')

iii) Converting dictionary to tuple

Example
>>>dict= {'Name' : 'Raju' , 'age' : 20, 'color' : 'white'}
>>>tuple(dict)
Output:
('Name', 'age', 'color')

DICTIONARY(*16 & 8 marks)*

DICTIONARY

"A dictionary is an unordered collection of key-value pairs which are separated by a colon and enclosed within curly braces { }"

- The mapping of a key and value is called as a ***key – value pair***

Key : value

- They together called as one item or element
- The ***keys are immutable*** but the ***values are mutable*** in dictionary
- The keys can be any data types like integer, float or a string.

CREATING A DICTIONARY

Syntax

dictionary_name = {key1:value1, key2: value2,……,key n : value n}

Example

 # Empty Dictionary

dict1 = { }

 # Dictionarywithinteger keys

dict1= {1: 'red', 2: 'green', 3: 'blue'}

 # Dictionarywithmixed keys

dict1 = {'Name': 'John', 2: 'Hello', 5.6: 'Height'}

ACCESSINGVALUES IN A DICTIONARY

- Dictionary elements can be accessed by using key enclosed in square brackets **[].**
- An error will occur when the key not exists in the dictionary is accessed.
- **Syntax**

dictionary_variable [key]

Example

```
>>>dict1 = {'Name': 'John', 'age': '25'}
>>>dict1 ['Name']
'John'     #output
>>>dict1 ['age']
 25        #output
```

- Accessing elements in dictionary using **get( ) method**

Example

```
>>>dict1 = {'Name': 'John', 'age': '25'}
>>>dict1.get('Name')
'John'      #output
>>>dict1.get('age')
 25         #output
```

UPDATING DICTIONARY

- The values in the dictionary can be changed, added or deleted.

Example

```
>>>dict1 = {'Name': 'John', 'age': '25'}

# updating a value
>>>dict1 ['Name'] = 30
>>> dict1
{'Name': 'John', 'age': '30'}

# Adding key: value pair
>>>dict1 ['address'] = 'Alaska'
>>> dict1
{'Name': 'John', 'age': '30', 'address': 'Alaska'}
```

REMOVING OR DELETING ELEMENTS FROM DICTIONARY (*6& 2 marks)*

- The items in the dictionary can be removed or deleted.

1. pop ( ) method

- The pop ( ) method is used to remove a specified key-value pair from a dictionary and return the value of the deleted key.

Example

```
>>>dict1 = {'Name': 'John', 'age': '30', 'address': 'Alaska'}
>>>dict1 .pop('age')
>>> dict1
{'Name': 'John', 'address': 'Alaska'}
```

2. popitem( ) method

- The popitem( ) method is used to remove a random key-value pair from a dictionary.

Example

```
>>>dict1 = {'Name': 'John', 'age': '30', 'address': 'Alaska'}
>>>dict1 .popitem( )
>>> dict1
{'Name': 'John', 'age': 30}
>>>dict1 .popitem( )
>>> dict1
{'Name': 'John'}
```

3. clear( ) method

- The popitem( ) method is used to remove all key-value pair from a dictionary.

Example

```
>>> dict1 = {'Name': 'John', 'age': '30', 'address': 'Alaska'}
>>>dict1 .clear( )
>>> dict1
{}      #Empty dictionary
```

4. **Using 'del' Keyword**
   - The ***del*** keyword is used to delete a dictionary.

Example
>>> dict1 = {'Name': 'John', 'age': '30', 'address': 'Alaska'}
>>>del dict1
>>> dict1
Output:
NameError: name 'dict1 is not defined

PROPERTIES OF DICTIONARY KEYS*(6 or 2 marks)*

- There are two important points to be followed about keys in dictionary they are,

1. **One key in a dictionary cannot have two values.**
   - This means no duplicate key is allowed.

Example
>>> dict1 = {'Name': 'John', 'age': '25', 'Name': 'Jim'}
>>>dict1 ['Name']
'Jim' ***#output***

   - When duplicate keys are encountered in a dictionary, the latest value is stored whereas the previous one is lost.

2. **Keys are immutable**
   - This means can use strings, numbers or tuples as dictionary keys but something like ['key'] is not allowed.

Example
>>> dict1 = {['Name']: 'John', 'age': '25'}
>>>print(dict1['Name'])
Output
Traceback (most recent call last):
File "test.py", line 2, in <module>
dict1 = {['Name']: 'John', 'age': '25'}
TypeError: list objects are unhashable

DICTIONARY OPERATIONS(*8 marks)*

DICTIONARY MEMBERSHIP TEST

- Can test if a key is in a dictionary or not using the keyword 'in'.
- Membership tests only for keys, not for values.

Example

```
>>>squares = {1: 1, 3: 9, 5: 25, 7: 49, 9: 81}
>>>print(1 in squares)
True      #output
>>>print(2 in squares)
False     #output
```

ITERATING THROUGH A DICTIONARY

- Using a '**for**' loop we can iterate though each key in a dictionary.

Example

```
squares = {1: 1, 3: 9, 5: 25, 7: 49, 9: 81}
for i in squares:
    print(squares[i])
```

Output

{1: 1, 3: 9, 5: 25, 7: 49, 9: 81}

BUILD-IN DICTIONARY METHODS*(16 &8 marks)*

keys() Method

- The keys( ) method returns a list of a dictionary ***keys.***

Syntax

dict.keys( )

Example
>>> dict1 = {'Name': 'John', 'age': '30', 'address': 'Alaska'}
>>>dict1.keys( )
Output
['Name', 'age', 'address']

values() Method

- The values( ) method returns a list of a dictionary ***values.***

Syntax

dict.values( )

Example
>>> dict1 = {'Name': 'John', 'age': '30', 'address': 'Alaska'}
>>>dict1.values( )
Output
['John', 30, 'Alaska']

Copy( )

- The copy method returns a shallow copy of the dictionary

Syntax

dictionary_name.copy( )

Example
>>>dict1 = {'Name': 'Manni', 'Age': 7, 'Class': 'First'}
>>>dict2 = dict1.copy()
>>> dict2
{'Name': 'Manni', 'Age': 7, 'Class': 'First'}

update() Method

- The update( ) method updates a dictionary with a set of key-value pairs from another dictionary.

Syntax

dict.update(dict)

Example
>>> dict1 = {'Name': 'John', 'age': '30', 'address': 'Alaska'}
>>>dict2 = {'dept' : 'production' , 'position': 'Manager'}
Output
{'Name': 'John', 'age': '30', 'address': 'Alaska', 'dept : 'production' , 'position': 'Manager'}

item( ) method

- The item( ) method returns a list of a dictionary's key-value pairs.

Syntax

dict.items( )

Example
>>> dict1 = {'Name': 'John', 'age': '25'}
>>>dict1.items( )
Output
[('Name','John'), ('age', '25')]

Clear ()

- The clear method removes all items from the dictionary

Syntax

dictionary_name.clear( )

Example
>>> dict1 = {'Name': 'John', 'age': '30', 'address': 'Alaska'}
>>>dict1 .clear( )
>>> dict1
{} ***#Empty dictionary***

get( )

- The get( ) method returns a value for the given key.
- If the key is not available then returns default value None.

Example
>>> dict1 = {'Name': 'John', 'age': '25'}
>>>dict1.get('Name')
'John' ***#output***

fromkeys( )

- The fromkeys( )method takes items on a sequence and uses them as keys to build a new dictionary.

Example

```
>>>keys = [‘monitor’, ‘CPU’, ‘mouse’]
>>>new_dict = dict.fromkeys(keys, 10)
>>>print(new_dict)
{'monitor': 10, 'CPU': 10, 'mouse': 10}
```

setdefault() Method

- The setdefault( ) method searches for a given key in a dictionary and returns the value if found. If not, it returns the given default value.

Syntax

dict.setdefault(key, default = None)

Example

```
>>> dict1 = {‘Name’: ‘John’, ‘age’: ‘30’, ‘address’: ‘Alaska’}
>>>dict1.setdefault(‘age’,None )
Output
30
```

BUILD-IN FUNCTIONS WITH DICTIONARY*(16 or 8 marks)*

len( )

- The length function returns the number of items in a dictionary.

Example

```
>>>squares = {1: 1, 3: 9, 5: 25, 7: 49, 9: 81}
>>>len(squares)
5
```

sorted( )

- The sorted function returns a new sorted list of keys in the dictionary.

Example
>>> squares = {1: 1, 3: 9, 5: 25, 7: 49, 9: 81}
>>>sorted(squares)
[1, 3, 5, 7, 9]

dict( )

- dict( ) function create dictionary out of a list of tuple pairs.
- Each pair will have two elements that can be used as a key and a value.

Example
>>>pairs = [('cat','kitten'),('dog','puppy'),('lion','cub')]
>>>dict(pairs)
Output:
('p', 'y', 't', 'h', 'o', 'n')

LIST COMPREHENSION *(8 marks)*

- List comprehension is an elegant and concise way to create new list from an existing list in Python.
- List comprehension consists of an expression followed by for statement inside square brackets.

Example: 1
>>>pow2 = [2 ** x for x in range(10)]
>>>print(pow2)

Output:
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512]

- This code is equivalent to

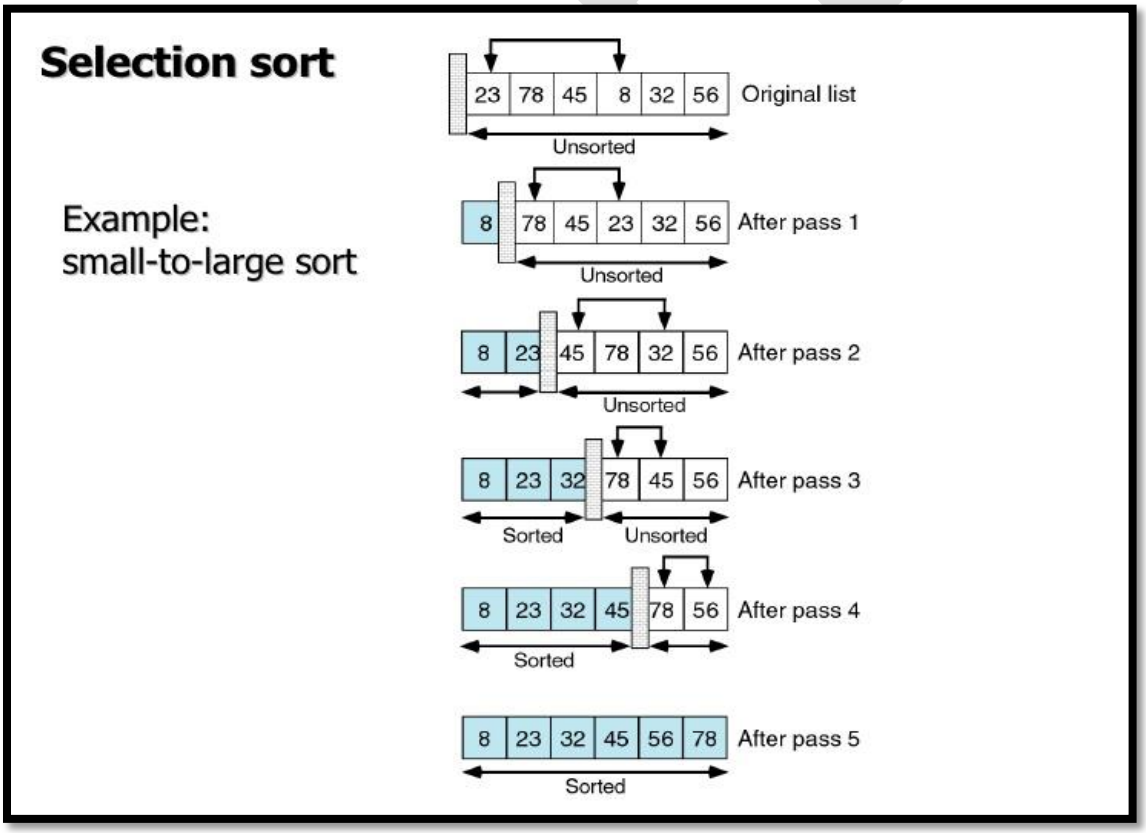
```
pow2 = []
for x in range(10):
pow2.append(2 ** x)
```

- A list comprehension can optionally contain more **for** or **if** statements.
- An optional if statement can filter out items for the new list.

Example: 2	**Example: 3**
>>>pow2 = [2 ** x for x in range(10) if x > 5] >>>print(pow2) ***Output:*** **[64, 128, 256, 512]**	>>>odd = [x for x in range(20) if x % 2 == 1] >>>print(odd) ***Output:*** **[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]**

1. Selection sort*(8 marks)*

Description



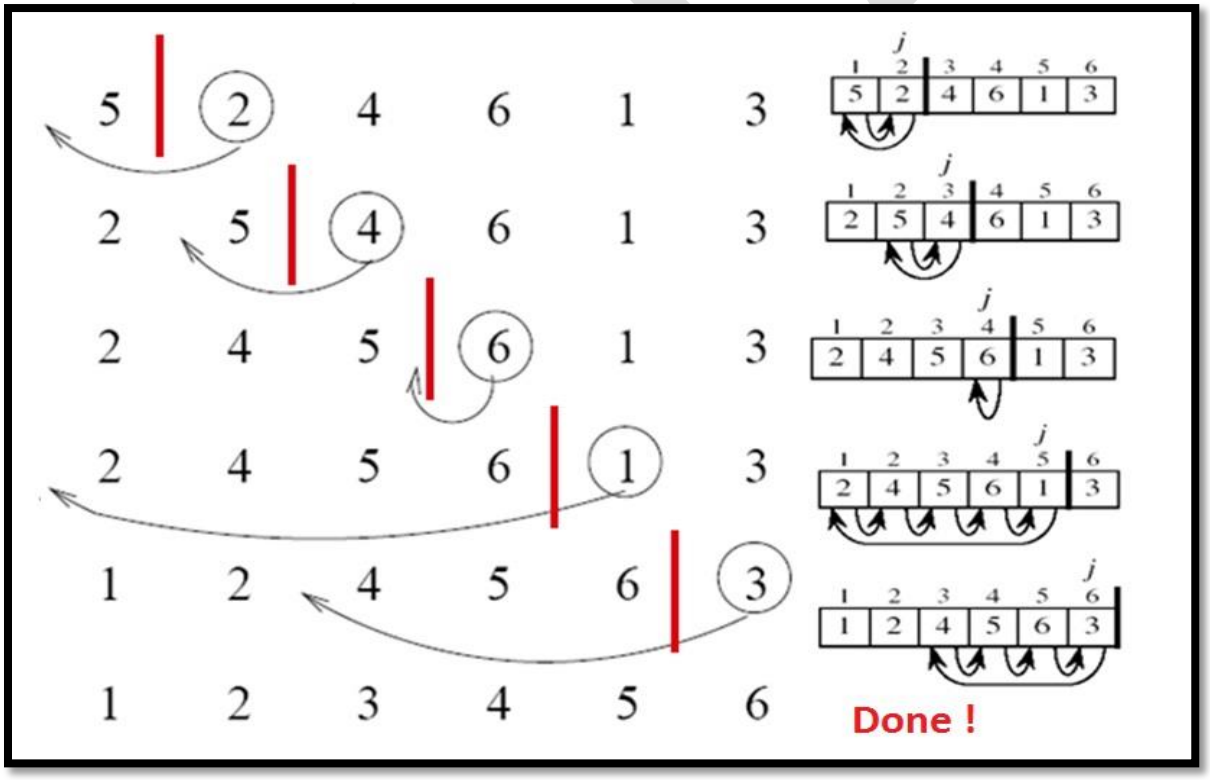
Program

```
defselectionSort(A):
        for i in range(len(A)-1,0,-1:
        max=0
for j in range(1,i+1):
        if A[j]>A[max]:
        max=j
        temp=A[i]
        A[i]=A[max
        A[max]=temp
    A=[54,26,93,17,77,31,44,55,20]
selectionSort(A)
print(A)
```

Output:
[17,20,26,31,44,54,55,77,93]

2. Insertion Sort *(8 marks)*

Description



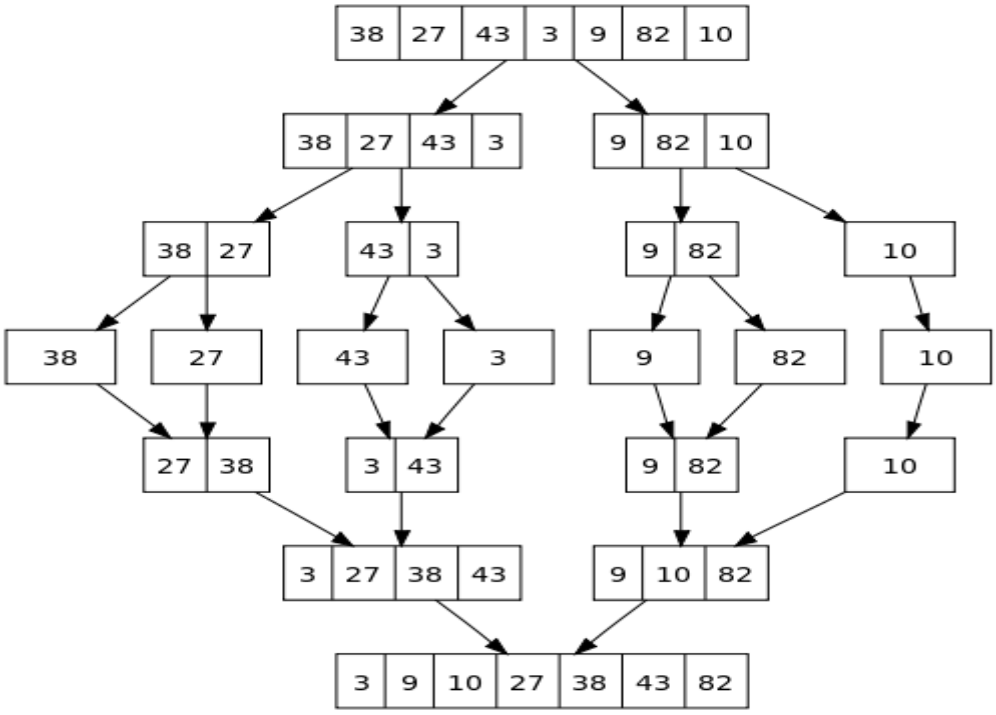
Program

```
definsertionSort(A):
for i in range(1,len(A)):
currentvalue=A[i]
position=i
while position>0 and A[position-1]>current value:
        A[position]=A[position-1]
position=position-1
    A[position]=currentvalue
A=[54,26,93,17,77,31,44,55,20]
insertionSort(A)
```

Output:
[17,20,26,31,44,54,55,77,93]

3. Merge Sort *(8 marks)*

Description



Program

```
defmergeSort(alist):
iflen(alist)>1:
mid=len(alist)//2
lefthalf=alist[:mid]
righthalf=alist[mid:]k
mergeSort(lefthalf)
mergeSort(righthalf)
        i=0
        j=0
        k=0
while i<len(lefthalf) and j<len(righthalf):
iflefthalf[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
      alist=[54,26,93,17,77,31,44,55,20]
      mergeSort(alist)
      print(alist)
```

Output:
[17,20,26,31,44,54,55,77,93]

4. Histogram*(8 marks)*

Program

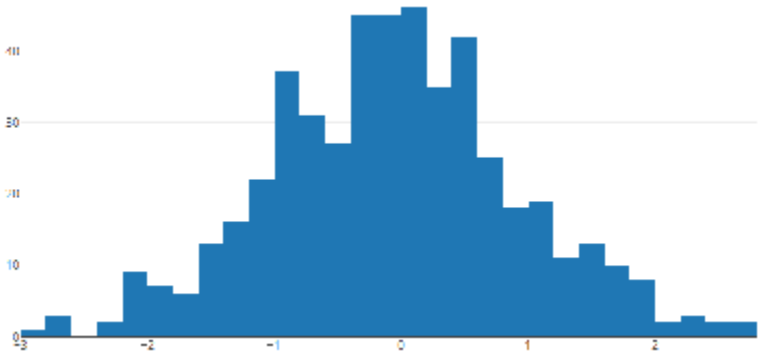
```
importplotly.plotlyaspy
importplotly.graph_objsasgo

importnumpyasnp
```

```
x=np.random.randn(500)
data=[go.Histogram(x=x)]

py.iplot(data,filename='basic histogram')
```

Output



IMPORTANT QUESTIONS
PART-B (2 Marks)

1. Define list with example ** ***Pg no:1***
2. How to access elements in a list ***Pg no:1***
3. How to create elements in a list ***Pg no:1***
4. Define nested list ***Pg no:1***
5. Define indexing ***Pg no:2***
6. Define forward and backward(negative) indexing ****Pg no:2-3***
7. How to get index of an object in a list ***Pg no:2***
8. How to slice a list with example ***Pg no:3***
9. How to change elements of list ***Pg no:4***
10. Define aliasing in list ** ***Pg no:4***
11. What is meant by cloning in list? ** ***Pg no:4***
12. What is meant by list loop (traversing a list)? ***Pg no:5***
13. Define list concatenation with example ** ***Pg no:5***
14. Define list replication(repetition)with example ** ***Pg no:6***
15. List some list methods ***Pg no:7***
16. List any four build in function in list ***Pg no:10***
17. Difference between del( ) & remove( ) methods of list. ***Pg no:9 & 10***

18. Define append( ) & extend( )***Pg no:7***
19. How to insert & reverse an object in a list ***Pg no:8***
20. How to remove last object from a list ***Pg no:9***
21. Define tuple with example ****Pg no:11***
22. How to access elements in a tuple ***Pg no:12***
23. How to create elements in a tuple ***Pg no:11***
24. How to slice a tuples with example ***Pg no:12***
25. Define tuple assignment with example ***Pg no:14***
26. Why tuple is immutable. Justify the answer? ***Pg no:14***
27. Define variable length argument in tuple ***Pg no:15***
28. How tuple is used as an return value in function ***Pg no:15***
29. List some tuple methods ***Pg no:18***
30. List any four build in function in tuple ***Pg no:19***
31. Difference between tuple & List ****Pg no:36***
32. Define dictionary with example ** ***Pg no:21***
33. How to create dictionary in python ***Pg no:21***
34. How to access elements in a dictionary ***Pg no:21***
35. How to create dictionary using tuples in python ***Pg no:29***
36. How elements in dictionary are removed or deleted ***Pg no:23***
37. How will you get all keys from the dictionary? ***Pg no:25***
38. How will you get all values from the dictionary? ***Pg no:26***
39. List some build in function in dictionary ***Pg no:25***
40. List some methods used in dictionary. ***Pg no:25***
41. List out the properties of Dictionary Keys. ***Pg no:24***
42. Define list comprehension ***Pg no:29***
43. What is the output of print list[0] if list=[‘abcd’,786,2.23,‘john’,70.2]
    Ans: ‘abcd’
44. What is the output of print list[1:3] if list=[‘abcd’,786,2.23,‘john’,70.2]
    Ans: [786,2.23]
45. What is the output of print list[2:] if list=[‘abcd’,786,2.23,‘john’,70.2]
    Ans: [2.23,‘john’,70.2]
46. What is the output of printtiny_list *2 if tiny_list= [123, ‘john’]
    Ans: [123, ‘john’, 123, ‘john’]
47. What is the output of print list + tiny_list *2 iflist=[‘abcd’,786,2.23,‘john’,70.2] and tiny_list= [123, ‘john’]
    Ans: [‘abcd’,786,2.23,‘john’,70.2,123, ‘john’, 123, ‘john’]

PART-C (6 Marks)

1. Explain in details about list methods
2. Discuss about operations in list
3. What is cloning? Explain it with example
4. What is aliasing? Explain with example
5. How can you pass list into function? Explain with example.
6. Explain tuples as return values with examples
7. Write a program for matrix multiplication
8. Write a program for matrix addition
9. Write a program for matrix subtraction
10. Write a program for matrix transpose
11. Write procedure for selection sort
12. Explain merge sort with an example
13. Explain insertion with example
14. Explain in detail about dictionaries and its methods.
15. Explain in detail about advanced list processing.

Questions	Option1	Option2	Option3
What is the output of the following code? for i in [1, 0]: print(i+1)	2,1	[2, 1]	2,0
What is the output of the following? x = ['ab', 'cd'] for i in x: i.upper() print(x)	['ab','cd'].	['AB','CD'].	[None, None].
What is correct syntax to copy one list into another?	listA = listB[]	listA = listB[:]	listA = listB[]()
Which statement is correct....?	List is mutable and Tuple is immutable	List is immutable and Tuple is mutable	Both are Immutable
Suppose list1 is [2, 33, 222, 14, 25], What is list1[-1] ?	Error	25	2
Which of the following date format should be used in place of ??? import time str = '02/06/1987' datetime_value = time.strptime(str, ???)	“%d/%m/%Y”	“%d/%M/%y”	“%D/%M/%Y”
Which of the following data type is used to store values in Key & Value format?	Class	List	Dictionary
What will be the output of 7^10 in python?	13	2	15
Which of the following is the use of function in python?	Functions are reusable pieces of programs	Functions don’t provide better modularity for your application	you can’t also create your own functions
What is the output of the following? i = 0 while i < 5: print(i) i += 1	0 1 2 0	0 1 2	error
A single object can be appended to the end of a list using the	insert()	read()	append()
A pair of _____ creates an empty dictionary	[]	{}	()
_____ on a dictionary returns a list of all the keys used in the dictionary	insert(d)	list(d)	append(d)
The ____ constructor builds dictionaries directly from sequences of key-value pairs	list()	tuple()	dict()

When looping through dictionaries, the key and corresponding value can be retrieved at the same time using the _____ method	list()	items()	dict()
When looping through a sequence, the position index and corresponding value can be retrieved at the same time using the _____ function	enumerate()	items()	dict()
To loop over two or more sequences at the same time, the entries can be paired with the _____ function	enumerate()	items()	dict()
To loop over a sequence in reverse, first specify the sequence in a forward direction and then call the _____ function.	dict()	reversed()	enumerate()
To loop over a sequence in sorted order, use the ____ function which returns a new sorted list while leaving the source unaltered	sorted()	zip()	reversed()
_____ add an item to the end of the list	list.extend(iterable)	list.append(x)	list.insert(i, x)
_____ extend the list by appending all the items from the iterable	list.extend(iterable)	list.append(x)	list.insert(i, x)
_____ insert an item at a given position	list.extend(iterable)	list.append(x)	list.insert(i, x)
____ raises a ValueError if there is no such items	list.append(x)	list.insert(i, x)	list.remove(x)
_____ remove the first item from the list whose value is equal to x	list.pop([i])	list.remove(x)	list.append(x)
_____ remove the item at the given position in the list, and return it	list.pop([i])	list.remove(x)	list.append(x)
_____ remove all items from the list	list.pop([i])	list.remove(x)	list.append(x)
_____ return the number of times x appears in the list	list.append(x)	list.count(x)	list.clear()

____ reverse the elements of the list in place	list.clear()	list.copy()	list.append(x)
____ is a computationally fast way to methodically access parts of given data	Tuple	Dictionary	List slicing
A ____ is a visual representation of the Distribution of a Quantitative variable	hologram	histogram	graph
A visual representation that gives a _____ display of value counts	binary	decimal	discretized
An object with more than one reference has more than one name, so we say that the object is _____	aliased	discretized	quantisized
_____ is a list of programming codes, including abstract data structure, used to calculate specified variables in a certain order	Tuple	Dictionary	List slicing
_____ removes and returns an element at the last element	a.pop(index)	a.pop()	a.remove(element)
Which of the following functions accepts only integers as arguments?	ord()	min()	chr()
Suppose there is a list such that: l=[2,3,4]. If we want to print this list in reverse order, which of the following	reverse(l)	list(reverse[(l)])	reversed(l)
What is the output of the functions shown below?	A & 65	Error & 65	A & Error
Which are the advantages of functions in python?	Reducing duplication of code	Decomposing complex problems into simpler	Improving clarity of the code
What are the two main types of functions?	Custom function	Built-in function & User defined function	User function
Where is function defined?	Module	Class	Another function

Option4	Answers
[2, 0]	2,1
["ab","cd"].	['ab','cd'].
listA = listB	listA = listB[:]
Both are Mutable	List is mutable and Tuple
None	25
“%d/%m/%y”	“%d/%m/%Y”
Tuple	Dictionary
21	13
All of the mentioned	Functions are reusable pieces of programs
0 1 2 1	0 1 2
write()	append()
All	{}
extend(d)	list(d)
extend()	dict()

tuple()	items()
tuple()	enumerate()
zip()	zip()
zip()	reversed()
enumerate()	sorted()
list.index(x[, start[, end]])	list.append(x)
list.remove(x)	list.extend (iterable)
list.remove(x)	list.insert(i, x)
list.pop([i])	list.remove(x)
list.append(x)	list.remove(x)
list.append(x)	list.pop([i])
list.clear()	list.clear()
list.pop([i])	list.count(x)

list.reverse()	list.reverse()
List	List slicing
flowchart	histogram
quantisized	discretized
histogram	aliased
List processing	List processing
a.append(x)	a.pop()
any()	chr()
list(reversed(l))	list(reversed(l))
Eror & Error	Error & 65
All of the mentioned	All of the mentioned
System function	Built-in function & User
All of the mentioned	All of the mentioned

SYLLABUS

Files: text files, reading and writing files, format operator, command line arguments, **Errors and Exceptions**: handling exceptions, **Modules, Packages**; Illustrative programs: word count, copy file.

File:

- File is a named location on disk to store related information. It is used to permanently store data in a memory (e.g. hard disk).

File Types:

1. Text file
2. Binary file

Text File	Binary file
Text file is a sequence of characters that can be sequentially processed by a computer in forward direction.	A binary files store the data in the binary format (i.e. 0's and 1's)
Each line is terminated with a special character, called the EOL or End of Line character	It contains any type of data (PDF , images , Word doc ,Spreadsheet, Zip files,etc)

Operations on Files:

In Python, a file operation takes place in the following order,

1. Opening a file
2. Reading / Writing file
3. Closing the file

How to open a file:

Syntax:	Example:
file_object=open("file_name.txt","mode")	f=open("sample.txt","w")

How to create a file:

Syntax:	Example:
file_object=open("file_name.txt","mode") file_object.write(string) file_object.close()	f=open("sample.txt","w") f.write("hello") f.close()

Modes in file:

modes	description
r	read only mode
w	write only
a	appending mode
r+	read and write mode
w+	write and read mode

Differentiate write and append mode:

write mode	append mode
It is use to write a string into a file.	It is used to append (add) a string into a file.
If file is not exist it creates a new file.	If file is not exist it creates a new file.
If file is exist in the specified name, the existing content will overwrite in a file by the given string.	It will add the string at the end of the old file.

File operations and methods:

S.No	Syntax	Example	Description
1	f.write(string)	f.write("hello")	Writing a string into a file.
2	f.writelines(sequence)	f.writelines("1st line \n second line")	Writes a sequence of strings to the file.
3	f.read(size)	f.read( ) #read entire file f.read(4) #read the first 4 charecter	To read the content of a file.
4	f.readline( )	f.readline( )	Reads one line at a time.
5	f.readlines( )	f.readlines( )	Reads the entire file and returns a list of lines.
6	f.seek(offset,whence) whence value is optional.	f.seek(0)	Move the file pointer to the appropriate position. It sets the file pointer to the starting of the file.
	whence =0 from begining	f.seek(3,0)	Move three character from the beginning.
	whence =1 from current position	f.seek(3,1)	Move three character ahead from the current position.
	whence =2 from last position	f.seek(-1,2)	Move to the first character from end of the file
7	f.tell( )	f.tell( )	Get the current file pointer position.
8	f.flush( )	f.flush( )	To flush the data before closing any file.
9	f.close( )	f.close( )	Close an open file.

10	f.name	f.name o/p: 1.txt	Return the name of the file.

11	f.mode	f.mode o/p: w	Return the Mode of file.
12	os.rename(old name,new name)	import os os.rename("1.txt","2.txt")	Renames the file or directory.
13	os.remove(file name)	import os os.remove("2.txt")	Remove the file.

Format operator

The argument of write( ) has to be a string, so if we want to put other values along with the string in a file, we have to convert them to strings.

Convert no into string:	**output**
>>> x = 52 >>> f.write(str(x))	"52"
Convert to strings using format operator, %	**Example:**
print ("format string"%(tuple of values)) file.write("format string"%(tuple of values)	>>>age=13 >>>print("The age is %d"%age) The age is 13
Program to write even number in a file using format operator	**OutPut**

f=open("t.txt","w")
n=eval(input("enter n:"))
for i in range(n):
  a=int(input("enter number:"))
  if(a%2==0):
  f.write(a)
f.close()
	enter n:4
	enter number:3
	enter number:4
	enter number:6
	enter number:8
	result in file t.txt
	4
	6
	8

- The first operand is the format string, which specifies how the second operand is formatted.
- The result is a string. For example, the format sequence '%d' means that the second operand should be formatted as an integer (d stands for "decimal"):

Format character	Description
%c	Character
%s	String formatting
%d	Decimal integer
%f	Floating point real number

Command line argument:

- The command line argument is used to pass input from the command line to your program when they are started to execute.
- Handling command line arguments with Python need sys module.
- sys module provides information about constants, functions and methods of the pyhton interpretor.

argv[ ] is used to access the command line argument. The argument list starts from 0.
sys.argv[0]= gives file name
sys.argv[1]=provides access to the first input

Example 1	**output**
import sys print("the file name is %s" %(sys.argv[0]))	python demo.py the file name is demo.py
addition of two num	**output**
import sys a= sys.argv[1] b= sys.argv[2] sum=int(a)+int(b) print("sum is",sum)	sam@sam~$ python sum.py 2 3 sum is 5
Word count using comment line arg:	**Output**

from sys import argv
a = argv[1].split()
dict = {}
for i in a:
  if i in dict:
  dict[i]=dict[i]+1
  else:
  dict[i] = 1
print(dict)
print(len(a))
	C:\Python34>python word.py
	"python is awesome lets program in python"
	{'lets': 1, 'awesome': 1, 'in': 1, 'python': 2, 'program': 1, 'is': 1}
	7

Errors and exception:

Errors

Errors are the mistakes in the program also referred as bugs. They are almost always the fault of the programmer. The process of finding and eliminating errors is called debugging. Errors can be classified into three major groups:

1. Syntax errors
2. Runtime errors
3. Logical errors

Syntax errors

- Syntax errors are the errors which are displayed when the programmer do mistakes when writing a program.
- When a program has syntax errors it will not get executed.
- Common Python syntax errors include:
    1. leaving out a keyword
    2. putting a keyword in the wrong place
    3. leaving out a symbol, such as a colon, comma or brackets
    4. misspelling a keyword
    5. incorrect indentation
    6. empty block

Runtime errors:

- If a program is syntactically correct – that is, free of syntax errors – it will be run by the Python interpreter.
- However, the program may exit unexpectedly during execution if it encounters a runtime error.
- When a program has runtime error I will get executed but it will not produce output.
- Common Python runtime errors include:
    1. division by zero
    2. performing an operation on incompatible types
    3. using an identifier which has not been defined
    4. accessing a list element, dictionary value or object attribute which doesn't exist
    5. trying to access a file which doesn't exist

Logical errors:

- Logical errors are the most difficult to fix.
- They occur when the program runs without crashing, but produces an incorrect result.
- Common Python logical errors include:
    1. using the wrong variable name
    2. indenting a block to the wrong level
    3. using integer division instead of floating-point division
    4. getting operator precedence wrong
    5. making a mistake in a boolean expression

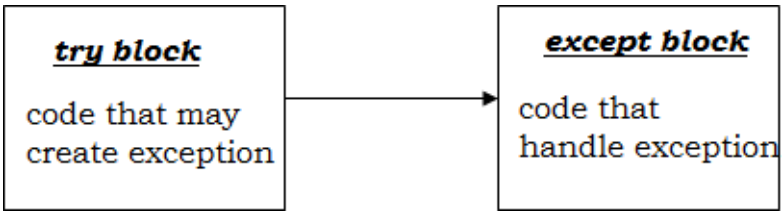
Exceptions:

- An exception(runtime time error)is an error, which occurs during the execution of a program that disrupts the normal flow of the program's instructions.
- When a Python script raises an exception, it must either handle the exception immediately otherwise it terminates or quit.

S.No.	Exception Name	Description
1	FloatingPointError	Raised when a floating point calculation fails.
2	ZeroDivisionError	Raised when division or modulo by zero takes place for all numeric types.
3	AttributeError	Raised in case of failure of attribute reference or assignment.
4	ImportError	Raised when an import statement fails.
5	KeyboardInterrupt	Raised when the user interrupts program execution, usually by pressing Ctrl+c.
6	IndexError	Raised when an index is not found in a sequence
7	KeyError	Raised when the specified key is not found in the dictionary.
8	NameError	Raised when an identifier is not found in the local or global name space
9	IOError	Raised when an input/ output operation fails, such as the print statement or the open() function when trying to open a file that does not exist.
10	SyntaxError	Raised when there is an error in Python syntax.
11	IndentationError	Raised when indentation is not specified properly.
12	SystemError	Raised when the interpreter finds an internal problem, but when this error is encountered the Python interpreter does not exit.
13	SystemExit	Raised when Python interpreter is quit by using the sys.exit() function. If not handled in the code, causes the interpreter to exit.
14	TypeError	Raised when an operation or function is attempted that is invalid for the specified data type.
15	ValueError	Raised when the built-in function for a data type has the valid type of arguments, but the arguments have invalid values specified.
16	RuntimeError	Raised when a generated error does not fall into any category.

Exception Handling:

- Exception handling is done by try and catch block.
- Suspicious code that may raise an exception, this kind of code will be placed in try block.
- A block of code which handles the problem is placed in except block.



Catching Exceptions:

1. **try…except**
2. **try…except…inbuilt exception**
3. **try… except…else**
4. **try…except…else….finally**
5. **try.. except..except..**
6. **try…raise..except..**

try … except

- In Python, exceptions can be handled using a try statement.
- A critical operation which can raise exception is placed inside the try clause and the code that handles exception is written in except clause.
- It is up to us, what operations we perform once we have caught the exception. Here is a simple example.

Syntax

```
try:
    code that create exception
except:
    exception handling statement
```

Example:	Output

try:
  age=int(input("enter age:"))
  print("ur age is:",age)
except:
  print("enter a valid age")
	enter age:8
	ur age is: 8
	enter age:f
	enter a valid age

try…except…inbuilt exception

Syntax

```
try:
    code that create exception
except inbuilt exception:
    exception handling statement
```

Example:	Output

try:
  age=int(input("enter age:"))
  print("ur age is:",age)
except ValueError:
  print("enter a valid age")
	enter age:d
	enter a valid age

try ... except ... else clause

- Else part will be executed only if the try block doesn't raise an exception.
- Python will try to process all the statements inside try block. If value error occurs, the flow of control will immediately pass to the except block and remaining statement in try block will be skipped.

Syntax

```
try:
   code that create exception
except:
   exception handling statement
else:
    statements
```

Example program	Output

try:
  age=int(input("enter your age:"))
except ValueError:
  print("entered value is not a number")
else:
  print("your age :",age)
	enter your age: six
	entered value is not a number
	enter your age:6
	your age is 6

try ... except...finally

A finally clause is always executed before leaving the try statement, whether an exception has occurred or not.

Syntax

```
try:
   code that create exception
except:
   exception handling statement
else:
  statements
finally:
  statements
```

Example program	Output

try:
age=int(input("enter your age:"))
except ValueError:
print("entered value is not a number")
else:
print("your age :",age)
finally:
print("Thank you")
	enter your age: six
	entered value is not a number
	Thank you
	enter your age:5
	your age is 5
	Thank you

try...multiple exception:

Syntax

```
try:
   code that create exception
except:
   exception handling statement
except:
  statements
```

Example	Output:
a=eval(input("enter a:")) b=eval(input("enter b:")) try: c=a/b print(c) except ZeroDivisionError: print("cant divide by zero") except ValueError: print("its not a number")	enter a:2 enter b:0 cant divide by zero enter a:2 enter b: h its not a number

Raising Exceptions

In Python programming, exceptions are raised when corresponding errors occur at run time, but we can forcefully raise it using the keyword raise.

Syntax:

>>> raise error name

Example:	Output:

try:
age=int(input("enter your age:"))
if (age<0):
raise ValueError("Age can't be negative")
except ValueError:
print("you have entered incorrect age")
else:
print("your age is:",age)
	enter your age:-7
	Age can't be negative

MODULES:

- A module is a file containing Python definitions ,functions, statements and instructions.
- Standard library of Python is extended as modules.
- To use modules in a program, programmer needs to import the module.
- To get information about the functions and variables supported module you can use the built-in function help(Module name), Eg: help("math").
- The dir() function is used to list the variables and functions defined inside a module. If an argument, i.e. a module name is passed to the dir, it returns that modules variables and function names else it returns the details of the current module. Eg: dir(math)

OS module

- The OS module in python provides functions for interacting with the operating system
- To access the OS module have to import the OS module in our program

import os

method	example	description
name	os.name 'nt'	This function gives the name of the operating system.
getcwd()	os.getcwd() 'C:\\Python34'	returns the Current Working Directory(CWD) of the file used to execute the code.
mkdir(folder)	os.mkdir("python")	Create a directory(folder) with the given name.
rename(oldname, new name)	os.rename("python","pspp")	Rename the directory or folder
remove("folder")	os.remove("pspp")	Remove (delete) the directory or folder.

getuid( )	os.getuid( )	Return the current process's user id.
environ	os.environ	Get the users environment

Sys module

- Sys module provides information about constants, functions and methods.
- It provides access to some variables used or maintained by the interpreter.

import sys

method	example	description
sys.argv	sys.argv	Provides The list of command line arguments passed to a Python script
	sys.argv[0]	Provides to access the file name
	sys.argv[1]	Provides to access the first input
sys.path	sys.path	It provides the search path for modules
sys.path.append()	sys.path.append()	Provide the access to specific path to our program
sys.platform	sys.platform 'win32'	Provides information about the operating system platform
sys.exit	sys.exit <built-in function exit>	Exit from python

Steps to create the own module

Here we are going to create calc module: our modules contains four functions (i.e) add(),sub(),mul(),div()

Program for calculator module	output

Module Name: calc.py
def add(a,b):
print(a+b)
def sub(a,b):
print(a-b)
def mul(a,b):
print(a*b)
def div(a,b):
print(a/b)
	import calculator
	calculator.add(2,3)
	
	Output
	>>> 5

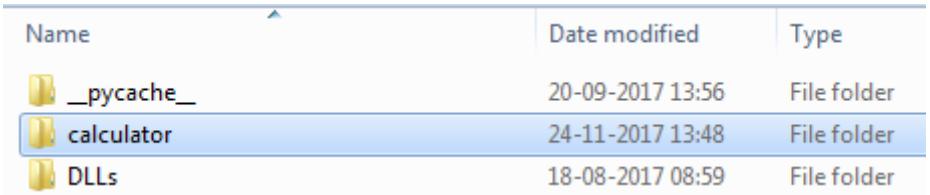
Package:

- A package is a collection of Python modules. Module is a single Python file containing function definitions; a package is a directory (folder) of Python modules containing an additional _ _init_ _.py file, to differentiate a package from a directory.
- Packages can be nested to any depth, provided that the corresponding directories contain their own _ _init_ _.py file.
- _ _init_ _.py file is a directory indicates to the python interpreter that the directory should be treated like a python package._ _init_ _.py is used to initialize the python package.

Steps to create a package

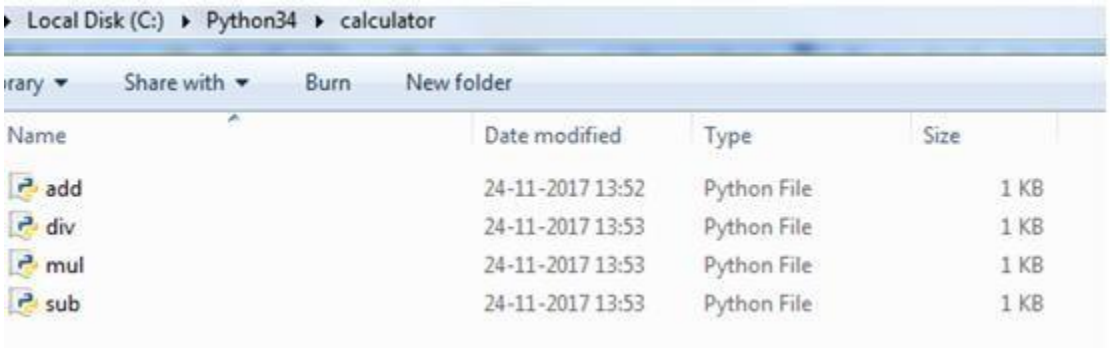
Step 1: Create the Package Directory

Create a directory(folder) and give it your package's name. Here the package name is calculator.



Step 2: write Modules for calculator directory add save the modules in calculator directory.

Here four modules have created for calculator directory.



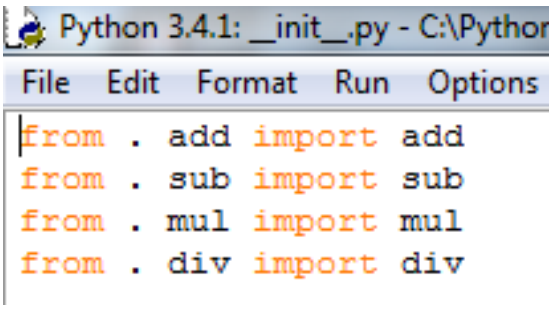
add.py	sub.py	mul.py	div.py
def add(a,b): print(a+b)	def sub(a,b): print(a-b)	def mul(a,b): print(a*b)	def div(a,b): print(a/b)

Step 3: Add the _ _init_ _.py File in the calculator directory

A directory must contain a file named _ _init_ _.py in order for Python to consider

it as a package.

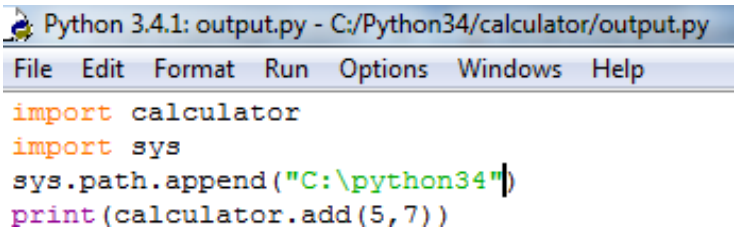
Add the following code in the __init__.py file



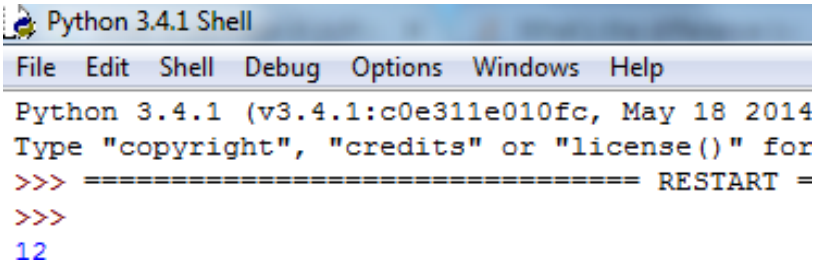
Step 4:To test your package.

Import calculator package in your program and add the path of your package in your program by using sys.path.append().

Here the path is "C:\python34"



OUTPUT:



Illustrative Programs in unit 5:

Word count	Output

from sys import argv
a = argv[1].split()
dict = {}
for i in a:
  if i in dict:
  dict[i]=dict[i]+1
  else:
  dict[i] = 1
print(dict)
print(len(a))
	C:\Python34>python word.py
	"python is awesome lets program in python"
	{'lets': 1, 'awesome': 1, 'in': 1, 'python': 2, 'program': 1, 'is': 1}
	7

Copy a file	**Output**
f1=open("1.txt","r") f2=open("2.txt","w") for i in f1: f2.write(i) f1.close() f2.close()	no output internally the content in f1 will be copied to f2
copy and display contents	**Output**
f1=open("1.txt","r") f2=open("2.txt","w+") for i in f1: f2.write(i) f2.seek(0) print(f2.read()) f1.close() f2.close()	hello welcome to python programming (content in f1 and f2)

Possible Questions

PART – B (2 marks)

1. Point out different modes of file opening
2. Differentiate text file and binary file.
3. Distinguish between files and modules
4. List down the operations on file.
5. list down some inbuilt exception.
6. Define read and write file.
7. Differentiate write and append mode.
8. Describe renaming and remove
9. Discover the format operator available in files.

10. Explain with example the need for exceptions
11. Explain built in exceptions
12. Difference between built in exceptions and handling exception
13. Write a program to write a data in a file for both write and append modes.
14. How to import statements?
15. Express about namespace and scoping.
16. Difference between global and local.
17. Identify what are the packages in python
18. Examine buffering.
19. Discover except Clause with Multiple Exceptions
20. Differentiate mutable.

Part-C (6 marks)

1. Write a Python program to demonstrate the file I/O operations
2. Discuss with suitable examples

i) Close a File.

ii) Writing to a File.

3. Write a program to catch a Divide by zero exception. Add a finally block too.
4. Explain format operator in detail.
5. Describe in detail about Exception with Arguments
6. Describe in detail about user – defined Exceptions
7. Explain with example of closing a file
8. Discover syntax for reading from a file
9. Structure Renaming a file
10. Explain about the Files Related Methods
11. Describe the import Statements
12. Describe the from…import statements
13. Describe in detail locating modules
14. Identify the various methods used to delete the elements from the dictionary
15. Describe in detail exception handling with program
16. Write a program to find the one's complement of binary number using file

Questions	Option1	Option2	Option3	Option4	Answers
The default type of reading file in python is _____	text mode	binary mode	character mode	string mode	text mode
____ is a sequence of characters that can be sequentially processed by a computer in forward direction	Text file	binary file	random file	sequential file	Text file
A _____ store the data in the binary format	random file	binary files	sequential file	Text file	binary files
____ is use to write a string into a file	append mode	read and write mode	write mode	read mode	write mode
____ is used to append (add) a string into a file	append mode	read and write mode	write mode	read mode	append mode
_____ reads one line at a time	f.write(string)	f.readline( )	f.read(size)	f.seek(0)	f.readline( )
_____ reads the entire file and returns a list of lines	f.seek(0)	f.readline( )	f.readlines( )	f.write(string)	f.readlines( )
____ move the file pointer to the appropriate position and it sets the file pointer to the starting of the file	f.seek(0)	f.readline( )	f.readlines( )	f.write(string)	f.seek(0)
_____ get the current file pointer position	f.seek(0)	f.tell( )	f.flush( )	f.close( )	f.tell( )
____ to flush the data before closing any file	f.seek(0)	f.tell( )	f.flush( )	f.close( )	f.flush( )
_____ move to the first character from end of the file	f.seek(0)	f.seek(-1,2)	f.flush( )	f.close( )	f.seek(-1,2)
____ move three character ahead from the current position	f.seek(-1,2)	f.close( )	f.seek(3,1)	f.seek(0)	f.seek(3,1)
_____ is used to pass input from the command line to your program when they are started to execute	string	format	command line argument	parameter	command line argument
____ gives file name	sys.argv[1]	sys.argv[2]	sys.argv[0]	sys.argv[3]	sys.argv[0]
____ provides access to the first input	sys.argv[1]	sys.argv[2]	sys.argv[0]	sys.argv[3]	sys.argv[1]
The process of finding and eliminating errors is called _____	error	exception	debugging	bug	debugging
_____ are the errors which are displayed when the programmer do mistakes when writing a program	logical error	semantical error	runtime error	Syntax errors	Syntax errors
_____ are the most difficult to fix	Logical error	semantical error	runtime error	Syntax errors	Logical error
_____ which occurs during the execution of a program that disrupts the normal flow of the program's instructions	Syntax errors	Exception	Logical error	runtime error	Exception
The _____ in python provides functions for interacting with the operating system	math	OS module	random	Sys	OS module
Standard library of Python is extended as _____	functions	modules	exceptions	try	modules
____ module provides information about constants, functions and methods	OS module	math	random	Sys	Sys
_____ provides the search path for modules	sys.argv	sys.path.append()	sys.path	sys.platform	sys.path
_____ provide the access to specific path to our program	sys.argv	sys.path.append()	sys.path	sys.platform	sys.path.append()
_____ provides information about the operating system platform	sys.path	sys.platform	sys.argv	sys.path.append()	sys.platform

A _____ is a collection of Python modules	sys.path	math	OS module	package	package
To open a file c:\scores.txt for reading, we use	infile = open("c:\scores.txt", "r")	infile = open("c:\\scores.txt", "r")	infile = open(file = "c:\scores.txt", "r")	infile = open(file = "c:\\scores.txt", "r")	infile = open("c:\\scores.txt", "r")
To open a file c:\scores.txt for writing, we use	outfile = open("c:\scores.txt", "w")	outfile = open("c:\\scores.txt", "w")	outfile = open(file = "c:\scores.txt", "w")	outfile = open(file = "c:\\scores.txt", "w")	outfile = open("c:\\scores.txt", "w")
To open a file c:\scores.txt for appending data, we use	outfile = open("c:\\scores.txt", "a")	outfile = open("c:\\scores.txt", "rw")	outfile = open(file = "c:\scores.txt", "w")	outfile = open(file = "c:\\scores.txt", "w")	outfile = open("c:\\scores.txt", "a")
Which of the following statements are true?	When you open a file for reading, if the file does not exist, an error occurs	When you open a file for writing, if the file does not exist, a new file is created	When you open a file for writing, if the file exists, the existing file is overwritten with the new file	All of the mentioned	All of the mentioned
To read two characters from a file object infile, we use	infile.read(2)	infile.read()	infile.readline()	infile.readlines()	infile.read()
To read the next line of the file from a file object infile, we use	infile.read(2)	infile.read()	infile.readline()	infile.readlines()	infile.readline()
To read the remaining lines of the file from a file object infile, we use	infile.read(2)	infile.read()	infile.readline()	infile.readlines()	infile.readlines()
The readlines() method returns	str	a list of lines	a list of single characters	a list of integers	a list of lines
Which are the two built-in functions to read a line of text from standard input, which by default comes from the keyboard?	Raw_input & Input	Input & Scan	Scan & Scanner	Scanner	Raw_input & Input
Which one of the following is not attributes of file	closed	softspace	rename	mode	rename
What is the use of tell() method in python?	tells you the current position within the file	tells you the end position within the file	tells you the file is opened or not	tells you the file is closed	tells you the current position within the file
What is the syntax of rename() a file?	rename(current_file_name, new_file_name)	rename(new_file_name, current_file_name,)	rename(()(current_file_name, new_file_name))	none	rename(current_file_name, new_file_name)
What is the syntax of remove() a file?	remove(file_name)	remove(new_file_name, current_file_name,)	remove(() , file_name))	none	remove(file_name)
What is the use of seek() method in files?	sets the file's current position at the offset	sets the file's previous position at the offset	sets the file's current position within the file	none	sets the file's current position at the offset
What is the use of truncate() method in file?	truncates the file size	deletes the content of the file	deletes the file size	none	truncates the file size
Which is/are the basic I/O connections in file?	Standard Input	Standard Output	Standard Errors	All of the mentioned	All of the mentioned
Which of the following mode will refer to binary data?	r	w	+	b	b
What is the pickling?	It is used for object serialization	It is used for object deserialization	Standard Errors	All	It is used for object serialization
What is the correct syntax of open() function?	file = open(file_name [, access_mode][, buffering])	file object = open(file_name [, access_mode][, buffering])	file object = open(file_name)	file = open(file_name)	file object = open(file_name [, access_mode][, buffering])
Correct syntax of file.writelines() is?	file.writelines(sequence)	fileObject.writelines()	fileObject.writelines(sequence)	none	fileObject.writelines(sequence)
Correct syntax of file.readlines() is?	fileObject.readlines(sizehint)	fileObject.readlines( )	fileObject.readlines(sequence)	none	fileObject.readlines(sizehint)

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed to be University)
(Established Under Section 3 of UGC Act 1956)
COIMBATORE – 641 021

INFORMATION TECHNOLOGY
Third Semester
FIRST INTERNAL EXAMINATION - July 2019

PROGRAMMING IN PYTHON

Class & Section: II B.Sc IT **Duration: 2 hours**
Date & Session: 23.7.19 (FN) **Maximum marks: 50 marks**
Sub.Code: I8ITU304B

PART- A (20 * 1= 20 Marks)
Answer ALL the Questions

1. Last step in process of problem solving is to
   a. design a solution
   b. define a problem
   c. practicing the solution
   d. organizing the data
2. Error in a program is called
   a. bug b. debug c. virus d. noise
3. Error which occurs when program tried to read from file without opening it is classified as
   a. execution error messages
   b. built in messages
   c. user-defined messages
   d. half messages
4. _____ is the process of formulating a problem, finding a solution, and expressing the solution
   a. problem solving b. Recover c. Format d. Retrieve
5. _____ is the output of the compiler after it translates the program.
   a. Coding b. source code c. object code d. program
6. _____ is the structure of a program
   a. Syntax b. Source c. Semantics d. Algorithm
7. What is a property of a program that can run on more than one kind of computer?
   a. executable b. source code c. Coding d. Portability
8. _____ is another name for object code that is ready to be executed.
   a. executable b. source code c. Coding d. program
9. _____ is the meaning of a program
   a. Syntax b. Source c. Semantics d. Algorithm
10. _____ is to examine a program and analyze the syntactic structure.
    a. Syntax b. Source c. Semantics d. parse
11. What is answer of this expression, 22 % 3 is?
    a. 7 b. 1 c. 0 d. 5
12. What is the output of this expression, 3*1**3?
    a. 27 b. 9 c. 3 d. 1
13. What datatype is the object below?
    a. list b. dictionary c. array d. tuple
14. What is the output of the following? print("Hello {name1} and {name2}".format(name1='foo', name2='bin'))?
    a. Hello foo and bin
    b. Hello {name1} and {name2}
    c. Error
    d. Hello

15. What is the output of the following? print("Hello {name1} and {name2}".format(name1='foo', name2='bin'))?
    a. The sum of 2 and 10 is 12
    b. The sum of 10 and a is 14
    c. The sum of 10 and a is c
    d. Error
16. What is the result of round(0.5) – round(-0.5)?
    a. 1 b. 2 c. 0 d. 3
17. What is the maximum possible length of an identifier?
    a. 31 characters
    b. 63 characters
    c. 79 characters
    d. None
18. All keywords in Python are in
    a. lower case
    b. UPPER CASE
    c. Capitalized
    d. Both lower case and UPPER CASE
19. Which of the following is an invalid statement?
    a. abc = 1,000,000
    b. a b c = 1000 2000 3000
    c. a,b,c = 1000, 2000, 3000
    d. a_b_c = 1,000,000
20. Which of these in not a core data type?
    a. Lists b. Dictionary c. Tuples **d.** Class

PART B (3 * 2 = 6 Marks)
Answer ALL the Questions

21. What is mean by problem solving?
22. Define iteration
23. Differentiate algorithm and pseudo code?

PART C (3 * 8 = 24 Marks)
Answer ALL the Questions

24. a. Write algorithm, pseudo code and flow chart for any example?

(OR)

b. Explain in detail about problem solving techniques

25. a. Discuss the symbols and rules for drawing flowchart with the example?

(OR)

b. Discuss the various operation that can be performed on a tuple and Lists (minimum 5) with an example program

26. a. Explain in detail about various data types in Python with an example?

(OR)

b. Explain the different types of operators in python with an example.

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed to be University)
(Established Under Section 3 of UGC Act 1956)
COIMBATORE – 641 021

INFORMATION TECHNOLOGY
Third Semester
FIRST INTERNAL EXAMINATION - July 2019

PROGRAMMING IN PYTHON

Class & Section: II B.Sc IT **Duration: 2 hours**
Date & Session: 23.7.19 (FN) **Maximum marks: 50 marks**
Sub.Code: I8ITU304B

PART- A (20 * 1= 20 Marks)
Answer ALL the Questions

1. Last step in process of problem solving is to
   practicing the solution
2. Error in a program is called
   bug
3. Error which occurs when program tried to read from file without opening it is classified as
   execution error messages
4. _____ is the process of formulating a problem, finding a solution, and expressing the solution
   problem solving
5. _____ is the output of the compiler after it translates the program.
   object code
6. _____ is the structure of a program
   Syntax
7. What is a property of a program that can run on more than one kind of computer?
   Portability
8. _____ is another name for object code that is ready to be executed.
   executable
9. _____ is the meaning of a program
   Syntax
10. _____ is to examine a program and analyze the syntactic structure.
    parse
11. What is answer of this expression, 22 % 3 is?
    1
12. What is the output of this expression, 3*1**3?
    3
13. What datatype is the object below?
    list
14. What is the output of the following? print("Hello {name1} and {name2}".format(name1='foo', name2='bin'))?
    Hello foo and bin

15. What is the output of the following? print("Hello {name1} and {name2}".format(name1='foo', name2='bin'))?
    The sum of 2 and 10 is 12
16. What is the result of round(0.5) – round(-0.5)?
    2
17. What is the maximum possible length of an identifier?
    None
18. All keywords in Python are in
    Both lower case and UPPER CASE
19. Which of the following is an invalid statement?
    a b c = 1000 2000 3000
20. Which of these in not a core data type?
    Class

PART B (3 * 2 = 6 Marks)
Answer ALL the Questions

21. What is mean by problem solving?
    Problem solving is the systematic approach to define the problem and creating number of solutions. The problem solving process starts with the problem specifications and ends with a correct program.
22. Define iteration
    In some programs, certain set of statements are executed again and again based upon conditional test. i.e. executed more than one time. This type of execution is called looping or iteration.
23. Differentiate algorithm and pseudo code?

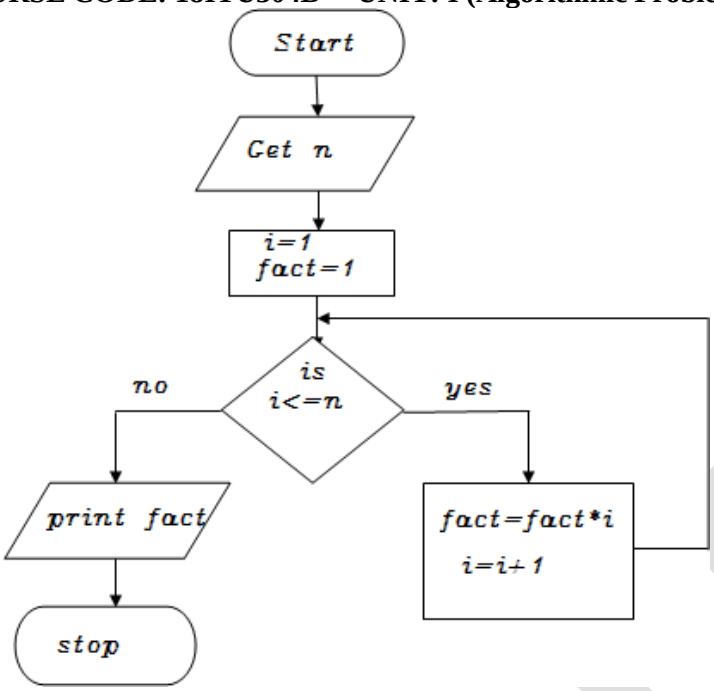
Flowchart	Pseudo code
It is a graphical representation of algorithm	It is a language representation of algorithm.
not need knowledge of program to draw or understand flowchart	Not need knowledge of program language to understand or write a pseudo code.

PART C (3 * 8 = 24 Marks)
Answer ALL the Questions

24. a. Write algorithm, pseudo code and flow chart for any example?

Step 1: start
step 2: get n value
step 3: set initial value i=1, fact=1
Step 4: check i value if(i<=n) goto step 5 else goto step8 step 5:
calculate fact=fact*i
step 6: increment i value by 1 step 7:
goto step 4
step 8: print fact value
step 9: stop

```
BEGIN GET n
INITIALIZE i=1,fact=1
WHILE(i<=n) DO
    fact=fact*i
    i=i+1
ENDWHILE
PRINT fact
END
```



(OR)

b. Explain in detail about problem solving techniques

Problem solving technique is a set of techniques that helps in providing logic for solving a problem.

Problem Solving Techniques:

Problem solving can be expressed in the form of

1. Algorithms.
2. Flowcharts.
3. Pseudo codes.
4. programs

ALGORITHM

It is defined as a sequence of instructions that describe a method for solving a problem. In other words it is a step by step procedure for solving a problem.

Properties of Algorithms

- Should be written in simple English
- Each and every instruction should be precise and unambiguous.
- Instructions in an algorithm should not be repeated infinitely.
- Algorithm should conclude after a finite number of steps.
- Should have an end point
- Derived results should be obtained only after the algorithm terminates.

Qualities of a good algorithm

The following are the primary factors that are often used to judge the quality of the algorithms.

Time – To execute a program, the computer system takes some amount of time. The lesser is the time required, the better is the algorithm.

Memory – To execute a program, computer system takes some amount of memory space. The lesser is the memory required, the better is the algorithm.

Accuracy – Multiple algorithms may provide suitable or correct solutions to a given problem, some of these may provide more accurate results than others, and such algorithms may be suitable.

Example:

Example

Write an algorithm to print „Good Morning"

Step 1: Start
Step 2: Print "Good Morning"
Step 3: Stop

2. BUILDING BLOCKS OF ALGORITHMS (statements, state, control flow, functions)

Algorithms can be constructed from basic building blocks namely, sequence, selection and iteration.

2.1. Statements:

Statement is a single action in a computer.

In a computer statements might include some of the following actions

- input data-information given to the program
- process data-perform operation on a given input
- output data-processed result

2.2. State:

Transition from one process to another process under specified condition with in a time is called state.

2.3. Control flow:

The process of executing the individual statements in a given order is called control flow.

The control can be executed in three ways

1. sequence
2. selection
3. iteration

Sequence:

All the instructions are executed one after another is called sequence execution.

Example:

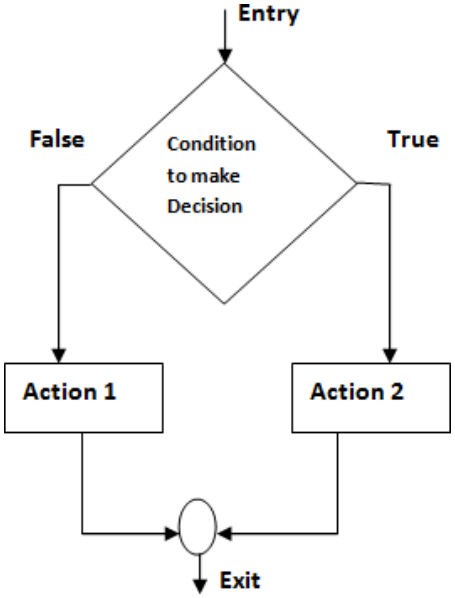
Add two numbers:

Step 1:
Start
Step 2:
get a,b
Step 3: calculate
c=a+b Step 4:
Display c
Step 5: Stop

Selection:

A selection statement causes the program control to be transferred to a specific part of the program based upon the condition.

If the conditional test is true, one part of the program will be executed, otherwise it will execute the other part of the program.



Example

Write an algorithm to check whether he is eligible to vote?

Step 1: Start
Step 2: Get age
Step 3: if age >= 18 print "Eligible to vote" **Step 4:** else print "Not eligible to vote" **Step 6:** Stop

Iteration:

In some programs, certain set of statements are executed again and again based upon conditional test. i.e. executed more than one time. This type of execution is called looping or iteration.

Example

Write an algorithm to print all natural numbers up to n

Step 1: Start
Step 2: get n value.
Step 3: initialize i=1
Step 4: if (i<=n) go to step 5 else go to step 7 **Step 5:** Print i value and increment i value by 1 **Step 6:** go to step 4
Step 7: Stop

PSEUDO CODE

- Pseudo code consists of short, readable and formally styled English languages used for explain an algorithm.
- It does not include details like variable declaration, subroutines.

- It is easier to understand for the programmer or non programmer to understand the general working of the program, because it is not based on any programming language.
- It gives us the sketch of the program before actual coding.
- It is not a machine readable
- Pseudo code can't be compiled and executed.
- There is no standard syntax for pseudo code.

Example: Greates of two numbers

```
BEGIN READ a,b
IF (a>b) THEN
DISPLAY a is greater ELSE
DISPLAY b is greater END IF
END
```

FLOWCHART

Flow chart is defined as graphical representation of the logic for problem solving. The purpose of flowchart is making the logic of the program clear in a visual representation.

Advantages of flowchart:

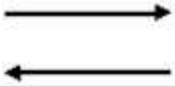
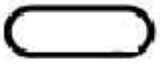
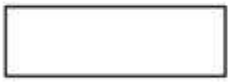
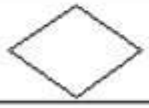
- Communication: - Flowcharts are better way of communicating the logic of a system to all concerned.
- Effective analysis: - With the help of flowchart, problem can be analyzed in more effective way.
- Proper documentation: - Program flowcharts serve as a good program documentation, which is needed for various purposes.
- Efficient Coding: - The flowcharts act as a guide or blueprint during the systems analysis and program development phase.
- Proper Debugging: - The flowchart helps in debugging process.
- Efficient Program Maintenance: - The maintenance of operating program becomes easy with the help of flowchart. It helps the programmer to put efforts more efficiently on that part.

Disadvantages of flow chart:

- Complex logic: - Sometimes, the program logic is quite complicated. In that case, flowchart becomes complex and clumsy.
- Alterations and Modifications: - If alterations are required the flowchart may require re-drawing completely.
- Reproduction: - As the flowchart symbols cannot be typed, reproduction of flowchart becomes a problem.
- Cost: For large application the time and cost of flowchart drawing becomes costly.

25. a. Discuss the symbols and rules for drawing flowchart with the example?

Symbols for drawing flowchart

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

Rules for drawing a flowchart

- 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.

(OR)

b. Discuss the various operation that can be performed on a tuple and Lists (minimum 5) with an example program

Lists

- List is an ordered sequence of items. Values in the list are called elements / items.
- It can be written as a list of comma-separated items (values) between **square brackets[ ]**.
- Items in the lists can be of different data types.

Creating a list	>>>**list1=["python", 7.79, 101, "hello"]** >>>**list2=["god",6.78,9]**	Creating the list with elements of different data types.
Indexing	>>>**print(list1[0])** python >>> **list1[2]** 101	❖ Accessing the item in the position 0 ❖ Accessing the item in the position 2
Slicing(ending position -1) Slice operator is used to extract part of a string, or some part of a list Python	>>> **print(list1[1:3])** [7.79, 101] >>>**print(list1[1:])** [7.79, 101, 'hello']	- Displaying items from 1st till 2nd. - Displaying items from 1st position till last.
Concatenation	>>>**print(list1+list2)** ['python', 7.79, 101, 'hello', 'god', 6.78, 9]	-Adding and printing the items of two lists.
Repetition	>>> **list2*3** **['god', 6.78, 9, 'god', 6.78, 9, 'god', 6.78, 9]**	Creates new strings, concatenating multiple copies of the same string
Updating the list	>>> **list1[2]=45** >>>print(list1) [‘python’, 7.79, 45, ‘hello’]	Updating the list using index value

Inserting an element	>>> **list1.insert(2,"program")**
	>>> print(list1)
	['python', 7.79, 'program', 45, 'hello']
		Inserting an element in 2nd position

Removing an element	>>> **list1.remove(45)**
	>>> print(list1)
	['python', 7.79, 'program', 'hello']
		Removing an element by giving the element directly

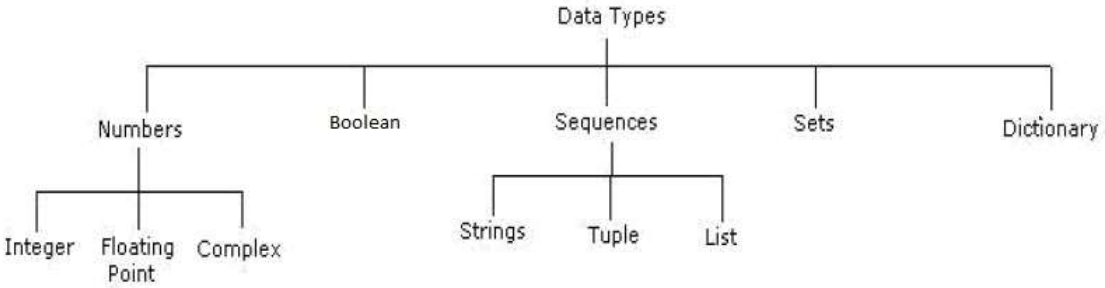
26. a. Explain in detail about various data types in Python with an example?

Data type:

Every value in Python has a data type.

It is a set of values, and the allowable operations on those values.

Python has four standard data types:



Numbers:

- Number data type stores **Numerical Values.**
- This data type is immutable [i.e. values/items cannot be changed].
- Python supports integers, floating point numbers and complex numbers. They are defined as,

Integers	Long	Float	Complex
- They are often called just integers or **int**. - They are positive or negative whole numbers with no decimal point.	-They are long integers. -They can also be represented in octal and hexadecimal representation.	-They are written with a decimal point dividing the integer and the fractional parts.	-They are of the form **a + bj**, where a and b are floats and j represents the square root of -1 (which is an imaginary number). -The real part of the number is a, and the imaginary part is b.
Eg, 56	**Eg, 5692431L**	**Eg, 56.778**	**Eg, square root of -1 is a complex number**

Sequence:

- A sequence is an **ordered collection of items**, indexed by positive integers.
- It is a combination **of mutable** (value can be changed) **and immutable** (values cannot be changed) data types.
- There are three types of sequence data type available in Python, they are
  1. **Strings**
  2. **Lists**
  3. **Tuples**

Strings:

- A String in Python consists of a series or sequence of characters - letters, numbers, and special characters.
- Strings are marked by quotes:

- single quotes (' ') Eg, 'This a string in single quotes'
- double quotes (" ") Eg, '"This a string in double quotes"'
- triple quotes(""" """) Eg, This is a paragraph. It is made up of multiple lines and sentences."""

➢ Individual character in a string is accessed using a subscript (index).

➢ Characters can be accessed using indexing and slicing operations

Strings are immutable i.e. the contents of the string cannot be changed after it is created.

Indexing:

String A	H	E	L	L	O
Positive Index	0	1	2	3	4
Negative Index	-5	-4	-3	-2	-1

- Positive indexing helps in accessing the string from the beginning
- Negative subscript helps in accessing the string from the end.
- Subscript 0 or –ve n(where n is length of the string) displays the first element.
  Example: A[0] or A[-5] will display "H"
- Subscript 1 or –ve (n-1) displays the second element.
  Example: A[1] or A[-4] will display "E"

Operations on string:

i. Indexing
ii. Slicing
iii. Concatenation
iv. Repetitions
v. Member ship

Creating a string	>>> s="good morning"	Creating the list with elements of different data types.

Indexing	>>> **print(s[2])**
	o
	>>> **print(s[6])**
	O
		❖ Accessing the item in the position 0
		❖ Accessing the item in the position 2

Slicing(ending position -1) Slice operator is used to extract part of a data type	>>> **print(s[2:])** od morning >>> **print(s[:4])** Good	- Displaying items from 2nd till last. - Displaying items from 1st position till 3rd .
Concatenation	>>>**print(s+"friends")** good morningfriends	-Adding and printing the characters of two strings.
Repetition	**>>>print(s*2)** good morninggood morning	Creates new strings, concatenating multiple copies of the same string
in, not in (membership operator)	>>> s="good morning" **>>>"m" in s True >>> "a" not in s True**	Using membership operators to check a particular character is in string or not. Returns true if present.

Lists

- List is an ordered sequence of items. Values in the list are called elements / items.
- It can be written as a list of comma-separated items (values) between **square brackets[ ]**.
- Items in the lists can be of different data types.

Operations on list:

Indexing
Slicing
Concatenation
Repetitions
Updation, Insertion, Deletion

Creating a list	>>>**list1=["python", 7.79, 101, "hello"]** >>>**list2=["god",6.78,9]**	Creating the list with elements of different data types.
Indexing	>>>**print(list1[0])** python >>> **list1[2]** 101	❖ Accessing the item in the position 0 ❖ Accessing the item in the position 2

Slicing(ending position -1) Slice operator is used to extract part of a string, or some part of a list Python	>>> **print(list1[1:3])** [7.79, 101] >>>**print(list1[1:])** [7.79, 101, 'hello']	- Displaying items from 1st till 2nd. - Displaying items from 1st position till last.
Concatenation	>>>**print(list1+list2)** ['python', 7.79, 101, 'hello', 'god', 6.78, 9]	-Adding and printing the items of two lists.
Repetition	>>> **list2*3** **['god', 6.78, 9, 'god', 6.78, 9, 'god', 6.78, 9]**	Creates new strings, concatenating multiple copies of the same string
Updating the list	>>> **list1[2]=45** >>>print(list1) ['python', 7.79, 45, 'hello']	Updating the list using index value
Inserting an element	>>> **list1.insert(2,"program")** >>> print(list1) ['python', 7.79, 'program', 45, 'hello']	Inserting an element in 2nd position
Removing an element	>>> **list1.remove(45)** >>> print(list1) ['python', 7.79, 'program', 'hello']	Removing an element by giving the element directly

Tuple:

- A tuple is same as list, except that the set of elements is **enclosed in parentheses** instead of square brackets.
- **A tuple is an immutable list.** i.e. once a tuple has been created, you can't add elements to a tuple or remove elements from the tuple.
- Benefit of Tuple:
- Tuples are faster than lists.
- If the user wants to protect the data from accidental changes, tuple can be used.
- Tuples can be used as keys in dictionaries, while lists can't.

Basic Operations:

Creating a tuple	>>>**t=("python", 7.79, 101, "hello")**	Creating the tuple with elements of different data types.

Indexing	>>>**print(t[0])** python >>> **t[2]** 101	❖ Accessing the item in the position 0 ❖ Accessing the item in the position 2
Slicing(ending position -1)	>>>**print(t[1:3])** (7.79, 101)	❖ Displaying items from 1st till 2nd.
Concatenation	>>> **t+("ram", 67)** ('python', 7.79, 101, 'hello', 'ram', 67)	❖ Adding tuple elements at the end of another tuple elements
Repetition	>>>**print(t*2)** ('python', 7.79, 101, 'hello', 'python', 7.79, 101, 'hello')	❖ Creates new strings, concatenating multiple copies of the same string

(OR)

b. Explain the different types of operators in python with an example.

OPERATORS

- Operators are the constructs which can manipulate the value of operands.
- Consider the **expression *4 + 5 = 9.*** Here, **4 and 5 are called operands** and **+ is called operator**
- Types of Operators:

-Python language supports the following types of operators

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

Arithmetic operators:

They are used to perform **mathematical operations** like addition, subtraction, multiplication etc. **Assume, a=10 and b=5**

Operator	Description	Example
+ Addition	Adds values on either side of the operator.	a + b = 30

- Subtraction	Subtracts right hand operand from left hand operand.	a – b = -10
* Multiplication	Multiplies values on either side of the operator	a * b = 200
/ Division	Divides left hand operand by right hand operand	b / a = 2
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 0
** Exponent	Performs exponential (power) calculation on operators	a**b =10 to the power 20
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed	5//2=2

Examples

```
a=10
b=5 print("a+b=",a+b)
print("a-b=",a-b)
print("a*b=",a*b)
print("a/b=",a/b)
print("a%b=",a%b)
print("a//b=",a//b)
print("a**b=",a**b)
```

Output:

```
a+b= 15
a-b= 5
a*b= 50
a/b= 2.0
a%b= 0
a//b= 2
a**b= 100000
```

Comparison (Relational) Operators:

- Comparison operators are used to compare values.
- It either returns True or False according to the condition. **Assume, a=10 and b=5**

Operator	Description	Example
==	If the values of two operands are equal, then the condition	(a == b) is
	becomes true.	not true.

!=	If values of two operands are not equal, then condition becomes true.	(a!=b) is true
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

Example

```
a=10
b=5 print("a>b=>",a>b)
print("a>b=>",a<b)
print("a==b=>",a==b)
print("a!=b=>",a!=b)
print("a>=b=>",a<=b)
print("a>=b=>",a>=b)
```

Output:
```
a>b=> True
a>b=> False
a==b=> False
a!=b=> True
a>=b=> False
a>=b=> True
```

Assignment Operators:

-Assignment operators are used in Python to assign values to variables.

Operator	Description	Example
=	Assigns values from right side operands to left side operand	c = a + b assigns value of a + b into c
+= Add AND	It adds right operand to the left operand and assign the result to left operand	c += a is equivalent to c = c + a

-= Subtract AND	It subtracts right operand from the left operand and assign the result to left operand	c -= a is equivalent to c = c - a
*= Multiply AND	It multiplies right operand with the left operand and assign the result to left operand	c *= a is equivalent to c = c * a
/= Divide AND	It divides left operand with the right operand and assign the result to left operand	c /= a is equivalent to c = c / ac /= a is equivalent to c = c / a
%= Modulus AND	It takes modulus using two operands and assign the result to left operand	c %= a is equivalent to c = c % a
**= Exponent AND	Performs exponential (power) calculation on operators and assign value to the left operand	c **= a is equivalent to c = c ** a
//= Floor Division	It performs floor division on operators and assign value to the left operand	c //= a is equivalent to c = c // a

Example

```
a = 21
b = 10
c = 0
c = a + b
print("Line 1 - Value of c is ", c) c +=
a
print("Line 2 - Value of c is ", c) c *=
a
print("Line 3 - Value of c is ", c) c /=
a
print("Line 4 - Value of c is ", c) c = 2
c %= a
print("Line 5 - Value of c is ", c) c
**= a
print("Line 6 - Value of c is ", c) c //=
a
print("Line 7 - Value of c is ", c)
```

Output

Line 1 - Value of c is 31 Line 2 - Value of c is 52 Line 3 - Value of c is 1092 Line 4 - Value of c is 52.0 Line 5 - Value of c is 2 Line 6 - Value of c is 2097152 Line 7 - Value of c is 99864

Logical Operators:

-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

Example a = True b = False

```
print('a and b is',a and b)
print('a or b is',a or b) print('not a
is',not a)
```

Output

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

Bitwise Operators:

- A **bitwise operation** operates on one or more **bit** patterns at the level of individual bits

Example: Let x = 10 (0000 1010 in binary) and y = 4 (0000 0100 in binary)

Operator	Meaning	Example
&	Bitwise AND	x& y = 0 (0000 0000)
|	Bitwise OR	x | y = 14 (0000 1110)
~	Bitwise NOT	~x = -11 (1111 0101)
^	Bitwise XOR	x ^ y = 14 (0000 1110)
>>	Bitwise right shift	x>> 2 = 2 (0000 0010)
<<	Bitwise left shift	x<< 2 = 40 (0010 1000)

Example

```
a = 60            # 60 = 0011 1100
b = 13            # 13 = 0000 1101
c = 0
c = a & b;        # 12 = 0000 1100
print "Line 1 - Value of c is ", c c = a
| b;              # 61 = 0011 1101
print "Line 2 - Value of c is ", c c = a
^ b;              # 49 = 0011 0001
print "Line 3 - Value of c is ", c
c = ~a;           # -61 = 1100 0011
print "Line 4 - Value of c is ", c
c = a << 2;       # 240 = 1111 0000
print "Line 5 - Value of c is ", c
c = a >> 2;       # 15 = 0000 1111
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
```

Membership Operators:

- Evaluates to find a value or a variable is in the specified sequence of string, list, tuple, dictionary or not.
- Let, **x=[5,3,6,4,1].** To check particular item in list or not**, in and not in** operators are used.

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

Example:

x=[5,3,6,4,1]

>>> **5 in x**

True

>>> **5 not in x**

False

Identity Operators:

❖ They are used to check if two values (or variables) are located on the same partof the memory.

Operator	Meaning	Example
is	True if the operands are identical (refer to the same object)	x is True
is not	True if the operands are not identical (do not refer to the same object)	x is not True

Example

```
x = 5
y = 5
x2 = 'Hello' y2
= 'Hello'
print(x1 is not y1)
print(x2 is y2)
```

Output

False True

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed to be University)
(Established Under Section 3 of UGC Act 1956)
COIMBATORE – 641 021

INFORMATION TECHNOLOGY
Third Semester
SECOND INTERNAL EXAMINATION - August 2019

PROGRAMMING IN PYTHON

Class & Section: II B.Sc IT **Duration: 2 hours**
Date & Session: 29.8.19 (AN) **Maximum marks: 50 marks**
Sub.Code: I8ITU304B

PART- A (20 * 1= 20 Marks)
Answer ALL the Questions

1. Which predefined Python function is used to find length of string?
   a. length() b. len() c. strlen() d. stringlength()
2. Syntax of constructor in Python?
   a. def __init__() b. def _init_() c. _init_() d. All options
3. How to find the last element of list in Python? Assume `bikes` is the name of list.
   a. bikes[0] b. bikes[-1] c. bikes[lpos] d. bikes[:-1]
4. If a='cpp', b='buzz' then which of the following operation would show 'cppbuzz' as output?
   a. a+b b. a+"+b c. a+""+b d. All the options
5. What datatype is the object below ?
   a. list b. dictionary c. array d. tuple
6. What is the data type of X in X = [12.12, 13, 'cppbuzz']
   a. Tuple b. Array c. List d. Dictionary
7. What is the output of the expression? round(4.5676,2)?
   a. 4.5 b. 4.6 c. 4.57 d. 4.56
8. What is the output of the function shown below?
   a. f b. oxF c. oXf d. oxf
9. What is the output of the following piece of code? a={1:"A",2:"B",3:"C"} print(a.get(1,4))
   a. 1 b. 4 c. A d. Invalid syntax
10. What is the data type of X in X = [12.12, 13, 'cppbuzz']
    a. Tuple b. Array c. List d. Dictionary
11. What is the output of the expression? round(4.5676,2)?
    a. 4.5 b. 4.6 c. 4.57 d. 4.56
12. What is the output of the function shown below?
    a. f b. oxF c. oXf d. oxf
13. What is the output of the following? print([i.lower() for i in "HELLO"])
    a. [‘h’, ‘e’, ‘l’, ‘l’, ‘o’] c. [‘hello’]
    b. ‘hello’ d. hello
14. The format function, when applied on a string returns :
    a. list b. bool c. int d. str
15. Select the command to Find and print Data types using the Type command. name="Hello World"
    a. type(name) c. print(name)
    b. print(type(name)) d. type()
16. what is the output for: name="Hello World" print(type(name))
    a. Hello World b. hello World c. <class 'str'> d. str

17. What data type is the object below? L = [1, 23, 'hello', 1]
    a. List b. Dictionary c. Tuple d. Array
18. Which of the following function convert a string to a float in python?
    a. int(x [,base]) b. long(x [,base]) c. float(x) d. str(x)
19. How many keyword arguments can be passed to a function in a single function call?
    a. zero b. one c. zero or more d. one or more
20. Suppose list1 = [0.5 * x for x in range(0, 4)], list1 is :
    a. [0, 1, 2, 3]
    b. [0, 1, 2, 3, 4]
    c. [0.0, 0.5, 1.0, 1.5]
    d. [0.0, 0.5, 1.0, 1.5, 2.0]

PART B (3 * 2 = 6 Marks)
Answer ALL the Questions

21. Write the syntax for while loop with flowchart.
22. List the operations on strings.
23. What is meant by parameter and list down its type?

PART C (3 * 8 = 24 Marks)
Answer ALL the Questions

24. a. Discuss about the various type of arguments in python.

(OR)

    b. Briefly discuss in detail about function prototyping in python with suitable example program
25. a. Write a program to search an element using linear search.

(OR)

    b. Discuss with an example about function composition
26. a. Explain conditional statements in detail with example (if, if..else, if..elif..else)

(OR)

    b. Explain in detail about iterations with example (for, while)

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed to be University)
(Established Under Section 3 of UGC Act 1956)
COIMBATORE – 641 021

INFORMATION TECHNOLOGY
Third Semester
SECOND INTERNAL EXAMINATION - August 2019

PROGRAMMING IN PYTHON

Class & Section: II B.Sc IT **Duration: 2 hours**
Date & Session: 29.8.19 (AN) **Maximum marks: 50 marks**
Sub.Code: I8ITU304B

PART- A (20 * 1= 20 Marks)
Answer ALL the Questions

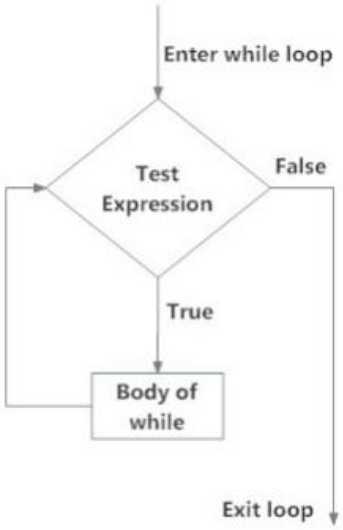
1. len()
2. def __init__()
3. bikes[-1]
4. All the options
5. list
6. List
7. 4.57
8. oXf
9. A
10. List
11. 4.57
12. oXf
13. [‘h’, ‘e’, ‘l’, ‘l’, ‘o’]
14. Str
15. print(type(name))
16. <class 'str'>
17. List
18. float(x)
19. zero or more
20. [0.0, 0.5, 1.0, 1.5]

PART B (3 * 2 = 6 Marks)
Answer ALL the Questions

21. Write the syntax for while loop with flowchart.

Syntax

```
while  test_expression:
      Body of while
```



22. List the operations on strings.

Operations on string:

1. Indexing
2. Slicing
3. Concatenation
4. Repetitions
5. Member ship

23. What is meant by parameter and list down its type?

Parameters are the variables which used in the function definition. Parameters are inputs to functions. Parameter receives the input from the function call. It is possible to define more than one parameter in the function definition.

Types of parameters/Arguments:

1. Required/Positional parameters
2. Keyword parameters
3. Default parameters
4. Variable length parameters

PART C (3 * 8 = 24 Marks)
Answer ALL the Questions

24. a. Discuss about the various type of arguments in python.

Parameters are the variables which used in the function definition. Parameters are inputs to functions. Parameter receives the input from the function call. It is possible to define more than one parameter in the function definition.

Types of parameters/Arguments:

1. Required/Positional parameters
2. Keyword parameters

3. Default parameters
4. Variable length parameters

Required/ Positional Parameter:
The number of parameter in the function definition should match exactly with number of arguments in the function call.

Example Output:

```
def student( name, roll ):
print(name,roll)
student(“George”,98)
```

o/p: George 98

Keyword parameter:
When we call a function with some values, these values get assigned to the parameter according to their position. When we call functions in keyword parameter, the order of the arguments can be changed.

Example Output:

```
def student(name,roll,mark):
print(name,roll,mark)
student(90,102,"bala")
```

o/p: 90 102 bala

Default parameter:
Python allows function parameter to have default values; if the function is called without the argument, the argument gets its default value in function definition.

Example Output:

```
def student( name, age=17):
print (name, age)
student( “kumar”):
student( “ajay”):
```

o/p: Kumar 17
Ajay 17

Variable length parameter
❖ Sometimes, we do not know in advance the number of arguments that will be passed into a function.
❖ Python allows us to handle this kind of situation through function calls with number of arguments.
❖ In the function definition we use an asterisk (*) before the parameter name to denote this is variable length of parameter.

Example Output:

```
def student( name,*mark):
print(name,mark)
student (“bala”,102,90)
```

o/p: bala (102 ,90)

(OR)

b. Briefly discuss in detail about function prototyping in python with suitable example program

i. Function without arguments and without return type

ii. Function with arguments and without return type
iii. Function without arguments and with return type
iv. Function with arguments and with return type

i) Function without arguments and without return type

- In this type no argument is passed through the function call and no output is return to main function
- The sub function will read the input values perform the operation and print the result in the same block

ii) Function with arguments and without return type

- Arguments are passed through the function call but output is not return to the main function

iii) Function without arguments and with return type

- In this type no argument is passed through the function call but output is return to the main function.

iv) Function with arguments and with return type

In this type arguments are passed through the function call and output is return to the main function

<table>
<tr><th colspan="2">Without Return Type</th></tr>
<tr><th>Without argument</th><th>With argument</th></tr>
<tr><td>def add():
a=int(input("enter a"))
b=int(input("enter b"))
c=a+b
print(c)
add()</td><td>def add(a,b):
c=a+b
print(c)
a=int(input("enter a"))
b=int(input("enter b")) add(a,b)</td></tr>
<tr><td>OUTPUT:
enter a 5
enter b 10
15</td><td>OUTPUT:
enter a 5
enter b 10
15</td></tr>
<tr><th colspan="2">With return type</th></tr>
<tr><th>Without argument</th><th>With argument</th></tr>
<tr><td>def add():
a=int(input("enter a"))
b=int(input("enter b"))
c=a+b
return c
c=add()
print(c)</td><td>def add(a,b):
c=a+b return c
a=int(input("enter a")) b=int(input("enter b")) c=add(a,b)
print(c)</td></tr>
<tr><td>OUTPUT:
enter a 5</td><td>OUTPUT:
enter a 5</td></tr>
</table>

enter b 10 15	enter b 10 15

25. a. Write a program to search an element using linear search.

Program

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

Output:

```
Enter a number: 8
Enter n numbers
8 4 7 5 6 3 9 2
Enter a key to be searched: 9
Key found
>>>
Enter a number: 8
Enter n numbers
8 4 7 5 6 3 9 2
Enter a key to be searched: 11
Key not found
```

(OR)

b. Discuss with an example about function composition

Function Composition:

❖ Function Composition is the ability to call one function from within another function

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

❖ In other words the output of one function is given as the input of another function is known as function composition.

Example:	**Output:**
math.sqrt(math.log(10))	

def add(a,b):
  c=a+b
  return c
def mul(c,d):
  e=c*d
  return e
c=add(10,20)
e=mul(c,30)
print(e)
	900

find sum and average using function composition	**output**

def sum(a,b):
  sum=a+b
  return sum
def avg(sum):
  avg=sum/2
  return avg
a=eval(input("enter a:"))
b=eval(input("enter b:"))
sum=sum(a,b)
avg=avg(sum)
	enter a:4
	enter b:8
	the avg is 6.0

print("the avg is",avg)	

26. a. Explain conditional statements in detail with example (if, if..else, if..elif..else)

- Decision structures evaluate multiple expressions, which produce TRUE or FALSE as the outcome
- Python programming language provides the following types of decision-making statements.
    1. If-statement
    2. If-else statement
    3. If-elif-else
    4. Nested if-elif-else

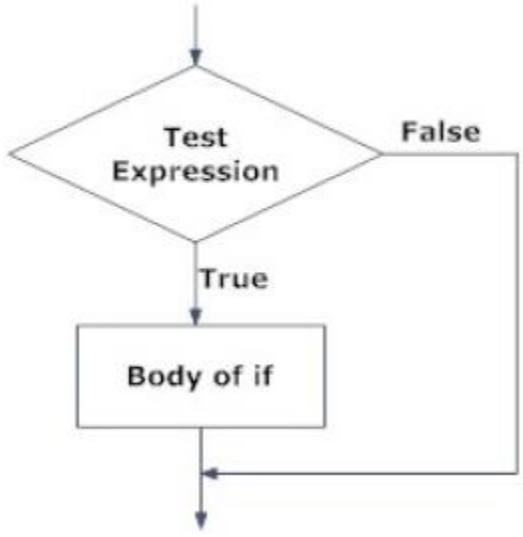
Statement	Description
if statements	An if statement consists of a Boolean expression followed by one or more statements.
if...else statements	An if statement can be followed by an optional else statement, which executes when the boolean expression is FALSE.
nested if statements	You can use one if or else if statement inside another if or else if statement(s).

1. **Conditional if Statement**
   - If condition (expression) is **TRUE**, then the block of statement(s) inside the 'if' statement is executed.
   - If condition is **FALSE**, the statement is not executed.

Syntax

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

Flow chart



Example

```
a = 200
b = 100
if a > b:
    print("a is greater")
```

Output:

a is greater

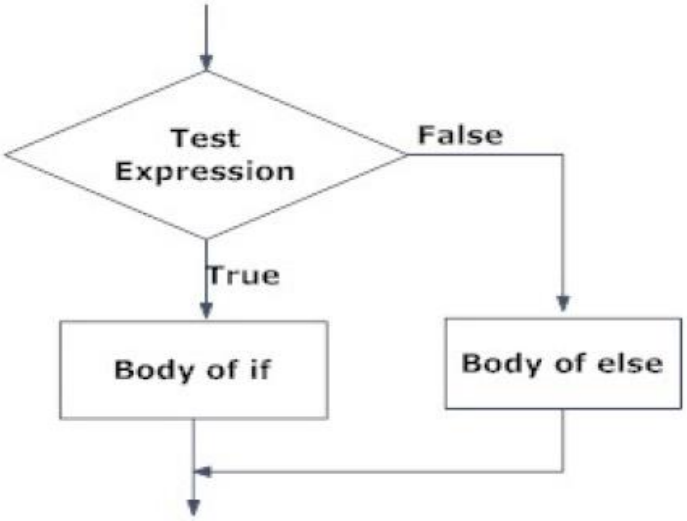
2. **Alternative if-else Statement**
   - If condition is **TRUE**, then the block of statement(s) inside the 'if' statement is executed.

- If condition is **FALSE**, the statement(s) inside else block is executed.

Syntax

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

Flow chart



Example

```
a = 200
b = 100
if a > b:
     print("a is greater")
else:
     print("b is greater")
```

Output:

a is greater

3. Chained conditional if-elif-else Statement

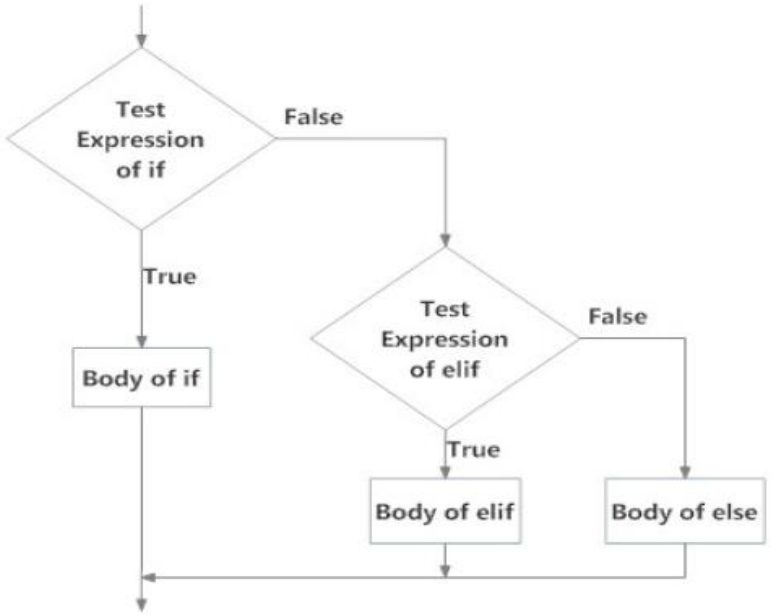
- An elif(else if) statement can be used to check ***multiple expressions***
- An if-elif-else structure first checks 'if condition'
- If condition is **TRUE**, execute the statements in the 'if ' block
- If condition is **FALSE**, it tests the condition in the elif block
- If elif statement is true, execute the statements in the elif block
- otherwise control passes to the else block

Syntax

```
if condition:

Statement(s)
elif condition:
```

Flow chart



Example

```
n = 50
if n > 50:
    print("Passed")
elif n == 50:
    print("Passed")
else:
    print("Failed")
```

Output:

Passed

4. Nested if-elif-else Statement

- Used to check another condition after the first condition has been evaluated as true.

Syntax

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

Example

```
n = 5
if  n > =0:
    if  n ==0:
        print("zero")
    else:
        print("Positive number")
else:
    print("Negative Number")
```

Output:

Positive number

(OR)

b. Explain in detail about iterations with example (for, while)

- Looping executes sequence of statement again and again until specific condition satisfies.
- Python programming language provides the following types of loops
    1. for loop
    2. While loop

Loop Type	Description
while loop	Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.
for loop	Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.

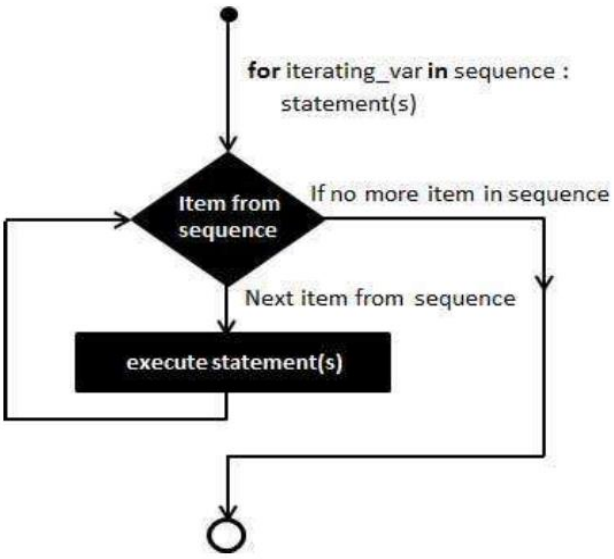
1. For Loops

- The 'for loop' repeats a given block of codes by specified number of times

Syntax

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

Flow chart



Example

```
fruits = ['Apple', 'Orange', 'Mango',
'Cherry']
for  i  in  fruits:
     print(i)
```

Output:

```
Apple
Orange
Mango
```

range() function

- Generate a sequence of numbers using range() function.
- **range(10)** will generate numbers from 0 to 9 (10 numbers).
- **Syntax**

  range(start, end, step)

- The function list() is used to force this function to output all the items

Example

```
>>>range(5)
Output:
    [0, 1, 2, 3, 4]
>>>range(3, 10)
>>>list(range(3,10))
Output:
    [3, 4, 5, 6, 7, 8, 9]
>>>range(4, 10, 2)
>>>list(range(4, 10, 2))
Output:
    [4, 6, 8]
```

for loop with else

- For loop can have an optional else block.
- The else part is executed if the items in the sequence used in for loop exhausts.

Example

```
numbers = [10, 99, 3]
for   i   in   numbers:
      if  i % 2 == 0:
            print(i, "is an even number")
      else:
           print(i, "is an odd numbers")
Output:
  10 is an even number
  99 is an odd number
   3 is an odd number
```

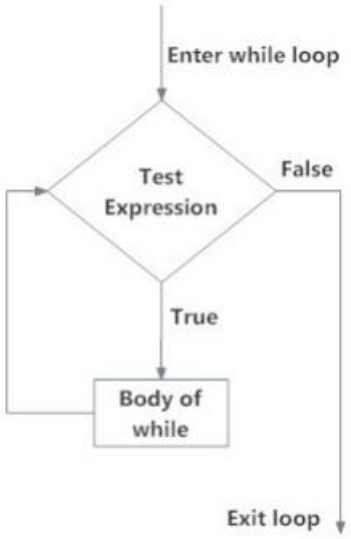
2. While Loop

- It is also called as ***Entry-Controlled Loop.***
- The while loop in Python is used to iterate over a block of code as long as the test expression (condition) is true.

Syntax

```
while  test_expression:
      Body of while
```

Flow chart



Example

```
Count = 1
While (count < =3):
    print("Python Programming")
    Count = count + 1
```

Output:

```
Python Programming
Python Programming
Python Programming
```

While loop with else

- While loop can have an optional else block.
- The else part is executed when the condition becomes false.

Example

```
Count = 1
While (count < =3):
    print("Python Programming")
    Count = count  + 1
else:
    print("Exit")
```

Output:

```
Python Programming
Python Programming
Python Programming
Exit
```

NESTED LOOPS

- Placing of one loop inside the body of another loop is called ***nested loop.***

Nested for loop

Syntax:

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

Nested while loop

Syntax:

```
while  test_expression:
      while  test_expression:
           statements(s)
    statements(s)
```