GE8151 PROBLEM SOLVING AND PYTHON PROGRAMMING
QUESTION BANK

UNIT I
ALGORITHMIC PROBLEM SOLVING
PART- A (2 Marks)

1. **What is an algorithm?**

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

2. **Write an algorithm to find minimum of 3 numbers in a list.**

   ALGORITHM : Find Minimum of 3 numbers in a list

   Step 1: Start
   Step 2: Read the three numbers A, B, C
   Step 3: Compare A and B.
   If A is minimum, go to step 4 else go to step 5.
   Step 4: Compare A and C.
   If A is minimum, output "A is minimum" else output "C is minimum". Go to step 6.
   Step 5: Compare B and C.
   If B is minimum, output "B is minimum" else output "C is minimum".
   Step 6: Stop

3. **List the building blocks of an algorithm.**

   The building blocks of an algorithm are
   - Statements
   - Sequence
   - Selection or Conditional
   - Repetition or Control flow
   - Functions

4. **Define statement. List its types.**

   Statements are instructions in Python designed as components for algorithmic problem solving, rather than as one-to-one translations of the underlying machine language instruction set of the computer.

   There are three types of high-level programming language statements Input/output statements make up one type of statement. An input statement collects a specific value from the user for a variable within the program. An output statement writes a message or the value of a program variable to the user's screen.

5. **Write the pseudo code to calculate the sum and product of two numbers and display it.**

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

6. How does flow of control work?

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

7. What is a function?

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

8. Write the pseudo code to calculate the sum and product displaying the answer on the monitor screen.

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

9. Give the rules for writing Pseudo codes.

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

10. Give the difference between flowchart and pseudo code.

Flowchart and Pseudo code are used to document and represent the algorithm. In other words, an algorithm can be represented using a flowchart or a pseudo code. Flowchart is a graphical representation of the algorithm. Pseudo code is a readable, formally styled English like language representation of the algorithm.

11. Define a flowchart.

A flowchart is a diagrammatic representation of the logic for solving a task. A flowchart is drawn using boxes of different shapes with lines connecting them to show the flow of control. The purpose of drawing a flowchart is to make the logic of the program clearer in a visual form.

12. Give an example of iteration.

```
a = 0
for i from 1 to 3        // loop three times
{
  a = a + i              // add the current value of i to a
}
print a                  // the number 6 is printed (0 + 1; 1 + 2; 3 + 3)
```

13. Write down the rules for preparing a flowchart.

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

(1) A flowchart should have a start and end,

(2) The direction of flow in a flowchart must be from top to bottom and left to right, and

(3) The relevant symbols must be used while drawing a flowchart.

14. List the categories of Programming languages.

Programming languages are divided into the following categories:
Interpreted, Functional, Compiled, Procedural, Scripting, Markup, Logic-Based, Concurrent and Object-Oriented Programming Languages

15. Mention the characteristics of an algorithm.

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

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

Machine language is a collection of binary digits or bits that the computer reads and interprets. This language is not easily understandable by the human.

An assembly language directly controls the physical hardware. A program written in assembly language consists of a series of instructions mnemonics that correspond to a stream of executable instructions, when translated by an assembler can be loaded into memory and executed. The programs written in this language are not portable and the debugging process is also not very easy.

A high level language is much more abstract, that must be translated or compiled in to machine language. It is easily understandable and the programs are portable. Debugging the code is easy and the program written is not machine dependent.

17. What is the difference between algorithm and pseudo code?

An algorithm is a systematic logical approach used to solve problems in a computer while pseudo code is the statement in plain English that may be translated later to a programming language. Pseudo code is the intermediary between algorithm and program.

18. List out the simple steps to develop an algorithm.

Algorithm development process consists of five major steps.

Step 1: Obtain a description of the problem.
Step 2: Analyze the problem.
Step 3: Develop a high-level algorithm.
Step 4: Refine the algorithm by adding more detail.
Step 5: Review the algorithm.

19.Give the differences between recursion and iteration.

Recursion	Iteration
Function calls itself until the base condition is reached.	Repetition of process until the condition fails.
Only base condition (terminating condition) is specified.	It involves four steps: initialization, condition, execution and updation.
It keeps our code short and simple.	Iterative approach makes our code longer.
It is slower than iteration due to overhead of maintaining stack.	Iteration is faster.
It takes more memory than iteration due to overhead of maintaining stack.	Iteration takes less memory.

20. What are advantages and disadvantages of recursion?

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

PART B (16 MARKS)

1. What are the building blocks of an algorithm? Explain in detail.

The building blocks of algorithm are

-Statements
-State
-Control flow
-Functions

Statements:

There are 3 types of statements:

-Input/Output Statement
-Assignment Statement
-Control Statement

State:

There are 3 types of state:

-Initial state
-Current state
-Final state

Control flow:

-Sequence

The sequence structure is the construct where one statement is executed after another

-Selection

The selection structure is the construct where statements can executed or skipped depending on whether a condition evaluates to TRUE or FALSE. There are three selection structures in C:

1. IF
2. IF – ELSE
3. SWITCH

-Repetition

The repetition structure is the construct where statements can be executed repeatedly until a condition evaluates to TRUE or FALSE. There are two repetition structures in C:

1. WHILE
2. FOR

Functions:

A function is a block of organized reusable code that is used to perform a single action.

2. Briefly describe iteration and recursion. Illustrate with an example.

ITERATION:

S.no.	Iteration	Recursion
1	The process is repeated until the condition fails.	The function calls itself until the base condition is satisfied.
2	It consumes less memory.	It consumes more memory
3	It is faster	It is slower
4	The code is long .	The code is short.
5	Tracing is easier if any problem occurs.	Tracing is difficult if any problem occurs.

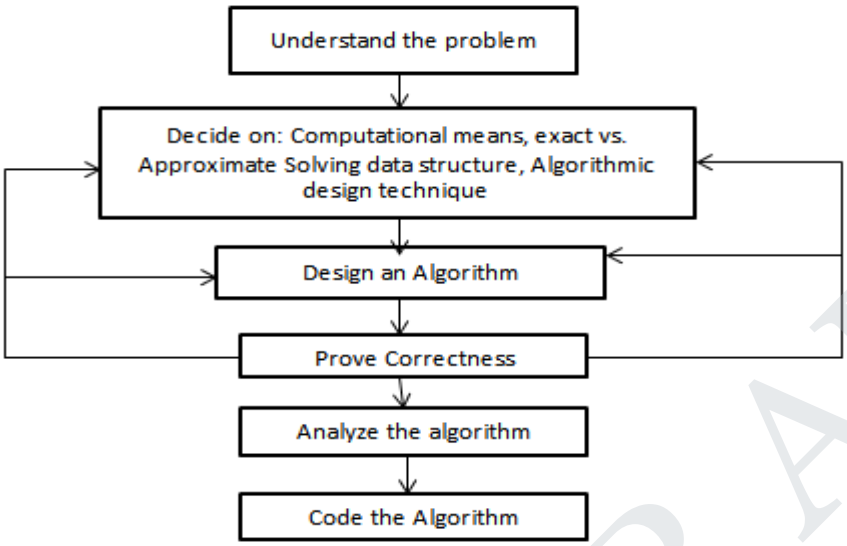
Example: **Iterative algorithm for factorial of a number**

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

factorial(n)
Step 1: Initialize f=1,i=1
Step 2: Repeat step 2.1 and 2.2 until i<=n
         Step 2.1: f= f*i
         Step 2.2: Increment i by 1  (i=i+1)
Step 3: Return f
```

Example: Recursive algorithm for factorial of number

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

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

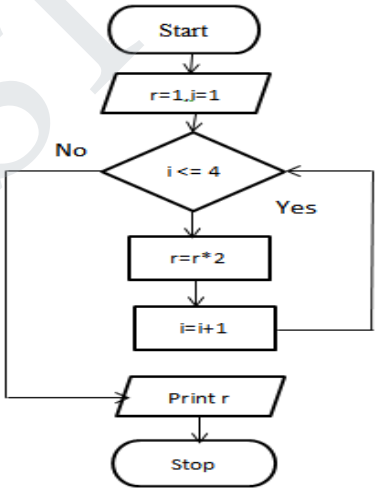
3. Explain in detail Algorithmic problem solving.



4. Write an algorithm and draw a flowchart to calculate 2^4.

Algorithm:
Step 1: Start
Step 2: Initialize the value of result, r=1.
Step 3: Repeat step4 for 4 times
Step 4: calculate r=r*2
Step 5: Print the value of **r**
Step 6: Stop

Flowchart:



5. a) Describe pseudo code with its guidelines.

Pseudo code consists of short, readable and formally-styled English language used for explaining an algorithm. Pseudo code does not include details like variable declarations, subroutines etc.

Preparing a Pseudo Code

- Pseudo code is written using structured English.
- In a pseudo code, some terms are commonly used to represent the various actions.
  For example,
  for inputting data the terms may be (INPUT, GET, READ),
  for outputting data (OUTPUT, PRINT, DISPLAY),
  for calculations (COMPUTE, CALCULATE),
  for incrementing (INCREMENT),
  in addition to words like ADD, SUBTRACT, INITIALIZE used for addition, subtraction, and initialization, respectively.
- The control structures—sequence, selection, and iteration are also used while writing the pseudo code.
- The sequence structure is simply a sequence of steps to be executed in linear order.
  The selection constructs—if statement and case statement. In the if-statement, if the condition is true then the THEN part is executed otherwise the ELSE part is executed. There can be variations of the if-statement also, like there may not be any ELSE part or there may be nested ifs. The case statement is used where there are a number of conditions to be checked. In a case statement, depending on the value of the expression, one of the conditions is true, for which the corresponding statements are executed. If no match for the expression occurs, then the OTHERS option which is also the default option, is executed.
- WHILE and FOR are the two iterative statements.

b) Give an example for pseudo code.

Pseudocode for finding maximum in a list:

```
BEGIN
SET numlist=[ ]
GET n
FOR i=1 to n
        GET numlist elements
ENDFOR
SET maximum = numlist[0]
FOR i in numlist
        IF (n > maximum)
                maximum = n
        ENDIF
ENDFOR
PRINT maximum
END
```

c) Write the pseudo code for Towers of Hanoi.

Pseudocode

```
START
Procedure Hanoi(disk, source, dest, aux)
```

```
IF disk = = 0, THEN
        move disk from source to dest
ELSE
        Hanoi(disk - 1, source, aux, dest)
         move disk from source to dest
        Hanoi(disk - 1, aux, dest, source)
END IF
END Procedure
```

6. a) What is flowchart?

Flowchart is a diagrammatic representation of the logic for solving a task. A flowchart is drawn using boxes of different shapes with lines connecting them to show the flow of control. The purpose of drawing a flowchart is to make the logic of the program clearer in a visual form.

b) List down symbols and rules for writing flowchart.

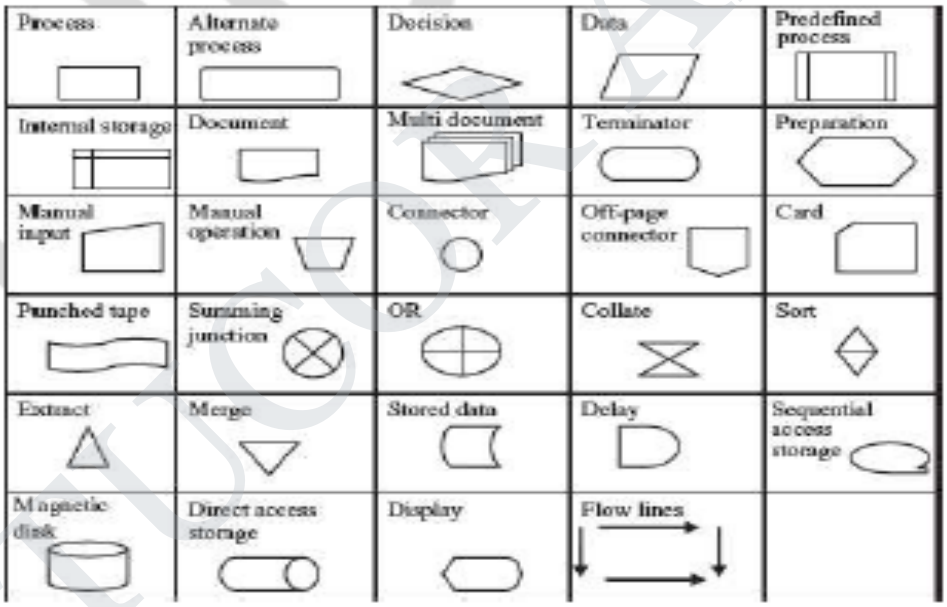
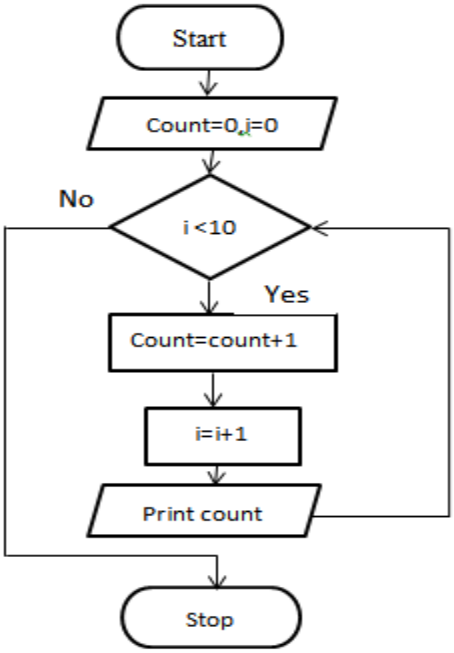


Figure : Flow chart Symbols

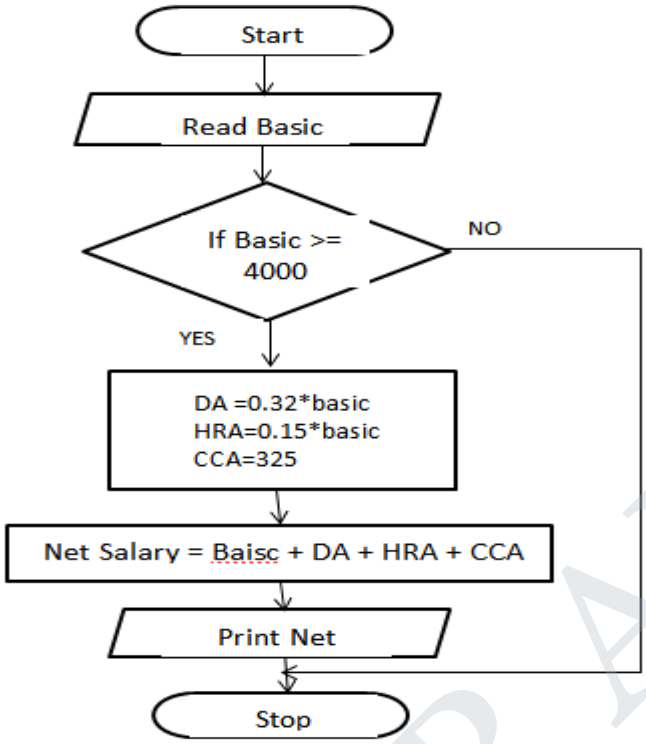
c) Draw a flowchart to count and print from1 to 10.



7. a) Write an algorithm and give the flowchart to find the net salary of an employee.

Algorithm:

Step 1: Start
Step 2 : Read the basic salary
Step 3 : IF the basic is greater than or equal to 4000 ELSE Goto Step 4
Step 3.1 : DA= 0.32 * basic (Dearness Allowance)
Step 3,2 : HRA = 0.15 * basic (House Rent Allowance)
Step 3.3 : CCA = 325 (City Compensatory Allowance)
Step 3.4 : Net Salary basic + DA HRA + CCA
Step 4 : Print the Net Salary
Step 5 : Stop



b) Write an algorithm and give the pseudo code to guess an integer number in a range.

Algorithm:
step 1: Start the program
step 2: Read an 'n' number
step 3: Read an Guess number
step 4: if Guess> n; print "Your Guess too high" Step 5: elif Guess <n ; print "Your Guess too low" step 6: elif Guess = = n; print "Good job"
Step 7: else print"Nope "
Step :8 Stop the program

Pseudocode:

```
BEGIN
READ n
READ Guess = 20
IF Guess> n
        print"Your Guess too High" elif Guess< n
        print "Your Guess too low" elif Guess = = 20
        print "Good Job"
ELSE
print"Nope"
```

8. a) Write an algorithm to insert a card in a list of sorted cards.

ALGORITHM:
Step 1: Start
Step 2: Declare the variables N, List[],I and X

Step 3: READ Number of element in sorted list as N
Step 4: SET i=0
Step 5: IF i<N THEN go to step 6 ELSE go to step 9
Step 6: READ Sorted list element as List[i]
Step 7: i=i+1
Step 8: go to step 5
Step 9: READ Element to be insert as X
Step 10: SET i=N-1
Step 11: IF i>0 AND X<List[i] THEN go to step 12 ELSE go to step 15
Step 13: i=i-1
Step 14: go to step 11
Step 15: List[i+1]=X
Step 16: Stop

b) Write an algorithm to find the minimum number in a list.

Algorithm:

Step 1 : Start
Step 2 : Initialize the value of minimum = 0
Step 3 : Enter the input number (n) of items in a list.
Step 4 : Get all the elements using for loop and store it in a list.
Step 5: Assign the first element in a list as minimum.
Step 6: Compare maximum with the first element in a list,n.
Step 7: Repeat step 8,9 until list becomes empty.
Step 8 : If n is less than the minimum
Step 9 : Assign minimum = n
Step 10 : Display minimum

Pseudocode:

```
BEGIN
SET numlist=[ ]
GET n
FOR i=1 to n
        GET numlist elements
ENDFOR
SET minimum = numlist[0]
FOR i in numlist
        IF (n < minimum)
                minimum = n
        ENDIF
ENDFOR
PRINT minimum
END
```

UNIT II
DATA, EXPRESSIONS, STATEMENTS
PART- A (2 Marks)

1. What is meant by interpreter?

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

2. How will you invoke the python interpreter?

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

3. What is meant by interactive mode of the interpreter?

Interactive mode is a command line shell which gives immediate feedback for each statement, while running previously fed statements in active memory. As new lines are fed into the interpreter, the fed program is evaluated both in part and in whole.

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

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

5. What is a value? What are the different types of values?

A value is one of the fundamental things – like a letter or a number – that a program manipulates. Its types are: integer, float, boolean, strings and lists.

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

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

7. Define keyword and enumerate some of the keywords in Python.

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

8. Define statement and what are its types?

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

9. What do you meant by an assignment statement?

An assignment statement creates new variables and gives them values:
Eg 1: Message = 'And now for something completely different'
Eg 2: n = 17

10. What is tuple?

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

11.What is an expression?

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

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

An operator is a symbol that specifies an operation to be performed on the operands. The data items that an operator acts upon are called operands. The operators +, -, *, / and ** perform addition, subtraction, multiplication, division and exponentiation.
Example: 20+32
In this example, 20 and 32 are operands and + is an operator.

13. What is the order in which operations are evaluated? Give the order of precedence.

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

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

The * operator performs repetition on strings and the + operator performs concatenation on strings.
Example:
>>> 'Hello*3'
Output: HelloHelloHello
>>>'Hello+World'
Output: HelloWorld

15. What is the symbol for comment? Give an example.

is the symbol for comments in Python.
Example:
compute the percentage of the hour that has elapsed

16. What is function call?

A function is a named sequence of statements that performs a computation. When we define a function, we specify the name and the sequence of statements. Later, we can "call" the function by its name called as function call.

17. Identify the parts of a function in the given example.

```
>>> betty = type("32")
>>> print betty
```

The name of the function is *type*, and it displays the type of a value or variable. The value or variable, which is called the argument of the function, is enclosed in parentheses. The argument is *32*. The function returns the result called return value. The return value is stored in *betty*.

18. What is a local variable?

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

15. What is the output of the following?

a. float(32)
b. float("3.14159")

Output:

a. 32.0 The float function converts integers to floating-point numbers.
b. 3.14159 The float function converts strings to floating-point numbers.

16. What do you mean by flow of execution?

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

17. Write down the output for the following program.

```
first = 'throat'
second = 'warbler'
print first + second
```

Output:

```
throatwarbler
```

18. Give the syntax of function definition.

```
def NAME( LIST OF PARAMETERS ):
STATEMENTS
```

19. Explain the concept of floor division.

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

20. What is type coercion? Give example.

Automatic method to convert between data types is called type coercion. For mathematical operators, if any one operand is a float, the other is automatically converted to float.

Eg:

```
>>> minute = 59
>>> minute / 60.0
        0.983333333333
```

21. Write a math function to perform √2 / 2.

```
>>> math.sqrt(2) / 2.0
        0.707106781187
```

22. What is meant by traceback?

A list of the functions that tells us what program file the error occurred in, and what line, and what functions were executing at the time. It also shows the line of code that caused the error.

PART B (16 MARKS)

1. What is the role of an interpreter? Give a detailed note on python interpreter and interactive mode of operation.

An interpreter is a computer program that executes instructions written in a programming language. It can either

- execute the source code directly or
- translates the source code in a first step into a more efficient representation and executes this code

Python interpreter and interactive mode

With the Python interactive interpreter it is easy to check Python commands. The Python interpreter can be invoked by typing the command "python" without any parameter followed by the "return" key at the shell prompt:

```
$ python

>>>
```

Once the Python interpreter is started, you can issue any command at the command prompt ">>>". For example,let us print the "Hello World" statement:

```
>>> print "Hello World"
Hello World
```

In the interactive Python interpreter the print is not necessary:

```
>>> "Hello World"
'Hello World'
>>> 3
3
>>>
```

Typing an end-of-file character (Ctrl+D on Unix, Ctrl+Z on Windows) at the primary prompt causes the interpreter to exit with a zero exit status. If that doesn't work, you can exit the interpreter by typing the following command: quit().

When commands are read from a tty, the interpreter is said to be in *interactive mode*. In this mode it prompts for the next command with the *primary prompt*, usually three greater-than signs (>>>); for continuation lines it prompts with the *secondary prompt*, by default three dots (...). The interpreter prints a welcome message stating its version number and a copyright notice before printing the first prompt:

Continuation lines are needed when entering a multi-line construct. As an example, take a look at this **if** statement:

```
>>> the_world_is_flat = 1
>>> if the_world_is_flat:
...     print "Be careful not to fall off!"
```

2. (a) List down the rules for naming the variable with example.

   A variable is a name that refers to a value. An assignment statement creates new variables and gives them values:

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

   The underscore character, _, can appear in a name. It is often used in names with multiple words, such as my_name or variable_name.

   If you give a variable an illegal name, you get a syntax error:

   (b) List down the different types of operators with suitable examples.

3. What do you mean by rule of precedence? List out the order of precedence and demonstrate in detail with example.

When more than one operator appears in an expression, the order of evaluation depends on the rules of precedence. For mathematical operators, Python follows mathematical convention. The acronym PEMDAS 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 2**1+1 is 3, not 4 and 3*1**3 is 3, not 27.

• Multiplication and Division have the same precedence, which is higher than Addition and Subtraction, which also have the same precedence. 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).

So in the expression degrees / 2 * pi, the division happens first and the result is multiplied by pi. To divide by 2π, you can use parentheses or write degrees / 2 / pi.

4. Explain the role of function call and function definition with example.

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

```
>>> type(32)
<type 'int'>
```

The name of the function is type. The expression in parentheses is called the argument of the function. The result, for this function, is the type of the argument. A function "takes" an argument and "returns" a result. The result is called the return value.

Type conversion functions

Python provides built-in functions that convert values from one type to another. The int function takes any value and converts it to an integer, if it can, or complains otherwise:

```
>>> int('32')
32
>>> int('Hello')

ValueError: invalid literal for int(): Hello
```

int can convert floating-point values to integers, but it doesn't round off; it chops off the fraction part:

```
>>> int(3.99999)
3
```

```
>>> int(-2.3)
-2
```

float converts integers and strings to floating-point numbers:

```
>>> float(32)
32.0
>>> float('3.14159')
3.14159
```

Finally, str converts its argument to a string:

```
>>> str(32)
'32'
>>> str(3.14159)
'3.14159'
```

5. How do you make use of math functions in Python?

Math functions

Python has a math module that provides most of the familiar mathematical functions. A module is a file that contains a collection of related functions.

Before we can use the module, we have to import it:

```
>>> import math
```

This statement creates a module object named math. If you print the module object, you get some information about it:

```
>>> print math
<module 'math' (built-in)>
```

The module object contains the functions and variables defined in the module. To access one of the functions,specify the name of the module and the name of the function, separated by a dot (also known as a period). This format is called dot notation.

```
>>> ratio = signal_power / noise_power
>>> decibels = 10 * math.log10(ratio)
>>> radians = 0.7
>>> height = math.sin(radians)
```

The first example uses log10 to compute a signal-to-noise ratio in decibels (assuming that signal_power and noise_power are defined). The math module also provides log, which computes logarithms base e.

The second example finds the sine of radians. The name of the variable is a hint that sin and the other trigonometric functions (cos, tan, etc.) take arguments in radians. To convert from degrees to radians, divide by 360 and multiply by 2π:

```
>>> degrees = 45
>>> radians = degrees / 360.0 * 2 * math.pi
>>> math.sin(radians)
0.707106781187
```

The expression math.pi gets the variable pi from the math module. The value of this variable is an approximation of π, accurate to about 15 digits.

```
>>> math.sqrt(2) / 2.0
0.707106781187
```

6. Write a Python program to swap two variables.

```
x = 5
y = 10
# create a temporary variable and swap the values
temp = x
x = y
y = temp
print("The value of x after swapping:”,x))
print("The value of y after swapping:”,y))
```

7. Write a Python program to check whether a given year is a leap year or not.

```
# To get year (integer input) from the user
year = int(input("Enter a year"))
if (year % 4) == 0:
   if (year % 100) == 0:
      if (year % 400) == 0:
         print("%d is a leap year"%year)
      else:
         print("%d is not a leap year"%d)
   else:
      print("%d is a leap year"%year)
else:
   print("%d is not a leap year"%year)
```

8. Write a Python program to convert celsius to fahrenheit .
(Formula: celsius * 1.8 = fahrenheit –32).

```
# Python Program to convert temperature in celsius to fahrenheit

# change this value for a different result
celsius = 37.5

# calculate fahrenheit
fahrenheit = (celsius * 1.8) + 32
print('%0.1f degree Celsius is equal to %0.1f degree Fahrenheit' %(celsius,fahrenheit))
```

UNIT III
CONTROL FLOW, FUNCTIONS
PART- A (2 Marks)

1. Define Boolean expression with example.

A boolean expression is an expression that is either true or false. The values true and false are called Boolean values.

Eg :

```
>>> 5 == 6
False
```

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

2. What are the different types of operators?

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

3. Explain modulus operator with example.

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

Eg:

```
>>> remainder = 7 % 3
>>> print remainder
1
```

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

4. Explain relational operators.

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

```
X! = y # x is not equal to y
x > y # x is greater than y
x < y # x is less than y
x >= y # x is greater than or equal to y
x <= y # x is less than or equal to y
```

5. Explain Logical operators

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

6. What is conditional execution?

The ability to check the condition and change the behavior of the program accordingly is called conditional execution. Example:

If statement:

The simplest form of if statement is:

Syntax:

if

statement:

Eg:

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

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

7. What is alternative execution?

A second form of if statement is alternative execution, that is, if …else, where there are two possibilities and the condition determines which one to execute.

Eg:

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

8. What are chained conditionals?

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

Eg:

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

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

9. Explain while loop with example.

Eg:

```
def countdown(n):
        while n > 0:
                print n
                n = n-1
                print 'Blastoff!'
```

More formally, here is the flow of execution for a while statement:

1. Evaluate the condition, yielding True or False.
2. If the condition is false, exit the while statement and continue execution at the next statement.
3. If the condition is true, execute the body and then go back to step 1

9. Explain 'for loop' with example.

The general form of a for statement is

Syntax:

```
for variable in sequence:
    code block
```

Eg:

```
x = 4
for i in range(0, x):
print i
```

10. What is a break statement?

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

Eg:

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

11. What is a continue statement?

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

Eg:

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

12. Compare return value and composition.

Return Value:

Return gives back or replies to the caller of the function. The return statement causes our function to exit and hand over back a value to its caller.

Eg:

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

Composition:

Calling one function from another is called composition.

Eg:

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

13. What is recursion?

The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function.

Eg:

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

14. Explain global and local scope.

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

Eg:

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

Eg:

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

15. Compare string and string slices.

A string is a sequence of character.

Eg: fruit = 'banana'

String Slices :

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

Eg:

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

16. Define string immutability.

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

Program:	Output:

a = "foo"
# a now points to foo 	
b=a
# b now points to the same foo that a points to 	
a=a+a
# a points to the new string "foofoo", but b points to the same old "foo" 	
print a
print b
	#foofoo
	#foo
	It is observed that 'b' hasn't changed even though '*a*' has changed.

17.Mention a few string functions.

s.captilize() – Capitalizes first character of string

s.count(sub) – Count number of occurrences of sub in string

s.lower() – converts a string to lower case

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

18. What are string methods?

A method is similar to a function—it takes arguments and returns a value—but the syntax is different. For example, the method upper takes a string and returns a new string with all uppercase letters:

Instead of the function syntax upper(word), it uses the method syntax word.upper()

```
.>>> word = 'banana'
>>> new_word = word.upper()
>>> print new_word
BANANA
```

19. Explain about string module.

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

Eg:

Program:	**Output:**
import string text = "Monty Python's Flying Circus" print "upper", "=>", string.upper(text) print "lower", "=>", string.lower(text) print "split", "=>", string.split(text) print "join", "=>", string.join(string.split(text), "+") print "replace", "=>", string.replace(text, "Python", "Java") print "find", "=>", string.find(text, "Python"), string.find(text, "Java") print "count", "=>", string.count(text, "n")	upper => MONTY PYTHON'S FLYING CIRCUS lower => monty python's flying circus split => ['Monty', "Python's", 'Flying', 'Circus'] join => Monty+Python's+Flying+Circus replace => Monty Java's Flying Circus find => 6 -1 count => 3

20. What is the purpose of pass statement?

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

Eg:

```
def bar():
    pass
```

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

PART B (16 MARKS)

1. Explain types of operators.

i)Arithmetic Operators

Assume variable a holds 10 and variable b holds 20, then-

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
// Floor Division	The division of operands where the result is the quotient in which the digits after the decimal point are removed	9//2 = 4 and 9.0//2.0 = 4.0

ii) Python Comparison Operators

These operators compare the values on either sides of them and decide the relation among them. They are also called Relational operators.

Assume variable a holds 10 and variable b holds 20, then −

Operator	Description	Example

==	If the values of two operands are equal, then the condition becomes true.	a == b is not true
!=	If values of two operands are not equal, then condition becomes true.	a!=b is 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

(iii) **Logical operators**

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

(iv) Assignment operators

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

2. Explain conditional alternative and chained conditional.

The simplest form of if statement is:

Syntax:

if

statement:

Eg:

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

The boolean expression after 'if' is called the condition. If it is true, then the indented statement gets executed. If not, nothing happens. if statements have the same structure as function definitions: a header followed by an indented body. Statements like this are called compound statements. There is no limit on the number of statements that can appear in the body, but there has to be at least one. Occasionally, it is useful to have a body with no statements .In that case, you can use the pass statement, which does nothing.

```
if x < 0:
pass # need to handle negative values!
```

Alternative execution (if-else):

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

Eg:

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

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

Chained conditionals(if-elif-else):

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

Eg:

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

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

Eg:

```
if choice == 'a':
        draw_a()
```

```
elif choice =='b':
        draw_b()
elif choice =='c':
        draw_c()
```

Each condition is checked in order. If the first is false, the next is checked, and so on. If one of them is true, the corresponding branch executes, and the statement ends. Even if more than one condition is true, only the first true branch executes.

3. Explain in detail about Fruitful Functions.

Return values

Some of the built-in functions we have used, such as the math functions, produce results. Calling the function generates a value, which we usually assign to a variable or use as part of an expression.

The first example is area, which returns the area of a circle with the given radius:

Eg:

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

We have seen the return statement before, but in a fruitful function the return statement includes an expression. This statement means: "Return immediately from this function and use the following expression as a return value." The expression can be arbitrarily complicated, so we could have written this function more concisely:

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

On the other hand, temporary variables like temp often make debugging easier. Sometimes it is useful to have multiple return statements, one in each branch of a conditional:

```
def absolute_value(x):
 if x < 0:
    return -x
 else:
    return x
```

4. Write a Python program to find the square root of a number.

```
n=int(input("Enter a number to find Square Root:"))
```

```
approx = 0.5 * n
better = 0.5 * (approx + n/approx)
while better != approx:
   approx = better
   better = 0.5 * (approx + n/approx)
print(approx)
```

5. Explain RECURSION.

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

Python program:

```
 # take input from the user
 num = int(input("Enter a number: "))
def fact(n):
    if n == 1:
        return n
    else:
        return n*fact(n-1)
print("Factorial of n numbers is :%d" %(fact(n)))
```

6. Explain string slices and string immutability.

String slices

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

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

The operator [n:m]returns the part of the string from the "n-eth" character to the "m-eth" character, including the first but excluding the last. If you omit the first index (before the colon), the slice starts at the beginning of the string. If you omit the second index, the slice goes to the end of the string:

```
>>> fruit = 'banana'
```

```
>>> fruit[:3]
'ban'
>>> fruit[3:]
'ana'
```

If the first index is greater than or equal to the second the result is an empty string, represented by two quotation marks:

```
>>> fruit ='banana'
>>> fruit[3:3]
```

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

String immutability.

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

Eg:

```
a = "foo"
# a now points to foo
b=a
# b now points to the same foo that a points to
a=a+a
# a points to the new string "foofoo", but b points to the same old "foo"
print a
print b
# Output
#foofoo
#foo
```

It is observed that b hasn't changed even though 'a' has changed the value.

7. Explain string functions and methods.

There are a number of useful operations that can be performed with string. One of the most useful of these is the function **split.** This function takes a string (typically a line of input from the user) and splits it into individual words.

Another useful function is **lower**, which converts text into lower case.

Eg:

>>> line = input("What is your name?")

What is your name? Timothy Alan Budd

>>> lowname = line.lower()

>>> print lowname.split()

['timothy', 'alan', 'budd']

Other useful functions will **search a string** for a given text value, or **strip leading or trailing** white space from a string. An alternative version of split takes as argument the separator string. The string is broken into a list using the separator as a division. This can be useful, for example, for breaking a file path name into parts:

Eg:

>>> pathname = '/usr/local/bin/ls'

>>> pathname.split('/')

['usr', 'local', 'bin', 'ls']

The inverse of split is the function **join**. The argument to join is a list of strings. The value to the left of the dot is the separator that will be placed between each element. Often this is simply an empty string. The values in the list are laminated along with the separator to produce the result string.

>>> lst = ['abc','pdq','xyz']

>>> pri

nt '::'.join(lst)

abc::pdq::xyz

String methods

A method is similar to a function—it takes arguments and returns a value—but the syntax is different. For example, the method upper takes a string and returns a new string with all uppercase letters:

Instead of the function syntax upper(word), it uses the method syntax word.upper()

.>>> word = 'banana'

>>> new_word = word.upper()

>>> print new_word

BANANA

This form of dot notation specifies the name of the method, upper, and the name of the string to apply the method to, word. The empty parentheses indicate that this method takes no argument. A method call is called an invocation ; in this case, we would say that we are invoking upper on the word. As it turns out, there is a string method named find that is remarkably similar to the function we wrote:

```
>>> word = 'banana'
>>> index = word.find('a')
>>> print index
1
```

In this example, we invoke find on word and pass the letter we are looking for as a parameter. Actually, the find method is more general than our function; it can find substrings, not just characters:

```
>>> word.find('na')
2
```

It can take as a second argument the index where it should start:

```
>>> word.find('n', 3)
4
```

And as a third argument the index where it should stop:

```
>>> name ='bob'
>>> name.find('b', 1, 2)
-1
```

This search fails because b does not appear in the index range from 1 to 2 (not including 2).

8. Explain string module.

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

Eg:

```
import string
text = "Monty Python's Flying Circus"
print "upper", "=>", string.upper(text)
print "lower", "=>", string.lower(text)
```

```
print "split", "=>", string.split(text)
print "join", "=>", string.join(string.split(text), "+")
print "replace", "=>", string.replace(text, "Python", "Java")
print "find", "=>", string.find(text, "Python"), string.find(text, "Java")
print "count", "=>", string.count(text, "n")
```

Eg: Using string methods instead of string module functions

```
text = "Monty Python's Flying Circus"
print "upper", "=>", text.upper()
print "lower", "=>", text.lower()
print "split", "=>", text.split()
print "join", "=>", "+".join(text.split())
print "replace", "=>", text.replace("Python", "Perl")
print "find", "=>", text.find("Python"), text.find("Perl")
print "count", "=>", text.count("n")
```

9. Explain list as arrays.
10. Write a Python program to find GCD of two numbers.

```
d1=int(input("Enter a number:"))
 d2=int(input("Enter another number"))
rem=d1%d2
while rem!=0 :
   print(rem)
   d1=d2
   d2=rem
   rem=d1%d2
print("gcd of given numbers is : %d" %(d2))
```

11. Write a Python program to find the exponentiation of a number.

```
print("Calculation of X^Y")
x=int(input("Enter X Value :"))
y=int(input("Enter Y Value :"))
powered = x
if y == 0:
   powered=1
else:
   while y > 1:
     powered *= x
      y -= 1
```

```
print(powered)
```

12. Write a Python program to sum an array of numbers.

```
def sum_arr (arr,size):
    if (size = = 0):
        return 0
    else:
        return arr[size-1]+ sum_arr (arr, size-1)
n = int(input("Enter the number of elements for the list:"))
a = [ ]
for I in range (0,n):
    element = int (input("Enter element:"))
    a.apppend (element)
print (" The list is: ")
print a
print ( " The Sum is : ")
b = sum_arr(a,n)
print(b)
```

13. Write a Python program to perform linear search.

```
data = []
n = int(raw_input('Enter Number of Elements in the Array: '))
for i in range(0, n):
   x = raw_input('Enter the Element %d :' %(i+1))
   data.append(x)
e= int(raw_input('Enter the Element to be Search '))
pos = 0
found= False
while pos < n and not found:
   if int(data[pos])==e:
      found= True
   else:
      pos = pos+1
if found:
     print('Element %d Found at Position %d ' %(e,pos+1))
else:
   print('Element %d is Not Found in the Array '%(e))
```

14. Write a Python program to perform binary search.

```
data = []
n = int(input('Enter Number of Elements in the Array: '))
print('Enter the Elements in Ascending Order' )
for i in range(0, n):
        x = int(input('Enter the Element %d :' %(i+1)))
   data.append(x)
```

```
e= int(input('Enter the Element to be Search '))
first = 0
last = n-1
found = False
while( first<=last and not found):
   mid = (first + last)/2
    if int(data[mid]) == e :
       found = True
    else:
       if e < int(data[mid]):
          last = mid - 1
       else:
          first = mid + 1
if found:
    print('Element %d Found at Position %d ' %(e,mid+1))
else:
    print('Element %d is Not Found in the Array '%(e))
```

UNIT IV
LISTS, TUPLES AND DICTIONARIES
PART- A (2 Marks)

1. What is a list?

A list is an ordered set of values, where each value is identified by an index. The values that make up a list are called its elements. Lists are similar to strings, which are ordered sets of characters, except that the elements of a list can have any type.

2. Solve a)[0] * 4 and b) [1, 2, 3] * 3.

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

3.Let list = ['a', 'b', 'c', 'd', 'e', 'f']. Find a) list[1:3] b) t[:4] c) t[3:] .

```
>>>  list = ['a', 'b', 'c', 'd', 'e', 'f']
>>>  list[1:3] ['b', 'c']
>>>  list[:4] ['a', 'b', 'c', 'd']
>>>  list[3:] ['d', 'e', 'f']
```

4. Mention any 5 list methods.

append() ,extend () ,sort(), pop(),index(),insert and remove()

5. State the difference between lists and dictionary.

List is a mutable type meaning that it can be modified whereas dictionary is immutable and is a key value store. Dictionary is not ordered and it requires that the keys are hashable whereas list can store a sequence of objects in a certain order.

6. What is List mutability in Python? Give an example.

Python represents all its data as objects. Some of these objects like **lists** and dictionaries are **mutable**, i.e., their content can be changed without changing their identity. Other objects like integers, floats, strings and tuples are objects that cannot be changed.

Example:

```
>>> numbers = [17, 123]
 >>> numbers[1] = 5
>>> print numbers [17, 5]
```

7. What is aliasing in list? Give an example.

An object with more than one reference has more than one name, then the object is said to be aliased. Example: If *a* refers to an object and we assign b = a, then both variables refer to the same object:

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

8. Define cloning in list.

In order to modify a list and also keep a copy of the original, it is required to make a copy of the list itself, not just the reference. This process is called cloning, to avoid the ambiguity of the word "copy".

9. Explain List parameters with an example.

Passing a list as an argument actually passes a reference to the list, not a copy of the list. For example, the function head takes a list as an argument and returns the first element:

```
 def head(list):
 return list[0]
```

output:

```
>>> numbers = [1, 2, 3]
>>> head(numbers)
```

10. Write a program in Python to delete first element from a list.

```
def deleteHead(list): del list[0]
```

Here's how deleteHead is used:

```
>>> numbers = [1, 2, 3]
>>> deleteHead(numbers)
 >>> print numbers [2, 3]
```

11. Write a program in Python returns a list that contains all but the first element of the given list.

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

Here's how tail is used:

```
>>> numbers = [1, 2, 3]
>>> rest = tail(numbers)
 >>> print rest [2, 3]
```

12. What is the benefit of using tuple assignment in Python?

It is often useful to swap the values of two variables. With conventional assignments a temporary variable would be used. For example, to swap a and b:

```
>>> temp = a
>>> a = b
>>> b = temp
```

This solution is cumbersome; tuple assignment is more elegant:

```
>>> a, b = b, a
```

13. Define key-value pairs.

The elements of a dictionary appear in a comma-separated list. Each entry contains an index and a value separated by a colon. In a dictionary, the indices are called keys, so the elements are called key-value pairs.

14. Define dictionary with an example.

A dictionary is an associative array (also known as hashes). Any key of the dictionary is associated (or mapped) to a value. The values of a dictionary can be any Python data type. So dictionaries are unordered key-value-pairs.

Example:

```
>>> eng2sp = {}  # empty dictionary
>>> eng2sp['one'] = 'uno'
>>> eng2sp['two'] = 'dos'
```

15. How to return tuples as values?

A function can only return one value, but if the value is a tuple, the effect is the same as returning multiple values. For example, if we want to divide two integers and compute the quotient and remainder, it is inefficient to compute x/y and then x%y. It is better to compute them both at the same time.

```
>>> t = divmod(7, 3)
>>> print t (2, 1)
```

16. List two dictionary operations.

- Del -removes key-value pairs from a dictionary
- Len - returns the number of key-value pairs

17.Define dictionary methods with an example.

A method is similar to a function—it takes arguments and returns a value— but the syntax is different. For example, the keys method takes a dictionary and returns a list of the keys that appear, but instead of the function syntax keys(eng2sp), method syntax eng2sp.keys() is used.

```
>>> eng2sp.keys() ['one', 'three', 'two']
```

18.Define List Comprehension.

List comprehensions apply an **arbitrary expression** to items in an iterable rather than applying function. It provides a compact way of mapping a list into another list by applying a function to each of the elements of the list.

19.Write a Python program to swap two variables.

```
x = 5
y = 10
temp = x
x = y
y = temp
```

```
print('The value of x after swapping: {}'.format(x))
print('The value of y after swapping: {}'.format(y))
```

20.Write the syntax for list comprehension.

The list comprehension starts with a '**[' and ']'**, to help us remember that the result is going to be a list. The basic syntax is [expression for item in list if conditional].

PART B (16 MARKS)

1. Explain in detail about lists, list operations and list slices.

A list is an ordered set of values, where each value is identified by an index. The values that make up a list are called its elements. Lists are similar to strings, which are ordered sets of characters, except that the elements of a list can have any type. There are several ways to create a new list. The simplest is to enclose the elements in square brackets ([and]):

[10, 20, 30, 40]

["spam", "bungee", "swallow"]

The following list contains a string, a float, an integer, and (mirabile dictu) another list:

["hello", 2.0, 5, [10, 20]]

A list within another list is said to be nested. Lists that contain consecutive integers are common, so Python provides a simple way to create them:

>>> range (1,5) [1, 2, 3, 4].

LIST OPERATIONS

The + operator concatenates lists
: >>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
>>> print c [1, 2, 3, 4, 5, 6]
Similarly, the * operator repeats a list a given number of times:

>>> [0] * 4 [0, 0, 0, 0]

>>> [1, 2, 3] * 3 [1, 2, 3, 1, 2, 3, 1, 2, 3]

LIST SLICES

The slice operator also works on lists:

>>> t = ['a', 'b', 'c','d', 'e', 'f']

>>> t[1:3] ['b', 'c']

>>> t[:4] ['a', 'b', 'c', 'd']

>>> t[3:] ['d', 'e', 'f']

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

>>> t[:] ['a', 'b', 'c', 'd', 'e', 'f']

Since lists are mutable, it is often useful to make a copy before performing operations that fold, spindle or mutilate lists. A slice operator on the left side of an assignment can update multiple elements:

>>> t = ['a', 'b', 'c',’d’, 'e', 'f']

>>> t[1:3] = ['x', 'y']

>>> print t ['a', 'x', 'y', 'd', 'e', 'f,]

Like strings, lists can be indexed and sliced:

```
>>> list = [2, 4, "usurp", 9.0, "n"]
>>> list[2]
'usurp'
>>> list[3:]
[9.0, 'n']
```

Much like the slice of a string is a substring, the slice of a list is a list. However, lists differ from strings in that we can assign new values to the items in a list:

```
>>> list[1] = 17
>>> list
[2, 17, 'usurp', 9.0, 'n']
```

We can assign new values to slices of the lists, which don't even have to be the same length:

```
>>> list[1:4] = ["opportunistic", "elk"]
>>> list
[2, 'opportunistic', 'elk', 'n']
```

It's even possible to append items onto the start of lists by assigning to an empty slice:

```
>>> list[:0] = [3.14, 2.71]
>>> list
[3.14, 2.71, 2, 'opportunistic', 'elk', 'n']
```

Similarly, you can append to the end of the list by specifying an empty slice after the end:

```
>>> list[len(list):] = ['four', 'score']
>>> list
```

```
[3.14, 2.71, 2, 'opportunistic', 'elk', 'n', 'four', 'score']
```

You can also completely change the contents of a list:

```
>>> list[:] = ['new', 'list', 'contents']
>>> list
['new', 'list', 'contents']
```

The right-hand side of a list assignment statement can be any iterable type:

```
>>> list[:2] = ('element',('t',),[])
>>> list
['element', ('t',), [], 'contents']
```

With slicing you can create copy of list since slice returns a new list:

```
>>> original = [1, 'element', []]
>>> list_copy = original[:]
>>> list_copy
[1, 'element', []]
>>> list_copy.append('new element')
>>> list_copy
[1, 'element', [], 'new element']
>>> original
[1, 'element', []]
```

Note, however, that this is a shallow copy and contains references to elements from the original list, so be careful with mutable types:

```
>>> list_copy[2].append('something')
>>> original
[1, 'element', ['something']]
```

Non-Continuous slices

It is also possible to get non-continuous parts of an array. If one wanted to get every n-th occurrence of a list, one would use the :: operator. The syntax is a:b:n where a and b are the start and end of the slice to be operated upon.

```
>>> list = [i for i in range(10) ]
```

```
>>> list
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list[::2]
[0, 2, 4, 6, 8]
>>> list[1:7:2]
[1, 3, 5]
```

2. Explain in detail about list methods and list loops with examples.

Python provides methods that operate on lists. Some of the methods are

- Append()
- Extend()
- Sort()
- Pop()

For example, append adds a new element to the end of a list:

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

>>> t.append('d')

>>> print t ['a', 'b', 'c', 'd']
```

Extend takes a list as an argument and appends all of the elements:

```
>>> t1 = ['a', 'b', 'c']

>>> t2 = ['d', 'e']

>>> t1.extend(t2)

>>> print t1 ['a', 'b', 'c', 'd', 'e']
```

This example leaves t2 unmodified. sort arranges the elements of the list from low to high:

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

>>> t.sort()

>>> print t ['a', 'b', 'c', 'd', 'e']
```

Remove the item in the list at the index *i* and return it. If *i* is not given, remove the the last item in the list and return it.

```
>>> list = [1, 2, 3, 4]

>>> a = list.pop(0)
```

```
>>> list
[2, 3, 4]
>>> a
```

List methods are all void; they modify the list and return None.

LIST LOOPS

Here are two functions that both generate ten million random numbers, and return the sum of the numbers. They both work.

```
import random
joe = random.Random()
 def sum1():
 """ Build a list of random numbers, then sum them """        # Generate one random ‹→number
xs = []
for i in range(10000000):
num = joe.randrange(1000 )
 xs.append(num)                                               # Save it in our list
tot = sum(xs)
return tot
def sum2():
 """ Sum the random numbers as we generate them """
 tot = 0
for i in range(10000000):
num = joe.randrange(1000)
 tot += num
 return tot
 print(sum1())
 print(sum2())
```

3. Explain in detail about mutability and tuples with a Python program.

MUTABILITY

Unlike strings, lists are mutable, which means we can change their elements. Using the bracket operator on the left side of an assignment, we can update one of the elements:

```
>>> fruit = ["banana", "apple", "quince"]
>>> fruit[0] = "pear"
>>> fruit[-1] = "orange"
>>> print fruit [’pear’, ’apple’, ’orange’]
```

With the slice operator we can update several elements at once:

```
>>> list = ['a', 'b', 'c', 'd', 'e', 'f']
>>> list[1:3] = ['x', 'y']
>>> print list ['a', 'x', 'y', 'd', 'e', 'f'] \
```

We can also remove elements from a list by assigning the empty list to them:

```
>>> list = ['a', 'b', 'c', 'd', 'e', 'f']
    >>> list[1:3] = []
>>> print list ['a', 'd', 'e', 'f']
```

And we can add elements to a list by squeezing them into an empty slice at the desired location:

```
    >>> list = ['a', 'd', 'f']
    >>> list[1:1] = ['b', 'c']
    >>> print list ['a', 'b', 'c', 'd', 'f']
    >>> list[4:4] = ['e']
    >>> print list ['a', 'b', 'c', 'd', 'e', 'f']
```

Tuple assignment

It is often useful to swap the values of two variables. With conventional assignments, you have to use a temporary variable. For example, to swap a and b:

```
>>> temp = a
>>> a = b
>>> b = temp
```

This solution is cumbersome; tuple assignment is more elegant:

```
>>> a, b = b, a
```

The left side is a tuple of variables; the right side is a tuple of expressions. Each value is assigned to its respective variable. All the expressions on the right side are evaluated before any of the assignments. The number of variables on the left and the number of values on the right have to be the same:

```
>>> a, b = 1, 2, 3
```

ValueError: too many values to unpack More generally, the right side can be any kind of sequence (string, list or tuple). For example, to split an email address into a user name and a domain, you could write:

```
>>> addr = 'monty@python.org'
```

>>> uname, domain = addr.split('@')

The return value from split is a list with two elements; the first element is assigned to uname, the second to domain.

>>> print uname monty

>>> print domain python.org

4. What is tuple assignment? Explain it with an example.

It is often useful to swap the values of two variables. With conventional assignments, you have to use a temporary variable. For example, to swap a and b:

>>> temp = a

>>> a = b

>>> b = temp

This solution is cumbersome; tuple assignment is more elegant:

>>> a, b = b, a

The left side is a tuple of variables; the right side is a tuple of expressions. Each value is assigned to its respective variable. All the expressions on the right side are evaluated before any of the assignments. The number of variables on the left and the number of values on the right have to be the same:

>>> a, b = 1, 2, 3

ValueError: too many values to unpack More generally, the right side can be any kind of sequence (string, list or tuple). For example, to split an email address into a user name and a domain, you could write:

>>> addr = 'monty@python.org'

>>> uname, domain = addr.split('@')

The return value from split is a list with two elements; the first element is assigned to uname, the second to domain.

>>> print uname monty

>>> print domain python.org

5. Is it possible to return tuple as values? Justify your answer with an example.

Yes, it is possible to return tuple as values. Example: Functions can return tuples as return values. For example, we could write a function that swaps two parameters:

def swap(x, y):

return y, x

Then we can assign the return value to a tuple with two variables:

a, b = swap(a, b)

In this case, there is no great advantage in making swap a function. In fact, there is a danger in trying to encapsulate swap, which is the following tempting mistake:

```
def swap(x, y): # incorrect version
```

x, y = y, x

If we call this function like this: swap(a, b) then a and x are aliases for the same value. Changing x inside swap makes x refer to a different value, but it has no effect on a in main. Similarly, changing y has no effect on b. This function runs without producing an error message, but it doesn't do what we intended. This is an example of a semantic error.

The built-in function divmod takes two arguments and returns a tuple of two values, the quotient and remainder. You can store the result as a tuple:

```
>>> t = divmod(7, 3)
>>> print t (2, 1)
```

Or use tuple assignment to store the elements separately:

```
>>> quot, rem = divmod(7, 3)
>>> print quot 2
>>> print rem 1
```

Here is an example of a function that returns a tuple: def min_max(t): return min(t), max(t) max and min are built-in functions that find the largest and smallest elements of a sequence. min_max computes both and returns a tuple of two values.

6. Explain in detail about dictionaries and its operations.

DICTIONARIES

A dictionary is like a list, but more general. In a list, the indices have to be integers; in a dictionary they can be (almost) any type. You can think of a dictionary as a mapping between a set of indices (which are called keys) and a set of values. Each key maps to a value. The association of a key and a value is called a key-value pair or sometimes an item. As an example, we'll build a dictionary that maps from English to Spanish words, so the keys and the values are all strings.

The function dict creates a new dictionary with no items. Because dict is the name of a built-in function, you should avoid using it as a variable name.

```
>>> eng2sp = dict()
```

>>> print eng2sp {}

The squiggly-brackets, {}, represent an empty dictionary. To add items to the dictionary, you can use square brackets: >>> eng2sp['one'] = 'uno' This line creates an item that maps from the key 'one' to the value 'uno'. If we print the dictionary again, we see a key-value pair with a colon between the key and value:

>>> print eng2sp {'one': 'uno'}

This output format is also an input format. For example, you can create a new dictionary with three items:

>>> eng2sp = {'one': 'uno', 'two': 'dos', 'three': 'tres'}

But if you print eng2sp, you might be surprised:

>>> print eng2sp {'one': 'uno', 'three': 'tres', 'two': 'dos'}

The order of the key-value pairs is not the same. In fact, if you type the same example on your computer, you might get a different result. In general, the order of items in a dictionary is unpredictable. But that's not a problem because the elements of a dictionary are never indexed with integer indices. Instead, you use the keys to look up the corresponding values:

>>> print eng2sp['two'] 'dos'

The key 'two' always maps to the value 'dos' so the order of the items doesn't matter. If the key isn't in the dictionary, you get an exception:

>>> print eng2sp['four'] KeyError: 'four'

The len function works on dictionaries; it returns the number of key-value pairs:

>>> len(eng2sp)

3

The in operator works on dictionaries; it tells you whether something appears as a key in the dictionary (appearing as a value is not good enough).

>>> 'one' in eng2sp True

>>> 'uno' in eng2sp False

To see whether something appears as a value in a dictionary, you can use the method values, which returns the values as a list, and then use the in operator:

>>> vals = eng2sp.values()

>>> 'uno' in vals

True

The in operator uses different algorithms for lists and dictionaries. For lists, it uses a search algorithm, as in Section 8.6. As the list gets longer, the search time gets longer in direct proportion. For dictionaries, Python uses an algorithm called a hashtable that has a remarkable property: the in operator takes about the same amount of time no matter how many items there are in a dictionary

Dictionary operations

The del statement removes a key-value pair from a dictionary. For example, the following dictionary contains the names of various fruits and the number of each fruit in stock:

>>> inventory = {'apples': 430, 'bananas': 312, 'oranges': 525, 'pears': 217}

>>> print inventory {'oranges': 525, 'apples': 430, 'pears': 217, 'bananas': 312}

If someone buys all of the pears, we can remove the entry from the dictionary:

>>> del inventory['pears']

>>> print inventory {'oranges': 525, 'apples': 430, 'bananas': 312}

Or if we're expecting more pears soon, we might just change the value associated with pears:

>>> inventory['pears'] = 0

>>> print inventory {'oranges': 525, 'apples': 430, 'pears': 0, 'bananas': 312}

The len function also works on dictionaries; it returns the number of key-value pairs:

>>> len(inventory) 4

7. Explain in detail about dictionary methods.

8. DICTIONARY METHODS

A method is similar to a function—it takes arguments and returns a value— but the syntax is different. For example, the keys method takes a dictionary and returns a list of the keys that appear, but instead of the function syntax keys(eng2sp), we use the method syntax eng2sp.keys().

>>> eng2sp.keys() ['one', 'three', 'two']

This form of dot notation specifies the name of the function, keys, and the name of the object to apply the function to, eng2sp. The parentheses indicate that this method has no parameters. A method call is called an invocation; in this case, we would say that we are invoking keys on the object eng2sp.

The values method is similar; it returns a list of the values in the dictionary:

>>> eng2sp.values() ['uno', 'tres', 'dos']

The items method returns both, in the form of a list of tuples—one for each key-value pair:

>>> eng2sp.items() [('one','uno'), ('three', 'tres'), ('two', 'dos')]

The syntax provides useful type information. The square brackets indicate that this is a list. The parentheses indicate that the elements of the list are tuples. If a method takes an argument, it uses the same syntax as a function call. For example, the method has key takes a key and returns true (1) if the key appears in the dictionary:

>>> eng2sp.has_key('one')

True

>>> eng2sp.has_key('deux')

False

If you try to call a method without specifying an object, you get an error. In this case, the error message is not very helpful:

>>> has_key('one') NameError: has_key 108

8. Explain in detail about list comprehension .Give an example.

List comprehensions

List comprehensions provide a concise way to create lists. It consists of brackets containing an expression followed by a for clause, then zero or more for or if clauses. The expressions can be anything, i.e., all kinds of objects can be in lists. The result will be a new list resulting from evaluating the expression in the context of the for and if clauses which follow it. The list comprehension always returns a result list.

Syntax

The list comprehension starts with a '**[**' **and** '**]**', to help you remember that the result is going to be a list.

The basic syntax is

[expression for item in list if conditional]

This is equivalent to:

```
for item in list:
  if conditional:
    expression
```

List comprehension is a method to describe the process using which the list should be created. To do that, the list is broken into two pieces. The first is a picture of what each element will look like, and the second is what is done to get it.

For instance, let's say we have a list of words:

```
listOfWords = ["this","is","a","list","of","words"]
```

To take the first letter of each word and make a list out of it using list comprehension:

```
>>> listOfWords = ["this","is","a","list","of","words"]
>>> items = [ word[0] for word in listOfWords ]
>>> print items
['t', 'i', 'a', 'l', 'o', 'w']
```

List comprehension supports more than one for statement. It will evaluate the items in all of the objects sequentially and will loop over the shorter objects if one object is longer than the rest.

```
>>> item = [x+y for x in 'cat' for y in 'pot']
>>> print item
['cp', 'co', 'ct', 'ap', 'ao', 'at', 'tp', 'to', 'tt']
```

List comprehension supports an if statement, to only include members into the list that fulfill a certain condition:

```
>>> print [x+y for x in 'cat' for y in 'pot']
['cp', 'co', 'ct', 'ap', 'ao', 'at', 'tp', 'to', 'tt']
>>> print [x+y for x in 'cat' for y in 'pot' if x != 't' and y != 'o' ]
['cp', 'ct', 'ap', 'at']
>>> print [x+y for x in 'cat' for y in 'pot' if x != 't' or y != 'o' ]
['cp', 'co', 'ct', 'ap', 'ao', 'at', 'tp', 'tt']
```

9. Write a Python program for a) selection sort b) insertion sort.

SELECTION SORT:

PROGRAM:

```
def selectionsort( aList ):
        for i in range( len( aList ) ):
            least = i
        for k in range( i + 1 , len( aList ) ):
            if aList[k] < aList[least]:
```

```
                    least = k
                    swap( aList, least, i )
    def swap( A, x, y ):
            tmp = A[x]
            A[x] = A[y]
            A[y] = tmp
            aList = [54,26,93,17,77,31,44,55,20]
    selectionsort(aList)
    print(aList)
```

Insertion sort:

```
    def insertionSort(alist):
      for index in range(1,len(alist)):
        currentvalue = alist[index]
        position = index
        while position>0 and alist[position-1]>currentvalue:
          alist[position]=alist[position-1]
          position = position-1
        alist[position]=currentvalue
    alist = [54,26,93,17,77,31,44,55,20]
    insertionSort(alist)
    print(alist)
```

10. Write a Python program for a) merge sort b) quick sort.

Merge sort:

```
def mergeSort(alist):
    print("Splitting ",alist)
    if len(alist)>1:
        mid = len(alist)//2
```

```
        lefthalf = alist[:mid]
        righthalf = alist[mid:]
        mergeSort(lefthalf)
        mergeSort(righthalf)
        i=0
        j=0
        k=0
        while i < len(lefthalf) and j < len(righthalf):
            if lefthalf[i] < righthalf[j]:
                alist[k]=lefthalf[i]
                i=i+1
            else:
                alist[k]=righthalf[j]
                j=j+1
            k=k+1

        while i < len(lefthalf):
            alist[k]=lefthalf[i]
            i=i+1
            k=k+1
        while j < len(righthalf):
            alist[k]=righthalf[j]
            j=j+1
            k=k+1
    print("Merging ",alist)
alist = [54,26,93,17,77,31,44,55,20]
mergeSort(alist)
print(alist)
```

Quicksort:

```
from random import randrange

def partition(lst, start, end, pivot):

    lst[pivot], lst[end] = lst[end], lst[pivot]

    store_index = start

    for i in xrange(start, end):

        if lst[i] < lst[end]:

            lst[i], lst[store_index] = lst[store_index], lst[i]

            store_index += 1

    lst[store_index], lst[end] = lst[end], lst[store_index]

    return store_index

def quick_sort(lst, start, end):

    if start >= end:
```

```
        return lst
    pivot = randrange(start, end + 1)
    new_pivot = partition(lst, start, end, pivot)
    quick_sort(lst, start, new_pivot - 1)
    quick_sort(lst, new_pivot + 1, end)
def sort(lst):
    quick_sort(lst, 0, len(lst) - 1)
    return lst
print sort([-5, 3, -2, 3, 19, 5])
print sort([345,45,89,569,23,67,56,90,100])
```

11. Write a Python program to implement histogram.

```
def histogram( items ):
    for n in items:
        output = ''
        times = n
        while( times > 0 ):
          output += '*'
          times = times - 1
        print(output)
histogram([2, 3, 6, 5])
```

UNIT V
FILES, MODULES AND PACKAGES
PART- A (2 Marks)

1. What is a text file?

A text file is a file that contains printable characters and whitespace, organized in to lines separated by newline characters.

2. Write a python program that writes “Hello world” into a file.

```
f =open("ex88.txt",'w')
f.write("hello world")
f.close()
```

3. Write a python program that counts the number of words in a file.

```
f=open("test.txt","r")
content =f.readline(20)
words =content.split()
```

```
print(words)
```

4. **What are the two arguments taken by the open() function?**

The open function takes two arguments : name of the file and the mode of operation.

Example: f = open("test.dat","w")

5. **What is a file object?**

A file object allows us to use, access and manipulate all the user accessible files. It maintains the state about the file it has opened.

6. **What information is displayed if we print a file object in the given program?**

```
f= open("test.txt","w")
print f
```

The name of the file, mode and the location of the object will be displayed.

7. **What is an exception?**

Whenever a runtime error occurs, it creates an exception. The program stops execution and prints an error message. For example, dividing by zero creates an exception:

```
print 55/0
ZeroDivisionError: integer division or modulo
```

8. **What are the error messages that are displayed for the following exceptions?**

a. Accessing a non-existent list item
b. Accessing a key that isn't in the dictionary
c. Trying to open a non-existent file
a. IndexError: list index out of range
b. KeyError: what
c. IOError: [Errno 2] No such file or directory: 'filename'

9. **What are the two parts in an error message?**

The error message has two parts: the type of error before the colon, and speci_cs about the error after the colon.

10. **How do you handle the exception inside a program when you try to open a non-existent file?**

```
filename = raw_input('Enter a file name: ')
try:
f = open (filename, "r")
except IOError:
print 'There is no file named', filename
```

11. **How does try and execute work?**

The *try* statement executes the statements in the first block. If no exception occurs, then *except* statement is ignored. If an exception of type *IOError* occurs, it executes the statements in the *except branch* and then continues.

12. **What is the function of raise statement? What are its two arguments?**

The raise statement is used to raise an exception when the program detects an error. It takes two arguments: the exception type and specific information about the error.

13. **What is a pickle?**

Pickling saves an object to a file for later retrieval. The pickle module helps to translate almost any type of object to a string suitable for storage in a database and then translate the strings back in to objects.

14. What are the two methods used in pickling?

The two methods used in pickling are pickle.dump() and pickle.load(). To store a data structure, dump method is used and to load the data structures that are dumped , load method is used.

15. What is the use of the format operator?

The format operator % takes a format string and a tuple of expressions and yields a string that includes the expressions, formatted according to the format string.

16. What are modules?

A module is simply a file that defines one or more related functions grouped together. To reuse the functions of a given module, we need to import the module.

Syntax: import <modulename>

17. What is a package?

Packages are namespaces that contain multiple packages and modules themselves. They are simply directories.

Syntax: from <mypackage> import <modulename>

18. What is the special file that each package in Python must contain?

Each package in Python must contain a special file called __init__.py

19. How do you delete a file in Python?

The remove() method is used to delete the files by supplying the name of the file to be deleted as argument.

Syntax: os.remove(filename)

20. How do you use command line arguments to give input to the program?

Python sys module provides access to any command-line arguments via sys.argv. sys.argv is the list of command-line arguments. len(sys.argv) is the number of command-line arguments.

PART B (16 MARKS)

1. Write a function that copies a file reading and writing up to 50 characters at a time.

```
def copyFile(oldFile, newFile):
        f1 = open(oldFile, "r")
        f2 = open(newFile, "w")
        while True:
                text = f1.read(50)
                if text == "":
                break
         f2.write(text)
         f1.close()
        f2.close()
        return
```

2. (a) Write a program to perform exception handling.

```
def exists(filename):
    try:
        f = open(filename)
        f.close()
        return True
    except IOError:
        return False
```

(b) Write a Python program to handle multiple exceptions.

```
try:
    x = float(raw_input("Your number: "))
     inverse = 1.0 / x
except ValueError:
    print "You should have given either an int or a float"
except ZeroDivisionError:
    print "Infinity"
```

3. Write a python program to count number of lines, words and characters in a text file.

```
def wordCount():
    cl=0
    cw=0
    cc=0
    f=open("ex88.txt","r")
    for line in f:
        words=line.split()
        cl +=1
        cw +=len(words)
        cc +=len(line)
    print('No. of lines:',cl)
    print('No. of words:',cw)
    print('No. of characters:',cc)
    f.close()
```

4. Write a Python program to illustrate the use of command-line arguments.

```
import sys
def inputCmd():
    print ("Name of the script:", sys.argv[0])
    print ("Number of arguments:", len(sys.argv))
    print ("The arguments are:", str(sys.argv))
```

5. Mention the commands and their syntax for the following: get current directory, changing directory, list, directories and files, make a new directory, renaming and removing directory.

(a) Get current directory: getcwd()
Syntax : import os
os.getcwd()

(a) Changing directory: chdir()
Syntax: os.chdir('C:\\Users')
os.getcwd()

(b) List directories and files: listdir()
Syntax: os.listdir()

(c) Making a new directory: mkdir()
Syntax: os.mkdir('Newdir')

(d) Renaming a directory: rename()
os.rename('Newdir','Newname')
os.listdir()

(e) Removing a directory: remove()
os.remove('NewName')

6. Write a Python program to implement stack operations using modules.

Module definition:

```
def getStack():
   return[]
def isempty(s):
   if s==[]:
      return True
   else:
      return False
def top(s):
      if isempty(s):
         return None
      else:
         return s[len(s)-1]
def push(s,item):
      s.append(item)
def pop(s):
      if isempty(s):
         return None
      else:
         item=s[len(s)-1]
```

```
        del s[len(s)-1]
        return item
```

Program to call stack :

```
import stack

def today():
   mystack=stack.getStack()
   for item in range(1,7):
      stack.push(mystack,item)
      print('Pushing',item,'to stack')
      print ('Stack items')
   while not stack.isempty(mystack):
      item=stack.pop(mystack)
      print('Poping',item,'from stack')
```

7. Write a program to illustrate multiple modules.

```
import wordcount,ex12,ex97

def test():
   wordcount.wordCount()

def test2():
   ex12.inputNumber()

def test3():
   ex97.fun()
```

ex97.py:

```
def fun():
   try:
      x = float(raw_input("Your number: "))
      inverse = 1.0 / x
   except ValueError:
      print "You should have given either an int or a float"
   except ZeroDivisionError:
      print "Infinity"
```

ex12.py:

```
def inputNumber () :
    x = input ('Pick a number: ')
    if x == 17 :
        raise ValueError, '17 is a bad number'
    return x
```

wordcount.py:

```
def wordCount():
    cl=0
    cw=0
    cc=0
    f=open("ex88.txt","r")
    for line in f:
        words=line.split()
        cl +=1
        cw +=len(words)
        cc +=len(line)
    print('No. of lines:',cl)
    print('No. of words:',cw)
    print('No. of characters:',cc)
    f.close()
```

8. Write a Python program to dump objects to a file using pickle.

```
import pickle
def funMap():
    cities = ["Chennai", "delhi","Mumbai","Kolkata"]
    fh=open("cities.pck","w")
    pickle.dump(cities,fh)
    fh.close()
    f=open("cities.pck","r")
    cts=pickle.load(f)
    print(cts)
```