PPDS MID 1 {Q AND A}

UNIT - 1

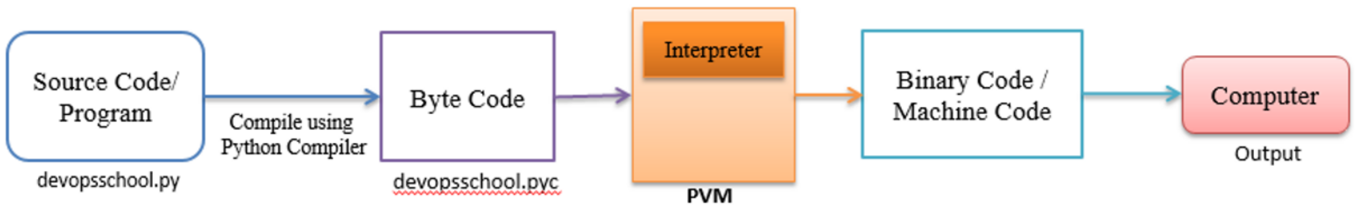
1) a) why Python is called as an interpreted Language? Briefly explain about PVM with neat diagram?

Ans: **Interpreted:** A Program code is called source code. After writing a python program we should compile the source code using python complier. Python complier translates the python program into an intermediate code called byte code. This byte code is then executed by PVM. Inside the PVM, an interpreter converts the byte code instructions into machine code so that the processor will understand and run that machine code to produce results.

Python is called an interpreted language because it goes through an interpreter, which turns code you write into the language understood by your computer's processor.

Python Virtual Machine (PVM)

- Computers understand only machine code that comprises 1s and 0s.
- A compiler normally used in programming to convert the program source code into machine code.
- A Python compiler converts the program source code into another code, called byte code. Each python program statement is converted into a group of byte code instructions.
- Byte code represents the fixed set of instructions created by python developers representing all types of operations.
- The role of Python Virtual Machine (PVM) is to convert the byte code instructions into machine code so that the computer can execute those machine code instructions and display the final output. To carry out this conversion, PVM is equipped with an interpret. The interpret converts the byte code into machine code to the computer processor for execution. Since interpreter is playing the main role, often the Python Virtual Machine is also called an interpreter.



b) Write short note on features of Python programming which are different from C programming?

Ans: **Dynamically Typed:**

- In Python, we need not to declare anything
- An assignment statement binds a name to an object, and the object can be of any type
- If a name is assign to an object of one type, it may later be assigned to an object of a different type

Plat form Independent:

- When a python program is compiled using a python compiler, it generates byte code
- Python's byte code represents a fixed set of instructions that runs on all operating systems and hardware
- Using a python virtual machine, anybody can run these byte code instructions on any computer system

Huge Library:

- Python has a big library which can be used on any operating system like UNIX, Windows or Macintosh

- Programmers can develop programs very easily using the modules available in the Python library

Scripting Language:

- A Scripting Language is a programming language that does not use a compiler for executing the source code
- Rather, it uses an interpreter to translate the source code into machine code on while running

Generally, scripting languages perform supporting tasks for a bigger applications or software

Batteries included:

- The Huge library of python contains several small applications which are already developed and immediately available to programmers
- The programmers need not to download separate packages or applications in many cases

Interpreted:

- Python complier translates the python program into an intermediate code called byte code
- Interpreter converts the byte code instructions into machine code so that the processor will understand and run that machine code to produce results

2) a) Distinguish between c and Python programming?

Ans:

S.NO	Key	C Language	Python Language
1	Definition	C is a general-purpose programming language that is extremely popular, simple and flexible. It is machine-independent, structured programming language which is used extensively in various applications.	Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language.
2	Type	**structured type** programming language and following Imperative programming model. Also it is **statically typed.**	**object-oriented** type programming language and **dynamically typed.**
3	Variable Declaration	Variables are need to be declared in C before get used in code further.	While on other hand in Python no need of variable declaration for its use.
4	Compilation	C language is compiled by the compiler hence is also known as **compiled language.**	On other hand **interpreter** is used in Python for interpreting the code and hence Python is known as Interpreted language.
5	Functions available	C language has **limited number of built-in functions** as compared to that in Python language.	On other hand Python has **a large library of built-in function** as compared to C language.
6	Execution	As mentioned in above point C is a compiled language hence its code is compiled direct to machine code which is executed directly by the CPU.	On other hand in case of Python code is firstly **compiled to a byte-code and then it is interpreted** by a large C program.

b) list of the steps used to view the byte code of a Python program?

Ans: Viewing the byte code

- Lets consider the following python program:

  #python program to add two numbers

  a = b = 10 #take two variables and store 10 in to them

  print("Sum= " ,(a+b)) # display their sum

- we can type this program in text editor like notepad and then save it as 'first.py'. it means, the first.py file contains the source code.
- Now, lets compile the program using python compiler as:

  ```
  C:\>python first.py
  ```

- It will display the results as:

  ```
  Sum = 20
  ```

- That is ok. But we do not want the output of the program .
- we want to see the byte code instructions that were created internally by the python compiler before they are executed by the PVM.
- For this purpose, we should specify the ***dis*** module while using python command as:

  ```
  C:\>python -m dis first.py
  ```

- The preceding byte code is displayed by the dis module, which is also known as ' disassembler' that displays the byte code in the human understandable format.
- If we observe the preceding code we can find 5 columns.
  - The left-most or first column represents the line number in our source program (first.py).
  - The second column represents the offset position of the byte code.
  - The third column shows the name of the byte code instructions.
  - The fourth column represents instructions argument and
  - The last column represents constants or name as specified by 4thcolumn .
- The LOAD_NAME specifies that this byte code instructions has 2 arguments .
- the name that has 2 arguments is (print)function . since there are 2 arguments .
- the next byte code instructions will represents those 2 arguments as LOAD_CONST represents the string constant name ('Sum = ') and (a) and (b) as names involved in the second arguments for the print function.
- Then BINARY_ADD instruction adds the previous (i.e., a and b)values

3) a) classify the built-in data types based on mutable and immutable. Explain any one data type of each with example.

Ans:

MUTABLE	IMMUTABLE
Object can be changed after created.	Object cannot be changed once created.
list dictionary set	int, float, long, complex string tuple bool

List:

- Lists are collections of items (strings, integers, or even other lists).
- Each item in the list has an assigned index value.
- Lists are enclosed in [ ]
- Each item in a list is separated by a comma

  Ex: list = [10, -20 , 15.5, ‘vijay’, “mary”]
- An empty list is created using just square brackets:

  emptyList = [ ]
- Accessing elements in a list is called indexing.

  list = ["first", "second", "third"]

  list[0] = "first"

Tuple:

- A tuple is a collection which is ordered and unchangeable.
- A tuple is similar to a list.
- A tuple contains a group of elements which can be of different types.
- The elements in the tuple are separated by commas and enclosed in parenthesis ().
- A tuple is a collection of objects which ordered and immutable

  Ex: thistuple = ("apple", "banana", "cherry")
- You can access tuple items by referring to the index number, inside square brackets:

  thistuple = ("apple", "banana", "cherry")

  thistuple[1]= apple

b) Describe list comprehension with an example program?

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, meaning you can put in all kinds of objects 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.
- **The basic syntax is**

  [expression for item in list if conditional]
- **This is equivalent to**:

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

Example:

```
x = [i for i in range(10)]
print x
output: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- If you used to do it like this:

```
new_list = []
for i in old_list:
   if filter(i):
      new_list.append(expressions(i))
print(new_list)
output:[0,1,2,3,4,5,6,7,8,9]
```

- You can obtain the same thing using list comprehension:

```
new_list = [expression(i) for i in old_list if filter(i)]
print(new_list)
output: [0,1,2,3,4,5,6,7,8,9]
```

4 a) **Explain the any 3 built-in data types and use of constants, identifiers, reserve words and naming conventions?**

Ans: str Data type:

- In python, str represents string datatype.
- A string is represented by a group of characters.
- Strings are enclosed in single quotes or double quotes. Both are valid.
- Python strings are immutable

Str = “welcome”
Str = ‘welcome’

Ex:

```
word = "Hello World"
>>> print (word)
Hello World
```

Floating Point Numbers:

- Floating point numbers or *floats*
- Floats are decimals, positive, negative and zero.
- Floats can also be represented by numbers in scientific notation which contain exponents.

- a float can be defined using a decimal point . when a variable is assigned.

Ex:

```
>>> c = 6.2
>>> type(c)
<class 'float'>
```

Complex Numbers

- A complex number is defined in Python using a real component + an **imaginary component j**.
- The letter j must be used to denote the imaginary component.
- Using the letter i to define a complex number returns an error in Python.

Ex:

```
>>> comp = 4 + 2j
>>> type(comp)
<class 'complex'>
```

USES OF constants, identifiers, reserve words and naming conventions:

Constants:

- A constant is a type of variable whose value cannot be changed.
- It is helpful to think of constants as containers that hold information which cannot be changed later.
- You can think of constants as a bag to store some books which cannot be replaced once placed inside the bag.

Identifiers:

- An identifier is a user-defined name to represent a variable, a function, a class, a module, or any other object.
- It is a programmable entity in Python- one with a name.
- It is a name given to the fundamental building blocks in a program.

reserve words:

- **Python** Server Side Programming Programming **Reserved words** (also called keywords) are defined with predefined meaning and syntax in the language.

- These keywords have to be **used** to develop programming instructions.

- **Reserved words** can't be **used** as identifiers for other programming elements like name of variable, function etc.

naming conventions:

- **Naming conventions** are a set of rules or best practices that are suggested to be followed while naming an identifier.

- The benefits of following the **naming conventions** are: They help us in understanding what a particular identifier is.

b) list out the commands used to install the python packages and explain what are the advantages of using packages

Ans:

Commands to install python packages:

Numpy:

– pip install numpy

Pandas:

– Pip install pandas

Matplotlib:

- pip install matplotlib

<u>commands to verify installation of Python packages.</u>

Numpy:

- pip show numpy

Pandas:

- Pip show pandas

Matplotlib:

```
>>> import matplotlib
>>> matplotlib. __version__
'3.1.1'
```

<u>Advantages of Python packages</u>:

Numpy:

- (Numerical Python) is an open-source library for the Python programming language.
- It is used for *scientific computing and working with arrays*.

Pandas:

- Pandas provide essential data structures like series, dataframes, and panels which help in manipulating data sets and time series.
- to get started with Data Science, to analyze huge sets of data, And to manipulate spreadsheets and CSVs with just a few lines of code, we use Pandas libraries

Matplotlib:

- Matplotlib is a popular Python package used to build plots.
- Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python
- plots included in scientific publications and presentations are generated.

UNIT 2

1 **Demonstrate the following with an example**

i) Membership operator ii) identity operator iii) operator precedence and associativity

Ans:

i) **Membership operator:**

- in and not in are the membership operators in Python.

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

- Example:

```
x = 'Hello world'
y = {1:'a',2:'b'}
print('H' in x)
                        Output: True
print('a' in y)
                        Output: False
```

ii) identity operator:

- is and is not are the identity operators in Python.
- They are used to check if two values (or variables) are located on the same part of the memory.
- Two variables that are equal does not imply that they are identical.

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:

```
x1 = 5
y1 = 5
x2 = 'Hello'
y2 = 'Hello'
x3 = [1,2,3]
y3 = [1,2,3]
print(x1 is not y1)
                        Output: False
print(x2 is y2)
                        Output: True
```

iii) operator precedence and associativity:

- The combination of values, variables, operators, and function calls is termed as an expression.
- The Python interpreter can evaluate a valid expression.
- To evaluate these types of expressions there is a rule of precedence in Python. It guides the order in which these operations are carried out.

Operators	Meaning
()	Parentheses
**	Exponent
+x, -x, ~x	Unary plus, Unary minus, Bitwise NOT
*, /, //, %	Multiplication, Division, Floor division, Modulus
+, -	Addition, Subtraction
<<, >>	Bitwise shift operators
&	Bitwise AND
^	Bitwise XOR
|	Bitwise OR
==, !=, >, >=, <, <=, is, is not, in, not in	Comparisons, Identity, Membership operators
not	Logical NOT
and	Logical AND
or	Logical OR

Multiplication has higher precedence than subtraction

```
>>> 10 - 4 * 2
Output: 2
```

Associativity is the order in which an expression is evaluated that has multiple operators of the same precedence. **Almost all the operators have left-to-right** associativity.

When **two operators have the same precedence**, associativity helps to determine the order of operations.

For example, multiplication and floor division have the same precedence. Hence, if both of them are present in an expression, the left one is evaluated first

Left-right associativity

```
print(5 * 2 // 3)

# Output: 3
```

2 a) **Describe about input and output statements with a sample program for each type of statement**

Ans: input statement: The data given to the computer is called input

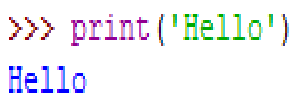
- To accept input from keyboard, python provides the input () functions.
- This function takes a value from the keyboard and returns it as a string .
- Once the value comes into the variable ‘str’, it can be converted into ‘int’ or ‘float’ etc. this is useful to accept number as:

Example:

```
x=int(input('enter a number:'))
#enter a number: 125
print(x)
125
```

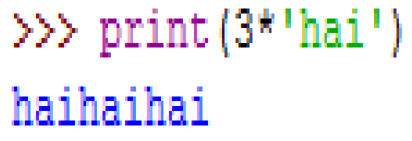
output statements: The results returned by the computer are called output.

- The display output or results, python provides the print () function. This function can be used in different formats which are discussed here under.



```
>>> print('Hello')
Hello
```

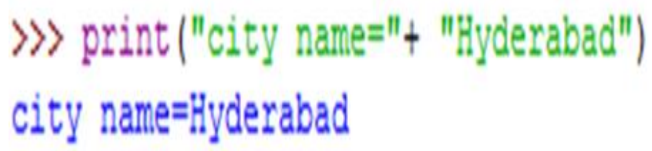
- ‘\n’ indicates new line. ‘\t’ represents tab space.
- \\ n displays ‘\n’ and \\t displays ‘\t’.
- We can use repetition operator (*) to repeat the strings in the output as:



```
>>> print(3*'hai')
haihaihai
```

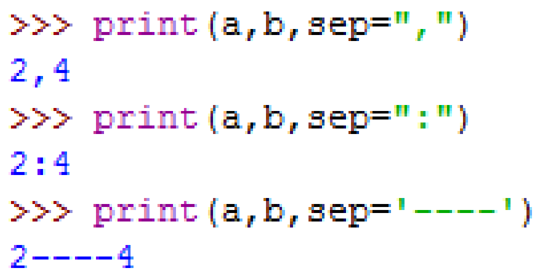
The operator ‘+’ when used on numbers will perform addition operation.

- Hence ‘+’ is called concatenation (joining) operator when used on strings.



```
>>> print("city name="+ "Hyderabad")
city name=Hyderabad
```

- The format is sep=”characters” which are used to separate the values in the output.



```
>>> print(a,b,sep=",")
2,4
>>> print(a,b,sep=":")
2:4
>>> print(a,b,sep='----')
2----4
```

b) **Write a program to Reverse The given Digits using python programming**

Ans:

code:

```
Number = int(input("Please Enter any Number: "))

Reverse = 0

while(Number > 0):

    Reminder = Number %10

    Reverse = (Reverse *10) + Reminder

     Number = Number //10

print("\n Reverse of entered number is = %d" %Reverse)
```

output:

```
Please Enter any Number: 1456



 Reverse of entered number is = 6541
```

3 a) **Write a Python program to count the number of even and odd numbers from a sequence of numbers.**

Ans: code:

```
x=int(input("enter the range of numbers:"))
i=1
even=0
odd=0
while i<x:
    if(i%2==0):
        even=even+1
    else:
        odd=odd+1
    i+=1
print("count of even numbers :",even)
print("count of odd numbers:", odd)
```

output:

```
enter the range of numbers:10
count of even numbers : 4
count of odd numbers: 5
```

b) **Illustrate the for and while loop with an example program**

Ans: <u>while loop</u>: Executes a group of statements as long as a condition is True.

good for *indefinite loops* (repeat an unknown number of times)

Syntax:

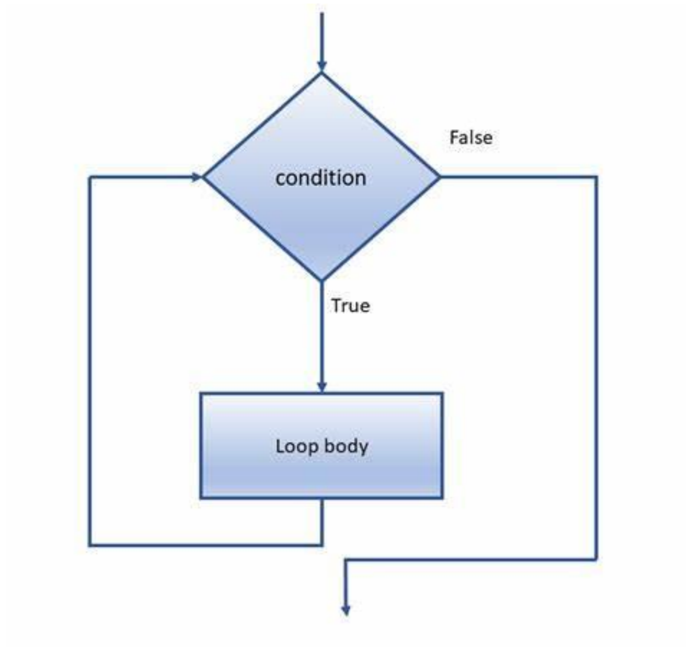
```
while condition:
        statements
```

Example:

```
number = 1
while number < 200:
   print (number)
   number = number * 2
```

Output:

```
1 2 4 8 16 32 64 128
```



for Loop Statements: The for loop is useful to iterate over the elements of a sequence.

- It means, the for loop can be used to execute a group of statements repeatedly depending upon the number of elements in the sequence.
- The for loop can work with sequence like list, tuple, range etc.

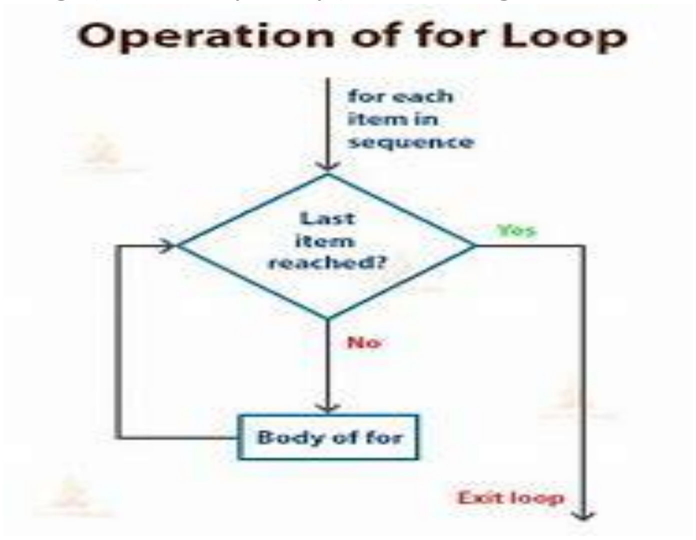
Syntax:

```
for var in sequence:
        Statements
```

- **Range**

The range function specifies a range of integers:
range(*start, stop*) - the integers between *start* (inclusive) and *stop* (exclusive)
It can also accept a third value specifying the change between values.

range(*start*, *stop*, *step*) - the integers between *start* (inclusive) and *stop* (exclusive) by *step*



Example:

```
for x in range(5, 0, -1):
    print x
print "Blastoff!"
```

Output:

```
5       4       3       2       1
Blastoff!
```

4 **a) What is an else suite? When this feature is helpful, explain with a sample program.**

Ans: The else suite:

In python, it is possible to use 'else' statement along with for loop or while loop in the form shown in table

for with else	while with else

for(var in sequence):
        statements
else:
        Statements
	while(condition):
	        statements
	else:
	        statements

- The statements written after 'else' are called suite.
- The else suite will be always executed irrespective of the statements in the loop are executed or not.

Example:

```
for i in range(0):
        print("yes")
else:
        print("no")
 output:
        no
```

example,

```
        count =0
        while count < 5:
                print(count , "is less than 5")
                count+=1
        else:
                print(count, "is not less than 5")
```

b) Explain the following statements with flowchart and sample program

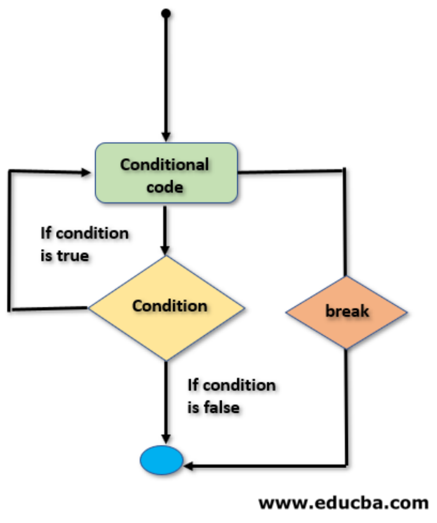
i) break   ii) continue   iii) pass   iv) Assert

Ans:

i) break: The break statement can be used inside a for loop or while loop to come out of the loop.

When 'break ' is executed the python interpreter jumps out of the loop to process the next statement in the program.

```
x=10
while x>=1:
        print(' x=',x)
        x-=1
        if x==5:
                break
print("out of loop")
```



output:

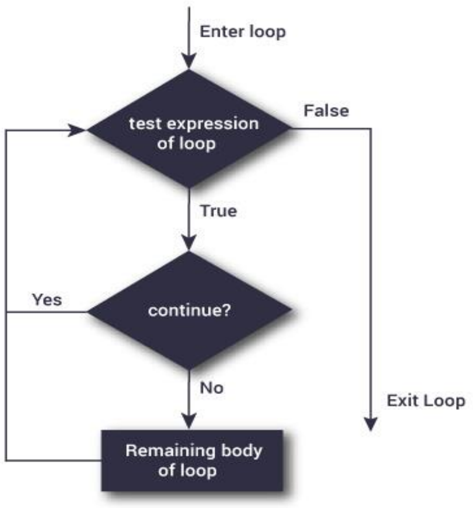
```
10      9       8       7       6       5
```

ii) continue: The continue statement is used in a loop to go back to the beginning of the loop.

- It means, when continue is executed, the next repetition will start, when continue is executed, the subsequent statement in the loop are executed.

Example:

```
x=0
while x<10:
        x+=1
        if x>5:
                continue
        print('x=',x)
print("out of loop")
```



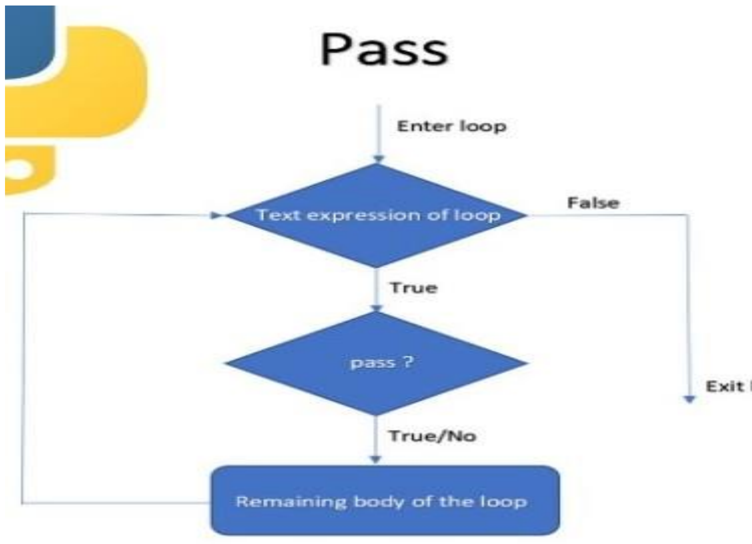
Output:

```
x= 1    x= 2    x= 3    x= 4    x= 5 out of loop
```

iii) pass: The pass statement does not do anything. It is used with 'if' statement or inside a loop to represent no operation.

- We use pass statement when need a statement syntactically but we do not want to do any operation.

```
num = [1,2,3,-4,-5,-6,-7,-8,9]
for i in num:
        if i>0:
                pass
        else:
                print(i)
```



output:

```
-4      -5      -6      -7      -8
```

iv) Assert: The assert statement is useful to **check if a particular condition is fulfilled or not**. The syntax is as follows:

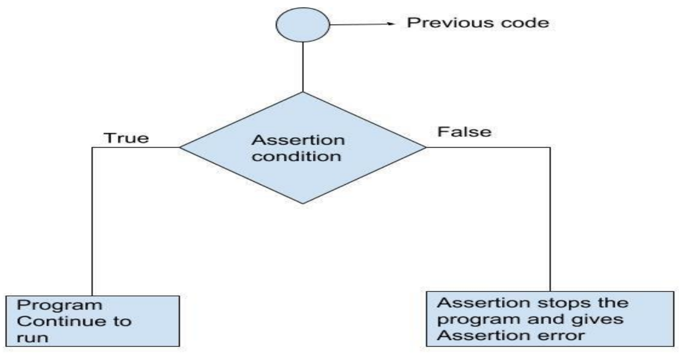
assert expression, message

in the above syntax, the 'message' is not compulsory

```
x = int(input('Enter a number greater than 0:   '))
        try:
                assert(x>0)
        print('U entered:  ',x)
    except  AssertionError:
        print("Wrong input entered")
```

Output:

```
Enter a number greater than 0:   -5
```



```
    Wrong input entered
```

Unit III

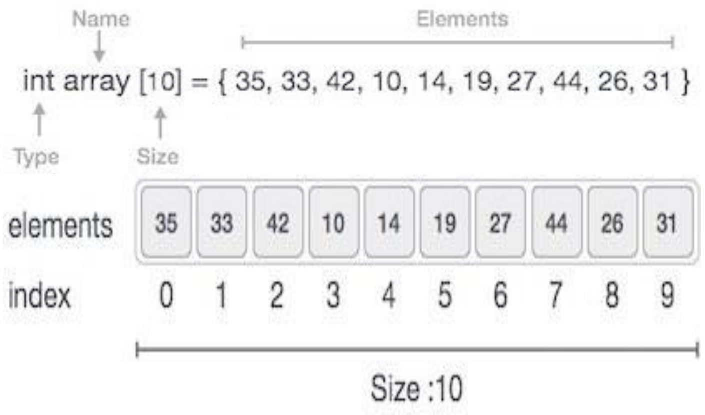
1 a) What is an array? Can we use List as a substitute of arrays. Justify.

Ans:

- An array is a collection of items stored at contiguous memory locations
- The idea is to store multiple items of the same type together
- Arrays are used to store multiple values in one single variable
- Element– Each item stored in an array is called an element.
- Index − Each location of an element in an array has a numerical index, which is used to identify the element.

Array Representation

- Arrays can be declared in various ways in different languages. Below is an illustration.



Yes, we use List as a substitute of arrays.

- We can treat lists as arrays. However, we cannot constrain the type of elements stored in a list.
- For example:

```
a = [1, 3.5, "Hello"]
```

- If you create arrays using array module, all elements of the array must be of the same numeric type.
- import array as arr
- a = arr.array('d', [1, 3.5, "Hello"])   // Error

b) How to use array module in the program? What are the advantages of using Array module explain in detail.

Ans: Creating an Array

- Array is created in Python by importing **array module** to the python program.
- Then the array is declared as shown below.

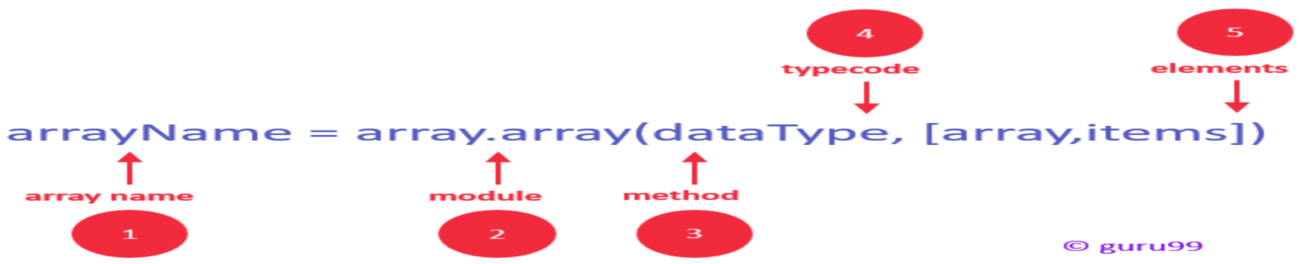
```
from array import *
arrayName = array(typecode, [Initializers])
                    or
import array
arrayName = array.array(type code, [array,items])
```

- **Array Syntax**

1. **Identifier**: specify a name like usually, you do for variables
2. **Module**: Python has a special module for creating array in Python, called "array" – you must import it before using it
3. **Method**: the array module has a method for initializing the array. It takes two arguments, type code, and elements.
4. **Type Code**: specify the data type using the type codes available (see list below)
5. **Elements**: specify the array elements within the square brackets, for example [130,450,103]

6.Typecode are the codes that are used to define the type of value the array will hold. Some common typecodes used are

Type code	C Type	Python Type	Minimum size in bytes
'b'	signed char	int	1
'B'	unsigned char	int	1
'u'	Py_UNICODE	Unicode character	2



Advantages

- In an array, accessing an element is very easy by using the index number.
- The search process can be applied to an array easily
- 2D Array is used to represent matrices
- For any reason a user wishes to store multiple values of similar type then the Array can be used and utilized efficiently

2 **Explain the following with an simple program**

i) Creating an array ii) Accessing array elements

ii) Different ways to add an element in to array

iv) Different ways to delete an element from array v)Searching an element

Ans:

i) Creating an array: lets create and print an array using python.

```
from array import *
array1 = array('i', [10,20,30,40,50])
for x in array1:
        print(x)
```

```
output:
10      20      30      40      50
```

ii) Accessing array elements: We can access each element of an array using the index of the element.

The below code shows how

```
from array import *
array1 = array('i', [10,20,30,40,50])
print (array1[0])
print (array1[2])
```

```
output:
10      30
```

iii) Different ways to add an element in to array :

Insert operation is to insert one or more data elements into an array.

We can add one item to a list using append() method or add several items using extend()method.

```
import array as arr
 numbers = arr.array('i', [1, 2, 3])
 numbers.append(4)
print(numbers)
```

```
Output:
 [1, 2, 3, 4]
```

```
# extend() appends iterable to the end of the array
numbers.extend([5, 6, 7])
print(numbers)
Output:
[1, 2, 3, 4, 5, 6, 7]
```

iv) Different ways to delete an element from array :

- Deletion refers to removing an existing element from the array and re-organizing all elements of an array.
- Here, we remove a data element at the middle of the array using the python in-built remove() method.

```
from array import *
array1 = array('i', [10,20,30,40,50])
array1.remove(40)
for x in array1:
        print(x)
output:
10   20    30    50
```

v)Searching an element:

- You can perform a search for an array element based on its value or its index.
- Here, we search a data element using the python in-built **index()** method.

```
from array import *
array1 = array('i', [10,20,30,40,50])
print (array1.index(40))
output:
   40
```

3 **A)Mention the advantages of using Numpy module instead of array module?**

Ans: the advantages of NumPy:

- The core of Numpy is its arrays. One of the main advantages of using Numpy arrays is that they take less memory space and provide better runtime speed when compared with similar data structures in python(lists and tuples).
- Numpy support some specific scientific functions such as linear algebra. They help us in solving linear equations.

- Numpy support vectorized operations, like elementwise addition and multiplication, computing Kronecker product, etc. Python lists fail to support these features.
- It is a very good substitute for MATLAB, OCTAVE, etc as it provides similar functionalities and supports with faster development and less mental overhead(as python is easy to write and comprehend)
- NumPy is very good for data analysis.

b) Write short notes on the following

i) arrange ii) reshape iii) array of zeros and ones iv) dimensions v) slicing

Ans:

i) arrange: The **arrange** () function of Python **numpy** class returns an array with equally spaced elements as per the interval where the interval mentioned is half opened, i.e. [Start, Stop).

```
import numpy as np
 A = np.arange(4)
print('A =', A)

B = np.arange(12).reshape(2, 6)
print('B =', B)

'''
Output:
A = [0 1 2 3]
B = [[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]]
'''
```

ii) reshape: Reshaping means changing the shape of an array. The shape of an array is the number of elements in each dimension.

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
newarr = arr.reshape(4, 3)
print(newarr)
 output:
[[ 1 2 3]
[ 4 5 6]
[ 7 8 9]
[10 11 12]]
```

iii) array of zeros and ones:

produce the ones array (np.ones)
produce the zeros array (np.zeros)

```
import numpy as np
zeros_array = np.zeros( (2, 3) )
print(zeros_array)

 Output:
 [[0. 0. 0.]
  [0. 0. 0.]]

ones_array = np.ones( (1, 5), dtype=np.int32 )
print(ones_array)

Output: [[1 1 1 1 1]]
```

iv) dimensions:

One Dimension: Many arrays have only one dimension, such as the number of people of each age.
Two Dimensions: Some arrays have two dimensions, such as the number of offices on each floor of each building on a...
Three Dimensions: A few arrays have three dimensions, such as values in three-dimensional space. Such an array uses...

- More than Three Dimensions. Although an array can have as many as 32 dimensions, it is rare to have more than three.

v) slicing: Slicing of a Matrix

Slicing of a one-dimensional NumPy array is similar to a list.
Let's take an example:

```
import numpy as np
letters = np.array([1, 3, 5, 7, 9, 7, 5])
 print(letters[2:5])
Output: [5, 7, 9]
```

4 **a) Define function. Create a simple user defined function to find the student percentage scored in 6 subject marks. (Maximum marks for each subject 60).**

Ans: Functions: In Python, a function is a group of related statements that performs a specific task.Functions help break our program into smaller and modular chunks. As our program grows larger and larger, functions make it more organized and manageable.

Furthermore, it avoids repetition and makes the code reusable.

a simple user defined function to find the student percentage scored in 6 subject marks.

Code:

```
 name=input("enter the student name:")

a=int(input("enter 1 subject marks:"))

b=int(input("enter 2 subject marks:"))

c=int(input("enter 3 subject marks:"))

d=int(input("enter 4 subject marks:"))

e=int(input("enter 5 subject marks:"))

f=int(input("enter 6 subject marks:"))

def stud_per(name,a,b,c,d,e,f):

    per=((a+b+c+d+e+f)/360)*100

    return per;

print(name,"percentage is ",stud_per(name,a,b,c,d,e,f))
```

output:

```
enter the student name:sai
enter 1 subject marks:50
enter 2 subject marks:50
enter 3 subject marks:50
enter 4 subject marks:50
enter 5 subject marks:50
enter 6 subject marks:50
sai percentage is  83.33333333333334
```

b) Explain different types of formal arguments with a sample program

Ans: You can call a function by using the following types of formal arguments-

- Required arguments
- Keyword arguments
- Default arguments
- Variable-length arguments

Required Arguments

- Required arguments are the arguments passed to a function in **correct positional order.**
- Here, the number of arguments in the function call should match exactly with the function definition.

```
def sum (a,b):
   return a+b;
a = int(input("Enter a: "))
b = int(input("Enter b: "))
   print("Sum = ",sum(a,b))
```

Output:

```
Enter a: 10
Enter b: 20
Sum = 30
```

Keyword Arguments

- Keyword arguments are related to the function calls. When you use keyword arguments in a function call, the caller identifies the arguments by the parameter name.
- This allows you to skip arguments or place them out of order because the Python interpreter is able to use the keywords provided to match the values with parameters

```
def student(firstname, lastname):
        print(firstname, lastname)
student(firstname=‘Mani ', lastname='Pal')
student(lastname='Pal', firstname=‘Mani')
```

Output:

```
Mani Pal
Mani Pal
```

Variable-length Arguments

- You may need to process a function for more arguments than you specified while defining the function. These arguments are called *variable-length arguments and are not named in* the function definition, unlike required and default arguments.

```
def printinfo( arg1, *vartuple ):
                 print ("Output is: " )
                print (arg1)
```

```
        for var in vartuple:
            print (var)
        return
    printinfo( 10 )
    printinfo( 70, 60, 50 )
```

```
Output is:
10
Output is:
70 60 50
```

Default Parameter Value

- The following example shows how to use a default parameter value.
- If we call the function ***without parameter***, it uses the default value:

```
    def my_function(country = "Norway"):
    print("I am from " + country)
my_function("Sweden")
my_function("India")
my_function()
my_function("Brazil")
```

Output:

```
I am from Sweden
I am from India
I am from Norway
I am from Brazil
```

Unit IV

1 **a) Explain built-in functions with suitable examples?**

Ans:

1: abs() : Returns the absolute value of a number

Code:

```
x = abs(-7.25)

print(x)
```

Output:

```
7.25
```

2: bool() : Returns the boolean value of the specified object

Code:

```
x = bool(1)

print(x)
```

Output:

```
True
```

3: complex() : Returns a complex number

Code:

```
x = complex(3, 5)

print(x)
```

Output:

```
(3+5j)
```

4: dict() : Returns a dictionary (Array)

Code:

```
x = dict(name = "John", age = 36, country = "Norway")

print(x)
```

Output:

```
{'name': 'John', 'age': 36, 'country': 'Norway'}
```

5 : filter() : Use a filter function to exclude items in an iterable object

Code:

```
ages = [5, 12, 17, 18, 24, 32]
def myFunc(x):
   if x < 18:
        return False
   else:
        return True
adults = filter(myFunc, ages)
for x in adults:
   print(x)
```

Output:

```
18   24   32
```

6 : float() : Returns a floating point number

Code:

```
x = float(3)

print(x)
```

Output:

```
3.0
```

7 : format() : Formats a specified value

Code:

```
x = format(0.5, '%')

print(x)
```

Output:

```
50.000000%
```

8 : input() : Allowing user input

Code:

```
print("Enter your name:")

x = input()

print("Hello, " + x)
```

Output:

```
Enter your name:

sai teja


Hello, sai teja
```

9 : int() : Returns an integer number

Code:

```
x=int(3.5)
print(x)
```

Output:

```
3
```

10: len():Returns the length of an object

Code:

```
mylist = ["apple", "orange", "cherry"]
x = len(mylist)
print(x)
```

Output:

```
3
```

b) Describe the advantages of user defined functions?

Ans:

Advantages of user-defined functions

1. User-defined functions help to decompose a large program into small segments which makes program easy to understand, maintain and debug.
2. If repeated code occurs in a program. Function can be used to include those codes and execute when needed by calling that function.
3. Programmars working on large project can divide the workload by making different functions.

Example of a user-defined function

```
def add_numbers(x,y):
  sum = x + y
  return sum
num1 = 5
num2 = 6
print("The sum is", add_numbers(num1, num2))
```

Output

The sum is 11

2 **a) Demonstrate function recursion with an example program**
Ans:

Recursion

Python also accepts function recursion, which means a defined function can call itself.

Recursion is a common mathematical and programming concept. It means that a function calls itself. This has the benefit of meaning that you can loop through data to reach a result.

The developer should be very careful with recursion as it can be quite easy to slip into writing a function which never terminates, or one that uses excess amounts of memory or processor power. However, when written correctly recursion can be a very efficient and mathematically-elegant approach to programming.

Advantages of using recursion

- A complicated function can be split down into smaller sub-problems utilizing recursion.
- Sequence creation is simpler through recursion than utilizing any nested iteration.
- Recursive functions render the code look simple and effective.

Disadvantages of using recursion

- A lot of memory and time is taken through recursive calls which makes it expensive for use.
- Recursive functions are challenging to debug.
- The reasoning behind recursion can sometimes be tough to think through.

Syntax:

```
def func(): <--
           |
           | (recursive call)
           |
   func() ----
```

example:
# Program to print the fibonacci series upto n_terms

```
def recursive_fibonacci(n):
  if n <= 1:
     return n
  else:
     return(recursive_fibonacci(n-1) + recursive_fibonacci(n-2))
n_terms = 10
if n_terms <= 0:
  print("Invalid input ! Please input a positive value")
else:
  print("Fibonacci series:")
  for i in range(n_terms):
    print(recursive_fibonacci(i))
```

Output:

```
Fibonacci series:
0
1
1
2
3
5
8
13
21
34
```

Tail-Recursion: A unique type of recursion where the last procedure of a function is a recursive call. The recursion may be automated away by performing the request in the current stack frame and returning the output instead of generating a new stack frame. The tail-recursion may be optimized by the compiler which makes it better than non-tail recursive functions.

Example:

```
def Recur_facto(n):
   if (n == 0):
      return 1
   return n * Recur_facto(n-1)
print(Recur_facto(6))
```

Output:

```
720
```

3 a) Demonstrate the types of formal arguments in functions with a sample program

Ans: **Arguments**

Information can be passed into functions as arguments.

Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

You can call a function by using the following types of formal arguments-

– Required arguments
– Keyword arguments
– Default arguments
– Variable-length arguments

Required Arguments

- Required arguments are the arguments passed to a function in **correct positional order.**

- Here, the number of arguments in the function call should match exactly with the function definition.

Example:

```
def sum (a,b):
   return a+b;

a = int(input("Enter a: "))
b = int(input("Enter b: "))

print("Sum = ",sum(a,b))
```

```
Output:
Enter a: 10
Enter b: 20
Sum = 30
```

Keyword Arguments

- Keyword arguments are **related to the function calls**. When you use keyword arguments in a function call, the caller identifies the arguments by the parameter name.
- This allows you to skip arguments or place them out of order because the Python interpreter is able to use the keywords provided to match the values with parameter.

Example:

```
def student(firstname, lastname):

        print(firstname, lastname)

student(firstname='Mani ', lastname='Pal')

student(lastname='Pal', firstname='Mani')
```

Output:

```
Mani Pal

Mani Pal
```

Default Parameter Value

- The following example shows how to use a default parameter value.
- If we call the function ***without parameter***, it uses the default value:

Example:

```
def my_function(country = "Norway"):
print("I am from " + country)

my_function("Sweden")
my_function("India")
my_function()
my_function("Brazil")
```

Output:

```
I am from Sweden
I am from India
I am from Norway
I am from Brazil
```

Variable-length Arguments

- You may need to process a function for more arguments than you specified while defining the function. These arguments are called *variable-length arguments and are not named in* the function definition, unlike required and default arguments.
- An asterisk (*) is placed before the variable name that holds the values of all **nonkeyword** variable arguments.
- This tuple remains empty if no additional arguments are specified during the function call.

Example:

```
def printinfo( arg1, *vartuple ):

    "This prints a variable passed arguments"

    print ("Output is: " )

    print (arg1)

    for var in vartuple:

        print (var)

    return

printinfo( 10 )

printinfo( 70, 60, 50 )
```

Output is:

```
10

70 60 50
```

b) In python functions are always called by passing reference, Justify.

Ans: Python utilizes a system, which is known as “Call by Object Reference” or “Call by assignment”. In the event that you pass arguments like whole numbers, strings or tuples to a function, the passing is like call-by-value because you can not change the value of the immutable objects being passed to the function. Whereas passing mutable objects can be considered as call by reference because when their values are changed inside the function, then it will also be reflected outside the function.

An immutable variable cannot be changed once created. If we wish to change an immutable variable, such as a string, we must create a new instance and bind the variable to the new instance. Whereas, mutable variable can be changed in place.

4 **a) Define user defined function.**

Ans: A function is a set of statements that take inputs, do some specific computation and produce output. The idea is to put some commonly or repeatedly done tasks together and make a function so that instead of writing the same code again and again for different inputs, we can call the function.

Functions that readily comes with Python are called built-in functions. Python provides built-in functions like print(), etc. but we can also create your own functions. These functions are known as **user defines functions.**

Syntax:

```
def function_name(parameters):
                                        """docstring"""          # optional
                                        statement(s)
                                        return <expression>     #optional
```

Creating a Function

In Python a function is defined using the def keyword:

```
def my_function():
    print("Hello from a function")
```

Calling a Function

To call a function, use the function name followed by parenthesis:

```
def my_function():
    print("Hello from a function")
my_function()
```

example:

```
def evenOdd( x ):
   if (x % 2 == 0):
      print("even")
   else:
      print("odd")
evenOdd(2)
evenOdd(3)
```

Output:

```
even
odd
```

Docstring

- The first string after the function is called the Document string or Docstring in short.
- This is used to describe the functionality of the function.

- The use of docstring in functions is optional but it is considered a good practice.

  Syntax:

  ```
  print(function_name.__doc__)
  ```

example:

```
def say_Hi():
        “’Hello!  ‘“
print(say_Hi.__doc__)
```

Output:

```
Hello!
```

b) Create a simple user defined function to find the student percentage scored in 6 subject marks. (Maximum marks for each subject 50)

Ans:

Code:

```
name=input("enter the student name:")
a=int(input("enter 1 subject marks:"))
b=int(input("enter 2 subject marks:"))
c=int(input("enter 3 subject marks:"))
d=int(input("enter 4 subject marks:"))
e=int(input("enter 5 subject marks:"))
f=int(input("enter 6 subject marks:"))
def stud_per(a,b,c,d,e,f):
   per=((a+b+c+d+e+f)/300)*100
   return per;
print(name,"percentage is ",stud_per(a,b,c,d,e,f))
```

Output:

```
enter the student name: sai teja
enter 1 subject marks:45
enter 2 subject marks:45
enter 3 subject marks:45
enter 4 subject marks:45
enter 5 subject marks:45
enter 6 subject marks:45
sai teja percentage is  90.0
```

5 a) **What are Anonymous functions in python?**

Ans: Anonymous function:

- Python anonymous functions are also known as Lambda function. The keyword used is lambda and they are written in one line and it can have only one expression. These functions don’t need to be defined or named and they are for one-time usage.

- Python Anonymous functions are popular among advance users as it saves time, effort & memory. though the code becomes less readable but works perfectly fine.

- An anonymous function means that a **function is without a name**

- A lambda function can take any number of arguments, but can only have one expression.

- As we already know the **def** keyword is used to define the normal functions and the **lambda** keyword is used to create anonymous functions

- The anonymous function contains a small piece of code.

The syntax to define an anonymous function is given below.

lambda arguments: expression

Program to show the use of lambda functions

```
double = lambda x: x * 2
print(double(5))
```

Output

```
10
```

In the above program, `lambda x: x * 2` is the lambda function. Here `x` is the argument and `x * 2` is the expression that gets evaluated and returned.

This function has no name. It returns a function object which is assigned to the identifier `double`.

We can now call it as a normal function.

b) Demonstrate the use of Lambda functions with a sample code.
Ans:
Lambda function:

A lambda function is a small anonymous function.
A lambda function can take any number of arguments, but can only have one expression.

Syntax

lambda arguments : expression

example:

```
x = lambda a: a + 10
print(x(5))
```

Output:

```
15
```

Use of Lambda functions: The power of lambda is better shown when you use them as an anonymous function inside another function. Say you have a function definition that takes one argument, and that argument will be multiplied with an unknown number.

EXAMPLE:

```
def myfunc(n):
  return lambda a : a * n
```

```
mydoubler = myfunc(2)

print(mydoubler(11))
```

Output:

```
22
```

Use with filter()

- The filter() function in Python takes in a function and a list as arguments.
- The function is called with all the items in the list and a new list is returned which contains items for which the function evaluates to True.
- Here is an example use of filter() function to filter out only even numbers from a list.

Example:

```
my_list = [1, 5, 4, 6, 8, 11, 3, 12]
new_list = list(filter(lambda x: (x%2 == 0) , my_list))
print(new_list)
```

Output:

```
[4, 6, 8, 12]
```

Use with map()

- The map() function in Python takes in a function and a list.
- The function is called with all the items in the list and a new list is returned which contains items returned by that function for each item.
- Here is an example use of map() function to double all the items in a list.

Example:

```
 my_list = [1, 5, 4, 6, 8, 11, 3, 12]
new_list = list(map(lambda x: x * 2 , my_list))
print(new_list)
```

Output:

```
[2, 10, 8, 12, 16, 22, 6, 24]
```

Unit V

1 **a) Define the error. Distinguish between Compile Time Errors, Run-time Error and Logical errors with example.**

Ans:

A software developer came across a lot of errors either in design of the software or in writing the code. The errors in the software are called 'bugs' and the process of removing them is called 'debugging'

In general, we can classify errors in a program into one of these three types:

1. Compile-time errors
2. Run-time errors
3. Logical errors

1.Compile-time errors

There are syntactical errors found in the code, due to which a program fails to compile. For example, forgetting a colon in the statements like if, while, for, def etc. will result in compile-time error. Such errors are detected by python compiler and the line number along with error description is displayed by the python compiler. Let's see program 1 to understand this better. In this program, we have forgotten to write colon in the if statement, after the condition. This will raise SyntaxError.

- Missing Parenthesis (})
- Printing the value of variable without declaring it
- Missing semicolon (terminator)

Program:

```
x = 10
if x%2==0:
   print(x,'is divisible by 2')
      print(x,'is even number')
```

Output:

```
C:\>python ex.py
  File "ex.py", line 5
     print(x,'is even number')
     ^
IndentationError: unexpected indent
```

Runtime Errors

When PVM cannot execute the byte code, it flags runtime error. For example, insufficient memory to store something or inability of the PVM to execute some statement come under runtime errors. Runtime errors are not detected by the python compiler. They are detected by the PVM, only at runtime. The following program explains this further:

Some common runtime errors:

- Division by Zero.
- ArrayIndexOutOfBounds Exception.
- StringIndexOutOfBounds Exception.
- Accessing an array out of bounds.
- Improper formatting specifiers in printf and scanf statements.
- Stack overflow.

Example:

```
animal = ['dog','cat','horse','donkey']
print(animal[4])
```

Output:

```
C:\>python ex.py
Traceback (most recent call last):
  File "ex.py", line 3, in <module>
   print(animal[4])
IndexError: list index out of range
```

Logical Errors

These errors depict flaws in the logic of the program. The programmer might be using a wrong formula or the design of the program itself. Logical errors are not detected either by python compiler or PVM. The programmer is solely **responsible** for them. In the following program, the programmer wants to calculate incremented salary of an employee, but he gets wrong output, since he uses wrong formula.

Some common types of mistakes by the programmer leading to logical errors are:

- Wrong indentation of the code.
- Solving the expression based on wrong operator precedence.
- Using wrong variable name
- Using the wrong operator to perform the operation.
- Some type errors that lead to data loss in the program.

Example: A python program to increment the salary of an employee by 15%.

```
 def increment(sal):
   sal = sal *15/100
   return sal
sal = increment(5000.00)
print('Incremented Salary: %.2f'%sal)
```

Output:

```
C:\>python ex.py
Incremented Salary: 750.00
```

By comparing the output of a program with manually calculated results, a programmer can guess the presence of a logical error. In program , we are using the following formula to calculate the incremented salary:

```
                    sal =  sal  *   15/100
```

This is wrong since the formula calculates only the increment but it is not adding in the original salary. So, the correct formula would be:

```
                    sal =  sal  +  sal   * 15/100
```

2 **a) Why exception handling is more important in Python? Briefly explain try-except-else-finally block.**

Ans: Exception Handling:

The purpose of handling errors is to make the program robust. The word 'robust' means 'strong'. A robust program doesnot terminate in the middle. Also, when there is an error in the

program, it will display an appropriate message to the user and continue execution. Designing such programs is needed in any software development. For this purpose, the programmer should handle the errors. When the errors can be handled, they are called exceptions
the complete exception handling syntax will be in the following format:

```
try:
    statements
except Exception1:
    handler1
except Exception2:
    handler2
else:
    statements:
finally:
    statements
```

try block: The try block contains one or more statements which are likely to encounter an exception. If the statements in this block are executed without an exception, the subsequent except: block is skipped.

If the exception does occur, the program flow is transferred to the except: block. The statements in the except: block are meant to handle the cause of the exception appropriately. For example, returning an appropriate error message.

Except block: The except block is useful to catch an expression that is raised in the try block. When there is an exception in the try block, then only the except block is executed. It is written in various formats

To catch multiple exceptions , we can write multiple catch blocks. The other ways is to use a single except block and write all the exceptions as a tuple inside parentheses as:

```
except (Exceptionalclass1, Exceptioncla2, ……..):
```

To catch any type of exception where we are not bothered about which type of exception it is, we can write except block without mentioning any

Exceptionclass name as:

```
except:
```

else block : While the except block is executed if the exception occurs inside the try block, the else block gets processed if the try block is found to be exception free.

Finally block: The finally block consists of statements which should be processed regardless of an exception occurring in the try block or not. As a consequence, the error-free try block skips the except clause and enters the finally block before going on to execute the rest of the code. If, however, there's an exception in the try block, the appropriate except block will be processed, and the statements in the finally block will be processed before proceeding to the rest of the code.

Example:

```
try:
   date = eval(input("Enter date: "))
except SyntaxError:
   print('Invalid date entered')
```

```
else:
   print('you enterd : ',date)
finally:
  print(' program is completed')
```

Output:

```
Enter date: 2016, 10, 3
program is completed
```

b) Write a python program to handle the ZeroDivisonError exception.

Ans:

```
try:
   x=int(input('Enter a number: '))
   y=int(input('Enter another number: '))
   z=x/y
except ZeroDivisionError:
   print("Division by 0 not accepted")
else:
   print("Division = ", z)
finally:
   x=0
   y=0
print ("Out of try, except, else and finally blocks." )
```

Output 1:

```
Enter a number: 10
Enter another number: 2

Division =  5.0

Out of try, except, else and finally blocks.
```

Output 2:

```
Enter a number: 10
Enter another number: 0

Division by 0 not accepted

Out of try, except, else and finally blocks.
```

3

a) Classify the Files in python with example.

Ans: FILES

To store data in a computer, we need files. For example, we can store employee data like employee number, name and salary in a file in the computer and later we use it whenever we want. In computer's view, a file is nothing but collection of data that is available to a program. Once we store data in a computer file, we can retrieve it and use it depending on our requirements. Figure shows the file in our daily life and the file stored in the computer.

Types of files in python

In python, there are two types of files. They are:

1. Text files
2. Binary files

1.Text files: Text files store the data in the form of characters. For example, if we store employee name "Ganesh". It will stored as 6 characters and the employee salary 8900.75 is stored as 7 characters. Normally, text files are used to store characters or strings**.**

Example:

```
f = open("demofile.txt", "r")
print(f.read())
```

Output:

```
C:\Users\My Name>python demo_file_open.py
Hello! Welcome to demofile.txt
This file is for testing purposes.
Good Luck!
```

2.**Binary files:** Binary files store entire data in the form of bytes i.e. a group of 8bits each. For example, a character is stored as a byte and an integer is stored in the form of 8 bytes (on a 64 bit machine). When the data is retrieved from the binary file, the programmer can retrieve the data as bytes. Binary files can be used to store text, images , audio and video.

Image files are generally available in .jpg, .gif or .png formats. We cannot use text lines to store images as the images do not contain characters. On the other hand, images contain pixels which are minute dots with which the picture is composed of. Each pixel can be represented by a bit, i.e either 1 or 0. Since these bits can be handled by binary files, we can say that they are highly suitable to store images.

Example:

```
file=open("array.bin","rb")
number=list(file.read(3))
```

```
print (number)
file.close()
```

Output:

```
[2, 4, 6]
```

b)Write a Python program to count number of lines, words and characters in a text file.

Ans:

```
 import os , sys
path="sai.txt"
if os.path.isfile(path):
   f=open(path,"r")
else:
   print("does not exists")
   sys.exit()
cl=cw=cc=0
for line in f:
   words=line.split()
   cl=cl+1
   cw=cw+len(words)
   cc=cc+len(line)
print("no.of lines: ",cl)
print("no.of words",cw)
print("no.of characters",cc)
```

output:

```
no.of lines:  4
no.of words 20
no.of characters 120
```

4 **a) Write a short note on:**

i)Advantages of storing data in a files

ii) Importance of Closing file

iii) seek () function

iv) tell () function

Ans:

i)Advantages of storing data in a files

There are four important advantages of storing data in a file:

1. When the data is stored in a file, it is stored permanently. This means that even though the computer is switched off, the data is not removed from the memory since the file is stored on hard disk or CD. This file data can be utilized later, whenever required.
2. It is possible to update the file data. For example, we can add new data to the existing file, delete unnecessary datafrom the file and modify the available data of the file. This makes the file more useful.
3. Once the data is stored in a file, the same data can be shared by various programs. For example, once employee data is stored in a file, it can be

used in a program to calculate employees’ net salaries or in another program to calculate income tax payable by the employees.

4. Files are highly useful to store huge amount of data. For example, voter’s list or census data.

ii) Importance of Closing file

1. It puts your program in the garbage collectors hands - though the file *in theory* will be auto closed, it may not be closed. Python 3 and Cpython generally do a pretty good job at garbage collecting, but not always, and other variants generally suck at it.
2. It can slow down your program. Too many things open, and thus more used space in the RAM, will impact performance.
3. For the most part, many changes to files in python do not go into effect until *after* the file is closed, so if your script edits, leaves open, and reads a file, it won't see the edits.
4. You could, theoretically, run in to limits of how many files you can have open.

iii) seek () function

In Python, seek() function is used to change the position of the File Handle to a given specific position. File handle is like a cursor, which defines from where the data has to be read or written in the file.

Syntax: f.seek()

GfG.txt : "Code is like humor. When you have to explain it, it’s bad."

Code:

```
f = open("GfG.txt", "r")
f.seek(20)
print(f.tell())
print(f.readline())
f.close()
```

Output:

```
20
When you have to explain it, it’s bad.
```

iv) tell () function

The `tell()` method returns the current file position in a file stream.

Syntax

```
file.tell()
```

example:

```
f = open("demofile.txt", "r")
print(f.readline())
print(f.tell())
```

Output:

```
Hello! Welcome to demofile.txt
32
```

5 **a) Explain the File modes for opening a file in Python.**

Ans:

Opening a File

Weshould use open() function to open a file. This function accepts 'filename' and 'openmode' in which to open the file.

file handler = open("file name", "open mode", "buffering")

Here, the 'file name' represents a name on which the data is stored. We can use any name to reflect the actual data. For example, we can use 'empdata' as file name to represent the employee data. The file 'open mode' represents the purpose of opening the file. Table specifies the file open modes and their meanings.

The file opening Modes

File open mode	Description
W	To write data into file. If any data is already present in the file, it would be deleted and the present data will be stored.
R	To read data from the file. The file pointer is positioned at the beginning of the file.
A	To append data to the file. Appending means adding at the end of existing data. The file pointer at the end of the file. If the file does not exist, it will create a new file for writing data.
w+	To write and read data of a file. The previous data in the file will be deleted.
r+	To read and write data into a file. the previous data in the file will not be deleted. The file pointer is placed at the beginning of the file.
a+	To append and read data of a file. the file pointer will be at the end of the file exists. If the file does not exist, it creates a new file for reading and writing.
X	To open the file in exclusive creation mode. The file creation fails if file already exists.

A buffer represents a temporary block of memory. 'buffering' is an optional integer used to set the size of the buffer for the file. In the binary mode, we can pass 0 as buffering integer to inform not to use any buffering. In text mode, we can use 1 for buffering to retrieve data from the file one line at a time.

Example:

```
f = open('myfile.txt','w')
str  = input('enter text: ')
```

```
f.write(str)
f.close()
```

Output:

```
enter text: This is my first program
```

b) Write a Python program to know whether a file exists or not, if it is existed display the content of a file.

Ans:

Code:

```
import os
path="sai.txt"
isfile=os.path.isfile(path)
print(isfile)
if os.path.isfile(path):
   f=open(path,"r")
   read=f.read()
   print(read)
else:
   print("file does not exist")
```

output :

```
True
Code is like humor. When you have to explain it, it’s bad.
```

Added questions for semester

Unit-1

5 **a) Explain the following built-in data types**

I) None II) Numeric III) Bool

I) None :

- None keyword is an object and is a data type of none type class.
- None datatype doesn’t contain any value.
- None keyword is used to define a null variable or object.
- None keyword is immutable.

Example:

```
x = None
if x:
  print("Do you think None is True?")
elif x is False:
  print ("Do you think None is False?")
else:
  print("None is not True, or False, None is just None...")
```
output:

```
None is not True, or False, None is just None...
```

II) Numeric: Python numeric data type is used to hold numeric values like;

int – holds signed integers of non-limited length.
long- holds long integers(exists in Python 2.x, deprecated in Python 3.x).
float- holds floating precision numbers and it's accurate up to 15 decimal places.
complex- holds complex numbers.

Example:
```
a = 5
print("Type of a: ", type(a))
b = 5.0
print("\nType of b: ", type(b))
c = 2 + 4j
print("\nType of c: ", type(c))
```

output:
```
Type of a:  <class 'int'>
Type of b:  <class 'float'>
Type of c:  <class 'complex'>
```
III) Bool: Booleans in Python. In Python, the boolean is a data type that has only two values and these are.

- Declaring a Boolean Value in Python.
- Exceptions in Declaring Booleans in Python.
- Python bool () function.
- More about Python Booleans.

Example:

```
print(10 > 9)
print(10 == 9)
print(10 < 9)
```

output:
```
True
False
False
```

b) Demonstrate the use of Indentation , Docstring and comments in python program with a sample program

Indentation :Python indentation is a way of telling a Python interpreter that the group of statements belongs to a particular block of code. A block is a combination of all these statements. Indentation is a very important concept of Python because without proper indenting the Python code, you will end up seeing *IndentationError* and the code will not get compiled.

Code:
```
site = 'gfg'
if site == 'gfg':
   print('Logging on to geeksforgeeks...')
```

```
else:
   print('retype the URL.')
print('All set !')
```

output:

```
Logging on to geeksforgeeks...
All set !
```

Docstring : Python documentation strings (or docstrings) provide a convenient way of **associating documentation with Python modules, functions, classes, and methods**. It's specified in source code that is used, like a comment, to document a specific segment of code.

Code:

```
def my_function():
   '''Demonstrates triple double quotes
   docstrings and does nothing really.'''
    return None
print("Using __doc__:")
print(my_function.__doc__)
```

output:

```
  Using __doc__:
Demonstrates triple double quotes
    docstrings and does nothing really.
```

Comments: Comments are non-executable statements in Python. It means neither the python compiler nor the PVM will execute them. Comments are intended for human understanding, not for the compiler or PVM. Therefore they are called non-executable statements.

A single-line comment begins with a hash (#) symbol and is useful in mentioning that the whole line should be considered as a comment until the end of the line.
Multi-line comment is useful when we need to comment on many lines. You can also use a single-line comment, but using a multi-line instead of single-line comment is easy to comment on multiple lines

Example:

```
"""
Author: Aditya.org
Description:
Writes the words Hello World on the screen
"""
```

3) Summarize at least three functions, 5 methods and constructs used in Tuple data type ?

Methods in tuple:

Count() : count() method returns the number of times a specified value appears in the tuple.

```
thistuple = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)

x = thistuple.count(5)

print(x)
```

output:

```
2
```

Index():The index() method finds the first occurrence of the specified value.

The index() method raises an exception if the value is not found.

Example:

```
thistuple = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)
x = thistuple.index(8)
print(x)
```

output:

```
3
```

Fucntions in tuple:

1. len():Like a list, a Python tuple is of a certain length. The len() function returns its length.

Ex:

```
>>>(1, 2, 3, [6, 5])
>>> len(my_tuple)
```

Output

```
4
```

2. max():It returns the item from the tuple with the highest value.

```
>>> a=(3,1,2,5,4,6)
>>> max(a)
```

Output:

```
6
```

3. min():Like the max() function, the min() returns the item with the lowest values.

```
>>> a=(3,1,2,5,4,6)
>>> min(a)
```

Output

```
1
```

Consturcts in tuple: i) type(sequence):In Python, type(seq) returns the type of sequence. Sequence can be lists, dictionaries, tuple, string, etc.

```
t = ("Codespeedy",2,800,7.9)
print(type(t))
```

output:

```
<class 'tuple'>
```

ii) tuple(list):In Python, tuple(list) takes list as sequence. List is collection of ordered and mutable items. It will convert list into tuple.

Example:

```
l = [1,2,3,"Codespeedy"]
tuple(l)
```

Output is:
(1, 2, 3, 'Codespeedy')

iii) tuple(string):In Python, tuple(string) takes string as sequence. It will convert string into tuple by breaking each character of string as element of tuple.
Example:

```
s = "Rani"
tuple(s)
```

OUTPUT:

```
('R', 'a', 'n', 'i')
```

iv) tuple(dictionary): In Python, tuple(dict) takes dictionary as sequence. A dictionary is a collection of unordered, changeable and indexed items. It will convert dictionary into tuple. Tuple will contain Keys of dictionaries.
Example:

```
d = {1:"One",2:"Two",3:"Three",4:"Four",5:"Five"}
tuple(d)
```

OUTPUT:

```
(1, 2, 3, 4, 5)
```

v) tuple():In Python, tuple() takes no parameter. Hence, it will create an empty tuple.
Example:

```
t = tuple()
print("Empty tuple:",t)
```

OUTPUT:

```
Empty tuple: ()
```

Unit -2

1.iii) Multivariable declaration : **Declare multiple variables and assign one value**. There's one more way of declaring multiple variables in Python and assign values.
Ex:

```
x, y, z = "Orange", "Banana", "Cherry"

print(x)
print(y)
print(z)
```

output:

```
Orange
Banana
Cherry
```

3) a) What is a command line argument? Explain briefly how to implement it in python programming with an example

Command line argumrnts:
Python allows for command line input.
That means we are able to ask the user for input.
The method is a bit different in Python 3.6 than Python 2.7.

Python 3.6 uses the input() method.
The following example asks for the user's name, and when you entered the name, the name gets printed to the screen:

Example:
```
print("Enter your name:")
x = input()
print("Hello ", x)
```

output:
Enter your name:
Linus
Hello Linus

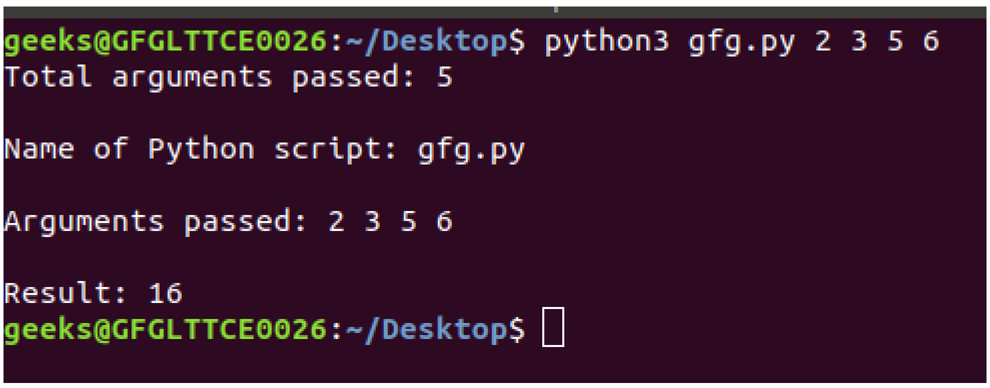
The sys module provides functions and variables used to manipulate different parts of the Python runtime environment. This module provides access to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter.
One such variable is sys.argv which is a simple list structure. It's main purpose are:

It is a list of command line arguments.
len(sys.argv) provides the number of command line arguments.
sys.argv[0] is the name of the current Python script.

Example: Let's suppose there is a Python script for adding two numbers and the numbers are passed as command-line arguments.

```
import sys
n = len(sys.argv)
print("Total arguments passed:", n)
print("\nName of Python script:", sys.argv[0])
print("\nArguments passed:", end = " ")
for i in range(1, n):
   print(sys.argv[i], end = " ")
Sum = 0
for i in range(1, n):
   Sum += int(sys.argv[i])
print("\n\nResult:", Sum)
```

Output:



```
geeks@GFGLTTCE0026:~/Desktop$ python3 gfg.py 2 3 5 6
Total arguments passed: 5

Name of Python script: gfg.py

Arguments passed: 2 3 5 6

Result: 16
geeks@GFGLTTCE0026:~/Desktop$
```

b) Illustrate the for and while loop with an example program

for loop:

- The for loop is useful to iterate over the elements of a sequence.
- It means, the for loop can be used to execute a group of statements repeatedly depending upon the number of elements in the sequence.
- The for loop can work with sequence like list, tuple, range etc.

the syntax of the for loop is given below:

```
for var in sequence:
        Statements
```

Example:

```
for x in range(1, 6):
  print (x, "squared is", x * x)
```

Output:

```
1 squared is 1
2 squared is 4
3 squared is 9
4 squared is 16
5 squared is 25
```

While loop: Executes a group of statements as long as a condition is True.

good for *indefinite loops* (repeat an unknown number of times)

Syntax:

```
while condition:
        statements
```

Example:

```
number = 1
while number < 200:
  print (number)
  number = number * 2
```

Output:

```
1 2 4 8 16 32 64 128
```

4) a) Write a Python program to find the minimum number of notes (Sample of notes: 10, 20, 50, 100, 200 , 500 and 2000) against an given amount.

Code:

```
x=int(input("enter the amount"))
A=[2000,500,200,100,50,20,10]
n=0
for i in range(7):
    a=A[i]
    b=int(x/a)
    x=int(x%a)
    n=n+b
    print("no of notes of denomination ",A[i],b)
print("total no of notes",n)
print("amount in the form of change",x)
```

output:

```
enter the amount 9999

no of notes of denomination  2000   4

no of notes of denomination  500    3

no of notes of denomination  200    2

no of notes of denomination  100    0

no of notes of denomination  50     1

no of notes of denomination  20     2

no of notes of denomination  10     0

total no of notes 12

amount in the form of change 9
```

b) Explain elseif ladder in python programming

Sometimes , the programmer has to test multiple conditions and execute statements depending on those conditions.
If….elif….else statement is useful in such situations.
Consider the following syntax of if…elif….else statement:

```
if condition1:
        statement1
elif condition2:
        statement2
elif condition3:
        statement3
else:
        statement4
```

The statements1, statements2,… represent one statement or a suite. The final 'else' part is not compulsory. It means, it is possible to write if….elif statement, without 'else' and statements4.

```
Example:
num = -5
if num==0:
        print(num,"is zero")
elif num>0:
        print(num,"is positive")
else:
        print(num,"is negative“)
```

output:

```
num is negative
```

5) a) Write a Python program to count the number of even and odd numbers from a sequence of numbers.

Code:

```
x=int(input("enter the range of numbers:"))
i=1
even=0
odd=0
while i<x:
    if(i%2==0):
        even=even+1
    else:
        odd=odd+1
    i+=1
print("count of even numbers :",even)
print("count of odd numbers:", odd)
```

output:

```
enter the range of numbers:10

count of even numbers : 4

count of odd numbers: 5
```

unit- 3

3)

c) Write a program to perform the following operations on 2 dimension matrix of 3x3 size
i) Addition ii) Subtraction iii) Multiplication iv) Transpose

i)addition:

```
import numpy as np

x=np.array([[1,2,3],[4,5,6],[7,8,9]])

y=np.array([[1,1,1],[1,1,1],[1,1,1]])

print(x+y)
```

output:

```
[[ 2  3  4]
 [ 5  6  7]
 [ 8  9 10]]
```

ii) Subtraction :

import numpy as np

x=np.array([[1,2,3],[4,5,6],[7,8,9]])

y=np.array([[1,1,1],[1,1,1],[1,1,1]])

print(x-y)

output:

```
[[0 1 2]
 [3 4 5]
 [6 7 8]]
```

iii) Multiplication :

import numpy as np

x=np.array([[1,2,3],[4,5,6],[7,8,9]])

y=np.array([[1,1,1],[1,1,1],[1,1,1]])

z=np.matmul(x,y)

print(z)

output:

```
[[ 6  6  6]
 [15 15 15]
 [24 24 24]]
```

iv) Transpose:

import numpy as np
x=np.array([[1,2,3],[4,5,6],[7,8,9]])
z=x.transpose()
print(z)

output:

```
[[1 4 7]
 [2 5 8]
 [3 6 9]]
```

5 **a) Demonstrate scope of a variables with a simple program**

In Python, variables are the containers for storing data values. They are reference, or pointers, to an object in memory which means that whenever a variable is assigned to an instance, it gets mapped to that instance. Unlike other languages like C/C++/JAVA, Python is not “statically typed”. We do not need to declare variables before using them or declare their type. A variable is created the moment we first assign a value to it.

Example:

```
def f():
  s = "Me too."
  print(s)

# Global scope
```

```
s = "I love Python"
f()
print(s)
```

output:
```
Me too.
I love Python
```

b) What is a recursion function? Write a program to find multiplication of a number using recursion.

Recursion

Python also accepts function recursion, which means a defined function can call itself.

Recursion is a common mathematical and programming concept. It means that a function calls itself. This has the benefit of meaning that you can loop through data to reach a result.

The developer should be very careful with recursion as it can be quite easy to slip into writing a function which never terminates, or one that uses excess amounts of memory or processor power. However, when written correctly recursion can be a very efficient and mathematically-elegant approach to programming.

Advantages of using recursion

- A complicated function can be split down into smaller sub-problems utilizing recursion.
- Sequence creation is simpler through recursion than utilizing any nested iteration.
- Recursive functions render the code look simple and effective.

Disadvantages of using recursion

- A lot of memory and time is taken through recursive calls which makes it expensive for use.
- Recursive functions are challenging to debug.
- The reasoning behind recursion can sometimes be tough to think through.

a program to find multiplication of a number using recursion.

Code:
```python
def mult(a, b):
    if a == 0:
        return 0
    elif a == 1:
        return b
    else:
        return b + mult(a-1, b)
a=10
b=10
print(mult(a,b))
```

output:
```
100
```

6 **a) Define Lambda function. Explain the working mechanism of lambda function with simple example**

Lambda function:

A lambda function is a small anonymous function.

A lambda function can take any number of arguments, but can only have one expression.

Syntax

lambda arguments : expression

example:

```
x = lambda a: a + 10
print(x(5))
```

Output:

```
15
```

Use of Lambda functions: The power of lambda is better shown when you use them as an anonymous function inside another function. Say you have a function definition that takes one argument, and that argument will be multiplied with an unknown number.

EXAMPLE:

```
def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)

print(mydoubler(11))
```

Output:

```
22
```

Use with filter()

- The filter() function in Python takes in a function and a list as arguments.
- The function is called with all the items in the list and a new list is returned which contains items for which the function evaluates to True.
- Here is an example use of filter() function to filter out only even numbers from a list.

Example:

```
my_list = [1, 5, 4, 6, 8, 11, 3, 12]
new_list = list(filter(lambda x: (x%2 == 0) , my_list))
print(new_list)
```

Output:

```
[4, 6, 8, 12]
```

Use with map()

- The map() function in Python takes in a function and a list.
- The function is called with all the items in the list and a new list is returned which contains items returned by that function for each item.
- Here is an example use of map() function to double all the items in a list.

Example:

```
 my_list = [1, 5, 4, 6, 8, 11, 3, 12]
new_list = list(map(lambda x: x * 2 , my_list))
print(new_list)
```

Output:

```
[2, 10, 8, 12, 16, 22, 6, 24]
```

b) Write a Python function that accepts a string and calculate the number of upper case letters and lower case letters

```
string=input("Enter string:")
count1=0
count2=0
for i in string:
    if(i.islower()):
        count1=count1+1
    elif(i.isupper()):
        count2=count2+1
print("The number of lowercase characters is:")
print(count1)
print("The number of uppercase characters is:")
print(count2)
```

output:

```
Enter string: Sai Teja
The number of lowercase characters is:
5
The number of uppercase characters is:
2
```