Model Question Paper-I with effect from 2022-23 (CBCS Scheme)

USN | | | | | | | | | | |

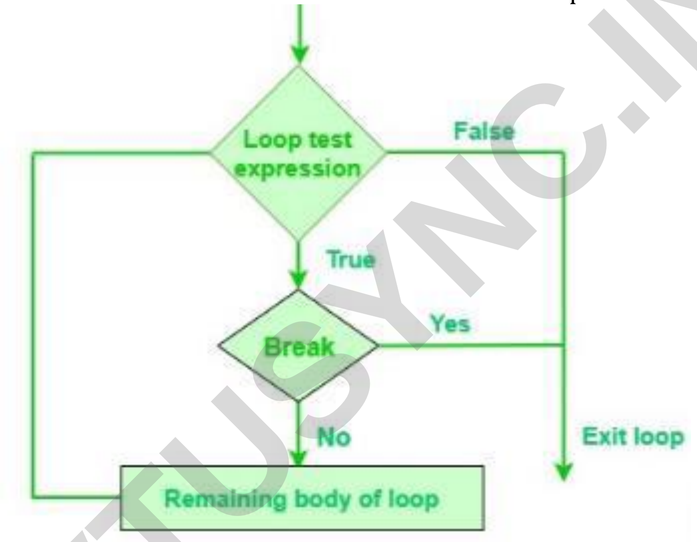
First/Second Semester B.E. Degree Examination

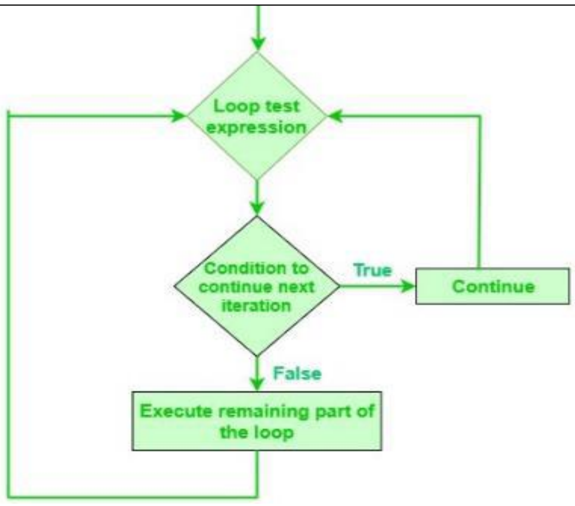
Introductionto Python Programming

TIME: 03 Hours **Max. Marks: 100**

Note: 01. Answer any **FIVE** full questions, choosing at least **ONE** question from each **MODULE**.

		Module -1	***Bloom's Taxonomy Level**	**Marks**

Q.01	**a**	**With Python programming examples to each, explain the syntax and controlflow diagrams of break and continue statements.**
		
		Python break is used to terminate the execution of the loop.
		 Loop test expression
		False
		True
		Break
		Yes
		No
		Exit loop
		Remaining body of loop
		i = 1
		 while i < 6:
		    print(i)
		if i == 3:
		    break
		 i += 1
		OUTPUT
		1
		2
		3
		Python Continue Statement skips the execution of the program block from after the continue statement and forces the control to start the next iteration.
			L2	**08**

		 Loop test expression Condition to continue next iteration True Continue False Execute remaining part of the loop i = 0 while i <= 20 : i += 1 #i=i+1 if i % 2 == 1 : continue print(i) **OUTPUT:** 2 4 6 8 10 12 14 16 18 20		
	b	**Explain TWO ways of importing modules into application in Python with syntax and suitable programming examples.** To use functions, classes, or variables defined in a module, it needs to be imported in the program. There are several ways to import modules in Python programs, including: **1.** **import module_name**: This statement imports the entire module and allows you to use its functions, classes, and variables with the **module_name.** prefix. For example: import math print(math.pi) **2.** **from module_name import function_name**: This statement imports a specific function from a module and allows you to use it without the **module_name.** prefix. For example: from math import pi print(pi) **3.** **import module_name as alias_name**: This statement imports a module and gives it an alias name to use in the program. For example: import numpy as np	**L2**	**06**

```
arr = np.array([1, 2, 3])
```

4.
from module_name import *****: This statement imports all functions, classes, and variables from a module into the program's namespace. However, this method is generally discouraged because it can lead to naming conflicts and make the code harder to read and understand.

	c	**Write a function to calculate factorial of a number. Develop a program to compute binomialcoefficient (Given N and R).**	**L3**	**06**

```
num = int(input("Enter a number: "))
factorial = 1
if num < 0:
       print(" Factorial does not exist for negative numbers")
elif num == 0:
    print("The factorial of 0 is 1")
else:
     for i in range(1,num + 1):
         factorial = factorial*i
print("The factorial of",num,"is",factorial)
```

OUTPUT:

```
Enter a number: 4
 The factorial of 4 is 24
```

OR

Q.02	**a**	**Explain looping control statements in Python with a syntax and example to each.**	**L2**	**06**

Looping means repeating something over and over until a particular condition is satisfied.

Python has two primitive loop commands:
while loops
for loops

The while Loop
With the while loop we can execute a set of statements as long as a condition is true.

```
i = 1
 while i < 6:
  print(i)
  i += 1
```

OUTPUT:

```
1
2
3
4
5
```

for Loops and the range() Function The while loop keeps looping while its condition is True, but what if you want to execute a block of code only a certain number of times. You can do this with a for loop statement and the range() function. For loop includes the following:
The **for** keyword
A variable name
The **in** keyword
A call to the **range()** method with up to three integers passed to it
A colon
Starting on the next line, an indented block of code
Example:

```
print('My name is')
for i in range(3):
        print('Jimmy Three Times ')


OUTPUT:
Jimmy Three Times
Jimmy Three Times
Jimmy Three Times
```

	b	**Develop a Python program to generate Fibonacci sequence of length (N). Read N from the console.**	L3	04

```
nterms = int(input ("How many terms the user wants to print? "))
# First two terms
n1 = 0
n2 = 1
count = 0
# Now, we will check if the number of terms is valid or not
if nterms <= 0:
   print ("Please enter a positive integer, the given number is not valid")
# if there is only one term, it will return n_1
elif nterms == 1:
    print ("The Fibonacci sequence of the numbers up to", nterms, ": ")
   print(n1)
# Then we will generate Fibonacci sequence of number
else:
  print ("The fibonacci sequence of the numbers is:")
while count < nterms:
   print(n1)
   nth = n1 + n2
# At last, we will update values
   n1 = n2
  n2 = nth
  count += 1
```

OUTPUT

```
How many terms the user wants to print? 8
The fibonacci sequence of the numbers is:
0
1
1
2
3
5
8
13
```

	c	**Write a function named DivExp which takes TWO parameters a, b and returns a value c (c=a/b). Write suitable assertion for a>0 in function DivExp and raise an exception for when b=0.Develop a Python program which reads two values from the console and calls a function DivExp.**	L3	06

```
def DivExp(a,b):
    try:
        c=a/b
        return(c)
    except:
        print(“divide by zero error”)

n1=int(input(“enter the value of first number”))
n2=int(input(“enter the value of Second number”))
result=DivExp(n1,n2)
print(“The division of”,n1,”and”,n2,”is=”,result)
```

OUTPUT 1:

```
enter the value of first number
5
enter the value of second number
0
 divide by zero error
```

d **Explain FOUR scope rules of variables in Python.** **L2** **04**

In Python, the scope rules of variables determine where a variable can be accessed or modified within a program. Here are the four scope rules of variables in Python:

Global scope: Variables defined outside any function or class have a global scope. These variables can be accessed or modified from anywhere in the program.

Example of global variable:

```
x = 300 Global Variable
 def myfunc():
      print(x)
      myfunc()
      print(x)
OUTPUT
300
300
```

Local scope: Variables defined inside a function have a local scope. These variables can only be accessed or modified within the function in which they are defined. Local variables take precedence over global variables with the same name.

Example of local variable:

```
def myfunc():
    x = 300 Local variable
    print(x)
myfunc()
```

OUTPUT

```
300
```

Enclosing scope: Variables defined in an enclosing function have an enclosing scope. These variables can be accessed or modified by nested functions, but not by functions outside the enclosing function.

Built-in scope: Python has a set of built-in functions and exceptions that are always available for use. Variables defined in the built-in scope can be accessed or modified from anywhere in the program.

Module-2

Q. 03 **a** **Explain with a programming example to each:** **L2** **06**

(ii) get()
(iii) setdefault()

(i) **get()**: The **get()** method in Python is used to access the value of a key in a dictionary. It takes one mandatory argument, which is the key to be searched in the dictionary.

```
# create a dictionary
my_dict = {'apple': 10, 'banana': 20, 'orange': 30}

# access the value of a key using get()
apple_count = my_dict.get('apple')
```

```
# print the values
print(apple_count)

 # Output: 10
```

(ii) **setdefault()**: The **setdefault()** method in Python is used to insert a key-value pair into a dictionary if the key does not exist. It takes two arguments - the first argument is the key to be inserted, and the second argument (optional) is the value to be assigned to the key.

```
# create a dictionary
my_dict = {'apple': 10, 'banana': 20, 'orange': 30}

# add a new key-value pair using setdefault()
mango_count = my_dict.setdefault('mango', 40)

# update the value of an existing key using setdefault()
apple_count = my_dict.setdefault('apple', 50)

# print the dictionary
print(my_dict)
```

```
 # Output: {'apple': 10, 'banana': 20, 'orange': 30, 'mango': 40}
```

b	**Develop suitable Python programs with nested lists to explain copy.copy( ) and copy.deepcopy( ) methods.**	**L3**	**08**

copy.copy() and **copy.deepcopy()** are methods from the Python **copy** module used for creating copies of objects. Both of these methods can be used with nested lists.

Using copy.copy()

copy.copy() creates a shallow copy of an object. This means that a new object is created which is a copy of the original object, but the references to the objects inside the original object are not copied. Instead, the new object contains references to the same objects as the original object.

```
import copy

original_list=[[1,2,3],[4,5,6],[7,8,9]]
# create a shallow copy of original_list
shallow_copy = copy.copy(original_list)

# modify the first element of the first sublist of shallow_copy
shallow_copy[0][0] = 100

# print both original_list and shallow_copy
print(original_list)
print(shallow_copy)
```

```
# Output:
 [[100, 2, 3], [4, 5, 6], [7, 8, 9]]
 [[100, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Using copy.deepcopy()

copy.deepcopy() creates a deep copy of an object. This means that a new object is created which is a copy of the original object, and the references to the objects inside the original object are also copied. This creates a completely new set of objects, independent of the original object.

```
import copy

original_list=[[1,2,3],[4,5,6],[7,8,9]]
# create a deep copy of original_list
deep_copy = copy.deepcopy(original_list)

# modify the first element of the first sublist of deep_copy
deep_copy[0][0] = 200

# print both original_list and deep_copy
print(original_list)
print(deep_copy)
```

```
# Output:
[[100, 2, 3], [4, 5, 6], [7, 8, 9]]
 [[200, 2, 3], [4, 5, 6], [7, 8, 9]]
```

c **Explain append() and index() functions with respect to lists in Python.** **L2** **06**

append() and **index()** are two built-in functions in Python that are used with lists.
append() function:
The **append()** function is used to add an element to the end of a list. The example for **append()** function is as follows:

```
ls=[1,2,3]
ls.append("hi")
ls.append(10)
print(ls)
```

OUTPUT: [1, 2, 3, “hi”, 10]

index() function:

The index() function is used to find the index of a specified element in a list. The example for the index() function is as follows

```
fruits = ["apple", "banana", "orange", "grape"]
index = fruits.index("banana")
print(index)
```

OUTPUT:
1
As we can see, the index variable now contains the index of the "banana" element in the fruits list, which is 1.

OR

Q.04 **a** **Explain different ways to delete an element from a list with suitable Python syntax and programming examples.** **L2** **10**

- **pop():** This method deletes the last element in the list, by default.

```
ls=[3,6,-2,8,10]
x=ls.pop()
print(ls)
```

OUTPUT: [3, 6, -2, 8]

- The del statement will delete values at an index in a list. All of the values in the list after the deleted value will be moved up one index.

```
ls=[3,6,-2,8,10]
del ls[2]
print(ls)
```
OUTPUT : [3, 6, 8, 10]

- **clear():** This method removes all the elements in the list and makes the list empty.

```
ls=[1,2,3]
ls.clear()
print(ls)
```

- You can use the remove() method to remove an element from a list by specifying its value. Here's an example:
  my_list = [1, 2, 3, 4, 5]
  my_list.remove(3)
  print(my_list) # Output: [1, 2, 4, 5]

	b	**Read a multi-digit number (as chars) from the console. Develop a program to print the frequency of each digit with suitable message.**	**L3**	**06**

```
def countoccurrences(n, d):
        count = 0
        # Loop to find the digits of N
       while (n > 0):
       # check if the digit is D
            if(n % 10 == d):
                count = count + 1
                n = n//10
              # return the count of the
             # occurrences of D in N
      return count
# Driver code
d = 2
n =int(input('Enter multi digit number:'))
print(countoccurrences(n, d))
```
OUTPUT
```
Enter multi digit number:223232
  4
```

	c	**Tuples are immutable. Explain with Python programming example.**	**L2**	**04**

The *tuple* data type is almost **identical** to the list data type, except in two ways. First, tuples are typed with parentheses, (and), instead of square brackets, [and].

Tuples are immutable:
Tuples cannot have their values modified, appended, or removed.
Example:

```
eggs = ('hello', 42, 0.5)
eggs[1] = 99
print(eggs).
```
OUTPUT:
```
  File "main.py", line 2, in <module>
```

		eggs[1] = 99 TypeError: 'tuple' object does not support item assignment		
		Module-3		
Q. 05	**a**	**Explain Python string handling methods with examples: split(),endswith(), ljust(), center(), lstrip()**	**L2**	**10**

1. **split()**
   The **split()** method is used to split a string into a list of substrings based on a specified delimiter. For example:

   ```
   text = "Hello world!"
   words = text.split(" ")
   print(words)

   # Output: ['Hello', 'world!']
   ```

2. **endswith()**
   The **endswith()** method is used to check if a string ends with a specified suffix. It returns **True** if the string ends with the specified suffix, and **False** otherwise. For example:

   ```
   text = "Hello world!"
   result = text.endswith("world!")
   print(result)

    # Output: True
   ```

3. **ljust()**
   The **ljust()** method is used to left-justify a string within a specified width by adding padding characters (usually spaces) to the right side of the string. For example:

   ```
   text = "Hello"
   padded_text = text.ljust(10)
   print(padded_text)

   # Output: 'Hello      '
   ```

4. **center()**
   The **center()** method is used to center a string within a specified width by adding padding characters (usually spaces) to both sides of the string. For example:

   ```
   text = "Hello"
   centered_text = text.center(10)
   print(centered_text)

   # Output:   '  Hello   '
   ```

5. **lstrip()**
   The **lstrip()** method is used to remove leading (leftmost) whitespace characters (or other specified characters) from a string. For example:

   ```
   text = "    Hello"
   stripped_text = text.lstrip()
   print(stripped_text)
   ```

```
# Output: 'Hello'
```

b **Explain reading and saving python program variables using shelve module with suitable Python program.** **L2** **06**

- The Shelve Module of Python is a very popular module of Python which works like an effective tool for persistent data storage inside files using a Python program.
- As the name of this module suggests, i.e., Shelve, we can easily interpret that it will work as a shelf object to keep all our data inside a file and save all the necessary information.
- In the Shelve Module, a shelf object is defined, which acts like a dictionary-type object, and it is persistently stored in the disk file of our computer.

Enter the following into the interactive shell:

```
import shelve
>>> shelfFile = shelve.open('mydata')
>>> cats = ['Zophie', 'Pooka', 'Simon']
>>> shelfFile['cats'] = cats
>>> shelfFile.close()
```

programs can use the shelve module to later reopen and retrieve the data from these shelf files. Shelf values don't have to be opened in read or write mode they can do both once opened. Enter the following into the interactive shell:

```
>>> shelfFile = shelve.open('mydata')
>>> type(shelfFile)
<class 'shelve.DbfilenameShelf'>
>>> shelfFile['cats']
['Zophie', 'Pooka', 'Simon']
>>> shelfFile.close()
```

c **Develop a Python program to read and print the contents of a text file.** **L3** **04**

An example Python program that reads and prints the contents of a text file:

```
filename = "example.txt"
with open(filename, "r") as file:
    contents = file.read()
    print(contents)
```

OR

Q. 06 **a** **Explain Python string handling methods with examples: join(), startswith(),rjust(),strip(),rstrip()** **L2** **10**

1. join():
The join() method is useful when you have a list of strings that need to be joined together into a single string value.
For example, enter the following into the interactive shell:

```
>>> ', '.join(['cats', 'rats', 'bats'])
'cats, rats, bats'
```

```
>>> ' '.join(['My', 'name', 'is', 'Simon'])
'My name is Simon'
```

2. **startswith()**

The **startswith()** method is used to check if a string starts with a specific substring. It takes a substring as input and returns a boolean value indicating whether the string starts with the given substring or not.

```
>>> 'Hello, world!'.startswith('Hello')
True
>>> 'abc123'.startswith('abcdef')
False
```

3. **rjust()**

The **rjust()** method is used to right-justify a string by adding spaces to the left of the string.

```
>>> 'Hello'.rjust(30)
    '                    Hello'
```

4. **strip()**

The **strip()** method is used to remove any leading or trailing whitespace from a string. It returns a new string with the whitespace removed.

```
>>> string = '      Hello World     '
>>> print(string.strip())
Hello World
```

5. **rstrip()**

   The **rstrip()** method is used to remove any trailing whitespace from a string. It returns a new string with the whitespace removed from the right side.

```
text = "        Hello, World!   "
result = text.rstrip()
print(result)
 # Output: '   Hello, World!'
```

b	**Explain with suitable Python program segments: (i) os.path.basename() (ii) os.path.join().**	L2	05

(i) os.path.basename() is a Python function that returns the base name of the file path given as input. The base name is the last part of the path, after the last occurrence of the path separator ('/' on Unix and '\' on Windows) or drive separator (':' on Windows). Here is an example of how to use os.path.basename():

```
import os

# Example file path
file_path = '/home/user/documents/example.txt'

# Get the base name of the file
file_name = os.path.basename(file_path)

# Print the base name
print(file_name)

 # Output:
example.txt
```

i) os.path.join() is a Python function that joins two or more path components into a single path. This function takes one or more path components as input and returns a single path string that represents the concatenated path components. Here is an example of how to use os.path.join():

		import os **# Example directory and file names** **dir_name = 'documents'** **file_name = 'example.txt'** **# Join the directory and file names** **file_path = os.path.join('/home/user', dir_name, file_name)** **# Print the file path** **print(file_path)** # Output: /home/user/documents/example.txt		

	c	**Develop a Python program find the total size of all the files in the given directory.**
		
		import os
		# assign size
		size = 0
		
		# assign folder path
		Folderpath = "/home/secabiet/test"
		
		# get size
		for ele in os.scandir(Folderpath):
		    size+=os.path.getsize(ele)
		
		print(size)
		
		Output:
		566434
			L3	**05**

		Module-4		
Q. 07	**a**	**Explain permanent delete and safe delete with a suitable Python programming example to each.**	**L2**	**08**

Permanent delete refers to the complete removal of data from a storage device, with no possibility of recovery. Once data has been permanently deleted, it cannot be retrieved or restored.

```
import os

# specify the file to be deleted
file_name = "example.txt"

# remove the file permanently
os.remove(file_name)
```

Safe delete, on the other hand, is a type of deletion that allows data to be removed from a storage device, but with the possibility of recovery if needed. This type of deletion involves a process of overwriting the data with new information

```
import os

# specify the file to be deleted
file_name = "example.txt"

# open the file in write mode and overwrite with new data
with open(file_name, "w") as file:
   file.write("This file has been deleted.")

# remove the file
os.remove(file_name)
```

	b	**Develop a program to backing Up a given Folder (Folder in a current working directory) into a ZIP File by using relevant modules and suitable methods.**	**L3**	**06**

```
import os
from zipfile import ZipFile
from os import path
from shutil import make_archive
   # Check if file exists
if os.path.exists("a.txt"):
   # get the path to the file in the current directory
       src = path.realpath("a.txt")
   # now put things into a ZIP archive
    #Split the path name into a pair head and tail
      root_dir,tail = path.split(src)
shutil.make_archive("a_archive","zip",root_dir)
   # more fine-grained control over ZIP files
with ZipFile("a_archive.zip", "w") as newzip:
      newzip.write("a.txt")
      newzip.write("a.txt.bak")
```

	c	**Explain the role of Assertions in Python with a suitable program.**	**L2**	**06**

In python, **assert is** keyword. This statement takes as input a boolean condition, which when returns true doesn't do anything and continues the normal flow of execution, but if it is computed to be false, then it raises an AssertionError along with the optional message provided.

Syntax : *assert condition, error_message(optional)*

Where:

- ***assert:keyword***
- ***condition:*** *The boolean condition returning true or false.*
- ***error_message :*** *The optional argument to be printed in console in case of AssertionError*

Example: Python assert keyword with error message

```
# initializing number
a =4
b =0

# using assert to check for 0
print("The value of a / b is : ")
assert b !=0, "Zero Division Error"
print(a /b)
```

Output:

```
AssertionError: Zero Division Error
```

OR

a **Explain the functions with examples: (i) shutil.copytree() (ii) shutil.move() (ii) shutil.rmtree().** **L3** **06**

shutil is a Python module that provides a file operations, such as copying, moving, and deleting files and directories.

1. In this below example, **shutil.copytree()** is used to copy the entire directory tree from the source directory to the destination directory.

Example:

```
import shutil
# copy a directory tree
shutil.copytree('/path/to/source/directory',
'/path/to/destination/directory')
```

2. **shutil.move(src, dst, copy_function=copy2)**: This function moves a file or directory from the source path to the destination path.

Example:

```
import shutil

# move a file or directory
shutil.move('/path/to/source/file_or_directory',
'/path/to/destination/file_or_directory')
```

3. **shutil.rmtree()** is used to recursively delete the directory tree located at the specified path

Example:

```
import shutil
# delete a directory tree
shutil.rmtree('/path/to/directory')
```

	b	**Develop a Python program to traverse the current directory by listing sub-folders and files.**	L2	06

```
import os
for foldername, subfolder, files in os.walk('/home/secabiet/test'):
        print('current folder is'+foldername)
        for subf in subfolder:
                print('subfolder of :'+foldername+':'+subf)
        for fname in files:
          print('file inside of :'+foldername+':'+fname)
```

OUTPUT

```
current folder is/home/secabiet/test
subfolder of :/home/secabiet/test:test1
file inside of :/home/secabiet/test:a.py
current folder is/home/secabiet/test/test1
```

	c	**Explain the support for Logging with logging module in Python.**	L2	08

Logging is a technique used in programming to record and report events that occur during the execution of a program. It provides a way to track the behavior of a program and helps in identifying issues and debugging problems. The **logging** module in Python provides a flexible and efficient way to handle logging in Python programs.

Let's understand the following example.

Example -

```
import logging
logging.debug('The debug message is displaying')
logging.info('The info message is displaying')
logging.warning('The warning message is displaying')
logging.error('The error message is displaying')
logging.critical('The critical message is displaying')
```

Output:

```
WARNING:root:The warning message is displaying
ERROR:root:The error message is displaying
CRITICAL:root:The critical message is displaying
```

Module-5

Q. 09	a	**Explain the methods __init__ and __str__ with suitable code example to each.**	L2	06

The __init__() method is used to initialize the object's attributes

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

The __str__() method is used to provide a string representation of the object.

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __str__(self):
        return f"Name: {self.name}, Age: {self.age}"
p = Person("John", 25)
print(p)


 # Output:
  Name: John, Age: 25
```

	b	**Explain the program development concept 'prototype and patch' with suitable example.**	L2	06
	c	**Define a function which takes TWO objects representing complex numbers and returns new complex number with a addition of two complex numbers. Define a suitable class 'Complex' to represent the complex number.Develop a program to read N (N >=2) complex numbers and to compute the addition of N complex numbers.**	L3	08

```
class Complex ():
        def initComplex(self):
             self.realPart = int(input("Enter the Real Part: "))
             self.imgPart = int(input("Enter the Imaginary Part: "))
       def display(self):
            print(self.realPart,"+",self.imgPart,"i", sep="")
      def sum(self, c1, c2):
            self.realPart = c1.realPart + c2.realPart
            self.imgPart = c1.imgPart + c2.imgPart
c1 = Complex()
c2 = Complex()
c3 = Complex()
print("Enter first complex number")
c1.initComplex()
print("First Complex Number: ", end="")
c1.display()
print("Enter second complex number")
c2.initComplex()
print("Second Complex Number: ", end="")
c2.display()
print("Sum of two complex numbers is ", end="")
c3.sum(c1,c2)
c3.display()
```

```
OUTPUT:
Enter first complex number
Enter the Real Part: 4
Enter the Imaginary Part: 6
First Complex Number: 4+6i
```

```
Enter second complex number
Enter the Real Part: 4
Enter the Imaginary Part: 6
Second Complex Number: 4+6i
Sum of two complex numbers is 8+12i
```

OR

Q. 10 a **Explain the following with syntax and suitable code snippet: i) Class definition ii) instantiation iii) passing an instance (or objects) as anargumentiv) instances as return values.** — **L2** — **10**

(i) Class definition: A class is a blueprint or a template for creating objects that encapsulate data and behavior. In Python, a class definition begins with the **class** keyword followed by the name of the class and a colon.

Syntax

```
class ClassName:
 #statements
```

(ii) Instantiation: Instantiation is the process of creating an object from a class. To create an object, you use the class name followed by parentheses.

Example: Observe the following statements:
Emp= Employee()

(iii) Passing an instance (or objects) as an argument: You can pass instances of a class (or objects) as arguments to functions or methods.

(iv) Instances as return values: You can also return instances of a class from functions or methods. This allows you to create and return objects based on some conditions or data.

b **Define pure function and modifier. Explain the role of pure functions and modifiers in application development with suitable python programs.** — **L2** — **10**

A pure function is **a function whose output value follows solely from its input values, without any observable side effects**.

Example:

```
class Time:
   hour=0
   minute=0
   second=0
   def print_time(t):
      print('%.2d:%.2d:%.2d' % (t.hour, t.minute, t.second))
   def add_time(t1, t2):
      sum=Time()
      sum.hour = t1.hour + t2.hour
      sum.minute = t1.minute + t2.minute
      sum.second = t1.second + t2.second
      return sum
t1 = Time()
t1.hour = 9
t1.minute = 45
t1.second = 0
```

```
t2 = Time()
t2.hour = 1
t2.minute = 35
t2.second = 0
t3 = Time()
t3=Time.add_time(t1,t2)
Time.print_time(t3)

OUTPUT:
 10:80:00
```

Functions which take mutable arguments (e.g. lists) and change them during execution are called **modifiers**

Example:

```
class Time:
   hour=0
   minute=0
   second=0
   def print_time(self):
      print('%.2d:%.2d:%.2d' % (self.hour, self.minute,
self.second))
   def increment(self):
      self.second += self.second

      if self.second >= 60:
         self.second -= 60
         self.minute += 1

      if self.minute >= 60:
         self.minute -= 60
         self.hour += 1

time = Time()
time.hour = 9
time.minute = 62
time.second = 12
time.increment()
time.print_time()
```

OUTPUT:

```
 10:02:24
```

*Bloom's Taxonomy Level: Indicate as L1, L2, L3, L4, etc. It is also desirable to indicate the COs and POs to be attained by every bit of questions.