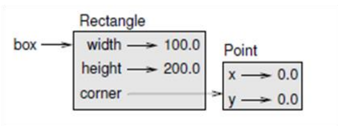
USN

Internal Assessment Test 3 –June 2024:solution

Sub:	**Introduction to Python Programming**					Sub Code:	BPLCK205B	Branch:	Chemistry Cycle
Date:	27-06-2024	Duration:	90 min's	Max Marks:	50	Sem/Sec:	II / Chemistry Cycle		OBE

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1.a	Explain the following file operations in Python with suitable example: i) Copying files and folders ii) Moving files and folders iii) Permanently deleting files and folders.	[6]	CO3	L2
1.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.	[4]	CO3	L3
2.a	Describe logging methods used in python to categorize log messages by importance	[5]	CO4	L2
2.b	Explain five buttons available in Debug control window.	[5]	CO4	L2
3.a	Write a program to implement the following object diagram and its functionality as shown. Initialize an attribute through a constructor and print the same  Rectangle box width 100.0 height 200.0 corner Point x 0.0 y 0.0	[5]	CO4	L3
3.b	Explain _ _init_ _ ( ) and _ _str_ _ ( ) method in detail.	[5]	CO2	L2
4.a	Define Pure function and Modifier function. Illustrate with an example Python program.	[5]	CO4	L3
4.b	Define a function which takes two objects representing complex numbers and returns a new complex number with an addition of two complex numbers. Define a suitable class Complex' to represent the complex number. Develop a program to read N complex numbers and to compute the addition of N complex numbers.	[5]	CO4	L3

.a	What is a class? How to define a class in python? How to initiate a class and how the class members are accessed?	[5]	CO4	L2
5.b	Discuss Operator overloading with an example program	[5]	CO4	L2
6.a	Explain Assertions with an example program of how assertions used in traffic light simulation	[5]	CO3	L2
6.b	List out the benefits of compressing files? Also explain reading of a zip file	[5]	CO3	L2
7.a	Explain the concept of copy.copy() and copy.deepcopy() module in class with an example object diagram.	[5]	CO4	L2
7.b	Briefly explain the printing of objects with examples.	[5]	CO4	L2

CI **CCI** **HOD**

Solution:

1a.Explain the following file operations in Python with suitable example:
i) Copying files and folders
ii) Moving files and folders
iii) Permanently deleting files and folders.
Answer:
Copying files and folders:
The shutil module provides functions for copying files, as well as entire folders.
Calling shutil.copy(source, destination) will copy the file at the path source to the folder at the path destination. (Both source and destination can be strings or Path objects.) If destination is a filename, it will be used as the new name of the copied file. This function returns a string or Path object of the copied file.
Enter the following into the interactive shell to see how shutil.copy() works:

```
 >>> import shutil, os
   >>> from pathlib import Path
   >>> p = Path.home()
❶ >>> shutil.copy(p / 'spam.txt', p / 'some_folder')
   'C:\\Users\\Al\\some_folder\\spam.txt'
❷ >>> shutil.copy(p / 'eggs.txt', p / 'some_folder/eggs2.txt')
   WindowsPath('C:/Users/Al/some_folder/eggs2.txt')
```

The first shutil.copy() call copies the file at *C:\Users\Al\spam.txt* to the folder *C:\Users\Al\some_folder*. The return value is the path of the newly copied file.
The second shutil.copy() call ❷ also copies the file at *C:\Users\Al\eggs.txt* to the folder *C:\Users\Al\some_folder* but gives the copied file the name *eggs2.txt*.
While shutil.copy() will copy a single file, shutil.copytree() will copy an entire folder and every folder and file contained in it. Calling shutil.copytree(source, destination) will copy the folder at the path source, along with all of its files and subfolders, to the folder at the path destination.
The source and destination parameters are both strings. The function returns a string of the path of the copied folder.
Enter the following into the interactive shell:

```
>>> import shutil, os
>>> from pathlib import Path
>>> p = Path.home()
>>> shutil.copytree(p / 'spam', p / 'spam_backup')
WindowsPath('C:/Users/Al/spam_backup')
```

The shutil.copytree() call creates a new folder named *spam_backup* with the same content as the original *spam* folder. You have now safely backed up your precious, precious spam.

Moving and Renaming Files and Folders

Calling shutil.move(source, destination) will move the file or folder at the path source to the path destination and will return a string of the absolute path of the new location.

If destination points to a folder, the source file gets moved into destination and keeps its current filename. For example, enter the following into the interactive shell:

```
>>> import shutil
>>> shutil.move('C:\\bacon.txt', 'C:\\eggs')
'C:\\eggs\\bacon.txt'
```

Permanently Deleting Files and Folders

- You can delete a single file or a single empty folder with functions in the os module,
- whereas to delete a folder and all of its contents, you use the shutil module.

- Calling os.unlink(path) will delete the file at path.
- Calling os.rmdir(path) will delete the folder at path. This folder must be empty of any files or folders.
- Calling shutil.rmtree(path) will remove the folder at path, and all files and folders it contains will also be deleted.

Be careful when using these functions in your programs! It’s often a good idea to first run your program with these calls commented out and with print() calls added to show the files that would be deleted.
Here is a Python program that was intended to delete files that have the *.txt* file extension but has a typo (highlighted in bold) that causes it to delete *.rxt* files instead:

```
import os
from pathlib import Path
for filename in Path.home().glob('*.rxt'):
    os.unlink(filename)
```

If you had any important files ending with *.rxt*, they would have been accidentally, permanently deleted. Instead, you should have first run the program like this:

```
import os
from pathlib import Path
for filename in Path.home().glob('*.rxt'):
    #os.unlink(filename)
    print(filename)
```

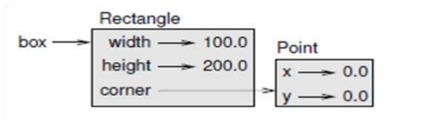
1b. Develop a program to backing Up a given Folder (Folder in a current working Directory) into a ZIP File by using relevant modules and suitable methods.
Answer:

```
import os
import zipfile

zf = zipfile.ZipFile("myzipfile.zip", "w") #zip file name
for dirname, subdirs, files in os.walk("../Python_Programs"): #folder name to create zip
    zf.write(dirname)
    for filename in files:
        zf.write(os.path.join(dirname, filename))
zf.close()
```
2a.Describe logging methods used in python to categorize log messages by importance?
Answer:
ogging.basicConfig(level=logging.DEBUG, format=' %(asctime)s - %(levelname)s - %(message)s')

The logging module's basicConfig() function lets you specify what details about the LogRecord object you want to see and how you want those details displayed.
level=logging.DEBUG: This line sets the logging level to DEBUG, which means that all log messages of severity DEBUG and above will be captured. The order of severity levels, from lowest to highest, is DEBUG, INFO, WARNING, ERROR, and CRITICAL.
2b.Explain five buttons available in Debug control window.
Answer:
There are five buttons located at the top left-hand corner of the Debug window: Go, Step, Over, Out, and Quit. These buttons control how the debugger moves through your code.

3a.Write a program to implement the following object diagram and its functionality as shown. Initialize an attribute through a constructor and print the same



Answer:
```
import math class Point:
""" This is a class Point representing a coordinate point
"""

def read_point(p):
p.x=float(input("x coordinate:")) p.y=float(input("y coordinate:"))

def print_point(p): print("(%g,%g)"%(p.x, p.y))

p2=Point()   #create second object print("Enter Second point:")
read_point(p2)      #read x and y for p2

dist=distance(p1,p2)       #compute distance print("First point is:")
print_point(p1)      #print p1 print("Second point is:") print_point(p2)  #print p2

print("Distance is: %g" %(distance(p1,p2)))  #print d
```
Output:
```
        Enter First point: x coordinate:10
        y coordinate:20 Enter Second point: x coordinate:3
        y coordinate:5
        First point is: (10,20)
```

3b.Explain _ _init_ _ ( ) and _ _str_ _ ( ) method in detail.

Answer:

A method init () has to be written with two underscores before and after the word *init*)

Python provides a special method called as __init_() which is similar to constructor method in other programming languages like C++/Java. The term *init* indicates initialization. As the name suggests, this method is invoked automatically when the object of a class is created. Consider the example given here –

```
import math class Point:
def __init__(self,a,b):
        self.x=a
        self.y=b

def dist(self,p2):
d=math.sqrt((self.x-p2.x)**2 + (self.y-p2.y)**2) return d

        def __str_(self):
return "(%d,%d)"%(self.x, self.y)

p1=Point(10,20)#__init__() is called automatically p2=Point(4,5) #__init__() is called automatically

print("P1 is:",p1)#__str__() is called automatically print("P2 is:",p2)#__str__() is called automatically

d=p1.dist(p2)       #explicit call for dist() print("The distance is:",d)
```

The sample output is –

```
P1 is: (10,20)
P2 is: (4,5)
Distance is: 16.15549442140351
```

- The next method inside the class is ***str (). It is a special method used for string representation of user-defined object.*** Usually, print() is used for printing basic types in Python. But, user-defined types (class objects) have their own meaning and a way of representation. To display such types, we can write functions or methods like print_point() as we did in Section 4.1.2. But, more polymorphic way is to use __str_() so that, when we write just print() in the main part of the program, the __str__() method will be invoked automatically. Thus, when we use the statement like –
  print("P1 is:",p1)

the ordinary print() method will print the portion “P1 is:” and the remaining portion is taken care by str () method. In fact, __str__() method will return the string format what we have given inside it, and that string will be printed by print()
method.

4a.Define Pure function and Modifier function. Illustrate with an example Python program.

Answer:

Pure Functions

To understand the concept of pure functions, let us consider an example of creating a class called Time. An object of class Time contains hour, minutes and seconds as attributes. Write a function to print time in HH:MM:SS format and another function to add two time objects. Note that, adding two time objects should yield proper result and hence we need to check whether number of seconds exceeds 60, minutes exceeds 60 etc, and take appropriate action.

```
class Time:
"""Represents the time of a day Attributes: hour, minute, second """

def printTime(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
if sum.second >= 60: sum.second -= 60
sum.minute += 1 if sum.minute >= 60:
sum.minute -= 60
sum.hour += 1 return sum
t1=Time() t1.hour=10 t1.minute=34 t1.second=25 print("Time1 is:") printTime(t1)

t2=Time() t2.hour=2 t2.minute=12 t2.second=41 print("Time2 is :") printTime(t2)

t3=add_time(t1,t2)
print("After adding two time objects:") printTime(t3)
```

Modifiers

Sometimes, it is necessary to modify the underlying argument so as to reflect the caller. That is, arguments have to be modified inside a function and these modifications should be available to the caller. The functions that perform such modifications are known as ***modifier function.*** Assume that, we need to add few seconds to a time object, and get a new time. Then, we can write a function as below –

```
def increment(t, seconds): t.second += seconds

while t.second >= 60: t.second -= 60
t.minute += 1

while t.minute >= 60: t.minute -= 60
t.hour += 1
```

4b.Define a function which takes two objects representing complex numbers and returns a new complex number with an addition of two complex numbers. Define a suitable class Complex' to represent the complex number. Develop a program to read N complex numbers and to compute the addition of N complex numbers.

Answer:
```
class Complex:
   # Constructor to accept real and imaginary part
   def __init__(self, tempReal, tempImaginary):
       self.real = tempReal
       self.imaginary = tempImaginary
```

```
    # Defining addComplex() method for adding two complex number
    def addComplex(self, C1, C2):

        # creating temporary object of class Complex
        temp=Complex(0, 0)
        # adding real part of complex numbers
        temp.real = C1.real + C2.real
        # adding Imaginary part of complex numbers
        temp.imaginary = C1.imaginary + C2.imaginary
        # returning the sum
        return temp
        csum = Complex(0,0)
# csum is the final result initized to 0,0
n = int(input("Enter the number of complex numbers to be added: "))
for i in range(1, n+1):
    # end = "" will replace the default newline with a sting within the " "
    print("Enter real and imaginary part of a complex number  %d :"%i, end=" ")
    c = input().split()
    csum = csum.addComplex(csum,Complex(int(c[0]), int(c[1])))
print("Sum of given Complex Numbers = %d + i%d"%(csum.real, csum.imaginary))
```

5a.What is a class? How to define a class in python? How to initiate a class and how the class members are accessed?

Answer:

Python is an object-oriented programming language, and *class* is a basis for any object oriented programming language. Class is a user-defined data type which binds data and functions together into single entity. Class is just a prototype (or a logical entity/blue print) which will not consume any memory. An object is an instance of a class and it has physical existence. One can create any number of objects for a class. A class can have a set of variables (also known as attributes, member variables) and member functions (also known as methods).

(Overview of general OOP concepts is given at the end of this module as an extra topic. Those who are new to OOP concepts, it is suggested to have a glance and then continue reading).

Programmer-defined Types

A class in Python can be created using a keyword class. Here, we are creating an empty class without any members by just using the keyword pass within it.

```
class Point:
pass print(Point)
```

The output would be –

```
<class ' main_.Point'>
```

The term main indicates that the class Point is in the main scope of the current module. In other words, this class is at the top level while executing the program.

Now, a user-defined data type Point got created, and this can be used to create any number of objects of this class. Observe the following statements –

```
p=Point()
```

Now, a reference (for easy understanding, treat reference as a pointer) to Point object is created and is returned. This returned reference is assigned to the object p. The process of creating a new object is called as ***instantiation*** and the object is ***instance*** of a class. When we print an object, Python tells which class it belongs to and where it is stored in the memory.

```
print(p)
```

The output would be –
```
< main .Point object at 0x003C1BF0>
```
5b. Discuss Operator overloading with an example program
Answer:
Operator Overloading
Ability of an existing operator to work on user-defined data type (class) is known as operator overloading. It is a polymorphic nature of any object oriented programming. Basic operators like +, -, * etc. can be overloaded. To overload an operator, one needs to write a method within user-defined class. Python provides a special set of methods which have to be used for overloading required operator. The method should consist of the code what the programmer is willing to do with the operator. Following table shows gives a list of operators and their respective Python methods for overloading.
```
class Point:
def init (self,a=0,b=0): self.x=a
self.y=b   +

def __add__(self,p2):
        p3=Point()
        p3.x=self.x+p2.x p3.y=self.y+p2.y
        return p3

def __str_(self):
return "(%d,%d)"%(self.x, self.y)


p1=Point(10,20) p2=Point(4,5) print("P1 is:",p1)
print("P2 is:",p2)
p4=p1+p2    #call for add () method print("Sum is:",p4)
```

The output would be –
```
P1 is: (10,20)
P2 is: (4,5)
Sum is: (14,25)
```
6a.Explain Assertions with an example program of how assertions used in traffic light simulation
Answer:
```
def trafficLight():
 signal = input("Enter the colour of the traffic light: ")
 if (signal not in ("RED","YELLOW","GREEN")):
  print("Please enter a valid Traffic Light colour in CAPITALS")
 else:
  value = light(signal) #function call to light()
 if (value == 0):
  print("STOP, Your Life is Precious.")
 elif (value == 1):
  print ("PLEASE GO SLOW.")
 else:
  print("GO!,Thank you for being patient.")
#function ends here

def light(colour):
 if (colour == "RED"):
  return(0);
 elif (colour == "YELLOW"):
  return (1)
 else:
  return(2)
#function ends here

trafficLight()
print("SPEED THRILLS BUT KILLS")
```

6b. List out the benefits of compressing files? Also explain reading of a zip file

- Reduce the size of files and their storage requirements without losing information.
- Improve transfer speed over the network due to reduced size and single-file transfer.
- Pack several related files together into a single archive for efficient management.

To read the contents of a ZIP file, first you must create a ZipFile object (note the capital letters *Z* and *F*). ZipFile objects are conceptually similar to the File objects you saw returned by the open() function in the previous chapter: they are values through which the program interacts with the file. To create a ZipFile object, call the zipfile.ZipFile() function, passing it a string of the *.ZIP* file's filename. Note that zipfile is the name of the Python module, and ZipFile() is the name of the function.
For example, enter the following into the interactive shell:

```
   >>> import zipfile, os

   >>> from pathlib import Path
   >>> p = Path.home()
   >>> exampleZip = zipfile.ZipFile(p / 'example.zip')
   >>> exampleZip.namelist()
   ['spam.txt', 'cats/', 'cats/catnames.txt', 'cats/zophie.jpg']
   >>> spamInfo = exampleZip.getinfo('spam.txt')
   >>> spamInfo.file_size
   13908
   >>> spamInfo.compress_size
   3828
❶ >>> f'Compressed file is {round(spamInfo.file_size / spamInfo
   .compress_size, 2)}x smaller!'
   )
   'Compressed file is 3.63x smaller!'
   >>> exampleZip.close()
```

7a. Explain the concept of copy.copy() and copy.deepcopy() module in class with an example object diagram.
Answer:

An object will be aliased whenever there an object is assigned to another object of same class. This may happen in following situations –

- Direct object assignment (like p2=p1)
- When an object is passed as an argument to a function
- When an object is returned from a function

The last two cases have been understood from the two programs in previous sections. Let us understand the concept of aliasing more in detail using the following program –

```
>>> import copy  #import module copy
>>> p3=copy.copy(p1)   #use the method copy()
>>> print(p1)
< main .Point object at 0x01581BF0>
>>> print(p3)
```

< main .Point object at 0x02344A50>
>>> print(p3.x,p3.y) 10 20
import copy class Point:
""" This is a class Point representing coordinate point
"""

class Rectangle:
""" This is a class Rectangle. Attributes: width, height and Corner Point """

box1=Rectangle() box1.corner=Point() box1.width=100 box1.height=200 box1.corner.x=0
box1.corner.y=0

box2=copy.copy(box1)
print(box1 is box2)   #prints False print(box1.corner is box2.corner) #prints True
7b.Briefly explain the printing of objects with examples.

You can use the python function vars() and pprint() to print current object properties and values in Python. The vars() returns a dictionary containing the object's attributes and their current values. We can then use the pprint() function to print the dictionary in a human-readable format

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

  def __str__(self):
    return f"My name is {self.name} and I am {self.age} years old."

person1 = Person("John", 30, "male")
person2 = Person("Jane", 25, "female")

print(person1)
print(person2)
```