Roll No. | | | | |

Internal Assessment Test III – April 2023

Sub:	Introduction to Python Programming--Scheme and Solution					Sub Code:	BPLCK105B	Branch:	Chemistry Cycle
Date:	05-04-2023	Duration:	90 min's	Max Marks:	50	Sem / Sec:	I / Chemistry Cycle		OBE

	Answer any FIVE FULL QUESTIONS	MARKS	CO	RBT

1(a)	**Explain how to copy, move, and rename files in Python using shutil module methods.**
	Explanation – [2.5 marks]
	Example code– [2.5 marks]
	
	The **shutil** module in Python provides functions for working with files and directories. You can use **shutil** to copy, move, and rename files
	i) **Copying files**: To copy a file from one location to another, use the **shutil.copy(src, dst)** method. The **src** argument is the path of the source file, and the **dst** argument is the path of the destination file. If **dst** is a directory, the file is copied with the same name to that directory
	
	`import shutil`
	`# Copy a file from src to dst`
	`src = 'path/to/source/file.txt'`
	`dst = 'path/to/destination/file.txt'`
	`shutil.copy(src, dst)`
	ii) **Moving files**: To move a file from one location to another, use the **shutil.move(src, dst)** method. The **src** argument is the path of the source file, and the **dst** argument is the path of the destination file. If **dst** is a directory, the file is moved with the same name to that directory
	
	`import shutil`
	`# Move a file from src to dst`
	`src = 'path/to/source/file.txt'`
	`dst = 'path/to/destination/file.txt'`
	`shutil.move(src, dst)`
	Both **shutil.copy()** and **shutil.move()** can handle both files and directories.
	
	iii) **Renaming files**: To rename a file, use the **os.rename(src, dst)** method. The **src** argument is the path of the source file, and the **dst** argument is the new name of the file`import os`
	
	`# Rename a file`
	`src = 'path/to/source/file.txt'`
	`dst = 'path/to/source/new_file.txt'`
	`os.rename(src, dst)`
		5	CO3	L2

(b)	**Write a python program to walk a directory tree and display the files with extensions .txt or .py,** **Full Program: [2 Marks]** **Logic for pattern Matching –[3 Marks]**	5	CO3	L3

Solution :

An example Python program that walks a directory tree and displays the files with extensions **.txt** or **.py**

```
import os
# The root directory to start the search from
root_dir = 'path/to/root/directory'
# Walk the directory tree
for dirpath, dirnames, filenames in os.walk(root_dir):
    for filename in filenames:
        # Check if the file has the .txt or .py extension
        if filename.endswith('.txt') or filename.endswith('.py'):
            # Display the file path
            file_path = os.path.join(dirpath, filename)
            print(file_path)
```

we first set the **root_dir** variable to the path of the directory we want to search in. We then use the **os.walk()** method to iterate over all the files and directories in the **root_dir** directory tree.

For each directory, we loop through the filenames in the **filenames** list. We use the **endswith()** method to check if the filename ends with either the **.txt** or **.py** extension. If it does, we join the directory path with the filename using **os.path.join()** and print the resulting file path.

The **os.walk()** method traverses the entire directory tree, so this program may take some time to complete if you have a large number of files and directories in the **root_dir.**

2(a) **What is compressing files? Explain reading, extracting and creating ZIP files with code snippets**

Explanation – [2.5 marks]

Example code– [2.5 marks]

5 CO3 L3

Solution :

Compressing files means reducing the size of one or more files to save disk space and make them easier to transfer over the internet. One way to compress files is to use a file archive format like ZIP .

A ZIP file is a compressed archive that can contain one or more files and directories. It is a popular file format used for compressing and archiving files on Windows, macOS, and Linux.

We can use the built-in **zipfile** module. Here's how you can read, extract, and create ZIP files with code snippets

1. **Reading a ZIP file** : To read the contents of a ZIP file in Python, you can use the **zipfile.ZipFile** class

Example:

```
import zipfile
# Open the ZIP file
with zipfile.ZipFile('example.zip', 'r') as zip_file:
    # Print the list of files in the ZIP file
    print(zip_file.namelist())
    # Print the contents of a specific file in the ZIP file
    with zip_file.open('example.txt') as file:
        print(file.read())
```

we use the **with** statement to open the ZIP file in read mode. We then use the **namelist()** method to print the list of files in the ZIP file. Finally, we use the **open()** method to open a specific file in the ZIP file and read its contents

ii) Extracting a ZIP file : To extract the contents of a ZIP file in Python, you can use the **zipfile.ZipFile.extractall()** method

Example:

```
import zipfile
```

	# Open the ZIP file
	with zipfile.ZipFile('example.zip', 'r') as zip_file:
	# Extract all the files in the ZIP file to a directory
	zip_file.extractall('extracted_files')
	iii) Creating a ZIP file : To create a new ZIP file in Python, you can use the **zipfile.ZipFile** class in write mode.
	Example
	import zipfile
	# Create a new ZIP file
	with zipfile.ZipFile('new.zip', 'w') as zip_file:
	# Add a file to the ZIP file
	zip_file.write('example.txt')
	we use the **with** statement to create a new ZIP file in write mode. We then use the **write()** method to add a file called '**example.txt**' to the ZIP file.			

(b)	**Write a program to rename the filename .contains American style dates to European style dates in the working directory?**
	Full Program: [2 Marks]
	Logic for pattern Matching –[3 Marks]
	Solution:
	import os
	import re
	# Regular expression pattern to match American style dates
	date_pattern = re.compile(r'([01]?\d)[-/.]([0-3]?\d)[-/.]((?:19|20)\d{2})')
	# Iterate over all files in the working directory
	for filename in os.listdir():
	# Check if the filename contains an American style date
	if date_pattern.search(filename):
	# Rename the file with a European style date
	new_filename = date_pattern.sub(r'\2-\1-\3', filename)
	os.rename(filename, new_filename)
	print(f'Renamed {filename} to {new_filename}')
	we first define a regular expression pattern to match American style dates. The pattern matches dates in the format **MM/DD/YYYY**, **MM.DD.YYYY**, or **MM-DD-YYYY.**
	We then use the **os.listdir()** function to iterate over all the files in the working directory. For each file, we use the **search()** method of the regular expression pattern to check if the filename contains an American style date.
	If the filename contains an American style date, we use the **sub()** method of the regular expression pattern to replace the date with a European style date in the format **DD-MM-YYYY**. We then use the **os.rename()** function to rename the file with the new filename. we print a message to the console indicating that the file has been renamed.
		5	CO3	L3

3(a)	**What is the use of send2trash module's send2trash method? How is it different from os.unlink()? Explain with code snippet.** **Explanation – [2.5 marks]** **Example code– [2.5 marks]** The **send2trash** module's **send2trash** function provides a way to delete files or directories by sending them to the operating system's trash or recycle bin instead of permanently deleting them with the **os.unlink** function. This can be useful to avoid accidental deletion of files, as files in the trash or recycle bin can be easily recovered if needed. example code snippet: import os	5	CO3	L2

	from send2trash import send2trash
	
	# create a file to be deleted
	with open('example.txt', 'w') as f:
	f.write('Hello, world!')
	
	# delete the file using os.unlink
	os.unlink('example.txt')
	
	# check if the file exists (should raise FileNotFoundError)
	if os.path.exists('example.txt'):
	print('File exists')
	else:
	print('File does not exist')
	
	# create the file again
	with open('example.txt', 'w') as f:
	f.write('Hello, world!')
	
	# delete the file using send2trash
	send2trash('example.txt')
	
	# check if the file exists (should return False)
	if os.path.exists('example.txt'):
	print('File exists')
	else:
	print('File does not exist')
	
	Here, we create a file called 'example.txt', and then delete it using both **os.unlink** and **send2trash**. After each deletion, we check if the file exists using **os.path.exists**. With **os.unlink**, the file is permanently deleted and **os.path.exists.**
	In this code, raises a **FileNotFoundError**. With **send2trash**, the file is moved to the trash or recycle bin and **os.path.exists** returns False.
	the main difference between **send2trash** and **os.unlink** is that **send2trash** moves files to the trash or recycle bin instead of permanently deleting them, which can be useful for avoiding accidental deletion and allowing for easy recovery if needed.			

(b)	**Define assertions. What does an assert statement in python consists of? Explain how assertions can be used with Python code snippets.** **Explanation – [2.5 marks]** **Example code– [2.5 marks]** Solution: **Assertions** are statements in Python that can be used to check if a condition is true. They are often used as a debugging aid to ensure that the assumptions made by the programmer about the state of the program are correct. If the assertion is false, an **AssertError** is raised, indicating that there is a bug in the program. An **assert statement** consists of the **assert** keyword, followed by a condition that is expected to be true. ***Example.*** x = 5 assert x == 5, "x is not 5" Here, we use the assert statement to check that the variable x has the value 5. If the condition **x == 5** is false, an **AssertionError** is raised with the message **"x is not 5"**	5	CO3	L2

	Assertions can be used to add automated tests in codes to catch bugs early in the development process. For example, we can use assertions to check that the output of a function is correct for a given input. def square(x): return x ** 2 assert square(2) == 4, "square(2) should be 4" assert square(3) == 9, "square(3) should be 9" assert square(-2) == 4, "square(-2) should be 4" Here, we define a function **square** that returns the square of a number. We then use assert statements to check that the output of the function is correct for different input values.			

4(a)	Differentiate Assertions and Exceptions **4 points 4 marks**		4	CO3	L2
	Assertion	**Exception**			
	An assertion is a sanity check to make sure your code isn't doing something obviously wrong.	Exceptions, on the other hand, are used to handle unexpected situations that occur during program execution, such as input errors or system failures.			

	In code, an assert statement consists of the following:
	The assert keyword
	A condition (that is, an expression that evaluates to True or False)
	A string to display when the condition is False
	>>> ages = [26, 57, 92, 54, 22, 15, 17, 80, 47, 73]
	>>> ages.sort()
	>>> ages
	[15, 17, 22, 26, 47, 54, 57, 73, 80, 92]
	>>> assert ages[0] <= ages[-1] "Sorting is wrong"
	# Assert that the first age is <= the last age.
		Try except block is used to handle exception
		def spam(n):
		try:
		return 42/n
		except :
		print("Divide by Zero Error")
		
		print(spam(12))
		print(spam(42))
		print(spam(0))
		print(spam(21))			

	Unlike exceptions, your code should not handle assert statements with try and except; if an assert fails, your program should crash.	When an exception is raised, the normal flow of the program is interrupted and control is transferred to an exception handler.			
	Assertions are for programmer errors, not user errors. Assertions should only fail while the program is under development; When an assertion fails, an AssertionError is raised, indicating that the code is not behaving as expected	.Exceptions are often used to gracefully handle errors and recover from unexpected situations, allowing the program to continue running			
(b)	Write a Python program to convert hours, minutes and seconds into seconds using class [Define Time Class with attribute hour, minute, second] Sample input / output		6	CO4	L3

	Enter time hrs, min and sec : 01 34 50
	Time in seconds : 5690
	Solution:
	class Time:
	def __init__(self, hour, minute, second):
	self.hour = hour
	self.minute = minute
	self.second = second
	
	def to_seconds(self):
	return self.hour * 3600 + self.minute * 60 + self.second
	
	# Get input from user
	hours, minutes, seconds = input("Enter time hrs, min and sec: ").split()
	hours, minutes, seconds = int(hours), int(minutes), int(seconds)
	
	# Create a Time object and convert to seconds
	time_obj = Time(hours, minutes, seconds)
	total_seconds = time_obj.to_seconds()
	
	# Display output
	print("Time in seconds:", total_seconds)
	
	Output:
	
	Enter time hrs, min and sec: 01 34 50
	Time in seconds: 5690			

5(a)	Implement a Time class with methods for following features:
	i) To display time in HH:MM:SS with __str__ () method
	ii) To add two time objects (using operator overloading)
	Full Program: [3 Marks]
	Logic for overloading –[3 Marks]
	
	Solution:
	***i)* To display time in HH:MM:SS with __str__ () method**
	class Time:
	def __init__(self, hour, minute, second):
	self.hour = hour
	self.minute = minute
	self.second = second
	
	def __str__(self):
	return f"{self.hour:02}:{self.minute:02}:{self.second:02}"
	
	def __add__(self, other):
	total_seconds = self.to_seconds() + other.to_seconds()
	return Time.from_seconds(total_seconds)
	
	def to_seconds(self):
	return self.hour * 3600 + self.minute * 60 + self.second
	
	def from_seconds(cls, seconds):
	hour, remaining_seconds = divmod(seconds, 3600)
	minute, second = divmod(remaining_seconds, 60)
	return cls(hour, minute, second)
		6	CO4	L3

	ii) To add two time objects (using operator overloading)
	# Create two Time objects
	time1 = Time(1, 30, 0)
	time2 = Time(0, 45, 15)
	
	# Display the Time objects using the __str__() method
	print(f"time1: {time1}")
	print(f"time2: {time2}")
	
	# Add the Time objects using the __add__() method
	total_time = time1 + time2
	
	# Display the result using the __str__() method
	print(f"total time: {total_time}")
	Output:
	time1: 01:30:00
	time2: 00:45:15
	total time: 02:15:15			

(b)	Differentiate pure functions and modifiers with example.
	4 Points at least – 4 Marks
	Solution:
	In Python, functions can be classified into two main categories: pure functions and modifiers (also known as impure functions).
	
	Pure functions are functions that do not modify the input arguments and do not have any side effects. They always return the same output given the same input arguments. Pure functions are predictable and easy to test.
	
	On the other hand, modifiers are functions that modify the input arguments and/or have side effects. They can return a value or not, but their primary purpose is to modify the state of the program. Modifiers are unpredictable and difficult to test.
	
	Here are examples of each type of function:
	
	Example of a pure function:
	
	python
	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
	This function takes two arguments and returns their sum. It does not modify the input arguments or have any side effects. It will always return the same output given the same input arguments, making it a pure function.
	
	Example of a modifier function:
		4	CO4	L2

	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
	
	This function takes t which is a time object. It modifies the attributes(second, minute and hour) of t object hence this function increment is modifier function			

6(a)	Explain __init__ () and __str__() methods with example program.
	Explanation – [2.5 marks]
	Example code– [2.5 marks]
	
	Solution:
	In Python, __init__() and __str__() are two special methods that are commonly used in classes.
	
	__init__() is a constructor method in Python that is used to initialize the object's attributes. It is called when an instance of the class is created. The self parameter in __init__() refers to the instance of the class that is being initialized, and can be used to set attributes for that instance.
	
	__str__() is a special method in Python that is used to define a string representation of an object. It is called when the str() function is called on an object. The self parameter in __str__() refers to the instance of the class, and can be used to access its attributes and return a string representation of the object.
	
	Here is an example program that demonstrates the use of __init__() and __str__() methods in Python:
	
	python
	
	class Person:
	def __init__(self, name, age):
	self.name = name
	self.age = age
	
	def __str__(self):
	return f"{self.name} ({self.age} years old)"
	
	person1 = Person("John", 30)
	person2 = Person("Jane", 25)
	
	print(person1) # Output: John (30 years old)
	print(person2) # Output: Jane (25 years old)
	In this program, we define a Person class that has two attributes: name and age. We use the __init__() method to initialize these attributes when an instance of the class is created.
	
	We also define the __str__() method to return a string representation of the object. When we print an instance of the Person class using the print() function, the __str__() method is called to return a string representation of the object.
		5	CO4	L3

(b)	What is operator overloading? Write a program to add to Point objects by overloading + operator. Also overload __str__() to display point as an ordered pair.
	Explanation – [2.5 marks]
	Example code– [2.5 marks]
	
	Solution:
	Operator overloading is the ability of a programming language to define operators for user-defined types or objects. In Python, the "+" operator can be overloaded to add two objects of a user-defined class.
	
	Here's an example program that demonstrates operator overloading for a Point class:
	
	python
	
	class Point:
	def __init__(self, x=0, y=0):
	self.x = x
	self.y = y
	
	def __add__(self, other):
	return Point(self.x + other.x, self.y + other.y)
	
	def __str__(self):
	return f'({self.x}, {self.y})'
	
	p1 = Point(2, 3)
	p2 = Point(-1, 5)
	p3 = p1 + p2
	
	print(p1) # Output: (2, 3)
	print(p2) # Output: (-1, 5)
	print(p3) # Output: (1, 8)
	In this example, we define a Point class with __add__() and __str__() methods. __add__() method is used to overload the "+" operator to add two Point objects, while __str__() method is used to overload the str() function to display the Point object as an ordered pair.
	
	We create two Point objects p1 and p2, and then add them together using the "+" operator, which calls the __add__() method of the Point class. The resulting Point object is stored in p3 and then displayed using the __str__() method.
		5	CO4	L3

7(a)	Explain the use of following debugging functions with example code snippets i) isinstance() ii) hasattr() iii) vars() **Explanation – [2.5 marks]** **Example code– [2.5 marks]** ***Solution:*** The isinstance() function is used to determine whether an object is an instance of a specific class or not. It returns True if the object is an instance of the specified class, otherwise returns False. Example:	5	CO4	L3

	python
	
	class Person:
	  def __init__(self, name, age):
	  self.name = name
	  self.age = age
	
	p = Person("John", 36)
	print(isinstance(p, Person)) # Output: True
	print(isinstance("hello", Person)) # Output: False
	hasattr() function:
	The hasattr() function is used to determine whether an object has a given named attribute or not. It returns True if the object has the named attribute, otherwise returns False.
	Example:
	
	python
	
	class Person:
	  def __init__(self, name, age):
	  self.name = name
	  self.age = age
	
	p = Person("John", 36)
	print(hasattr(p, "name")) # Output: True
	print(hasattr(p, "gender")) # Output: False
	vars() function:
	The vars() function returns the __dict__ attribute of an object. It returns a dictionary containing the object's attributes.
	Example
	
	class Person:
	  def __init__(self, name, age):
	  self.name = name
	  self.age = age
	
	p = Person("John", 36)
	print(vars(p)) # Output: {'name': 'John', 'age': 36}			

(b)	Differentiate aliasing & copying objects and shallow copy & deep copy with example program in Python.
	Explanation – [2.5 marks]
	Example code– [2.5 marks]
	
	Solution:
	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
	>>> class Point:
	pass
	
	>>> p1=Point()
	>>> p1.x=10
		5	CO4	L3

```
>>> p1.y=20
>>> p2=p1
>>> print(p1)
        < main .Point object at 0x01581BF0>
>>> print(p2)
        < main .Point object at 0x01581BF0>
```

If we need a copy of an object, but not an alias, do this, Python provides a module called copy and a method called copy(). Consider the below given program to understand the concept.

```
>>> class Point:
        pass

>>> p1=Point()
>>> p1.x=10
>>> p1.y=20

>>> 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)
```

This is called shallow copying.
The copy() method of copy module duplicates the object.

```
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
```

If we use copy.copy()inner object point will not be copied. It will be shared by box1 and box2 objects. That's is the reason second print statement output True
If we use copy.deepcopy() inner object point(corner) is also copied and occupy separate memory. This is deep copying.

```
box3=copy.deepcopy(box1)
print(box1 is box3)#prints False
 print(box1.corner is box3.corner)
#prints False
```