Python OOPs Concept

Introduction:

Object-Oriented Programming (OOP) is a programming paradigm based on the concept of "objects," which can contain data in the form of fields (attributes or properties) and code in the form of procedures (methods). The main concepts of OOP include:

Class: A blueprint for creating objects. A class defines a datatype by bundling data and methods that work on the data into one single unit.

Object: An instance of a class. It is created using the class blueprint and can use the class's methods and properties.

Inheritance: A mechanism for creating a new class using the properties and methods of an existing class. The new class is called a subclass or derived class, and the existing class is called a superclass or base class.

Polymorphism: The ability of different classes to be treated as instances of the same class through a common interface. It is often used to implement method overriding, where a subclass provides a specific implementation of a method that is already defined in its superclass.

Encapsulation: The bundling of data and methods that operate on the data within one unit, typically a class. This concept restricts direct access to some of the object's components, which can prevent the accidental modification of data.

Abstraction: The concept of hiding the complex implementation details and showing only the necessary features of an object. This is achieved through abstract classes and interfaces.

These concepts help in organizing code, making it more modular, reusable, and easier to maintain. The questions cover the implementation of classes and inheritance, which are key aspects of OOP.

Question 1: (5 Marks)

Build a program to manage a university's course catalog. You want to define a base class Course that has the following properties:

course_code: a string representing the course code (e.g., "CS101")

course_name: a string representing the course name (e.g., "Introduction to Computer Science")

credit_hours: an integer representing the credit hours for the course (e.g., 3)

You also want to define two subclasses CoreCourse and ElectiveCourse, which inherit from the Course class.

CoreCourse should have an additional property required_for_major which is a boolean representing whether the course is required for a particular major.

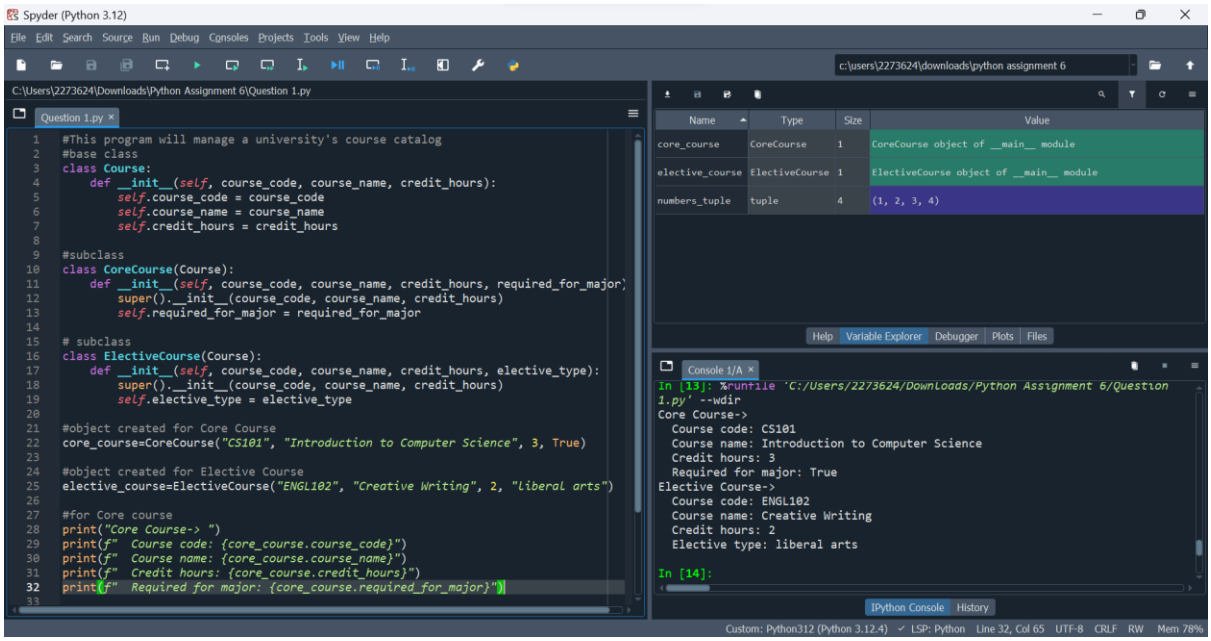
ElectiveCourse should have an additional property elective_type which is a string representing the type of elective (e.g., "general", "technical", "liberal arts").

a. Description: This question requires the implementation of a program to manage a university's course catalog using OOP principles. Specifically, you are asked to create a base class Course and two subclasses CoreCourse and ElectiveCourse.
b. Advantage: This approach provides modularity, reusability, and maintainability. The program is divided into separate classes, each handling specific aspects of the course catalog. Properties and methods of the base class Course can be reused in the subclasses.
c. Code:

```
#This program will manage a university's course catalog
#base class
class Course:
  def __init__(self, course_code, course_name, credit_hours):
    self.course_code = course_code
    self.course_name = course_name
    self.credit_hours = credit_hours

#subclass
class CoreCourse(Course):
  def __init__(self, course_code, course_name, credit_hours,
required_for_major):
    super().__init__(course_code, course_name, credit_hours)
    self.required_for_major = required_for_major

# subclass
class ElectiveCourse(Course):
  def __init__(self, course_code, course_name, credit_hours, elective_type):
    super().__init__(course_code, course_name, credit_hours)
    self.elective_type = elective_type

#object created for Core Course
core_course=CoreCourse("CS101", "Introduction to Computer Science", 3, True)
```

```
#object created for Elective Course
elective_course=ElectiveCourse("ENGL102", "Creative Writing", 2, "liberal arts")

#for Core course
print("Core Course-> ")
print(f"  Course code: {core_course.course_code}")
print(f"  Course name: {core_course.course_name}")
print(f"  Credit hours: {core_course.credit_hours}")
print(f"  Required for major: {core_course.required_for_major}")

#for Elective course
print("Elective Course-> ")
print(f"  Course code: {elective_course.course_code}")
print(f"  Course name: {elective_course.course_name}")
print(f"  Credit hours: {elective_course.credit_hours}")
print(f"  Elective type: {elective_course.elective_type}")
```

- Question1.py screenshot.



Question 2: (5 Marks)

Create a Python module named employee that contains a class Employee with attributes name, salary and methods get_name() and get_salary(). Write a program to use this module to create an object of the Employee class and display its name and salary.

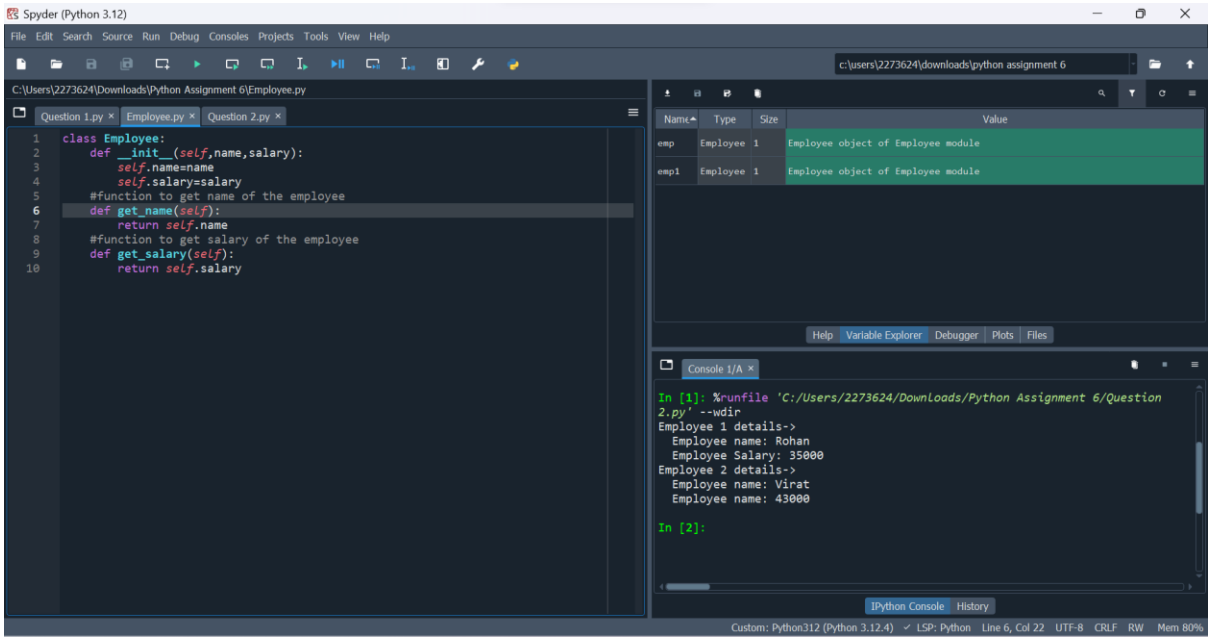
a. Description: This question requires creating a Python module named employee that defines an Employee class with attributes and methods. The module is then used to create an Employee object and display its attributes.
b. Advantage: This approach provides encapsulation, reusability, and modularity. The Employee class encapsulates the attributes and methods, providing a clear structure for managing employee data. The Employee class can be reused in different programs by importing the employee module.
c. Code:
   - Employee.py(module)

```
class Employee:
  def __init__(self,name,salary):
    self.name=name
    self.salary=salary
  #function to get name of the employee
  def get_name(self):
    return self.name
  #function to get salary of the employee
  def get_salary(self):
    return self.salary
```

   - Question2.py

```
#importing employee class from Employee module
from Employee import Employee

#object for Employee 1
emp=Employee("Rohan",35000)
# object for Employee 2
emp1=Employee("Virat", 43000)

#for Employee 1
print("Employee 1 details->")
print(f"  Employee name: {emp.get_name()}")
print(f"  Employee Salary: {emp.get_salary()}")

#for Employee 2
print("Employee 2 details->")
print(f"  Employee name: {emp1.get_name()}")
print(f"  Employee name: {emp1.get_salary()}")
```

- Employee.py screenshot.



- Question2.py screenshot.