Assignment-03

(Object-Oriented Programming)

Submission Deadline:	**10PM 9th November 2025**
Submission Link:	https://forms.office.com/r/1nrW4F6Ls3
Instructions	1- All answers must be written on A4 sheets. 2- Only handwritten submissions will be accepted. 3- Write your Name and Roll No. at the top of every page.

Q1. Create a class Book with attributes title, author, and price.
Add a method display_info() that prints all details of the book.

Conceptual Part:
What is the difference between a **class** and an **object** in Python?

Q2. Define a class Student with attributes name, roll_no, and marks.
Implement the __init__ and __str__ methods to automatically print a formatted description when a Student object is printed.

Conceptual Part:
Explain the role of the __init__ and __str__ methods in class design.

Q3. Create a class Circle with a class variable pi = 3.14159.
Include:

- An instance method to calculate the area.
- A static method is_valid_radius(r) that checks if the radius is positive.
- A class method unit_circle() that returns a Circle object with radius 1.

Conceptual Part:
Differentiate between **class methods** and **static methods**.

Q4. Create:

- Class Person → attributes: name, age
- Class Employee(Person) → adds emp_id, salary
- Class Manager(Employee) → adds department

Write a method display() in each class and show how multilevel inheritance works.

Conceptual Part:
What is **method overriding**, and how does Python determine which method to call?

Q5. Create two classes A and B, each having a method show().
Create class C(A, B) that inherits both.
Create an object of C and call show().

Q6. Create a class Engine with a method start().
Then create a class Car that ***uses*** *an Engine object instead of inheriting from it.*
Demonstrate composition in your code.

Conceptual Part:
When should you prefer **composition** over **inheritance**?

Q7. Use the abc module to create an abstract class Shape with an abstract method area().
Create subclasses Rectangle and Circle implementing the area() method.

Conceptual Part:
Why are **abstract classes** used, and how are they different from **interfaces**?

Q8. Create two classes Dog and Cat, both having a speak() method.
Write a function animal_sound(animal) that calls speak() on any object passed to it.

Conceptual Part:
Explain **duck typing** and how Python achieves polymorphism without explicit interfaces.

Q9. Create a class BankAccount with private attributes _balance.
Provide:

- A property balance for reading the balance.
- A setter method that ensures balance cannot be negative.

Conceptual Part:
How do **property decorators** help achieve encapsulation?

Q10. Create a class SecureData with attributes that are hidden (name starts with __).
Override __getattr__ and __setattr__ to print messages when accessing or modifying attributes.

Conceptual Part:
Differentiate between **data hiding** and **abstraction** in object-oriented programming.
