DAY 8 … FINAL TASK 8:
OOP PRINCIPLES & CODE-BASED PROJECT

✓ Assigned Final Task

🔷 Part 1: Theoretical (Pure Conceptual Tasks)

Answer the following **clear and to-the-point questions** based on all the OOP topics.

These are grouped by category and will help cement your understanding.

❖ Section A: Core OOP Concepts

1. Define the term *object* in your own words and give a real-life analogy.
2. What is the difference between a class and an object?
3. Explain the 4 principles of OOP with short definitions.
4. Why is OOP considered modular and maintainable?
5. How is encapsulation different from data hiding?

❖ Section B: Constructors & Destructors

6. What is the purpose of the __init__() method in Python?
7. Can a class have more than one constructor in Python? If yes, how?
8. What is the role of __del__() in memory management?
9. Why is the destructor rarely used in Python?

❖ Section C: Methods in Classes

10. What’s the difference between an instance method and a static method?
11. When should you use a class method over a static method?
12. How is self used inside a method, and why is it important?
13. When do we use super() in OOP?
14. Explain method overriding with an example scenario.
15. How can you perform method overloading in Python without support for it natively?

❖ Section D: Special Methods & Keywords

16. What is the role of __str__() and how is it different from __repr__()?
17. Define the use of super() in multiple inheritance.
18. What is a callable object and how can you implement it?
19. What are dunder methods, and why are they powerful?

❖ Section E: Access Specifiers

20. Define and differentiate public, protected, and private access in Python.
21. How can you access a private attribute from outside the class?
22. What is name mangling in the context of private variables?

❖ Section F: Inheritance Types

23. Define multiple inheritance with an example.
24. What is multilevel inheritance? Explain using a family tree analogy.
25. Differentiate between hierarchical and hybrid inheritance.
26. What is the method resolution order (MRO), and how does Python resolve conflicts?
27. What problems can arise in hybrid inheritance?

❖ **Section G: Advanced OOP Concepts**

28. Differentiate between **aggregation** and **composition**.
29. What is duck typing and how does it reflect Python's flexibility?
30. What is operator overloading? Name 3 magic methods used for this purpose.
31. What is dynamic method dispatch and how does Python implement it?
32. What are abstract classes? How do they enforce implementation?
33. What is an interface in Python (via ABCs)?
34. Explain the use of @property, @setter, and @deleter.
35. What are class vs static variables? Give an example of each.
36. What is the unified type system in Python?
37. What is the difference between object-based and object-oriented languages?
38. What is a metaclass? How is it different from a class?
39. What is a singleton pattern and when is it used?
40. How can you create a factory method in OOP?
41. How does Python implement iterable protocols using __iter__ and __next__?
42. What are some best practices for writing clean OOP code in Python?
43. What is the SOLID principle? Name all five principles briefly.
44. How can OOP code be tested using pytest?
45. How can modules and packages help organize OOP code?
46.

Note:

It's one of the best ways to **test yourself honestly**.
Try to complete today's **theoretical task** *on your own first*, **without using any AI**. This will help you understand exactly **where you stand** which concepts you truly understand, and which ones need more clarity.
Once you've given it a genuine try and still feel stuck, **then you can use AI as a tool to learn better**, not as a shortcut. **Challenge yourself first. Growth begins there**

Last Practical Coding Task

➢ **Scenario: School Management System**

A simple school system where you manage Teachers and Students using Classes and OOP principles.

🔷 Step-by-Step Task Breakdown

❖ Step 1: Encapsulation (Data Hiding)

- Create a class Person
- Keep the data members _name and __age (protected & private)
- Create get_age() and set_age() methods to access/update age safely.

❖ Step 2: Inheritance

- Create two classes Student and Teacher that inherit from Person.
- Add extra fields:
  Student → student_id, grade
  Teacher → subject, salary

❖ Step 3: Abstraction

- Use the ABC module and make an abstract class SchoolMember
- Add abstract methods like show_details() in it.
- Inherit Student and Teacher from SchoolMember

❖ Step 4: Polymorphism

- Define show_details() method in both Student and Teacher to show their respective data.
- Create a loop that stores both objects and calls show_details() this demonstrates polymorphism.

❖ **Code Requirements:**

- Use __init__() constructor in every class
- Use super() to reuse constructor of the base class
- Use __str__() method in at least one class
- Use a custom method greet() in both Student and Teacher to show method overriding

---

🕒 **Deadline:**

Submit your work by **Tommorrow – 11:59 PM**.

Viva Tips (Prepare These):

1. What is encapsulation? Give example.
2. Difference between abstraction and encapsulation?
3. How did you implement inheritance in this task?
4. What is method overriding? Explain with your code.
5. How does your code show polymorphism?

Final Note……..From Yusra Saleem to You:

Congratulations 🎓

You've reached the final day of your OOP journey. You've built a rock solid foundation in Object-Oriented Programming.

Remember🙌, *concepts get stronger only through practice and explanation.*

So during your viva, **stay confident**, refer to **real code** examples from your task, and **don't panic if something is tricky** explain what you understand.

You've done great so far. Now go and **finish strong**💗 Allah Hafiz ...Tack Care✨
