Theoretical Questions

1. What is Object-Oriented Programming (OOP) ?

Object-Oriented Programming (OOP) is a way of designing programs by creating 'objects' that combine data (attributes) and actions (methods), making code organized, reusable and easier to manage.

2. What is a class in OOP ?

In OOP, a class is a blueprint or template for creating objects, defining the data (attributes) and actions (mmethods) that those objects will have.

3. What is an object in OOP ?

In OOP, an **object** is an instance of a class that contains its own data and can perform actions defined by the class.

4. What is the difference between abstraction and encapsulation ?

- **Abstraction :** Abstraction is about **hiding implementation details** and showing only the essential features to the user.
  Example : We use TV remote without knowing how the internal circuits work.
- **Encapsulation :** Encapsulattion is about **wrapping data and methods** together inside a class and restricting direct acess to the data.
  Example : The TV's internal components are enclosed inside the TV body, preventing direct tampering.

5. What are dunder methods in Python ?

Dunder Methods (short of **double underscore methods**) in Python are special built-in methods that start and end with double underscore (**method**).
They let us define how objects of our class behave in certain situations (e.g, printing, addition, comparisons etc.)

Example :

```python
class School:
    def __init__(self, name):  # called when creating an object
        self.name = name

    def __str__(self):  # called when printing an object
        return (f'School object with name {self.name}')

obj = School('Delhi Public School')
print(obj)
```

```
School object with name Delhi Public School
```

6. Explain the concept of inheritance in OOP.

Inheritance in OOP is a feature that allows one class (child or subclass) to use the properties and methods of another class (parent or superclass).
It helps to **reuse code** and create a hierarchy between classes.

- the child class can **use, extend** or **override** the parent class's behavior.

7. What is polymorphism in OOP ?

Polymorphism in OOP means the ability of different classes to respond to the same method name in different ways.

- It allows the **same function or method call** to behave differently depending on the object it is acting on.

8. How is encapsulation achieved in Python ?

In Python **encapsulation** is achieved by **restricting direct access** to an object's data and controling it through methods.

This is done using access modifiers:

1. **Public →** No underscore (accessible everywhere)
2. **protected →** Single underscore '_var' (convention: should not be accessed directly outside the class.)
3. **Private →** Double underscore '__var' (name mangling to prevent diect access).

   Example :

```python
class Person:
    def __init__(self, name, age):
        self.name = name           # Public
        self._address = 'unknown'  # Protected
        self.__salary = 50000      # Private

    def get_salary(self):
        return self.__salary   # Access via method

p = Person('john', 30)

print(p.name)          # Public
print(p._address)      # Possible but discouraged.
# print(p.__salary)    # Error
print(p.get_salary()) # Access via method
```

```
john
unknown
50000
```

9. What is a constructor in Python ?

In Python, a **constructor** is a special method named __ *init*__ that is automatically called when a new object of a class is created.

It's used to **initialize the object's attributes**.

12. What is method overriding in OOP ?

Method Overriding is OOP is when a subclass provides a specific implementation of a method that is already defined in its parent class.

- The overridden method in the child has the **same name, parameters and return type** as in the parent class but performs a different task.

13. What is a property decorator in Python ?

The '@property decorator' in Python is used to covert a class method into a read-only attribute. It alows you to define getters (and optionally setters and deleters) to access and modify rivate attributes in a controlled way, supporting **encapsulation, data hiding** and making the class interface cleaner and more aesthetic.

14. Why is polymorphism important in OOP ?

Polymorphism in OOP allows the same method name to behave differently based on the object calling it, enabling flexibility, code reusability, and easier maintenance.

15. What is an abstract class in Python ?

An abstract class in Python is a class that cannot be instantiated and is meant to be inherited by other classes. It can define abstract methods using the @abstractmethod decorator from the abc module, which must be implemented by its subclasses, enforcing a common interface and promoting consistent design.

16. What are the advantages of OOP ?

The advantages of OOP include better code organization through classes and objects, reusability via inheritance, flexibility through polymorphism, data protection using encapsulation, and improved maintainability and scalability by modeling real-world entities in a modular and structured way.

17. What is the difference between a class variable and an instance variable ?

A **class variable** is shared by all instances of a class and is defined outside of any instance methods, while an **instance variable** is unique to each object and is defined inside methods using self. Class variables maintain a common value across all objects, whereas instance variables store data specific to each object.

18. What is multiple inheritance in Python ?

Multiple inheritance in Python is a feature where a class can inherit from **more than one parent class**, allowing it to access attributes and methods from all its base classes. It enables code reuse across multiple classes but can introduce complexity, especially with method resolution order (MRO) when there are conflicts or overlapping methods.

19. Explain the purpose of ‘’**str**’ and ‘**repr**’ ‘ methods in Python.

The __str__ method in Python defines the human-readable string representation of an object, used by functions like print(), while __repr__ returns an unambiguous, developer-focused string that ideally can recreate the object and is used by default in the interpreter. Both help in customizing how objects are displayed for readability and debugging.

20. What is the significance of the ‘super()’ function in Python ?

The super() function in Python is used to call methods from a parent class within a child class, allowing access to inherited methods without explicitly naming the parent. It is especially useful in multiple inheritance to ensure the correct method resolution order (MRO) and helps maintain cleaner, more maintainable code when extending functionality.

21. What is the significance of the **del** method in Python?

The __del__ method in Python is a special method called when an object is about to be destroyed, typically used to perform cleanup tasks like closing files or releasing resources. It acts as a destructor but should be used carefully, as its execution timing is determined by Python's garbage collector and may not be predictable.

22. What is the difference between @staticmethod and @classmethod in Python ?

In Python, @staticmethod defines a method that belongs to the class but doesn't access or modify class or instance data, making it act like a regular function placed inside a class. In contrast, @classmethod takes the class (cls) as its first argument and can access or modify class-level data, making it useful for alternative constructors or class-wide changes.

23. How does polymorphism work in Python with inheritance ?

In Python, polymorphism with inheritance allows subclasses to override methods of a parent class, enabling different behaviors for the same method name depending on the object's class. This means a single interface can work with objects of different types, supporting dynamic method resolution and flexible, reusable code.

24. What is method chaining in Python OOP ?

Method chaining in Python OOP is a technique where multiple methods are called on the same object in a single line by having each method return the object itself (self). This allows for cleaner and more readable code by linking method calls together sequentially.

25. What is the purpose of the **call** method in Python ?

The __call__ method in Python allows an object to be called like a function. When defined in a class, it lets you use the object’s instance followed by parentheses (e.g., obj()) to execute custom behavior, making objects behave like callable functions.