**Python programming OOPS Theory Questions.**

1. Explain the difference between a class and an object in Python.
2. What is the purpose of the \_\_init\_\_ method in Python?
3. What is the difference between instance variables and class variables?
4. How can you create an object of a class in Python?
5. What is the purpose of the self keyword in Python?
6. Explain the concept of inheritance in Python.
7. What is method overriding? Provide an example in Python.
8. What is polymorphism in Python? Can you provide an example?
9. What is encapsulation, and how is it achieved in Python?
10. What are private and public members in Python?
11. Explain the concept of multiple inheritance in Python.
12. What is an abstract class in Python? How is it defined and used?
13. What is the difference between a class method and an instance method?
14. What is the @staticmethod decorator used for in Python?
15. What is the @classmethod decorator used for in Python?
16. What are magic methods in Python? Provide examples.
17. Explain the concept of the super() function in Python.
18. How does Python handle method resolution order (MRO) in case of multiple inheritance?
19. What is the difference between shallow copy and deep copy in Python?
20. What is the purpose of the \_\_del\_\_ method in Python?
21. What is a static method? How is it different from a class method and instance method?
22. How would you prevent an instance variable from being modified outside the class?
23. Explain how the isinstance() function works in Python.
24. What are abstract methods, and how do you define them in Python?
25. How does Python handle class inheritance with respect to method overriding and method resolution order (MRO)?
26. What is the use of the abc module in Python?
27. What is a metaclass in Python, and how is it used?
28. Can you override the \_\_str\_\_ method in a Python class? What does it do?
29. What is the difference between \_\_call\_\_ and \_\_init\_\_ methods in Python?
30. What is the difference between @property and @staticmethod decorators in Python?
31. What is a class-level attribute in Python, and how does it differ from an instance-level attribute?
32. What are the advantages of using encapsulation in Python?
33. How would you implement operator overloading in Python?
34. Explain how you would use a property to create getter and setter methods in Python.

**Programming Questions**

1. Create a base class Animal with methods speak and move. Derive Dog and Bird classes and override the speak method. Demonstrate polymorphism.
2. Write a Python program with a Student class that has an \_\_init\_\_ method for initializing attributes and a destructor method (\_\_del\_\_).
3. Create a class Employee with a class variable company\_name and instance variables like name, salary. Demonstrate class variables vs instance variables.
4. Create a BankAccount class with a private balance attribute and provide public getter and setter methods to interact with it (encapsulation).
5. Demonstrate multiple inheritance with a Person class and an Employee class, where Employee inherits from Person.
6. Demonstrate Method Resolution Order (MRO) with multiple inheritance. Create classes ClassA, ClassB, and ClassC and display the method calling order.
7. Write a class MathOperations that has both static methods and instance methods. Show the difference by calling methods using the instance and class name.
8. Create an abstract class Shape with an abstract method area(). Derive Circle and Rectangle classes and implement the area() method.
9. Define a ComplexNumber class with attributes real and imaginary. Overload the + operator to add two complex numbers and the str() method for string representation.
10. Create a class Reverse that implements the iterator protocol to iterate over a list in reverse order using the \_\_iter\_\_() and \_\_next\_\_() methods.
11. Demonstrate composition by creating a Car class with an object of the Engine class as an attribute.
12. Write a class Person with a class method from\_string(cls, string) to create an instance from a formatted string. Also, write a method decorator that logs the method name and arguments.
13. Create a LoggerMixin class that logs method calls. Then, create Database and User classes that inherit from LoggerMixin and demonstrate logging.
14. Create a custom exception class InvalidAgeError. Write a function that raises this exception if the age is less than 18 and handle it.
15. Create a Vehicle class with a method info(). Derive a Car class, override the info() method, and use super() to call the base class method.