**1. Polymorphism Questions**

1. Write a Calculator class with overloaded add() methods for different parameter types.
2. Create a Shape class with area() method, then override it in Circle and Rectangle classes.
3. Create an interface Payment with pay() method, implement it in CreditCardPayment and UPIPayment, and call using the interface reference.
4. Demonstrate method overloading for multiply() with different parameter types in a MathOperations class.
5. Create a parent Animal class with eat() method and override it in Dog and Cat classes.
6. Create a Printer class and override a print() method in InkjetPrinter and LaserPrinter.
7. Write a program where a Bird class has a sound() method, and Parrot and Crow override it.
8. Show method overriding by creating a Vehicle class with start() and extending it in Car and Bike.
9. Use a single Shape reference to hold different shapes and call draw() on each (runtime polymorphism).
10. Create a MediaPlayer interface with play() method, implement it in AudioPlayer and VideoPlayer.

**2. Abstraction Questions**

1. Create an abstract class Vehicle with an abstract method start() and a concrete method fuelType().
2. Create an abstract class BankAccount with abstract methods for transactions and implement in SavingsAccount and CurrentAccount.
3. Create an abstract class Shape with abstract methods area() and perimeter() and implement in Square and Circle.
4. Create an interface Playable with methods play() and pause() and implement in MusicPlayer and VideoPlayer.
5. Create an abstract class Report with a method generateReport() that calls steps implemented by child classes.
6. Create an abstract class Appliance with turnOn() as abstract and powerSource() as concrete.
7. Make an interface Transport with method move() and implement in Car and Bicycle.
8. Create an abstract class Employee with calculateSalary() method implemented differently in FullTimeEmployee and PartTimeEmployee.
9. Make an interface Game with start() and end() methods, implemented in Cricket and Football.
10. Create an abstract class Phone with methods call() (abstract) and message() (concrete).

**3. Encapsulation Questions**

1. Create a Student class with private fields and provide getters and setters to access them.
2. Create a BankAccount class with private balance and allow updates only via methods with validation.
3. Create an immutable Person class with all fields private and final, initialized via constructor.
4. Create an Employee class with private fields, a constructor, and methods to display details.
5. Create a ShoppingCart class with a private list of items and methods to manage and calculate total.
6. Create a Book class with private fields title and author and provide getters and setters.
7. Make a Mobile class with a private price field, validate it in the setter so it cannot be negative.
8. Create a Laptop class with private fields and initialize them through a constructor, providing only getters.
9. Make a Car class where speed is private and can only be changed using accelerate() and brake() methods.
10. Create a Person class where age cannot be changed once set (read-only property).

**4. Functions Questions**

1. Write a function to calculate the factorial of a given number.
2. Create a function that takes two numbers as arguments and returns their sum.
3. Write a function to check whether a number is prime or not.
4. Create a function that accepts a list/array and returns the largest number.
5. Write a function to reverse a string without using built-in reverse functions.
6. Create a function that accepts a string and counts the number of vowels.
7. Write a function that converts Celsius to Fahrenheit.
8. Create a function with **default parameters** and call it with and without arguments.
9. Write a function that calculates the sum of all even numbers in a given range.
10. Create a recursive function to find the nth Fibonacci number.

**5. Class & Object Questions**

1. Create a Car class with properties brand and model and a method to display them.
2. Write a program to create a Student class with properties name and marks and display student details.
3. Create a Rectangle class with methods to calculate area and perimeter.
4. Write a BankAccount class with methods to deposit and withdraw money.
5. Create a Book class with attributes for title, author, and price, and a method to display details.
6. Create a Circle class with a method to calculate circumference and area.
7. Write a program to create a Person class with a constructor to initialize name and age.
8. Create a Mobile class with attributes brand and price and a method to update the price.
9. Write a program to create a Laptop class and display specifications using a method.
10. Create an Employee class with a method to calculate yearly salary from monthly salary.

**6. Inheritance Questions.**

**Single Inheritance**

1. **Vehicle Example:** Create a Vehicle class with attributes like brand and model. Create a Car class that inherits from Vehicle and adds an attribute number\_of\_doors. Demonstrate creating and displaying details of a Car.
2. **Shape Example:** Create a Shape class with a method calculate\_area(). Create a Circle class that inherits from Shape and implements the calculate\_area() method. Test it by creating a Circle object and calculating its area.
3. **Animal Example:** Create an Animal class with attributes name and species, and a method speak(). Create a Dog class that inherits Animal and overrides the speak() method to return “Woof!”.

**Multiple Inheritance**

1. **Person and Worker Example:** Create a Person class with attributes like name and age. Create a Worker class with attributes like company and position. Create an Employee class that inherits from both Person and Worker and adds an employee\_id attribute.
2. **Device Example:** Create a Keyboard class with a method type(). Create a Screen class with a method display(). Create a Laptop class that inherits from both and implements methods to type and display content.
3. **Sport and Music Example:** Create a Sport class with a method play\_sport(). Create a Music class with a method play\_music(). Create a TalentedPerson class that inherits from both and demonstrates the ability to play a sport and music.

**Multilevel Inheritance**

1. **Library Example:** Create a LibraryItem base class with attributes title and author. Create a Book class that inherits LibraryItem and adds attributes like publisher. Create an EBook class that inherits Book and adds an attribute file\_size.
2. **Family Tree Example:** Create a Grandparent class with an attribute family\_name. Create a Parent class that inherits Grandparent and adds an attribute occupation. Create a Child class that inherits Parent and adds an attribute school.
3. **Education Example:** Create a Person class with basic details like name and age. Create a Student class that inherits from Person and adds attributes like grade. Create a GraduateStudent class that inherits from Student and adds an attribute thesis\_title.

**Hierarchical Inheritance**

1. **Bank Account Example:** Create a BankAccount base class with methods like deposit() and withdraw(). Create a SavingsAccount class that inherits from BankAccount and adds an attribute interest\_rate. Create a CheckingAccount class that inherits from BankAccount and adds an attribute overdraft\_limit.
2. **Animal Example:** Create an Animal base class with methods eat() and sleep(). Create a Bird class that inherits Animal and adds a fly() method. Create a Fish class that inherits Animal and adds a swim() method.
3. **Shape Example:** Create a Shape base class with a method calculate\_perimeter(). Create a Rectangle class that inherits Shape and implements the perimeter calculation. Create a Triangle class that inherits Shape and implements the perimeter calculation.

**7 .Exception handling**

1. Write a Python program to divide two numbers and handle ZeroDivisionError.
2. Write a program that asks for an integer input and prints its square. Handle ValueError if the input is not an integer.
3. Create a program to read a file and handle FileNotFoundError if the file doesn’t exist.
4. Write a program that uses try-except-finally to demonstrate execution of finally block.
5. Write a program that converts a string to an integer. Handle exceptions if conversion fails.
6. Write a program that handles both ZeroDivisionError and ValueError separately.
7. Write a program that tries to open a file, read its contents, and handles any exception using a single except block.
8. Write a program to read a list element at a user-given index. Handle IndexError and ValueError.
9. Write a program to demonstrate the use of try-except-else-finally.
10. Write a program that reads two numbers from the user and performs division. If denominator is zero, handle the exception and re-raise it.

**🔹 Custom Exception Practice**

1. Create a custom exception InvalidAgeError. Raise it if the entered age is below 18.
2. Create a custom exception InsufficientBalanceError. Raise it if withdrawal amount is greater than balance.
3. Create a custom exception InvalidEmailError. Raise it if email does not contain "@" or ".com".
4. Create a program that defines a custom exception PasswordTooWeakError. Raise it if password length is less than 6.
5. Create a program that raises a custom exception NegativeNumberError if the user enters a negative number.
6. Write a program that asks for 3 attempts to enter the correct PIN. If all attempts fail, raise a custom AccountLockedError.
7. Write a program that loads a JSON file. Handle FileNotFoundError and json.JSONDecodeError.
8. Write a program that demonstrates **exception chaining** using raise ... from ....
9. Write a program that handles multiple exceptions (ZeroDivisionError, KeyError, TypeError) using a single except block.
10. Write a program to set a global exception handler using sys.excepthook to catch unhandled exceptions.

**OOPs Combine 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