PYTHON CODING TEST — CLASSES, INHERITANCE & DATA STRUCTURES (40 QUESTIONS)

Each question expects a **code snippet or full program** as an answer.

Section 1: Basic Classes (Q1–Q10)

1. Create a class `Car` with attributes `brand` and `model`, and print them.
2. Write a class `Student` with attributes `name` and `age`. Create two objects and display their attributes.
3. Define a class `Rectangle` that takes `length` and `width`, and has a method `area()` that returns the area.
4. Create a class `Circle` with a method to compute its **circumference** and **area**.
5. Write a class `BankAccount` with `deposit()` and `withdraw()` methods. Ensure balance never goes below 0.
6. Create a `Book` class that stores title, author, and year. Add a `__str__()` method for clean printing.
7. Write a class `Temperature` that can convert Celsius to Fahrenheit and vice versa.
8. Define a class `Counter` with an internal variable that increases every time `increment()` is called.
9. Write a class `Calculator` that can add, subtract, multiply, and divide two numbers.
10. Create a `Person` class that greets another person by name using a method `greet()`.

Section 2: Inheritance (Q11–Q20)

11. Create a base class `Animal` with a method `sound()`, and a subclass `Dog` that prints “Bark!”.

12. Extend the above example with another subclass `Cat` that prints “Meow!”.

13. Write a base class `Shape` with an `area()` method. Create subclasses `Square` and `Circle` that override it.

14. Create a class `Vehicle` with attributes `brand` and `wheels`. Subclass `Bike` and `Car` with specific defaults.

15. Implement **multi-level inheritance**: `Person` → `Employee` → `Manager`. Each should print a message from its constructor.

16. Demonstrate **multiple inheritance** using classes `Flyer` and `Swimmer`, then subclass `Duck` that inherits both.

17. Create a base class `Device` and two subclasses `Phone` and `Laptop`. Add a method `specs()` in each.

18. Write a class `Teacher` that inherits from `Person` and adds a method `teach(subject)`.

19. Use `super()` in a subclass constructor to initialize the parent attributes.

20. Write a program showing **method overriding** — a subclass changes a method’s behavior from the parent class.

Section 3: Data Structures Inside Classes (Q21–Q30)

21. Write a class `ShoppingCart` that stores items in a **list**, and has methods `add_item()` and `remove_item()`.

22. Create a class `PhoneBook` that stores contacts in a **dictionary**. Include methods to add, remove, and search by name.

23. Write a class `Library` that uses a **set** to store book titles (to avoid duplicates).

24. Implement a class `Grades` that keeps student scores in a dictionary and calculates the average.

25. Create a class `ToDoList` that stores tasks in a list and can display completed and pending tasks separately.

26. Write a class `Queue` using a Python **list** to simulate enqueue and dequeue operations.

27. Implement a `Stack` class using list methods `append()` and `pop()`.

28. Create a class `Bank` that holds multiple `BankAccount` objects in a dictionary keyed by account numbers.

29. Write a class `Inventory` that stores product names and quantities in a dictionary. Add methods to add, remove, and display items.

30. Write a class `UniqueNumbers` that maintains a set of unique numbers and provides `add_number()` and `display()` methods.

Section 4: Intermediate Class Applications (Q31–Q40)

31. Create a class `Student` that stores a list of subjects and marks. Add a method `highest_score()`.

32. Write a class `Playlist` that stores songs in a list. Add methods to `add_song()`, `remove_song()`, and `shuffle()`.

33. Define a class `BankSystem` that manages multiple customers (use a list of objects). Add a method to display all balances.

34. Implement a class `Matrix` that supports adding two 2x2 matrices using a method `add_matrix()`.

35. Write a class `ContactManager` that prevents duplicate names using a set and supports listing all contacts alphabetically.

36. Create a class `Animal` with subclasses `Bird` and `Fish`, each overriding `move()` method to print their specific movements.

37. Write a class `Employee` that stores details in a dictionary and a subclass `Developer` that adds programming languages.

38. Create a class `HistoryTracker` that uses a list to track every action performed by a user.

39. Implement a class `VotingSystem` that keeps votes in a dictionary and determines the winner.

40. Create a class `DataAnalyzer` that accepts a list of numbers and provides methods to get min, max, mean, and median.
