🔷 What is a Class?

A **class** in Python is a **blueprint** for creating objects.
It defines how an object should look (attributes/variables) and behave (methods/functions).

Think of it like:

- **Class** → blueprint of a car.
- **Object** → the actual car made from that blueprint.

🔷 Defining a Class

```
class Car:
    # attribute (variable)
    wheels = 4

    # constructor (called when creating an object)
    def __init__(self, brand, color):
        self.brand = brand
        self.color = color

    # method (function inside a class)
    def drive(self):
        print(f"{self.color} {self.brand} is driving 🚗")
```

🔷 Creating Objects

```
my_car = Car("Toyota", "Red")
your_car = Car("BMW", "Black")

print(my_car.brand)   # Toyota
```

```
print(your_car.color) # Black


my_car.drive()  # Red Toyota is driving 🚗
```

🔷 Key Components of a Class

1. **Attributes** → variables inside a class (data about the object).
    - Class attributes: shared by all objects.
    - Instance attributes: unique to each object.
2. **Methods** → functions inside a class.
    - Regular methods: take self (the instance itself).
    - Class methods: take cls (the class itself).
    - Static methods: no self or cls.

🔷 Example of Class, Static, and Instance Methods

```
class Student:
    school_name = "Global High School"  # class attribute

    def __init__(self, name, age):
        self.name = name   # instance attribute
        self.age = age

    def introduce(self):  # instance method
        print(f"My name is {self.name}, I am {self.age} years old.")

    @classmethod
    def school_info(cls):  # class method
```

```
        print(f"School: {cls.school_name}")


    @staticmethod
    def greet():  # static method
        print("Welcome to the school!")
s1 = Student("John", 16)
s1.introduce()       # My name is John, I am 16 years old.
Student.school_info() # School: Global High School
Student.greet()      # Welcome to the school!
```

🔷 Inheritance

One class can **inherit** from another.

```
class Animal:
    def speak(self):
        print("Animal speaks")


class Dog(Animal):
    def speak(self):
        print("Woof! 🐶")


dog = Dog()
dog.speak()   # Woof! 🐶
```

🔷 Encapsulation & Private Variables

By convention, variables with _ or __ are considered private.

```
class BankAccount:
```

```
    def __init__(self, balance):
        self.__balance = balance  # private attribute

    def deposit(self, amount):
        self.__balance += amount

    def get_balance(self):
        return self.__balance

acct = BankAccount(1000)
acct.deposit(500)
print(acct.get_balance())  # 1500
```

✅ **Summary**

- **Class** = Blueprint.
- **Object** = Instance of class.
- Attributes = Data.
- Methods = Behaviors.
- Supports **inheritance, encapsulation, polymorphism**.

📝 **20 Quiz Questions on Python Classes**

🔷 **Multiple Choice & Short Answer**

1. What keyword is used to define a class in Python?
2. What is the first argument of every instance method in a class?
3. What is the difference between a class attribute and an instance attribute?
4. Which method in a class is automatically called when an object is created?
5. True or False: A class can have multiple constructors in Python.

6. Write a simple class Book with attributes title and author.
7. What does self represent in a class?
8. Which decorator is used to define a class method?
9. Which decorator is used to define a static method?
10. What is the difference between @classmethod and @staticmethod?
11. Define a class Person with a method say_hello() that prints “Hello, I am a person.”
12. Create an object of the Person class above and call the method.
13. What is inheritance in classes? Give an example.
14. What will this code output?

```
class A:
    x = 10
obj1 = A()
obj2 = A()
obj1.x = 20
print(obj1.x, obj2.x)
```

15. What is encapsulation in OOP?
16. How do you define a private variable in a class?
17. What is the difference between __init__ and a normal method?
18. Can a class inherit from multiple classes in Python? (Yes/No)
19. What is method overriding? Give an example.
20. Write a class Calculator with a method add(a, b) that returns the sum.

📝 5 Theory Questions

1. Explain the concept of **Object-Oriented Programming (OOP)** and how classes fit into it.
2. Discuss the role of the __init__ method in Python classes. Why is it important?

3. Differentiate between **class methods**, **instance methods**, and **static methods** with examples.
4. Explain the concepts of **inheritance, polymorphism, and encapsulation** in Python OOP.
5. Why do we use self in Python classes, and how does it differ from cls?

🚀 3 Capstone Projects on Classes

📌 Project 1: Library Management System

- Create a Book class with attributes: title, author, available_copies.
- Create a Library class that stores books in a list.
- Add methods to:
    - Add a book
    - Borrow a book (reduce available copies)
    - Return a book (increase available copies)
- Test the system with multiple books and borrow/return operations.

📌 Project 2: Banking System

- Create a BankAccount class with attributes: account_number, owner, balance.
- Add methods to:
    - deposit(amount)
    - withdraw(amount) (check if sufficient balance)
    - get_balance()
- Create multiple accounts and simulate transactions.

📌 Project 3: School Management System

- Create a Student class with attributes: name, age, grades.
- Add methods to:

    - Add a grade
    - Calculate average grade
- Create a Teacher class with attributes: name, subject.
- Create a School class to manage students and teachers.
- Simulate enrolling students, assigning teachers, and displaying average student performance.