Python OOPs Concepts:

Object-Oriented Programming (OOP) is a programming approach that models software using real-world objects. It uses classes as blueprints and objects as real instances. The four core principles are:

Encapsulation: Keeping data and methods together for protection.

Inheritance: Reusing code by allowing one class to inherit features from another.

Polymorphism: Using one interface to perform different tasks.

Abstraction: Hiding complex details and showing only the necessary parts.

In short: OOP helps you write clean, reusable, and understandable code — just like organizing tools in labeled boxes.

1. What Is a Class?

A **class** is a **blueprint or template** used to create **objects**. It defines the **attributes (data)** and **methods (functions)** that the objects will have.

Example: A Car class defines what every car should have (like color, model) and what it can do (like drive()).

✅ 2. Class Kaise Banate Hain (Python Example)?

python Copy code

```python
class Car:  # Class declaration
    def __init__(self, color, model):  # Constructor method
        self.color = color      # attribute
        self.model = model      # attribute

    def drive(self):            # method (function)
        print(f"The {self.color} {self.model} is driving.")
```

Explanation:

- class Car: → Declares a class named Car.
- __init__() → Constructor: It runs automatically when the object is created.
- self.color and self.model → Attributes of the object.
- drive() → A method (behavior or action the object can perform).

✅ 3. Object Kaise Banate Hain?

```python
car1 = Car("Red", "Toyota")
car2 = Car("Blue", "Honda")
```

Yahan `car1` aur `car2` do alag **objects** hain **Car** class ke.

✅ 4. Object Kaise Use Karte Hain?

```python
car1.drive()  # Output: The Red Toyota is driving.
car2.drive()  # Output: The Blue Honda is driving.
```

Humne `car1` aur `car2` ke methods call kiye.

.

✅ **5. What Else Can a Class Contain?**

- **Multiple methods** (like brake(), honk())
- **Default attribute values**
- **Inheritance** (one class can extend another)
- **Polymorphism** (same method name behaves differently)
- **Encapsulation** (hiding internal details)

IN SHORT:

A class defines what an object is and what it can do. You use classes to build structured, clean, and reusable code in OOP.

2. What Is an Object?

An **object** is a **real-world instance** created from a **class blueprint**.
It represents **actual data and behavior**.

Class = Plan
Object = Actual thing built using the plan

Example:

```python
class Car:
    def __init__(self, color, model):
        self.color = color
        self.model = model

    def drive(self):
        print(f"{self.color} {self.model} is driving.")

car1 = Car("Red", "Toyota")  # Object
```

Yahan `car1` aik **object** hai jo `Car` class se banaya gaya hai.

✅ 2. What Does an Object Contain?

- **Attributes (Variables/Properties)** – Hold data
  Example: color, model
- **Methods (Functions)** – Define behavior
  Example: drive(), stop()

✅ 3. How to Create an Object?

```python
car1 = Car("Red", "Toyota")
car2 = Car("Blue", "Honda")
```

Now car1 is an object of the Car class.

✅ 4. How to Use an Object?

✅ 4. Object Ko Kaise Use Karte Hain?

```python
car1.drive()  # Output: Red Toyota is driving.
car2.drive()  # Output: Blue Honda is driving.
```

You can use the object to access its data and behavior.

3.SELF in OOP – A to Z (in ENGLISH)

✅ 1. What Is self?

self refers to the **current object (instance)** of the class inside a method.

It allows you to **access attributes and methods** that belong to that specific object.

✅ 2. Why Do We Use self?

We use self to:

- assign or access object-specific variables
- differentiate between instance and local variables

✅ 3. Example:

```python
class Animal:
    def __init__(self, name):
        self.name = name

    def speak(self):
        print(f"{self.name} makes a sound.")
```

```python
a1 = Animal("Dog")
a2 = Animal("Cat")

a1.speak()  # Dog makes a sound.
a2.speak()  # Cat makes a sound.
```

- Here, self.name points to each object's individual name.
- self is automatically passed when a1.speak() is called.

✅ 4. What If We Don't Use self?

If you remove self, Python won't know which object's data you're referring to. It'll give an error.

✅ 5. Key Notes:

Feature	Details
Keyword	self is not a Python keyword, just a convention
Meaning	Refers to the current object
Scope	Only inside class methods
Automatically Sent	Yes, Python sends self when method is called

---

✅ In One Line:

self is used inside a class to refer to the current object and access its attributes and methods.

4.What is __init__?

__init__ is a **special method** (called a **constructor**) in Python classes.
It is **automatically called when an object is created**.

✅ 2. What Does It Do?

- Initializes object properties
- Sets default or passed-in values

✅ 3. Example:

python Copy code

```python
class Car:
    def __init__(self, brand, color):
        self.brand = brand
        self.color = color

    def show(self):
        print(f"{self.color} {self.brand}")
```

python Copy code

```python
c1 = Car("Toyota", "Red")
c1.show()  # Output: Red Toyota
```

- __init__ sets the brand and color for the c1 object.

✅ 4. Without __init__?

You would have to set all values **manually after creating the object**, which is less clean and more error-prone.

✅ 5. Important Points:

Feature	Detail
Name	Always __init__ with double underscores

Feature	Detail
Called When?	When an object is created
Purpose	To initialize the object
First Argument	Must be self
Optional Args	You can pass any number of custom parameters

✅ **In One Line:**

__init__ is a special method in Python that automatically runs when an object is created, and it's used to initialize object properties.

5. What is a Method?

A **method** is a **function defined inside a class**.
It defines what an object can do (its behavior).

✅ **2. How to Define a Method:**

python Copy code

```python
class Lamp:
    def turn_on(self):
        print("Lamp is on")
```

Here, `turn_on()` is a method inside the `Lamp` class.

Here, turn_on() is a method inside the Lamp class.

✅ 3. How to Call a Method:

```python
l1 = Lamp()
l1.turn_on()
```

- `l1` is the object
- `turn_on()` is called on that object

✅ 4. Types of Methods:

Type	Uses	Decorator
Instance Method	Works on object data	None
Class Method	Works on class data	@classmethod
Static Method	Utility method (no self)	@staticmethod

✅ 5. With Parameters Example:

```python
class Math:
    def multiply(self, a, b):
        return a * b
```

Call:

```python
m = Math()
print(m.multiply(3, 4))  # Output: 12
```

In One Line:

A method is a function inside a class that defines what an object can do.