Chapter 9 – OOPs (Object-Oriented Programming) in Python

Object-Oriented Programming (OOP) helps us structure programs using **real-world concepts** like objects, classes, attributes, and behaviors. Instead of writing everything in one place, we organize code into **objects** - just like real-life entities.

Example:
A **Student**, **Car**, **Bank Account**, **Mobile Phone** - each can be represented as an object in Python.

1️⃣ What is OOP?

OOP is a programming style where we use:

- **Class** → Blueprint
- **Object** → Real instance
- **Attributes** → Data
- **Methods** → Behaviors (functions inside classes)

Using OOP makes code:

- Reusable
- Organized
- Easy to maintain
- Similar to real-world objects

2️⃣ Class and Object

Class: A blueprint/template

Object: A real entity created from the class

Example:

```
class Student:
    name = "Saumya Singh"   # attribute

# creating object
s1 = Student()
print(s1.name)
```

Output:

```
Saumya Singh
```

Practice Questions – Class & Object

1. Create a class `Car` with attribute `brand = "Scorpio"`.
2. Create a class `Laptop` with attributes: brand, RAM, price. Create 2 objects with different values.

3️⃣ Instance Attributes vs Class Attributes

Class Attribute

Shared by all objects

```
class Student:
```

```
    college = "XYZ Institute"
```

Instance Attribute

Unique for each object

```
class Student:
    college = "XYZ Institute"
    def __init__(self, name):
        self.name = name

s1 = Student("Saumya")
s2 = Student("Aman")
```

Practice Questions – Attributes

1. Create a class `FoodItem` with class attribute `category = "Snacks"` and instance attribute name (“Samosa”, “GulabJamun”).

4️⃣ The `__init__()` Constructor

The `__init__()` function runs automatically whenever an object is created.
It is used to **initialize attributes**.

Example

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

s1 = Student("Saumya Singh")
print(s1.name)
```

5️⃣ Methods (Functions inside Class)

Methods define what an object **can do**.

Example

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

    def hello(self):
        print("Hello", self.name)

s1 = Student("Saumya")
s1.hello()
```

Practice Questions – Methods

1. Create class `Student` that takes 3 marks and has a method `average()`.

6️⃣ Static Methods

Static methods do **not** use `self`.
They are used for utility-level functions.

Example

```
class Student:
    @staticmethod
    def school():
        print("ABC Public School")  # generic name
```

Practice Questions – Static Methods

1. Create static method to validate if a number is even.

7️⃣ OOP Concepts – Abstraction & Encapsulation

Abstraction

Showing **only essential details**, hiding internal complexity.

Example:
You use **Instagram** without knowing its backend code.

In Python:

```
class Payment:
    def pay(self):
        print("Payment Successful")
```

Encapsulation

Wrapping **data + methods** inside a single unit (class).
Data is protected using private variables.

Example:

```
class Account:
    def __init__(self, bal):
        self.__balance = bal   # private

    def show_balance(self):
        print("Balance:", self.__balance)
```

Inheritance (Very Simple Explanation + Full Code)

Meaning:
When one class (child) gets the properties and methods of another class (parent).
It avoids repeating code.

Think:
Child class uses everything the parent class already has.

Full Code:

```
# Parent class

class Vehicle:

    def start(self):

        print("Vehicle is starting")

# Child class 1

class Car(Vehicle):

    def drive(self):

        print("Car is now moving")

# Child class 2

class Bike(Vehicle):

    def ride(self):

        print("Bike is now riding")

# Child class 3

class Truck(Vehicle):

    def load(self):

        print("Truck is loading goods")
```

```
# Using the classes

c = Car()

c.start()   # from parent

c.drive()   # child specific


b = Bike()

b.start()   # from parent

b.ride()    # child specific


t = Truck()

t.start()   # from parent

t.load()    # child specific
```

Polymorphism (Very Simple Explanation + Full Code)

Meaning:
Same function name, but different behaviour in different classes.

One function name → **<u>many</u>** different behaviours depending on the object calling it.

Think:
Different objects respond in their own unique way.

If I say **"sound"**,

- A dog will bark

- A cat will meow
- A cow will moo

Same word → different actions.
That is *polymorphism*.

Full Code:

Example 1: Animal Sounds

```
class Dog:

    def sound(self):

        print("Dog says: Bark")


class Cat:

    def sound(self):

        print("Cat says: Meow")


class Cow:

    def sound(self):

        print("Cow says: Moo")


# Polymorphism in action

animals = [Dog(), Cat(), Cow()]
```

```
for a in animals:

    a.sound()   # same function name, different output
```

Example 2: Shapes (Same method, different behavior)

One function: area()
Different shapes → different formula

```
class Circle:

    def area(self):

        print("Area = πr²")


class Square:

    def area(self):

        print("Area = side × side")


class Rectangle:

    def area(self):

        print("Area = length × width")


shapes = [Circle(), Square(), Rectangle()]
```

```
for s in shapes:
    s.area()    # same function, different calculations
```

SUPER SIMPLE WAY TO REMEMBER POLYMORPHISM

Polymorphism = **"Same name, different action."**

- same method name
- different output
- depends on the object

8️⃣ Practice – Build Account Class

Task:
Create an **Account** class with:

- Attributes → balance, account number
- Methods → debit, credit, print balance

Example (Enhanced)

```
class Account:
    def __init__(self, acc_no, balance):
        self.acc_no = acc_no
        self.balance = balance

    def credit(self, amount):
        self.balance += amount
        print("Amount credited:", amount)

    def debit(self, amount):
        if amount <= self.balance:
            self.balance -= amount
```

```
            print("Amount debited:", amount)
        else:
            print("Insufficient funds")

    def show_balance(self):
        print("Account:", self.acc_no, "| Balance:", self.balance)

# Example
a1 = Account("SAU123", 5000)
a1.credit(2000)
a1.debit(1500)
a1.show_balance()
```

📘 Chapter 9 – Practice Set

1. Student Class

Create class `Student` with name, class, marks. Add method `get_percentage()`.

2. Creator Portfolio

Create class `Creator` with attributes (name, username). Add method `bio()`.

3. Food Order

Create class `FoodOrder` with item name, quantity, price. Add method to calculate bill.

4. Employee Salary Manager

Add method to increase salary by a percentage.

5. Bank Account (Final Project)

Add debit/credit logs and private balance.

Summary Table

Concept	Meaning
Class	Blueprint/template
Object	Real instance
Attributes	Data variables
Methods	Functions inside class
Constructor	`__init__()` initializer
Static Method	Method without `self`
Abstraction	Hide internal details
Encapsulation	Protect data
Polymorphism	Same method name, different behaviour depending on object
Inheritance	Child class reuses parent class features (no code repetition)