**Object-Oriented Programming (OOP) in Python**

**1. What is OOP?**

* **OOP** = Object-Oriented Programming.
* It is a way of writing programs using **classes** and **objects**.
* Instead of just functions and variables, you structure code into objects that **bundle data (attributes)** and **behavior (methods)** together.

**Why use OOP?**

* Organizes code → cleaner, reusable, and modular.
* Easier to maintain → especially in large projects.
* Real-world modeling → cars, students, employees, etc.
* Reusability via inheritance and polymorphism.

**2. Basic Concepts of OOP**

The **4 main pillars** of OOP in Python:

1. **Encapsulation** → Hiding the details and controlling access.
2. **Abstraction** → Showing only essential features, hiding background details.
3. **Inheritance** → Reusing code by creating new classes from existing ones.
4. **Polymorphism** → Same function/method behaving differently based on object.

**3. Classes and Objects**

**Theory:**

* A **class** is like a **blueprint** (e.g., Car design).
* An **object** is a **real-world instance** (e.g., Honda Civic, Toyota Corolla).

**Example:**

# Define a class

class Car:

# Constructor (special method to initialize object data)

def \_\_init\_\_(self, brand, model):

self.brand = brand # attribute (data)

self.model = model # attribute (data)

# Method (behavior)

def display\_info(self):

print(f"Car: {self.brand} {self.model}")

# Create objects (instances)

car1 = Car("Toyota", "Corolla")

car2 = Car("Honda", "Civic")

# Call methods

car1.display\_info() # Output: Car: Toyota Corolla

car2.display\_info() # Output: Car: Honda Civic

**4. Encapsulation**

**Theory:**

* Binding variables (data) and methods (functions) together.
* You can make attributes **public, protected, private** using naming conventions:
  + self.name → public
  + self.\_name → protected (convention only)
  + self.\_\_name → private (name mangling)

**Example:**

class BankAccount:

def \_\_init\_\_(self, account\_number, balance):

self.\_\_account\_number = account\_number # private

self.\_\_balance = balance # private

# Getter method

def get\_balance(self):

return self.\_\_balance

# Setter method

def deposit(self, amount):

if amount > 0:

self.\_\_balance += amount

else:

print("Deposit must be positive!")

# Create object

account = BankAccount("12345", 1000)

print(account.get\_balance()) # Output: 1000

account.deposit(500) # Deposit 500

print(account.get\_balance()) # Output: 1500

**5. Abstraction**

**Theory:**

* **Hiding unnecessary details** and exposing only essential functionality.
* Achieved in Python using **abstract classes** (abc module).

**Example:**

from abc import ABC, abstractmethod

class Vehicle(ABC):

@abstractmethod

def start\_engine(self):

pass

class Car(Vehicle):

def start\_engine(self):

print("Car engine started 🚗")

class Bike(Vehicle):

def start\_engine(self):

print("Bike engine started 🏍️")

# car = Vehicle() # ❌ Error (abstract class cannot be instantiated)

car = Car()

bike = Bike()

car.start\_engine() # Output: Car engine started 🚗

bike.start\_engine() # Output: Bike engine started 🏍️

**6. Inheritance**

**Theory:**

* One class can **inherit** properties and methods from another.
* Promotes **code reusability**.
* Types of inheritance in Python:
  + Single
  + Multiple
  + Multilevel
  + Hierarchical
  + Hybrid

**Example:**

# Base class

class Animal:

def speak(self):

print("This is an animal.")

# Derived class (Single Inheritance)

class Dog(Animal):

def speak(self):

print("Woof! 🐶")

# Another Derived class

class Cat(Animal):

def speak(self):

print("Meow! 🐱")

# Create objects

dog = Dog()

cat = Cat()

dog.speak() # Output: Woof! 🐶

cat.speak() # Output: Meow! 🐱

**7. Polymorphism**

**Theory:**

* **Poly** = many, **Morphism** = forms.
* Same function/method behaves differently depending on the object.

**Example:**

class Bird:

def fly(self):

print("Most birds can fly 🕊️")

class Penguin(Bird):

def fly(self):

print("Penguins cannot fly ❄️")

# Polymorphism

def bird\_fly(bird):

bird.fly()

sparrow = Bird()

penguin = Penguin()

bird\_fly(sparrow) # Output: Most birds can fly 🕊️

bird\_fly(penguin) # Output: Penguins cannot fly ❄️

**8. Class Methods, Static Methods, Instance Methods**

**Example:**

class Student:

school\_name = "ABC School" # Class variable

def \_\_init\_\_(self, name):

self.name = name # Instance variable

# Instance method → works with object

def show(self):

print(f"Student name: {self.name}")

# Class method → works with class, not object

@classmethod

def change\_school(cls, new\_name):

cls.school\_name = new\_name

# Static method → utility method, doesn’t depend on class/object

@staticmethod

def welcome():

print("Welcome to the school!")

# Usage

s1 = Student("Ali")

s1.show() # Instance method

Student.change\_school("XYZ School") # Class method

Student.welcome() # Static method

**9. Magic/Dunder Methods**

**Example:**

class Book:

def \_\_init\_\_(self, title, pages):

self.title = title

self.pages = pages

# \_\_str\_\_ → string representation

def \_\_str\_\_(self):

return f"{self.title}, {self.pages} pages"

# \_\_len\_\_ → length function

def \_\_len\_\_(self):

return self.pages

book1 = Book("Python OOP", 350)

print(book1) # Output: Python OOP, 350 pages

print(len(book1)) # Output: 350

**10. Difference: Encapsulation vs Abstraction**

| **Feature** | **Encapsulation** | **Abstraction** |
| --- | --- | --- |
| Definition | Bundling data & methods into a single unit (class). | Hiding implementation details and showing only functionality. |
| Focus | Restricting access (using private/protected). | Hiding complexity. |
| Implementation | Using private variables, getters, setters. | Using abstract classes & interfaces. |
| Example | Bank account balance hidden. | Vehicle class with abstract start\_engine(). |

**11. Interview Questions on OOP in Python**

1. What are the four pillars of OOP?
2. Difference between @staticmethod and @classmethod?
3. Can Python support multiple inheritance? (Yes)
4. What is method overloading in Python? (Not supported directly, can be achieved with default args)
5. What is method overriding? (Subclass changes parent class method)
6. Difference between Encapsulation & Abstraction?
7. What are dunder methods?