🐍 Python OOP (Object-Oriented Programming) Complete Notes with Examples & Practice Problems

---

📘 What is Object-Oriented Programming (OOP)?

Object-Oriented Programming (OOP) is a programming paradigm based on the concept of "objects", which can contain data (attributes) and code (methods). Python is an object-oriented language that allows you to build applications using classes and objects.

🎯 Key Concepts of OOP in Python

1. 🔷 Class

A class is a blueprint for creating objects.

```python
class Car:
    pass
```

2. 🚙 Object

An object is an instance of a class.

```python
my_car = Car()
```

3. 📦 Attributes and Methods

Attributes store data; methods define behaviors.

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

    def get_full_name(self):
        return f"{self.brand} {self.model}"

my_car = Car("Tesla", "Model S")
print(my_car.get_full_name())  # Tesla Model S
```

4. 🔁 Inheritance

Allows a class to inherit properties/methods from another class.

```python
class ElectricCar(Car):
    def __init__(self, brand, model, battery_size):
        super().__init__(brand, model)
        self.battery_size = battery_size
```

5. 🔒 Encapsulation

Restrict access to some components. Use double underscores (`__`) to make private attributes.

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

    def get_brand(self):
        return self.__brand
```

6. 🎭 Polymorphism

Same method name with different behaviors.

```python
class Car:
    def fuel_type(self):
        return "Petrol or Diesel"

class ElectricCar(Car):
    def fuel_type(self):
        return "Electric"
```

7. 📊 Class Variables

Shared by all instances of the class.

```python
class Car:
    car_count = 0

    def __init__(self):
        Car.car_count += 1
```

8. 🛠️ Static Methods

Do not access or modify class state.

```python
class Car:
    @staticmethod
    def general_info():
        return "Cars have wheels and are used for transportation."
```

9. 🔁 Property Decorators

Control attribute access with `@property`.

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

    @property
    def model(self):
        return self._model
```

10. 🧩 Multiple Inheritance

Inherit from more than one class.

```python
class Battery:
    def battery_info(self):
        return "Battery capacity: 100 kWh"

class Engine:
    def engine_info(self):
        return "Electric engine"

class ElectricCar(Battery, Engine):
    pass

my_ecar = ElectricCar()
print(my_ecar.battery_info())
print(my_ecar.engine_info())
```

✅ Practice Problems with Solutions

1. 📦 Basic Class and Object

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

```python
my_car = Car("Toyota", "Corolla")
print(my_car.brand, my_car.model)
```

2. 🔁 Class Method and `self`

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

    def get_full_name(self):
        return f"{self.brand} {self.model}"

car = Car("Ford", "Mustang")
print(car.get_full_name())
```

3. 🧬 Inheritance

```python
class ElectricCar(Car):
    def __init__(self, brand, model, battery_size):
        super().__init__(brand, model)
        self.battery_size = battery_size
```

4. 🔒 Encapsulation

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

    def get_brand(self):
        return self.__brand

car = Car("Hyundai", "i20")
print(car.get_brand())
```

5. 🎭 Polymorphism

```python
class Car:
    def fuel_type(self):
        return "Petrol"

class ElectricCar(Car):
    def fuel_type(self):
        return "Electric"

print(Car().fuel_type())
print(ElectricCar().fuel_type())
```

6. 📊 Class Variables

```python
class Car:
    car_count = 0

    def __init__(self):
        Car.car_count += 1

car1 = Car()
car2 = Car()
print(Car.car_count)  # 2
```

7. 🛠 Static Method

```python
class Car:
    @staticmethod
    def general_info():
        return "Cars have wheels and are used for transport."

print(Car.general_info())
```

8. 🔁 Property Decorator

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

    @property
    def model(self):
        return self._model

car = Car("Swift")
print(car.model)
```

9. 🔎 isinstance()

```python
my_tesla = ElectricCar("Tesla", "Model 3", 75)
print(isinstance(my_tesla, Car))         # True
print(isinstance(my_tesla, ElectricCar)) # True
```

10. 🧩 Multiple Inheritance

```
class Battery:
    def battery_info(self):
        return "Battery: 90 kWh"

class Engine:
    def engine_info(self):
        return "Dual Motor"

class ElectricCar(Battery, Engine):
    pass

ecar = ElectricCar()
print(ecar.battery_info())
print(ecar.engine_info())
```

🚀 Learn Object-Oriented Programming Through Practical Questions

Explore the core principles of OOP by solving real-world inspired coding tasks. Expand your understanding of classes, inheritance, encapsulation, and more — step by step.

✅ Solutions Below ⬇️

▶ **1. 📦 Basic Class and Object**

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

# Creating object
my_car = Car("Toyota", "Camry")
print(f"🚙 Brand: {my_car.brand}, Model: {my_car.model}")
```

▶ **2. 🔁 Instance Method and `self`**

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

    def get_full_name(self):
        return f"{self.brand} {self.model}"

my_car = Car("Tesla", "Model S")
print("🚘 Full Name:", my_car.get_full_name())
```

▶ **3. 🧬 Inheritance**

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

class ElectricCar(Car):
    def __init__(self, brand, model, battery_size):
        super().__init__(brand, model)
        self.battery_size = battery_size

my_tesla = ElectricCar("Tesla", "Model 3", "75 kWh")
print(f"⚡ {my_tesla.brand} {my_tesla.model} with {my_tesla.battery_size}
battery")
```

▶ **4. 🔒 Encapsulation**

```python
class Car:
    def __init__(self, brand, model):
        self.__brand = brand  # Private
        self.model = model

    def get_brand(self):
        return self.__brand

my_car = Car("Honda", "Civic")
print("🔐 Brand (using getter):", my_car.get_brand())
```

▶ **5. 🎭 Polymorphism**

```python
class Car:
    def fuel_type(self):
        return "⛽ Uses petrol or diesel"
```

```python
class ElectricCar(Car):
    def fuel_type(self):
        return "🔋 Uses electricity"

vehicle1 = Car()
vehicle2 = ElectricCar()
print(vehicle1.fuel_type())
print(vehicle2.fuel_type())
```

▶ **6. 📊 Class Variables**

```python
class Car:
    car_count = 0

    def __init__(self, brand, model):
        self.brand = brand
        self.model = model
        Car.car_count += 1

car1 = Car("Ford", "Mustang")
car2 = Car("BMW", "X5")
print("📈 Total Cars Created:", Car.car_count)
```

▶ **7. 🛠️ Static Method**

```python
class Car:
    @staticmethod
    def general_info():
        return "🛞 Cars have wheels and can be used for transport."

print(Car.general_info())
```

▶ **8. 🔁 Property Decorators**

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

    @property
    def model(self):
        return self._model

my_car = Car("Audi", "A4")
```

```
print("📦 Model (read-only):", my_car.model)
# my_car.model = "Q5"  # ❌ Will raise AttributeError if you uncomment this
```

▶ **9. 🔎 isinstance() and Inheritance Check**

```
class Car:
    pass

class ElectricCar(Car):
    pass

my_tesla = ElectricCar()
print("🔍 Is my_tesla an instance of Car?", isinstance(my_tesla, Car))
print("🔍 Is my_tesla an instance of ElectricCar?", isinstance(my_tesla,
ElectricCar))
```

▶ **10. 🧩 Multiple Inheritance**

```
class Battery:
    def battery_info(self):
        return "🔋 Battery size is 85 kWh"

class Engine:
    def engine_info(self):
        return "⚙️ Dual motor system"

class ElectricCar(Battery, Engine):
    def info(self):
        return f"{self.battery_info()} | {self.engine_info()}"

my_ecar = ElectricCar()
print(my_ecar.info())
```
