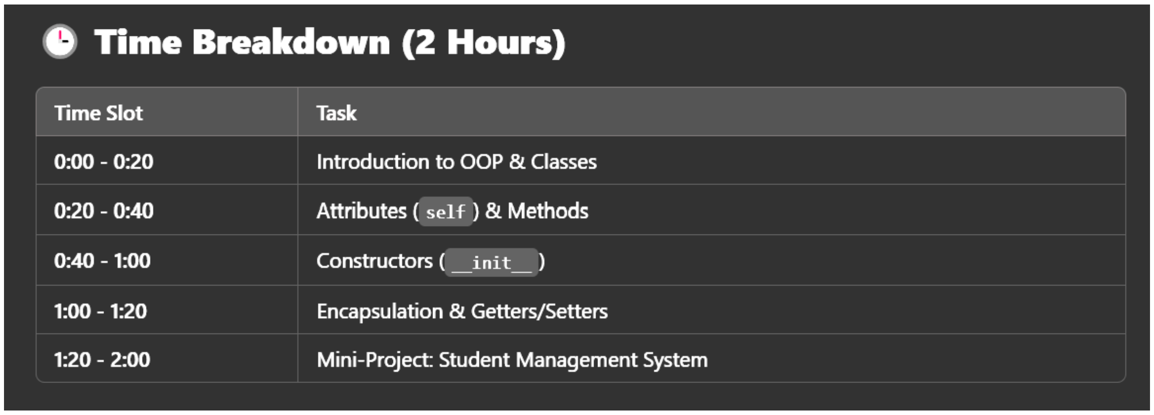
Day 11 of 6 weeks Python course:



🕒 Time Breakdown (2 Hours)

Time Slot	Task
0:00 - 0:20	Introduction to OOP & Classes
0:20 - 0:40	Attributes (`self`) & Methods
0:40 - 1:00	Constructors (`__init__`)
1:00 - 1:20	Encapsulation & Getters/Setters
1:20 - 2:00	Mini-Project: Student Management System

📌 Step-by-Step Learning Guide 1️⃣ What is Object-Oriented Programming? (20 mins) ✅ Definition: OOP is a programming paradigm based on objects that contain data (attributes) and functions (methods). ✅ Why Use OOP? • Modular Code – Divides a program into reusable components. • Encapsulation – Protects data inside a class. •Code Reusability – Reduces redundancy by using inheritance. ✅ Basic Class & Object Example

In [2]:
```python
class Student:
    def __init__(self, name, age):
        self.name = name
        self.age = age

student1 = Student("pavesh", 18)  # Creating an object
print(student1.name)  # Output: Deepak
```

```
pavesh
```

2️⃣ Instance Attributes & Methods (20 mins) ✅ Attributes (Instance Variables) •Attributes store object-specific data. •Defined inside the __init__ constructor. ✅ Example:

In [6]:
```python
class Car:
    def __init__(self, brand, model):
        self.brand = brand
        self.model = model

car1 = Car("Tesla", "Model S")
car2 = Car("Toyota", "Camry")

print(car1.brand, car1.model)  # Tesla Model S
print(car2.brand, car2.model)  # Toyota Camry
```

```
Tesla Model S
Toyota Camry
```

✅ Methods (Functions in a Class)

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

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

```
car1 = Car("Tesla", "Model S")
car1.display_info()  # Output: Car: Tesla Model S
```

3️⃣ Using Constructors (__init__) (20 mins) ✅ What is a Constructor? •__init__ is called automatically when creating an object. •Initializes object attributes. ✅ Example:

In [14]:
```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

person1 = Person("Pavesh", 18)
print(person1.name)  # Output: Deepak
```

```
Pavesh
```

4️⃣ Encapsulation: Private Variables & Getters/Setters (20 mins) ✅ Encapsulation protects data by hiding internal details. •Use private attributes (__variable) to restrict access. ✅ Example:

In [18]:
```
class BankAccount:
    def __init__(self, account_number, balance):
        self.account_number = account_number
        self.__balance = balance  # Private attribute

    def deposit(self, amount):
        self.__balance += amount
        print(f"₹{amount} deposited. New balance: ₹{self.__balance}")

    def get_balance(self):  # Getter method
        return self.__balance

account = BankAccount("12345", 1000)
account.deposit(500)
print(account.get_balance())  # Output: ₹1500
```

```
₹500 deposited. New balance: ₹1500
1500
```

🎯 Mini-Project: Student Management System 📌 Project Goal •Store student details (name, age, grades). •Allow users to add, update, and view student records. •Use OOP concepts (classes, methods, encapsulation). 💻 Code Implementation

In [22]:
```
class Student:
    def __init__(self, name, age, grade):
        self.name = name
        self.age = age
        self.__grade = grade  # Private attribute

    def update_grade(self, new_grade):
        self.__grade = new_grade
        print(f"{self.name}'s grade updated to {new_grade}.")

    def get_details(self):
        return f"Name: {self.name}, Age: {self.age}, Grade: {self.__grade}"

# Student List
students = []

# Function to add a student
def add_student():
    name = input("Enter student's name: ")
    age = int(input("Enter student's age: "))
    grade = input("Enter student's grade: ")

    student = Student(name, age, grade)
    students.append(student)
    print(f"Student {name} added successfully!\n")

# Function to view all students
def view_students():
    if students:
        print("\n📚 Student Records 📚")
        for i, student in enumerate(students, start=1):
            print(f"{i}. {student.get_details()}")
    else:
        print("No students found.\n")

# Function to update student grade
def update_student_grade():
    view_students()
    if students:
        try:
            student_no = int(input("Enter student number to update grade: ")) - 1
            if 0 <= student_no < len(students):
                new_grade = input("Enter new grade: ")
                students[student_no].update_grade(new_grade)
            else:
                print("Invalid student number!\n")
        except ValueError:
            print("Invalid input! Please enter a number.\n")

# Main Menu
while True:
    print("\n🎓 Student Management System 🎓")
    print("1. Add Student")
    print("2. View Students")
    print("3. Update Student Grade")
    print("4. Exit")

    choice = input("Enter your choice (1-4): ")

    if choice == "1":
        add_student()
    elif choice == "2":
        view_students()
    elif choice == "3":
        update_student_grade()
    elif choice == "4":
        print("Exiting Student Management System. Goodbye!")
        break
    else:
        print("Invalid choice! Please enter 1-4.\n")
```

```
🎓 Student Management System 🎓
1. Add Student
2. View Students
3. Update Student Grade
4. Exit
Student Pavesh added successfully!


🎓 Student Management System 🎓
1. Add Student
2. View Students
3. Update Student Grade
4. Exit
📚 Student Records 📚
1. Name: Pavesh, Age: 17, Grade: 9.52

🎓 Student Management System 🎓
1. Add Student
2. View Students
3. Update Student Grade
4. Exit
📚 Student Records 📚
1. Name: Pavesh, Age: 17, Grade: 9.52
Invalid student number!


🎓 Student Management System 🎓
1. Add Student
2. View Students
3. Update Student Grade
4. Exit
Exiting Student Management System. Goodbye!
```

📌 Summary of Day 11 ✔️ Learned Object-Oriented Programming (OOP) ✔️ Practiced Encapsulation, Methods, & Constructors ✔️ Completed a Mini-Project: Student Management System