# Internship Assignment: Object-Oriented Programming (OOP) in Python

📅 Week: 2

📂 Topic: Object-Oriented Programming in Python

🎯 Objective: Understand the fundamentals of Object-Oriented Programming (OOP) and apply OOP principles in Python.

## ✅ Learning Outcomes

By the end of this assignment, students will be able to:

* • Define and use classes and objects.
* • Apply encapsulation using methods and private attributes.
* • Understand and implement inheritance and polymorphism.
* • Use constructors (`\_\_init\_\_`) and destructors (`\_\_del\_\_`).
* • Override methods and understand `super()`.

## 🔹 Part 1: Define a Class and Create Objects

### 🧪 Example 1: Basic class definition

class Student:  
 def \_\_init\_\_(self, name, roll):  
 self.name = name  
 self.roll = roll  
  
 def display(self):  
 print(f"Name: {self.name}, Roll: {self.roll}")  
  
# Create object  
s1 = Student("Anita", 101)  
s1.display()

Task:  
1. Create a `Car` class with attributes `brand`, `model`, and `year`.  
2. Add a method `get\_details()` to display all the information.

## 🔹 Part 2: Encapsulation - Private Attributes and Methods

### 🧪 Example 2: Using private members

class Account:  
 def \_\_init\_\_(self, owner, balance):  
 self.owner = owner  
 self.\_\_balance = balance  
  
 def deposit(self, amount):  
 self.\_\_balance += amount  
  
 def get\_balance(self):  
 return self.\_\_balance  
  
acc = Account("Ravi", 1000)  
acc.deposit(500)  
print(acc.get\_balance())

Task:  
- Create a `BankAccount` class with private `balance` and public methods `deposit()`, `withdraw()`, and `check\_balance()`.

## 🔹 Part 3: Inheritance

### 🧪 Example 3: Inheriting from a parent class

class Animal:  
 def speak(self):  
 print("Animal sound")  
  
class Dog(Animal):  
 def speak(self):  
 print("Bark")  
  
d = Dog()  
d.speak()

Task:  
- Create a base class `Employee` and a derived class `Manager` that adds a `department` field.  
- Override a method in the derived class.

## 🔹 Part 4: Polymorphism and Method Overriding

### 🧪 Example 4: Method overriding

class Shape:  
 def area(self):  
 pass  
  
class Circle(Shape):  
 def area(self):  
 print("Area of Circle")  
  
class Square(Shape):  
 def area(self):  
 print("Area of Square")  
  
for shape in (Circle(), Square()):  
 shape.area()

Task:  
- Create an abstract `Shape` class with method `area()`. Derive `Rectangle` and `Triangle` classes with custom implementations.

## 🔹 Part 5: Constructor and Destructor

### 🧪 Example 5: \_\_init\_\_ and \_\_del\_\_ methods

class Demo:  
 def \_\_init\_\_(self):  
 print("Object created")  
  
 def \_\_del\_\_(self):  
 print("Object deleted")  
  
obj = Demo()  
del obj

Task:  
- Create a class `Book` with `\_\_init\_\_()` initializing book details. Use `\_\_del\_\_()` to print a message when the book object is deleted.

## 📂 Submission Checklist:

- [ ] Python script implementing all the above tasks

- [ ] Output screenshots for each example and task

- [ ] Optional: Submit as a Jupyter Notebook (.ipynb) for better presentation