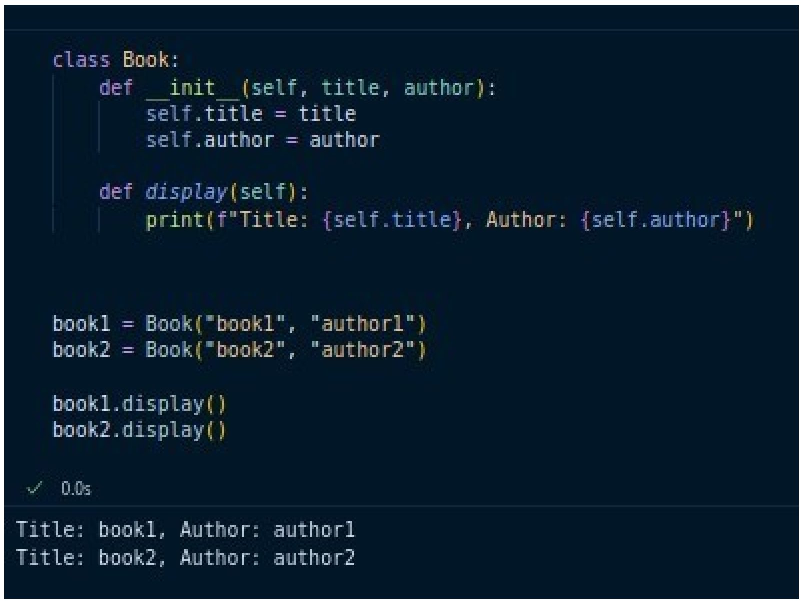
OOP in Python - Tasks

Task 1: Classes & Objects (Basic)

Create a class `Book` with attributes `title` and `author`. Create two book objects and print their details.



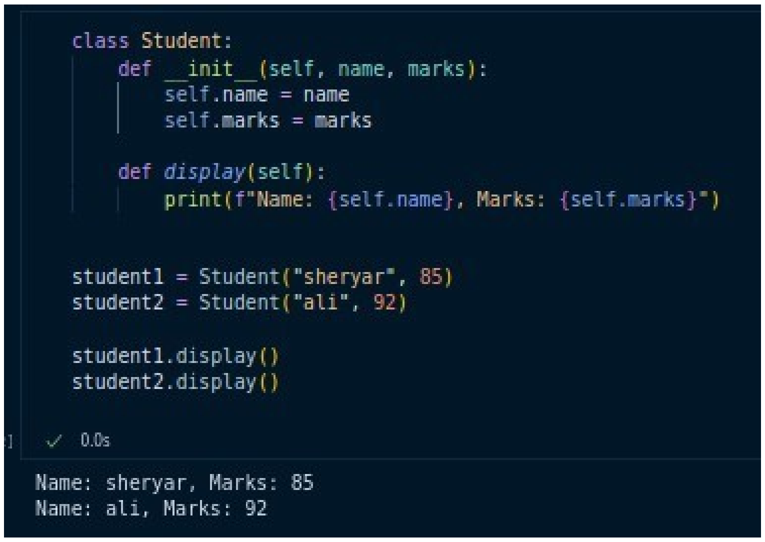
```python
class Book:
    def __init__(self, title, author):
        self.title = title
        self.author = author

    def display(self):
        print(f"Title: {self.title}, Author: {self.author}")


book1 = Book("book1", "author1")
book2 = Book("book2", "author2")

book1.display()
book2.display()
```

```
✓ 0.0s
```

```
Title: book1, Author: author1
Title: book2, Author: author2
```

Task 2: Instance Variables & Methods (Basic)

Make a class `Student` with instance variables `name` and `marks`. Add a method `display()` to print the student's details.



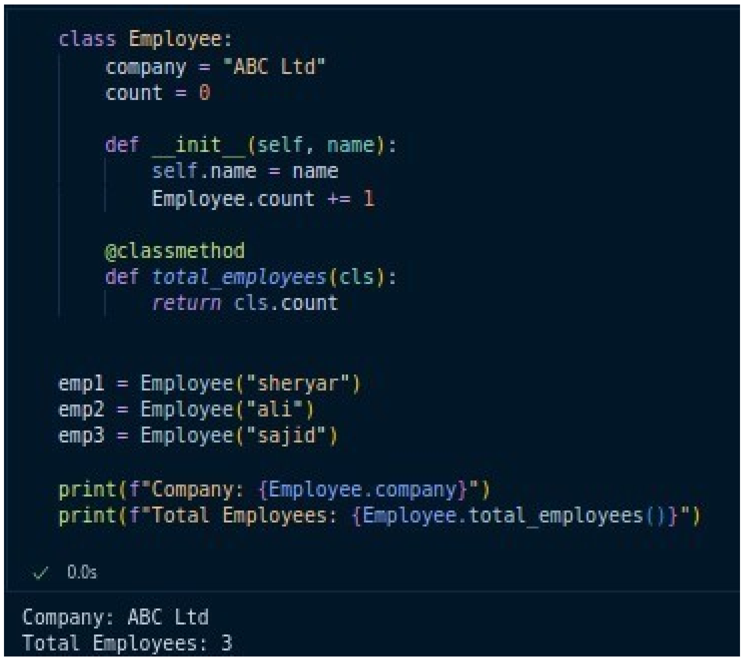
```python
class Student:
    def __init__(self, name, marks):
        self.name = name
        self.marks = marks

    def display(self):
        print(f"Name: {self.name}, Marks: {self.marks}")


student1 = Student("sheryar", 85)
student2 = Student("ali", 92)

student1.display()
student2.display()
```

```
✓ 0.0s
```

```
Name: sheryar, Marks: 85
Name: ali, Marks: 92
```

Task 3: Class Variables & Methods (Intermediate)

Create a class `Employee` with a class variable `company = "ABC Ltd"`. Add a method to count how many employees have been created.



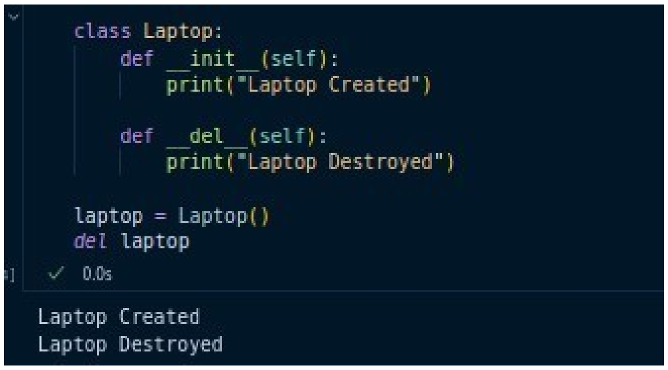
```python
class Employee:
    company = "ABC Ltd"
    count = 0

    def __init__(self, name):
        self.name = name
        Employee.count += 1

    @classmethod
    def total_employees(cls):
        return cls.count


emp1 = Employee("sheryar")
emp2 = Employee("ali")
emp3 = Employee("sajid")

print(f"Company: {Employee.company}")
print(f"Total Employees: {Employee.total_employees()}")
```

```
✓ 0.0s
```

```
Company: ABC Ltd
Total Employees: 3
```

Task 4: Constructor & Destructor (Intermediate)

Write a class `Laptop` with a constructor that prints 'Laptop Created' when an object is made, and a destructor that prints 'Laptop Destroyed' when the object is deleted.



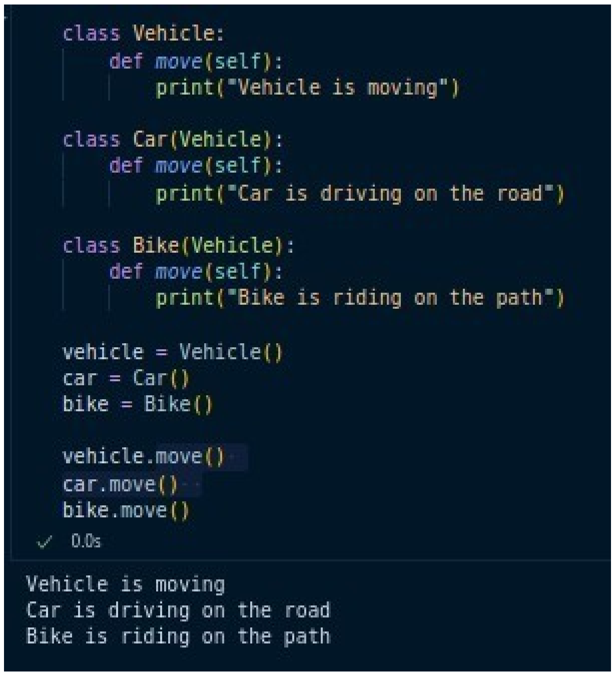
```python
class Laptop:
    def __init__(self):
        print("Laptop Created")

    def __del__(self):
        print("Laptop Destroyed")

laptop = Laptop()
del laptop
```

```
✓ 0.0s
```

```
Laptop Created
Laptop Destroyed
```

Task 5: Inheritance (Intermediate)

Create a base class `Vehicle` with a method `move()`. Create two child classes `Car` and `Bike` that override the method.



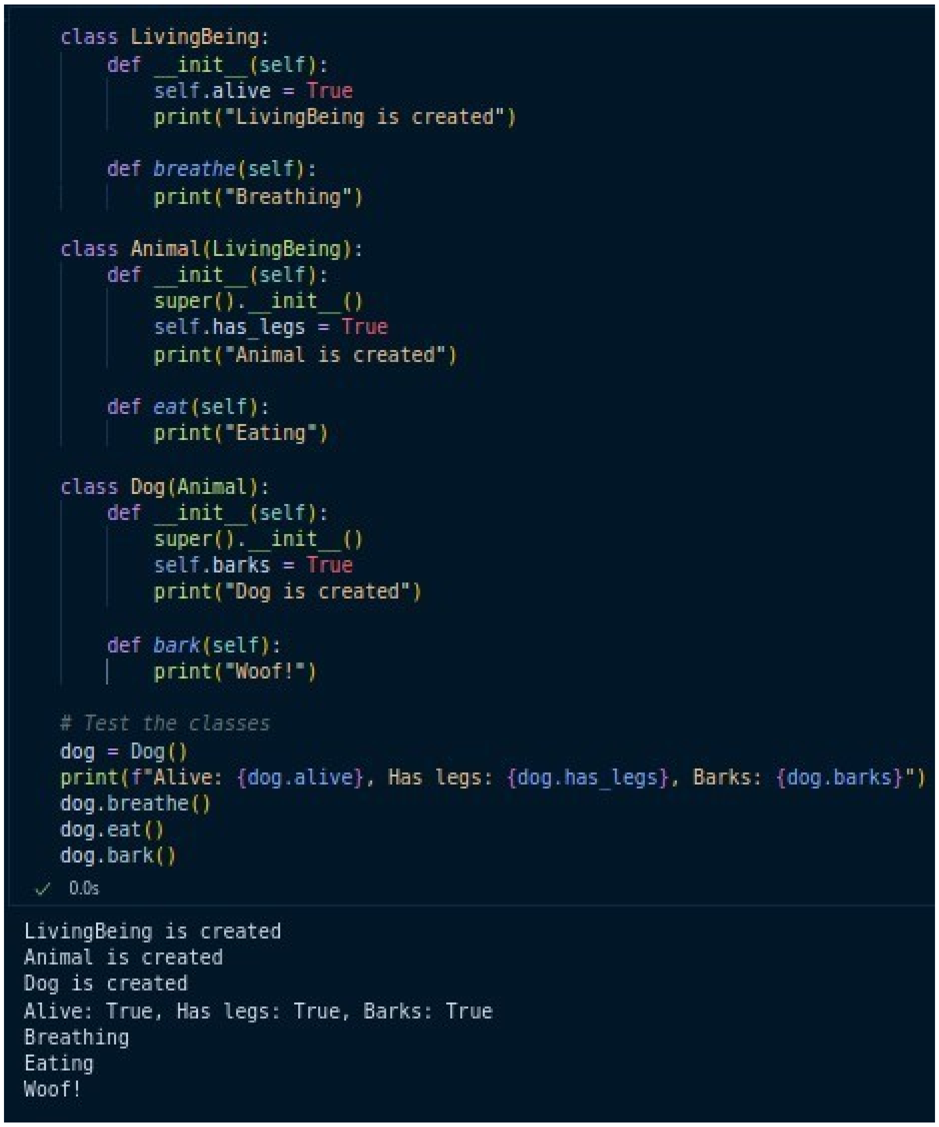
```python
class Vehicle:
    def move(self):
        print("Vehicle is moving")

class Car(Vehicle):
    def move(self):
        print("Car is driving on the road")

class Bike(Vehicle):
    def move(self):
        print("Bike is riding on the path")

vehicle = Vehicle()
car = Car()
bike = Bike()

vehicle.move()
car.move()
bike.move()
```

✓ 0.0s

```
Vehicle is moving
Car is driving on the road
Bike is riding on the path
```

Task 6: Multilevel Inheritance (Intermediate)

Build three classes: `LivingBeing` → `Animal` → `Dog`. Show how `Dog` inherits properties step by step.



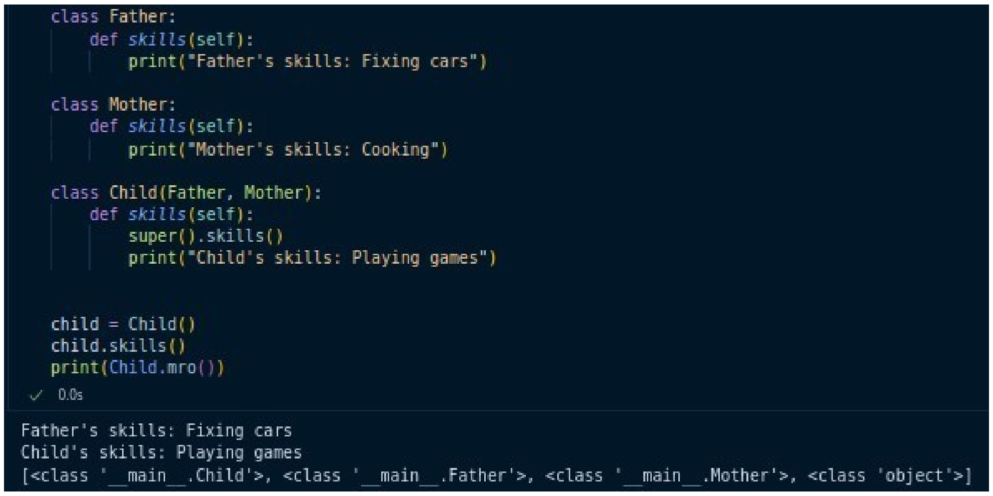
```python
class LivingBeing:
    def __init__(self):
        self.alive = True
        print("LivingBeing is created")

    def breathe(self):
        print("Breathing")

class Animal(LivingBeing):
    def __init__(self):
        super().__init__()
        self.has_legs = True
        print("Animal is created")

    def eat(self):
        print("Eating")

class Dog(Animal):
    def __init__(self):
        super().__init__()
        self.barks = True
        print("Dog is created")

    def bark(self):
        print("Woof!")

# Test the classes
dog = Dog()
print(f"Alive: {dog.alive}, Has legs: {dog.has_legs}, Barks: {dog.barks}")
dog.breathe()
dog.eat()
dog.bark()
```

✓ 0.0s

```
LivingBeing is created
Animal is created
Dog is created
Alive: True, Has legs: True, Barks: True
Breathing
Eating
Woof!
```

Task 7: Multiple Inheritance & MRO (Slightly Advanced)

Make two classes `Father` and `Mother` with a method `skills()`. Then create a `Child` class that inherits from both and demonstrate how Python resolves conflicts (using `mro()`).



```python
class Father:
    def skills(self):
        print("Father's skills: Fixing cars")

class Mother:
    def skills(self):
        print("Mother's skills: Cooking")

class Child(Father, Mother):
    def skills(self):
        super().skills()
        print("Child's skills: Playing games")


child = Child()
child.skills()
print(Child.mro())
```

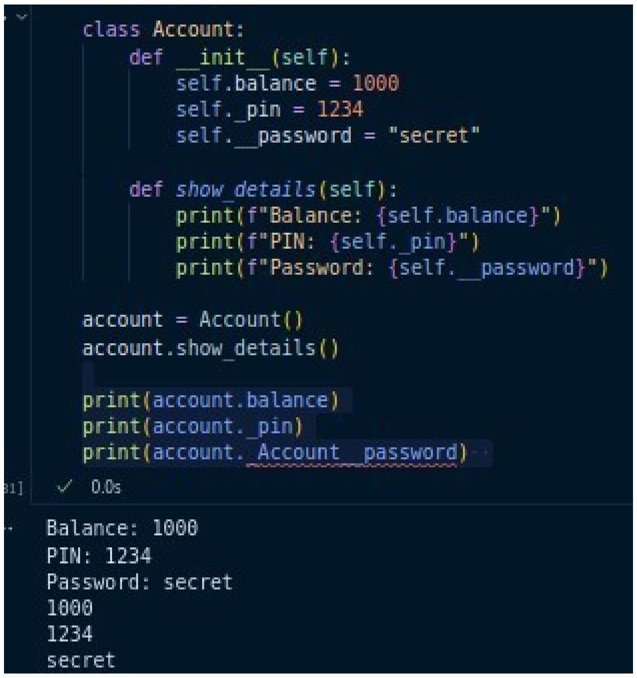
✓ 0.0s

```
Father's skills: Fixing cars
Child's skills: Playing games
[<class '__main__.Child'>, <class '__main__.Father'>, <class '__main__.Mother'>, <class 'object'>]
```

Task 8: Access Specifiers (Intermediate)

Create a class `Account` with:
- a public variable `balance`,
- a protected variable `_pin`,
- a private variable `__password`.

Show how they can be accessed or restricted.

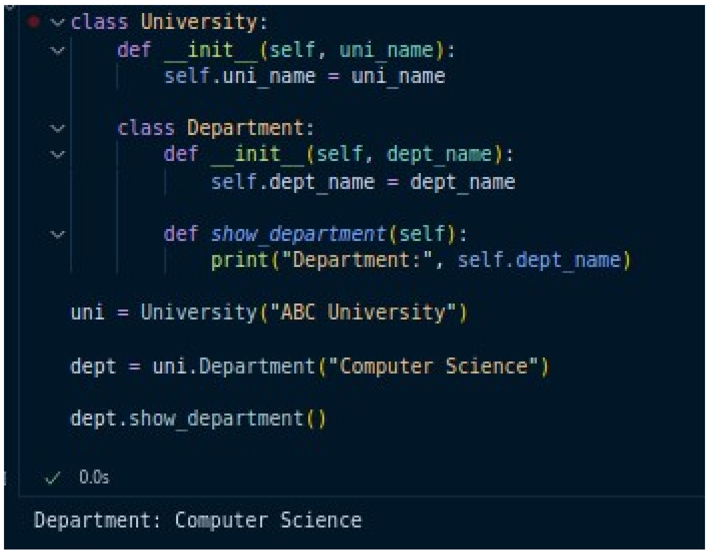


```
class Account:
    def __init__(self):
        self.balance = 1000
        self._pin = 1234
        self.__password = "secret"

    def show_details(self):
        print(f"Balance: {self.balance}")
        print(f"PIN: {self._pin}")
        print(f"Password: {self.__password}")

account = Account()
account.show_details()

print(account.balance)
print(account._pin)
print(account._Account__password)
```

```
Balance: 1000
PIN: 1234
Password: secret
1000
1234
secret
```

Task 9: Inner/Nested Class (Intermediate)

Make a class `University` with a nested class `Department`. Print the department name by accessing the inner class from an outer class object.



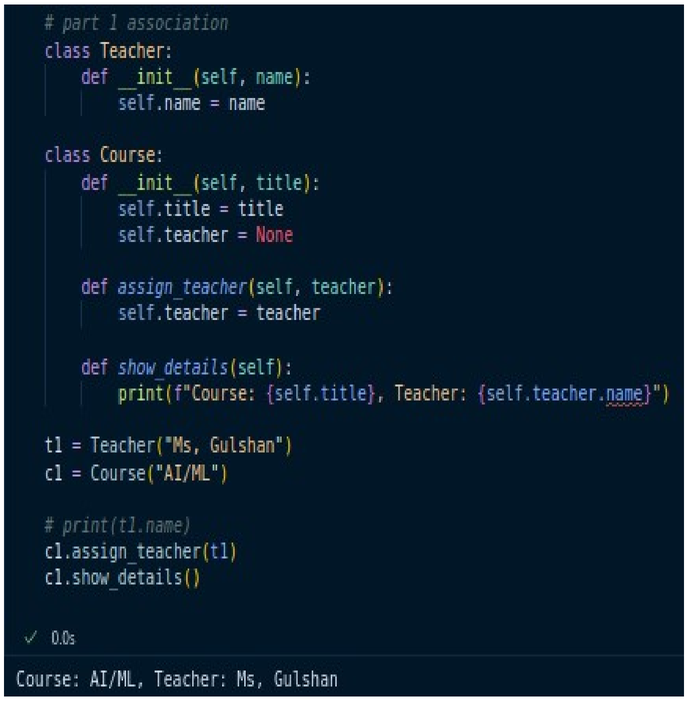
```
class University:
    def __init__(self, uni_name):
        self.uni_name = uni_name

    class Department:
        def __init__(self, dept_name):
            self.dept_name = dept_name

        def show_department(self):
            print("Department:", self.dept_name)

uni = University("ABC University")

dept = uni.Department("Computer Science")

dept.show_department()
```

```
Department: Computer Science
```

Task 10: Association, Aggregation & Composition (Advanced)

- Association: Create two independent classes `Teacher` and `Course` and link them

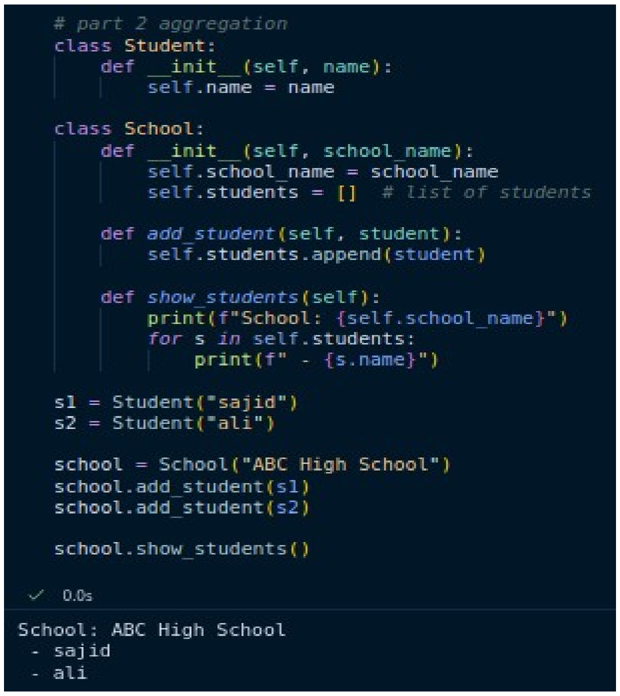
by assigning a teacher to a course.
- Aggregation: A `School` class contains multiple `Student` objects but students can exist without the school.
- Composition: A `Car` class always creates an `Engine` object inside it, showing that the engine cannot exist without the car.



```python
# part 1 association
class Teacher:
    def __init__(self, name):
        self.name = name

class Course:
    def __init__(self, title):
        self.title = title
        self.teacher = None

    def assign_teacher(self, teacher):
        self.teacher = teacher

    def show_details(self):
        print(f"Course: {self.title}, Teacher: {self.teacher.name}")

t1 = Teacher("Ms, Gulshan")
c1 = Course("AI/ML")

# print(t1.name)
c1.assign_teacher(t1)
c1.show_details()
```

✓ 0.0s

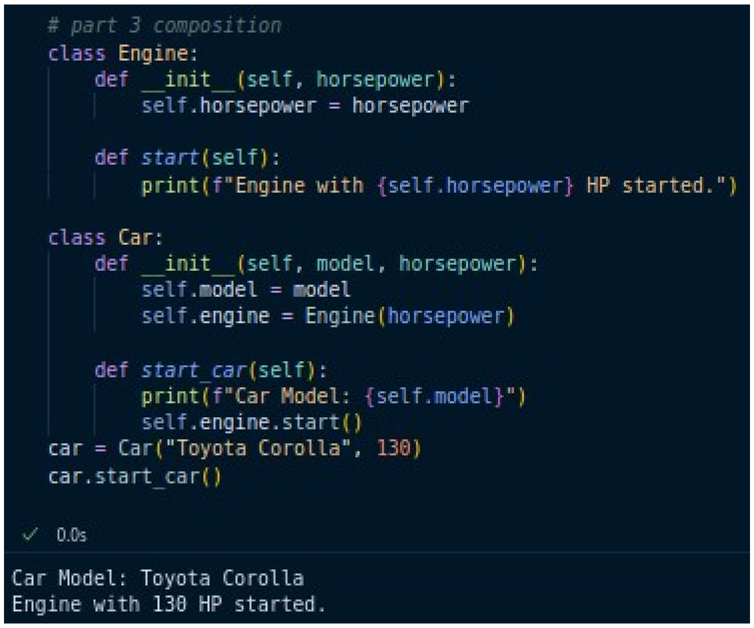
```
Course: AI/ML, Teacher: Ms, Gulshan
```



```python
# part 2 aggregation
class Student:
    def __init__(self, name):
        self.name = name

class School:
    def __init__(self, school_name):
        self.school_name = school_name
        self.students = []  # list of students

    def add_student(self, student):
        self.students.append(student)

    def show_students(self):
        print(f"School: {self.school_name}")
        for s in self.students:
            print(f" - {s.name}")

s1 = Student("sajid")
s2 = Student("ali")

school = School("ABC High School")
school.add_student(s1)
school.add_student(s2)

school.show_students()
```

✓ 0.0s

```
School: ABC High School
 - sajid
 - ali
```



```python
# part 3 composition
class Engine:
    def __init__(self, horsepower):
        self.horsepower = horsepower

    def start(self):
        print(f"Engine with {self.horsepower} HP started.")

class Car:
    def __init__(self, model, horsepower):
        self.model = model
        self.engine = Engine(horsepower)

    def start_car(self):
        print(f"Car Model: {self.model}")
        self.engine.start()
car = Car("Toyota Corolla", 130)
car.start_car()
```

✓ 0.0s

```
Car Model: Toyota Corolla
Engine with 130 HP started.
```