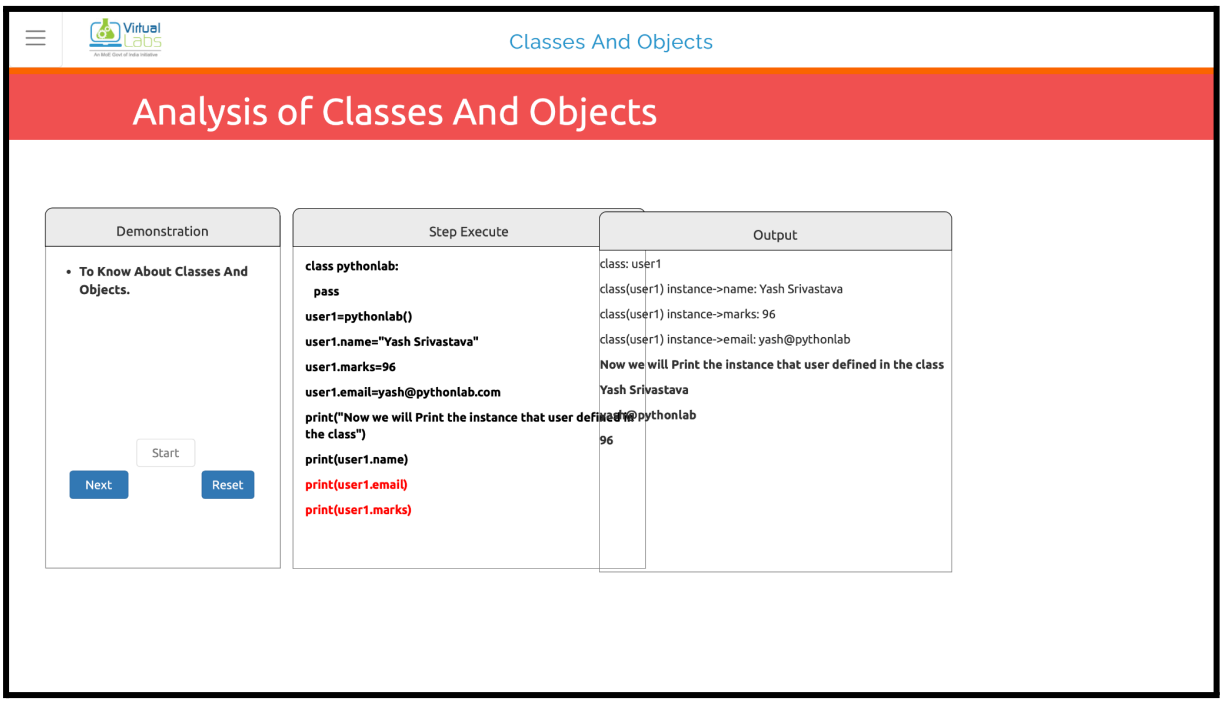
Artificial Intelligence Programming

Assignment 1- Simulation and code on classes and objects

Simulation:



In this simulation, the class is seen as a blueprint, and `user1` is a specific example created from that blueprint. The class is like a template, and `user1` is like a form filled out with details. The details include a person's name (`Yash Srivastava`), their marks (`96`), and email (`yash@pythonlab.com`). The simulation then shows how to print and access these details for the `user1` example. It's just like creating a personalized card with your name, marks, and email, and then printing it whenever you want.

Code Implementation:

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

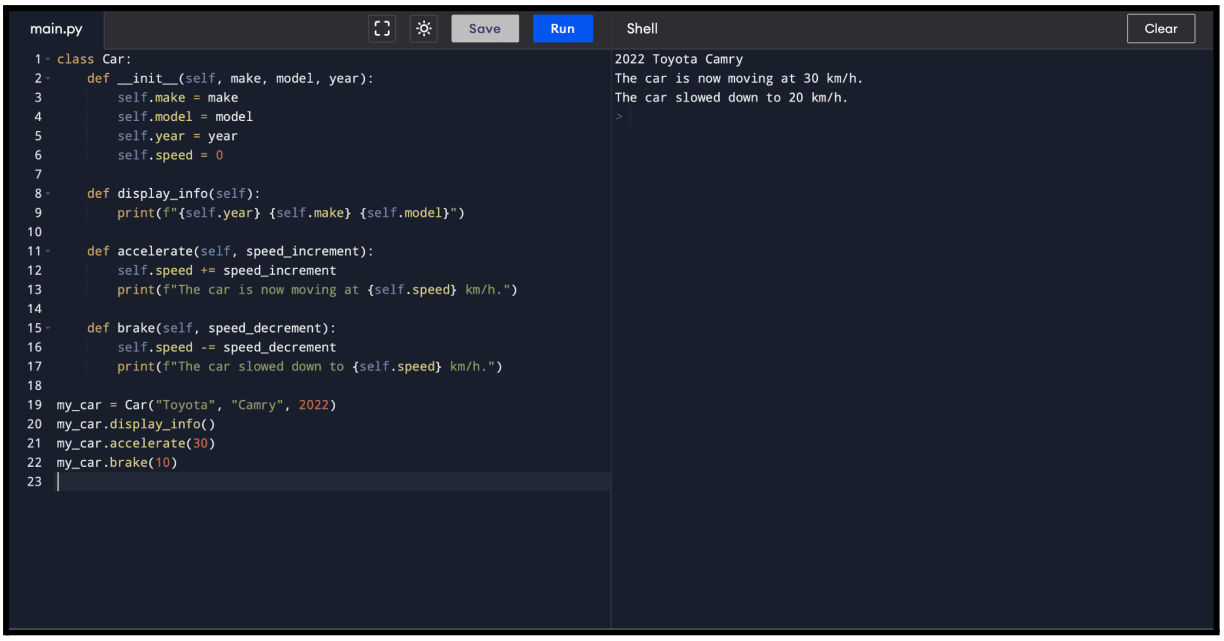
```
        self.model = model
        self.year = year
        self.speed = 0

    def display_info(self):
        print(f"{self.year} {self.make} {self.model}")

    def accelerate(self, speed_increment):
        self.speed += speed_increment
        print(f"The car is now moving at {self.speed} km/h.")

    def brake(self, speed_decrement):
        self.speed -= speed_decrement
        print(f"The car slowed down to {self.speed} km/h.")

my_car = Car("Toyota", "Camry", 2022)
my_car.display_info()
my_car.accelerate(30)
my_car.brake(10)
```



<u>Explanation of the code:</u>

class Car:

➔ We're creating a blueprint called ‘Car’ that will be used to make cars in our program.

def __init__(self, make, model, year):

- ➔ This function sets up a new car with specific characteristics: ‘make’ (brand), ‘model’, and ‘year’ (manufacturing year).
- ➔ Additionally, it initializes the car's speed to 0.

def display_info(self):

- ➔ This function prints out information about the car - its year, make, and model.

def accelerate(self, speed_increment):

- ➔ This function increases the car's speed by a specified amount (‘speed_increment’).
- ➔ It then prints a message indicating the new speed of the car.

def brake(self, speed_decrement):

- ➔ This function decreases the car's speed by a specified amount (‘speed_decrement’).
- ➔ It then prints a message indicating the new speed of the car.

my_car = Car("Toyota", "Camry", 2022):

- ➔ This line creates a specific car instance called ‘my_car’ with the make "Toyota," model "Camry," and year 2022.

my_car.display_info():

- ➔ This line calls the ‘display_info’ method for the ‘my_car’ instance, which prints out the car's information.

my_car.accelerate(30):

- ➔ This line calls the ‘accelerate’ method for the ‘my_car’ instance, increasing its speed by 30 km/h.

my_car.brake(10):

- ➔ This line calls the ‘brake’ method for the ‘my_car’ instance, decreasing its speed by 10 km/h.

Summary:

In this assignment, the simulation and code implementation on classes and objects provided a practical introduction to the fundamental concepts of object-oriented programming (OOP). The simulation showed the class as a blueprint and an instance (`user1`) as a personalized form filled with specific details. The details, such as name, marks, and email, showcased the attributes associated with an individual instance of the class.

Moving on I implemented my code with the 'Car' class. The class served as a template for creating individual cars (`my_car`), each with unique attributes like make, model, year, and speed. The `__init__` method acted as a constructor, initializing these attributes upon the creation of a new car instance. The methods (`display_info`, `accelerate`, `brake`) demonstrated encapsulation, providing a clear structure for interacting with and changing the car's properties. The process of creating `my_car` and calling methods like `display_info`, `accelerate`, and `brake` helped with the understanding of instantiation, method calling, and data access in the context of OOP. It highlights the importance of proper documentation, indentation, and code aesthetics for clarity and maintainability.

This assignment showed the significance of OOP in organizing code, enhancing code reusability, and promoting a modular programming approach. The ability to create classes and objects not only facilitates efficient code organization but also mirrors real-world scenarios, making it a crucial paradigm in the development of AI applications. Overall, the hands-on experience with the simulation and code implementation served as a foundational step in grasping the principles of OOP and its relevance in artificial intelligence programming.

References:

https://python-iitk.vlabs.ac.in/exp/classes-and- objects/pretest.html

https://python-iitk.vlabs.ac.in/exp/classes-and- objects/posttest.html

W3Schools