# Assignment 5 oops

August 25, 2023

Q1. Create a vehicle class with an init method having instance variables as name_of_vehicle, max_speed and average_of_vehicle.

[1]:
```
class vehicle:

        def __init__(self,name_of_vehicle, max_speed, average_of_vehicle):
            self.name_of_vehicle=name_of_vehicle
            self.max_speed=max_speed
            self.average_of_vehicle=average_of_vehicle

        def car_info(self):
            return self.name_of_vehicle,self.max_speed,self.average_of_vehicle
```

[3]:
```
obj_car=vehicle('maserati',350,12.25)
```

[5]:
```
obj_car.car_info()
```

[5]:
```
('maserati', 350, 12.25)
```

[ ]:
```

```

Q2. Create a child class car from the vehicle class created in Que 1, which will inherit the vehicle class. Create a method named seating_capacity which takes capacity as an argument and returns the name of the vehicle and its seating capacity.

[37]:
```
class vehicle:
    def __init__(self, name_of_vehicle, max_speed, average_of_vehicle):
        self.name_of_vehicle = name_of_vehicle
        self.max_speed = max_speed
        self.average_of_vehicle = average_of_vehicle

class car(vehicle):
    def __init__(self, name_of_vehicle, max_speed, average_of_vehicle):
        super().__init__(name_of_vehicle, max_speed, average_of_vehicle)

    def seating_capacity(self, capacity):
        return f"{self.name_of_vehicle} has a seating capacity of {capacity}␣
 ↪people."
```

```
car_instance = car("BMW M4", 180, 25)
capacity_result = car_instance.seating_capacity(5)
print(capacity_result)
```

```
BMW M4 has a seating capacity of 5 people.
```

[ ]:

Q3. What is multiple inheritance? Write a python code to demonstrate multiple inheritance.

ANS: Multiple inheritance is a feature in object-oriented programming languages that allows a class to inherit attributes and methods from more than one parent class. In other words, a class can inherit from multiple base classes, acquiring the characteristics of all those classes.

[38]:
```
class Parent1:
    def method1(self):
        print("Method 1 from Parent1")

class Parent2:
    def method2(self):
        print("Method 2 from Parent2")

class Child(Parent1, Parent2):
    def method3(self):
        print("Method 3 from Child")
```

[39]:
```
child_instance = Child()
```

[41]:
```
child_instance.method1()
```

```
Method 1 from Parent1
```

[42]:
```
child_instance.method2()
```

```
Method 2 from Parent2
```

[44]:
```
child_instance.method3()
```

```
Method 3 from Child
```

[ ]:

Q4. What are getter and setter in python? Create a class and create a getter and a setter method in this class.

ANS: In Python, getters and setters are methods used to access and modify the attributes (instance variables) of a class in a controlled manner. They are used to provide a layer of abstraction over

attribute access and modification, allowing you to apply validation, computation, or other logic before getting or setting the values.

[45]:
```python
class Person:
    def __init__(self, name, age):
        self._name = name
        self._age = age

    def get_name(self):
        return self._name

    def set_name(self, new_name):
        if isinstance(new_name, str):
            self._name = new_name
        else:
            print("Name must be a string.")

    def get_age(self):
        return self._age

    def set_age(self, new_age):
        if isinstance(new_age, int) and new_age >= 0:
            self._age = new_age
        else:
            print("Age must be a non-negative integer.")
```

[54]:
```python
person_instance = Person("Jarvis", 45)
```

[55]:
```python
print("Initial name:", person_instance.get_name())
print("Initial age:", person_instance.get_age())
```

```
Initial name: Jarvis
Initial age: 45
```

[56]:
```python
person_instance.set_name("Veronica")
person_instance.set_age(25)
```

[57]:
```python
print("Updated name:", person_instance.get_name())
print("Updated age:", person_instance.get_age())
```

```
Updated name: Veronica
Updated age: 25
```

[ ]:
```python

```

Q5.What is method overriding in python? Write a python code to demonstrate method overriding.

ANS: Method overriding in Python occurs when a subclass provides a specific implementation for a method that is already defined in its parent class. This allows the subclass to customize or extend

the behavior of the inherited method without changing its name or parameters.

[59]:
```python
class Parent:
    def show(self):
        print("This is the Parent class")

class Child(Parent):
    def show(self):
        print("This is the Child class")
```

[60]:
```python
parent_instance = Parent()
child_instance = Child()
```

[62]:
```python
parent_instance.show()
child_instance.show()
```

```
This is the Parent class
This is the Child class
```

[ ]:
```python

```