Q1, Create a vehicle class with an init method having instance variables as name\_of\_vehicle, max\_speed

and average\_of\_vehicle.

an example of a Vehicle class in Python with an \_\_init\_\_ method and instance variables for the name of the vehicle, maximum speed, and average speed:

class Vehicle:

def \_\_init\_\_(self, name\_of\_vehicle, max\_speed, average\_speed):

self.name\_of\_vehicle = name\_of\_vehicle

self.max\_speed = max\_speed

self.average\_speed = average\_speed

the \_\_init\_\_ method is used to initialize the instance variables name\_of\_vehicle, max\_speed, and average\_speed. These variables can be accessed and modified using dot notation, as shown in the example below:

car = Vehicle("Car", 200, 100) # Creating an instance of the Vehicle class

print(car.name\_of\_vehicle) # Output: Car

print(car.max\_speed) # Output: 200

print(car.average\_speed) # Output: 100

car.average\_speed = 120 # Modifying the average\_speed variable

print(car.average\_speed) # Output: 120

an instance of the Vehicle class is created with the name "Car", maximum speed 200, and average speed 100. The instance variables are accessed using dot notation (car.name\_of\_vehicle, car.max\_speed, car.average\_speed), and the average\_speed variable is modified and printed again.

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.

example of how you can create a child class called "Car" that inherits from the "Vehicle" class and includes a method called "seating\_capacity":

class Vehicle:

def \_\_init\_\_(self, name):

self.name = name

def display\_info(self):

return f"Name: {self.name}"

class Car(Vehicle):

def seating\_capacity(self, capacity):

return f"{self.name} - Seating Capacity: {capacity}"

# Create an instance of the Car class

my\_car = Car("My Car")

# Call the method to get the seating capacity

capacity = 5

result = my\_car.seating\_capacity(capacity)

# Display the result

print(result)

Output:

My Car - Seating Capacity: 5

the Car class is defined as a child class of the Vehicle class. The seating\_capacity method is defined within the Car class, which takes the seating capacity as an argument. It returns a string that includes the name of the vehicle and its seating capacity. We create an instance of the Car class, and then we call the seating\_capacity method on that instance with a capacity of 5. The resulting string is then printed to the console.

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

Multiple inheritance is a feature in object-oriented programming languages, including Python, where a class can inherit attributes and methods from multiple parent classes. This means that a child class can derive characteristics from more than one base class.

Here's an example code in Python to demonstrate multiple inheritance:

class Animal:

def \_\_init\_\_(self, name):

self.name = name

def eat(self):

print(f"{self.name} is eating.")

def sleep(self):

print(f"{self.name} is sleeping.")

class Flyable:

def fly(self):

print(f"{self.name} is flying.")

class Bird(Animal, Flyable):

def \_\_init\_\_(self, name):

super().\_\_init\_\_(name)

def chirp(self):

print(f"{self.name} is chirping.")

# Create an instance of the Bird class

bird = Bird("Sparrow")

bird.eat() # Output: Sparrow is eating.

bird.sleep() # Output: Sparrow is sleeping.

bird.fly() # Output: Sparrow is flying.

bird.chirp() # Output: Sparrow is chirping.

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

In Python, getter and setter methods are used to encapsulate the access and modification of class attributes, providing control over how the attributes are read and updated. They allow you to enforce data validation, perform additional actions, or hide the underlying implementation details.

Here's an example of a class with a getter and a setter method for an attribute:

class Person:

def \_\_init\_\_(self, name):

self.\_name = name # Note the use of a single underscore to indicate it's a private attribute

def get\_name(self):

return self.\_name

def set\_name(self, name):

self.\_name = name

# Creating an instance of the Person class

person = Person("Alice")

# Using the getter method to retrieve the attribute

print(person.get\_name()) # Output: Alice

# Using the setter method to update the attribute

person.set\_name("Bob")

print(person.get\_name()) # Output: Bob

example, the Person class has a private attribute \_name which can only be accessed and modified through the getter and setter methods, get\_name() and set\_name() respectively. The getter method simply returns the value of \_name, while the setter method allows you to update the value of \_name by providing a new name.

By using getter and setter methods, you have control over how the attribute is accessed and modified, enabling you to implement additional logic or validations if needed.

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

Method overriding in Python is a concept where a subclass provides a different implementation of a method that is already defined in its superclass. It allows the subclass to provide its own specific implementation of a method, thereby modifying or extending the behavior defined in the superclass.

Here's an example code that demonstrates method overriding in Python:

class Vehicle:

def accelerate(self):

print("The vehicle is accelerating.")

class Car(Vehicle):

def accelerate(self):

print("The car is accelerating.")

class Motorcycle(Vehicle):

def accelerate(self):

print("The motorcycle is accelerating.")

# Create objects

vehicle = Vehicle()

car = Car()

motorcycle = Motorcycle()

# Call accelerate method

vehicle.accelerate() # Output: The vehicle is accelerating.

car.accelerate() # Output: The car is accelerating.

motorcycle.accelerate() # Output: The motorcycle is accelerating.