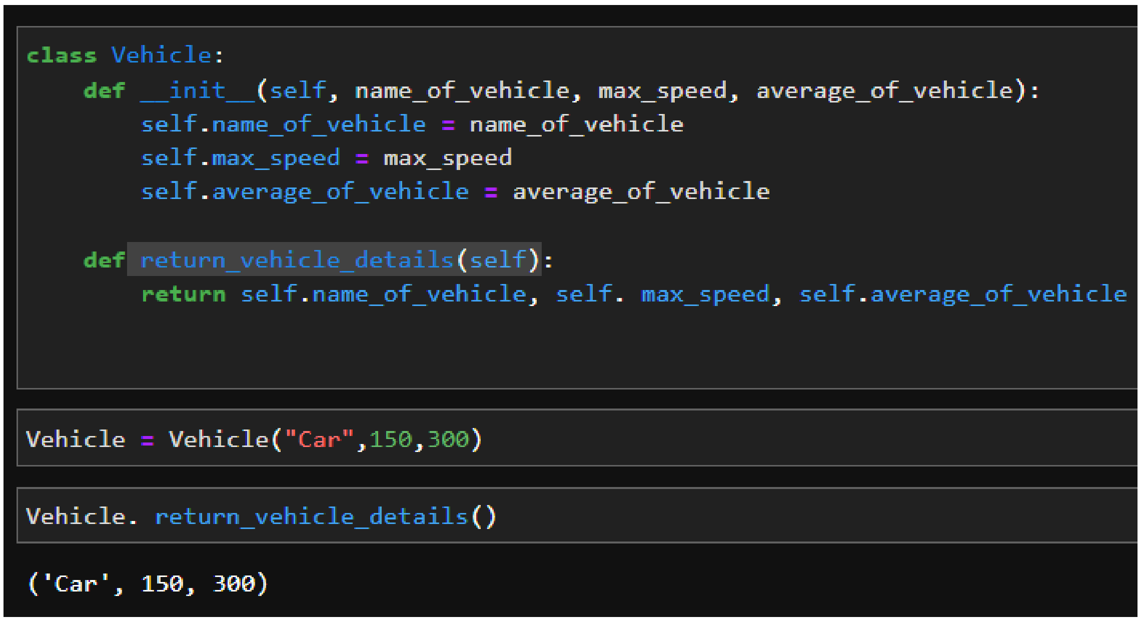
Q1. Create a vehicle class with an init method having instance variables as name_of_vehicle, max_speed and average_of_vehicle.



```
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 return_vehicle_details(self):
        return self.name_of_vehicle, self. max_speed, self.average_of_vehicle
```

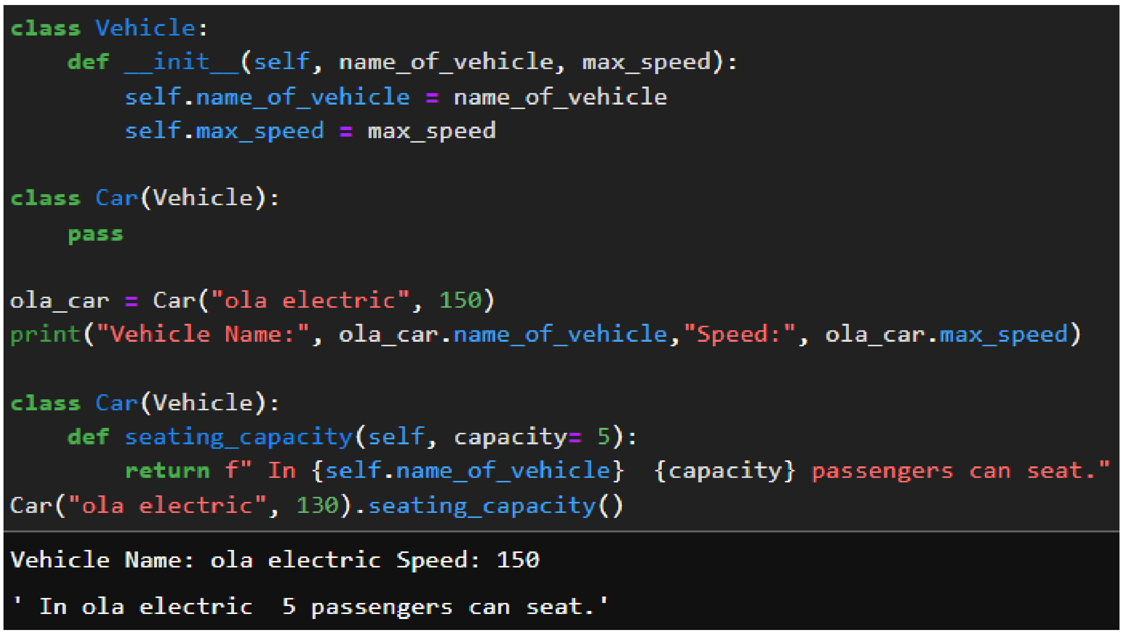
```
Vehicle = Vehicle("Car",150,300)
```

```
Vehicle. return_vehicle_details()
```

```
('Car', 150, 300)
```

Q2. Create a child class car from the vehicle class created in Que 1, which will inherit the vehicle class.

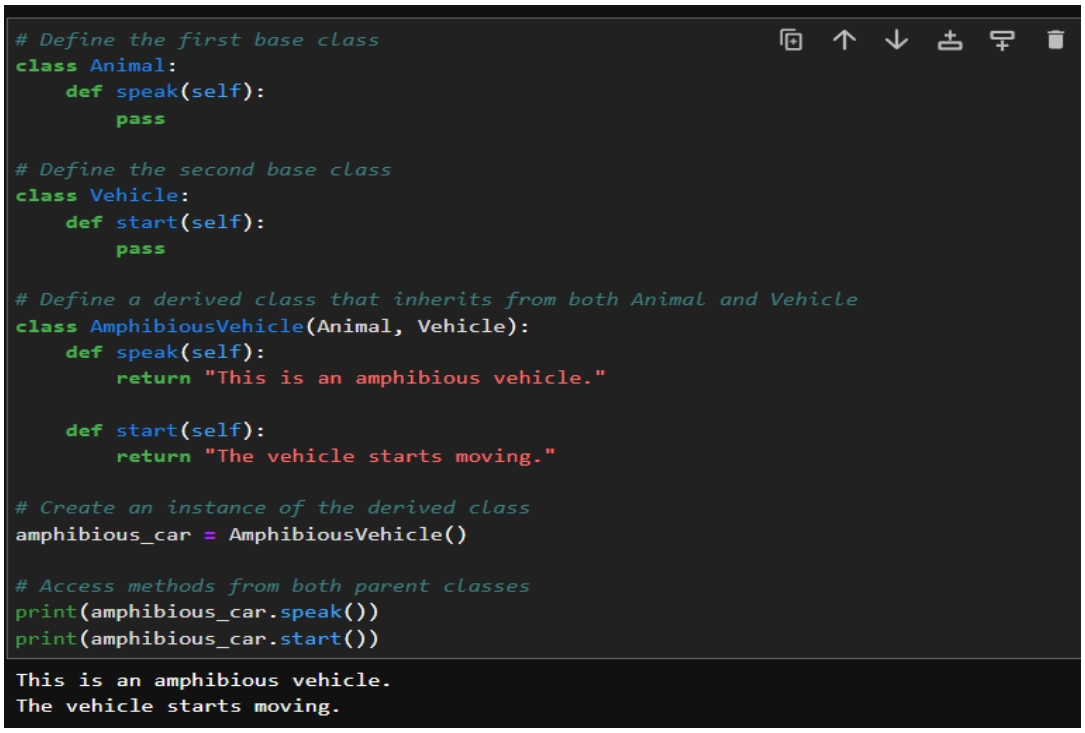
Create a method named seating_capacity which takes capacity as an argument and returns the name of the vehicle and its seating capacity.



```
class Vehicle:
    def __init__(self, name_of_vehicle, max_speed):
        self.name_of_vehicle = name_of_vehicle
        self.max_speed = max_speed

class Car(Vehicle):
    pass

ola_car = Car("ola electric", 150)
print("Vehicle Name:", ola_car.name_of_vehicle,"Speed:", ola_car.max_speed)

class Car(Vehicle):
    def seating_capacity(self, capacity= 5):
        return f" In {self.name_of_vehicle}  {capacity} passengers can seat."
Car("ola electric", 130).seating_capacity()
```

```
Vehicle Name: ola electric Speed: 150
' In ola electric  5 passengers can seat.'
```

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

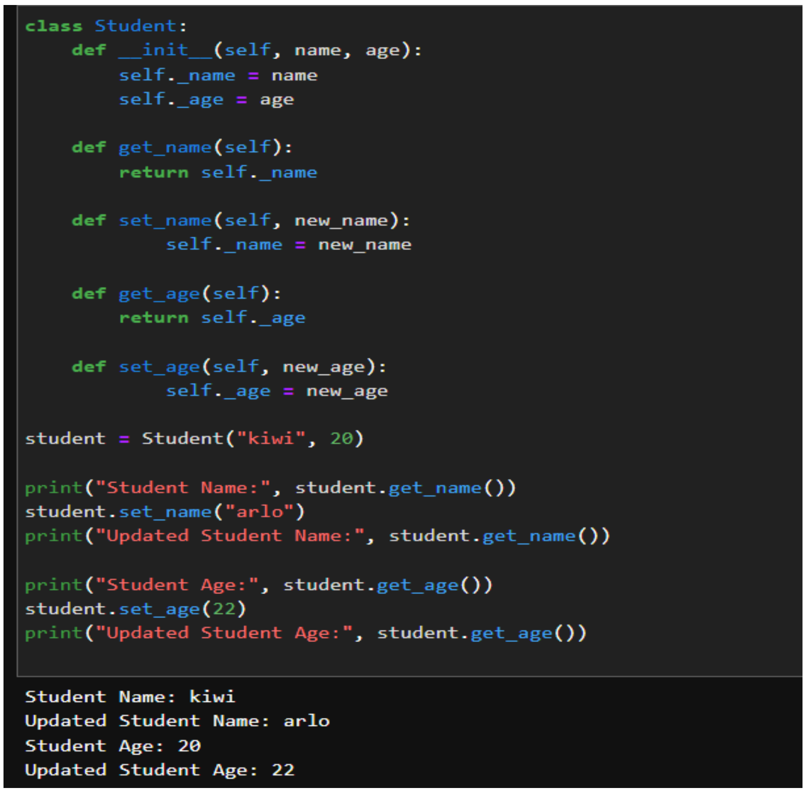


```python
# Define the first base class
class Animal:
    def speak(self):
        pass

# Define the second base class
class Vehicle:
    def start(self):
        pass

# Define a derived class that inherits from both Animal and Vehicle
class AmphibiousVehicle(Animal, Vehicle):
    def speak(self):
        return "This is an amphibious vehicle."

    def start(self):
        return "The vehicle starts moving."

# Create an instance of the derived class
amphibious_car = AmphibiousVehicle()

# Access methods from both parent classes
print(amphibious_car.speak())
print(amphibious_car.start())
```

```
This is an amphibious vehicle.
The vehicle starts moving.
```

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

-Getters and setters are methods used to access and modify the attributes (instance variables) of a class in a controlled way. They are often used to enforce encapsulation and provide a mechanism for controlling access to class attributes, allowing you to add validation or custom logic when getting or setting values. Conventionally, Python uses property methods to create getters and setters.

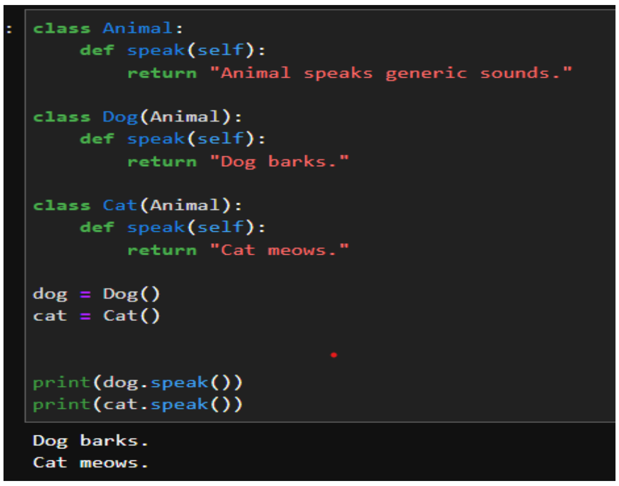


```python
class Student:
    def __init__(self, name, age):
        self._name = name
        self._age = age

    def get_name(self):
        return self._name

    def set_name(self, new_name):
            self._name = new_name

    def get_age(self):
        return self._age

    def set_age(self, new_age):
            self._age = new_age

student = Student("kiwi", 20)

print("Student Name:", student.get_name())
student.set_name("arlo")
print("Updated Student Name:", student.get_name())

print("Student Age:", student.get_age())
student.set_age(22)
print("Updated Student Age:", student.get_age())
```

```
Student Name: kiwi
Updated Student Name: arlo
Student Age: 20
Updated Student Age: 22
```

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

Method overriding is a feature in object-oriented programming (OOP) that allows a subclass to provide a specific implementation of a method that is already defined in its parent class. When a subclass overrides a method, it provides a new implementation for that method while keeping the method's name and parameters the same. The overridden method in the subclass is then used instead of the method in the parent class when called on an instance of the subclass.



```python
class Animal:
    def speak(self):
        return "Animal speaks generic sounds."

class Dog(Animal):
    def speak(self):
        return "Dog barks."

class Cat(Animal):
    def speak(self):
        return "Cat meows."

dog = Dog()
cat = Cat()


print(dog.speak())
print(cat.speak())
```

```
Dog barks.
Cat meows.
```