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

In [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
```

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.

In [2]:

```
class car(vehicle):
    def seating_capacity(self,capacity):
        self.capacity = capacity
        return self.capacity,self.name_of_vehicle
```

In [3]:

```
car1 = car("range rover",250,13.07)
```

In [4]:

```
car1.seating_capacity(7)
```

Out[4]:

```
(7, 'range rover')
```

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

ANS : When a class is derived from more than one base class it is called multiple Inheritance. The derived class inherits all the features of the base case

In [5]:

```
class class1:
    def class_1(self):
        print("this is class1")
class class2:
    def class_2(self):
        print("this is class2")
class class3(class1,class2):
    def class_3(self):
        print("this is class3")
    pass

obj = class3()
obj.class_1()
obj.class_2()
obj.class_3()
```

```
this is class1
this is class2
this is class3
```

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

ANS : For the purpose of data encapsulation, most object oriented languages use getters and setters method. This is because we want to hide the attributes of a object class from other classes so that no accidental modification of the data happens by methods in other classes.

As the name suggests, getters are the methods which help access the private attributes or get the value of the private attributes and setters are the methods which help change or set the value of private attributes.

In [6]:

```
class aging:
    def __init__(self):
        self._age = 0

    def get_age(self):
        print("getter method called")
        return self._age


    def set_age(self, a):
        print("setter method called")
        self._age = a


    def del_age(self):
        del self._age

    age = property(get_age, set_age, del_age)

mark = aging()

mark.age = 10

print(mark.age)
```

```
setter method called
getter method called
10
```

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

ANS : Method overriding is an ability of any object-oriented programming language that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super-classes or parent classes. When a method in a subclass has the same name, same parameters or signature and same return type(or sub-type) as a method in its super-class, then the method in the subclass is said to override the method in the super-class.

In [7]:

```
class Parent():
    def __init__(self):
        self.value = "Inside Parent"
    def show(self):
        print(self.value)

class Child(Parent):

    def __init__(self):
        self.value = "Inside Child"

    def show(self):
        print(self.value)

obj1 = Parent()
obj2 = Child()
obj1.show()
obj2.show()
```

```
Inside Parent
Inside Child
```