In [ ]:
```
#Q1
```

In [5]:
```
class Vehicle:
    def __init__(self,name_of_vehicle,max_speed,average_of_vehicle):
        self.name_of_vehicle1=name_of_vehicle
        self.max_speed1=max_speed
        self.average_of_vehicle1=average_of_vehicle
veh=Vehicle('honda',240,62.25)
```

In [6]:
```
veh.name_of_vehicle1
```

Out[6]:
```
'honda'
```

In [7]:
```
veh.average_of_vehicle1
```

Out[7]:
```
62.25
```

In [8]:
```
veh.max_speed1
```

Out[8]:
```
240
```

In [17]:
```
#Q2
```

In [18]:
```
class Vehicle:
    def seating_capacity(self,capacity=50):
        return"seating capacity of a honda is 50 passengers"
class car(Vehicle):
    pass
car_obj=car()
car_obj.seating_capacity()
```

Out[18]:
```
'seating capacity of a honda is 50 passengers'
```

In [19]:
```
#Q3
```

In [20]:
```
#multiple inhertance means one class inherit the property of two classes is called
```

In [24]:
```
#Ex
class Class1:
    def test_class1(self):
        return"this is a class1"
class Class2:
    def test_class2(self):
        return"this is a class2"
class Class3(Class1,Class2):
    pass
obj_class3=Class3()
obj_class3.test_class2()
```

Out[24]:
```
'this is a class2'
```

In [25]:
```
#Q4
```

In [26]:
```
#
#Getter: A method that allows you to access an attribute in a given class.
#Setter: A method that allows you to set or mutate the value of an attribute in a c
```

In [37]:
```
#Ex
class Car:
    def __init__(self,year,make,model,speed):
        self.__year=year
        self.__make=make
        self.__model=model
        self.__speed=0
    def set_speed(self,speed):
        self.__speed=0 if speed<0 else speed
    def get_speed(self):
        return self.__speed
c=Car(2021,"toyoto","innova",234)
c.set_speed(234)
c.get_speed()
```

Out[37]: 234

In [39]:
```
#Q5
```

In [40]:
```
#Method overriding is an ability of any object-oriented programming language that a
#When a method in a subclass has the same name, same parameters or signature and sa
```

In [41]:
```
#Ex
# Defining parent class
class Parent():

    # Constructor
    def __init__(self):
        self.value = "Inside Parent"

    # Parent's show method
    def show(self):
        print(self.value)

# Defining child class
class Child(Parent):

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

    # Child's show method
    def show(self):
        print(self.value)
# Driver's code
obj1 = Parent()
```

```
obj2 = Child()

obj1.show()
obj2.show()
```

```
Inside Parent
Inside Child
```

In [ ]: