Department of Computer Science and Engineering (Data Science)

JHANVI PAREKH

60009210033

CSE(Data Science)

A-1

Experiment No 2

AIM: Implement Object Oriented Programming Concepts using Python Theory:

Object Oriented Programing

Object-oriented programming (OOP) is a computer programming model that organizes software design around data, or objects, rather than functions and logic. This includes programs for manufacturing and design, as well as mobile applications; for example, OOP can be used for manufacturing system simulation software.

Object Oriented Programming in Python

Python is a multi-paradigm programming language. It supports different programming approaches. One of the popular approaches to solve a programming problem is by creating objects. This is known as Object-Oriented Programming (OOP).

An object has two characteristics:

attributes behavior

Let's take an example:

A parrot is an object, as it has the following properties:

name, age, color as attributes singing,

dancing as behavior

The concept of OOP in Python focuses on creating reusable code. This concept is also known as DRY (Don't Repeat Yourself).

In Python, the concept of OOP follows some basic principles:

Class

A class is a blueprint for the object.

We can think of class as a sketch of a parrot with labels. It contains all the details about the name, colors, size etc. Based on these descriptions, we can study about the parrot. Here, a parrot is an object.

The example for class of parrot can be:

class Parrot:

pass

Here, we use the class keyword to define an empty class Parrot. From class, we construct instances. An instance is a specific object created from a particular class.

Object

An object (instance) is an instantiation of a class. When class is defined, only the description for the object is defined. Therefore, no memory or storage is allocated.

The example for object of parrot class can be:

obj = Parrot()

Here, obj is an object of class Parrot.

Suppose we have details of parrots. Now, we are going to show how to build the class and objects of parrots.

Example 1: Creating Class and Object in Python

class Parrot: **# class attribute** species = "bird"

 # instance attribute def __init__(self, name,

age):

self.name = name

self.age = age

 # instantiate the Parrot class blu = Parrot("Blu", 10)

woo = Parrot("Woo", 15) **# access the class attributes**

print("Blu is a {}".format(blu.__class__.species))

print("Woo is also a

{}".format(woo.__class__.species))

 # access the instance attributes print("{} is {} years

old".format(blu.name, blu.age)) print("{} is {} years

old".format(woo.name, woo.age))

Output

Blu is a bird

Woo is also a bird

Blu is 10 years old

Woo is 15 years old

Methods

Methods are functions defined inside the body of a class. They are used to define the behaviors of an object.

Example 2 : Creating Methods in Python class

Parrot:

 # instance attributes def

__init_(self, name, age):

self.name = name

self.age = age

 # instance method

def sing(self, song):

return "{} sings {}".format(self.name, song)=

def dance(self):

return "{} is now dancing".format(self.name)

 # instantiate the object blu

= Parrot("Blu", 10) **# call**

our instance methods

print(blu.sing('"Happy"'))

print(blu.dance())

Output

Blu sings 'Happy'

Blu is now dancing

In the above program, we define two methods i.e `sing()` and `dance()`. These are called instance methods because they are called on an instance object i.e `blu`.

Inheritance

Inheritance is a way of creating a new class for using details of an existing class without modifying it. The newly formed class is a derived class (or child class). Similarly, the existing class is a base class (or parent class).

Example 3: Use of Inheritance in Python

 # parent class class Bird:

def__init_(self):

print("Bird is ready") def

whoisThis(self):

print("Bird") def

swim(self): print("Swim

faster") **# child class** class

Penguin(Bird): def

__init__(self): #

callsuper() function

super()._init_()

print("Penguin is ready")

def whoisThis(self):

print("Penguin") def

run(self): print("Run

faster") peggy =

Penguin()

peggy.whoisThis()

peggy.swim() peggy.run()

Output

Bird is ready

Penguin is ready

Penguin

Swim faster

Run faster

In the above program, we created two classes i.e. Bird (parent class) and Penguin (child class). The child class inherits the functions of parent class. We can see this from the swim() method.

Again, the child class modified the behavior of the parent class. We can see this from the whoisThis() method. Furthermore, we extend the functions of the parent class, by creating a new run() method.

Additionally, we use the `super()` function inside the `__init__()` method. This allows us to run the `__init__()` method of the parent class inside the child class.

Encapsulation

Using OOP in Python, we can restrict access to methods and variables. This prevents data from direct modification which is called encapsulation. In Python, we denote private attributes using underscore as the prefix i.e single `_` or double `__`.

E class Computer:

def___init__(self):

self._maxprice = 900

def sell(self):

print("Selling Price: {}".format(self._maxprice))

def setMaxPrice(self, price): self._maxprice =

price c = Computer()

c.sell()

change the price

c._maxprice = 1000

c.sell()

using setter function c.setMaxPrice(1000)

c.sell()

Output

Selling Price: 900

Selling Price: 900

Selling Price: 1000xample 4: Data Encapsulation in Python In

the above program, we defined a `Computer` class.

We used `__init__()` method to store the maximum selling price of `Computer`. Here, notice the code

c.__maxprice = 1000

Here, we have tried to modify the value of `__maxprice` outside of the class. However, since `__maxprice` is a private variable, this modification is not seen on the output.

As shown, to change the value, we have to use a setter function i.e setMaxPrice() which takes price as a parameter.

Polymorphism

Polymorphism is an ability (in OOP) to use a common interface for multiple forms (data types).

Suppose, we need to color a shape, there are multiple shape options (rectangle, square, circle). However, we could use the same method to color any shape. This concept is called Polymorphism.

Example 5: Using Polymorphism in Python

```
class Parrot:     def fly(self):
      print("Parrot can fly")
def swim(self):
      print("Parrot can't swim")
 class Penguin:     def fly(self):
       print("Penguin can't fly")
def swim(self):
print("Penguin can swim")
# common interface
def  flying_test(bird):
bird.fly() #instantiate
objects blu = Parrot()

peggy = Penguin()


# passing the object
flying_test(blu)
flying_test(peggy)
```

Output

Parrot can fly

Penguin can't fly

Key Points to Remember:

Object-Oriented Programming makes the program easy to understand as well as efficient.

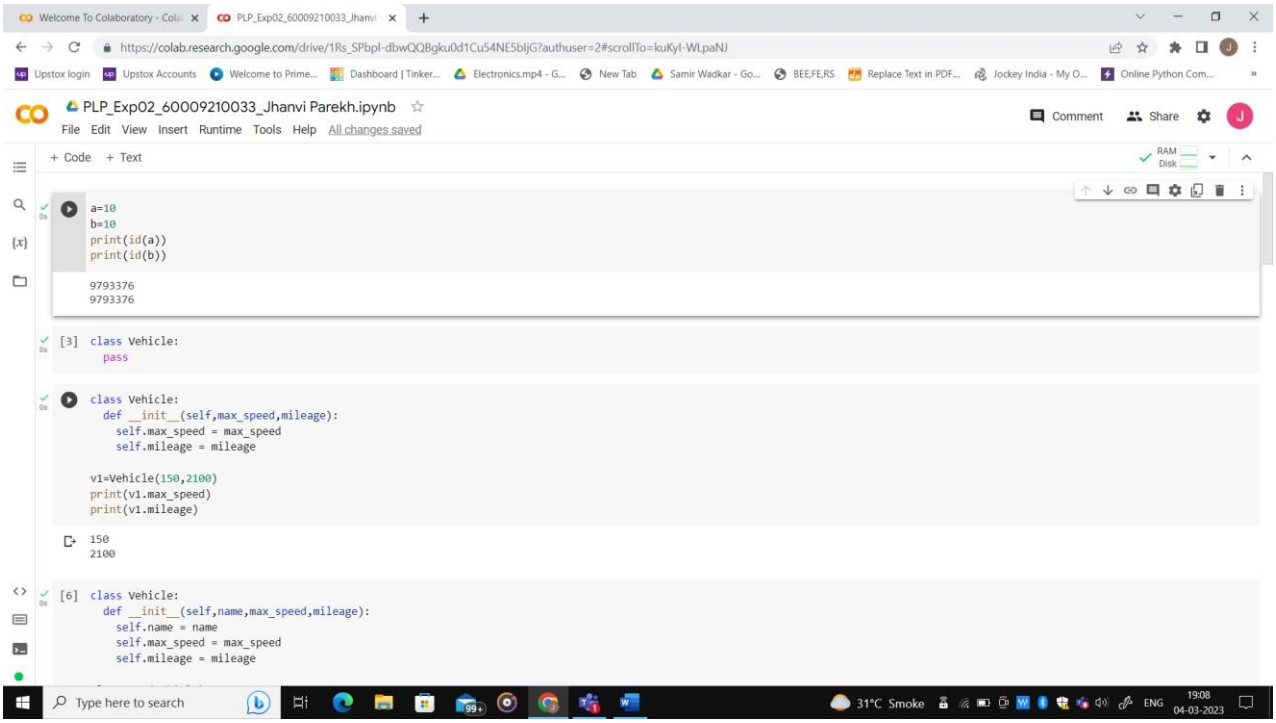
Since the class is sharable, the code can be reused.

Data is safe and secure with data abstraction.

Polymorphism allows the same interface for different objects, so programmers can write efficient code.

Lab Assignments to complete in this session Exercise 1: Implement following programs using Python Inheritance

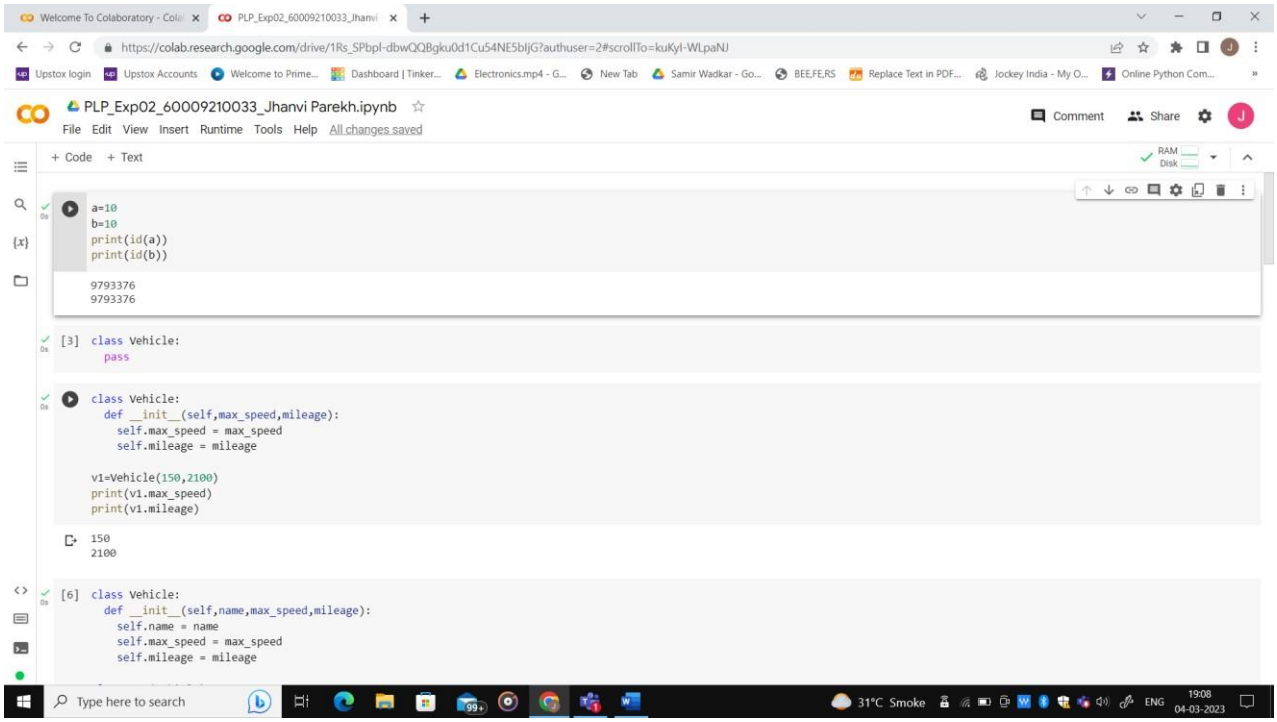
1. Create a Vehicle class without any variables and methods



Department of Computer Science and Engineering (Data Science)

2. Create a Class with instance attributes

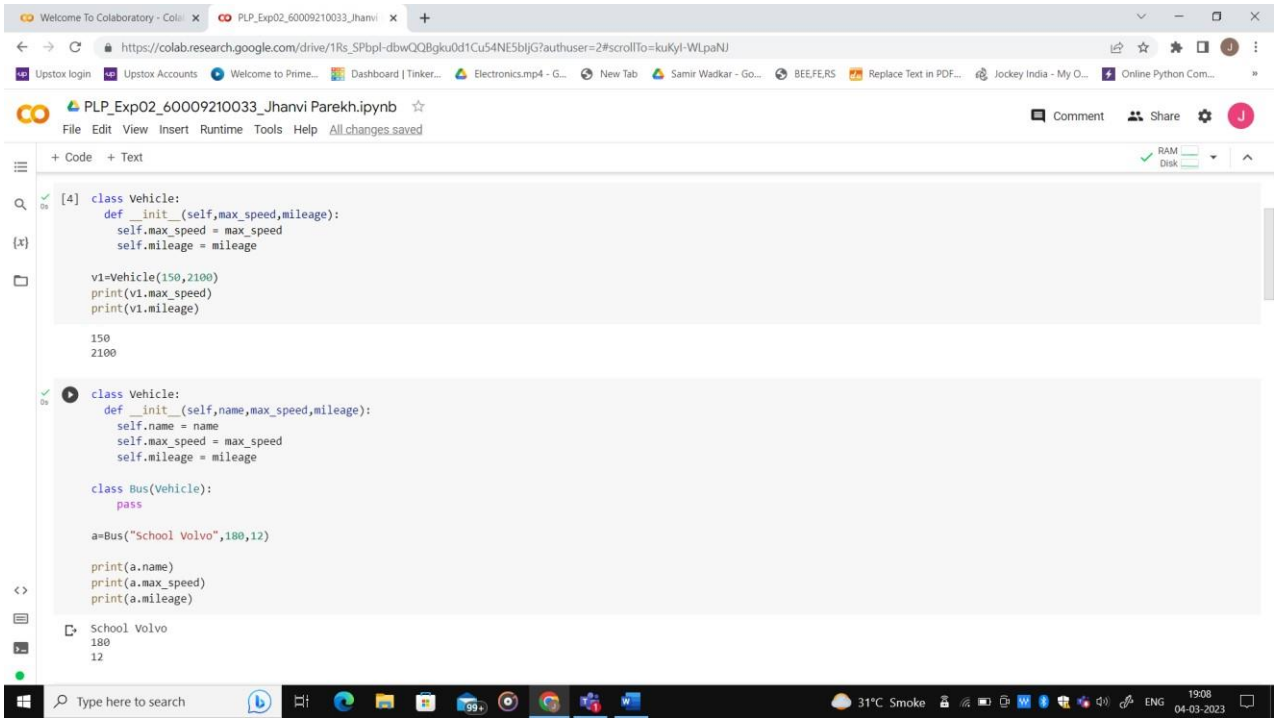
Write a Python program to create a Vehicle class with max_speed and mileage instance attributes.



3. Create a child class Bus that will inherit all of the variables and methods of the Vehicleclass

class Vehicle: def __init_(self, name, max_speed, mileage):

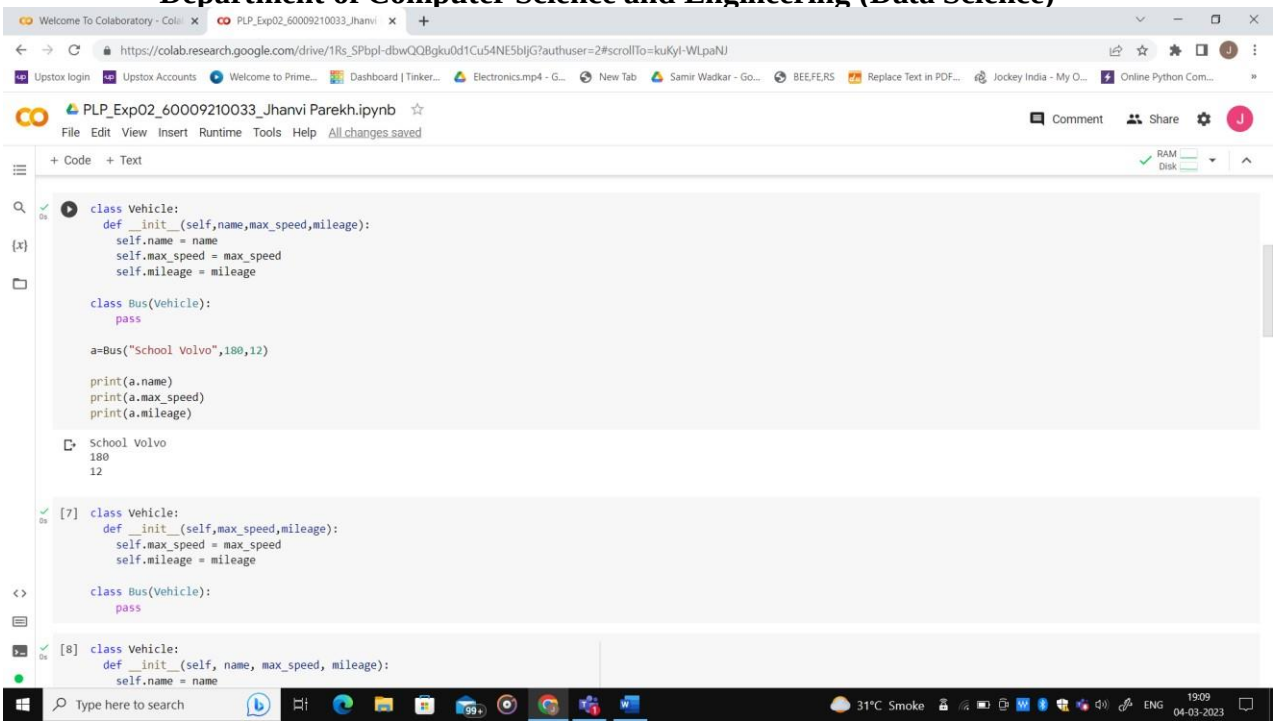
self.name = name self.max_speed = max_speed self.mileage = mileage



4.Create a Bus object that will inherit all of the variables and methods of the parent Vehicle class and display it.

Expected Output:

Vehicle Name: School Volvo Speed: 180 Mileage: 12



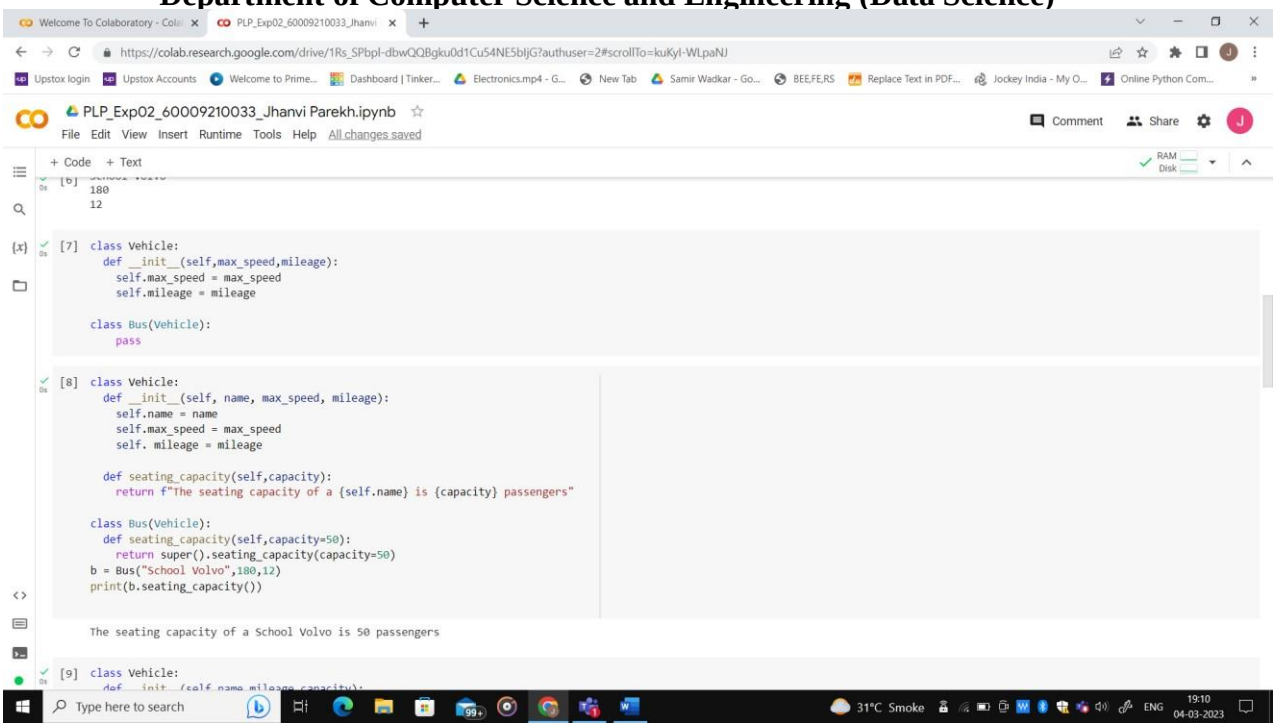
5. Create a Bus class that inherits from the Vehicle class. Give the capacity argument of Bus.seating_capacity() a default value of 50.

Use the following code for your parent Vehicle class.

class Vehicle: def __init_(self, name, max_speed, mileage): self.name = name self.max_speed = max_speed self. mileage = mileage

def seating_capacity(self, capacity): return f"The seating capacity of a {self.name} is {capacity} passengers" **Expected Output**:

```
The seating capacity of a bus is 50 passengers
```



6. Class Inheritance

Given:

Create a **Bus** child class that inherits from the Vehicle class. The default fare charge of any vehicle is **seating capacity * 100**. If Vehicle is **Bus** instance, we need to add an extra 10% on full fare as a maintenance charge. So total fare for bus instance will become the **final amount = total fare + 10% of the total fare.**

Note: The bus seating capacity is **50**. so the final fare amount should be **5500.** You need to override the fare() method of a Vehicle class in Bus class.

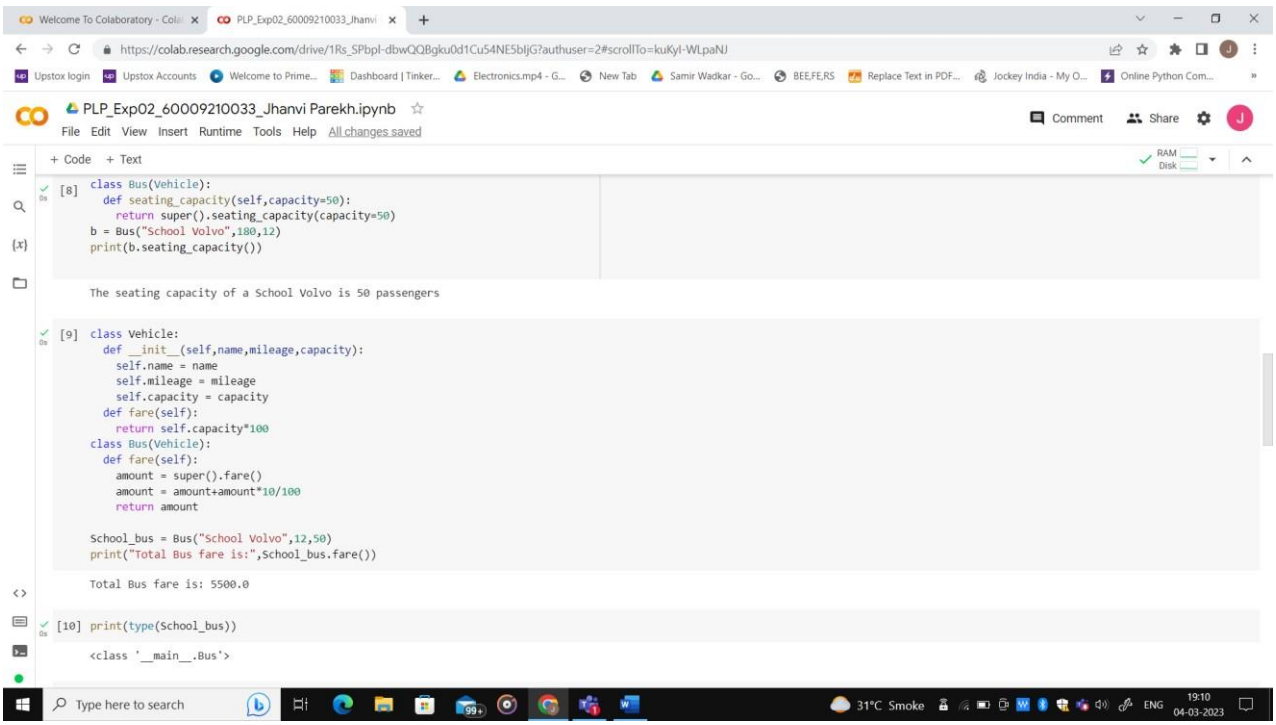
Use the following code for your parent Vehicle class. We need to access the parent class from inside a method of a child class.

```
class Vehicle:    def __init (self, name, mileage, capacity):
```

```
        self.name = name
self.mileage = mileage
        self.capacity = capacity

    def fare(self):        return
self.capacity * 100

class Bus(Vehicle):
    pass

School_bus = Bus("School Volvo", 12, 50)
print("Total Bus fare is:", School_bus.fare())
```

Expected Output:

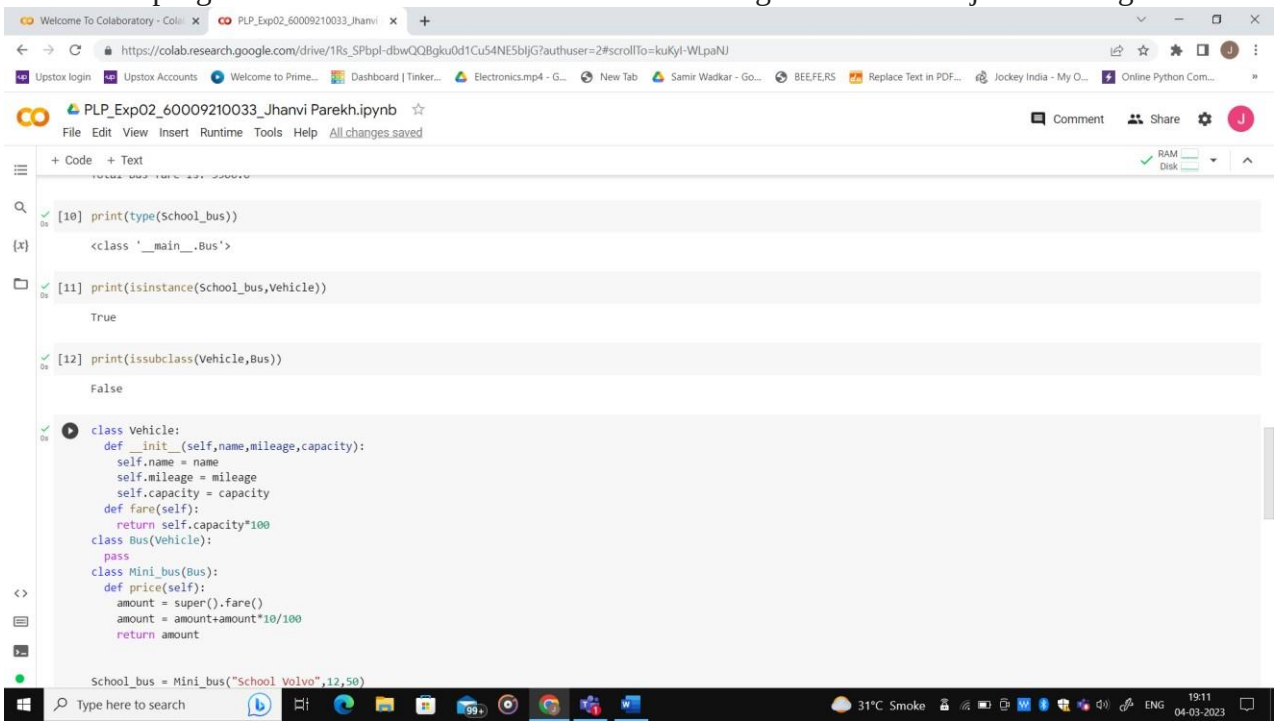
```
Total Bus fare is: 5500.0
```



7.Check type of an object

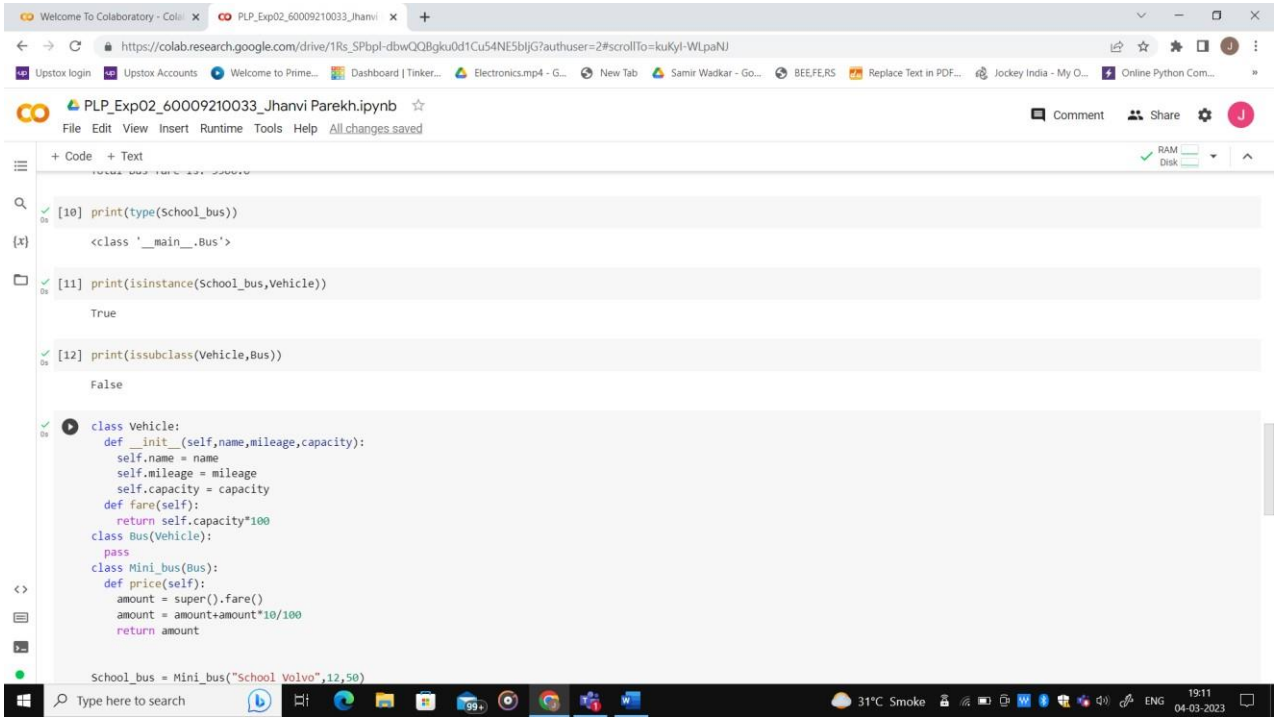
Department of Computer Science and Engineering (Data Science)

Write a program to determine which class a given Bus object belongs to.

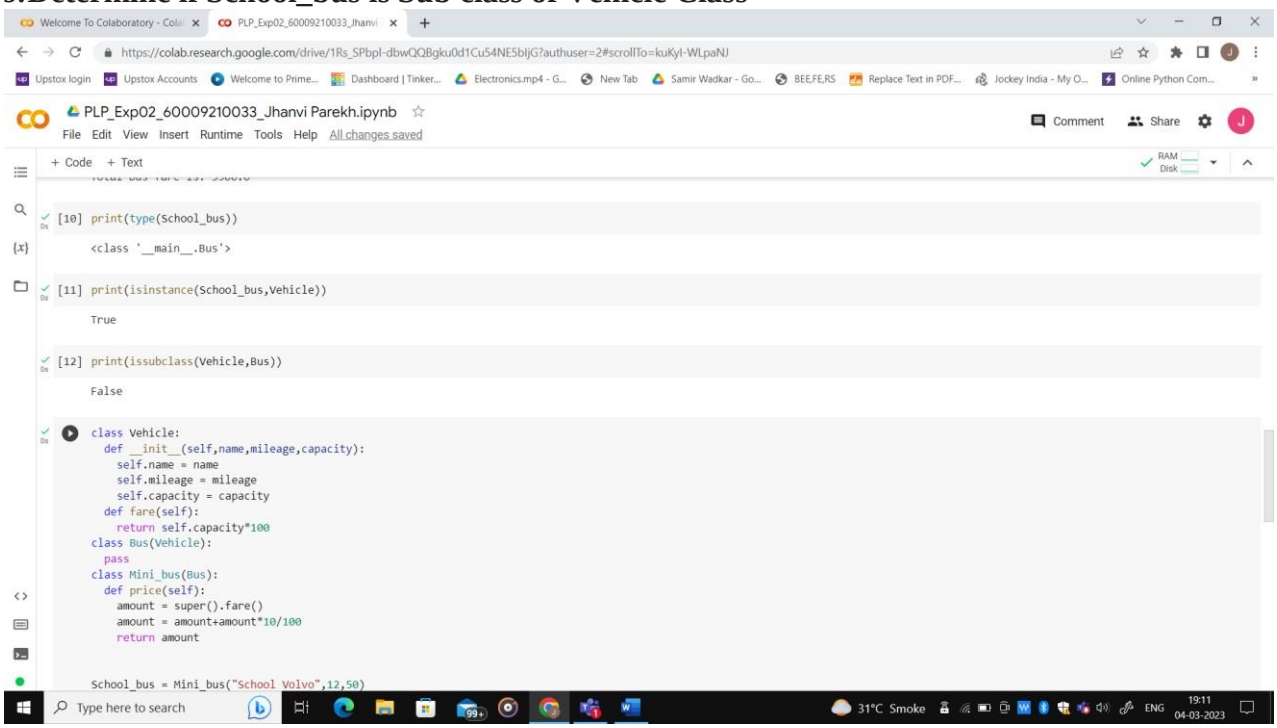


Department of Computer Science and Engineering (Data Science)

8. Determine if School_bus is also an instance of the Vehicle class



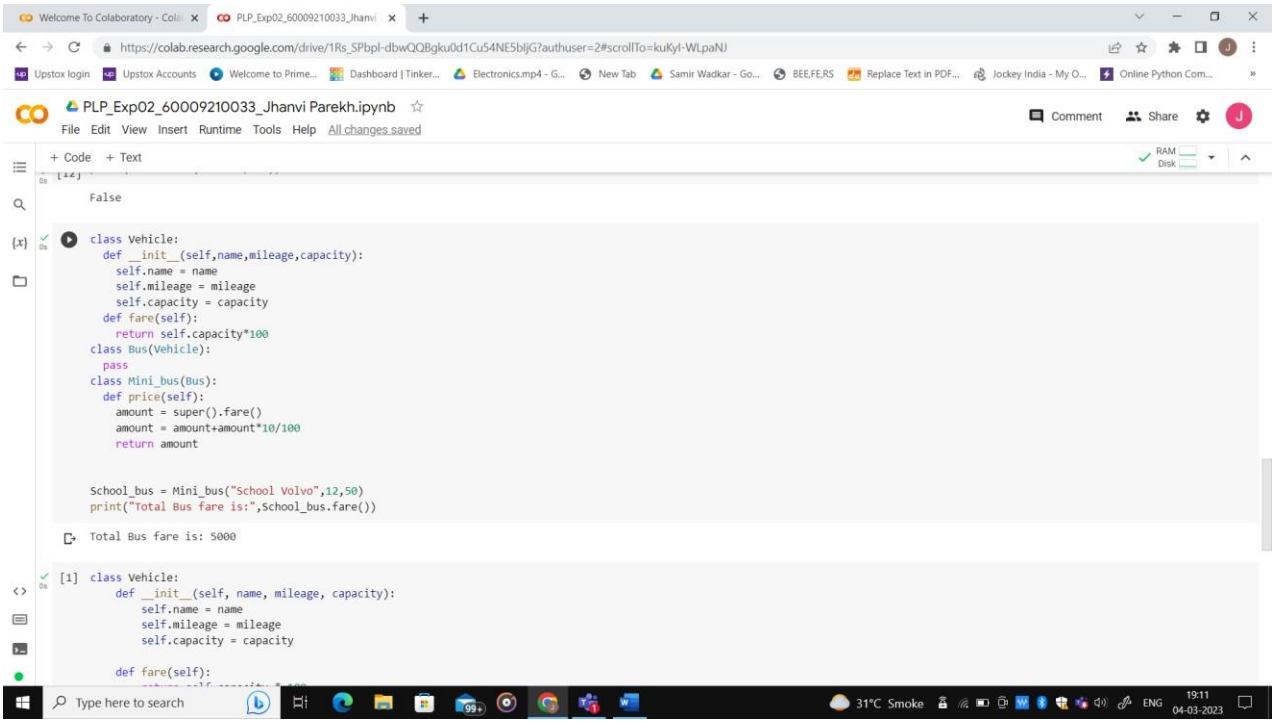
9. Determine if School_bus is Sub class of Vehicle Class



10. Create a child class for Bus Class named Mini bus inheriting Bus class and priceattribute and Print Price Method (Multilevel Inheritance)

Use the following code for your parent Vehicle class. We need to access the parent class from inside a method of a child class.

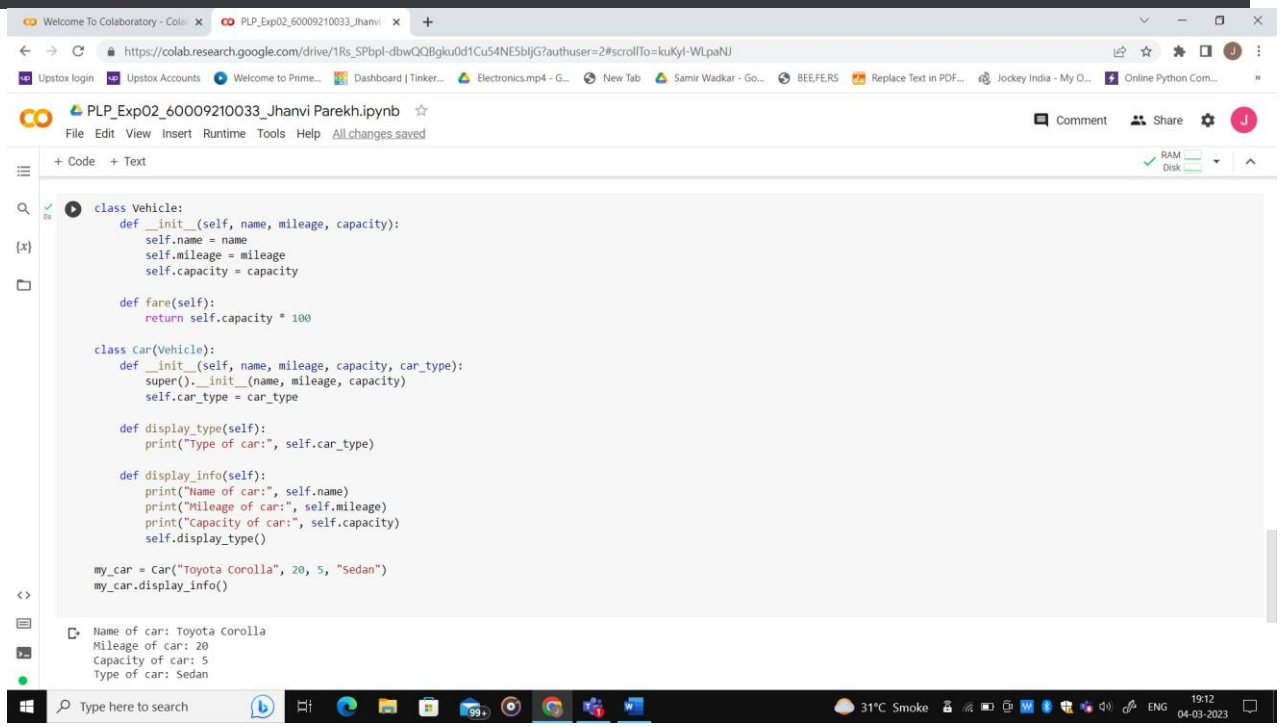
```
class Vehicle:    def__init_(self, name, mileage,
capacity):
      self.name = name       self.mileage
= mileage
      self.capacity = capacity

  def fare(self):       return
self.capacity * 100



  class Bus(Vehicle):
     pass

  School_bus = Bus("School Volvo", 12, 50) print("Total
  Bus fare is:", School_bus.fare())
```



11. Create a car class inheriting Vehicle class and add type and Print attribute and displayType method method for the same

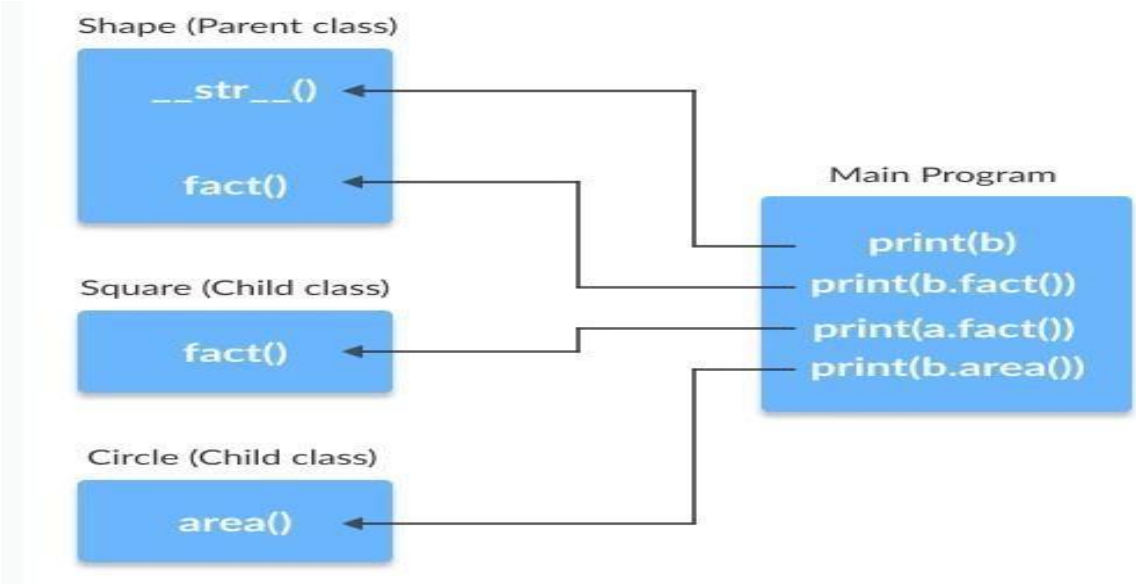
Use the following code for your parent Vehicle class. We need to access the parent class from inside a method of a child class.

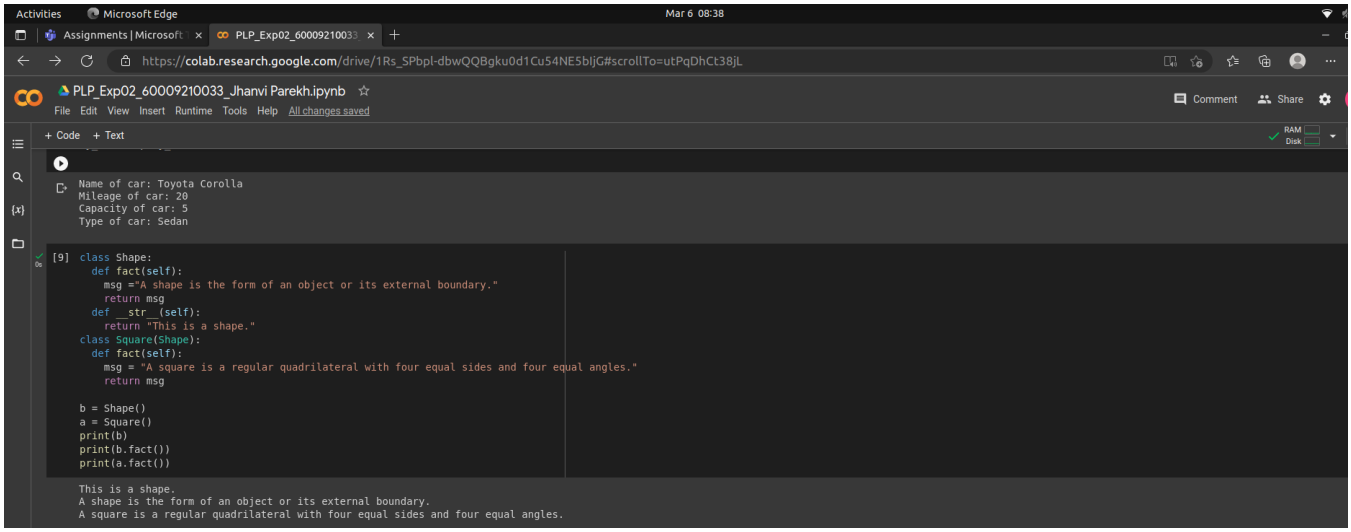
```
class Vehicle:def init (self, name, mileage, capacity):self.name = name self.mileage = mileage
self.capacity = capacity

def fare(self):return
self.capacity * 100

class Bus(Vehicle):
pass

School_bus = Bus("School Volvo", 12, 50)
print("Total Bus fare is:", School_bus.fare())
```



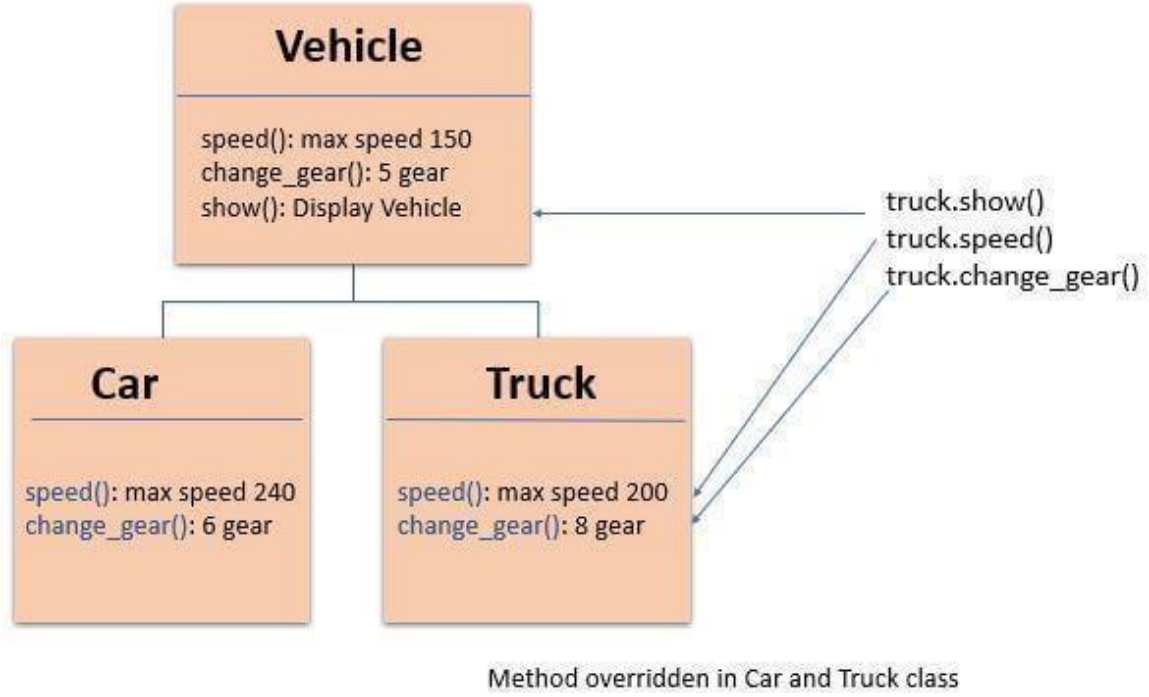
Department of Computer Science and Engineering (Data Science)

12. Write a program in python to Implement Method Overriding considering the following

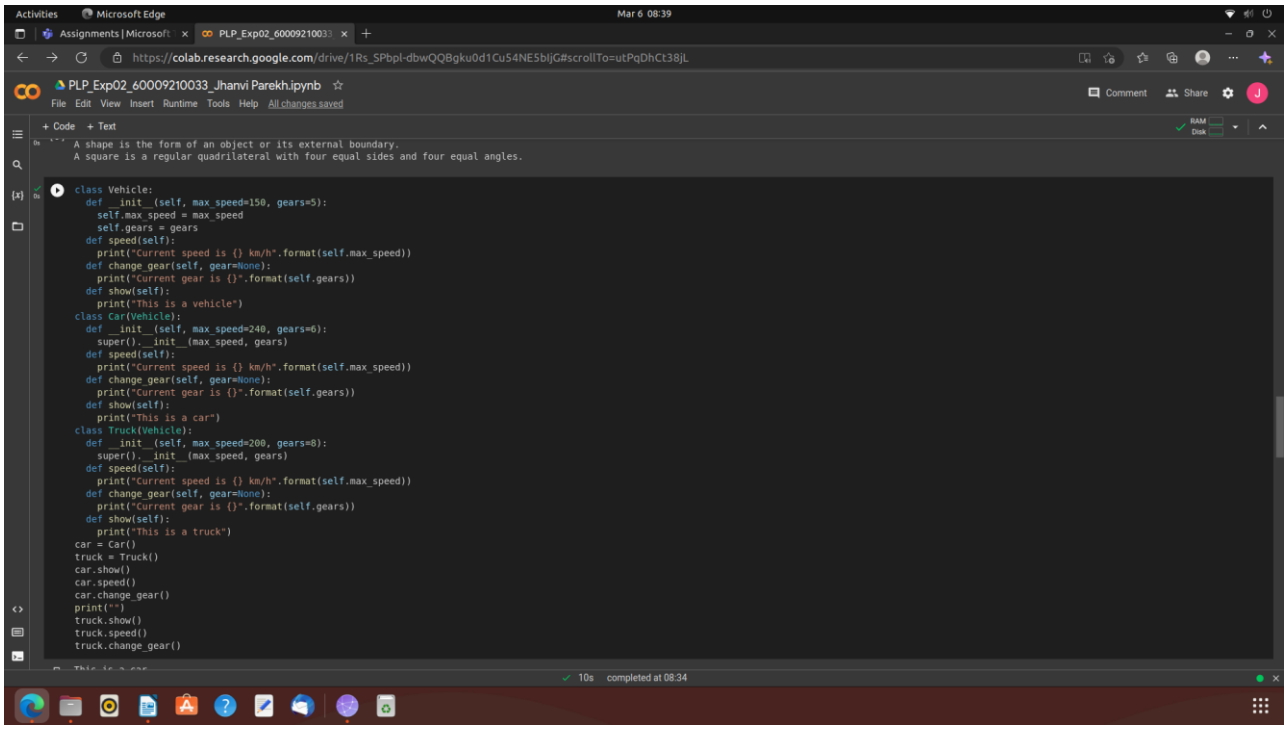


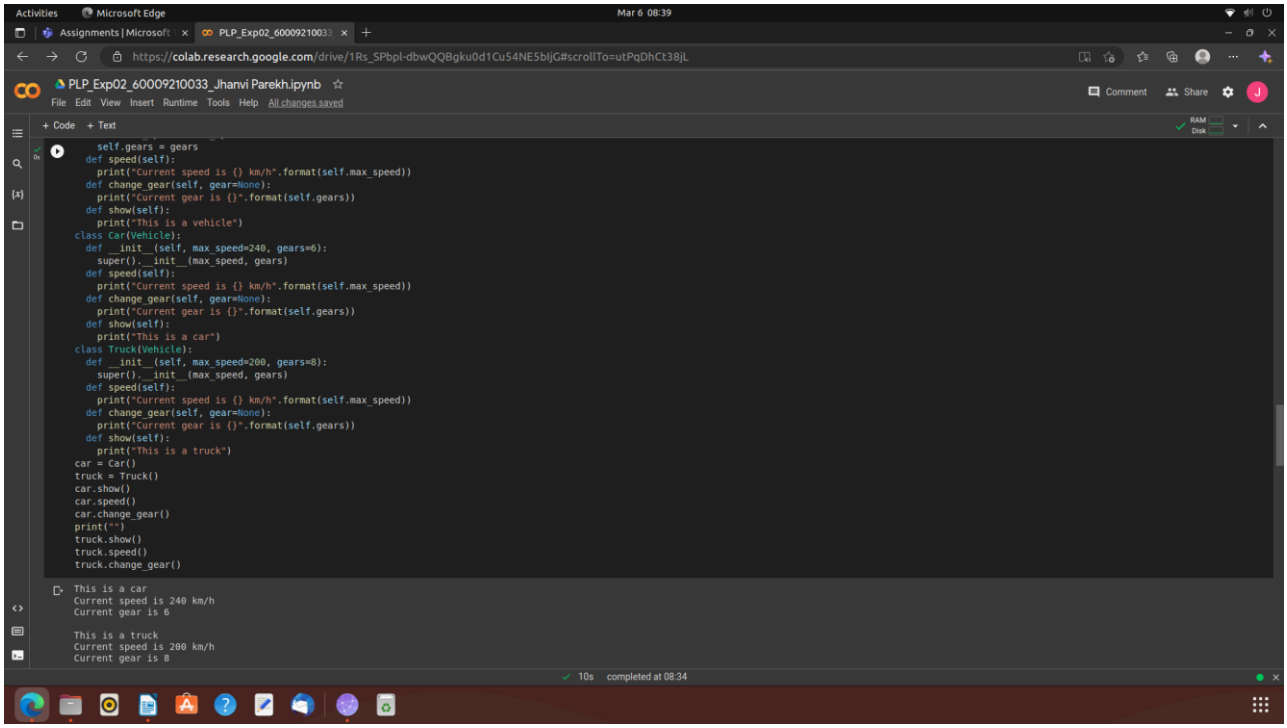


13. Write a program in python to Implement Method Overriding considering the followingclasses, methods and Variables

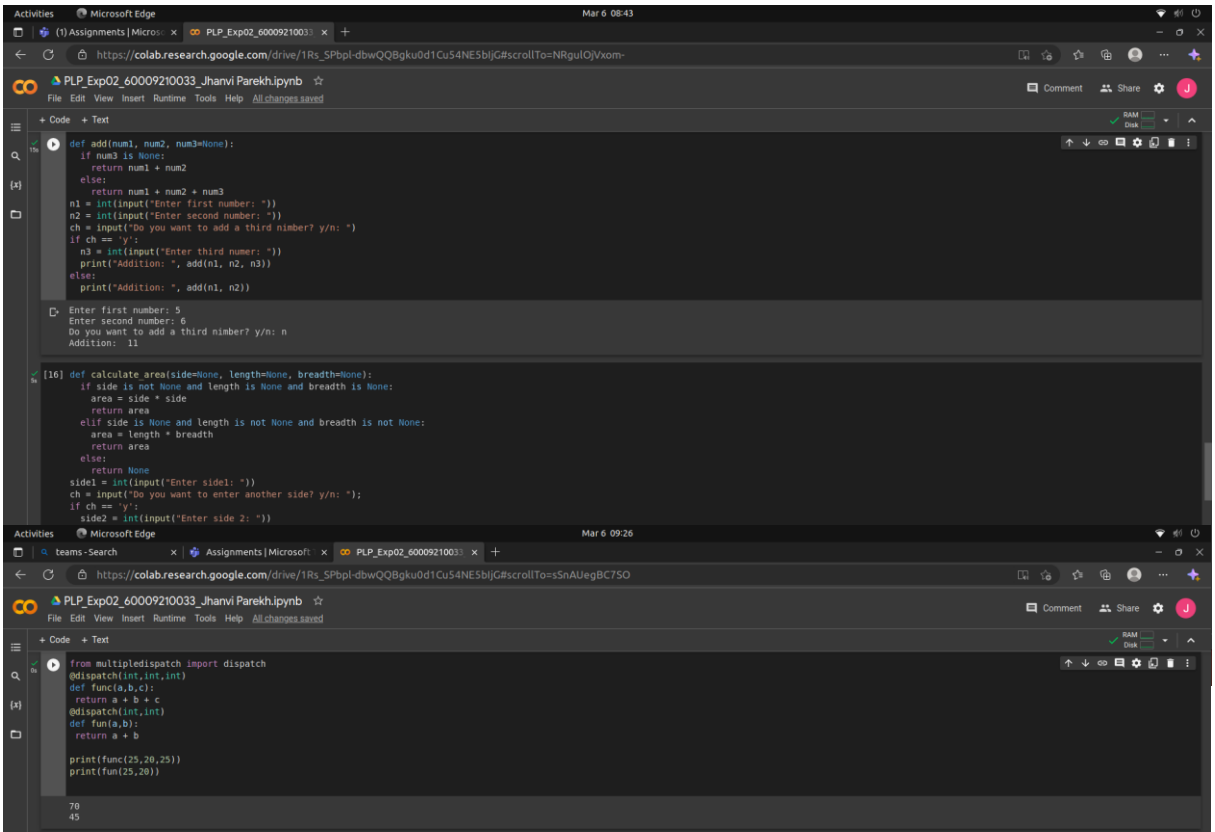


Method overridden in Car and Truck class

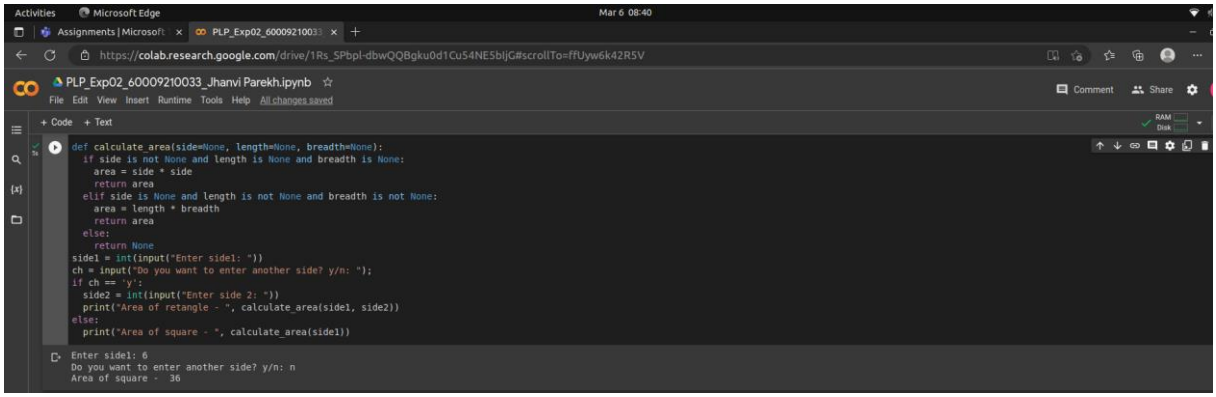




14. Write a program in python to demonstrate how method overloading is achieved. Create a function to perform addition operation. If 2 no's are given it should perform addition of 2 no's otherwise it should add 3 nos.



15. Write a program in python to perform method overloading. Create a function to calculate area of given shape. If 1 side is given calculate area of square, if Length and breadth is given calculate the area of Rectangle



16. Write a program to Implement Abstract class using Following problem statement The Employee class represents an employee, either full-time or hourly. the Employee class should be an abstract class because there're only full-time employees and hourly employees, no general employees exist.

The Employee class should have a property that returns the full name of an employee. In addition, it should have a method that calculates salary. The method for calculating salary should be an abstract method.

The Full_time_Employee class inherits from the Employee class. It'll provide the implementation for the get_salary () method.

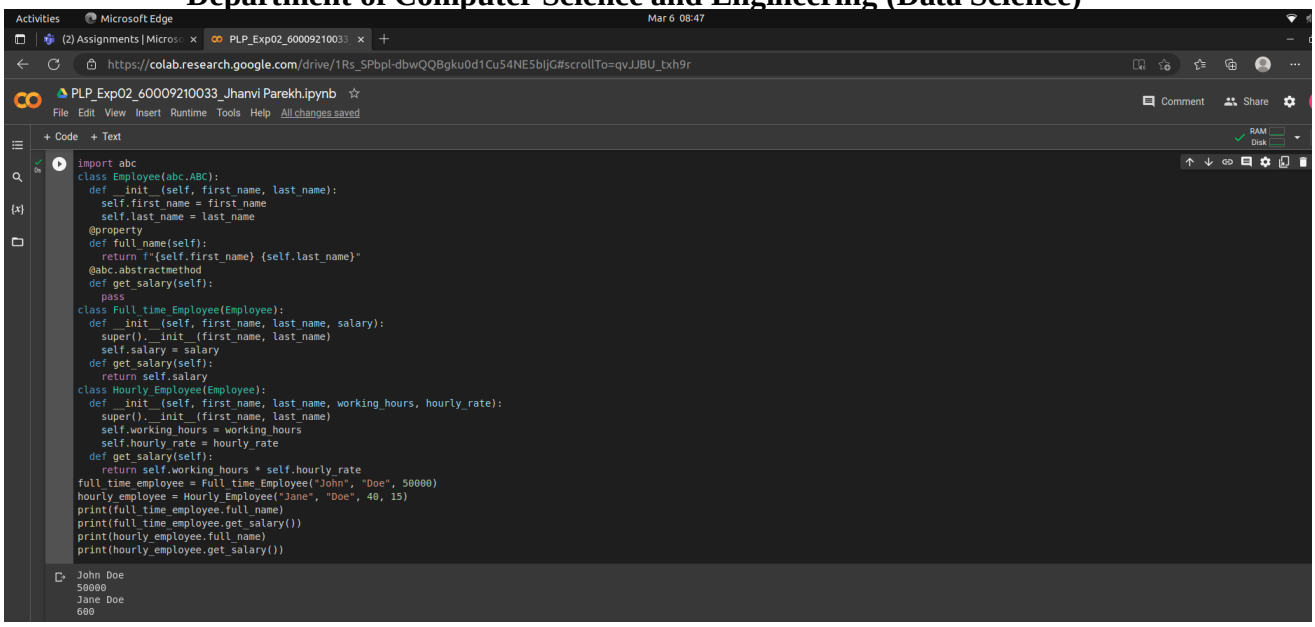
Since full-time employees get fixed salaries, you can initialize the salary in the constructor of the class.

The HourlyEmployee also inherits from the Employee class.

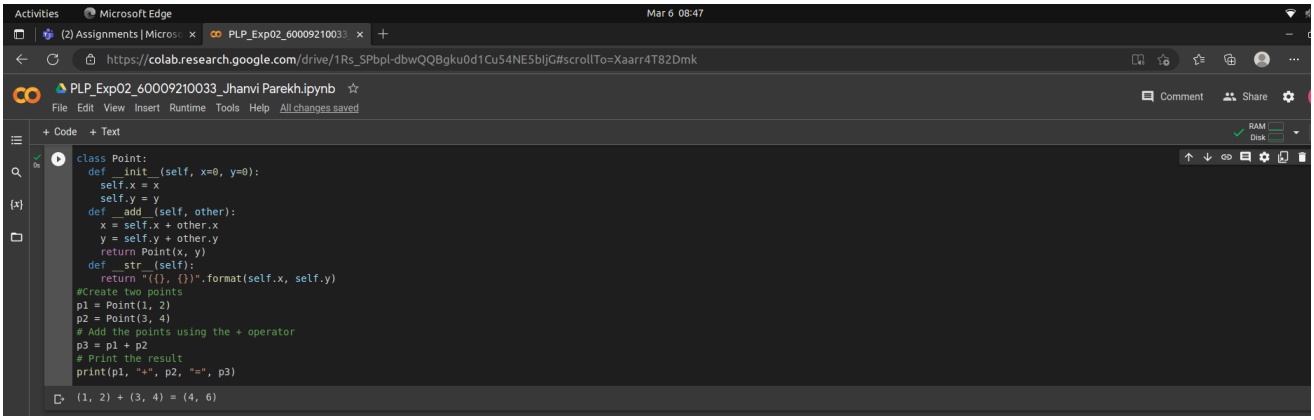
However, hourly employees get paid by working hours and their rates. Therefore, you can initialize this information in the constructor of the class.

To calculate the salary for the hourly employees, you multiply the working hours and rates.

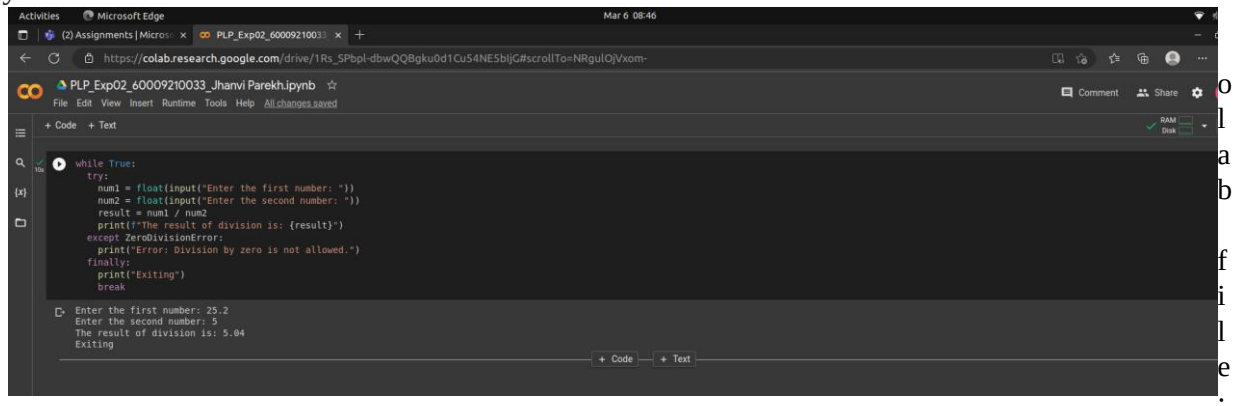
Department of Computer Science and Engineering (Data Science)



17. Write a Python Program to overload + operator to add Point (X1, Y1) and Point (X2, Y2)



18. Write a Python Program to perform division of Two Numbers, take both the nos from users from user, use TRY, EXCEPT and FINALLY block to raise an Exception when Diving Number by Zero.



Colab file:

https://colab.research.google.com/drive/1Rs_SPbpl-dbwQQBgku0d1Cu54NE5bIjG?usp=sharing