Experiment 5

Name : Smayan Kulkarni, SAP ID : 60009230142 , Roll Number : D100, Division : D2-2

1. Create a Vehicle class without any variables and methods

In [1]:
```
class Vehicle:
    pass
vehicle_obj = Vehicle()
print(vehicle_obj)
```

```
<__main__.Vehicle object at 0x7b9ba1530f50>
```

2. Create a Class with instance attributes

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

In [2]:
```
class Vehicle:
    def __init__(self, max_speed, mileage):
        self.max_speed = max_speed
        self.mileage = mileage
car = Vehicle(150, 20000)
print(f"Max Speed: {car.max_speed}")
print(f"Mileage: {car.mileage}")
```

```
Max Speed: 150
Mileage: 20000
```

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

of the Vehicle class class Vehicle: def init (self, name, max_speed, mileage): self.name = name self.max_speed = max_speed self.mileage = mileage

In [3]:
```
class Vehicle:
    def __init__(self, name, max_speed, mileage):
        self.name = name
        self.max_speed = max_speed
        self.mileage = mileage

class Bus(Vehicle):
    pass
bus = Bus("School Bus", 80, 12000)
print(f"Vehicle Name: {bus.name}")
print(f"Max Speed: {bus.max_speed}")
print(f"Mileage: {bus.mileage}")
```

```
Vehicle Name: School Bus
Max Speed: 80
Mileage: 12000
```

4. Create a Bus object that will inherit all of the variables and methods of

the parent Vehicle class and display it.

In [4]:
```
class Vehicle:
    def __init__(self, name, max_speed, mileage):
        self.name = name
        self.max_speed = max_speed
        self.mileage = mileage
class Bus(Vehicle):
    pass
bus = Bus("School Volvo", 180, 12)
print(f"Vehicle Name: {bus.name} Speed: {bus.max_speed} Mileage: {bus
```

```
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"

In [5]:
```
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}
class Bus(Vehicle):
    def seating_capacity(self, capacity=50):
     return super().seating_capacity(capacity)
bus = Bus("bus", 180, 12)
print(bus.seating_capacity())
```

```
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.

In [6]:
```
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):
    def fare(self):
        total_fare = super().fare()
        maintenance_charge = total_fare * 0.10
        final_fare = total_fare + maintenance_charge
        return final_fare
School_bus = Bus("School Volvo", 12, 50)
print("Total Bus fare is:", School_bus.fare())
```

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

7. Check type of an object

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

In [7]:
```
class Vehicle:
    def __init__(self, name, mileage, capacity):
        self.name = name
        self.mileage = mileage
        self.capacity = capacity
class Bus(Vehicle):
    def __init__(self, name, mileage, capacity, bus_type):
        super().__init__(name, mileage, capacity)
        self.bus_type = bus_type
school_bus = Bus("School Volvo", 12, 50, "School Bus")
print("The object school_bus is of type:", type(school_bus))
print("The object school_bus belongs to class:", school_bus.__class__
```

```
The object school_bus is of type: <class '__main__.Bus'>
The object school_bus belongs to class: Bus
```

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

In [8]:
```
if isinstance(school_bus, Vehicle):
    print("school_bus is an instance of the Vehicle class.")
else:
    print("school_bus is not an instance of the Vehicle class.")
```

```
school_bus is an instance of the Vehicle class.
```

9. Determine if School_bus is Sub class of Vehicle Class

In [9]:
```
if isinstance(school_bus, Bus):
    print("school_bus is an instance of the Bus class.")
else:
    print("school_bus is not an instance of the Bus class.")
if isinstance(school_bus, Vehicle):
    print("school_bus is an instance of the Vehicle class.")
else:
    print("school_bus is not an instance of the Vehicle class.")
if issubclass(Bus, Vehicle):
    print("Bus is a subclass of Vehicle.")
else:
    print("Bus is not a subclass of Vehicle.")
```

```
school_bus is an instance of the Bus class.
school_bus is an instance of the Vehicle class.
Bus is a subclass of Vehicle.
```

10. Create a child class for Bus Class named Mini bus inheriting Bus class

and price attribute and Print Price Method (Multilevel Inheritance)

In [10]:
```
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):
    def __init__(self, name, mileage, capacity, price):
        super().__init__(name, mileage, capacity)
        self.price = price
    def print_price(self):
            print("The price of the Bus is:", self.price)
class MiniBus(Bus):
    def __init__(self, name, mileage, capacity, price):
        super().__init__(name, mileage, capacity, price)
mini_bus = MiniBus("Mini Bus", 10, 30, 15000)

print("Total Bus fare is:", mini_bus.fare())
mini_bus.print_price()
```

```
Total Bus fare is: 3000
The price of the Bus is: 15000
```

11. Create a car class inheriting Vehicle class and add type and Print

attribute and display Type method method for the same

In [11]:
```
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 Car(Vehicle):
    def __init__(self, name, mileage, capacity, car_type):
        super().__init__(name, mileage, capacity)
        self.car_type = car_type
    def display_type(self):
        print(f"Vehicle Type: {self.car_type}")
my_car = Car("Toyota Camry", 15, 5, "Sedan")
print("Total Car fare is:", my_car.fare())
my_car.display_type()
```

```
Total Car fare is: 500
Vehicle Type: Sedan
```

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.

In [12]:
```
class Addition:
    def add(self, a, b, c=0):
     return a + b + c
obj = Addition()
print("Sum of two numbers:", obj.add(5, 10))
print("Sum of three numbers:", obj.add(5, 10, 15))
```

```
Sum of two numbers: 15
Sum of three numbers: 30
```

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.

In [13]:
```
class Shape:
    def area(self, side, breadth=0):
        if breadth == 0:
            return side * side
        else:
            return side * breadth
shape = Shape()
print("Area of square:", shape.area(5))
print("Area of rectangle:", shape.area(10, 5))
```

```
Area of square: 25
Area of rectangle: 50
```

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 ofIn [ ]:In [2]:In [3]: the class. To calculate the salary for the hourly employees, you multiply the working hours and rates.

In [14]:
```
from abc import ABC, abstractmethod
class Employee(ABC):
    def __init__(self, first_name, last_name):
        self.first_name = first_name
        self.last_name = last_name
    @property
    def full_name(self):
        return f"{self.first_name} {self.last_name}"
    @abstractmethod
    def get_salary(self):
        pass
class FullTimeEmployee(Employee):
    def __init__(self, first_name, last_name, salary):
        super().__init__(first_name, last_name)
        self.salary = salary
    def get_salary(self):
        return self.salary
class HourlyEmployee(Employee):
    def __init__(self, first_name, last_name, hourly_rate, hours_work
        super().__init__(first_name, last_name)
        self.hourly_rate = hourly_rate
        self.hours_worked = hours_worked
    def get_salary(self):
        return self.hourly_rate * self.hours_worked
full_time_emp = FullTimeEmployee("Tony", "Stark", 50000)
hourly_emp = HourlyEmployee("Evan", "Smith", 20, 160)
print(f"Full-time employee {full_time_emp.full_name} salary: {full_ti
print(f"Hourly employee {hourly_emp.full_name} salary: {hourly_emp.ge
```

```
Full-time employee Tony Stark salary: <bound method FullTimeEmployee
.get_salary of <__main__.FullTimeEmployee object at 0x7b9ba15336b0>>
Hourly employee Evan Smith salary: 3200
```

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

and Point (X2, Y2)

In [15]:
```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def __add__(self, other):
        return Point(self.x + other.x, self.y + other.y)
    def __str__(self):
        return f"({self.x}, {self.y})"
point1 = Point(2, 3)
point2 = Point(4, 5)
result = point1 + point2
print("Result of addition:", result)
```

```
Result of addition: (6, 8)
```

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.

In [16]:
```
def divide_nos():
    try:
        num1 = float(input("Enter the first number: "))
        num2 = float(input("Enter the second number: "))
        result = num1 / num2
    except ZeroDivisionError:
        print("Error: Cannot divide by zero.")
    finally:
        print("Execution completed.")

divide_nos()
```

```
Error: Cannot divide by zero.
Execution completed.
```