Exercise 7

1. Create a Python class Bank to simulate basic banking operations. The class should have the following features:

- Class Variables:
  - `total_balance`: A class variable to keep track of the total money available in the bank.
- Instance Variables
  - `name`: The name of the account holder.
  - `balance`: The current balance of the account holder.
- `__init__` method to initialize a new account holder with their initial amount
- `deposit` method to add amount to user's account as well as banks available amount
- `check_balance` method to check current balance.
- `withdraw` method to withdraw amount.

> **NOTE**: For withdrawl the user should have enough money in their account. Also you should make sure that the bank has enough total balance to give amount. If withdrawl cannot be conducted, give user appropriate message.

```
print("Bank's total balance:", Bank.total_balance)

ram_account = Bank('Ram', 5000)
ram_account.deposit(2000)
ram_account.withdraw(1000)
ram_account.check_balance()
print("Bank's total balance:", Bank.total_balance)

sita_account = Bank('Sita', 3000)
sita_account.withdraw(7000)
print("Bank's total balance:", Bank.total_balance)

ram_account.withdraw(1000)
print("Bank's total balance:", Bank.total_balance)
```

2. Define a class Car with attributes make, model, and year.

- Use `__init__` method to initialize the class.
- Create an instance of the class and print its attributes.

- Have a class variable to store the number of car instance created. Create 10 cars and check the count.
- Add a method start_engine() to the Car class that prints "Engine started". Call this method on an instance of the Car class.
- Create a class ElectricCar that inherits from Car and adds an attribute battery_capacity. Add a method charge() that prints "Charging the battery".
- Overload the == operator for the Car class using the `__eq__` method to compare two Car instances based on their attributes.
- Implement the `__str__` and `__repr__` methods in the Car class to provide a user-friendly and developer-friendly string representation of the object.

3. Create a Python class `Point` to represent a point in a 2D space with x and y coordinates. Implement the following methods to perform various operations:

- `__init__(self, x, y)`: Initializes a new point with the given x and y coordinates.
- `__add__(self, other)`: Adds two points by adding their respective x and y coordinates, returning a new Point instance.
- `__mul__(self, scalar)`: Multiplies the point's coordinates by a scalar, returning a new Point instance.
- `__eq__(self, other)`: Checks if two points are equal by comparing their x and y coordinates.
- `__lt__(self, other)`: Compares two points based on their distance from the origin (0, 0). Returns True if the current point is closer to the origin than the other point.
- `__gt__(self, other)`: Compares two points based on their distance from the origin (0, 0). Returns True if the current point is farther from the origin than the other point.
- `distance_from_origin(self)`: Returns the distance of the point from the origin (0, 0).
- I want to make my class have ability to do a+=b what can I do?
- I want to find unique elements in my class using set how can I create a set with different points. s = {Point(1,2), Point(2,3), Point(1, 2)}