Laboratory Activity No. 1

Introduction to Object-Oriented Programming

Course Code: CPE009B	**Program:** BSCPE
Course Title: Object-Oriented Programming	**Date Performed:29/8/2024**
Section: CPE21S1	**Date Submitted:29/8/2024**
Name: Jimenez, Christian Joros R.	**Instructor: Ms. Sayo**

1. Objective(s):

This activity aims to familiarize students with the concepts of Object-Oriented Programming

2. Intended Learning Outcomes (ILOs):

The students should be able to:
2.1 Identify the possible attributes and methods of a given object
2.2 Create a class using the Python language
2.3 Create and modify the instances and the attributes in the instance.

3. Discussion:

Object-Oriented Programming (OOP) is an approach to programming that views the world and systems as consisting of objects that relate and interact with each other. This involves identifying the characteristics that describe the object which are known as the Attributes of the object. Furthermore, it also deals with identifying the possible capabilities or actions that an object is able to do which are called Methods.

An object is simply composed of Attributes and Methods wherein Attributes are variables that hold the information describing the object and Methods are functions which allow the object to perform its defined capabilities/actions. A UML Class Diagram is used to formally represent the collection of Attributes and Methods.

An example is given below considering a simple banking system.

Accounts ATM
+ account_number: int + serial_number: int
+ account_firstname: string
+ account_lastname: string
+ current_balance: float
+ address: string + deposit(account: Accounts, amount: int) + email: string + widthdraw(account: Accounts, amount: int) + update_address(new_address: string) + check_currentbalance(account: Accounts) + update_email(new_email: string) + view_transactionsummary()

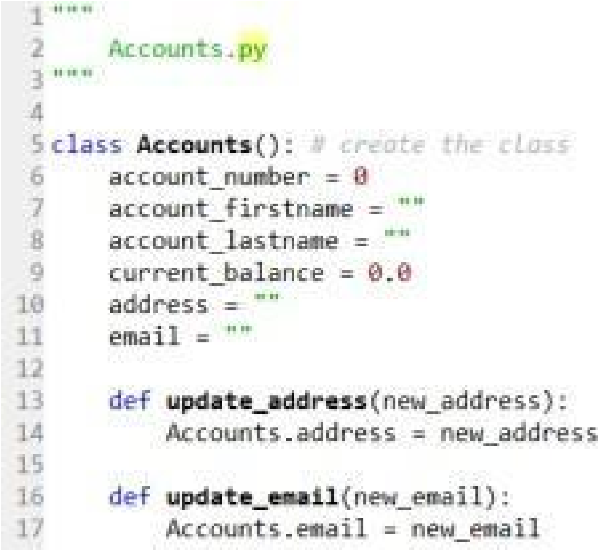
4. Materials and Equipment:

Desktop Computer with Anaconda
Python Windows Operating System

5. Procedure:

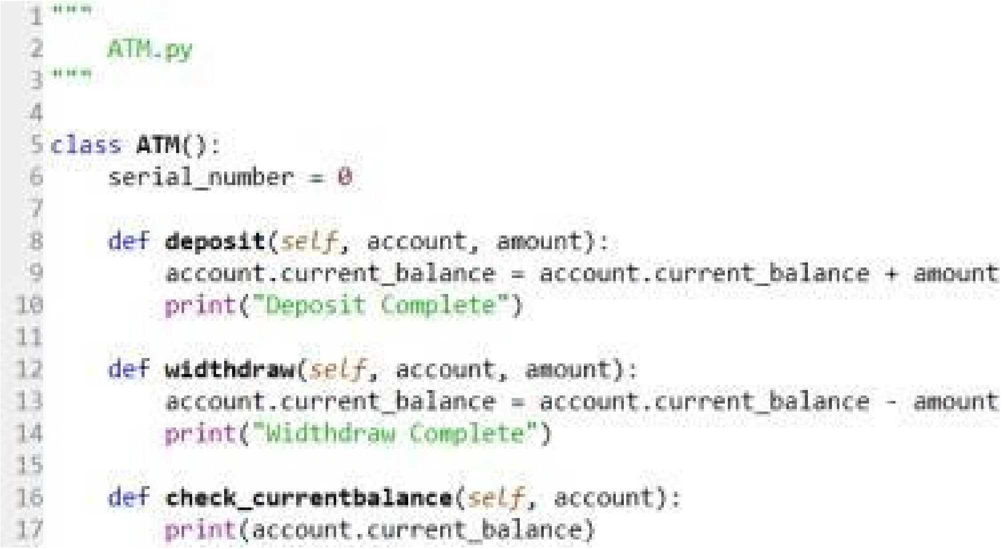
Creating Classes

1. Create a folder named **OOPIntro_LastName**
2. Create a Python file inside the **OOPIntro_LastName** folder named **Accounts.py** and copy the code shown below:



```python
"""
    Accounts.py
"""

class Accounts(): # create the class
    account_number = 0
    account_firstname = ""
    account_lastname = ""
    current_balance = 0.0
    address = ""
    email = ""

    def update_address(new_address):
        Accounts.address = new_address

    def update_email(new_email):
        Accounts.email = new_email
```

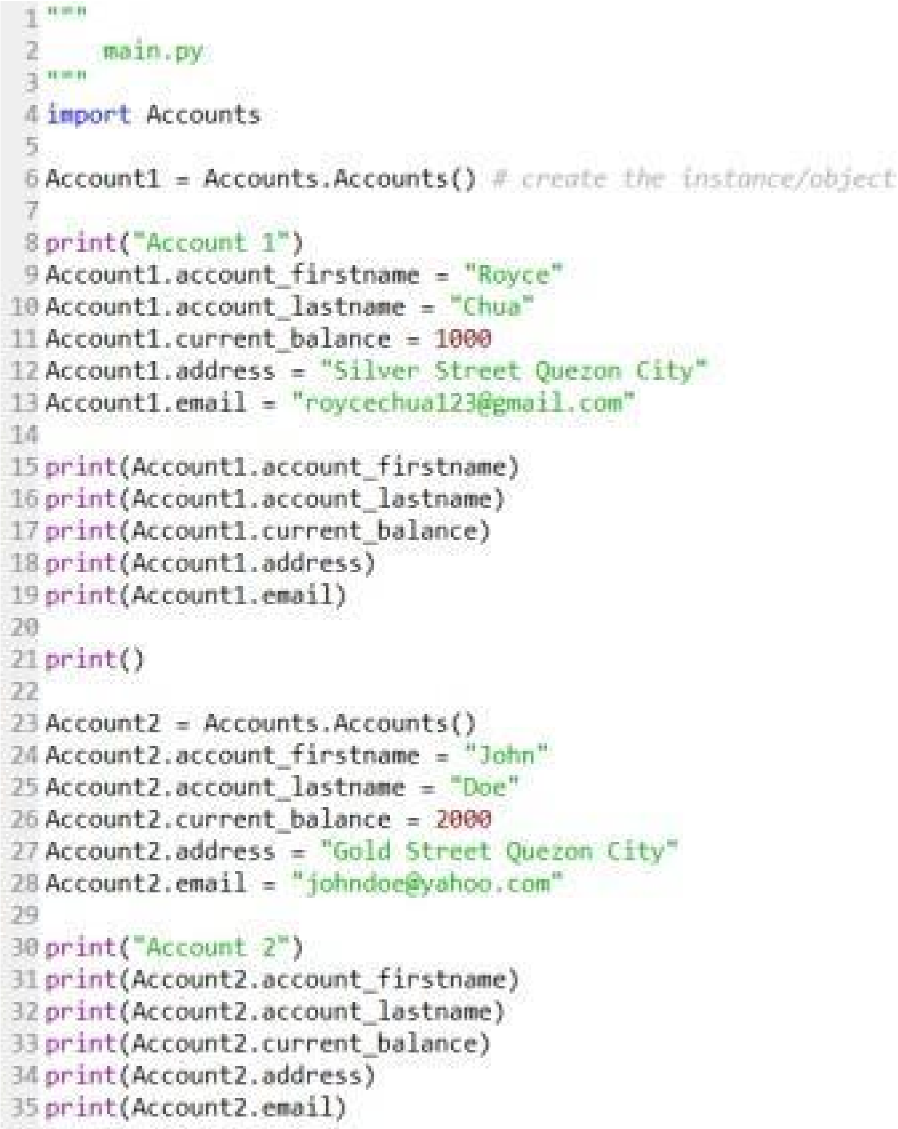
3. Modify the Accounts.py and add ***self,*** before the new_address and new_email.
4. Create a new file named ATM.py and copy the code shown below:



```python
"""
    ATM.py
"""

class ATM():
    serial_number = 0

    def deposit(self, account, amount):
        account.current_balance = account.current_balance + amount
        print("Deposit Complete")

    def widthdraw(self, account, amount):
        account.current_balance = account.current_balance - amount
        print("Widthdraw Complete")

    def check_currentbalance(self, account):
        print(account.current_balance)
```

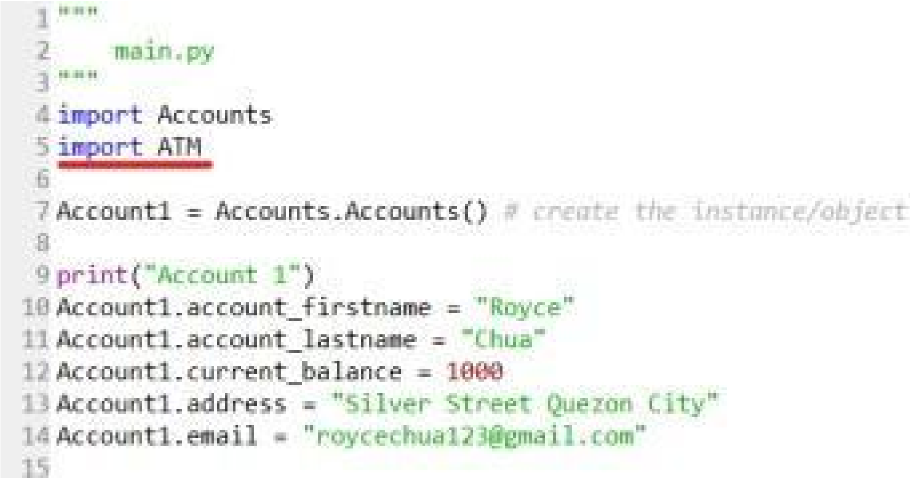
Creating Instances of Classes

5. Create a new file named main.py and copy the code shown below:



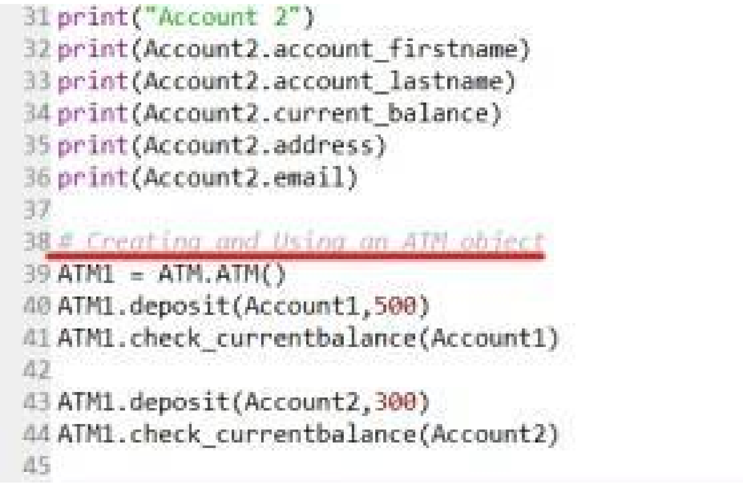
```
1 """
2     main.py
3 """
4 import Accounts
5
6 Account1 = Accounts.Accounts() # create the instance/object
7
8 print("Account 1")
9 Account1.account_firstname = "Royce"
10 Account1.account_lastname = "Chua"
11 Account1.current_balance = 1000
12 Account1.address = "Silver Street Quezon City"
13 Account1.email = "roycechua123@gmail.com"
14
15 print(Account1.account_firstname)
16 print(Account1.account_lastname)
17 print(Account1.current_balance)
18 print(Account1.address)
19 print(Account1.email)
20
21 print()
22
23 Account2 = Accounts.Accounts()
24 Account2.account_firstname = "John"
25 Account2.account_lastname = "Doe"
26 Account2.current_balance = 2000
27 Account2.address = "Gold Street Quezon City"
28 Account2.email = "johndoe@yahoo.com"
29
30 print("Account 2")
31 print(Account2.account_firstname)
32 print(Account2.account_lastname)
33 print(Account2.current_balance)
34 print(Account2.address)
35 print(Account2.email)
```

6. Run the main.py program and observe the output. Observe the variables names account_firstname, account_lastname as well as other variables being used in the Account1 and Account2. 7. Modify the main.py program and add the code underlined in red.



```
1 """
2     main.py
3 """
4 import Accounts
5 import ATM
6
7 Account1 = Accounts.Accounts() # create the instance/object
8
9 print("Account 1")
10 Account1.account_firstname = "Royce"
11 Account1.account_lastname = "Chua"
12 Account1.current_balance = 1000
13 Account1.address = "Silver Street Quezon City"
14 Account1.email = "roycechua123@gmail.com"
15
```

8. Modify the main.py program and add the code below line 38.

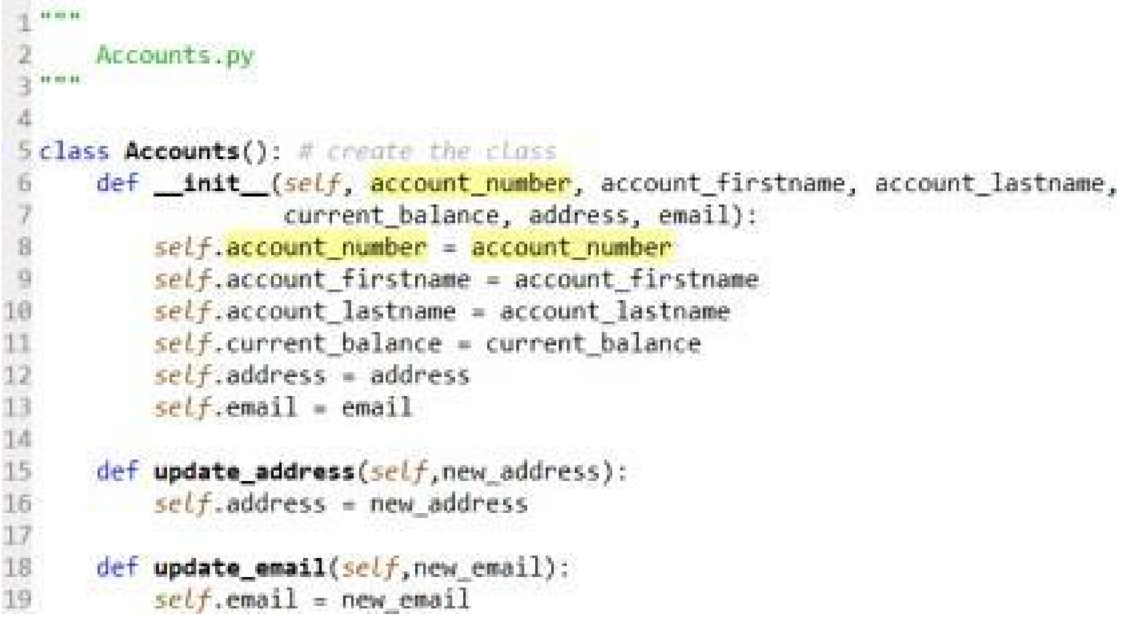


```
print("Account 2")
print(Account2.account_firstname)
print(Account2.account_lastname)
print(Account2.current_balance)
print(Account2.address)
print(Account2.email)

# Creating and Using an ATM object
ATM1 = ATM.ATM()
ATM1.deposit(Account1,500)
ATM1.check_currentbalance(Account1)

ATM1.deposit(Account2,300)
ATM1.check_currentbalance(Account2)

```

9. Run the main.py program.

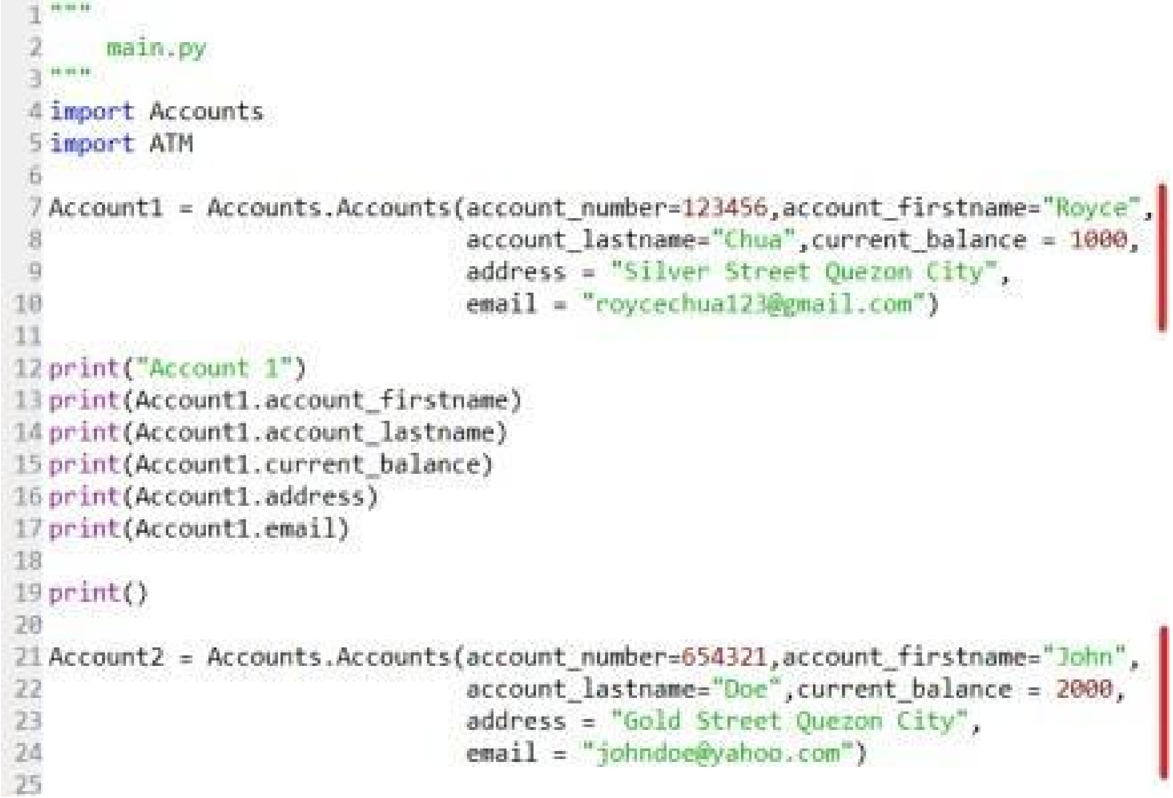
Create the Constructor in each Class

1. Modify the Accounts.py with the following code:
   Reminder: def _init_(): is also known as the constructor class



```
"""
    Accounts.py
"""

class Accounts(): # create the class
    def __init__(self, account_number, account_firstname, account_lastname,
                 current_balance, address, email):
        self.account_number = account_number
        self.account_firstname = account_firstname
        self.account_lastname = account_lastname
        self.current_balance = current_balance
        self.address = address
        self.email = email

    def update_address(self,new_address):
        self.address = new_address

    def update_email(self,new_email):
        self.email = new_email
```

2. Modify the main.py and change the following codes with the red line. Do not remove the other codes in the program.



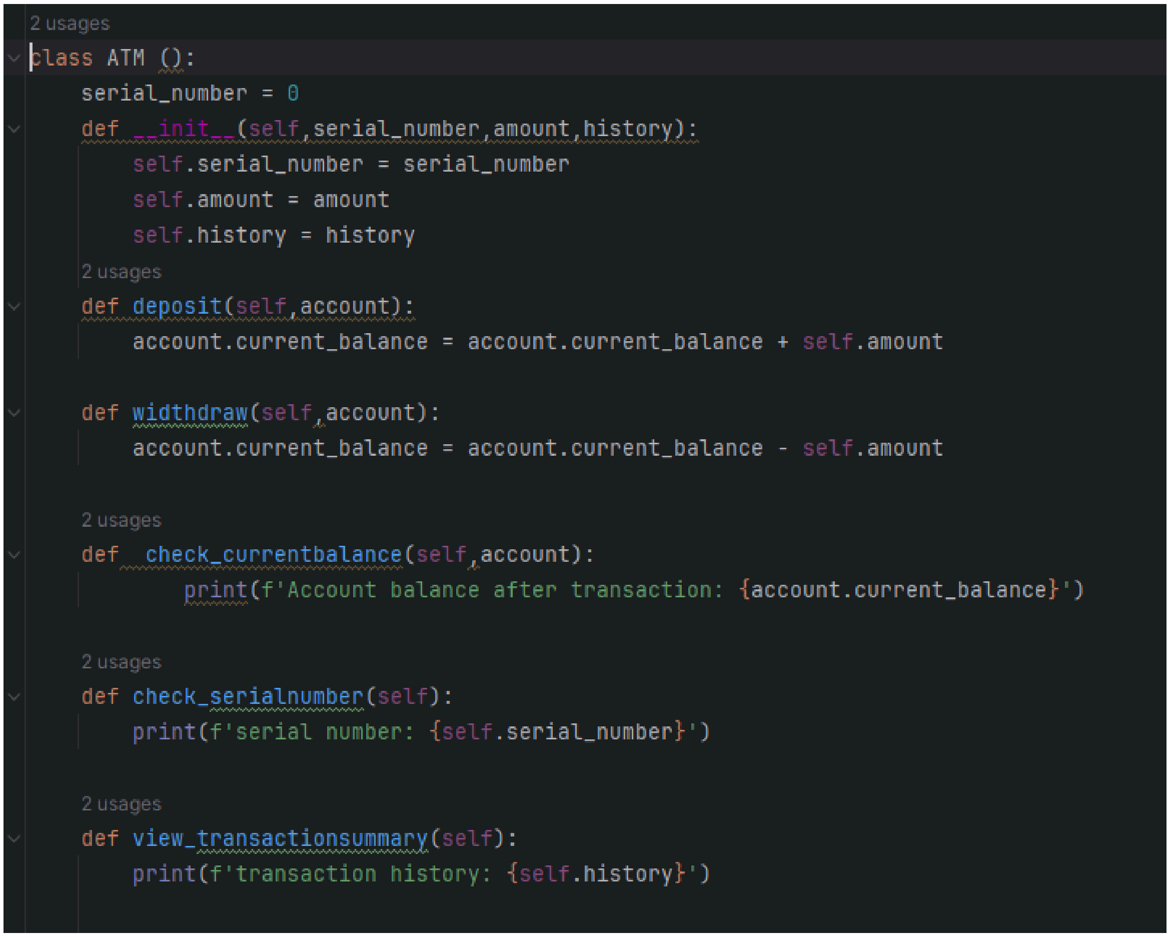
```
"""
    main.py
"""
import Accounts
import ATM

Account1 = Accounts.Accounts(account_number=123456,account_firstname="Royce",
                             account_lastname="Chua",current_balance = 1000,
                             address = "Silver Street Quezon City",
                             email = "roycechua123@gmail.com")

print("Account 1")
print(Account1.account_firstname)
print(Account1.account_lastname)
print(Account1.current_balance)
print(Account1.address)
print(Account1.email)

print()

Account2 = Accounts.Accounts(account_number=654321,account_firstname="John",
                             account_lastname="Doe",current_balance = 2000,
                             address = "Gold Street Quezon City",
                             email = "johndoe@yahoo.com")

```

3. Run the main.py program again and run the output.

6. Supplementary Activity:

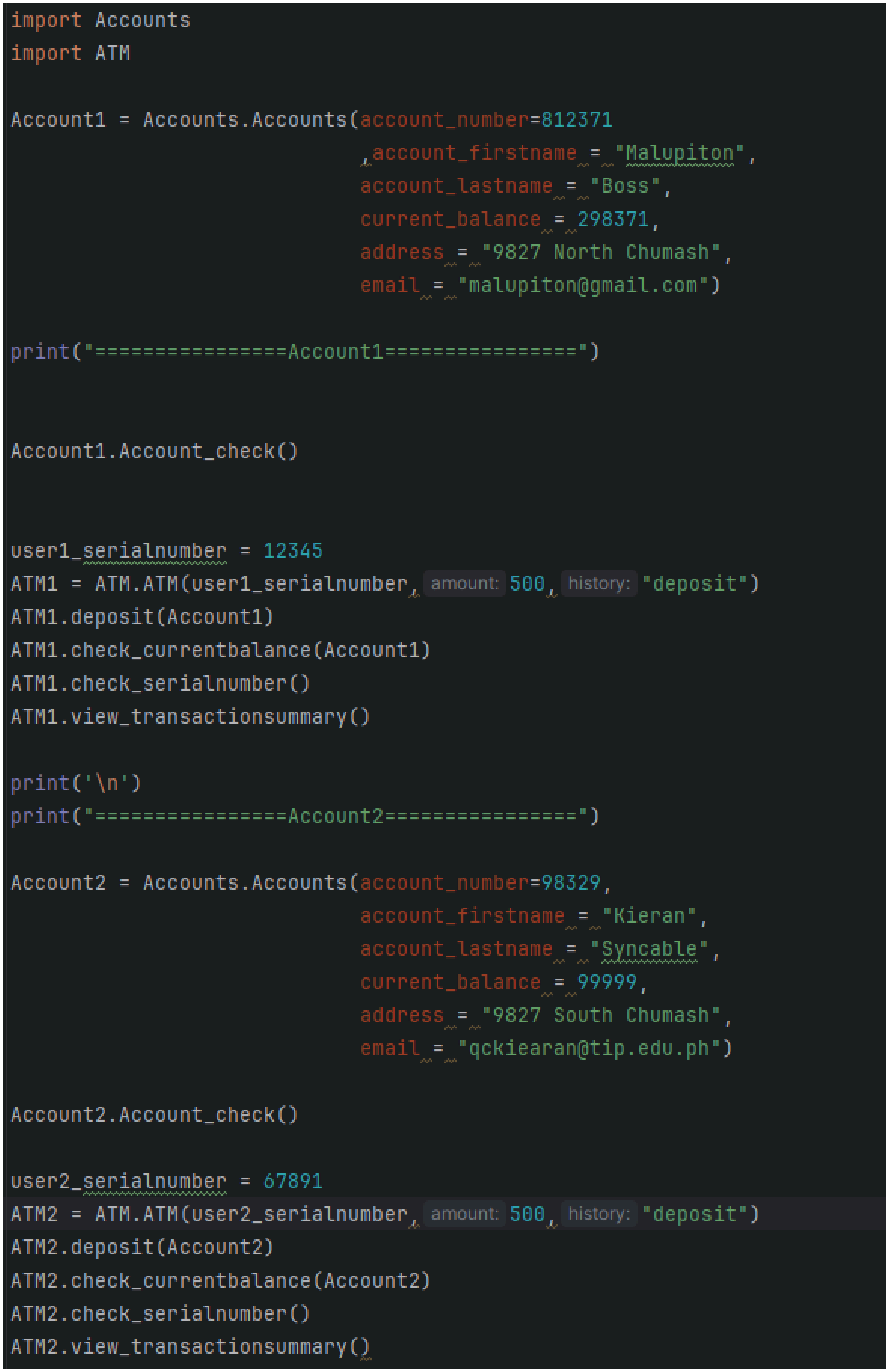
Tasks

1. Modify the ATM.py program and add the constructor function.

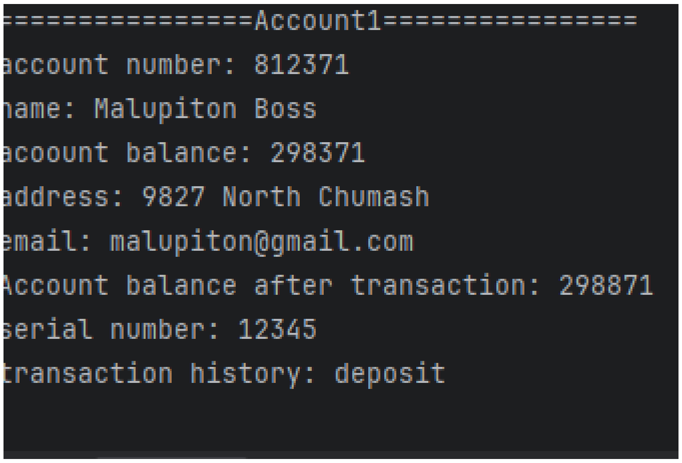


```
2 usages
class ATM ():
    serial_number = 0
    def __init__(self,serial_number,amount,history):
        self.serial_number = serial_number
        self.amount = amount
        self.history = history
    2 usages
    def deposit(self,account):
        account.current_balance = account.current_balance + self.amount

    def widthdraw(self,account):
        account.current_balance = account.current_balance - self.amount

    2 usages
    def  check_currentbalance(self,account):
            print(f'Account balance after transaction: {account.current_balance}')

    2 usages
    def check_serialnumber(self):
        print(f'serial number: {self.serial_number}')

    2 usages
    def view_transactionsummary(self):
        print(f'transaction history: {self.history}')
```

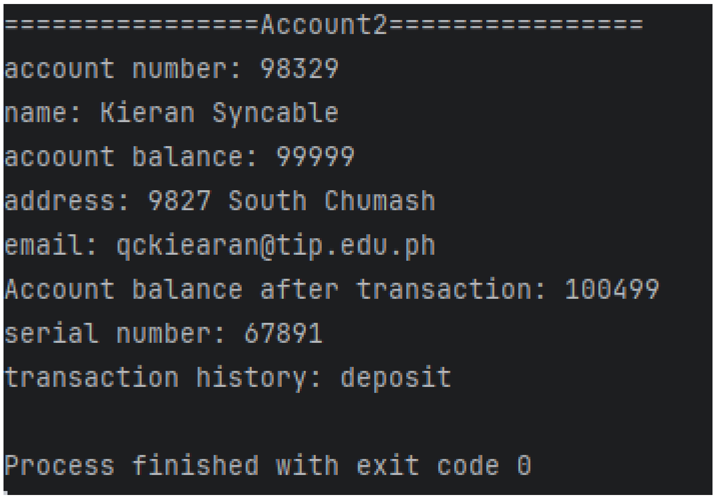
2. Modify the main.py program and initialize the ATM machine with any integer serial number combination and display the serial number at the end of the program.



```python
import Accounts
import ATM

Account1 = Accounts.Accounts(account_number=812371
                             ,account_firstname = "Malupiton",
                             account_lastname = "Boss",
                             current_balance = 298371,
                             address = "9827 North Chumash",
                             email = "malupiton@gmail.com")

print("================Account1================")


Account1.Account_check()


user1_serialnumber = 12345
ATM1 = ATM.ATM(user1_serialnumber, amount: 500, history: "deposit")
ATM1.deposit(Account1)
ATM1.check_currentbalance(Account1)
ATM1.check_serialnumber()
ATM1.view_transactionsummary()

print('\n')
print("================Account2================")

Account2 = Accounts.Accounts(account_number=98329,
                             account_firstname = "Kieran",
                             account_lastname = "Syncable",
                             current_balance = 99999,
                             address = "9827 South Chumash",
                             email = "qckiearan@tip.edu.ph")

Account2.Account_check()

user2_serialnumber = 67891
ATM2 = ATM.ATM(user2_serialnumber, amount: 500, history: "deposit")
ATM2.deposit(Account2)
ATM2.check_currentbalance(Account2)
ATM2.check_serialnumber()
ATM2.view_transactionsummary()
```

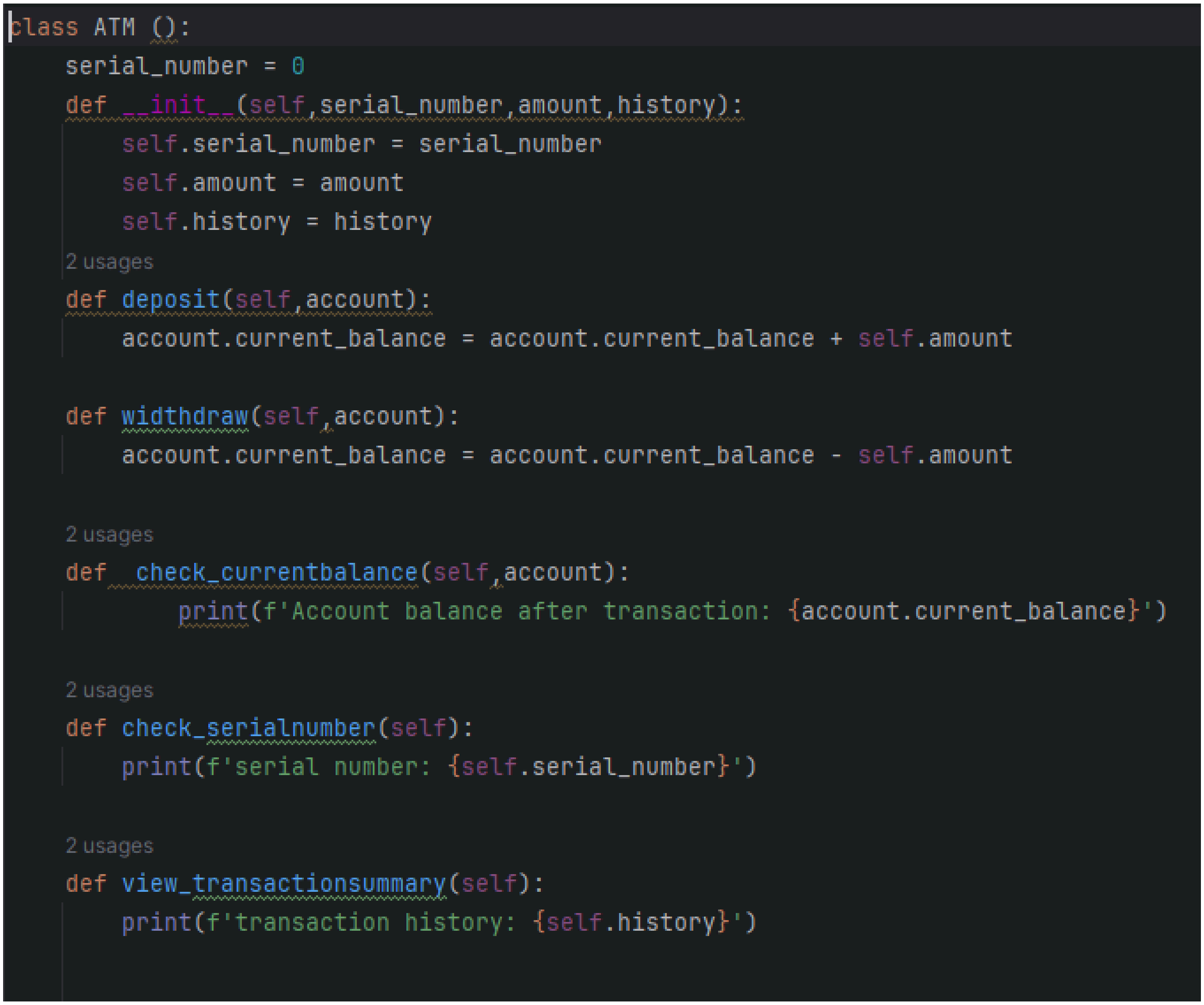


```
================Account1================
account number: 812371
name: Malupiton Boss
acoount balance: 298371
address: 9827 North Chumash
email: malupiton@gmail.com
Account balance after transaction: 298871
serial number: 12345
transaction history: deposit
```



```
================Account2================
account number: 98329
name: Kieran Syncable
acoount balance: 99999
address: 9827 South Chumash
email: qckiearan@tip.edu.ph
Account balance after transaction: 100499
serial number: 67891
transaction history: deposit

Process finished with exit code 0
```

3. Modify the ATM.py program and add the **view_transactionsummary()** method. The method should display all the transaction made in the ATM object.



```
class ATM ():
    serial_number = 0
    def __init__(self,serial_number,amount,history):
        self.serial_number = serial_number
        self.amount = amount
        self.history = history
    2 usages
    def deposit(self,account):
        account.current_balance = account.current_balance + self.amount

    def widthdraw(self,account):
        account.current_balance = account.current_balance - self.amount

    2 usages
    def  check_currentbalance(self,account):
            print(f'Account balance after transaction: {account.current_balance}')

    2 usages
    def check_serialnumber(self):
        print(f'serial number: {self.serial_number}')

    2 usages
    def view_transactionsummary(self):
        print(f'transaction history: {self.history}')
```

Questions

1. What is a class in Object-Oriented Programming?

a class is a blueprint that is used by programmers to create objects. It specifies the design and actions of a specific kind of objects

2. Why do you think classes are being implemented in certain programs while some are sequential(line-by-line)?

classes are more preferred because it makes it easier for programmers to understand complexity of codes by grouping classes and methods together using classes, classes also promote code reusability, making it more work efficient for programmers and easier to maintain compared to sequential scripts

3. How is it that there are variables of the same name such account_firstname and account_lastname that exist but have different values?

These 2 variables can exist and have different values because they can have different scopes, such as both of them being defined in different functions, which means that they can have different values and will not conflict against each other. They can also be attributed to different classes which they can have a different values, variables can also exist in different modules or libraries but also have different values.

4. Explain the constructor functions role in initializing the attributes of the class? When does the Constructor function execute or when is the constructor function called?

a constructor’s job is to set up values of an object when it is made, they can also allocate resources that the object requires, it can also assign values to objects if the programmers decides not to assign a value to an object. The constructor is executed as soon as an object is instantiated.

5. Explain the benefits of using Constructors over initializing the variables one by one in the main program?

constructors ensure that objects are always called upon in a consistent manner, when u create a new object, the constructor can set up values for that specific object, which validates the existence of the object. constructors also allow yout o set default values to newly created objects, which makes it more efficient in creating objects if no specific value is set for that specific object.

7. Conclusion:

In this activity, i was able to learn how to identify methods and attributes af a given object. I also learned how to modify attributes of instances and objects. I learned about classes in OOP and as well as sequential method. I learned about their differences and how each method has their own pros and cons. I learned about the definition of classes and how they are implemented in codes and how they make it easier to maintain and reuse codes. I also learned about the constructor function, its role of setting up values for an object when it is made, how it allocates resources that the object needs, how it executes, as well as its benefits.

8. Assessment Rubric: