Laboratory Activity No. _

Class, Objects, Methods

Course Code: PSMDSRC103	**Program: PSMDS**
Course Title: Programming	**Date Performed:**
Section:	**Date Submitted:**
Name: RAFFY BASAL	**Instructor: Engr. ROMAN RICHARD**

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
+ account_number: int + account_firstname: string + account_lastname: string + current_balance: float + address: string + email: string

+ update_address(new_address: string)
+ update_email(new_email: string)

ATM
+ serial_number: int
+ deposit(account: Accounts, amount: int)
+ widthdraw(account: Accounts, amount: int)
+ check_currentbalance(account: Accounts)
+ view_transactionsummary()

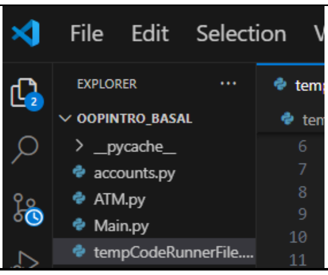
4. Materials and Equipment:

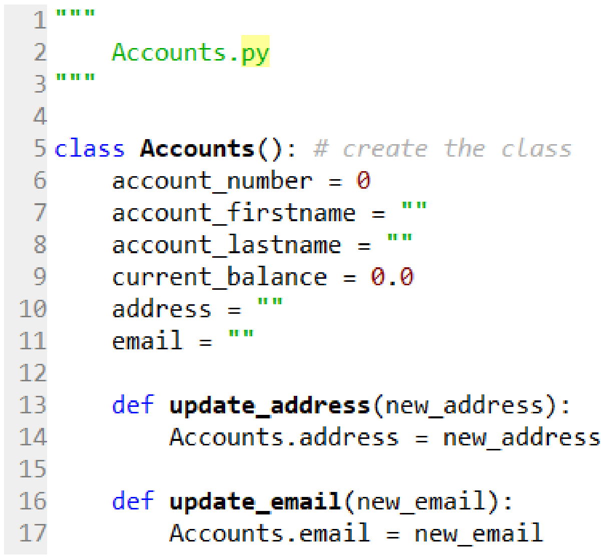
Desktop Computer with Anaconda Python
Windows Operating System

5. Procedure:

Creating Classes

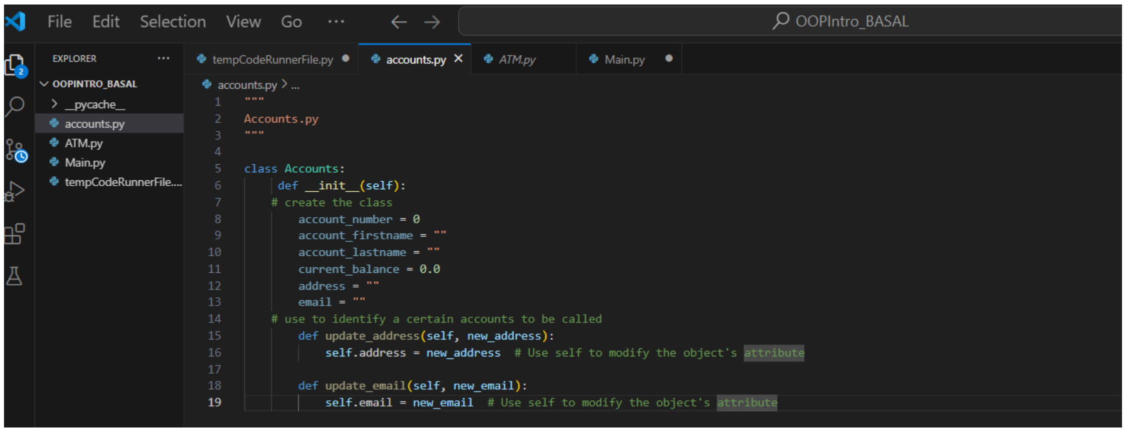
1. Create a folder named **OOPIntro_LastName**
2. Create a Python file inside the **OOPIntro_LastName** folder named **Accounts.py** and copy the code shown below:





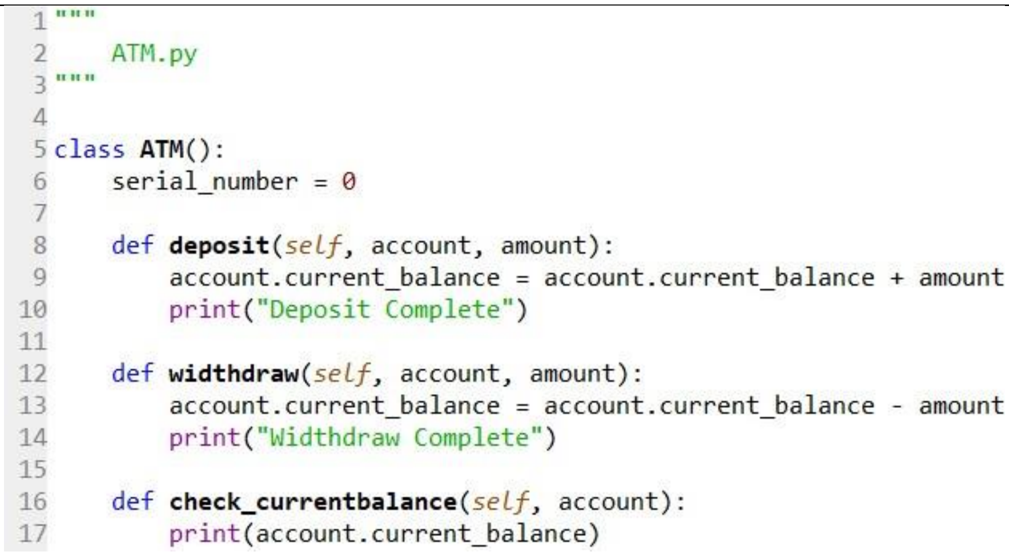
```python
"""
    Accounts.py
"""

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.



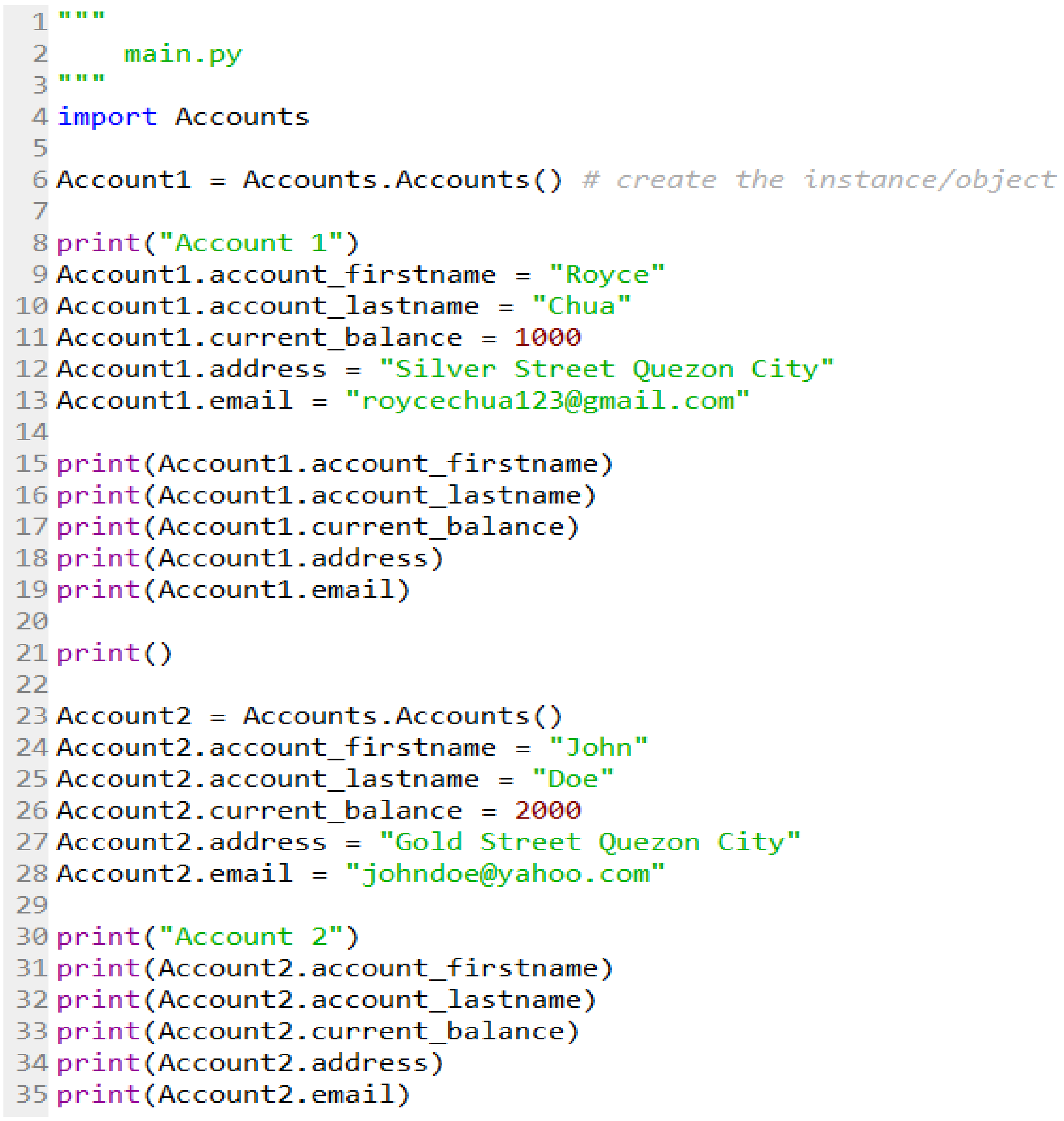
```python
"""
Accounts.py
"""

class Accounts:
    def __init__(self):
    # create the class
        account_number = 0
        account_firstname = ""
        account_lastname = ""
        current_balance = 0.0
        address = ""
        email = ""
    # use to identify a certain accounts to be called
        def update_address(self, new_address):
            self.address = new_address  # Use self to modify the object's attribute

        def update_email(self, new_email):
            self.email = new_email  # Use self to modify the object's attribute
```

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:

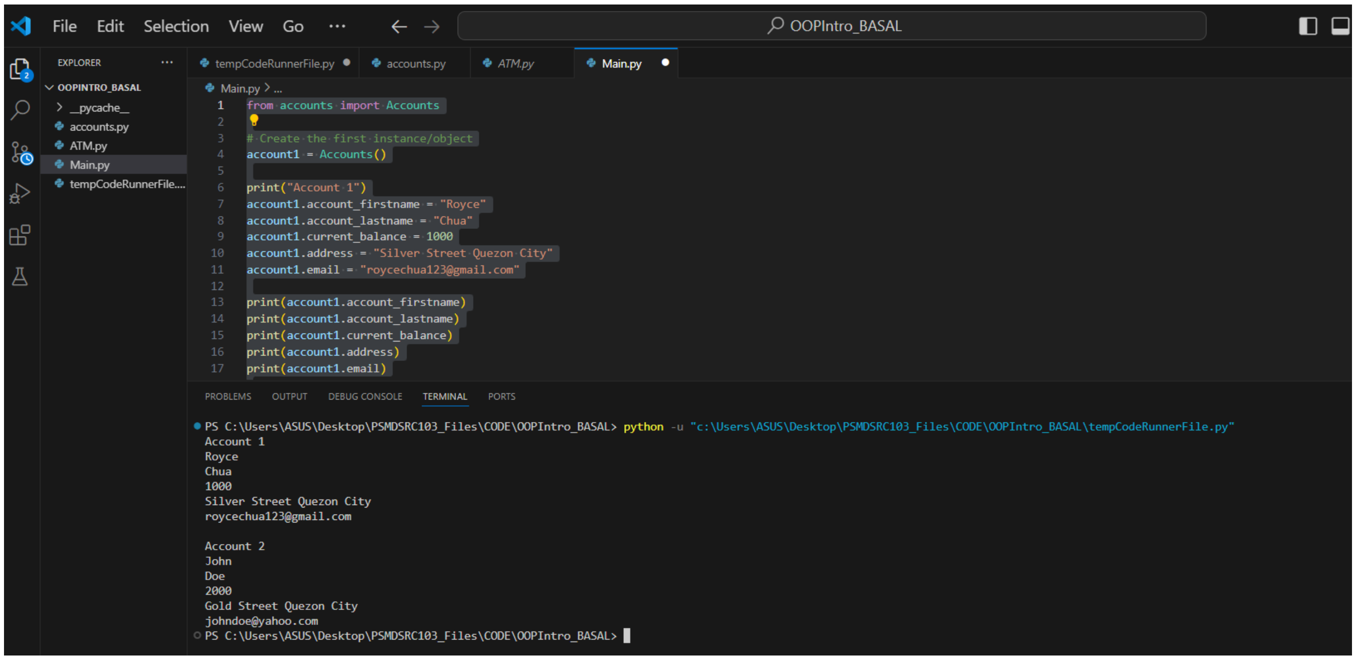


```python
"""
    main.py
"""
import Accounts

Account1 = Accounts.Accounts() # create the instance/object

print("Account 1")
Account1.account_firstname = "Royce"
Account1.account_lastname = "Chua"
Account1.current_balance = 1000
Account1.address = "Silver Street Quezon City"
Account1.email = "roycechua123@gmail.com"

print(Account1.account_firstname)
print(Account1.account_lastname)
print(Account1.current_balance)
print(Account1.address)
print(Account1.email)

print()

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

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

   At first, it can't be imported because the file name and the class name is both with Capital Letter "A", it caused a confusion on the system. Second, I revised the Import thru "accounts.py", to specify on what file will the import will be from, and remove the ".Accounts" since it is a class and don't need to add other function.

   For the variables use, the main.py calling the descriptive variable (e.g. account_firstname) together with a prefix "account1 or account2", to specify that they are called on that on that specific prefixes. For the "Accounts" class, it also used a prefix of the name "self" in the constructor"__init__code, also called an Instance Variable.

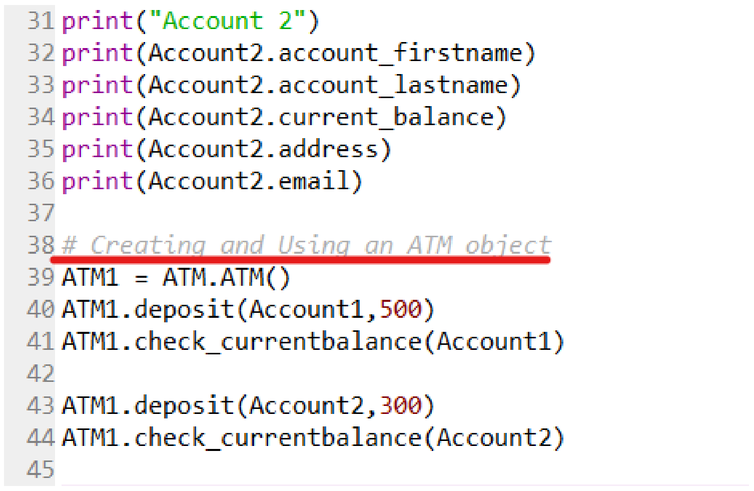


```python
from accounts import Accounts

# Create the first instance/object
account1 = Accounts()

print("Account 1")
account1.account_firstname = "Royce"
account1.account_lastname = "Chua"
account1.current_balance = 1000
account1.address = "Silver Street Quezon City"
account1.email = "roycechua123@gmail.com"

print(account1.account_firstname)
print(account1.account_lastname)
print(account1.current_balance)
print(account1.address)
print(account1.email)
```

```
PS C:\Users\ASUS\Desktop\PSMDSRC103_Files\CODE\OOPIntro_BASAL> python -u "c:\Users\ASUS\Desktop\PSMDSRC103_Files\CODE\OOPIntro_BASAL\tempCodeRunnerFile.py"
Account 1
Royce
Chua
1000
Silver Street Quezon City
roycechua123@gmail.com

Account 2
John
Doe
2000
Gold Street Quezon City
johndoe@yahoo.com
PS C:\Users\ASUS\Desktop\PSMDSRC103_Files\CODE\OOPIntro_BASAL>
```

7. Modify the main.py program and add the code underlined in red.
8. Modify the main.py program and add the code below line 38.

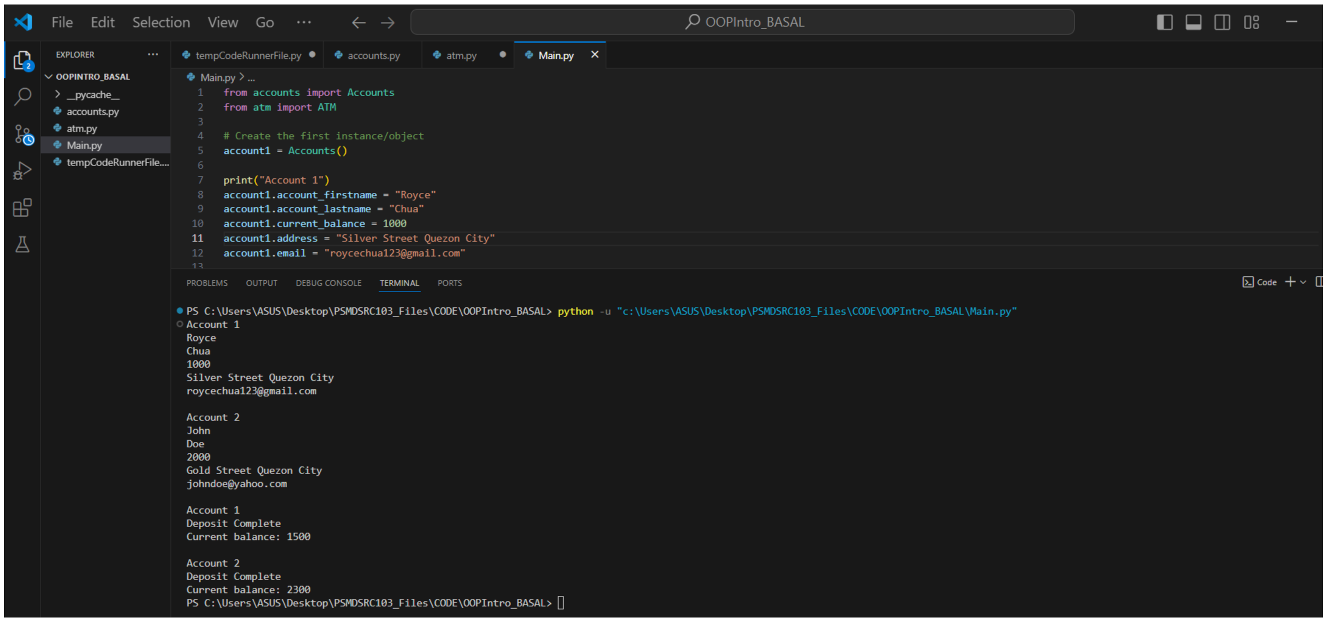
```python
 1 """
 2     main.py
 3 """
 4 import Accounts
 5 import ATM
 6 
 7 Account1 = Accounts.Accounts() # create the instance/object
 8 
 9 print("Account 1")
10 Account1.account_firstname = "Royce"
11 Account1.account_lastname = "Chua"
12 Account1.current_balance = 1000
13 Account1.address = "Silver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



```python
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.

The program run smoothly. There’s a need of revision of some codes due to, it were not supported by “’atm.py”. It returns an error every time I tried to run. A perfect example is the “accounts.py”, due to improper use of the instance variable which is “self”. I though that having a constructor __init__(self) will be enough because it runs when the “accounts.py” were imported.



```python
from accounts import Accounts
from atm import ATM

# Create the first instance/object
account1 = Accounts()

print("Account 1")
account1.account_firstname = "Royce"
account1.account_lastname = "Chua"
account1.current_balance = 1000
account1.address = "Silver Street Quezon City"
account1.email = "roycechua123@gmail.com"
```

```
PS C:\Users\ASUS\Desktop\PSMDSRC103_Files\CODE\OOPIntro_BASAL> python -u "c:\Users\ASUS\Desktop\PSMDSRC103_Files\CODE\OOPIntro_BASAL\Main.py"
Account 1
Royce
Chua
1000
Silver Street Quezon City
roycechua123@gmail.com

Account 2
John
Doe
2000
Gold Street Quezon City
johndoe@yahoo.com

Account 1
Deposit Complete
Current balance: 1500

Account 2
Deposit Complete
Current balance: 2300
PS C:\Users\ASUS\Desktop\PSMDSRC103_Files\CODE\OOPIntro_BASAL> 
```

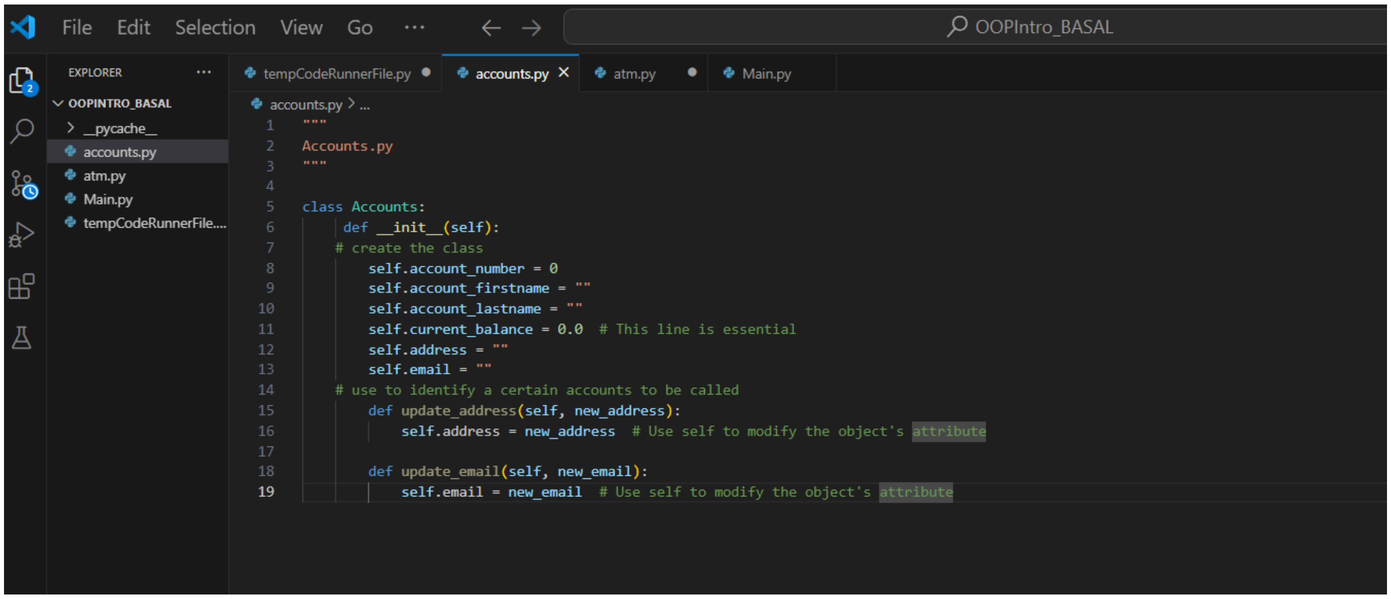
Create the Constructor in each Class

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

```python
"""
    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. Run the main.py program.

Using “self.” In the constructor __init__ method will allow each instance of the “Accounts” to have its own values for an attribute. This helps in when “atm.py” were imported to the main.py. It is recognized when “accounts.py” were Imported but when with “atm.py” an error occurs due to improper use of instance variable.

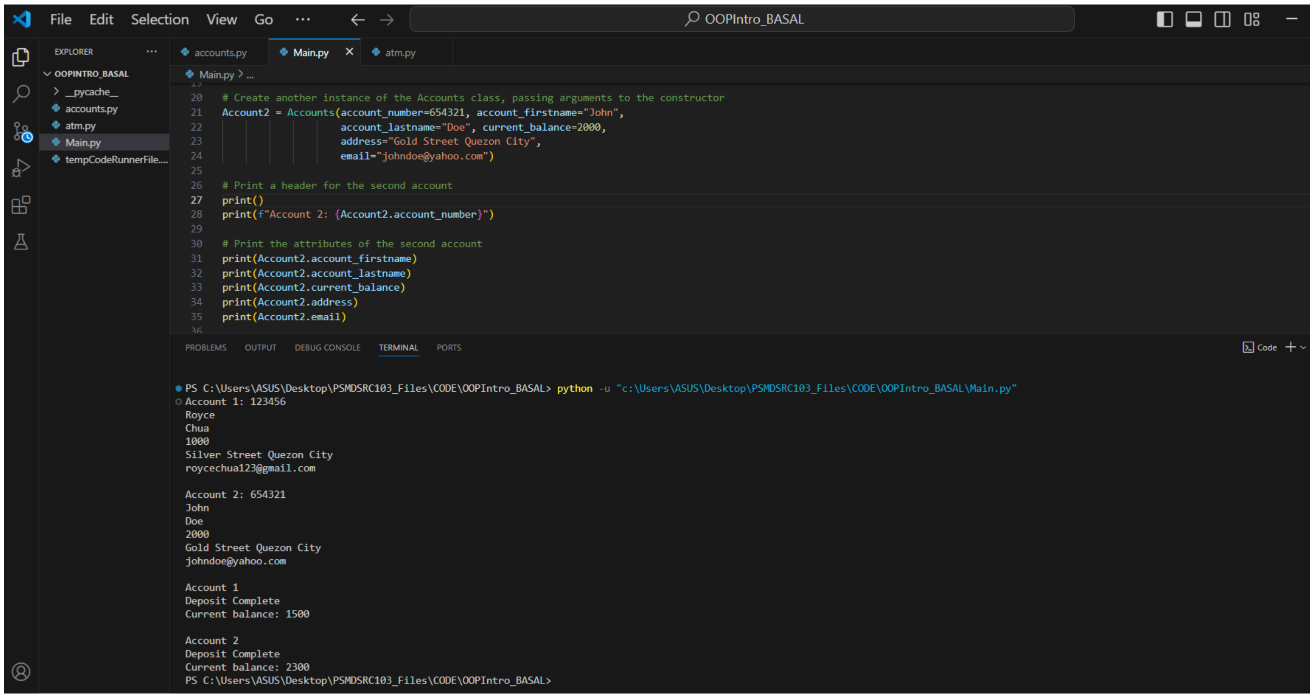


```python
"""
Accounts.py
"""

class Accounts:
    def __init__(self):
    # create the class
        self.account_number = 0
        self.account_firstname = ""
        self.account_lastname = ""
        self.current_balance = 0.0  # This line is essential
        self.address = ""
        self.email = ""
    # use to identify a certain accounts to be called
        def update_address(self, new_address):
            self.address = new_address  # Use self to modify the object's attribute

        def update_email(self, new_email):
            self.email = new_email  # Use self to modify the object's attribute
```

3. Modify the main.py and change the following codes with the red line.
4. Run the main.py program again and run the output.

```python
"""
    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")

```

It displays same result but easier way of assigning value. Just like SQL, but the difference is you call each instance every time you will assign. This process is much more convenient than the other example.



```python
# Create another instance of the Accounts class, passing arguments to the constructor
Account2 = Accounts(account_number=654321, account_firstname="John",
                    account_lastname="Doe", current_balance=2000,
                    address="Gold Street Quezon City",
                    email="johndoe@yahoo.com")

# Print a header for the second account
print()
print(f"Account 2: {Account2.account_number}")

# Print the attributes of the second account
print(Account2.account_firstname)
print(Account2.account_lastname)
print(Account2.current_balance)
print(Account2.address)
print(Account2.email)
```

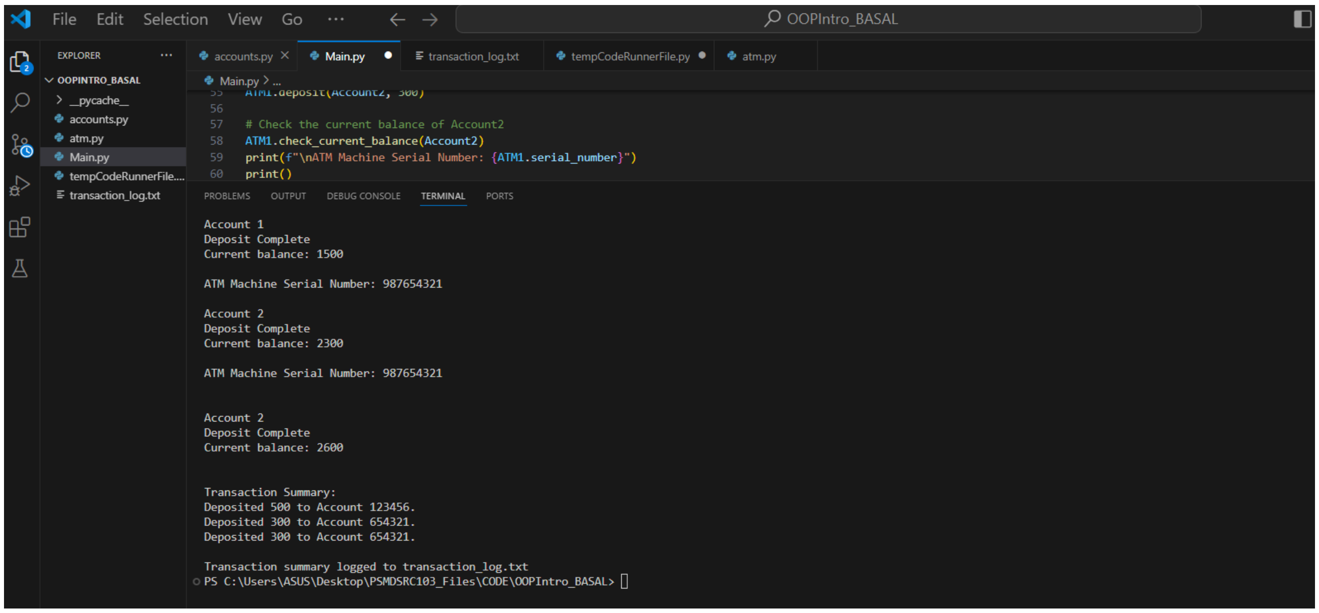
```
PS C:\Users\ASUS\Desktop\PSMDSRC103_Files\CODE\OOPIntro_BASAL> python -u "c:\Users\ASUS\Desktop\PSMDSRC103_Files\CODE\OOPIntro_BASAL\Main.py"
Account 1: 123456
Royce
Chua
1000
Silver Street Quezon City
roycechua123@gmail.com

Account 2: 654321
John
Doe
2000
Gold Street Quezon City
johndoe@yahoo.com

Account 1
Deposit Complete
Current balance: 1500

Account 2
Deposit Complete
Current balance: 2300
PS C:\Users\ASUS\Desktop\PSMDSRC103_Files\CODE\OOPIntro_BASAL>
```

6. Supplementary Activity:

Tasks

1. Modify the ATM.py program and add the constructor function.
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.
3. Modify the ATM.py program and add the **view_transactionsummary()** method. The method should display all the transaction made in the ATM object and log it to a text file.

This function, adding a constructor __init__ to the atm code will allow us to initialize the new added custom value "serial_number". Just like what we did in the "accounts.py", we add an constructor to be able to access the custom value that are unique in the code. On the other hand "view_transactionsummary()", is a function that will be a record of all the transaction history logs.



```python
ATM1.deposit(Account2, 300)

# Check the current balance of Account2
ATM1.check_current_balance(Account2)
print(f"\nATM Machine Serial Number: {ATM1.serial_number}")
print()
```

```
Account 1
Deposit Complete
Current balance: 1500

ATM Machine Serial Number: 987654321

Account 2
Deposit Complete
Current balance: 2300

ATM Machine Serial Number: 987654321


Account 2
Deposit Complete
Current balance: 2600


Transaction Summary:
Deposited 500 to Account 123456.
Deposited 300 to Account 654321.
Deposited 300 to Account 654321.

Transaction summary logged to transaction_log.txt
PS C:\Users\ASUS\Desktop\PSMDSRC103_Files\CODE\OOPIntro_BASAL>
```

Questions

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

   OOP class serves like a blueprint or a template for creating an object, where it defines the attributes and function of a class that has been created. In a class it was encapsulated, meaning that attributes and the functions that operate on that data are bundled together into one unit.

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

   Classes are implemented in programs to structure and organize code, especially in complex systems. When a program involves many related data and operations that need to interact consistently (e.g. above activity), classes provide a way to manage these entities easily. Classes support concepts like encapsulation, inheritance, and polymorphism, which are useful for building reusable and modular code.

While sequential programs (line-by-line) are often simpler and suitable for small tasks. In simpler programs where only a sequence of operations is needed (e.g. basic calculations or data processing), a line-by-line approach is sufficient. There is no need for object-oriented abstractions if the task doesn’t involve multiple interrelated entities.

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

   The variables exist with different values because each instance of a class has its own copy of the attributes (e.g. Account1 and Account2). They are objects created from the same class “Account”, but they store their data separately. The class serves as a template, while the objects store individual values.

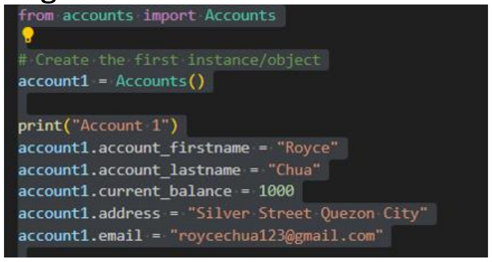
4. Explain the constructor functions role in initializing the attributes of the class? When does the Constructor function execute or when is the constructor function called?

   The constructor function “__init__” in Python is a special method in a class that is called when an object is created from that class. It’s used to **initialize** the object's attributes with default or provided values. The “__init__” method is automatically called, initializing the attributes of instance (e.g. Account1). The constructor is executed immediately upon object creation and ensures the object is set up properly before it is used.

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

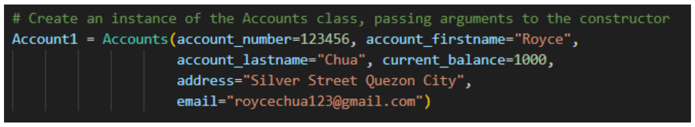
   Automatic Initialization - A constructor ensures that the object is fully initialized in a single step when it’s created. This avoids the risk of forgetting to initialize certain attributes, which can lead to errors. A Cleaner Code with constructors, the code becomes cleaner and easier to read. Instead of setting each attribute manually.

   e.g. from this:



```python
from accounts import Accounts

# Create the first instance/object
account1 = Accounts()

print("Account 1")
account1.account_firstname = "Royce"
account1.account_lastname = "Chua"
account1.current_balance = 1000
account1.address = "Silver Street Quezon City"
account1.email = "roycechua123@gmail.com"
```

   to this:



```python
# Create an instance of the Accounts class, passing arguments to the constructor
Account1 = Accounts(account_number=123456, account_firstname="Royce",
                    account_lastname="Chua", current_balance=1000,
                    address="Silver Street Quezon City",
                    email="roycechua123@gmail.com")
```

   Reusability of Constructors allow you to reuse the same initialization logic across multiple instances of a class, reducing code duplication. Using constructors helps maintain consistency. The initialization process is centralized in one place (the constructor), ensuring that all objects are initialized in the same manner, which avoids inconsistencies in your code.

7. Conclusion:

LEARNING:

In learning about Object-Oriented Programming (OOP), I discover that classes serve as blueprints for creating objects, encapsulating related data (attributes) and functionality (methods) to enhance code organization and modularity.

Constructors play a crucial role in initializing these objects with specific values upon instantiation, ensuring they are set up correctly from the start, which helps prevent errors and reduces redundancy. This structured approach contrasts with sequential programming, which is suited for simpler tasks, as OOP allows for more complex systems where multiple entities can interact independently.

CONCLUSION:

Using classes and object-oriented principles improves the organization, clarity, and reusability of your code. Constructors offer an efficient way to initialize objects, providing a reliable starting state for each object in a program. By structuring code in this manner, it's easier to scale up, manage complex interactions, and avoid common issues like repeated code or inconsistent initialization.

OOP practices, such as constructors and encapsulation, ensure that code is more maintainable, flexible, and suited for long-term projects, whereas sequential programming is best for simple, straightforward tasks.

8. Assessment Rubric: