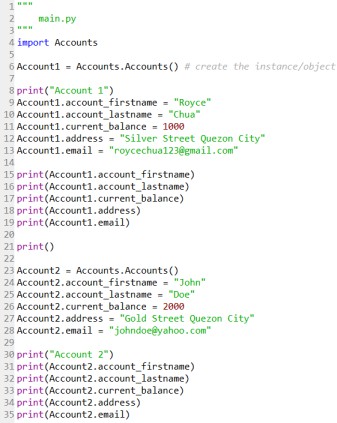
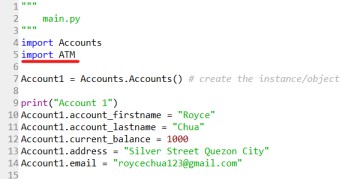
|  |  |
| --- | --- |
| **Laboratory Activity No. 3.1** | |
| **Introduction to Object-Oriented Programming** | |
| **Course Code:** CPE103 | **Program:** BSCPE |
| **Course Title:** Object-Oriented Programming | **Date Performed:** Jan 25, 2025 |
| **Section:** 1A | **Date Submitted:** Jan 31, 2025 |
| **Name:** Eulin, Ryan Bertrand B. | **Instructor:** Engr. Maria Rizette 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:   * 1. Identify the possible attributes and methods of a given object   2. Create a class using the Python language   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/Python Colab 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: |

|  |
| --- |
| 1. Modify the Accounts.py and add ***self,*** before the new\_address and new\_email. 2. Create a new file named ATM.py and copy the code shown below:     **Creating Instances of Classes**   1. Create a new file named main.py and copy the code shown below: |

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.

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

|  |
| --- |
| 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   1. Modify the main.py and change the following codes with the red line. Do not remove the other codes in the program. |

|  |
| --- |
| 1. Run the main.py program again and run the output.   <https://colab.research.google.com/drive/1TNe7aTR_BkZZR-7HPqrEQpMMy3aQqczK#scrollTo=2D3TTXGVEkUG> |
| **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.     <https://colab.research.google.com/drive/1TNe7aTR_BkZZR7HPqrEQpMMy3aQqczK#scrollTo=2D3TTXGVEkUG>  **Questions**   1. What is a class in Object-Oriented Programming?   A class in OOP is like a blueprint or template used to create object. It defines the attributes, the  methods that the object will have.   1. Why do you think classes are being implemented in certain programs while some are sequential(line-by-line)?   Classes are implemented in programs when there is a need to structurize the source codes.   1. How is it that there are variables of the same name such account\_firstname and account\_lastname that exist but have different values?   Because they are stored in different contexts, that allows them to 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 function is a special type of function that set the value or put in the condition appropriate to the start of an operation |

|  |
| --- |
| 5. Explain the benefits of using Constructors over initializing the variables one by one in the main program?  Constructor function is important since, this type of function they centralize and automate the object attributes and also, they ensure the consistency in every lines of the source code. Constructors can help to maintain the code by preventing redundancy or more than what is the necessary in the program, more cleaner and readable program  **7. Conclusion:** |
| Object-Oriented Programming (OOP) introduces classes as blueprints to define and organize attributes and methods, enabling better structure and modularity in programs. While sequential programming is suitable for simpler, linear tasks, classes provide a more organized approach for complex systems, allowing for easier maintenance and scalability. Constructor functions play a key role in initializing object attributes, ensuring consistency and efficiency by automating the setup process. By using constructors, redundancy is minimized, and code readability is improved, making it a critical feature in modern programming. |
| **8. Assessment Rubric:** |