UNIVERSITY OF CALOOCAN CITY
Caloocan, 1400 Metro Manila, Philippines

COLLEGE OF ENGINEERING
Computer Engineering
2nd Semester, School Year 2024-2025

Object-Oriented Programming

Laboratory Activity No. 1

Review of Technologies

Submitted by:

Catahan, Joshua A.

Saturday / BSCpE 1A

Submitted to

Engr. Maria Rizette Sayo

Professor

Date Performed:

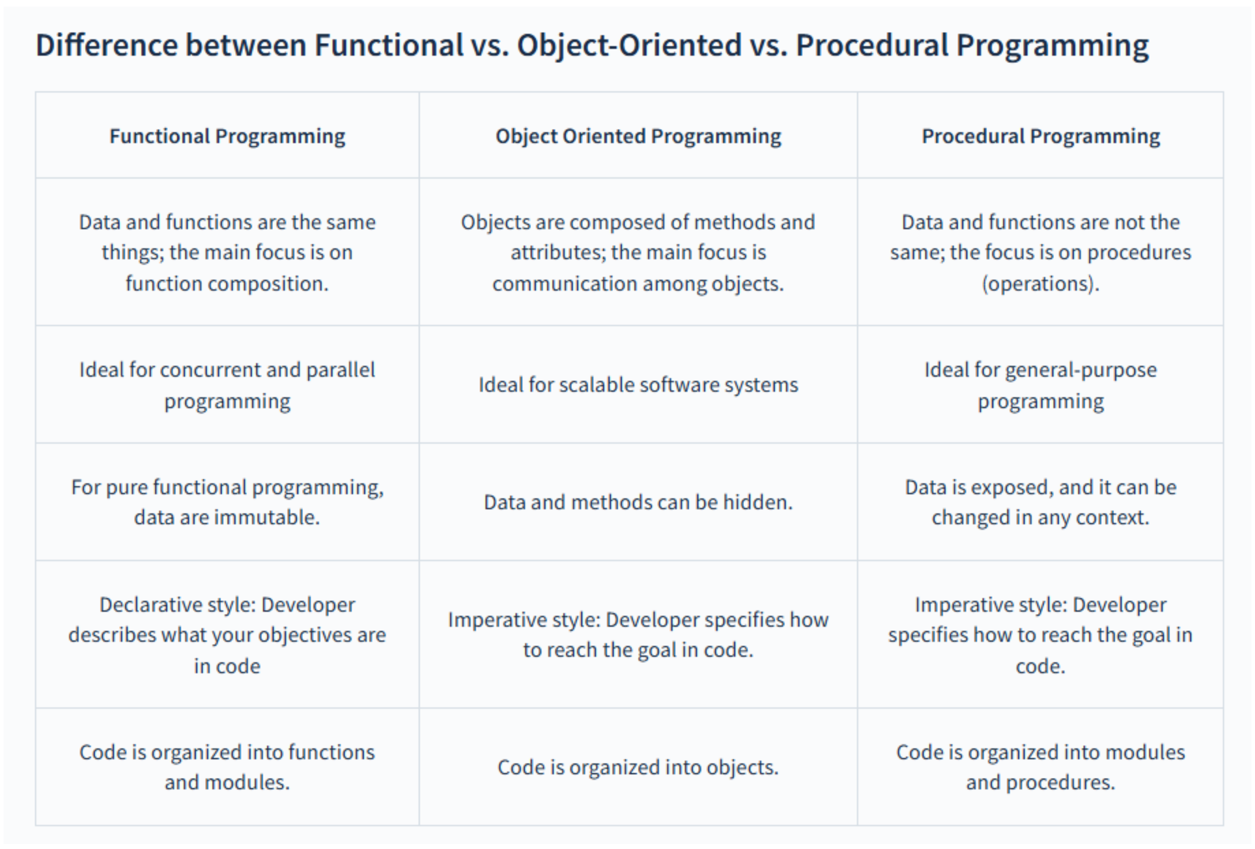
18-01-2025

Date Submitted

21-01-2025

Object Oriented Programming (OOP) pertains to languages that rely on the concept of objects and classes in programming. Its central aim is to bind the data and functions that operate on them so that no other part of the code can access this data except that function. Furthermore, the goal of object-oriented programming is to incorporate real-world concepts into programming, such as inheritance, hiding, polymorphism, etc. (GeeksforGeeks, 2023). Additionally, OOP promotes code modularity and reusability by grouping data and behavior into objects, simplifying program maintenance (Amos, 2024). Whereas, it encourages the DRY (Don't Repeat Yourself) principle for cleaner and more scalable code (Wagner, 2022).

Other than OOP there are other programming paradigms that exist, such as procedural programming and functional programming. All of these offers different advantages and disadvantages in solving problems based on specific principles.



Difference between Functional vs. Object-Oriented vs. Procedural Programming

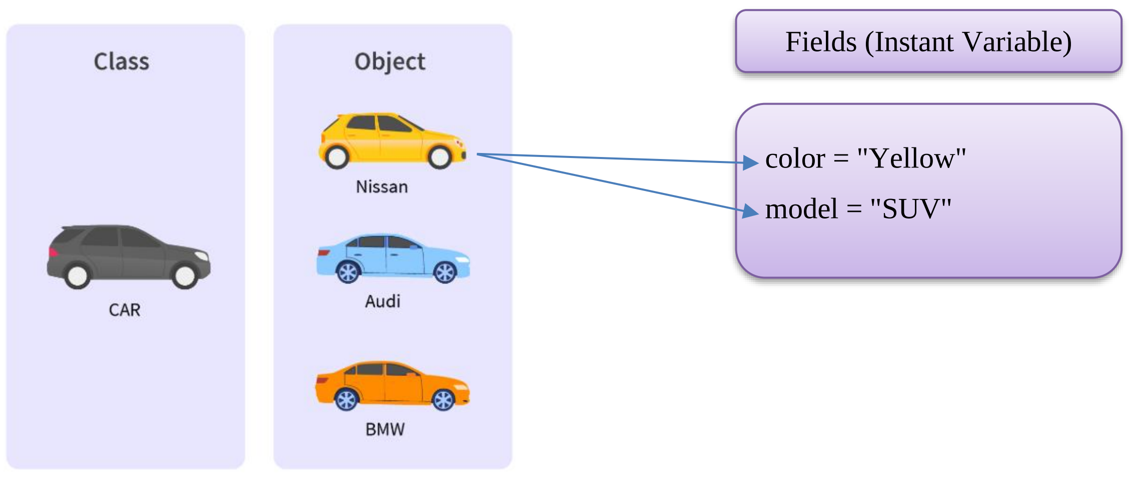
Functional Programming	Object Oriented Programming	Procedural Programming
Data and functions are the same things; the main focus is on function composition.	Objects are composed of methods and attributes; the main focus is communication among objects.	Data and functions are not the same; the focus is on procedures (operations).
Ideal for concurrent and parallel programming	Ideal for scalable software systems	Ideal for general-purpose programming
For pure functional programming, data are immutable.	Data and methods can be hidden.	Data is exposed, and it can be changed in any context.
Declarative style: Developer describes what your objectives are in code	Imperative style: Developer specifies how to reach the goal in code.	Imperative style: Developer specifies how to reach the goal in code.
Code is organized into functions and modules.	Code is organized into objects.	Code is organized into modules and procedures.

https://www.scaler.com/topics/java/oop-vs-functional-vs-procedural/

This table above came from Amiana (2022), in which it explains the key differences and advantages of these programming paradigms. Amiana (2022) states that procedural programming prioritizes functions over data, thereby fostering reusability and modularity in its general tasks. While for OOP, it treats everything as objects to enhance scalability and reusability in large systems. On the other hand, functional programming views functions as data, which excels in concurrent programming and has two types: pure and impure. This table does not necessarily mean that this one paradigm is the best, and this one is the worst. Each of them has their own functions and characteristics that fit a specific purpose. Knowing your goals when creating a program will help you pick out the most suitable type, which would help improve the quality of your work. As well as to avoid wasting time on something that isn’t meant for it.

Moving on to the main subject matter of this laboratory report. In Object Oriented Programming there are many key concepts that is essential in understanding how this paradigm works. Some of the key factors in understanding these are the **classes**, **objects**, **fields**, **methods**, and **properties**.

A **class** is a blueprint for creating objects. It defines attributes (data) and methods (functions) that describe an object's behavior. For example, a Car class might have attributes like color and speed and methods like drive or brake. While **objects** in python refers to the instance of a class. It holds specific values for the attributes defined in the class. For example, a red sports car moving at 100 km/h is an object of the Car class. (PYnative, 2024). If object is the instant of the class, then, fields are the instant of the objects. **Fields** are also known as instance variable or attributes of an object. Instance variables in Python are unique to each object, with each object having its own copy. Instance methods, which operate on these variables, require an object to execute the defined actions. It is declared using the self keyword.

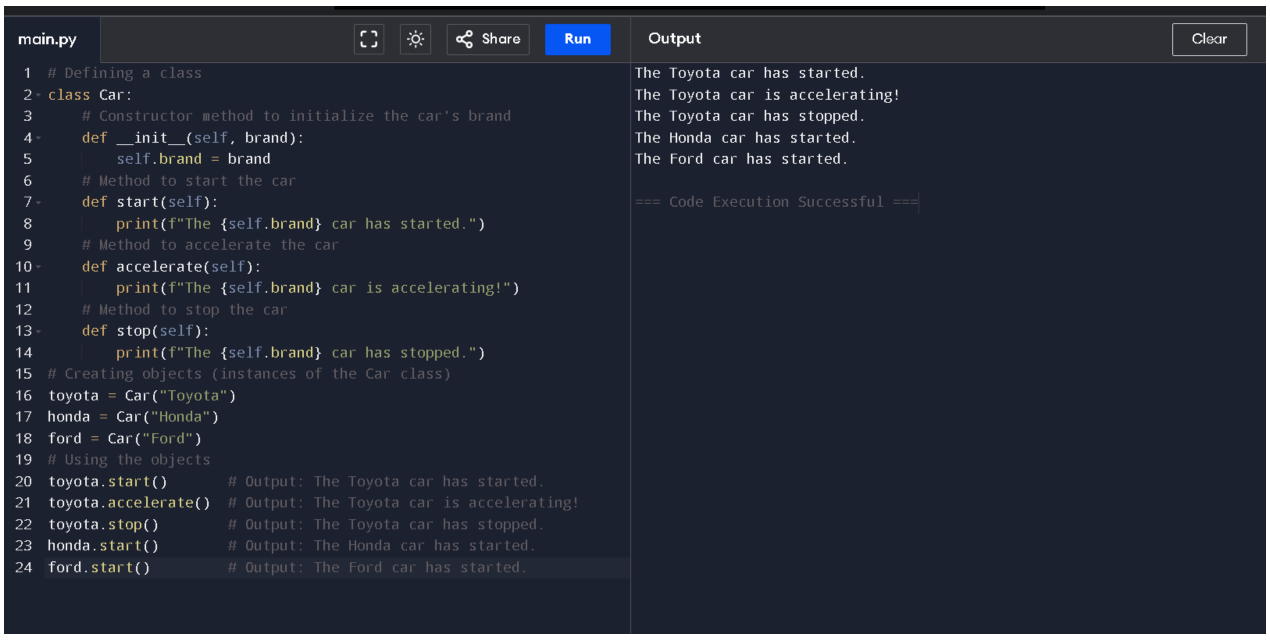


https://www.scaler.com/topics/java/oop-vs-functional-vs-procedural/

As seen based on the given figure above, the class refers to the car while the object refers to the brands. On the other hand, the fields was the observed attributes or characteristics of the object. In Object-Oriented Programming (OOP), the core idea is to treat things as "objects" that reflect real-world entities. These objects have two key aspects: their attributes, which describe their characteristics, like a car's model name or speed, and their behaviors, which are actions they can perform such as starting, accelerating, or stopping. The idea is to interact with these objects mainly through their public attributes and actions, much like how we interact with real-world objects. For example, a car object allows you to check its speed or ask it to accelerate, just like you would with an actual car.

However, class and objects are just a declaration of your variables like in C++. To make the objects do something, we need methods. **Methods** are functions that belong to a class and define how its objects behave. They allow objects to perform actions, manipulate data, and interact with one another, helping developers build organized and modular systems. When creating objects, these methods can be used by instances of the class. For example, in a Car class, the start() method is defined, and a specific car object can call this method to start the engine. Python also provides three types of methods: instance methods (operating on specific class instances), class methods (bound to the class and modifying class-level attributes), and static methods (independent of class and instance data, often used for utility functions). These features enable Python developers to write cleaner, more modular, and scalable code (Zaidi, 2023).

Let's try to go a little deeper into learning the basics and apply the principles through coding. This picture is a screenshot of the code I typed in an online compiler called programiz.com.



```python
# Defining a class
class Car:
    # Constructor method to initialize the car's brand
    def __init__(self, brand):
        self.brand = brand
    # Method to start the car
    def start(self):
        print(f"The {self.brand} car has started.")
    # Method to accelerate the car
    def accelerate(self):
        print(f"The {self.brand} car is accelerating!")
    # Method to stop the car
    def stop(self):
        print(f"The {self.brand} car has stopped.")
# Creating objects (instances of the Car class)
toyota = Car("Toyota")
honda = Car("Honda")
ford = Car("Ford")
# Using the objects
toyota.start()       # Output: The Toyota car has started.
toyota.accelerate()  # Output: The Toyota car is accelerating!
toyota.stop()        # Output: The Toyota car has stopped.
honda.start()        # Output: The Honda car has started.
ford.start()         # Output: The Ford car has started.
```

```
The Toyota car has started.
The Toyota car is accelerating!
The Toyota car has stopped.
The Honda car has started.
The Ford car has started.

=== Code Execution Successful ===
```

1. **Class Declaration**:
   In the code, the class keyword is used to define the blueprint for a Car. This blueprint contains methods (functions) that describe the actions a car can perform, such as start, accelerate, and stop. Declaring a class is the first step in creating objects in Object-Oriented Programming.

EXAMPLE:	class Car:

   The name of the class is Car, which is written in PascalCase as per Python's naming conventions (McKenzie, 2021).

2. **Constructor Method**:
   The constructor method, __init__, is a special method in Python used to initialize a new object. In this code, it takes self (a reference to the current object) and brand as parameters. The self.brand stores the car's brand name, so every object knows its specific brand.

EXAMPLE:	def __init__(self, brand):
	self.brand = brand

   Although fields like brand are included here for object uniqueness, the focus is on methods that define the car's actions.

3. **Defining Methods**:
   Methods in the Car class define what the car can do. For example:

   - start prints that the car has started.
   - accelerate shows that the car is speeding up.
   - stop indicates that the car has stopped.

   These methods use the self parameter to refer to the object calling the method. This ensures the actions are tied to the specific car object.

EXAMPLE:	def start(self):
	print(f"The {self.brand} car has started.")

4. **Creating Objects**:
   Objects are instances of a class created by calling the class name as if it were a function.

EXAMPLE:	toyota = Car("Toyota")

   This creates an object named toyota of the Car class, initializing its brand to "Toyota." Similarly, honda and ford are other instances with unique brands.

5. **Using Methods**:
   Once an object is created, you can use its methods to perform actions. For example:

EXAMPLE:	toyota.start()

   This calls the start method on the toyota object, displaying "The Toyota car has started." Similarly, calling accelerate or stop on any car object produces results based on that object.

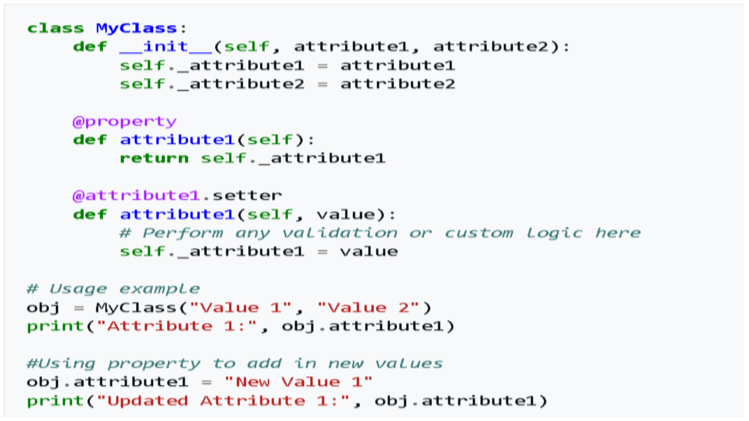
6. **How the Code Works**:

   - The Car class serves as a generic template.
   - Objects are specific instances (e.g., toyota and honda).
   - Methods like start and stop enable interaction with these objects, allowing each car to perform actions independently.

By combining class declarations, constructors, and methods, this code demonstrates how Python allows developers to model real-world entities (like cars) and their behaviors in an organized, reusable way.

After understanding the basics, advance concepts such as properties can further elevate the things we can do in coding. We can think of properties as functions in OOP, it has a function where you can alter variables without editing the source code through user inputs. This means that it can be used on user-interactive programs were information's updates frequently. In Python, the property keyword lets you define getter, setter, and deleter methods for class attributes, making them behave like properties. This provides control over how class attributes are accessed and modified, while offering a clear interface for external code (Wikibooks, n.d.).

For example, the @property decorator is used to create getter methods for each attribute, allowing access to them with syntax like obj.attribute1



```python
class MyClass:
    def __init__(self, attribute1, attribute2):
        self._attribute1 = attribute1
        self._attribute2 = attribute2

    @property
    def attribute1(self):
        return self._attribute1

    @attribute1.setter
    def attribute1(self, value):
        # Perform any validation or custom logic here
        self._attribute1 = value

# Usage example
obj = MyClass("Value 1", "Value 2")
print("Attribute 1:", obj.attribute1)

#Using property to add in new values
obj.attribute1 = "New Value 1"
print("Updated Attribute 1:", obj.attribute1)
```

https://en.wikibooks.org/wiki/Object_Oriented_Programming/Properties

Analysis of the Code (Focusing on Property):

1. **Encapsulation with Property:**

   - The code uses the @property decorator to provide controlled access to the private attribute _attribute1.
   - This helps encapsulate the data and prevents direct modification.

2. **Getter Method (@property):**

   - The method attribute1(self) acts as a getter, allowing access to _attribute1 in a controlled manner without exposing the private variable directly.

3. **Setter Method (@attribute1.setter):**

   - The setter method attribute1(self, value) allows modification of _attribute1 with the potential for validation or additional logic before updating the value.

4. **Usage Example:**

   - An instance of MyClass is created with initial values.
   - The property allows retrieval and modification of attribute1 in an intuitive and controlled way.

As previously stated, in OOP, you treat what you code as real-life objects with its own attribute. Let's try to look at a real life product that corporates this principles when coding and determine the class, objects, methods, fields, and its properties.

ATM Machine

https://jeemariyana.medium.com/oop-concepts-with-real-world-examples-cda1cd277f4f

The ATM machine in the image can be analyzed using Object-Oriented Programming (OOP) concepts. The **ATM** can be considered a **class**, representing the blueprint for various ATMs. Each specific ATM in different locations acts as an **object**, which is the instant from the ATM class. The ATM may have **fields** such as balance, cardInserted, and transactionHistory, representing its state. It provides **methods** like insertCard(), enterPIN(), withdrawCash(), and printReceipt() to define its behavior. Using properties, such as accountBalance, allows controlled access to sensitive data, ensuring encapsulation and security by restricting direct modification and allowing validation within setter methods.

In conclusion, Object-Oriented Programming (OOP) provides a structured and efficient approach to software development by organizing data and behavior into objects that mirror real-world entities. Making OOP a user-interactive paradigm with its various functions. Through key concepts such as classes, objects, fields, and methods, OOP enables code reusability, modularity, and scalability, making it ideal for developing complex systems. Additionally, properties in OOP allow for better control over data access and modification, enhancing security and flexibility in code management. While other programming paradigms such as procedural and functional programming offer their own advantages, understanding the principles of OOP equips developers with the tools to create organized and maintainable software solutions that align with specific project goals. Overall, this activity has helped me expand my knowledge through researching the required information in order to write this paper. The main lesson I received from this activity is that there are countless and amazing programming knowledge that are still waiting for me to discover.

References

Website:

Amiana, D. (2022, April 08). OOP vs Functional vs Procedural - Scaler Topics. Scaler Topics. Retrieved from https://www.scaler.com/topics/java/oop-vs-functional-vs-procedural/

Amos, D. (2024, December 15). Object-Oriented Programming (OOP) in Python. Real Python. Retrieved from https://realpython.com/python3-object-oriented-programming/

Design Gurus. Beginner's Guide to Object-Oriented Programming (OOP). Design Gurus: One-Stop Portal for Tech Interviews. Retrieved from https://www.designgurus.io/blog/object-oriented-programming-oop

GeeksforGeeks. (2023, February 09). Introduction of object-oriented programming. GeeksforGeeks. Retrieved from https://www.geeksforgeeks.org/introduction-of-object-oriented-programming/

McKenzie, C. (2021, March 24). What is Pascal case? TheServerSide.com. Retrieved from https://www.theserverside.com/definition/Pascal-case#:~:text=Pascal%20case%20%2D%2D%20or%20PascalCase,a%20software%20development%20best%20practice

Object Oriented Programming/Properties - Wikibooks, open books for an open world. (n.d.). Retrieved from https://en.wikibooks.org/wiki/Object_Oriented_Programming/Properties

PYnative. (2024, February 24). Python Object-Oriented Programming (OOP) archives. PYnative. Retrieved from https://pynative.com/python/object-oriented-programming/

Riyana, J. (2022, December 10). OOP Concepts with real-world examples. Medium. Retrieved from https://jeemariyana.medium.com/oop-concepts-with-real-world-examples-cda1cd277f4f

Wagner, L. (2022, October 20). Object Oriented Programming in Python – full crash course. freeCodeCamp.org. Retrieved from https://www.freecodecamp.org/news/crash-course-object oriented-programming-in-python/

Zaidi, E. (2023, December 25). The core properties of Object-Oriented Programming (OOP) in Python. Medium. Retrieved from https://medium.com/@eliyazaidi16/the-core-properties-of-object oriented-programming-oop-in-python-1f2fafd72161