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:
Beguia, Den Mark T.
Sat 4:30-8:30 / BSCPE 1-A

Submitted to
Maria Rizette Sayo
Instructor

Date Performed:
01-18-2025

Date Submitted
01-18-2025

I. Objectives

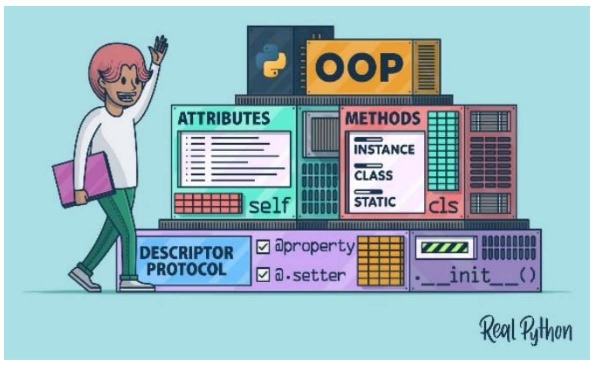
In this section, the goals in this laboratory are:

- To define the key terms in Object-oriented programming
- To be able to know the construction of OO concepts in relation to other types of programming such as procedural or functional programming

II. Methods

General Instruction:

A. Define and discuss the following Object-oriented programming concepts:



1. Classes

   By instantiating that class, you can access and make use of its member functions and data members. It represents a collection of characteristics or methods common to all objects of a certain type. A class is akin to a blueprint for an object.

2. Objects

   It signifies actual entities and is a crucial part of object-oriented programming. Memory is assigned when a class is created or when an object is generated, but not at the point of definition. Each object possesses a state, identity, and behavior.

3. Fields

   Object-oriented programming (OOP) is a fundamental programming paradigm used by nearly every developer at some point in their career. OOP is the **most popular programming paradigm** used for software development and is taught as the standard way to code for most of a programmer's educational career.

4. Methods

   A method is a programmed function defined within a class and available to any object created from that class in object-oriented programming (OOP). Any object can invoke the method, and it runs within the context of that object.

This allows the procedure to be reused multiple times among objects instantiated from the same class.

5. Properties

Abstraction

Abstraction involves hiding the complex implementation details and showing only the essential features of an object. This helps in reducing complexity and increasing efficiency.

Encapsulation

Encapsulation is the process of wrapping data and methods that operate on the data within a single unit, typically a class. This helps to hide the internal state of the object and only expose a controlled interface to the outside world.

Inheritance

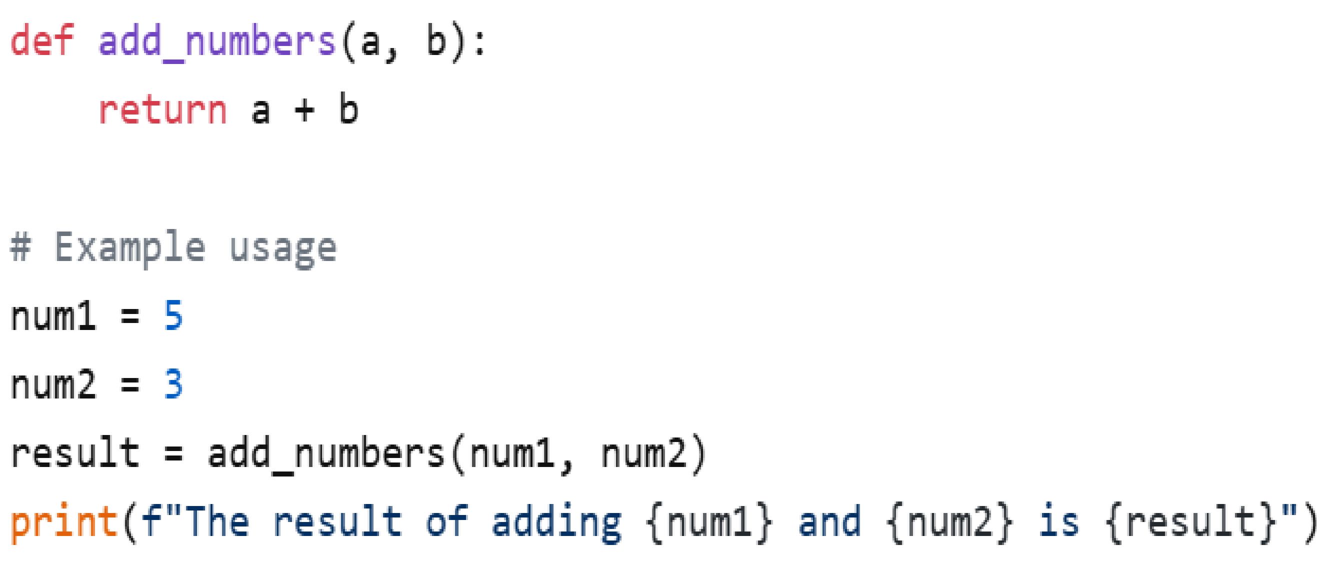
Inheritance allows a class to inherit properties and methods from another class. This promotes code reusability and establishes a relationship between classes.

Polymorphism

Polymorphism allows objects to be treated as instances of their parent class rather than their actual class. This can be achieved through method overloading and method overriding.

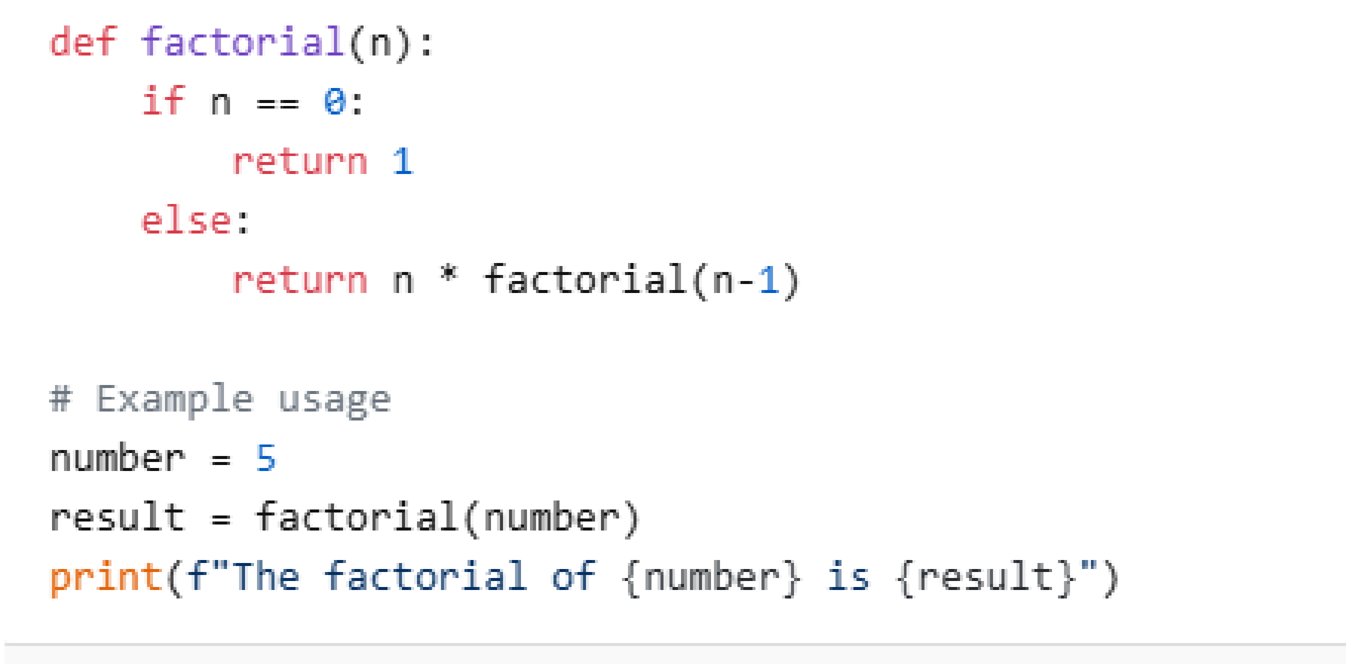
III. Results

This is an examples of an results of a phyton:



```python
def add_numbers(a, b):
    return a + b

# Example usage
num1 = 5
num2 = 3
result = add_numbers(num1, num2)
print(f"The result of adding {num1} and {num2} is {result}")
```



```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)

# Example usage
number = 5
result = factorial(number)
print(f"The factorial of {number} is {result}")
```

IV.Conclusion

The conclusion of the corrected code is that it successfully defines a function add_numbers(a, b) to add two numbers and prints the result. When executed, correctly computes the sum of num1(5) and num2(3), outputting: “The result of adding 5 and 3 is 8”

The corrected code defines a recursive function factorial(n) that computes the factorial of a given number n.The base case handles the factorial of 0 as 1, while the recursive case multiplies n by the factorial of n-1. The function will correctly return the factorial of any non-negative integer.

V.References

Book
[1]

Clean Code: A Handbook of Agile Software Craftsmanship: Robert C. Martin: 9780132350884: Amazon.com: Books

Website
[2]

Cox, B. J., & Novobilski, A. (1986). *Object-oriented programming ; an evolutionary approach*. http://ci.nii.ac.jp/ncid/BA13403262

Szyperski, C. (2002). *Component software: Beyond Object-Oriented programming*.

Agha, G. (1990). Concurrent object-oriented programming. *Communications of the ACM*, *33*(9), 125–141. https://doi.org/10.1145/83880.84528

Subcommittee, P. (1979). IEEE Reliability Test System. *IEEE Transactions on Power Apparatus and Systems*, *PAS-98*(6), 2047–2054. https://doi.org/10.1109/tpas.1979.319398