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:
Villanueva, Bryan O.
Sat 12:00 PM – 4:00 PM / 4:30 PM – 8:30 PM / BSCpE 1-A

Submitted to
Engr. Maria Rizette H. Sayo
Instructor

Date Performed:
18-01-2025

Date Submitted
18-01-2025

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

- The class is a user-defined data structure that binds the data members and methods into a single unit. Class is a blueprint or code template for object creation. Using a class, you can create as many objects as you want. Classes in Python OOP provide a powerful way to structure code by encapsulating data and behavior into reusable components. Understanding how to define classes, instantiate objects, and utilize inheritance is crucial for effective programming in Python. By leveraging these concepts, developers can create modular, maintainable applications that model real-world scenarios effectively.

2. Objects

- An object is an instance of a class. It is a collection of attributes (variables) and methods. We use the object of a class to perform actions. Objects have two characteristics. They have states and behaviors (an object has attributes and methods attached to it). Attributes represent their state, and methods represent its behavior. Using its methods, we can modify its state.

3. Fields

- In Python Object-Oriented Programming (OOP), fields refer to variables that store data within a class or an object. They are also commonly referred to as attributes or data members and represent the state or properties of an object. Fields can be classified into class attributes and instance attributes, each serving specific purposes. Fields in Python OOP are essential for defining the state of objects and classes. Class attributes provide shared data across all instances, while instance attributes store unique data for each object. Understanding their differences is key to designing effective and maintainable OOP-based applications in Python.

4. Methods

- Methods in Python's Object-Oriented Programming (OOP) are functions that are defined within a class and are associated with the objects created from that class. They define the behaviors and actions that these objects can perform, encapsulating functionality and promoting code reusability. Methods are essential components of classes in Python

OOP, allowing for defining behaviors and interactions for objects. By understanding different types of methods instances, class, and static developers can create more organized and efficient code structures that model real-world entities effectively.

5. **Properties**
   - Properties in Python's Object-Oriented Programming (OOP) provide a way to manage attribute access through getter and setter methods, allowing for controlled access to class attributes. This mechanism enhances encapsulation and data integrity by enabling validation or transformation of values when attributes are accessed or modified.

III. Results

FIGURE 1: EXAMPLE OF CLASS AND OBJECT

Source: https://www.programiz.com/python-programming/object-oriented-programming

FIGURE 2: WHAT IS PYTHON?

Source: https://www.python.org/doc/essays/blurb/

Python has dynamic semantics and is a high-level, object-oriented, interpreted programming language. It is highly appealing for Rapid Application Development and for usage as a scripting or glue language to join pre-existing components because of its high-level built-in data structures, dynamic typing, and dynamic binding. Python's easy to learn syntax prioritizes readability, which lowers software maintenance costs. Python promotes code reuse and software modularity by supporting modules and packages. For all major platforms, the Python interpreter and the large standard library are freely distributable and available in source or binary form.

The edit test debug cycle is extremely quick because there is no compilation stage. Python programs are straightforward to debug since a segmentation failure is never caused by a bug or incorrect input. Rather, the interpreter raises an exception when it finds a mistake. The interpreter prints a stack trace if the application fails to catch the exception. Setting breakpoints, evaluating arbitrary expressions, inspecting local and global variables, stepping through the code line by line, and more are all made possible by a source level debugger. The debugger's introspective nature is demonstrated by the fact that it was written in Python. However, adding a few print statements to the source code is frequently the fastest way to debug a program, this straightforward method works very well due to the short edit test debug cycle.

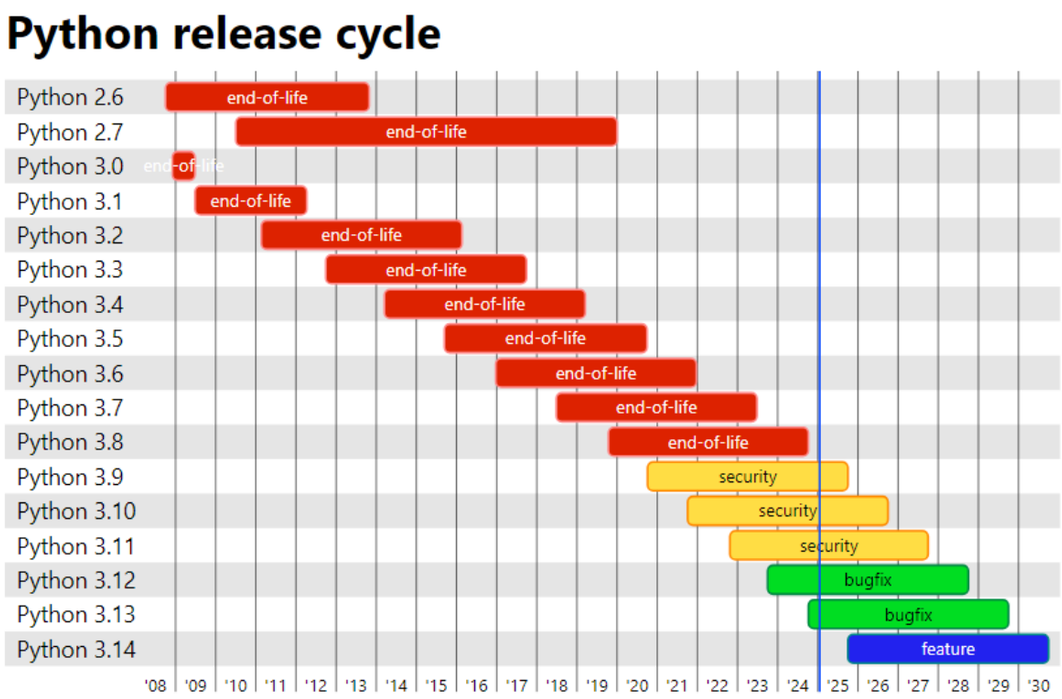

FIGURE 3: PYTHON RELEASE CYCLE

Source: https://devguide.python.org/versions/

This chart illustrates the Python release cycle, detailing the lifecycle of different Python versions from 2008 to 2030. Each version progresses through three main phases: feature updates (blue), bug fixes (green), and security updates (yellow) before reaching its end-of-life (red). For instance, Python 2.7 reached its end-of-life in 2020, marking the official end of support for the Python 2 series. The newer Python 3.x versions follow a more structured timeline, with overlapping releases ensuring a smooth transition between versions. As of 2025, Python 3.10 is in the security phase, while 3.11 is in the bug-fix phase. Python 3.12 and 3.13 are actively receiving feature updates, with Python 3.14 expected to enter this phase soon. This cycle highlights the importance of updating Python projects to supported versions to maintain security and access new features.

IV. Conclusion

To sum up, learning the basics of Python has given me a solid basis for comprehending programming ideas. One important lesson is that Python is easier to understand and less complicated than C++, which makes it a language that both novice and experienced programmers can use. Gaining knowledge of the foundations of both languages has improved my critical thinking abilities and given me insights into ways to problem-solving. The learning process emphasizes the value of comprehending fundamental programming concepts by applying past C++ knowledge to Python. These transferable abilities enhance critical thinking by enforcing a methodical investigation of problems and effective solution formulation, in addition to improving multiple language competency.

Reference

Website

“Python Classes and Objects (With examples).” https://www.programiz.com/python-programming/class

Vishal, “Classes and objects in Python,” PYnative, Feb. 24, 2024. https://pynative.com/python-classes-and-objects/

L. P. Ramos, “Python’s property(): Add Managed Attributes to Your Classes,” Dec. 15, 2024. https://realpython.com/python-property/

“OOP Terminology: class, attribute, property, field, data member,” Stack Overflow. https://stackoverflow.com/questions/16751269/oop-terminology-class-attribute-property-field-data-member

GeeksforGeeks, “Define and call methods in a Python class,” GeeksforGeeks, Feb. 14, 2024. https://www.geeksforgeeks.org/define-and-call-methods-in-a-python-class/

“What is Python? Executive Summary,” Python.org. https://www.python.org/doc/essays/blurb/

“Status of Python versions,” Python Developer’s Guide. https://devguide.python.org/versions/