Lab 9: Object Oriented Design (OOD)

- **OOD** stands for object-oriented design, also called object-oriented programming (OOP).

The benefits of OOD include:

- code reuse
- better organization

The core of OOD is to have **classes** and **objects**.

Some languages are fully OOD. Python is not one of them; Python is multi-paradigm.

A **class** represents an abstract concept of a category of things, whereas an object represents an instantiated real example of the class. For this reason, an **object** of a class is also called an instance of a class.

This whole time we have been using Objects without knowing about it. For example, a list is an object inside python. It has instance variables which are the contents of the object, a constructor (list() function) as well as class methods such as .sort(), and .remove(). When learning to create your own custom classes it may be helpful to link the new ideas you learn back to lists.

→ Class

A class represents an abstract concept of a category of things.

Syntax

```
class Class_name: #naming convention, capital first letter
    #static class variable
    staticVar = "someValue"

    #constructor
    def __init__(self, instanceVar, ...):
        self.instanceVar = instanceVar
        <body>

    #static class function, without self
    def classFunc(...):
        <body>

    #a normal function, with self
    def instanceFunc(self, ...):
        <body>
```

Explanation for the syntax above.

The big idea to cover before going on is clarifying what we mean by an *instance* of a class.

1. **class declaration** starts with keyword `class`, class name, and then an indented class body
2. **Static class variable**: a variable shared by all instances of the class. It can be viewed as an attribute associated with the class, but not with any specific instance. In short, it means the information in this variable will be the same across **all instances** of the class
3. **Instance variable: Elements in a class object specific to a certain instance of the class.** You modify an *instance variable* for a certain **instance** by referring to it by `instance_name.var` or if you are in the class, you use `self.var`
4. **Constructor**: a function you use to create an instance of the class. Students might also hear the term **initializer**
5. **class function**: a function shared by all instances of the class.
6. **instance function**: a function used by an instance; `self` is a placeholder for the instance calling the function

→ **self**

Before we get into the Code Demonstration, we need to understand about `self`. One way to think about it is with multiple instances. You must use yourself in the class so that each instance knows itself from the other. For every class, the code is the same, the data is different. `self` helps distinguish which data you are working with.

Try thinking about enemies in a video game. There might be multiple enemies of the same type, but they each have health bars at different levels. When the player shoots an enemy their health goes down, but that is specific to one instance of the enemy, so we would use `self` so only that enemy's health is updated.

Examples of using objects and classes: `String(format)`, `List(append,remove,count)`, `Dictionary(keys,items)`. You all have been using classes and objects, just not knowingly. This is exactly the purpose of OOD, hiding detail, exposing only behavioral/functional properties.

- **What is to be performed today:**

All files should be there in Laboratory/Lab9.

File Locations:

- Laboratory/Lab9/verboseClass.py
- Laboratory/Lab9/Contact.py
- Laboratory/Lab9/PhoneBook.py
- Laboratory/Lab9/main.py
- Laboratory/Lab9/contacts.txt

At the end, Check output.txt so you all can see the final output.

- verboseClass.py

This file does not need to change anything; it allows you to understand some running features. You all do not need to edit this file. You all need to have it in your repo, but it is a visual reference.

Walk through the comments and try to understand what is happening (there is code there just to have code, but the comments highlight the important stuff)

- Start working on Contact.py and PhoneBook.py

All code can be run in main.py

- **1st part of the lab:**

Start working on Contact.py.

- Constructor
- getName
- __str__
- call
- sendBirthdayText
- sendText
- UpdateNumber

- **2nd Part of the lab:**

Start working on PhoneBook.py:

- Constructor
- addContact
- getContactCount
- findContact
- groupChat
- __str__