Jennifer Grace L. Duldulao
06/01/2025
Foundations Of Programming: Python
Assignment 07
https://github.com/hihellojini/IntroToProg-Python-Mod07

Classes and Objects

Introduction

In the last module, I learned about defining functions and calling those functions in the main body of the script. I also learned to create classes to group together the functions.

In this module, we will deep dive about classes and object and learn how differences between fields, attributes, properties, and methods. We will also learn about constructor and class inheritance.

Understanding Classes and Objects

A program can have multiple statements, functions and classes depending on its complexity. The functions organize statements and classes organize functions for ease of management of a very complex program. Doing this makes the code isolated such that changing a part of the code does not affect the whole program but only on the specific function where the changes was made. It also makes the code reusable and helps simplify the code development and maintenance.

An object is an instance of a class such that its data is unique to that particular object. Each object has its own memory location in the computer to store its data.

Data variables and data constants in a class are called fields, attributes, or properties. On the other hand, functions in a class are called methods and using ***ClassName.function_name()*** if ***@staticmethod*** function decorator is included.

Classes can focus on process, presentation or data. Process and presentation classes typically has methods while data process may have attributes, constructors, and properties on top of the methods.

Attributes, Constructors, and Properties

Attributes are variables or methods that store data about its properties and behavior. Class attributes are variables that are associated with a class and are shared among all instances of a class and are defined within the class itself. Instance attributes, unlike class attributes, is not shared and is specific to a particular object or instance in a class.

Properties provide controlled access to instance attributes by defining a getter, setter, and delete methods for attributes.

Attributes are typically use for simple data storage when special logic is not required. Properties are used when there is a need to control access, validate data, or perform calculations before getting or setting an attributes value.

A constructor is a special method in a class that is called automatically when an object is created from a class. A constructor assigns values to data members of the class when an object of the class is created.

Assignment Discussion

The program for assignment 7 is very similar to assignment 6 with the addition of data classes.

Here we added classes named Person (Figure 1) and Student (Figure 2).

In class Person (Figure 1), in lines 243 to 246, I added the first and last name properties to the constructor. The keyword ***self*** is used to refer to data or functions found in an object instance, and not directly in the class. The use of "__" or two underscores before the attributes name marks the attribute as private and indicates to not access the attribute from code outside of the class. In lines 248 to 258, I added a getter and setter property function for ***student_first_name***. The ***@property*** decorator indicates the function as a getter and enables to access data. On the other hand, the ***@name_of_property.setter*** decorator indicates the function as a setter and allows to add validation and error handling.

Lines 260 to 270 is another set of getter and setter for ***student_last_name.***

Lines 272 to 274, is a special method called __str__ within a class. This provide a human-readable string representation of an object when it is converted to a string using str(). In this specific case, the method is designed to return a string that combines the object's student_first_name and student_last_name to allow to easily view the data in a simple string format.

```python
# TODO Create a Person Class
# Start of class Person
# ------------------------------------------------------------------------------------------ #
class Person:  1 usage
    """
    A class representing person data.

    Properties:
        student_first_name (str): The student's first name.
        student_last_name (str): The student's last name.

    ChangeLog:
        JDuldulao, 05/30/2025, Created the class.
    """

    # TODO Add student_first_name and student_last_name properties to the constructor
    def __init__(self, student_first_name: str = "", student_last_name: str = ""):
        self.__student_first_name = student_first_name
        self.__student_last_name = student_last_name

    # TODO Create a getter and setter for the student_first_name property
    @property  # (Use this decorator for the getter or accessor)
    def student_first_name(self):
        return self.__student_first_name.title() # formatting code

    @student_first_name.setter
    def student_first_name(self, value: str):
        if value.isalpha() or value == "":  # is character or empty string
            self.__student_first_name = value
        else:
            raise ValueError("The first name should not contain numbers.")

    # TODO Create a getter and setter for the student_last_name property
    @property
    def student_last_name(self):
        return self.__student_last_name.title()  # formatting code

    @student_last_name.setter
    def student_last_name(self, value: str):
        if value.isalpha() or value == "":  # is character or empty string
            self.__student_last_name = value
        else:
            raise ValueError("The last name should not contain numbers.")

    # TODO Override the __str__() method to return Person data
    def __str__(self):
        return f"{self.student_first_name},{self.student_last_name}"
# ------------------------------------------------------------------------------------------ #
# End of class Person
```

Figure 1. Class Person Code

Now, in class Student in Figure 2, the class Student inherited the properties and methods from class Person. This means that an object in class Student will automatically have all the attributes and behaviors of an object from class Person. On top of that, additional attributes or behaviors specific to class Student may also be added.

Below you will see that, course_name has been added using the getter and setter decorator and has inherited the attributes and behaviors of student_first_name and student_last_name from class Person.

```
278     # TODO Create a Student class the inherits from the Person class
279     # Start of class Student
280     # ------------------------------------------------------------------------------------------ #
281     class Student(Person):
282         """
283         A class representing student data.
284
285         Properties:
286             student_first_name (str): The student's first name.
287             student_last_name (str): The student's last name.
288             course_name (str): The student's course name.
289
290         ChangeLog: (Who, When, What)
291             JDuldulao, 05/30/2025, Created Class
292         """
293
294         # TODO call to the Person constructor and pass it the student_first_name and student_last_name data
295         def __init__(self, student_first_name: str = "", student_last_name: str = "", course_name: str = ""):
296             super().__init__(student_first_name=student_first_name, student_last_name=student_last_name)
297
298             # TODO add a assignment to the course_name property using the course_name parameter
299             self.__course_name = course_name
300
301         # TODO add the getter for course_name
302         @property  4 usages (2 dynamic)
303         def course_name(self):
304             return self.__course_name.title()
305
306         # TODO add the setter for course_name2
307         @course_name.setter  3 usages (2 dynamic)
308         def course_name(self, value: str):
309             self.__course_name = value
310
311         # TODO Override the __str__() method to return the Student data
312         def __str__(self):
313             return f"{self.student_first_name},{self.student_last_name},{self.course_name}"
314     # ------------------------------------------------------------------------------------------ #
315     # End of class Student
```

Figure 2. Class Student Code

In addition to the 2 data classes discussed above, we kept class FileProcessor and class IO and all the methods therein. Also, nothing was change on the main body script.

The program was tested and runs correctly whether it is run on PyCharm, Console and IDLE.

Summary

Understanding classes whether the focus is on process, presentation or data is very important to programmers because this is very helpful when the program becomes very complex. Classes helps the program or code organized and manageable. It is a bit complicated though, I would say. With all the modules we've gone through, this is so far the one where I have to read and try again multiple times before I get to understand the concept. I would say that for a programmer to be better, it takes a lot of practice to get be able to get the concept and that's something I would like to put my focus into.