Tom Lowder

11/26/2024

IT FDN 110 A

Assignment07

GitHub Repository: https://github.com/tlowder24/IntroToProg-Python-Mod07

# Classes and Objects

## Introduction

Module07 taught the distinction between statements, functions and classes with emphasis on classes, data, presentation, and processing classes. Essential topics of object-oriented programming, including constructors, attributes, properties, and inheritance were also talked about in detail. Lastly, GitHub desktop and GitHub version control and collaboration was discussed.

## Creating the program

Assignment07 was created to understand how to use classes to manage data. Assignment07 was similar to Assignment06 however it includes two added data classes to enable the program to more flexible and maintainable. The remainder of the paper describes modfications made to the program to achieve the task.

First of all, the program header was modified from the Assginment07-Starter program.

Then the Person class was created. A docstring was included for common practice purposes.

```
28  # TODO Create a Person Class
29  class Person:  1 usage
30      """
31      A class representing person data.
32
33      Properties:
34          first_name (str): The student's first name.
35          last_name (str): The student's last name.
36
37      ChangeLog:
38          - Tom Lowder, 11.25.2024: Created the class.
```

**Figure 1. Person Class**

The def __init__ constructor was used to assign initial values to the object's member variables to ensure the object starts with a defined state. In python, you can define the instance variables and assign values to them directly in the constructor without the need to declare them as class-level variables so that was done for first_name and last_name with default values of empty strings. This constructor was

automatically called when an object of the class was created. Self was specfically used to refer to data or functions found in a object instance and not directly in the class.

```
40  # TODO Add first_name and last_name properties to the constructor (Done)
41  def __init__(self, first_name: str = "", last_name: str = ""): # (Use this decorator for the setter or mutator)
42      self.first_name = first_name # set the attribute using the property to provide validation
43      self.last_name = last_name # set the attribute using the property to provide validation
```

**Figure 2. Person __init__ constructor**

Next a getter and setter was created for first_name and last_name properties. Getter functions enable you to access data while optionally applying formatting. @property decorator was used to indicate a getter funciton. Setter property functions allow you to add validation and error handling. @first_name.setter and @last_name.setter were used for setter functions.

```
45  # TODO Create a getter and setter for the first_name property (Done)
46  @property # (Use this decorator for the getter or accessor)
47  def first_name(self):
48      return self.__first_name.title() #formatting code
49
50  @first_name.setter # (Use this decorator for the setter or mutator)
51  def first_name(self, value: str):
52      if value.isalpha() or value == "": # allow alphabetic characters or the default empty string
53          self.__first_name = value
54      else:
55          raise ValueError("The first name should not contain numbers.")
56
57  # TODO Create a getter and setter for the last_name property (Done)
58  @property # (Use this decorator for the getter or accessor)
59  def last_name(self):
60      return self.__last_name.title()
61
62  @last_name.setter # (Use this decorator for the setter or mutator)
63  def last_name(self, value: str):
64      if value.isalpha() or value == "": # allow alphabetic characters or the default empty string
65          self.__last_name = value
66      else:
67          raise ValueError("The last name should not contain numbers.")
68
```

**Figure 2. Person getter and setter functions**

After that, the __str__() method was defined and is responsible for returning a human-readable string representation of an object.

```
69  # TODO Override the __str__() method to return Person data (Done)
70  def __str__(self):
71      return f'{self.first_name},{self.last_name}'
```

**Figure 3. Person __str__() method**

Then the Student class was created to directly inherit code from the Person class. Student class is a subclass of Person class by specifying Person within parentheses. This allowed the student class to

inherit all the attributes and methods from the Person class. Again, a docstring was included for common practice purposes.

```
# TODO Create a Student class the inherits from the Person class (Done)
class Student(Person): # Add a class name to indicate explicit inheritance.
    """
        A class representing student data.

        Properties:
            first_name (str): The student's first name.
            last_name (str): The student's last name.
            gpa (float): The gpa of the student.

        ChangeLog: (Who, When, What)
        Tom Lowder,11.25.2024,Created Class
        """

```

**Figure 4. Student class**

The constructor def __init__ of the Student class was extended to include additional parameter course_name. Then super().__init__ was used to call the constructor of the parent class, which initializes the first_name and last_name. The super() method connects the first and last name to the Person class, so first_name and last_name attributes and properties in the Student constructor were not needed. However, since course_name was unique to the Student class, the course_name attributes and properties were included.

```
    # TODO call to the Person constructor and pass it the first_name and last_name data (Done)
    def __init__(self, first_name: str = "", last_name: str = "",course_name: str = ""):
        super().__init__(first_name=first_name, last_name=last_name)

    # TODO add a assignment to the course_name property using the course_name parameter (Done)
        self.course_name = course_name

    # TODO add the getter for course_name (Done)
    @property  5 usages (1 dynamic)
    def course_name(self):
        return self.__course_name

    # TODO add the setter for course_name (Done)
    @course_name.setter  4 usages (1 dynamic)
    def course_name(self, value: str):
        self.__course_name = value
```

**Figure 5. Student __init__ constructor and getter and setter functions**

Lastly, another default __str__() method was defined to return a coma-separated string of the Student class's data. This overrode the __str__() method used in the Person class.

```
    # TODO Override the __str__() method to return the Student data (Done)
    def __str__(self):
        return f'{self.first_name},{self.last_name},{self.course_name}'
```

**Figure 7. Student __str__ () method**

## Testing the program

The program was tested in PyCharm and the command prompt. Tests included:

- The program takes the user's input for a student's first, last name, and course name.
- The program displays the user's input for a student's first, last name, and course name.
- The program saves the user's input for a student's first, last name, and course name to a coma-separated string file.
- The program allows users to enter multiple registrations (first name, last name, course name).
- The program allows users to display multiple registrations (first name, last name, course name).
- The program allows users to save multiple registrations to a file (first name, last name, course name).
- The program runs correctly in both **PyCharm and** from the **console or terminal**.

The program successfully ran and saved to Enrollments.json in both PyCharm and the command prompt.

I saved the program and this assignment to a new GitHub repository for module07 so others can review my work. The link to the repository is in the header of this document.

## Summary

With the resources provided in module 07 and my newfound knowledge I was able to create the Assignment07 program. The program demonstrates my understanding of how to manage data with classes.