Edwin Kintu-Lubowa

November 24, 2024

Foundations of Programming: Python

Assignment07

https://github.com/EdwinLubowa/IntroToProg-Python-Mod07

Classes, Objects and Inheritance

Introduction

This week's assignment was to create a program that uses constants, variables, and print statements to display a message about a student's registration. The assignment is very similar to Assignment06, however, it introduces two data classes: Person and Student to handle the program data. In this assignment, the focus was, still, centered around the pattern of 'Separation of Concerns' and in particular, the Data layer. In this assignment, I learned how to create a Data layer that included two classes and the methods to handle the data. In this knowledge document, I will chronicle the concepts that I learned, and how they aided and guided me to the successful completion of this assignment.

Writing the program

I started the assignment by loading the starter file into PyCharm, updated the file header with my names and the date, then renamed and saved the file. The starter file had specific "TODO" comments that offered a structured orderly guide for where to write the code.

The first thing I did was to run the code in the starter file before I made any modifications to the file, entered some data and double checked the 'Enrollments.json' file to ensure that the program was running as expected, and it was. The data in the .JSON file was saved as a comma separated list of dictionaries.

A turning point

In this assignment, the challenge was to create student objects from the Data classes to work with. In the prior assignments, I was working, first: with string data; then next I learned how to save formatted data to CSV files; followed by using looping and conditional logic to control the program flow; then data processing using lists and files; then, Advanced Collections and Error handling where I learned about lists and dictionaries, and the JSON module; and then last week's assignment that centered around Functions, classes and the 'Separation of Concerns' pattern; and until this point, I had primarily dealt with data in form of collections of lists and

dictionaries. This week marks a major turning point towards more advanced programming methods and concepts.

The Data Layer Classes

I created two classes: Person and Student, with Person as the superclass and Student as the subclass. For each class I included document strings and then created a constructor, first, for the Person class with private attributes: first_name and last_name, followed by the property methods. Below is a screen shot of the Person class definition including the document string, the constructor, and the property methods. (Figure 1)

```python

class Person:  1 usage
    """
    A class representing person data

    Properties:
    - first_name (str): The student's first name
    - last_name (str): The student's last name

    ChangeLog: (Who, When, What)
    Edwin Kintu-Lubowa,11/23/2024,Created the class.
    """

    # Create a constructor with private attributes for the first_name and last_name data
    def __init__(self, first_name: str = "", last_name: str = ""):
        self.first_name = first_name
        self.last_name = last_name

```

Figure 1: Person class definition alongside the document string and constructor

After creating the constructor for the Person class, I created the property method for the private attributes ‘first_name’ and ‘last_name’. The property getter method has some formatting using .title() to capitalize the first letter of each name whilst the rest of the letters are displayed in lower case. The setter method has includes some error checking for the name values — allowing for only alphabetic characters — and also allows for an empty string. After the property methods for the private attributes, the overrides the __str__() method with first name and last names.

“The __str__ method in Python is responsible for returning a human-readable string representation of an object. As we have seen, by default, the__str__() method in your class, Python's default implementation of __str__() will return a string that includes the object's memory address.” (Randal Root: Introduction to Programming with Python: Module 07 - Classes and Objects: Page 22 - Fall 2024)

Below is a screenshot of the property methods for the private attributes “first_name” and “last_name” for the Person class. (Figure 2)

```python
class Person:  1 usage

    # Create property getter and setter for first name using the same code as in the Student class
    @property  # (Use this decorator for the getter or accessor)  10 usages (8 dynamic)
    def first_name(self):
        return self.__first_name.title()   # formatting code

    @first_name.setter  # (use this decorator for the setter or mutator)  9 usages (8 dynamic)
    def first_name(self, value: str):
        if value.isalpha() or value == "":
            self.__first_name = value
        else:
            raise ValueError("First name should not contain numbers!")

    # Create property getter and setter for last name using the same code as in the Student class
    @property  # (Use this decorator for the getter or accessor)  10 usages (8 dynamic)
    def last_name(self):
        return self.__last_name.title()   # formatting code

    @last_name.setter  # (use this decorator for the setter or mutator)  9 usages (8 dynamic)
    def last_name(self, value: str):
        if value.isalpha() or value == "":
            self.__last_name = value
        else:
            raise ValueError("Last name should not contain numbers!")

    # Add code to inherit code from the person class
    def __str__(self):
        return f"{self.first_name},{self.last_name}"
```

Figure 2 - The getter, the setter, and __str__() methods

The Errors

Upon completion of the coding for the data layer, I run the program, it displayed the menu of choices: when I selected Option 2 - Show Current Data, the program displayed the data from the JSON file. However, when I entered new student data and tried to display it, the program threw an error message. I am trying to use indexing on an object that is a function, perhaps?! So, I had to go back to Lab03 and revisit how the FileProcessor methods were processing Student objects. Below is the screen shot of the error. (Figure 3)

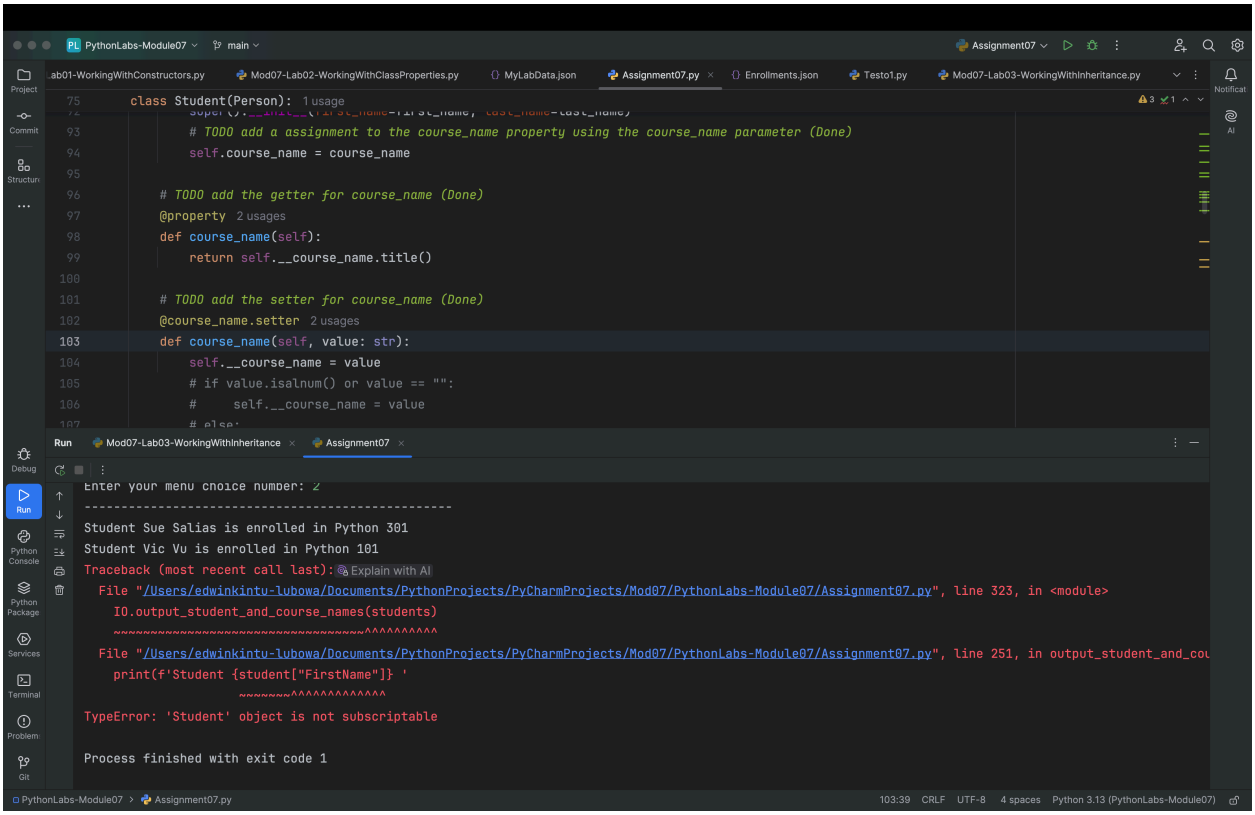

Figure 3 - error trying to process student object using 'Display Current Data'

Rewriting FileProcessor methods

After reviewing Lab03, I had to rewrite read_from_file() and write_to_file() methods. In the read_from_file() method, I defined a list attribute 'list_of_dictionary_data' to receive a list of dictionary rows from the json file. Then, iterated through the list using a for loop and converting each dictionary row into an object 'student_object' and then appending it to the list table 'student-data'. With the new code running, I commented out the old code from the starter file. The code still has the same lines of error handling as did the old code. Below is a screen shot of the updated code. (Figure 4)

```python
class FileProcessor:  2 usages
    def read_data_from_file(file_name: str, student_data: list):
            file = open(file_name, "r")

            list_of_dictionary_data = json.load(file)  # the load function returns a list of dictionary rows.
            for student in list_of_dictionary_data:
                student_object: Student = Student(first_name=student["FirstName"],
                                                  last_name=student["LastName"],
                                                  course_name=student["CourseName"])
                student_data.append(student_object)

            file.close()
        except FileNotFoundError as e:
            IO.output_error_messages( message: "Text file must exist before running this script!", e)
        except Exception as e:
            IO.output_error_messages( message: "There was a non-specific error!", e)
        finally:
            if not file.closed:
                file.close()
        return student_data

        # I will not be using the code below; it does not handle student objects
        # try:
        #     file = open(file_name, "r")
        #     student_data = json.load(file)
        #     file.close()
        # except Exception as e:
        #     IO.output_error_messages(message="Error: There was a problem with reading the file.", error=e)
        # finally:
        #     if file.closed == False:
```

Figure 4 - Code converting list of dictionaries into list of student objects

Conversely, I needed to convert the list of student objects into a list of dictionary rows before it could be written into a json file. Again, I consulted with Lab03 to learn how this was attained. In this scenario, again I defined a list attribute ‘list_of_dictionary_data’ that was set to empty. I then iterated through list ‘student_data’ and converted each student object into a dictionary row using the ‘key:value’ pairing and accessing the object data for the value bit of the pair and then appending the each row into the list attribute ‘list_of_dictionary_data’ The screen shot below shows the code that converted the student objects into a list of dictionaries (Figure 5)

```python
class FileProcessor:  2 usages
    def write_data_to_file(file_name: str, student_data: list):
        try:
            list_of_dictionary_data: list = []
            for student in student_data:  # Convert List of Student objects to list of dictionary rows.
                student_json: dict \
                    = {"FirstName": student.first_name, "LastName": student.last_name, "CourseName": student.course_name}
                list_of_dictionary_data.append(student_json)

            file = open(file_name, "w")
            json.dump(list_of_dictionary_data, file)
            file.close()
        except TypeError as e:
            IO.output_error_messages( message: "Please check that the data is a valid JSON format", e)
        except Exception as e:
            IO.output_error_messages( message: "There was a non-specific error!", e)
        finally:
            if not file.closed:
                file.close()

        # I will not be using the code below; it does not handle student objects
        # try:
        #     file = open(file_name, "w")
        #     json.dump(student_data, file)
        #     file.close()
        #     IO.output_student_and_course_names(student_data=student_data)
        # except Exception as e:
        #     message = "Error: There was a problem with writing to the file.\n"
        #     message += "Please check that the file is not open by another program."
        #     IO.output_error_messages(message=message,error=e)
```

Figure 5 - converting student objects into a list of dictionaries

---

Consolidating Error Handling

Because the Data Layer, was now performing most of the validations and error handling bits, I removed these tasks from the input_student_data() method. Below is a screen shot of the updated code. (Figure 6)

```python
class IO:  10 usages
    def input_student_data(student_data: list):
            # Input the data
            student = Student()
            student.first_name = input("What is the student's first name? ")
            student.last_name = input("What is the student's last name? ")
            student.course_name = input("Please enter the name of the course:")
            student_data.append(student)

        except ValueError as e:
            IO.output_error_messages( message: "That value is not the correct type of data!", e)
        except Exception as e:
            IO.output_error_messages( message: "There was a non-specific error!", e)
        return student_data
        # try:
        #     student_first_name = input("Enter the student's first name: ")
        #     if not student_first_name.isalpha():
        #         raise ValueError("The last name should not contain numbers.")
        #     student_last_name = input("Enter the student's last name: ")
        #     if not student_last_name.isalpha():
        #         raise ValueError("The last name should not contain numbers.")
        #     course_name = input("Please enter the name of the course: ")
        #     student = {"FirstName": student_first_name,
        #                "LastName": student_last_name,
        #                "CourseName": course_name}
        #     student_data.append(student)
        #     print()
        #     print(f"You have registered {student_first_name} {student_last_name} for {course_name}.")
        # except ValueError as e:
        #     IO.output_error_messages(message="One of the values was the correct type of data!", error=e)
        # except Exception as e:
        #     IO.output_error_messages(message="Error: There was a problem with your entered data.", error=e)
        # return student_data
```

Figure 6 - Updated input_student_data() method without validation and error handling

---

Testing Program in Terminal

After successfully running the program in PyCharm, here is a scree shot of tests in Terminal. In these tests, I started by displaying current data, then selected choice one: I entered the student data in uppercase and the title() function reformatted the data.

On the second screen, I enter no data by hitting the Enter key through all the required field and the program accepts the empty data - thereby displaying a strange custom message with the user input missing. I enter the first name containing numbers and the program catches it. (Figure 9)

```
Enter your menu choice number: 2
-------------------------------------------------
Student Sue Jones is enrolled in Python 301
Student Vic Vu is enrolled in Python 101
-------------------------------------------------


---- Course Registration Program ----
  Select from the following menu:
    1. Register a Student for a Course.
    2. Show current data.
    3. Save data to a file.
    4. Exit the program.
----------------------------------------


Enter your menu choice number: 1
What is the student's first name? MARCUS
What is the student's last name? CICERO
Please enter the name of the course: ORATIONS 101

You have registered Marcus Cicero for Orations 101.


---- Course Registration Program ----
  Select from the following menu:
    1. Register a Student for a Course.
    2. Show current data.
    3. Save data to a file.
    4. Exit the program.
----------------------------------------
```

```
    3. Save data to a file.
    4. Exit the program.
----------------------------------------


Enter your menu choice number: 1
What is the student's first name?
What is the student's last name?
Please enter the name of the course:

You have registered   for .


---- Course Registration Program ----
  Select from the following menu:
    1. Register a Student for a Course.
    2. Show current data.
    3. Save data to a file.
    4. Exit the program.
----------------------------------------


Enter your menu choice number: 1
What is the student's first name? mark112
That value is not the correct type of data!

-- Technical Error Message --
First name should not contain numbers!
Inappropriate argument value (of correct type).
<class 'ValueError'>


---- Course Registration Program ----
  Select from the following menu:
    1. Register a Student for a Course.
    2. Show current data.
    3. Save data to a file.
    4. Exit the program.
----------------------------------------


Enter your menu choice number:
```

Figure 9 - Testing Program in Terminal

Since the program was able to accept empty strings, I checked “Enrollments.json” to verify whether the initial values for the object were saved as empty strings. Below is a screen shot of the data in “Enrollments.json” (Figure 10)

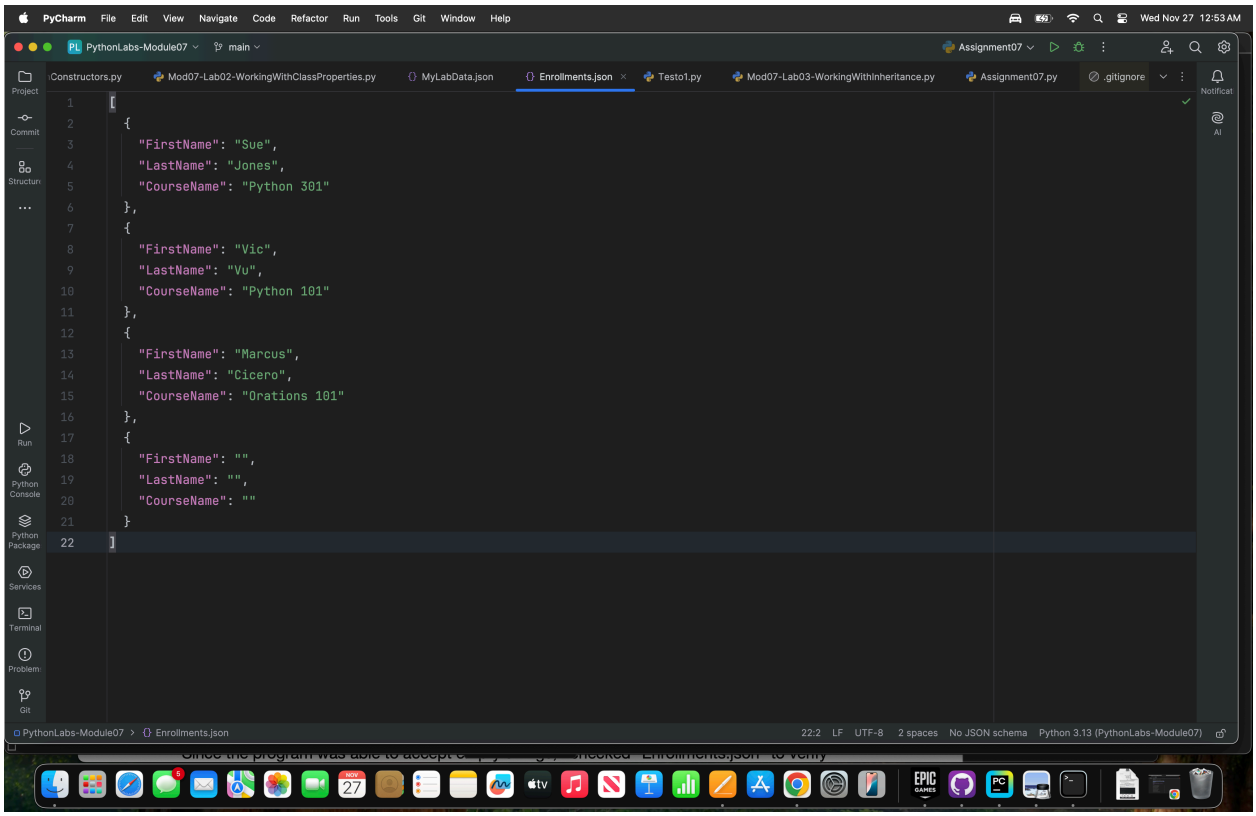

Figure 10 - results showing empty strings in "Enrollments.json"

Summary

This assignment was a confluence of many new concepts combined with some I have gotten proficient at, and others that I am still coming to grasps with. The pattern of "Separation of Concerns" played a crucial role in completing this assignment. First, there are methods in the program that I did not touch or update at all, they retained the functionality they had in the last assignment. Secondly, there are some methods that needed to be updated in order to handle the new data layer; specifically converting dictionary rows into student objects and, then, converting student objects to dictionary lists. And finally, I had to create a new data layer, complete with supporting data processing methods.

When dealing with classes, I performed minor updates to the methods in this layer: I removed the validation and error handling from the input method. In the processing layer, I made major code updates to handle the conversions between lists of dictionaries and objects, but still retaining the overall functionality for the program. And for the Data layer, I created the layer anew along with the supporting methods.

This was a good exercise and a practical display of how the program was made functional with varying degrees of focus on different layers.