Justin Spencer

June 9, 2025

IT FDN 110 A

Assignment 07

https://github.com/JSpencer45/IntroToProg-Python-Mod07

Assignment 07: Classes and Objects

Introduction

In this assignment, I utilized the concepts presented in Module 07 to create a Python program that uses statements, functions, data classes, constructors, objects, and inheritance to capture multiple user inputs for student class registration.

Creating the Code

Defining the Constants & Variables

The assignment provided the framework script that I worked from which first started with ensuring that the constants and variables were defined. The constants included the “MENU” which housed the string of characters that would be presented to the user on program start and the “FILE_NAME” string constant that would identify the file (“Enrollements.json”) that would be read and written to. The variables that I worked with were “students”, a table of the student data, and “menu_choice”, a string variable that would hold the choice made by the user after being prompted by the menu (Figure 1).

```
import json

# Define the Data Constants
MENU: str = '''
---- 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.
-----------------------------------------
'''
FILE_NAME: str = "Enrollments.json"

# Define the Data Variables
students: list = []  # a table of student data
menu_choice: str  # Hold the choice made by the user.
```

Figure 1 - Constants & Variables

Person & Student Classes and Constructors:

After ensuring that the constants and variables did not require editing, I created the “Person” and “Student” classes. Both of these classes are being used to store the data for a student’s first and last name, but the separation of these two classes is important because the Student class would inherit the presentation of the “Person” class while also handling the specific data related to a student’s registered course (Figure 2 & 3). In these classes, the “getter and setter” properties were also established so that the “getter” property would be able to access the defined data, with specific formatting, and then the “setter” property could add in data validation and error handling. This section of script also made the first_name, last_name and course_name object data to be private to signify that the data harder to access and signify that it’s for internal use only (per pg. 18 of Module07-Notes).

```
class Person:
    """
        A class representing person data.

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

        ChangeLog:
        - JSpencer, 6.4.2025: Created the class.
        - JSpencer, 6.5.2025: Edited class to make gett&setter pairs for first_name \
        and last_name private
    """
    def __init__(self, first_name: str = '', last_name: str = ''):
        self.first_name = first_name
        self.last_name = last_name

# TODO Create a getter and setter for the first_name property
    @property
    def first_name(self):
        return self.__first_name.title()
    @first_name.setter
    def first_name(self, value: str):
        if value.isalpha() or value == '':
            self.__first_name = value
        else:
            raise ValueError("Student first name should not contain numbers.")

# TODO Create a getter and setter for the last_name property
    @property
    def last_name(self):
        return self.__last_name.title()
    @last_name.setter
    def last_name(self, value: str):
        if value.isalpha() or value == '':
            self.__last_name = value
        else:
            raise ValueError("Student last name should not contain numbers.")

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

Figure 2 - Class Person

```
class Student(Person):
    """
    A collection data about students
    Properties:
     first_name (str): The student's first name.
     last_name (str): The student's last name.
     course_name(str): The name of the course.

    ChangeLog: (Who, When, What)
    JSpencer,6/4/2025,Created Class
    JSpencer,6/5/2025,Edited Class to make course_name getter & setter pair private
    JSpencer,6/6/2025, Edited if/else statement for course_name to try/else statement \
    to fix ValueError on program start, Removed erroneous space in inheritance statement \
    (first_name =first_name) to fix read error on program start
    JSpencer,6.8.2025, Edited setter for clearer code

    """
# TODO call to the Person constructor and pass it the first_name and last_name data
    def __init__(self, first_name: str = '', last_name: str = '', course_name: str = ''):
        super().__init__(first_name=first_name, last_name=last_name)
        self.course_name = course_name
# TODO add the getter for course_name
    @property
    def course_name(self):
        return self.__course_name
# TODO add the setter for course_name
    @course_name.setter
    def course_name(self, value: str):
        if value != '':
            self.__course_name = value
        else:
            raise ValueError("Course name must be alphanumeric.")

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

Figure 3 - Class Student(Person)

After creating the previous two classes, I checked the FileProcessor and IO classes for needed edits. In the FileProcessor class, the read_data_from_file and write_data_to_file functions were updated to handle converting dictionary data to student data and back again, respectively (Figures 4 & 5).

```python
class FileProcessor:
    A collection of processing layer functions that work with Json files

    ChangeLog: (Who, When, What)
    RRoot,1.1.2030,Created Class
    """
    @staticmethod
    def read_data_from_file(file_name: str, student_data: list):
        """ This function reads data from a json file and loads it into a list of dictionary rows
        then returns the list filled with student data.

        ChangeLog: (Who, When, What)
        RRoot,1.1.2030,Created function
        JSpencer,6.8.2025, Edited function to convert dict data to student object data

        :param file_name: string data with name of file to read from

        :return: list
        """

        try:
            # Get a list of dictionary rows from the data file
            file = open(file_name, "r")
            json_students = json.load(file)

            # Convert the list of dictionary rows into a list of Student objects
            # TODO replace this line of code to convert dictionary data to Student data
            for student in json_students:
                student_object: Student = Student(first_name=student["FirstName"],
                                                  last_name=student["LastName"],
                                                  course_name=student["CourseName"])
                student_data.append(student_object)
                file.close()
        except Exception as e:
            IO.output_error_messages(message="Error: There was a problem with reading the file.", error=e)
        finally:
            if not file.closed:
                file.close()
        return student_data
```

Figure 4 - Class FileProcessor - read_data_from_file function

```python
class FileProcessor:
    @staticmethod
    def write_data_to_file(file_name: str, student_data: list):
        """ This function writes data to a json file with data from a list of dictionary rows

        ChangeLog: (Who, When, What)
        RRoot,1.1.2030,Created function
        JSpencer,6.8.2025 - Edited function to convert student obj into dict

        :param file_name: string data with name of file to write to
        :param student_data: list of dictionary rows to be writen to the file

        :return: None
        """

        try:
            # TODO Add code to convert Student objects into dictionaries (Done)
            json_students: list = []
            for student in student_data:
                student_json: dict \
                    = {"FirstName": student.first_name, "LastName": student.last_name,
                       "CourseName": student.course_name}
                json_students.append(student_json)
            file = open(file_name, "w")
            json.dump(json_students, file, indent=2)
            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)
        finally:
            if not file.closed:
                file.close()
```

Figure 5 - Class FileProcessor - write_data_to_file function

The final class that was checked and edited was the IO class which would handle user input. Of the code provided in the starter document, the only code that required editing was the output_student_and_course_names and input_student_data functions. In these functions, I made the functions use student object data instead of the dictionary data (Figure 6 & 7).

```python
class IO:

    @staticmethod
    def output_student_and_course_names(student_data: list):
        """ This function displays the student and course names to the user

        ChangeLog: (Who, When, What)
        RRoot,1.1.2030,Created function
        JSpencer,6.8.2025 - Edited function to access student object data instead of dict data
        :param student_data: list of dictionary rows to be displayed

        :return: None
        """

        print("-" * 50)
        for student in student_data:

            # TODO Add code to access Student object data instead of dictionary data
            print(f'Student {student.first_name} {student.last_name} '
                  f'is enrolled in {student.course_name}')
        print("-" * 50)
```

Figure 6 - output_student_and_course_names function

```python
class IO:

    @staticmethod
    def input_student_data(student_data: list):
        """ This function gets the student's first name and last name, with a course name from the user

        ChangeLog: (Who, When, What)
        RRoot,1.1.2030,Created function
        JSpencer,6.8.2025 - Edited function to use student objects instead of dict objects
        :param student_data: list of dictionary rows to be filled with input data

        :return: list
        """

        try:
            first_name = input("Enter the student's first name: ")
            if not first_name.isalpha():
                raise ValueError("The last name should not contain numbers.")
            last_name = input("Enter the student's last name: ")
            if not last_name.isalpha():
                raise ValueError("The last name should not contain numbers.")
            course_name = input("Please enter the name of the course: ")

            # TODO Replace this code to use a Student objects instead of a dictionary objects
            student = Student(first_name=first_name,
                              last_name=last_name,
                              course_name=course_name)
            student_data.append(student)
            print()
            print(f"You have registered {first_name} {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 7 - input_student_data function

Testing the Code:

After creating the main body of the script, I tested the code in PyCharm and in the Windows Command Shell. I made sure to test the script to handle multiple inputs from the user that would be stored and written to the "Enrollments.json" file present in the relative location of the script (Figure 8 & 9). I was also able to successfully test the script in the CMD Shell with an additional user input (Figure 10).

```
---- 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
Enter the student's first name: Justin
Enter the student's last name: Spencer
Please enter the name of the course: Python 100

You have registered Justin Spencer for Python 100.


---- 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
Enter the student's first name: Soobin
Enter the student's last name: Jeong
Please enter the name of the course: Python 100

You have registered Soobin Jeong for Python 100.
```

```
Enter your menu choice number: 3
-------------------------------------------------
Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Justin Spencer is enrolled in Python 100
Student Soobin Jeong is enrolled in Python 100
-------------------------------------------------


---- 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: 2
-------------------------------------------------
Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Justin Spencer is enrolled in Python 100
Student Soobin Jeong is enrolled in Python 100
-------------------------------------------------
```

Figure 8 - PyCharm test of multiple user entries

[{"FirstName": "Bob", "LastName": "Smith", "CourseName": "Python 100"}, {"FirstName": "Sue", "LastName": "Jones", "CourseName": "Python 100"}]


```
[
  {
    "FirstName": "Bob",
    "LastName": "Smith",
    "CourseName": "Python 100"
  },
  {
    "FirstName": "Sue",
    "LastName": "Jones",
    "CourseName": "Python 100"
  },
  {
    "FirstName": "Justin",
    "LastName": "Spencer",
    "CourseName": "Python 100"
  },
  {
    "FirstName": "Soobin",
    "LastName": "Jeong",
    "CourseName": "Python 100"
  }
]
```


Figure 9 - Before & after PyCharm test results in the Enrollments.json file

```
---- 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
Enter the student's first name: Vic
Enter the student's last name: Vu
Please enter the name of the course: Python 100

You have registered Vic Vu for Python 100.


---- 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: 3
--------------------------------------------------
Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Justin Spencer is enrolled in Python 100
Student Soobin Jeong is enrolled in Python 100
Student Vic Vu is enrolled in Python 100
--------------------------------------------------
```


```
[
  {
    "FirstName": "Bob",
    "LastName": "Smith",
    "CourseName": "Python 100"
  },
  {
    "FirstName": "Sue",
    "LastName": "Jones",
    "CourseName": "Python 100"
  },
  {
    "FirstName": "Justin",
    "LastName": "Spencer",
    "CourseName": "Python 100"
  },
  {
    "FirstName": "Soobin",
    "LastName": "Jeong",
    "CourseName": "Python 100"
  },
  {
    "FirstName": "Vic",
    "LastName": "Vu",
    "CourseName": "Python 100"
  }
]
```


Figure 10 - CMD Shell test & results in Enrollments.json file

Summary:

With the successful test of the script in both PyCharm and the CMD Shell, I believe I have displayed my understanding of the material presented in Module 07.