Richard Eidswick

11/28/2025

Foundations of Programming: Python - IT FDN 110 A

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

Assignment 07

Introduction

For this week's assignment, I continued to build on the Python program that created an interactive registration menu that outputs information to a JSON file. The key difference between this week's version and the previous version was converting the list of students read from the JSON file into individual objects created from classes. Also included was an inherited class to expand the capabilities of the program and it was tested in both PyCharm and the command prompt.

New Topics

Classes and Objects

This assignment went further into the capabilities of classes by adding in class inheritance and the benefits of objects. Objects are stand-alone instances of a class. Multiple objects can be created from a single class. As in this assignment, each student was an object created from the Student class. Class inheritance has a general class that can then be called from other classes to increase modularity. In this assignment, the Person class was the general that defined the first and last name of a person. However, not all Person objects will be student and enrolled in a course. The Student class inherited properties from the Person class but also had independent, specific properties – course.

Program

This assignment used an existing starter file that had all the necessary variables, including the "MENU" global constant. The things that needed to be added were two classes – Person and Student. Then the code had to be modified in the function that handled interacting with the file and the display portions. The Json files cannot interact directly with objects. The data read from the file came in as a list of dictionaries that needed to be converted to a list of objects. The write to file section needed to convert the list of objects back into a list of dictionaries.

The Person class had the properties for first name and last name. It did not need to have the course information since not all people take courses. A nice addition to the Person class was the inclusion of error catching within the class that raised the value error. This removes the error catching from the IO class (Figure 1).

```python
class Person:
    """
    # TODO Add first_name and last_name properties to the constructor
    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):...

    # 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):
        if value.isalpha() or value == "":
            self.__last_name = value
        else:
            raise ValueError("Last name must be alphabetic characters.")

    # TODO Override the __str__() method to return Person data
    def __str__(self):...
```

Figure 1: Person Class

The Student Class was created similarly to the Person Class and inherited some of the properties from the Person Class (Figure 2).

```python
class Student(Person):
    """
    A class representing student data.
    """
    def __init__(self, first_name: str="", last_name: str="", course_name: str=""):
        # TODO call to the Person constructor and pass it the first_name and last_name data
        super().__init__(first_name=first_name, last_name=last_name)
        # TODO add a assignment to the course_name property using the course_name parameter
        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):
        self.__course_name = value

    # 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 2: Student Class

Next were the changes to the file processing class. The read data from file processing section needed to be updated to convert the list of JSON data into the list of objects (Figure 3).

```python
class FileProcessor:
    def read_data_from_file(file_name: str, student_data: list):
        """
        file = _io.TextIOWrapper

        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 row in json_students:
                student_obj: Student = Student(first_name=row["FirstName"],
                                               last_name=row["LastName"],
                                               course_name=row["CourseName"])
                student_data.append(student_obj)

        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:
                file.close()

        return student_data
```

Figure 3: Converting from dictionaries to objects.

Similarly, the write to file needed to do the reverse to be in the correct format for the JSON file to take in the data. The list of objects needed to be converted into a list of dictionaries (Figure 4).

```
class FileProcessor:
    def write_data_to_file(file_name: str, student_data: list):

        try:
            # TODO Add code to convert Student objects into dictionaries (Done)
            list_of_dictionary_data = []
            for student in student_data:
                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, indent=3)
            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 file.closed == False:
                file.close()
```

Figure 4: Converting from objects to dictionaries.

Lastly, the input section of the IO class was updated to remove the error catching that is now built into the Person class and correctly call out the object properties (Figure 5).

```
class IO:
    def input_student_data(student_data: list):
        """
        try:
            # TODO Replace this code to use a Student objects instead of a dictionary objects
            student = Student()
            student.first_name = input("Please enter the first name: ")
            student.last_name = input("Please enter the last name: ")
            student.course_name = input("Please enter the course name: ")
            student_data.append(student)
            print()
            print(f"You have registered {student.first_name} {student.last_name} for {student.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 5: Display code updated for object properties.

Testing

With all the changes implemented, the code needed to be tested in PyCharm and then in the command prompt. Starting with running the program in PyCharm and displaying the data (Figure 6).

```
---- 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 Vic Vu is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Bob Ross is enrolled in Art450
-------------------------------------------------
```

Figure 6: Displaying data from the file.

Then adding a new student to the data (Figure 7).

```
---- 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
Please enter the first name: Tester
Please enter the last name: Test
Please enter the course name: Test100

You have registered Tester Test for Test100.
```

Figure 7: Adding a user.

Next, the data is saved to the file and the output is checked in the JSON file (Figure 8).

Figure 8: JSON file with updated list.

The test is successful and the next test is in the command prompt (Figure 9).

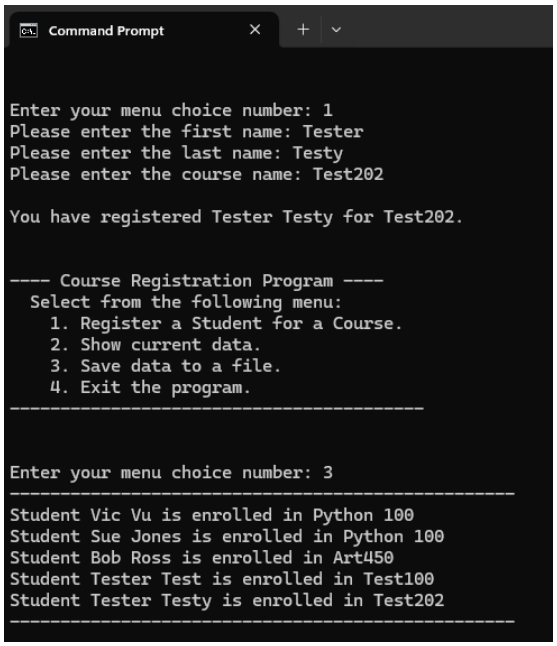

Figure 9: Running the tests in command prompt.

Summary

In this assignment, I went through the process of adding classes, using inheritance, and creating objects from classes. This helps increase modularity and scalability. I combined this with the previous error handling to make a robust program It ran successfully in both PyCharm and command prompt.