Geoff DuBuque

August 13, 2024

IT FDN 110 A Su 24

Assignment 07

https://github.com/gdubuque/IntroToProg-Python-Mod07.git

Assignment 07 – Classes and Objects

Introduction

This module's programming assignment keeps all the same functionality as the previous program but adds a set of data classes to keep track of the student data. Instead of saving the data in the program in a list of dictionaries, we use an instance of a class, aka an object, and save the objects into a list. Each object has attributes to save data for that object. This assignment adds a Person class with a *first_name* and *last_name* attribute, and a Student class that inherits from the Person class giving it all the same methods and attributes as the Person class and adds a *course_name* attribute. Because this script is very similar to the last assignment, I will only highlight the differences added to this script. All these programming ideas and syntax can be found in the module's notes, labs, and starter Python file, referenced at the end of this paper.

Creating the Python script

Data Classes

Within the program's data layer, I added the data classes. I first created the Person class starting with a document string and the *__init__* method to initialize the class attributes when a Person object is created. I then created the class property functions to set and get the attributes using the class's private attributes. Each setter property also has a simple format validation check that raises a *ValueError* if the data is not in the desired format. At the end of the class, we override the *__str__* method to return a CSV formatted string of the attributes. See **Figure 1** on the next page for the Person data class.

```python
# Data Classes------------------------------------------------------------- #
1 usage  gdubuque
class Person:
    """
    A data class to represent a person.

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

    ChangeLog: (Who, When, What)
    G.DuBuque, 2024/08/12, Created Class
    """
    gdubuque
    def __init__(self, first_name: str = '', last_name: str = ''):
        self.first_name = first_name
        self.last_name = last_name

    gdubuque
    @property
    def first_name(self):
        return self.__first_name.title()    # Optionally format as title case (GD)

    gdubuque
    @first_name.setter
    def first_name(self, value: str):
        if value.isalpha() or value == '':  # Check for format errors (GD)
            self.__first_name = value
        else:
            raise ValueError('First name must not contain numbers.')

    gdubuque
    @property
    def last_name(self):
        return self.__last_name.title()  # Optionally format as title case (GD)

    gdubuque
    @last_name.setter
    def last_name(self, value: str):
        if value.isalpha() or value == '':  # Check for format errors (GD)
            self.__last_name = value
        else:
            raise ValueError('Last name must not contain numbers.')

    gdubuque
    def __str__(self):
        """
        :return: The Person's attributes in a CSV string
        """
        return f'{self.first_name},{self.last_name}'
```

Figure 1: Person Class

After the Person class I created the Student class, which inherits from the Person class. The Student *__init__* method uses the *super.()__init()* syntax to initialize the *first_name* and *last_name* attributes, and then initializes the *course_name* attribute unique to the student class. I then create the getter and setter properties for the *course_name* attribute, with a check to make sure the *course_name* value is formatted in the desired way, and override the *__str__* method, also using the *super()* method, to return a CSV string of the Student attributes. Finally, I created my own method *get_data_dict()*, to return a dictionary of the Student attributes to format the data in a JSON format. This will make the code shorter later in the program when saving the data to the .json file. See **Figure 2** below for the Student data class.

```python
class Student(Person):
    """
    A data class to represent a Student. Inherits from Person.

    Properties:
    first_name (str): The student's first name.
    last_name (str): The student's last name.
    course_name (str): The course name the student is registered for.

    ChangeLog: (Who, When, What)
    G.DuBuque, 2024/08/12, Created Class
    """

    gdubuque
    def __init__(self, first_name: str = '', last_name: str = '', course_name: str = ''):
        super().__init__(first_name, last_name)
        self.course_name = course_name

    gdubuque
    @property
    def course_name(self):
        return self.__course_name.title()   # Optionally format as title case (GD)

    gdubuque
    @course_name.setter
    def course_name(self, value: str):
        value_list = value.split()  # Slit course name into title and number (GD)
        # Check for format errors, must be in the form of "Name Number" (GD)
        if len(value_list) == 2 and value_list[0].isalpha() and value_list[1].isnumeric():
            self.__course_name = value
        else:
            raise ValueError("Course name must be in the form of 'Name Number'")

    gdubuque
    def __str__(self):
        """
        return: The Student's attributes in a CSV string
        """
        return f'{super().__str__()},{self.course_name}'

    1 usage (1 dynamic)  gdubuque
    def get_data_dict(self):
        """
        return: The Student's attributes in a dictionary.
        For working with JSON data.
        """
        return {"FirstName": self.first_name, "LastName": self.last_name, "CourseName": self.course_name}
```

Figure 2: Student Class

Processing Layer Updates

A few changes need to be made to some functions in the Processing layer now that we are saving the data as Students objects instead of dictionaries. The first change is in the *read_data_from_file()* function. When we read the data from the .json file into a list, we need to use that data to create Student objects that will be added to the *students* list. I used a *for* loop to create a Student object from each dictionary in the JSON data using their Keys to set the attributes. I also added a *ValueError* exception to catch any of the format errors when creating the Student objects. See **Figure 3** below for updates to the *read_data_from_file()* function.

```
128     @staticmethod
129     def read_data_from_file(file_name: str, student_data: list):
130         """ This function reads data from a json file and loads it into a list of Student objects
131
132         ChangeLog: (Who, When, What)
133         R.Root, 2030/01/01, Created function
134         G.DuBuque, 2024/08/12, Updated function to use a list of Student objects
135
136         :param file_name: string data with name of file to read from
137         :param student_data: list of Student objects created from JSON file data (GD)
138
139         :return: list of Student objects (GD)
140         """
141
142         try:
143             file = open(file_name, "r")
144             student_json_data = json.load(file)
145             file.close()
146
147             # For each student in the JSON list, create a Student object from the JSON data
148             # and add the Student object to the students list. (GD)
149             for student in student_json_data:
150                 student_data.append(Student(student['FirstName'],
151                                             student['LastName'],
152                                             student['CourseName']))
153
154         except ValueError as e:     # Check for Value Errors when creating Student objects (GD)
155             IO.output_error_messages( message: "There was problem with the format of the data!", e)
```

Figure 3: Read Data from File Updates

The next update was to the *write_data_to_file()* function, where we need to create a dictionary of data from each Student object, add that dictionary to a list, and save that list to the .json file. Here I use a *for* loop and my *get_data_dict()* method to easily get the dictionary data from each Student object. See **Figure 4** on the next page for updates to the *write_data_to_file()* function.

```python
    @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 Student objects

        ChangeLog: (Who, When, What)
        R.Root, 2030/01/01, Created function
        G.DuBuque, 2024/08/12, Updated function to use a list of Student objects

        :param file_name: string data with name of file to write to
        :param student_data: list of Student objects to write data to JSON file (GD)

        :return: None
        """

        try:
            student_json_data: list = []    # Temporary list of student JSON data (GD)
            # Get the dictionary data from each Student object in the list and add it
            # to the JSON list (GD)
            for student in student_data:
                student_json_data.append(student.get_data_dict())

            file = open(file_name, "w")
            json.dump(student_json_data, file)
```

Figure 4: Write Data to File Updates

Presentation Layer Updates

A couple of functions in the Presentation layer needed to be updated as well. The *output_student_and_course_names()* function needed to be updated to use the Student object attributes instead of dictionary keys. See **Figure 5** below.

```python
    @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)
        R.Root, 2030/01/01, Created function
        G.DuBuque, 2024/08/12, Updated function to use a list of Student objects

        :param student_data: list of Student objects (GD)

        :return: None
        """

        print("-" * 50)
        for student in student_data:
            print(f'Student {student.first_name} {student.last_name} '
                  f'is enrolled in {student.course_name}')
        print("-" * 50)
```

Figure 5: Output Student Data Updates

The *input_student_data()* function needed to be updated to create a Student object instead of a dictionary, and the formatted string to use the Student object's attributes instead of dictionary keys. See **Figure 6** below.

```
280     @staticmethod
281     def input_student_data(student_data: list):
282         """ This function gets the student's first name and last name, with a course name from the user
283
284         ChangeLog: (Who, When, What)
285         R.Root, 2030/01/01, Created function
286         G.DuBuque, 2024/08/12, Updated function to use a list of Student objects
287
288         :param student_data: list of Student objects (6D)
289
290         :return: list of Student objects (6D)
291         """
292
293         try:
294             student_first_name = input("Enter the student's first name: ")
295             student_last_name = input("Enter the student's last name: ")
296             course_name = input("Please enter the name of the course: ")
297
298             student = Student(student_first_name, student_last_name, course_name)
299
300             student_data.append(student)
301             print()
302             print(f"You have registered {student.first_name} {student.last_name} for {student.course_name}.")
303         except ValueError as e:  # Check for Value Errors when creating Student objects (6D)
304             IO.output_error_messages( message: "There was problem with the format of the data!", e)
```

Figure 6: Input Student Data Updates

The Main body of the script remains unchanged.

Testing the Program

I tested the program by first choosing menu choice 2 to display the existing data in the file and make sure it is displayed in the correct format. Then I use menu choice 1 a couple times to add some new data. I used choice 2 again to make sure the new data was added. I then use choice 3 to save the data and make sure it is displayed correctly. I ended the program with choice 4.

I tested the error handling by changing the file name in the FILE_NAME variable and entering numbers in the first and last name inputs to make sure the corresponding errors are displayed. The program is run in both PyCharm and Windows PowerShell. The results of the program running correctly in PyCharm is shown on the next page.

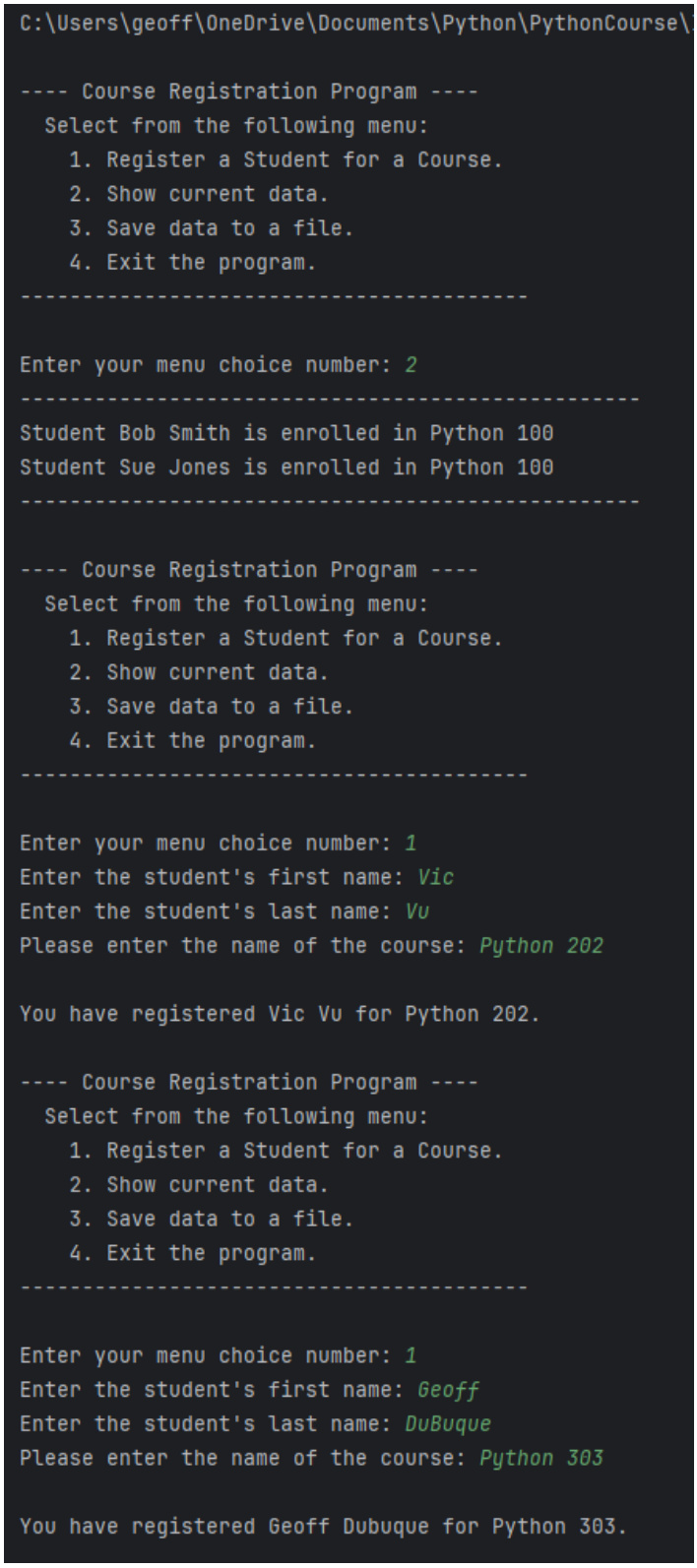

Figure 7: Program Ran in PyCharm

Summary

This programming assignment introduced data classes as a way of working with data using objects of those classes. The data classes had attributes to store data for each object, and getter and setter property functions as an abstract way to get and set an object's attributes. Formatting checks were added to the setter properties, and a custom method was added to get an object's data as a dictionary, simplifying some of the code in other parts for the script.

References

Randal Root. 2024. Mod07-Notes.docx. IT FDN 110 A Su 24: Foundations of Programming: Python. University of Washington, Professional & Continuing Education

Randal Root. 2024. Mod07-Assignment.docx. IT FDN 110 A Su 24: Foundations of Programming: Python. University of Washington, Professional & Continuing Education

Randal Root. 2024. Assignment07-Starter.py. IT FDN 110 A Su 24: Foundations of Programming: Python. University of Washington, Professional & Continuing Education

Randal Root. 2024. Mod07-Lab03-WorkingWithInheritance.py. IT FDN 110 A Su 24: Foundations of Programming: Python. University of Washington, Professional & Continuing Education