Miles Devine

8/15/2024

IT FDN 110

Assignment 07- Classes and Objects

GitHub Link: https://github.com/miles-devine/IntroToProg-Python-Mod07

Introduction

This week's module took us deeper into classes and what can be done with them. We reviewed different types of classes, class inheritance, constructors, attributes, and properties. We also covered version control using Git and learned about the built-in support for GitHub within PyCharm.

The Assignment

This week's assignment was very similar to Assignment 06. We were given the task of creating a Python program that uses constants, variables, print statements, flow control, conditional logic, dictionaries, and lists to display user entered information and then save that information to a JSON file. The catch was that in Assignment 07 we used a set of data classes.

As usual, our script began with a header containing information that is important for the user. This included the title of the program, the description of the program, and a change log (Fig. 1). We went on to import the json package and define the constants of the program.

```
# ------------------------------------------------------------------------------------------ #
# Title: Assignment07
# Desc: This assignment demonstrates using data classes with structured error handling
# Change Log: (Who, When, What)
#   Miles Devine, 8/11/2024, Created script
#   <Your Name Here>,<Date>,<Activity>
# ------------------------------------------------------------------------------------------ #
import json
```

Fig. 1- (Header with important information and imported module)

Next, we needed to create a Person class. This was a class that would represent a person's data. We created a constructor for the class and made sure to create getters and setters for the properties so that we could access the attributes and control the values that were assigned to them. After this we went on to override the __str__() method for the Person class so that it would return Person data. Similar steps were followed to create the Student class which would inherit properties from the Person class as well as add its own property (Fig. 2). We added a setter and getter for the new property, worked to override the __str__() method inherited from the Person class and moved on.

```python
class Student(Person):

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

    4 usages (2 dynamic)
    @property  #course_name getter
    def course_name(self):
        return self.__course_name

    2 usages (2 dynamic)
    @course_name.setter
    def course_name(self, value: str):
        self.__course_name = value
        #     self.course_name = value
        # else:
        #     raise ValueError("The course name should be alphanumeric. ")

    def __str__(self):
        return f'{super().__str__()},{self.course_name}'
```

Fig. 2-(Student class which inherits from the Person class)

For the processing class we had to change our read method to make sure that when reading the json data from our file we loaded it into a temporary variable holding a list of objects and appended it to a table of student data (Fig. 3).

```python
class FileProcessor:
    @staticmethod
    def read_data_from_file(file_name: str, student_data: list[Student]):
        """ This function reads data from a json file and loads it into a list of dictionary rows

        ChangeLog: (Who, When, What)
        RRoot,1.1.2030,Created function

        :param file_name: string data with name of file to read from
        :param student_data: list of dictionary rows to be filled with file data

        :return: list
        """
        file_data = []
        file = None
        try:
            file = open(file_name, "r")  #Enrollments.json
            file_data = json.load(file)  #Loads contents of Enrollments.json to the file_data variable
            file.close()  #Saves and closes the Enrollments.json file
        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()
        for row in file_data:  #Converts the json data into a list of objects
            student_data.append(
                Student(row["FirstName"], row["LastName"], row["CourseName"])
            )
        return student_data
```

Fig. 3-(Converting list of dictionaries to list of Student objects)

We adjusted our write to file method and made sure that the data that was converted back from a list of objects to a list of dictionaries. We accomplished this by using another temporary variable and a for loop to work through the student data table and convert the rows back to a dictionary and dump it into our json file. Further down in the IO class we needed to make changes to the input_student_data method. Since we covered error handling in our Student class we didn't need to handle errors after each variable. We assigned the user's inputted values to a Student object and appended it to the student_data table (Fig 4.).

```python
@staticmethod
def input_student_data(student_data: list[Student]):
    """ 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

    :param student_data: list of dictionary rows to be filled with input data

    :return: list
    """

    try:
        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("What course are you registering the student for? ")
        student = Student(student_first_name,
                          student_last_name,
                          student_course_name)
        student_data.append(student)
    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
```

Fig. 4-(input_student_data method that appends to student_data table)

I tested the rest of the program in PyCharm as well as the command console to make sure that the program would work regardless of what software the user is using.

Summary

This assignment was fun as well as challenging. I think it did a great job of tying together what we've been learning and I felt like some concepts started to make more sense as I worked through the assignment. I'm looking forward to Assignment08.