Jeff Howard

11/27/2024

Foundations Of Programming: Python

Assignment 07

GitHub

Classes and Objects

Introduction

In this paper, I will review the steps I took to create a program in Python that presents a menu of options for a user to select, and depending on the user's selection, will either gather their data, present the data back, save the data to a file, or exit the program. This program builds on our 6th assignment by using data classes and objects.

Topic 1: Defining Variables and Constants

In this assignment, we wanted to present a menu of options for the user to choose from - to be used in our conditional logic later in the code. The first constant is the menu of options and the second constant is the JSON file name where we will be saving the data.

The variables for this project are user inputted values - which are the first and last name of the student, and the list of students. (Fig 1)

```
# 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"

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

Figure 1: Defined Constants & Variables

Topic 2: Class Definitions

SubTopic 2a: Data Class

Module 7 introduced the concept of classes and properties for the program data - the students and their courses. I created 2 classes, the first one is called 'Person' and contains the student's first and last name attributes. After initializing, I added the getter and setter methods below which hold the private attributes of first and last name. One advantage to this approach is that we can add the validation once, and it will be applied anywhere the person is called in the program (see "ValueError" in fig 2a)

```python
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:
        - JHoward, 11.24.2024: Created the class.
    """
    def __init__(self, first_name: str = '', last_name: str = ''):
        self.first_name = first_name
        self.last_name = last_name

    @property
    def first_name(self):
        return self.__first_name.title()

    @first_name.setter
    def first_name(self, value: str):
        if value.isalpha() or value == "":  # is character or empty string
            self.__first_name = value
        else:
            raise ValueError("The first name should not contain numbers.")
```

Figure 2a: Data Class: Person

After repeating these steps for the last name attribute, I then created the 'Student' class, which adds the course name attribute to our object. 'Student' is defined as a child of 'Person' by passing the 'Person' in the parenthesis following the class name. As a child of 'Person', 'Student' inherits the first and last name attributes and I call them using the 'Super()' method (see fig 2b). I applied the same approach to the properties with the exception of excluding the validation on the course name.

```python
class Student(Person):
    """
    A class representing student data including the first name, last name and course name.
    Properties:
        first_name (str): The student's first name.
        last_name (str): The student's last name.
        course_name (str): The student's course name'.
    ChangeLog: (Who, When, What)
    JHoward,11/24/2024,Created Class
    """

    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

    @property
    def course_name(self):
        return self.__course_name.title()

    @course_name.setter
    def course_name(self, value: str):
        self.__course_name = value

    def __str__(self):
        result = super().__str__()
        return f'{result},{self.last_name}'
```

Figure 2b: Data Class: Student

SubTopic 2b: File Processor Class

Similar to module 6, the file processing class (FileProcessor) is designed to read the json file and write our student data back to the json file. Because we are now using objects and classes to define our data, I revised the code to read and write a list of student objects instead of lists of dictionary objects. The 'for' loop in figure 2c handles the conversion from json dictionary to list of objects.

```python
@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

    ChangeLog: (Who, When, What)
    JHoward,11/24/2024,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
    """
    try:
        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:  # Convert the list of dictionary rows into Student objects
            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
```

Figure 2c: Data processing: reading data function

The code is also adjusted so that we convert the list back to dictionaries so that we can write the data in the correct format back to our json file (see ‘for’ loop in fig 2d).

```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 dictionary rows

    ChangeLog: (Who, When, What)
    JHoward,11/24/2024,Created function

    :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:
        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()
```

Figure 2d: Data processing: writing data

SubTopic 2c - Presentation (Input/Output) Class

The first function defined in the IO class was input_student_data. If the user wants to register a student for a course by selecting option 1 in our menu, the program calls input_student_data to gather the user's first, last and course name. Since our student name validation is in the new student data class, we no longer need this piece in our input_student_data function . (Fig 3a)

```python
@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)
    JHoward,11/24/2024,Created function
    :param student_data: list of dictionary rows to be filled with input data
    :return: list
    """
    try:
        student = Student()
        student.first_name = input("Enter the student's first name: ")
        student.last_name = input("Enter the student's last name: ")
        student.course_name = input("Please enter the name of the course: ")
        student_data.append(student)
        print()
        print(f"You have registered {student.first_name} {student.last_name} for {student.course_name}.")
    except Exception as e:
        IO.output_error_messages(message="Error: There was a problem with your entered data.", error=e)
    return student_data
```

Figure 3a: Input/Output: gathering user data

The next two functions present the menu of options to the user and gather their inputted selection. The input_menu_choice function also has custom error handling based on the user input. (Fig 3b)

```python
@staticmethod
def output_menu(menu: str):
    """ This function displays the menu of choices to the user
    :return: None
    """
    print()
    print(menu)
    print()  # Adding extra space to make it look nicer.

@staticmethod
def input_menu_choice():
    """ This function gets a menu choice from the user
    :return: string with the users choice
    """
    choice = "0"
    try:
        choice = input("Enter your menu choice number: ")
        if choice not in ("1","2","3","4"):  # Note these are strings
            raise Exception("Please, choose only 1, 2, 3, or 4")
    except Exception as e:
        IO.output_error_messages(e.__str__())  # Not passing the exception object to avoid the t
    return choice
```

Figure 3b : Menu and User Selection

The next and final two functions in the class definitions section are outputs presented back to the user; output_error_messages & output_student_courses. Output_error_messages is used to print a custom error message to the user if they occur. Output_student_courses prints the current student data in the

program. The print() statement in output_student_and_course_names is updated to print the list of objects. (Fig 3c)

```python
@staticmethod
def output_error_messages(message: str, error: Exception = None):
    """ This function displays a custom error messages to the user
    ChangeLog: (Who, When, What)
    JHoward,11/24/2024,Created function
    :param message: string with message data to display
    :param error: Exception object with technical message to display
    :return: None
    """
    print(message, end="\n\n")
    if error is not None:
        print("-- Technical Error Message -- ")
        print(error, error.__doc__, type(error), sep='\n')

@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)
    JHoward,11/24/2024,Created function
    :param student_data: list of dictionary rows to be displayed
    :return: None
    """

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

Figure 3c : Outputs

Topic 3: Conditional Logic

After defining the class functions, the main body of our program uses a while loop to cycle through each condition of the menu. First, I used FileProcessor.read_data_from_file to open the json and read the current file data into the program. Next I entered the appropriate function for each condition of the if statement and based on the user's entry. (Fig 4)

```python
#Load current json data
students = FileProcessor.read_data_from_file(file_name=FILE_NAME, student_data=students)

# Repeat the follow tasks
while True:
    IO.output_menu(menu=MENU)
    menu_choice = IO.input_menu_choice()

    if menu_choice == "1":  # Get new student data
        IO.input_student_data(student_data=students)
        continue

    elif menu_choice == "2":  # Show all current data
        IO.output_student_courses(student_data=students)
        continue

    elif menu_choice == "3":  # Save data in a file
        FileProcessor.write_data_to_file(file_name=FILE_NAME, student_data=students)
        continue

    elif menu_choice == "4":  # End the program
        break  # out of the while loop

print("Program Ended")
```

Figure 4 : Main Body

Topic 4: Testing code

SubTopic 4a - Testing in PyCharm

To test that the script is working properly in PyCharm and the console, I ran the script and followed each of the prompts to ensure the program was saving the user-inputted data to the Enrollemnts.json file. (Fig 8 and Fig 9)

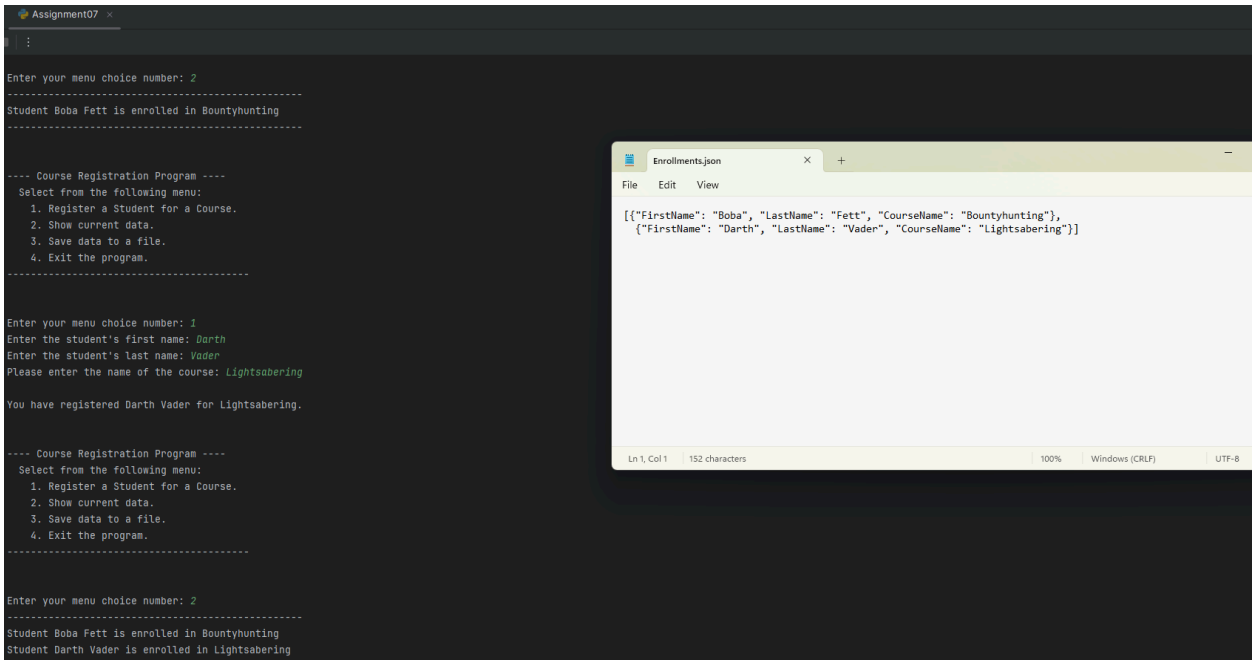

Figure 5: Testing in PyCharm

SubTopic 5b - Testing in Console

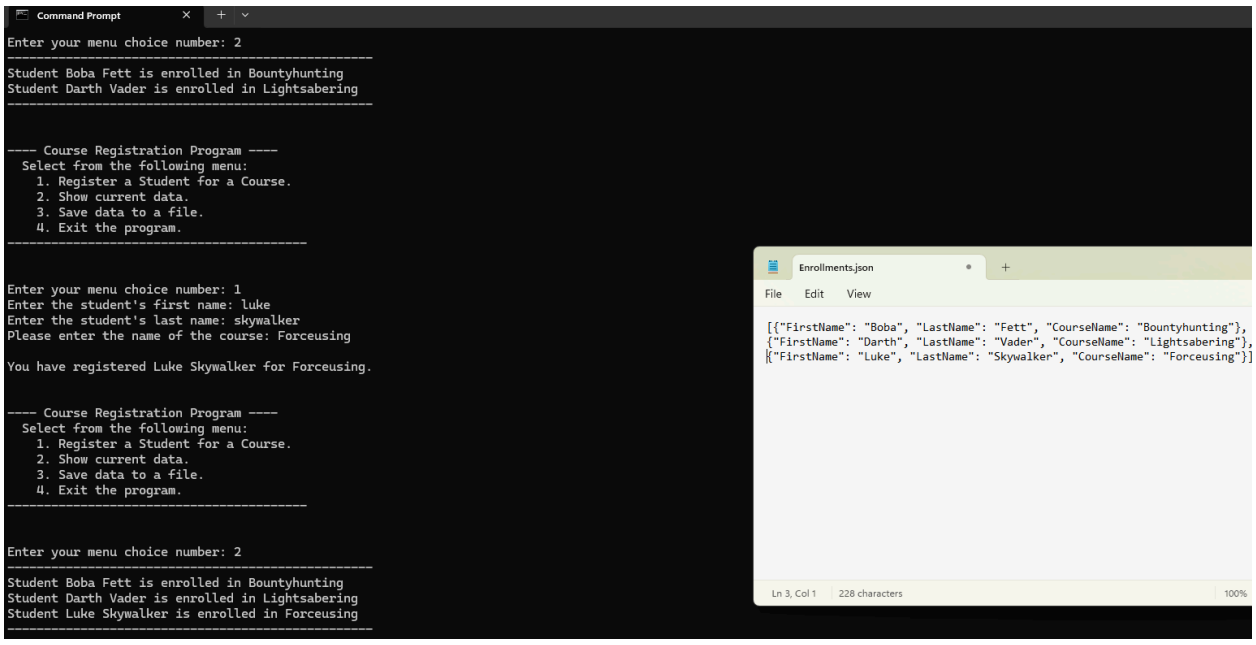

Figure 5b: Testing in Console

Summary

In summary, I reviewed the steps I took to create a program in Python that opens and reads a json file's data into the program, presents a menu of options for a user to select, and depending on the user's

selection, will either gather their data, present the data back, save the data to a file, or exit the program.