Name: Reilly Thomer
Date: November 20th, 2024
Course: IT FDN 110 A – Foundations of Programming: Python
Assignment: Module 6 Assignment
https://github.com/RobotNerd417/IntroToProg-Python-Mod06.git

Module 6 Assignment

Introduction

This week I created a Python Program that demonstrates reading from a json file, creating lists, taking user input, exception handling, and storing formatted data in a json file. In this script the user is presented a menu of options for student course registration. Depending on which menu option the user selects they are then prompted to enter student data, are given a display of all of the registered students data, or are able to save all student data to a json file, as well as exit the program. This script incorporates elements such as classes and functions. Exception handling is also included in this script. The following sections contain information regarding the Python Scripts construction and testing.

Data Constants & Variables Declaration

I began writing my python script in a new python file in PyCharm. Following the header section came the setup section of the script containing the data constants and variable declaration. (Figure 1.1) When initializing these constants, the names were written in all capitals to indicate they are a constant and are not to be changed. It can also be noted that the menu string is formatted as a multi-line string. On line 9 it can be seen that json was imported into the script.

```
9   import json
10
11  # Define the Data Constants
12  MENU: str = '''
13  ---- Course Registration Program ----
14    Select from the following menu:
15      1. Register a Student for a Course.
16      2. Show current data.
17      3. Save data to a file.
18      4. Exit the program.
19  -----------------------------------------
20  '''
21  # Define the Data Constants
22  FILE_NAME: str = "Enrollments.json"
```

Figure 1.1: Data Constant Declarations

Once the Data Constant section was complete the Data Variables were then declared. (Figure 1.2) Note that the names of the data variables are all lowercase to indicate they are variables. The Data Variable types declared included strings, and lists.

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

Figure 1.2: Data Variable Initialization

IO Class

Following the declaration of constants and variables came the declaration of the first class named *IO*. In this class there were numerous functions written for collecting input from the user as well as outputting data to the user. In Figure 2.1 below you can see the declaration of class IO as well as a docstring explaining what is contained in the class. Note that all functions within the class are static methods.

Output Error Messages

Seen below in Figure 2.1 is the script for the output error message function. Note after the declaration of the function is a docstring explaining the function. This will remain a pattern throughout this script. The Output Error Messages function takes in arguments of the non-technical error message as well as the error. These arguments are then processed, and the error message is printed to the user to view.

```
29    class IO:
30        """
31        A collection of functions that obtain user input,
32        display data and strings, as well as contains an error handler
33        ChangeLog: Reilly Thomer, 11.19.2024, Created Class
34        """
35        @staticmethod
36        def output_error_messages(message: str, error: Exception = None):
37            """ This function prints an error message out
38            ChangeLog: Reilly Thomer, 11.19.2024, Created Function
39            :param message: string data that is to be displayed to the user
40            :param error: exception that was raised to the error handler
41            :return: back to the main script
42            """
43            print("-" * 50)
44            print("Error: " + message)
45            print("-- Technical Error Message -- ")
46            print(error.__doc__)
47            print(error.__str__())
48            return
```

Figure 2.1: Output Error Message Function

Output Menu

The following function is the output menu function. This function takes in a string, the menu string, and prints the string to the user. It then returns back to the main script upon completion.

```
50      @staticmethod
51      def output_menu(menu: str):
52          """ This function displays the menu to the user
53          ChangeLog: Reilly Thomer, 11.19.2024, Created Function
54          :param menu: menu to be displayed to the user
55          :return: back to the main script
56          """
57          print(menu)
58          return
```

Figure 2.2: Output Menu Function

Input Menu Choice

The following function is the input menu choice function. This function prompts the user for a menu choice input. It then returns the user menu choice back to the main script.

```
61      def input_menu_choice():
62          """ This function obtains the user input choice to the menu
63          ChangeLog: Reilly Thomer, 11.19.2024, Created Function
64          :return: the menu choice of the user
65          """
66          return input("What would you like to do: ")
```

Figure 2.3: Input Menu Choice Function

Input Student Data

The following function is the input student data function. This function prompts the user for the student data. This is completed with a series of while loops prompting the user for input. The first while loop is executed until the user inputs a valid student first name. The second while loop is then entered and is executed until the user inputs a valid student last name. This is done by completing error handling only allowing letters as valid input for the students first and last name.Next the user is prompted to enter the course name. Upon entry the student data just entered by the user is appended to the students list. Lastly the function returns to the main script.

```
68      @staticmethod
69      def input_student_data(student_data: list):
70          """ This function obtains the student data through user prompts
71           and appends them to the student data list
72           ChangeLog: Reilly Thomer, 11.19.2024, Created Function
73          :param student_data: list of student data that is added to
74          :return: back to the main script
75          """
76          while(True):
77              student_first_name = input("Enter the student's first name: ")
78              if not student_first_name.isalpha():
79                  IO.output_error_messages( message: "The first name should not contain numbers.",Exception())
80              else:
81                  break
82          while(True):
83              student_last_name = input("Enter the student's last name: ")
84              if not student_last_name.isalpha():
85                  IO.output_error_messages( message: "The last name should not contain numbers.", Exception())
86              else:
87                  break
88          course_name = input("Please enter the name of the course: ")
89          student = {"FirstName": student_first_name,
90                     "LastName": student_last_name,
91                     "CourseName": course_name}
92          student_data.append(student)
93          return
```

Figure 2.4: Input Student Data Function

Output Student Courses

The following function is the output student courses function. This function displays the student data I the students list for the user to view. This is completed with a for loop printing out each individual student's data in the students list. Lastly the function returns to the main script.

```
95      @staticmethod
96      def output_student_courses(student_data: list):
97          """ This function prints all student data in the list for the user to view
98          ChangeLog: Reilly Thomer, 11.19.2024, Created Function
99          :param student_data: list of student data that is printed to the user
100         :return: back to the main script
101         """
102         for student in student_data:
103             print(f'Student {student["FirstName"]} '
104                   f'{student["LastName"]} is enrolled in {student["CourseName"]}')
105         print("-" * 50)
106         return
```

Figure 2.5: Output Student Courses Function

File Processor Class

The second class in this script is the File Processor class. In this script is the declaration of functions that read and write to a json file. Below are explanations of the functions.

Read Data from File

In the *read_data_from_file* function the data from the passed in *file_name* is read. This read data is saved to the students list declared at the beginning of the program. This was done inside a try block by first opening the json file in read mode ('r') followed by loading the json file into the string variable file_data. Next a for loop was used to iterated between the items in the json file and add each students data in the file to a list. By using a for loop the student data can be processed one row at a time and then added to the students list utilizing the append( ) call. Once all rows/items in the json file were read and added to the students list the loop ended and the json file was closed.

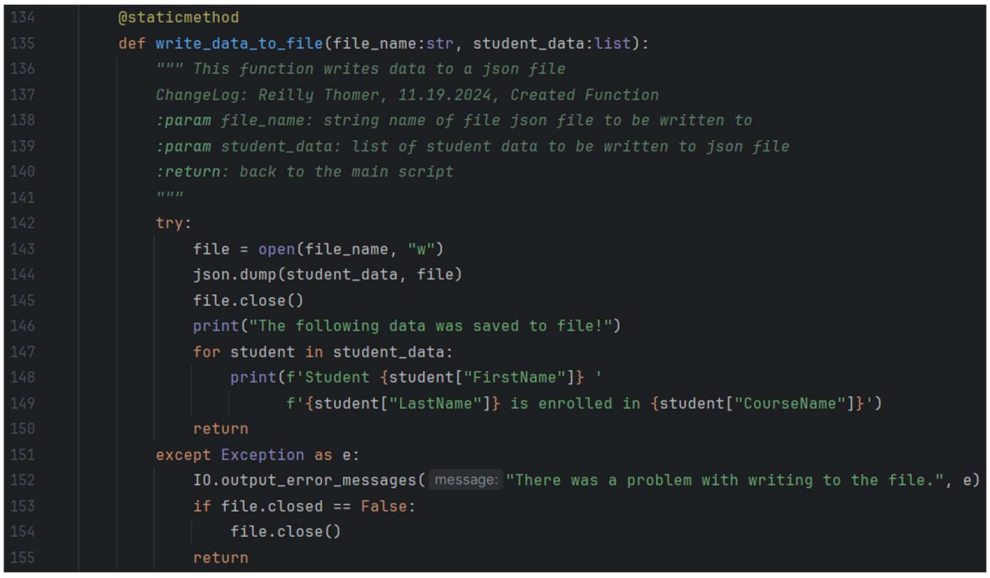
```python
class FileProcessor:
    """
    A collection of functions that perform file processing
    ChangeLog: Reilly Thomer, 11.19.2024, Created Class
    """
    @staticmethod
    def read_data_from_file(file_name: str, student_data: list):
        """ This function reads all the data from the Json File
        ChangeLog: Reilly Thomer, 11.19.2024, Created Function
        :param file_name: string name for json file to be read
        :param student_data: list of student data that is to be writen to
        :return: back to the main script
        """
        try:
            file = open(file_name, "r")
            file_data = json.load(file)
            for item in file_data:
                student_data.append(item)
            return
        except Exception as e:
            IO.output_error_messages( message: "There was a problem with reading the file.", e)
        finally:
            if file.closed == False:
                file.close()
            return
```

Figure 3.1: Read Data from file function for reading Enrollments.json

When reading from the json files many errors can occur. Some of these errors include the "Enrollments.json" file not existing and improper data in the json file, as well as others. Seen above in Figure 3.1 such errors are handled in the error handler. The function for outputting error messages was called. Then in the finally statement the file is ensured that it is closed before returning to the main script.

Write Data to File

Below in Figure 3.2 is the script contained within the *write_data_to_file* function. This function takes in the file name as well as the student data list to be written to the json file. To write to the json file the *file_name* entered is used to open the file followed by the utilization of the dump function to add the students list data to the file. Following the writing to the json file the students list is then printed out for the user to view. This function also includes error handling to ensure that the file is closed. Additionally, the function *output_error_message* is used to print the error message to the user.



```
134     @staticmethod
135     def write_data_to_file(file_name:str, student_data:list):
136         """ This function writes data to a json file
137         ChangeLog: Reilly Thomer, 11.19.2024, Created Function
138         :param file_name: string name of file json file to be written to
139         :param student_data: list of student data to be written to json file
140         :return: back to the main script
141         """
142         try:
143             file = open(file_name, "w")
144             json.dump(student_data, file)
145             file.close()
146             print("The following data was saved to file!")
147             for student in student_data:
148                 print(f'Student {student["FirstName"]} '
149                       f'{student["LastName"]} is enrolled in {student["CourseName"]}')
150             return
151         except Exception as e:
152             IO.output_error_messages( message: "There was a problem with writing to the file.", e)
153             if file.closed == False:
154                 file.close()
155             return
```

Figure 3.2: Write Data to file function for writing to Enrollments.json

Main Script: Data Entry & Processing

After scripting of the classes and functions described above came the main body of the script that utilizes these functions. The beginning of the main body starts with the reading of the json file declared at the beginning of the script. This is completed by utilizing the *read_data_from_file* function passing in the FILE_NAME and the students list. Once the data from the json file was read then a while loop was used containing processes to be executed depending on the users input. The while loop is excited when the user is to select menu choice 4 as seen on line 176 and 177 in Figure 3.1 below. Depending on the user's menu choice the appropriate case statement will be true and the appropriate code will be executed. At the beginning of the while loop are the calls of functions *output_menu* which is passed the MENU constant, and the function *input_menu_choice* is also called. The *input_menu_choice* function returns a string that is the users menu choice so as seen on line 165 in Figure 3.2 the variable *menu_choice* is set equal to *input_menu_choice*.

```
158  #Beginning of the Main Body
159  FileProcessor.read_data_from_file(FILE_NAME,students)
160
161  while (True):
162      #Outputs User Menu
163      IO.output_menu(MENU)
164      #Obtains User Menu Choice
165      menu_choice = IO.input_menu_choice()
166      # Process input user data
167      if menu_choice == "1":  # This will not work if it is an integer!
168          IO.input_student_data(students)
169      # Present the current data
170      elif menu_choice == "2":
171          IO.output_student_courses(students)
172      # Save the data to a file
173      elif menu_choice == "3":
174          FileProcessor.write_data_to_file(FILE_NAME,students)
175      # Stop the loop
176      elif menu_choice == "4":
177           break  # out of the loop
178      #In case any other manu number is entered
179      else:
180          print("Please only choose option 1, 2, or 3")
181  print("Program Ended")
```

Figure 3.1: While Loop Declaration and Menu User Input

Menu Option 1

Above in Figure 3.1 lines 167 and 168 are to be executed when menu option 1 is selected. Inside the if statement is a call of the *input_student_data* function that was declared previously. After the *input_student_data* function is executed the program pointer returns to the top of the while loop.

Menu Option 2

Above in Figure 3.1 lines 170 and 171 are to be executed when menu option 2 is selected. Inside the elif statement is a call of the *ouput_student_data* function that was declared previously. The students list is passed into the *output_student_data* function. The same as mentioned previously when discussing menu option 1 once the function and else statements are completed the program pointer returns to the while loop, then prompting the user to input another menu option.

Menu Option 3

Above in Figure 3.1 lines 173 and 174 are to be executed when menu option 3 is selected. Inside the elif statement is a call of the *write_data_to_file* function that was declared previously. The FILE_NAME and students data is passed into the function. After the *write_data_to_file* function is executed the program pointer returns to the top of the while loop.

Menu Option 4 & Other

The last of the menu choices is menu option 4. The script for this portion is seen in Figure 3.1 lines 176 and 177. Inside the elif statement is the call break. When this line is reached the while loop is broken and the print statement on line 181 is executed. On lines 179 and 180 is the script that is executed when the user enters any other menu choice than 1 through 4.

Testing & Running Script

After writing this assignment's script it was then tested. The script was first run in PyCharm as seen below in Figure 4.1 and Figure 4.2. The user was prompted with a menu and appropriate actions were completed by the script based on the user's input. In this case after selecting the menu option 1 the user then proceeded to enter the students name, and course name. After selecting option 2 all student data in the students list was printed out for the user to view. Upon selecting option 3 the student data in the list students was then saved to an Enrollments json file. All the student enrollment information in the students list was also printed out for the user to view. Finally, upon selecting option 4 the program ended. Additionally, if any other menu option input was sent in by the user the script informs the user to only choose option 1, 2, or 3. The user is then prompted to enter another menu option.

```
---- 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.
----------------------------------------

What would you like to do: 1
Enter the student's first name: James
Enter the student's last name: Jackson
Please enter the name of the course: Math 417

---- 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.
----------------------------------------

What would you like to do: 2
Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Sarah Smith is enrolled in Physics 161
Student James Jackson is enrolled in Math 417
----------------------------------------------
```

Figure 4.1: Script Testing reading current student list and adding user input

The student data in the students list was printed out upon selection of menu option 3 as seen in Figure 4.2.

```
---- 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.
----------------------------------------

What would you like to do: 3
The following data was saved to file!
Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Sarah Smith is enrolled in Physics 161
Student James Jackson is enrolled in Math 417

---- 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.
----------------------------------------

What would you like to do: 4
Program Ended
```

Figure 4.2: Enrollment.json file saved to folder

Seen below in Figure 4.3 is the student enrollment data stored in the json file.

Figure 4.3: Enrollment.json file saved to folder

Command Prompt

After the script ran successfully in PyCharm it was then tested when ran through Command Prompt. (Figure 4.4 and Figure 4.5) When running the script in command prompt it also functioned as desired.

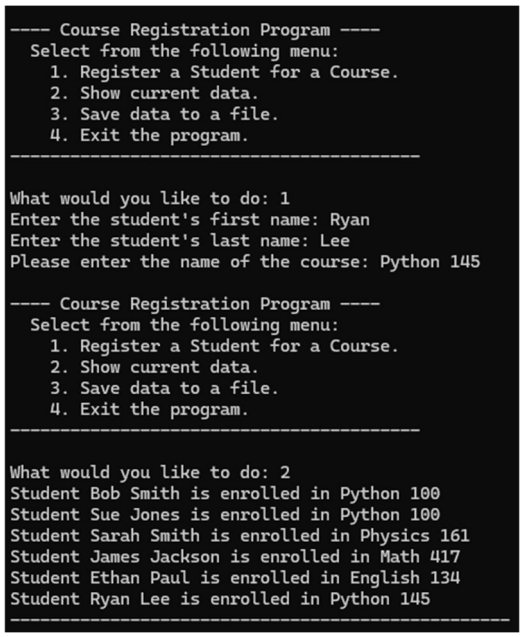


Figure 4.4: Script Ran through Command Prompt Menu Options 1 & 2

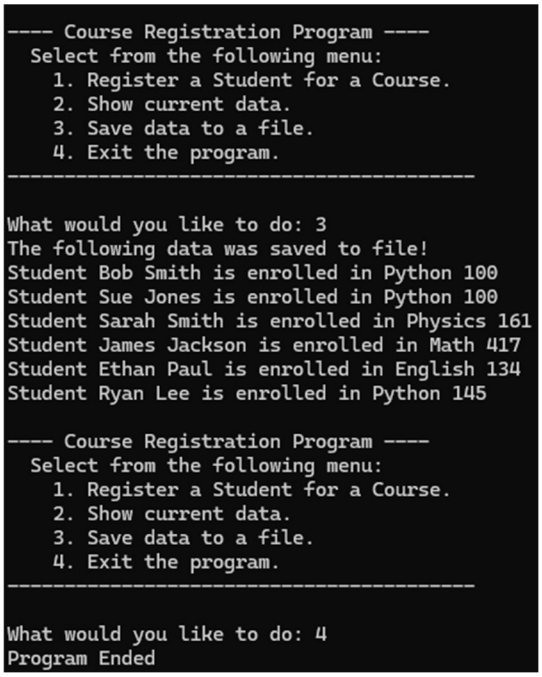


Figure 4.8: Script Ran through Command Prompt Menu Options 3 & 4

Summary

Writing this script was a very helpful exercise building upon the learning from the previous module. I continue to find declaring the names of Constants in all capital letters as well as variables in lowercase with words separated by underscores very helpful when managing data. This assignment was a good example of utilizing classes and functions as well as loops and conditional logic and case statements. Additionally, reading the json file and later writing to the json file was a great example for using lists, and utilizing the method dump( ). It was also a great example of utilizing an error handling function throughout the code. The videos for this assignment were very informative and easy to follow when learning more about functions. Overall, this exercise was a great example of utilizing classes with functions and a main script utilizing previously declared functions.