Name: Reilly Thomer
Date: November 24th, 2024
Course: IT FDN 110 A – Foundations of Programming: Python
Assignment: Module 7 Assignment
https://github.com/RobotNerd417/IntroToProg-Python-Mod07.git

Module 7 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, objects, 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  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.

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

Figure 1.2: Data Variable Initialization

Person Class

Below in Figure 2.1 is the declaration and Person Class. Inside the Persons class are the constructors used to initialize the object's attributes. On line 40 is the call to the constructor method __init__ which takes two parameters: first_name and last_name both of which also have default values of empty strings. After, on lines 45 and lines 56 are the property declarations of the first_name and last_name. The setters for first_name and last_name were written on lines 49 and 60. Line 67 the method __str__ was written to return the Persons name as a formatted string that is used later in the script.

```
27      # Manager of Data ------------------------------------ #
28      class Person:
29          """
30          A class representing person data
31
32          Properties:
33              first_name (str): The student's first name.
34              last_name (str): The student's last name.
35
36          ChangeLog:
37              Reilly Thomer,11/24/2024,Created Class
38
39          """
40          def __init__(self,first_name: str = "", last_name: str=""):
41              self.first_name = first_name  # set the attribute using the property to provide validation
42              self.last_name = last_name  # set the attribute using the property to provide validation
43
44          @property
45          def first_name(self):
46              return self.__first_name.title()
47
48          @first_name.setter
49          def first_name(self, value: str):
50              if value.isalpha() or value == "":  # allow characters or the default empty string
51                  self.__first_name = value
52              else:
53                  raise ValueError("The first name should not contain numbers.")
54
55          @property
56          def last_name(self):
57              return self.__last_name.title()  # Optional formatting code
58
59          @last_name.setter
60          def last_name(self, value: str):
61              if value.isalpha() or value == "":  # allow characters or the default empty string
62
63                  self.__last_name = value
64              else:
65                  raise ValueError("The last name should not contain numbers.")
66
67          def __str__(self):
68              return f'{self.first_name},{self.last_name}'
```

Figure 2.1: Person Class

Student Class

```
70      # Student Class that Inherits Person ------------------------------------- #
71      class Student(Person):
72          """
73          A class representing student data.
74
75          Properties:
76              first_name (str): The student's first name.
77              last_name (str): The student's last name.
78              course_name (str): The course name.
79
80          ChangeLog:
81              Reilly Thomer,11/24/2024,Created Class
82
83          """
84          def __init__(self,first_name: str ="", last_name: str = "",course_name: str = ""):
85              super().__init__(first_name=first_name, last_name=last_name)
86              self.course_name = course_name  # set the attribute using the property to provide validation
87
88          @property
89          def course_name(self):
90              return self.__course_name.title()
91
92          @course_name.setter
93          def course_name(self, value: str):
94                  self.__course_name = value
95
96          def __str__(self):
97              return f'{self.first_name},{self.last_name},{self.course_name}'
```

Figure 3.1: Student Class

Above is the declaration of the Class Student. On line 71 with the declaration of the class is the specification that the Students class inherits code from the Person class. This is done by specifying Person within parentheses after the class name. This indicates explicit inheritance which means that Student inherits all attributes and methods from Person. Within the class is the property declaration and setter for the course_name attribute on lines 89 and 93. In the initialization constructor the method super() is used to connect the first and last name to the Person class. Because of this the first and last name attributes are no longer needed in the student constructor only the course_name attribute is needed.

File Processor Class

The third 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 list_student_data. Next a for loop was used to iterate between the items in the json file and add each students data in the file as student_objects 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.

```
# Processing ------------------------------------- #
class FileProcessor:
    """
    A collection of processing layer functions that work with Json files

    ChangeLog: (Who, When, What)
    RRoot,1.1.2030,Created Class
    """
    @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)
        RRoot,1.1.2030,Created function
        Reilly Thomer, 11/24/2024, Converted list of dictionaries to list of student objects

        :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_student_data = json.load(file) # the load function returns a list of dictionary rows.
            for student in list_student_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 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 4.1: 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 4.1 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 4.2 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. A new list

save_student_data is appended to in a for loop. Inside the for loop each student (student class object) in the student_data list is iterated over creating individual student_json dictionaries appended to the save_student_data list. Once all the students are appended to the list the json file is opened. 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.

```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)
        RRoot,1.1.2030,Created function
        Reilly Thomer,11/24/2024,Converted code to use student objects instead of dictionaries

        :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:
            save_student_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}
                save_student_data.append(student_json)

            file = open(file_name, "w")
            json.dump(save_student_data, file)
            file.close()
            print(f"Data saved to {file_name}:")
            for student in student_data:
                print(student)

        except Exception as e:
            message = "Error: There was a problem with writing to the json 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 4.2: Write Data to file function for writing to Enrollments.json

IO Class

Following the declaration of File Processor class came the declaration of the 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 5.1 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 5.1 is the script for the output error message function. Note after the declaration of the function is a docstring explaining the function. This will remains 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.

```
175     # Presentation ------------------------------------ #
176     class IO:
177         """
178         A collection of presentation layer functions that manage user input and output
179 
180         ChangeLog: (Who, When, What)
181         RRoot,1.1.2030,Created Class
182         RRoot,1.2.2030,Added menu output and input functions
183         RRoot,1.3.2030,Added a function to display the data
184         RRoot,1.4.2030,Added a function to display custom error messages
185         Reilly Thomer,11/24/2024,Converted code to use student objects instead of dictionaries
186 
187         """
188 
189         @staticmethod
190         def output_error_messages(message: str, error: Exception = None):
191             """ This function displays the a custom error messages to the user
192 
193             ChangeLog:
194             RRoot,1.3.2030,Created function
195 
196             :param message: string with message data to display
197             :param error: Exception object with technical message to display
198 
199             :return: None
200             """
201             print(message, end="\n\n")
202             if error is not None:
203                 print("-- Technical Error Message -- ")
204                 print(error, error.__doc__, type(error), sep='\n')
```

Figure 5.1: 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.

```python
    @staticmethod
    def output_menu(menu: str):
        """ This function displays the menu of choices to the user

        ChangeLog:
        RRoot,1.1.2030,Created function

        :return: None
        """
        print()  # Adding extra space to make it look nicer.
        print(menu)
        print()  # Adding extra space to make it look nicer.
```

Figure 5.2: 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.

```python
    @staticmethod
    def input_menu_choice():
        """ This function gets a menu choice from the user

        ChangeLog:
        RRoot,1.1.2030,Created function

        :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 e to avoid the technical message

        return choice
```

Figure 5.3: 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. On line 270 an object of type student is declared. The user input is then taken followed by an if statement. Note the input take for the student first name is assigned to the student object by calling student.first_name. This same action is completed when obtaining the students last name.

The first if statement in the try loop is true if the user inputs a valid student first name. The second if statement is executed if 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 and is assigned to the student objects attribute 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.

```
256     @staticmethod
257     def input_student_data(student_data: list):
258         """ This function gets the student's first name and last name, with a course name from the user
259 
260         ChangeLog: (Who, When, What)
261         RRoot,1.1.2030,Created function
262         Reilly Thomer,11/24/2024,Converted code to use student objects instead of dictionaries
263 
264         :param student_data: list of dictionary rows to be filled with input data
265 
266         :return: list
267         """
268 
269         try:
270             student = Student()
271             student.first_name = input("Enter the student's first name: ")
272             if not student.first_name.isalpha():
273                 raise ValueError("The last name should not contain numbers.")
274             student.last_name = input("Enter the student's last name: ")
275             if not student.last_name.isalpha():
276                 raise ValueError("The last name should not contain numbers.")
277             student.course_name = input("Please enter the name of the course: ")
278             student_data.append(student)
279             print(f"You have registered {student.first_name} {student.last_name} for {student.course_name}.")
280         except ValueError as e:
281             IO.output_error_messages(message="One of the values was the correct type of data!", error=e)
282         except Exception as e:
283             IO.output_error_messages(message="Error: There was a problem with your entered data.", error=e)
284         return student_data
```

Figure 5.4: 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. Note that the list is a list of student objects so in order to obtain the students first and last name as well as their course name they must be called using the student.first_name for example. After each students information is printed the function returns to the main script.

```
238     @staticmethod
239     def output_student_and_course_names(student_data: list):
240         """ This function displays the student and course names to the user
241
242         ChangeLog:
243         RRoot,1.1.2030,Created function
244         Reilly Thomer,11/24/2024,Converted code to use student objects instead of dictionaries
245
246         :param student_data: list of dictionary rows to be displayed
247
248         :return: None
249         """
250
251         print("-" * 50)
252         for student in student_data:
253             print(f'Student {student.first_name} {student.last_name} is enrolled in {student.course_name}')
254         print("-" * 50)
```

Figure 5.5: Output Student Courses Function

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 and classes. 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 exited when the user is to select menu choice 4 as seen on line 316 and 317 in Figure 6.1 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 298 in Figure 6.1 the variable *menu_choice* is set equal to *input_menu_choice*.

```
288  # Start of main body
289  # Extract the data from the file
290  students = FileProcessor.read_data_from_file(file_name=FILE_NAME, student_data=students)
291
292  # Present and Process the data
293  while (True):
294
295      # Present the menu of choices
296      IO.output_menu(menu=MENU)
297
298      menu_choice = IO.input_menu_choice()
299
300      # Input user data
301      if menu_choice == "1":  # This will not work if it is an integer!
302          students = IO.input_student_data(student_data=students)
303          continue
304
305      # Present the current data
306      elif menu_choice == "2":
307          IO.output_student_and_course_names(student_data=students)
308          continue
309
310      # Save the data to a file
311      elif menu_choice == "3":
312          FileProcessor.write_data_to_file(file_name=FILE_NAME, student_data=students)
313          continue
314
315      # Stop the loop
316      elif menu_choice == "4":
317          break  # out of the loop
318      else:
319          print("Please only choose option 1, 2, or 3")
320
321  print("Program Ended")
```

Figure 6.1: While Loop Declaration and Menu User Input

Menu Option 1

Above in Figure 6.1 lines 301 and 302 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 6.1 lines 306 and 307 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 6.1 lines 311 and 312 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 6.1 lines 316 and 317. Inside the elif statement is the call break. When this line is reached the while loop is broken and the print statement on line 319 is executed. On lines 318 and 319 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 and Command Prompt. Seen below in Figure 7.1 and Figure 7.2 are the outputs when executing the program in Command Prompt. 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.

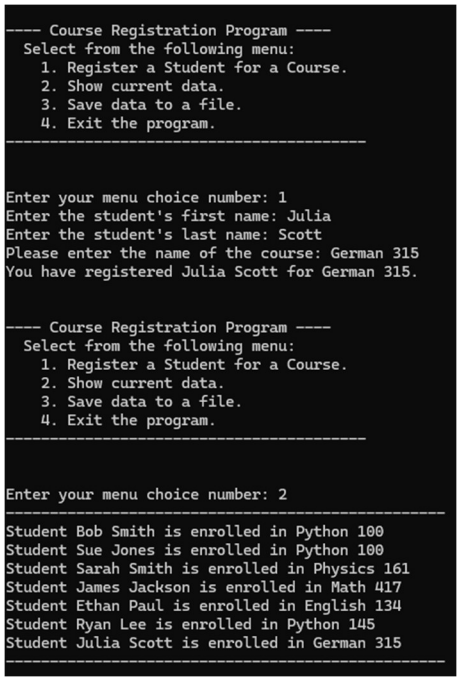

Figure 7.1: Script Testing reading current student list and adding user input

The student data in the students list was printed out upon selection of menu option 3 as seen in Figure 7.2.

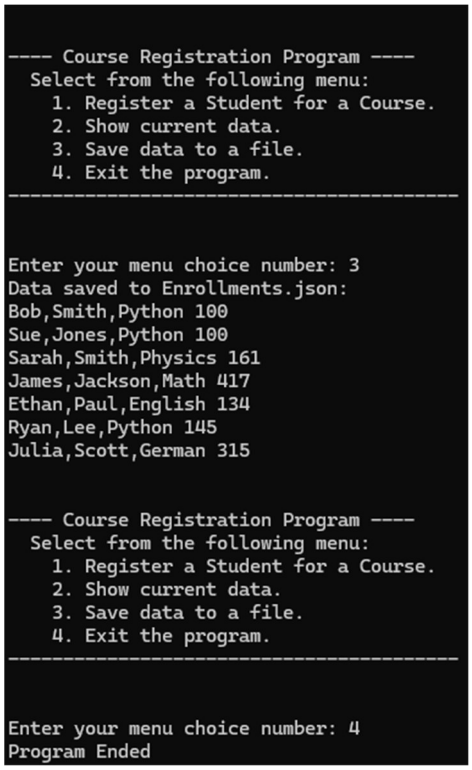

Figure 7.2: Enrollment.json file saved to folder

Seen below in Figure 7.3 is the student enrollment data stored in the json file.

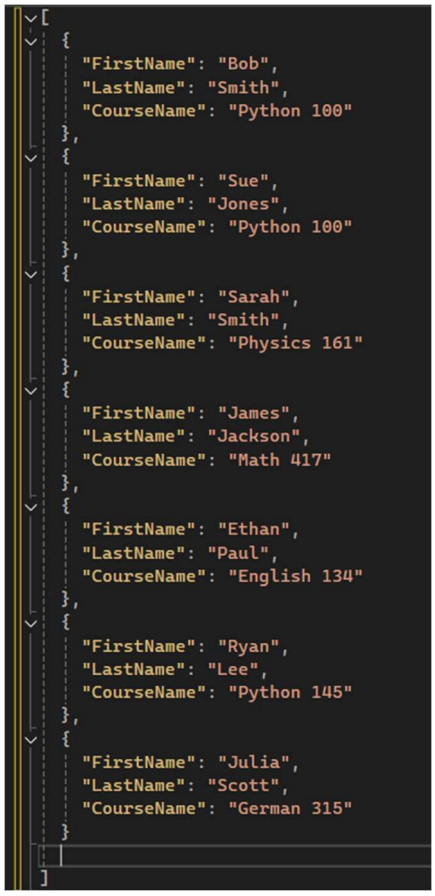

Figure 7.3: Enrollment.json file saved to folder

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 with inheritance as well as functions, loops, conditional logic and case statements. Additionally, reading the json file and later writing to the json file was a great example for working with lists of objects, and utilizing the method dump( ). It was also a great example of utilizing an error handling function throughout the code. Overall, this exercise was a great example of utilizing classes and inheritance as well as functions and a main script utilizing previously declared functions.