Ogonna Anunoby

09/04/2024

IT FDN 110 B Su 24: Foundations of Programming: Python

Assignment 06

Assignment 06 Report

Introduction

In Assignment 06, I wrote a script that registers students for various courses. It was similar to Assignment 05, but I used a got to use classes and methods organized according to the separation of concerns design principle. Assignment 06 allowed me to continue developing my Python skills.

Script Testing

Assignment06.py

```python
# ------------------------------------------------------------------------------------------ #
# Title: Assignment06
# Desc: This assignment demonstrates using classes and methods
# with the seperation of concerns design principle
# Change Log: (Who, When, What)
#   RRoot,1/1/2030,Created Script
#   Ogonna Anunoby, 09/04/2024, First attempt at Assignment 06
# ------------------------------------------------------------------------------------------ #
import json

# Define the Data 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"

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

Figure 1: Script Header, Constants, and Variables for Assignment 06 Script Code

Figure 1 shows the header and data constants for the Assignment 06 script code. Lines 1 through 8 consist of the script header. The change log has one entry, showing that I made my first attempt of Assignment 06 on 09/04/2024. Line 9 shows that I imported the Json library to allow for use of its library function when processing Json data. Lines 11 through 22 are where I have defined my data constants. In this script, there are two data constants. MENU is a string that displays the registration menu to the user. FILE_NAME is a string that holds the name of the file, Enrollments.json. This file will save the registration data. Lines 24 through 26 show the data variables for this assignment. The variable students is initialized to an empty list. The variable menu_choice is initialized to an empty string.

```
28  # Processing ------------------------------------- #
29  class FileProcessor:
30      """
31      A collection of processing layer methods that work with Json files
32
33      ChangeLog: (Who, When, What)
34      Ogonna Anunoby, 09/04/2024, created the class and wrote methods to read from and write to the Json file
35      """
```

Figure 2: FileProcessor class and its docstring

Figure 2 shows the FileProcessor class and its docstring. A class is blueprint that allows for the grouping of functions, variables, and constants. Instances of classes are called objects, which can be created and used to handle data. This class deals with the data storage concern in the the separation of concerns design principle. This class has methods that deal the management, processing, and storage of data in the Json file. The FileProcessor class runs from lines 28 through 90. Lines 30 through 35 show the docstring for this class. This code serves as developer notes and is enclosed in three quotation marks. Here the docstring explains that this class has methods to handle Json file processing. It also shows the change log for this class.

```
37      # When the program starts, read the file data into table
38      # Extract the data from the file
39      # Read from the Json file
40      @staticmethod
41      def read_data_from_file(file_name: str, student_data: list) -> list:
42          """
43          A method to read student registraion data from a Json file and store it into a list.
44
45          ChangeLog: (Who, When, What)
46          Ogonna Anunoby, 09/04/2024, Created method
47
48          :param file_name: string containing the name of the file
49          :param student_data: list for holding student registration information
50          :return student_data: list containing student registration information stored in the Json file
51          """
52          try:
53              file = open(file_name, "r")
54              student_data = json.load(file)
55              file.close()
56          except FileNotFoundError as e: # Catch FileNotFound exception
57              IO.output_error_messages("Text file must exist before running this script!", e)
58          except Exception as e: # Catch general exception
59              IO.output_error_messages("There was a non-specific error!", e)
60          finally:
61              if file.closed == False:
62                  file.close()
63          return student_data
```

Figure 3: The read_data_from_file() method

Figure 3 shows the read_data_from_file() method. A method is a function that belongs to a class FileProcessor. Meanwhile, a function is a reusable block of code that performs a specific task or a set of tasks. Line 60 uses the @staticmethod decorator to indicate that this method belongs to the class. So, a FileProcessor object does not need to be created in to use the method. Line 41 has the method definition. The “def” keyword is used to define the method. Inside the parentheses are the arguments that the method requires. Here we have a string which will replace the file_name variable when passed to the method. We also have student_data as a parameter of type list. This will replace student_data in this method. To the right of the arrow symbol (->), is the return value. For this method, it is a list. Everything indented after line 41 is included in this method. So, the method runs from lines 40 through 63 (including the @staticmethod decorator). Lines 42 through 51 shows the docstring for this method. Here, there is a general description of the method and a change log. There is also a description of the parameters and return value. This method reads data from the Json file into the student_data list.

Lines 52 through 62 show the try-except-except-finally block that is used to process reading data from the file. In the try block, the code attempts to open the file in read mode, load the Json data into the student_data dictionary, then close the file. If the file cannot be found, then the code will throw a FileNotFoundError exception and go to line 56. The output_error_messages() method belonging to the IO class will be called, with a specified error message and the error object passed to it. If the try block fails for any other reason, a general error exception will be thrown and the code will go to line 58. The output_error_messages() method belonging to the IO class will be called, with a general error message and the error object passed to it. In the finally block, which will always be called, if the file is still open, it will be closed. On line 63, the student_data list is returned.

```python
65      @staticmethod
66      def write_data_to_file(file_name: str, student_data: list) -> None:
67          """
68          A method to write and save student registration data to a Json file.
69
70          ChangeLog: (Who, When, What)
71          Ogonna Anunoby, 09/04/2024, Created method
72
73          :param file_name: string data holding the name of the file
74          :param student_data: list data used to hold student registration information
75          :return None: No data returned
76          """
77          try:
78              file = open(file_name, "w")
79              json.dump(student_data, file)
80              file.close()
81              print("The Json file has the following registration data saved:")
82              for student in student_data:
83                  print(f"You have registered {student["FirstName"]} {student["LastName"]} for {student["CourseName"]}.")
84          except TypeError as e: # Catch TypeError exception
85              IO.output_error_messages("Please check that the data is a valid JSON format.", e)
86          except Exception as e: # Catch general exception
87              IO.output_error_messages("There was a non-specific error!", e)
88          finally:
89              if file.closed == False:
90                  file.close()
```

Figure 4: The write_data_to_file() method

Figure 4 shows the write_data_to_file() method. This also belongs to the FileProcessor class and runs from lines 65 through 90 (including the @static method decorator). The decorator on line 65 indicates that this method is static. Lines 67 through 76 show the docstring giving a general description of this method, the change log, and a description of the parameters and return value for this method. This method writes data from the student_data list to the Json file. This method takes in a sting for the file_name and list for student_data. Since there is no return statement in this method, the method will return a None object. In other words, this method returns nothing.

Lines 77 through 90 show the try-except-except-finally block that is used to process writing data to the file. In the try block, the code attempts to open the file in write mode, put the data in the student_data dictionary into the Json file, then close the file. If the type of data being put into the Json file is incompatible with it, a TypeError exception will be thrown, and the code will go to line 84. The output_error_messages() method belonging to the IO class will be called, with a specified error message and the error object passed to it. If the try block fails for any other reason, a general error exception will be thrown, and the code will go to line 86. The output_error_messages() method belonging to the IO class will be called, with a general error message and the error object passed to it. In the finally block, which will always be called, if the file is still open, it will be closed.

```python
91
92      # Presentation ------------------------------------- #
93      class IO:
94          """
95          A collection of presentation layer methods that manage user input and output
96
97          ChangeLog: (Who, When, What)
98          Ogonna Anunoby, 09/04/2024, Created the class and added methods for input, output, data display, and displaying custom error messages
99          """
100
```

Figure 5: IO class and its docstring

Figure 5 shows the IO class and its docstring. The IO class runs from lines 92 through 204 Lines 94 through 99 show the docstring for this class, which explains that this class has methods to deal with input and output. It also lists the change log for this class.

```
101     @staticmethod
102     def output_error_messages(message: str, error: Exception = None) -> None:
103         """ This method displays the a custom error messages to the user
104
105         ChangeLog: (Who, When, What)
106         Ogonna Anunoby, 09/02/2024, Created method
107
108         :param menu: string containing user menu
109         :param error: exception object for the error
110         :return None: No data returned
111         """
112         print(message, end="\n\n")
113         if error is not None: # prints error information if error object is passed
114             print("-- Technical Error Message -- ")
115             print(error, error.__doc__, type(error), sep='\n')
```

Figure 6: The output_error_messages() method

Figure 6 shows the output_error_messages() method. The method runs from lines 101 through 115 (including the @staticmethod decorator). The docstring runs from lines 103 through lines 111. The docstring explains that the method displays custom error messages to the user. It also has a change log and explains the method takes a string for message and exception object for error. The method returns nothing. It prints the message string that was passed to is, followed by two new lines as indicated by the end parameter in the print statement on line 112. Is should be noted that on line 102, the error exception object has a default value of None. So, this parameter is optional. The if statement on line 112 only runs if an exception object was passed to it. If this is the case, then an error message is printed, as well as the error object, error docstring, error type, each separated by a new line.

```
118     @staticmethod
119     def output_menu(menu: str) -> None:
120         """ This method displays the a menu of choices to the user
121
122         ChangeLog: (Who, When, What)
123         Ogonna Anunoby, 09/02/2024, Created method
124
125         :param menu: string containing user menu
126         :return None: No data returned
127         """
128         print()
129         print(menu)
130         print()  # Adding extra space to make it look nicer.
```

Figure 7: The output_menu() method

Figure 7 shows the output_menu() method. The method runs from lines 118 through 130 (including the @staticmethod decorator). The docstring runs from lines 120 through lines

127. The docstring explains that the method displays the menu to the user. It also has a change log and explains the method takes no parameters and returns nothing.

```
133     @staticmethod
134     def input_menu_choice() -> str:
135         """ This method gets a menu choice from the user
136
137         ChangeLog: (Who, When, What)
138         Ogonna Anunoby, 09/02/2024, Created function
139
140         :param None: No parameters
141         :return choice: string with user's menu choice
142         """
143         choice = "0"
144         try:
145             choice = input("Enter your menu choice number: ")
146             if choice not in ("1","2","3","4"):  # Note these are strings
147                 raise Exception("Please, choose only 1, 2, 3, or 4")
148         except Exception as e:
149             IO.output_error_messages(e.__str__())  # Not passing the exception object to avoid the technical message
150
151         return choice
```

Figure 8: The input_menu_choice() method

Figure 8 shows the input_menu_choice() method. The method runs from lines 133 through 151 (including the @staticmethod decorator). The docstring runs from lines 135 through lines 142. The docstring explains that the method gets the menu choice from the user. It also has a change log and explains the method takes no parameters and returns the string choice. Furthermore, this method uses a try-except block to ensure that only valid choices “1”, “2”, “3”, and “4” are entered. If an invalid choice is entered, an exception is raised on line 146, then caught on line 148, Then the output_error_messages method is called, passing in the error’s string object. If a valid choice is entered, the choice is returned.

```
154     @staticmethod
155     def output_student_courses(student_data: list) -> None:
156         """ This method displays the class each student registered for to the user
157
158         ChangeLog: (Who, When, What)
159         Ogonna Anunoby, 09/02/2024, Created method
160
161         :param student_data: list containing student registration information
162         :return None: No data returned
163         """
164
165         # Process the data to create to display current student registration data
166         print()
167         print("-" * 50)
168         for student in student_data:
169             print(f"Student {student['FirstName']} {student['LastName']} is enrolled in {student['CourseName']}")
170         print("-" * 50)
171         print()
```

Figure 9: The output_student_courses() method

Figure 9 shows the output_student_courses() method. This method runs from lines 154 through 171 (including the @staticmethod decorator). The docstring runs from lines 156 through lines 163. The docstring explains that the method displays the registration data to

the user. It also has a change log and explains the method takes a list for student_data and returns nothing. This method prints a boarder to surround the registration data on lines 167 and 170 printing. It does this by printing "-" fifty times to the screen. Lines 168 and 169 iterate through each student in the student_data dictionary in the students list and prints out each student's information using the student dictionary and the FirstName, LastName, and CourseName keys.

```
174     @staticmethod
175     def input_student_data(student_data: list) -> list:
176
177         """ This method gets the student's first name, last name, course they want to register for from the user
178
179         ChangeLog: (Who, When, What)
180         Ogonna Anunoby, 09/04/2024, Created method
181
182         :param student_data: list containing student registration information
183         :return student_data: list containing updated student registration information
184         """
185
186         try:
187             # Input the data
188             student_first_name = input("Enter the student's first name: ")
189             if not student_first_name.isalpha(): # Check if only letters are entered
190                 raise ValueError("The first name should only contain letters.")
191
192             student_last_name = input("Enter the student's last name: ")
193             if not student_last_name.isalpha(): # Check if only letters are entered
194                 raise ValueError("The last name should only contain letters.")
195         except ValueError as e:
196             IO.output_error_messages("That value is not the correct type of data!", e)
197         except Exception as e:
198             IO.output_error_messages("There was a non-specific error!", e)
199         else:
200             course_name = input("Enter the name of the course: ")
201             student = {"FirstName": student_first_name,"LastName": student_last_name, "CourseName" : course_name}
202             student_data.append(student) # Load student dicitionary into our collection (list of dictionaries)
203             print(f"You have registered {student_first_name} {student_last_name} for {course_name}.")
204         return student_data
205
206
207 # End of function definitons
```

Figure 10: The input_student_data() method

Figure 10 shows the input_student_data() method. This method runs from lines 174 through 204 (including the @staticmethod decorator). The docstring runs from lines 177 through lines 184. The docstring explains that this method gets the student's first name, last name, and course choice from the user. It also has a change log and explains the method takes a list for student_data and returns the student_data list.

In the try block from lines 186 through 194, the user is asked to enter a first name on line 188. Line 189 checks if the user entered all alphabetical characters for the student's first name. If they did, the code asks the user to enter the student's last name on line 192. If they did not enter all alphabetical characters for the student's first name, a ValueError exception will be thrown, and the code will jump to the ValueError exception on line 195. Then, the output_error_messages() method belonging to the IO class will be called with a specified error message and the error message passed as parameters.

Line 192 checks if the user entered all alphabetical characters for the student's last name. If they did, the code would move to else block on line 199. The else block on lines 199 though 203 will only run if no exceptions were thrown. If non-alphabetical characters were

entered for the student's last name, a ValueError exception will be thrown, and the code will jump to line 197. The output_error_messages() method belonging to the IO class will be called with a specified error message and the error message passed as parameters.

The else block on lines 199 through 203 will be run if no exceptions are thrown. Here, the user is asked to enter the name of the course. Then, on line 201 the dictionary for the student is created with FirstName, LastName, and CourseName as keys and student_first_name, student_last_name, and course_name as values. The student dictionary is appended to the students list on line 202. Then a message letting the user know that the student was registered is printed to the screen. Line 207 has a comment indicating that the script is at the end of all of the function definitions.

```
211 # Beginning of the main body of this script
212
213 # When the program starts, read the file data into table
214 # Extract the data from the Json file into a list of dictionaries
215 students = FileProcessor.read_data_from_file(file_name=FILE_NAME, student_data=students)
216
217 # Repeat the follow tasks to present and process the data
218 while True:
219     IO.output_menu(menu=MENU)
220
221     menu_choice = IO.input_menu_choice()
222
223     if menu_choice == "1":  # Get new data (and display the change)
224         students = IO.input_student_data(student_data=students)
225         continue
226
227     elif menu_choice == "2":  # Display current data
228         IO.output_student_courses(student_data=students)  # Added this to improve user experience
229         continue
230
231     elif menu_choice == "3":  # Save data in a file
232         FileProcessor.write_data_to_file(file_name=FILE_NAME, student_data=students)
233         continue
234
235     elif menu_choice == "4":  # End the program
236         break  # out of the while loop
237
238 print("Program Ended")
```

Figure 11: Main body while loop

Line 211 has a comment indicating that the script is at the main body of the script. Line 215 calls the read_data_from_file() method belonging to the FileProcessor class. It passes in the file_name string and student list. The list returned from this function is stored in the students variable. The infinite while loop running from lines 218 through 236 allows the user to interact with the application. This is shown in Figure 11. It continuously prints the menu choice one line 219 using the output_menu() function from the IO class, with MENU passed to it. On line 221, the input_menu_choice() method from the IO class is called to get a valid menu choice from the user. The value returned from the function is stored in menu_choice. If menu_choice is "1", it calls the input_student_data() function from the IO class, passing the students list as a parameter. It stores the returned list in students. If menu_choice is "2", it calls the output_student_courses() function from the IO class, passing the students list as a parameter. It stores the returned list in students. If menu_choice is "3", it calls the write_data_to_file() function from the FileProcessor class, passing the file_name string and students list as parameters. It stores the returned list in

students. If the menu_choice is “4”, the code breaks out of the loop. Then on line 238, “Program Ended” is printed onto the screen, and the program ends.

Script Testing

After writing the code for the script, I needed to test it. First, I saved the script using the name Assignment06.py to my _Module06/Assignments folder in my downloads folder. I ran the script and tested all of the options to ensure that my script worked as I expected.

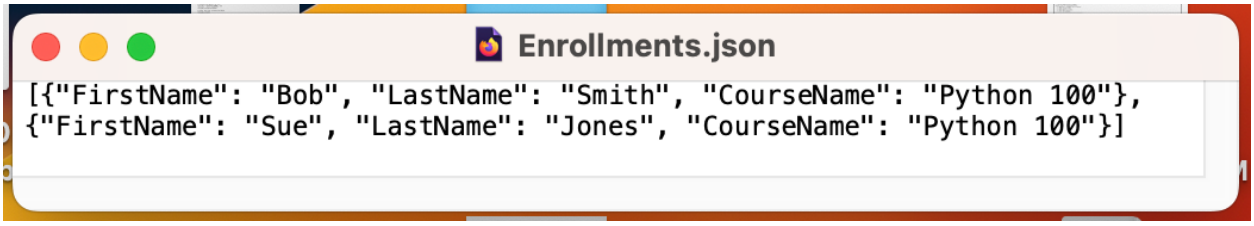

Figure 12: Enrollments.json Before Running the Script

Figure 12 shows the contents of Enrollments.json before running the script. I ran the script in PyCharm and chose choice “2” to display each registered student. The contents of the Json file were printed to the screen, as expected, as shown in Figure 13.

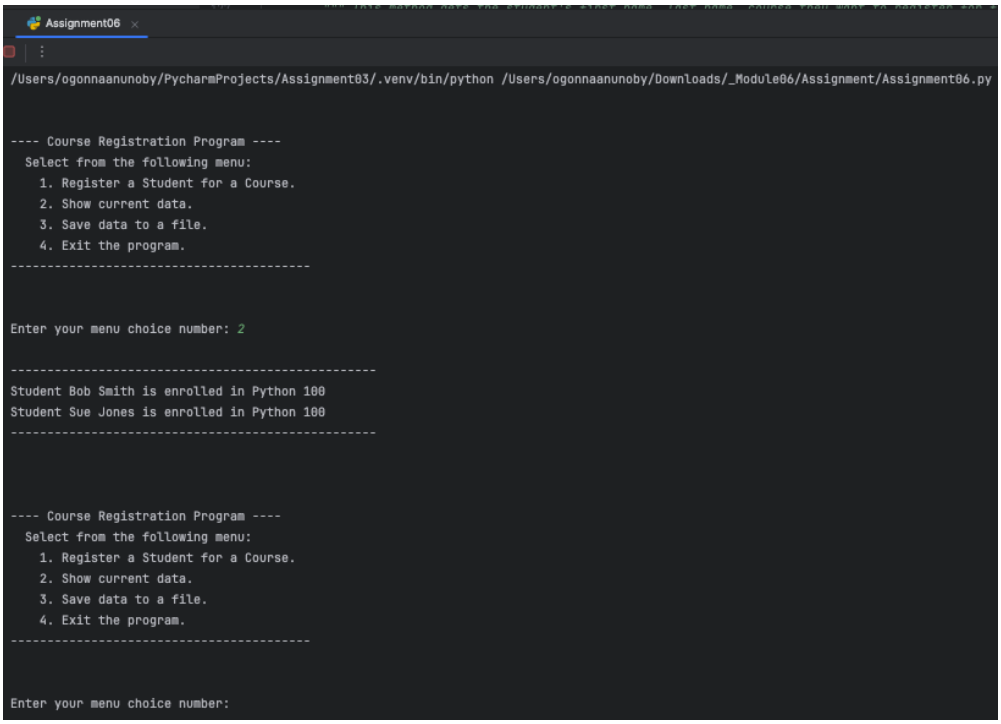

Figure 13: Students Enrolled Before Running the Script

Next, I made choice “1” to register a student for the course. I entered a name with numbers in it and the ValueError exception was thrown.

```
----------------------------------------


Enter your menu choice number: 1
Enter the student's first name: sdf8
That value is not the correct type of data!

-- Technical Error Message --
The first name should only contain letters.
Inappropriate argument value (of correct type).
<class 'ValueError'>


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

Figure 14: ValueError Exception for the Student’s First Name

Then, I made choice “1” again and entered all alphabetic characters for the student’s first name. The code then prompted me to enter the student’s last name. I entered a name with numbers in it, and the ValueError exception was thrown again.

```
----------------------------------------


Enter your menu choice number: 1
Enter the student's first name: Bob
Enter the student's last name: dsr23
That value is not the correct type of data!

-- Technical Error Message --
The last name should only contain letters.
Inappropriate argument value (of correct type).
<class 'ValueError'>


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

Figure 15: ValueError Exception for the Student’s Last Name

I then entered the name of the course, and the student was registered for the course.

```
Enter your menu choice number: 1
Enter the student's first name: Bob
Enter the student's last name: Baker
Enter the name of the course: Python 300
You have registered Bob Baker for Python 300.


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

Figure 16: Successfully Registering a Student

Figure 17 shows me making choice “3”, displaying the data and saving the data to Enrollments.csv.

```
Enter your menu choice number: 3
The Json file has the following registration data saved:
You have registered Bob Smith for Python 100.
You have registered Sue Jones for Python 100.
You have registered Bob Baker for Python 300.


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

Figure 17: Saving All of the Registered Students

In Figure 18, I enter an invalid choice, and I get an error message and am prompted to try again. I then choose “4” and exit the program.

```
Enter your menu choice number: 5
Please, choose only 1, 2, 3, or 4



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


Enter your menu choice number: 4
Program Ended
```

Figure 18: Entering an invalid Option and Exiting the Program

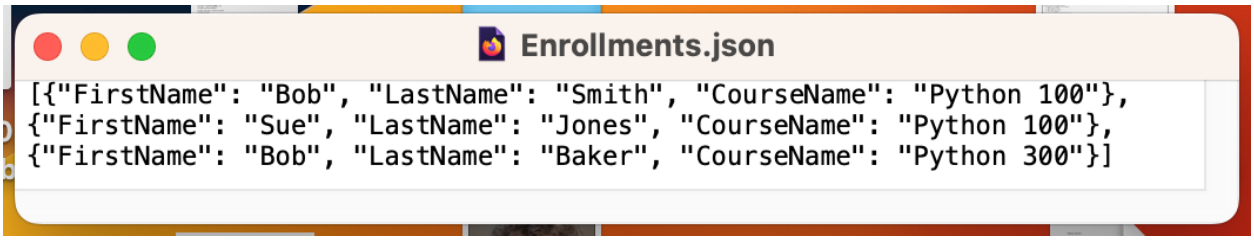

Figure 19: Showing All Registered Students in the Json File

Figure 19 shows that each registered student has been saved to Enrollments.json. I also ran the script in the terminal using the same commands as I did in PyCharm. I started with the Enrollments.json file shown in Figure 11. Then, I ran the script in the terminal with the same results. The script output in the terminal is shown in Figures 20 and 21. Figure 22 shows the final Enrollments.json file I got.

```
ogonnas-mbp:Assignment ogonnaanunoby$ python3 Assignment06.py


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


Enter your menu choice number: 2

--------------------------------------------------
Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Bob Baker is enrolled in Python 300
--------------------------------------------------



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


Enter your menu choice number: 1
Enter the student's first name: sd43
That value is not the correct type of data!

-- Technical Error Message --
The first name should only contain letters.
Inappropriate argument value (of correct type).
<class 'ValueError'>


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


Enter your menu choice number: 1
Enter the student's first name: Vic
Enter the student's last name: ds84
That value is not the correct type of data!

-- Technical Error Message --
The last name should only contain letters.
Inappropriate argument value (of correct type).
<class 'ValueError'>
```

Figure 20: Terminal Script Results, Part 1

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


Enter your menu choice number: 1
Enter the student's first name: Vic
Enter the student's last name: Tu
Enter the name of the course: Python 200
You have registered Vic Tu for Python 200.


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


Enter your menu choice number: 3
The Json file has the following registration data saved:
You have registered Bob Smith for Python 100.
You have registered Sue Jones for Python 100.
You have registered Bob Baker for Python 300.
You have registered Vic Tu for Python 200.


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


Enter your menu choice number: 5
Please, choose only 1, 2, 3, or 4



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


Enter your menu choice number: 4
Program Ended
ogonnas-mbp:Assignment ogonnaanunoby$
```

Figure 21: Terminal Script Results, Part 2

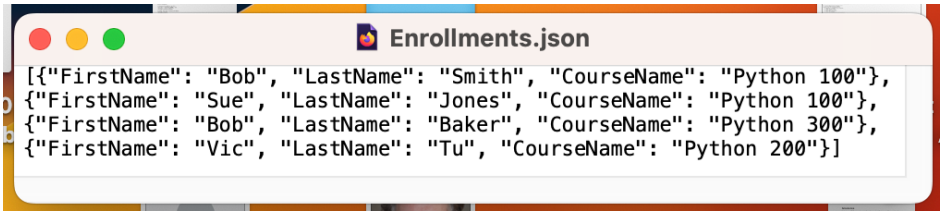

Figure 22: Again, Showing All Registered Students in the Json File

Finally, I uploaded my report and python script to the GitHub repository I created for this assignment. This can be found at https://github.com/944695/IntroToProg-Python-Mod06.

Conclusion

In conclusion, Assignment 06 allowed me to practice the skills I gained from lesson 06. It allowed me to practice programing with the separation of concern design principle, as well as practice using classes and methods. Overall, completing Assignment 06 has helped me continue building my Python knowledge.

Citations

Mod06-Notes.docx
Assignment06.py