

Name: Reilly Thomer
Date: November 11th, 2024
Course: IT FDN 110 A – Foundations of Programming: Python
Assignment: Module 5 Assignment

Module 5 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 csv file, as well as exit the program. This script incorporates elements such as while and for loops, reading and writing to files, programming menus, conditional logic, formatting data, etc. 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.

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

```
22 # Define the Data Variables and constants
23 student_first_name: str = '' # Holds the first name of a student entered by the user.
24 student_last_name: str = '' # Holds the last name of a student entered by the user.
25 course_name: str = '' # Holds the name of a course entered by the user.
26 csv_data: str = '' # Holds combined string data separated by a comma.
27 json_data: str = '' #Holds the json data
28 file = None # Holds a reference to an opened file.
29 menu_choice: str # Hold the choice made by the user.
30 student_data: dict = {} # dictionary of student data
31 students: list = [] # a list of student data
```

Figure 1.2: Data Variable Initialization

Reading Data in JSON File

In the main body of the script the first task to be completed is to read the data from the “Enrollments.json” file. 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 json_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.

```
33 # When the program starts, read the file data into a list of lists (table)
34 # Extract the data from the file
35 import json
36 try:
37     file = open(FILE_NAME, "r")
38     json_data = json.load(file)
39     for item in json_data:
40         # Transform the data from the file
41         student_data = {"FirstName":item["FirstName"],"LastName":item["LastName"],"CourseName":item["CourseName"]}
42         #student_data = [student_data[0], student_data[1], student_data[2].strip()]
43         # Load it into our collection (list of lists)
44         students.append(student_data)
45     file.close()
```

Figure 2.1: Try method for reading Data from 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 below in Figure 2.2 are the except blocks written for handling these errors. In this code example there are 2 main errors handled with an additional exception block used to handle all other errors that occur. On lines 46 through 55 is the code executed when a FileNotFoundError occurs. In this case the user is informed of the error with both a simple explanation of the error and the technical error message.

It should also be noted that after writing the error message for the user it is ensured that the json file is closed. If the json file is not closed properly then other errors may occur later in the script. At the end of the FileNotFoundError exception block the program is ended as a file must be present for the script to execute properly. The next exception block is the KeyError. This error occurs when a dictionary Key ID is not correct. This might occur when an improper dictionary id is assigned in the json file. In this exception block the error is printed out both in simple and technical format for the user. Similar to the FileNotFoundError it is ensured that the json file is closed as well as the script is ended. This is also completed in the last exception block. The last exception block as seen below in Figure 2.2 catches all the other errors that might occur when reading the data from the json file.

```
46 except FileNotFoundError as e:
47     print("-" * 50)
48     print("Error:\n JSON file must exist before running this script!\n")
49     print("-- Technical Error Message -- ")
50     print(e, e.__doc__, type(e), sep='\n')
51     if file.closed == False: #Makes sure the file is open before trying to close it.
52         file.close()
53     print("\n-- Program Ended --")
54     print("-" * 50)
55     exit()
56 except KeyError as e:
57     print("-" * 50)
58     print("Error:\n JSON file contains improper data. Check the data and fix any incorrect keys before running this script!\n")
59     print("-- Technical Error Message -- ")
60     print(e, e.__doc__, type(e), sep='\n')
61     if file.closed == False: #Makes sure the file is open before trying to close it.
62         file.close()
63     print("\n-- Program Ended --")
64     print("-" * 50)
65     exit()
66 except Exception as e:
67     print("-" * 50)
68     print("Error:\n There was an error reading the JSON file.")
69     print(e, e.__doc__, type(e), sep='\n')
70     if file.closed == False: #Makes sure the file is open before trying to close it.
71         file.close()
72     print("\n-- Program Ended --")
73     print("-" * 50)
74     exit()
75
```

Figure 2.2: Error handling when reading Data from Enrollments.json

Data Entry & Processing

Once the Enrollment.json file data was read a while loop was created, seen below in Figure 3.1. This while loops continue execution until the user selects menu option 4 to end the program. The beginning of the while loop contains the code for presenting the menu of choices to the user and taking their menu choice input. (Figure 3.1) Inside the while loop following the menu option input from the user is a series of conditional statements. Depending on the user's menu choice the appropriate case statement will be true and the appropriate code will be executed.

```

76     # Present and Process the data
77     while (True):
78
79         # Present the menu of choices
80         print(MENU)
81         menu_choice = input("What would you like to do: ")

```

Figure 3.1: While Loop Declaration and Menu User Input

Menu Option 1

Below in Figure 3.2 is the python script that is executed in the if statement when the user selects menu option 1. Inside the if statement is a try block that is executed. The user is prompted through inputs to enter the students first name, last name, and course name. Upon entry of the data by the user the student data is then formatted and added to the students list. Formatting of the data can be seen on lines 94 and 95 in Figure 3.2, and the adding of said formatted data to the students list can be seen on line 96. The call `append()` is used in order to add an additional line to the end of the list. At the end of this If statement the program pointer returns to the top of the while loop.

```

83     # Input user data
84     if menu_choice == "1": # This will not work if it is an integer!
85         try:
86             print("-" * 50)
87             student_first_name = input("Enter the student's first name: ")
88             if not student_first_name.isalpha():
89                 raise ValueError("The first name should not contain numbers.")
90             student_last_name = input("Enter the student's last name: ")
91             if not student_last_name.isalpha():
92                 raise ValueError("The last name should not contain numbers.")
93             course_name = input("Please enter the name of the course: ")
94             csv_data = f'{student_first_name},{student_last_name},{course_name}'
95             student_data = {"FirstName":student_first_name,"LastName":student_last_name,"CourseName":course_name}
96             students.append(student_data)
97             print(f"You have registered {student_first_name} {student_last_name} for {course_name}.")
98             print("-" * 50)

```

Figure 3.2: Student Data Entry upon Menu Option 1

When working with the users input many errors can occur. One of these errors includes the user entering numbers or symbols when prompted to enter the students first and last name, other errors can happen as well. Seen above in Figure 3.2 upon the entry of the students' first name the data entered is checked to ensure only letters were entered. If anything other than the expected letters are entered a Value Error is raised that is then handles in the except blocks seen in Figure 3.3. This same data entry check is completed when the user enters the students' last name. The exception blocks for handling a Value error prints out the error information in a simple and technical format. Upon completion of the exception handling the program pointer goes back to printing of the menu options for the user. The other exception block seen in Figure 3.3 catches all the other errors that might occur when the user enters data.

```

99         except ValueError as e:
100             print("-" * 50)
101             print(f"Error:\n {e}\n")
102             print("-- Technical Error Message -- ")
103             print(e, e.__doc__, type(e), sep='\n')
104             print("-" * 50)
105         except Exception as e:
106             print("-" * 50)
107             print(f"Error:\n There was a non-specific error!\n")
108             print("-- Technical Error Message -- ")
109             print(e, e.__doc__, type(e), sep='\n')
110             print("-" * 50)
111         continue

```

Figure 3.3: Error handling for Student Data Entry upon Menu Option 1

Menu Option 2

Below in Figure 3.4 is the python script that is executed when the user selects menu option 2. If menu options 2 is entered by the user the below else if statement is found true and the script inside the else if is executed. This menu option prints all the student information in the students list. To print out each students information a for loop was used to go through each row of the students list. The same as mentioned previously when discussing menu option 1 once the continue call is reached the program pointer returns to the while loop, then prompting the user to input another menu option.

```

113         # Present the current data
114         elif menu_choice == "2":
115
116             # Process the data to create and display a custom message
117             print("-"*50)
118             for student in students:
119                 print(student)
120             print("-"*50)
121             continue

```

Figure 3.4: Student Data formatting upon Menu Option 2

Menu Option 3

Below in Figure 3.5 is the python script that is executed in the case that the user selects menu option 3. When this else if statement is found to be true a json file named "Enrollments.json" is opened in write mode. To open in write mode a "w" was indicated in the open() call as seen on line 126 in Figure 3.5. After the file was opened the students list was then written to the file utilizing the dump() method. Once the students list was added to the json file the json file was then closed using the close() command. After writing the student list to the json file the students list was also printed out for the user to see. This was completed by a For loop seen on line 131. Upon completion

of the else if statement for menu option 3 the program pointer then returned to the while loop after reaching the continue call. Once back at the top of the while loop the user is prompted again for another menu choice. All the above-described script was written in a try block. The exception handling of this

```
123     # Save the data to a file
124     elif menu_choice == "3":
125         try:
126             file = open(FILE_NAME, "w")
127             json.dump(students, file)
128             file.close()
129             print("-" * 50)
130             print("The following data was saved to file!")
131             for student in students:
132                 print(f"{student['FirstName']},{student['LastName']},{student['CourseName']}")
133             print("-" * 50)
134             continue
```

Figure 3.5: Student Data written to JSON File upon Menu Option 3

Following the try block above in Figure 3.5 came the Except block seen in Figure 3.6. When writing the students data to the JSON file errors can occur. To handle these errors an exception block was created. This block prints the error to the user. It also checks to ensure that the json file is closed if an error occurs. If the json file is not closed the file will then close. This is important because other errors might occur later on in the script if the json file is not closed.

```
135     except Exception as e:
136         print("-" * 50)
137         print(f"Error:\n There was a non-specific error!\n")
138         print("-- Technical Error Message -- ")
139         print(e, e.__doc__, type(e), sep='\n')
140         print("-" * 50)
141         if file.closed == False: # Makes sure the file is open before trying to close it.
142             file.close()
```

Figure 3.6: Error handling for writing to JSON file upon Menu Option 3

Menu Option 4 & Other

```
144     # Stop the loop
145     elif menu_choice == "4":
146         break # out of the loop
```

Figure 3.7: While loop break upon Menu Option 4

In the case the user selects menu option 4 the program is to end. Seen above in Figure 3.7 is the script within this else if statement that is to be found true when the user selects menu option 4. In

order to break the case statement and while loop “break” is called. This call breaks the program pointed out of the while loop thus executing the lines of code after the while loop followed by ending the script in this case. As seen in Figure 3.8 below upon finishing the while loop the program is ended with a print statement informing the user that the program has ended.

```
148         else:
149             print("Please only choose option 1, 2, or 3")
150
151     print("Program Ended")
```

Figure 3.8: When Menu option is not entered properly

The final statement in the if/elseif evaluations of the user's menu selection is the else statement. This can be seen above in Figure 3.8. When a user input is entered that is not evaluated as true by any of the other if/elseif statements this else statement is then executed. In this case a print statement is issued informing the user to enter an option 1, 2, or 3. Upon completion of the else statement the while loop is ran again.

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 student's 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: Sarah
Enter the student's last name: Smith
Please enter the name of the course: Math 165
You have registered Sarah Smith for Math 165.
-----

---- 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
-----
{'FirstName': 'Bob', 'LastName': 'Smith', 'CourseName': 'Python 100'}
{'FirstName': 'Sue', 'LastName': 'Jones', 'CourseName': 'Python 100'}
{'FirstName': 'Sarah', 'LastName': 'Smith', 'CourseName': 'Math 165'}
-----

```

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!
Bob,Smith,Python 100
Sue,Jones,Python 100
Sarah,Smith,Math 165
-----

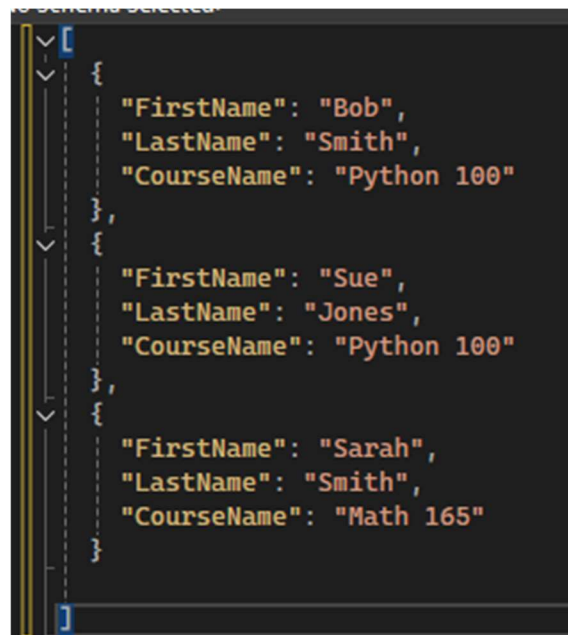
---- Course Registration Program ----
Select from the following menu:
  1. Register a Student for a Course.
  2. Show current data.
  3. Save data to a file.
  4. Exit the program.
-----

What would you like to do: 4
Program Ended

```

Figure 4.2: Enrollment.csv file saved to folder

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

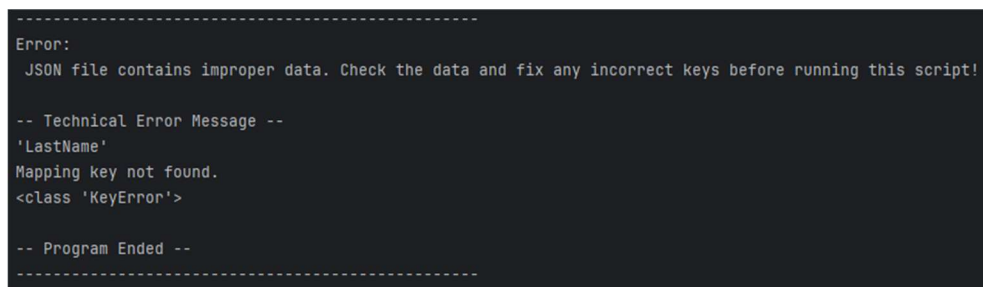


```
[{"FirstName": "Bob", "LastName": "Smith", "CourseName": "Python 100"}, {"FirstName": "Sue", "LastName": "Jones", "CourseName": "Python 100"}, {"FirstName": "Sarah", "LastName": "Smith", "CourseName": "Math 165"}]
```

Figure 4.3: Enrollment.csv file saved to folder

Error Handling

When testing the scripts functionality it was also checked for its error handling. Below in Figure 4.4 is an example of an error that was presented when the Enrollments json file had an incorrect data name. This incorrect data name can be seen in Figure 4.5. In the Json file on row 2 the label of “Email” is utilized when “LastName” should be in its place.

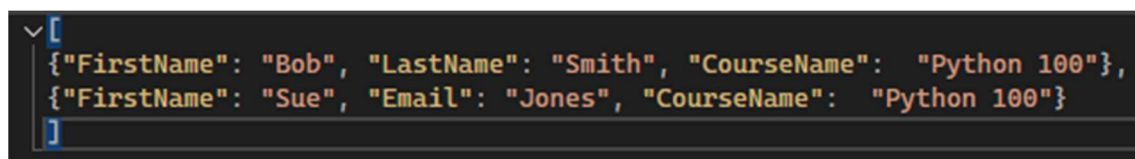


```
-----
Error:
JSON file contains improper data. Check the data and fix any incorrect keys before running this script!

-- Technical Error Message --
'LastName'
Mapping key not found.
<class 'KeyError'>

-- Program Ended --
-----
```

Figure 4.4: Enrollment.json file Data Error upon Execution



```
[{"FirstName": "Bob", "LastName": "Smith", "CourseName": "Python 100"}, {"FirstName": "Sue", "Email": "Jones", "CourseName": "Python 100"}]
```

Figure 4.5: Enrollment.json file improper Data

An additional error that was tested was when the user entered anything other than a character in either of the student name and last name fields. Seen below in Figure 4.6 is the error that occurred when numbers were entered into the student first name field.

```
---- 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: 123
-----

Error:
The first name should not contain numbers.

-- Technical Error Message --
The first name should not contain numbers.
Inappropriate argument value (of correct type).
<class 'ValueError'>
-----
```

Figure 4.6: Improper Student Name Data Entry Error

Command Prompt

After the script ran successfully in PyCharm it was then tested when ran through Command Prompt. (Figure 4.7 and Figure 4.8) When running the script in command prompt it also functioned as desired.

```

PS C:\Users\knuff\OneDrive\Documents\Python\PythonCourse\Module1\.venv\Scripts>

---- 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: Sarah
Enter the student's last name: Smith
Please enter the name of the course: Physics 161
You have registered Sarah Smith for Physics 161.
-----

---- 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

{'FirstName': 'Bob', 'LastName': 'Smith', 'CourseName': 'Python 100'}
{'FirstName': 'Sue', 'LastName': 'Jones', 'CourseName': 'Python 100'}
{'FirstName': 'Sarah', 'LastName': 'Smith', 'CourseName': 'Physics 161'}
-----

```

Figure 4.7: Script Ran through Command Prompt Menu Options 1 & 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!
Bob,Smith,Python 100
Sue,Jones,Python 100
Sarah,Smith,Physics 161
-----

---- 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 4.8: 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 try and except blocks with imbedded while and for loops as well as

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 error handling when reading the file. The videos for this assignment were very informative and easy to follow when learning more about exceptions and dictionaries. Overall, this exercise was a great example of utilizing error handling while taking user input based off a menu and completing actions such as formatting said data, adding data to lists, printing lists as strings, and reading/writing to json files.