Erin Fox
11/13/2024
IT FDN 110
Assignment05
https://github.com/emfox55/IntroToProg-Python-Mod05

Advanced Collections and Error Handling

Introduction

In this assignment we were asked to expand on our prior work on files, collections of data and control flow using JSON files, dictionaries, and exception handling. In this document I will detail the steps I took to complete this task. All assignment instructions and informational materials that informed my approach can be found in the UW IT FDN 110 “Module05 - Advanced Collections and Error Handling” course materials.

Notes on Previous Documentation

The previous modules covered variables, constants, input/output, string formatting, saving data to files, loops, menus, conditional logic, and collections of data so I don’t go into detail on my approaches for them in this document. Similarly, the main functionality of this code (read data from a file, ask user for additional data, show user currently stored data, save all data back to file) was established in Assignment04, so I focus primarily on the new methods here. Please refer to the previous assignment documents for more detail. All development and testing was completed using the PyCharm IDE.

Reading & Writing JSON File Data

In order to get the existing dictionary data from the JSON file, we can either use readlines() to pull the file data, then use a for loop to parse each row into individual data components before re-formatting into a list of dictionary rows, *or* go easy on ourselves and use Python’s built-in JSON module (***Mod05 Notes - Working with JSON Files***). To access this module we must first “import json” in our script, then we have access to the json.dump() function to write the data to a table list. This does all the work of pulling, parsing, and storing the already properly configured data into our table list:

Figure 1: Using functions from the JSON module.

```
31  # When the program starts:
32  # Read the file data into a list of dictionary rows,
33  try:
34      file = open(FILE_NAME,'r')
35      students = json.load(file)
36      file.close()
```

This code is executed “automatically” at the beginning of our script, before displaying the Menu options to the end user.

The JSON module saves us some complexity again later when we write our table data, including any new entries, back to the JSON file.

Figure 2: Writing data using json.dump().

```
89      # Save the data to a file, with structured error handling
90      elif menu_choice == '3':
91          try:
92              file = open(FILE_NAME,'w')
93              json.dump(students,file)
94              file.close()
```

Error Handling

As mentioned previously, this code follows Assignment04 in reading and storing data from a file, gathering new data based on user input, displaying current data to the user, and writing all the data back to the file. Aside from using the JSON module, the biggest advancement is the addition of error, or exception, handling. When an error occurs during execution, it typically ends the whole program, and may or may not give very useful information back to a non-technical user to help troubleshoot. Exception handling allows us to use both built-in (ValueError, TypeError, etc.) and custom exception methods to control what happens when errors occur, and how information is conveyed to the user (***Mod05 Notes - Structure Error Handling (Try/Except)***).

In this assignment, we use Try/Except to refine the error output in the event of an invalid file name, and still close the file if the code errors out while the file is open. In Figure 3 we give a more detailed instruction to the user with the FileNotFoundError exception type, and include more detailed error information for any other kind of exception:

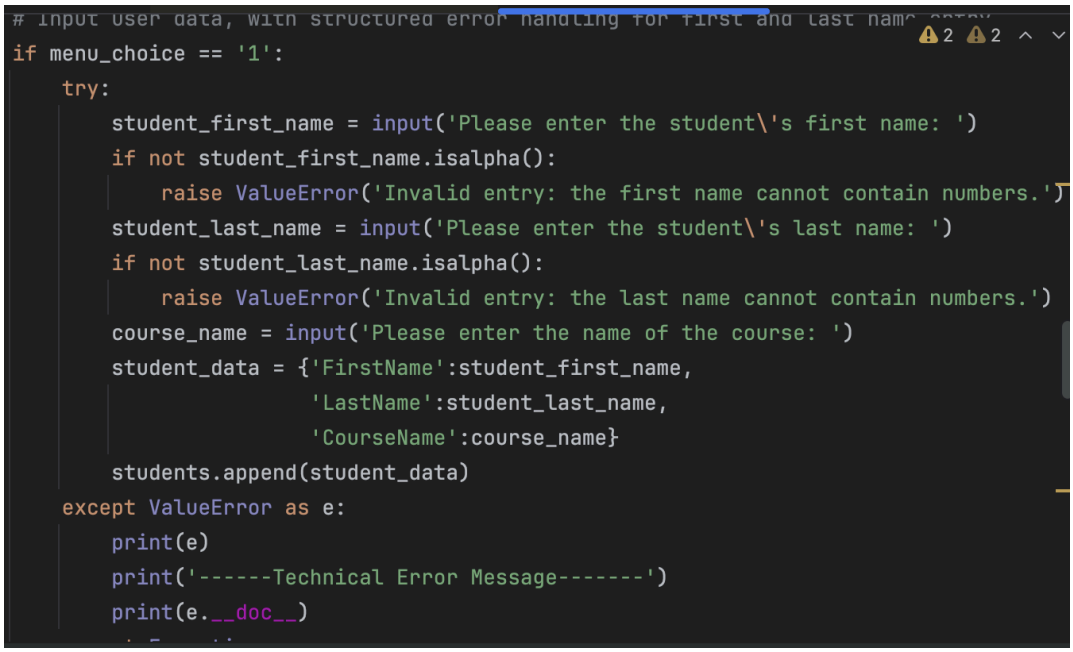
Figure 3: Try/except for file errors.

```
33  try:
34      file = open(FILE_NAME,'r')
35      students = json.load(file)
36      file.close()
37  except FileNotFoundError as e:
38      print('Text file must exist before running this script!')
39      print('-- Technical Error Message --')
40      print(e, e.__doc__, type(e), sep='\n')
41  except Exception as e:
42      print('There was a non-specific error!')
43      print('-- Technical Error Message -- ')
44      print(e, e.__doc__, type(e), sep='\n')
45  finally:
46      if file.closed == False:
47          file.close()
```

The “finally” functionality allows us to still close the file even if an exception is thrown.

Similar try/except functionality is used to disallow numeric characters in the user-input first name and last name, and help the user in the event of bad JSON data when writing to the file at the end.

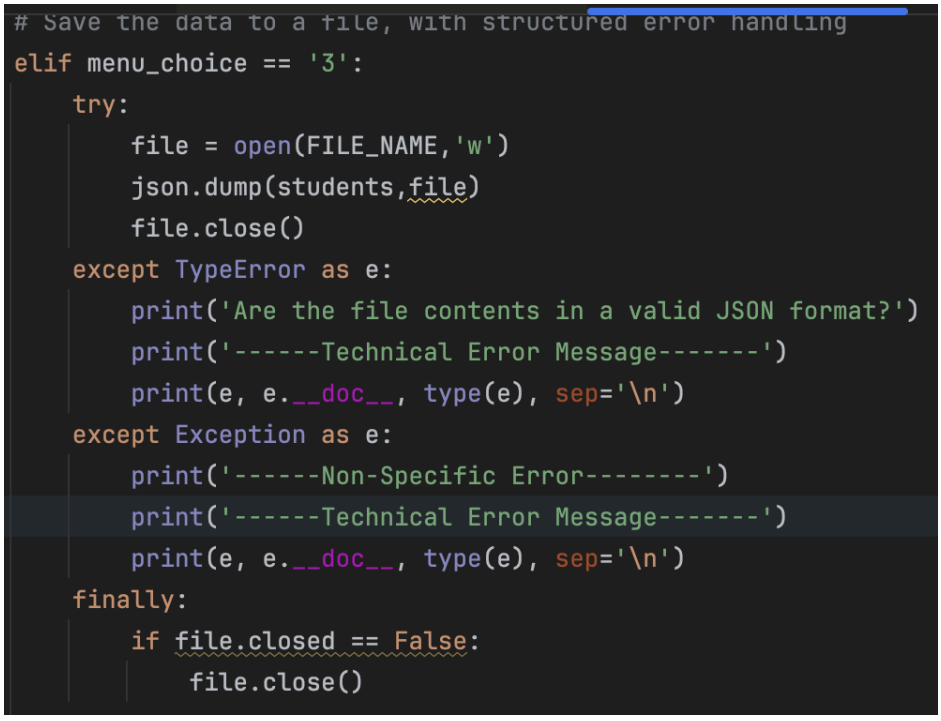
Figure 4: Try/except and raise error.



```
# Input user data, with structured error handling for first and last name entry
if menu_choice == '1':
    try:
        student_first_name = input('Please enter the student\'s first name: ')
        if not student_first_name.isalpha():
            raise ValueError('Invalid entry: the first name cannot contain numbers.')
        student_last_name = input('Please enter the student\'s last name: ')
        if not student_last_name.isalpha():
            raise ValueError('Invalid entry: the last name cannot contain numbers.')
        course_name = input('Please enter the name of the course: ')
        student_data = {'FirstName':student_first_name,
                        'LastName':student_last_name,
                        'CourseName':course_name}
        students.append(student_data)
    except ValueError as e:
        print(e)
        print('------Technical Error Message-------')
        print(e.__doc__)
```

In Figure 4 we use “raise error” to force Python to throw a ValueError when a numeric character is detected in student_first_name or student_last_name. This would not otherwise throw an exception to begin with.

Figure 5: Try/except in writing to file.



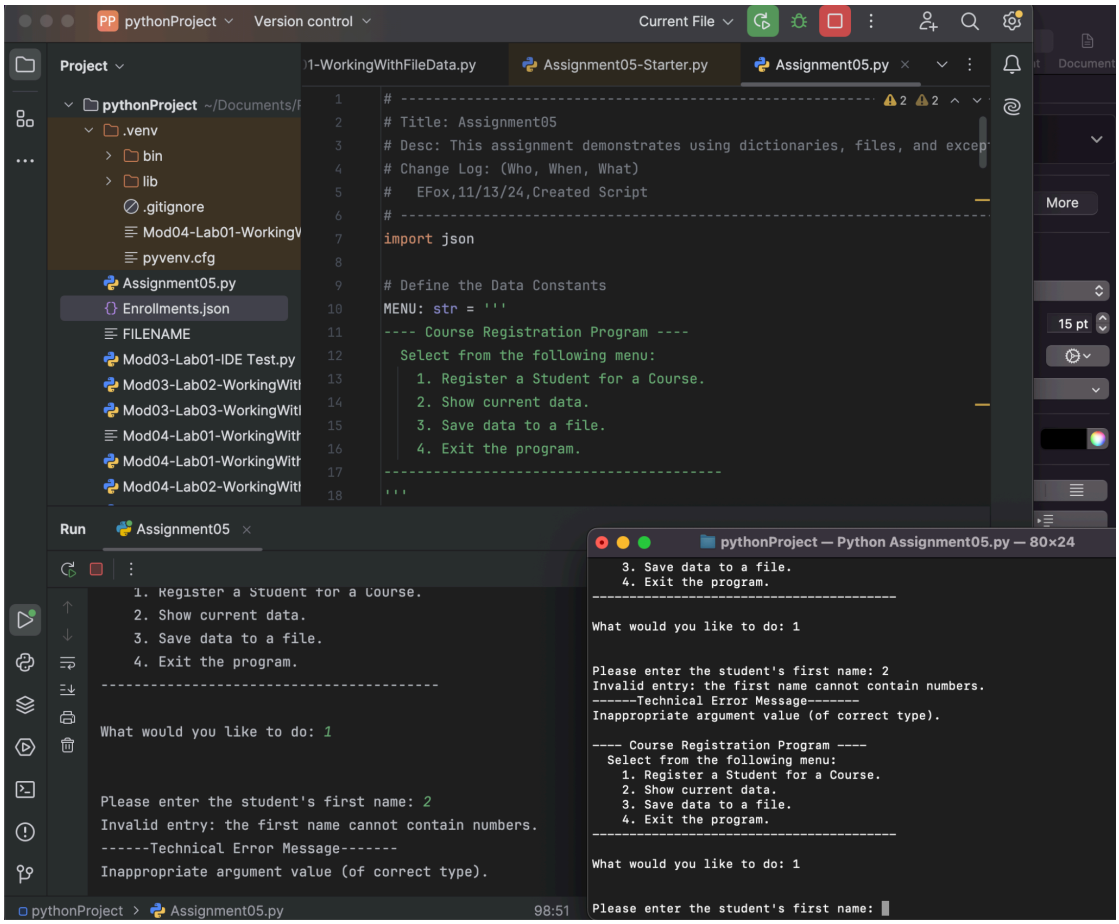
```
# Save the data to a file, with structured error handling
elif menu_choice == '3':
    try:
        file = open(FILE_NAME,'w')
        json.dump(students,file)
        file.close()
    except TypeError as e:
        print('Are the file contents in a valid JSON format?')
        print('------Technical Error Message-------')
        print(e, e.__doc__, type(e), sep='\n')
    except Exception as e:
        print('------Non-Specific Error--------')
        print('------Technical Error Message-------')
        print(e, e.__doc__, type(e), sep='\n')
    finally:
        if file.closed == False:
            file.close()
```

(***Mod05 Notes - Lab03 Working With Exceptions***)

Testing

I was able to successfully run this code in both PyCharm and Terminal

Figure 6: Testing code



Summary

Using the documents and videos provided in the Module05 course materials, I was able to understand and display how to use JSON files, dictionaries, and try/except for error handling. From the previous module work I used variables, constants, input/output, string formatting, saving data to files, loops, menus, conditional logic, and collections of data. I was able to validate the accuracy of my script by running it in both IDLE and Mac Terminal.