Sarah West

11/11/24

IT FDN 110 A

Assignment 05

https://github.com/sarahmwest/Python110_Fall2024

Advanced Collections with Error Handling

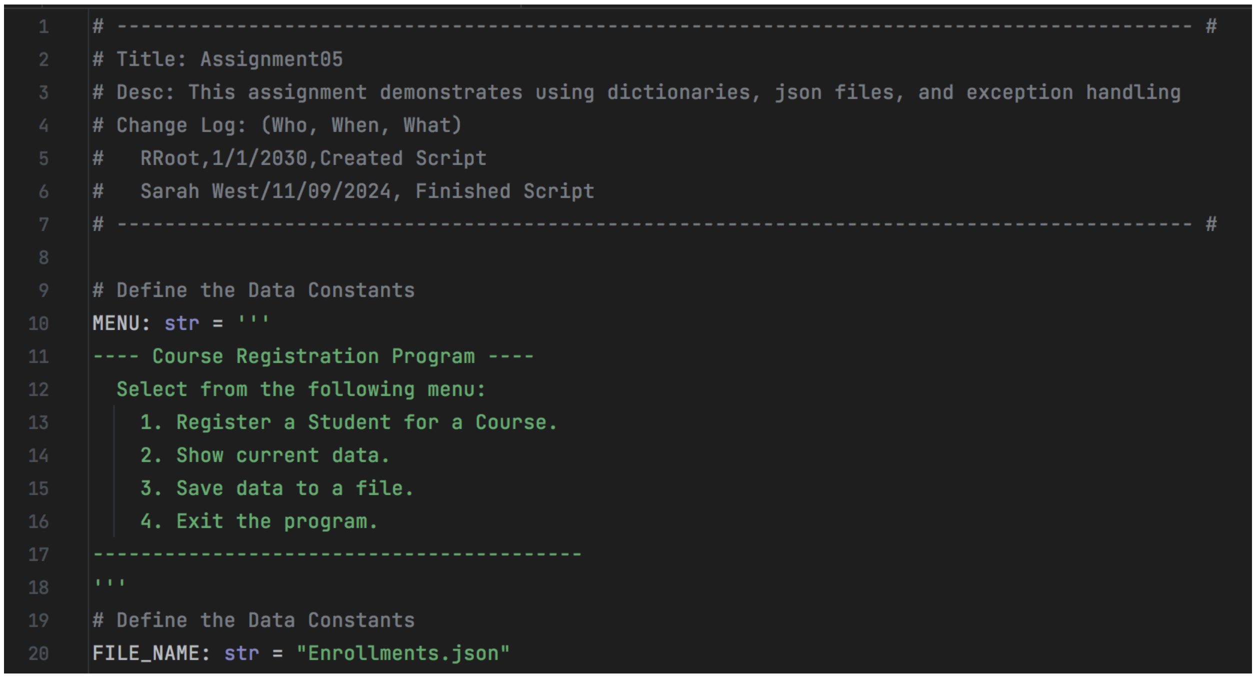
Introduction

In this assignment, I built an interactive program using a collection of dictionaries that reads and writes data to JSON (JavaScript Object Notation) files. The program allowed the user to select an option from a menu to input student data, read the current data from a file, and save the new data to a JSON file. I used dictionaries in a 2D array (i.e., a table) that states the student's first and last name and the course they are registered for from data in JSON file. I used structured error handling to catch common errors when reading and writing data to and from the file, and when collecting user imputed data. This assignment required us to code the script based off a starter file that already contained basic code.

Creating the Script

Script Header and Defining the Constants

I began by opening the "Assignment05-starter.py" that was provided us. Some basic pseudo-code and starter code was already written. I updated the script header to contain my information, and the constants (**Figure 1**). Since we are using JavaScript Object Notation (JSON) files to store the data, I changed the name of the Constant Enrollments.csv to Enrollments.json and created the JSON file. JSON files are convenient to use in this script since the program we are building is designed to work with dictionaries, which are similar to lists except that they use key-value pairs that allow for simple indexing. For example, instead of pulling out the second item in a list with index notation of "…[1]", we can use the dictionary key to specify what item in the dictionary we want. If we wanted to pull out the first name of a list, we could use the key notation: "…["FirstName"]. Since JSON files are formatted the same as Python dictionaries, using both together simplifies the code.

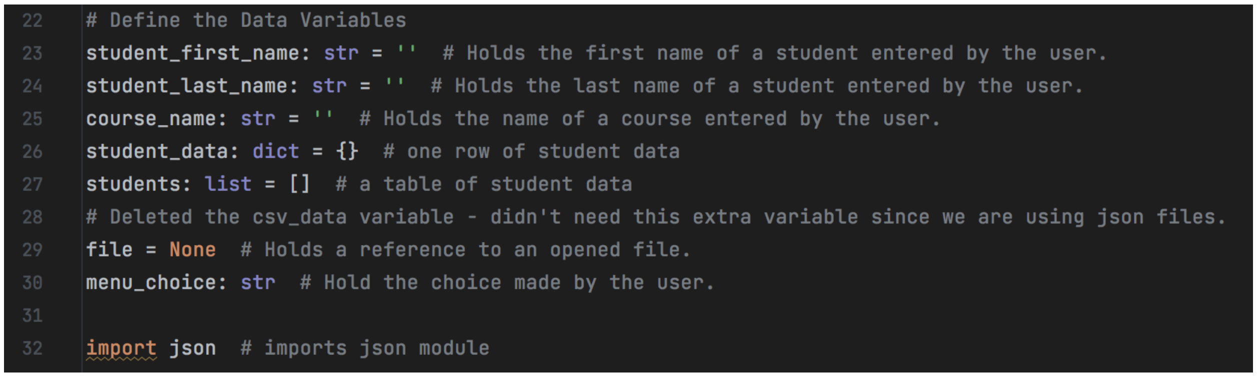


```python
# ------------------------------------------------------------------------------------------ #
# Title: Assignment05
# Desc: This assignment demonstrates using dictionaries, json files, and exception handling
# Change Log: (Who, When, What)
#   RRoot,1/1/2030,Created Script
#   Sarah West/11/09/2024, Finished Script
# ------------------------------------------------------------------------------------------ #

# 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.
-----------------------------------------
'''
# Define the Data Constants
FILE_NAME: str = "Enrollments.json"
```

Figure 1. A screenshot showing the updated script header and variables.

Defining the Variables

Defining the variables was simple since most were given to us in the starter file. As shown in **Figure 2**, I changed the variable student_data from an empty list to an empty dictonary indicated by the curly braces "{}". Also, I deleted the csv_data variable since I don't need that variable in this script. Lastly, I imported the json module which will allow me to use json specific statements in this script.

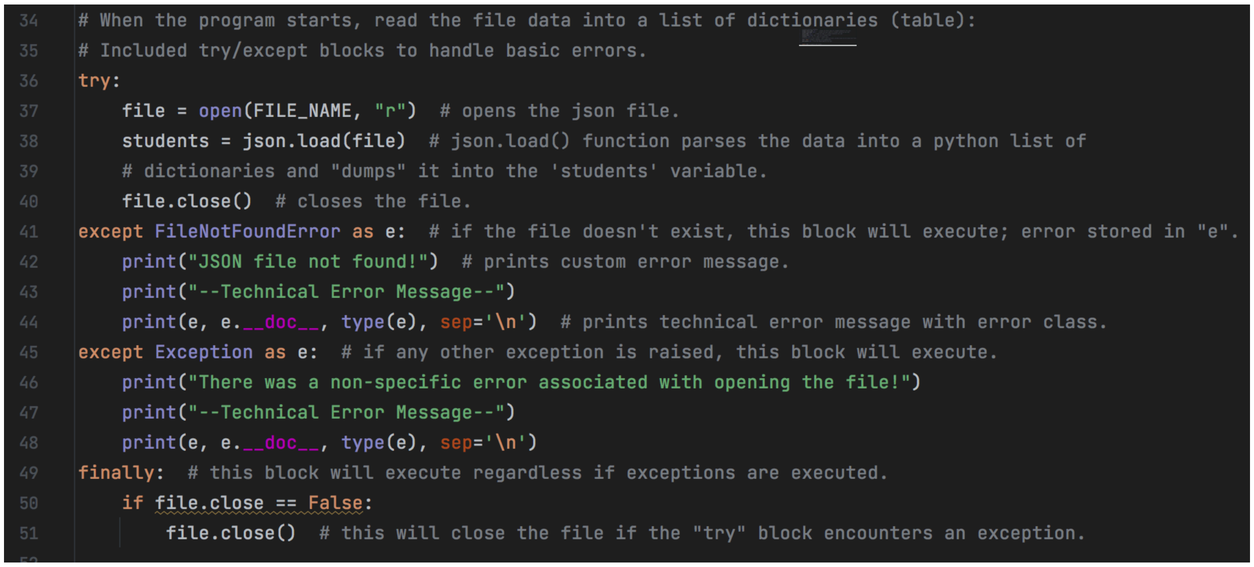


```python
# Define the Data Variables
student_first_name: str = ''  # Holds the first name of a student entered by the user.
student_last_name: str = ''  # Holds the last name of a student entered by the user.
course_name: str = ''  # Holds the name of a course entered by the user.
student_data: dict = {}  # one row of student data
students: list = []  # a table of student data
# Deleted the csv_data variable - didn't need this extra variable since we are using json files.
file = None  # Holds a reference to an opened file.
menu_choice: str  # Hold the choice made by the user.

import json  # imports json module
```

Figure 2. A screenshot showing the updated variables.

Reading the JSON File Data

After the constants and variables were defined, I moved on the block of code that reads the file "Enrollments.json" containing preexisting data (**Figure 3**). The JSON starter file contained some student information (i.e., student first name, student last name, and course name) to avoid errors when the program was inititated.



```python
# When the program starts, read the file data into a list of dictionaries (table):
# Included try/except blocks to handle basic errors.
try:
    file = open(FILE_NAME, "r")  # opens the json file.
    students = json.load(file)  # json.load() function parses the data into a python list of
    # dictionaries and "dumps" it into the 'students' variable.
    file.close()  # closes the file.
except FileNotFoundError as e:  # if the file doesn't exist, this block will execute; error stored in "e".
    print("JSON file not found!")  # prints custom error message.
    print("--Technical Error Message--")
    print(e, e.__doc__, type(e), sep='\n')  # prints technical error message with error class.
except Exception as e:  # if any other exception is raised, this block will execute.
    print("There was a non-specific error associated with opening the file!")
    print("--Technical Error Message--")
    print(e, e.__doc__, type(e), sep='\n')
finally:  # this block will execute regardless if exceptions are executed.
    if file.close == False:
        file.close()  # this will close the file if the "try" block encounters an exception.
```

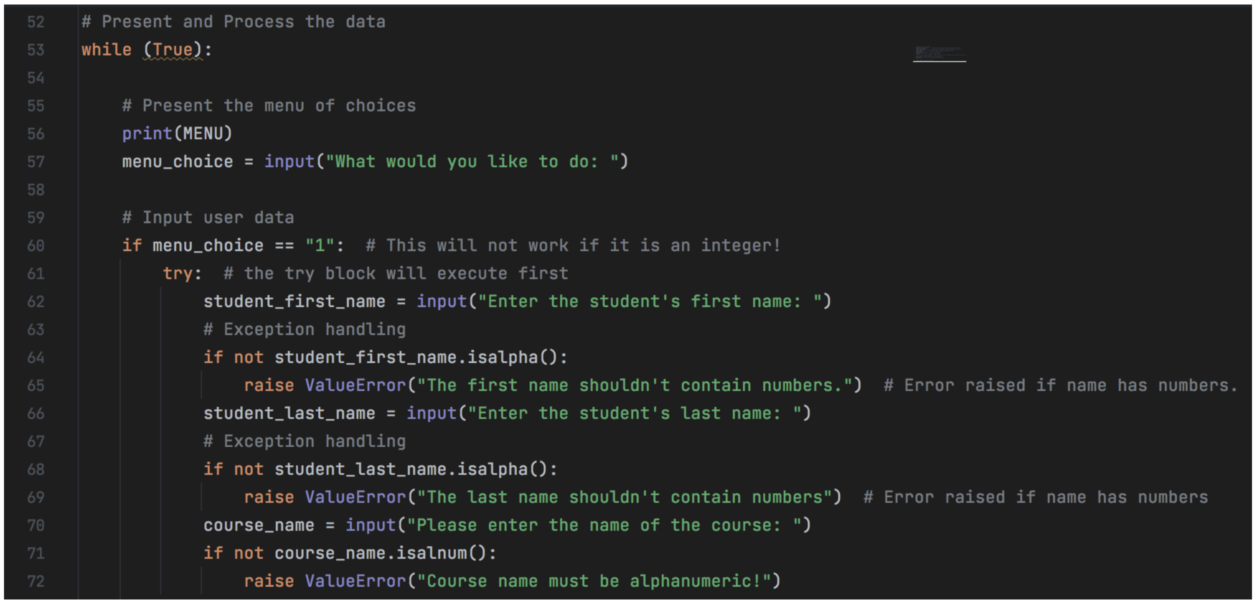
Figure 3. A screenshot showing the code that reads the json file upon program initiation.

Upon initiation, the program opens the JSON file and reads the data using the open() funtion with the read "r" permission. Then, the program will load and parse the JSON file data into a Python dictonary and store the data in the students variable using the json.load() funtion. Of course, the file is then closed with the close() function.

To handle errors that may occur, I have nested this whole block of code into a try/except block (**Figure 3**). In the try/except block, the program will first try to run the block of code in the "try" block, and it will be successful if there are no errors encountered. However, if there is an error (i.e., an exception) found, the "except" blocks will run sqeuntially. I chose to call a FileNotFound error since this may be the most common type of error when reading from a file. In this case, if the program tries to read from a file that doesn't exist in the location of the python script, or if the name of the file doesn't match the value of FILE_NAME, then the first except block raises this error and stores the error in the variable "e". Then, I coded the except block to tell the user using print() statements that the JSON file was not found and to show the technical error message. Specifically, the error "e" will be printed, followed by the docstring telling the user what the error is, followed by the error type. This information will be useful for the user to troubleshoot the error. Next, I included a generic exception as Exception. This is a non-specific error (a catch-all error) in case a different error is found (not a FileNotFound error). The program will then print out the custom message followed by the technical message. Lastly, I included a "finally" block which will execute regardless if the exeptions execute. In this case, the program will close the file and save the data, just to ensure that any data in the file is not lost. Note that the use of try/except blocks allows the program to continue running without crashing as it normally would when exceptions are encountered.

Menu Choice Number 1 (Data Collection)

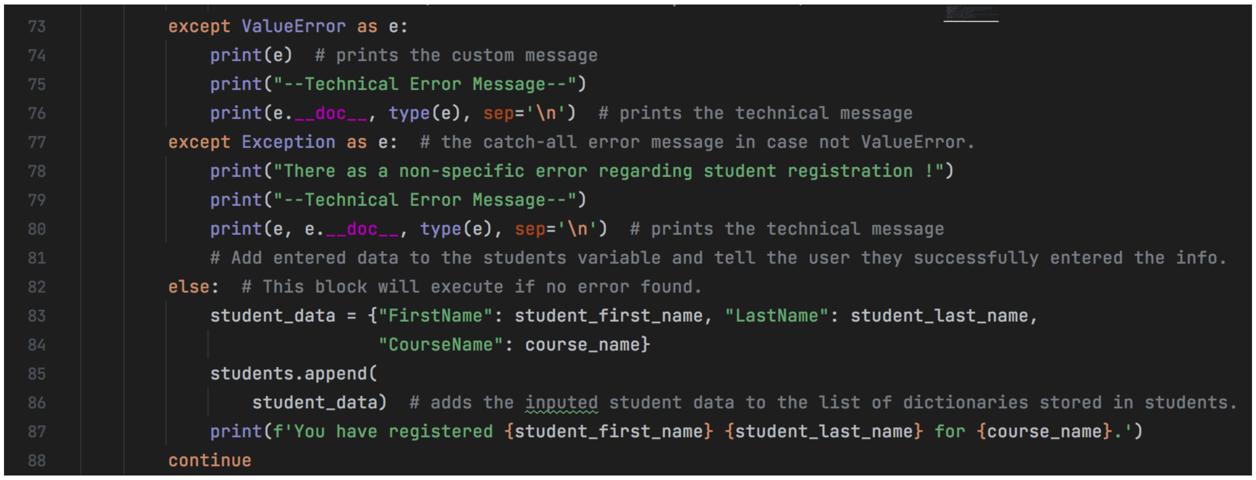
The interactive menu that allows the user to select an option is controlled with the while loop. The starter file contained the basics of this while loop. Menu choice 1 asks the user to register a new student for the course using input() funtions. I nested the input options in a try/except block to handle errors that could occur when entering data (**Figure 4**).



```python
# Present and Process the data
while (True):

    # Present the menu of choices
    print(MENU)
    menu_choice = input("What would you like to do: ")

    # Input user data
    if menu_choice == "1":  # This will not work if it is an integer!
        try:  # the try block will execute first
            student_first_name = input("Enter the student's first name: ")
            # Exception handling
            if not student_first_name.isalpha():
                raise ValueError("The first name shouldn't contain numbers.")  # Error raised if name has numbers.
            student_last_name = input("Enter the student's last name: ")
            # Exception handling
            if not student_last_name.isalpha():
                raise ValueError("The last name shouldn't contain numbers")  # Error raised if name has numbers
            course_name = input("Please enter the name of the course: ")
            if not course_name.isalnum():
                raise ValueError("Course name must be alphanumeric!")
```

Figure 4. A screenshot showing the first half of the code for menu choice 1.

The try block asks the user to enter the first name, last name, and course that the student is registered for. For each, I added code that would raise ValueErrors if the inputted strings contained numbers (for student's name) or was not alphanumeric (for course number). If these errors were raised, the program would execute the first except block shown below.



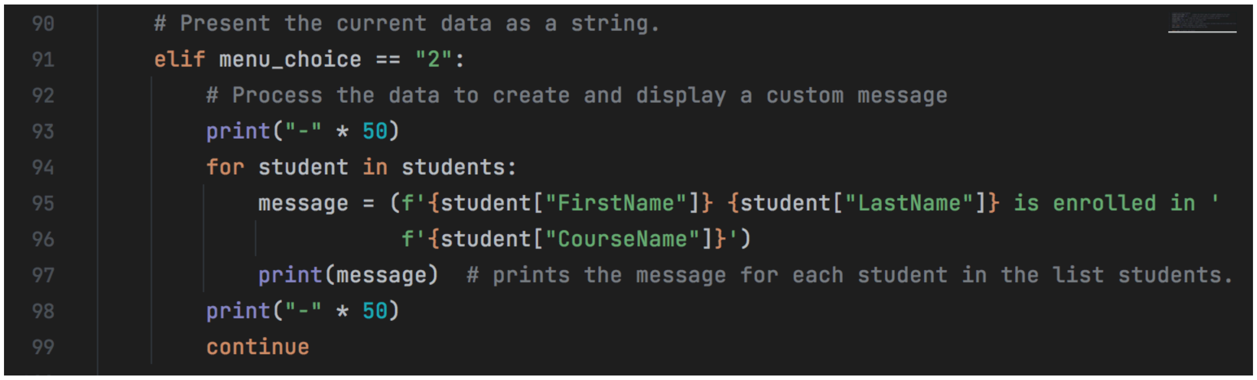
```python
        except ValueError as e:
            print(e)  # prints the custom message
            print("--Technical Error Message--")
            print(e.__doc__, type(e), sep='\n')  # prints the technical message
        except Exception as e:  # the catch-all error message in case not ValueError.
            print("There as a non-specific error regarding student registration !")
            print("--Technical Error Message--")
            print(e, e.__doc__, type(e), sep='\n')  # prints the technical message
            # Add entered data to the students variable and tell the user they successfully entered the info.
        else:  # This block will execute if no error found.
            student_data = {"FirstName": student_first_name, "LastName": student_last_name,
                            "CourseName": course_name}
            students.append(
                student_data)  # adds the inputed student data to the list of dictionaries stored in students.
            print(f'You have registered {student_first_name} {student_last_name} for {course_name}.')
        continue
```

Figure 5. A screenshot showing the second half of the code for menu choice 1.

Figure 5 shows the continuation of the try/except block. If the inputed information raises a ValueError, then the first except block will run, which prints the custom message for the rasied ValueError shown in **Figure 4** above. The except block will also print the technical error message. If any other exception is raised (i.e., not a ValueError), then the second except block will run, which is the catch-all Exception error. If no error is found, then the "else" block will run, which stores the inputted data in the student_data variable and appends it to the students variable. A print statement is added to let the user know the student was sucessfully registered for the class. Of course, the continue statement will bring the user back to the start of the while loop to select another menu choice.

Menu Choice Number 2 (Print the Current Data)

Menu choice 2 prints the current data stored in the students variable (**Figure 6).** The for loop was already present in the starter file, however it was formatted for csv files and lists. I edited the message to reflect a dictionary format since JSON files use Python dictionary formats. In this case, I changed the list indexing ([0], [1], and [2]) to the dictionary keys to call the student's first name, last name, and course. These keys will call the respective values in the JSON file for each student and therefore each student's data will be printed out for the user.

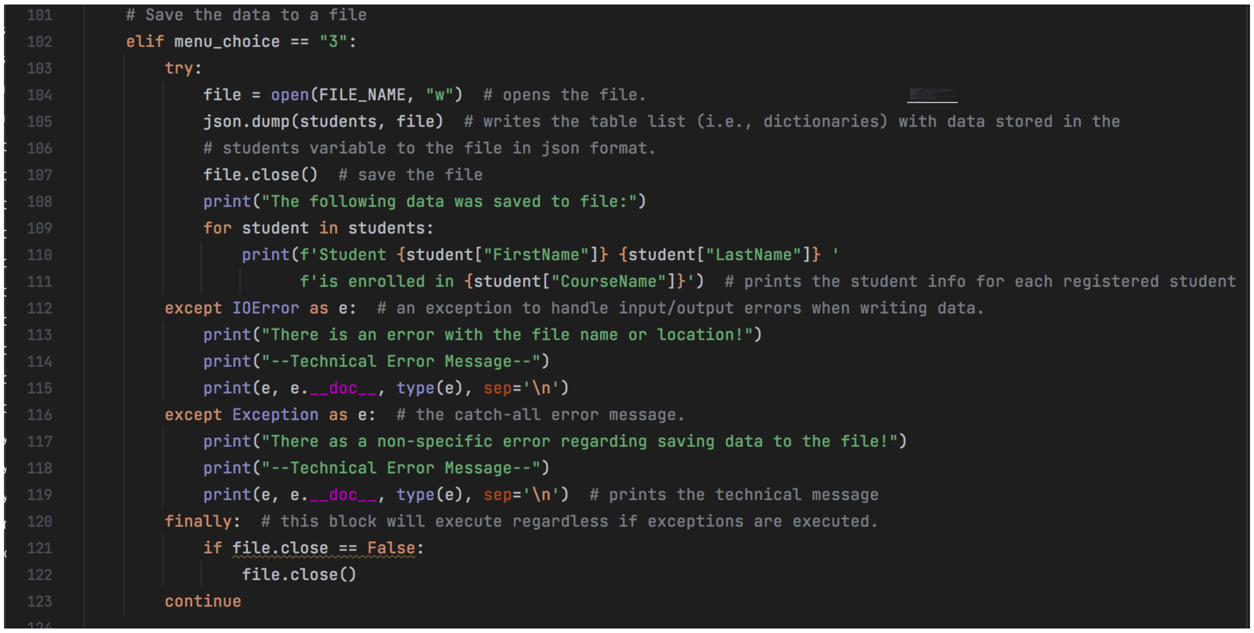


```python
    # Present the current data as a string.
    elif menu_choice == "2":
        # Process the data to create and display a custom message
        print("-" * 50)
        for student in students:
            message = (f'{student["FirstName"]} {student["LastName"]} is enrolled in '
                       f'{student["CourseName"]}')
            print(message)  # prints the message for each student in the list students.
        print("-" * 50)
        continue
```

Figure 6. A screenshot of the blocks of code for menu choices 2.

Menu Choice Numbers 3 and 4 (Save the Data to a File and Exit the Program)

Menu choice 3 saves the inputted information into the JSON file that contains previously entered information. **Figure 7** shows the blocks of code for menu choice 3.



```python
    # Save the data to a file
    elif menu_choice == "3":
        try:
            file = open(FILE_NAME, "w")  # opens the file.
            json.dump(students, file)  # writes the table list (i.e., dictionaries) with data stored in the
            # students variable to the file in json format.
            file.close()  # save the file
            print("The following data was saved to file:")
            for student in students:
                print(f'Student {student["FirstName"]} {student["LastName"]} '
                      f'is enrolled in {student["CourseName"]}')  # prints the student info for each registered student
        except IOError as e:  # an exception to handle input/output errors when writing data.
            print("There is an error with the file name or location!")
            print("--Technical Error Message--")
            print(e, e.__doc__, type(e), sep='\n')
        except Exception as e:  # the catch-all error message.
            print("There as a non-specific error regarding saving data to the file!")
            print("--Technical Error Message--")
            print(e, e.__doc__, type(e), sep='\n')  # prints the technical message
        finally:  # this block will execute regardless if exceptions are executed.
            if file.close == False:
                file.close()
        continue
```

Figure 7. A screenshot of the block of code for menu choice 3.

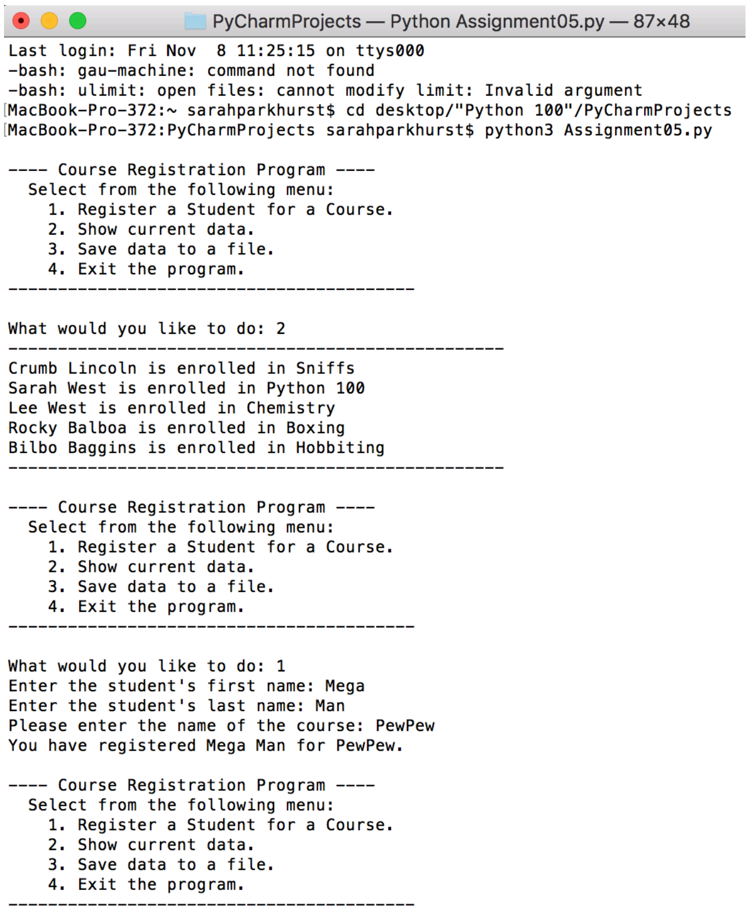
In menu choice 3, I wanted the program to add the new data that was entered in menu choice 1 to the JSON file while preserving the data that was already stored in the file. The starting code already contained the open() function with the "w" permission, which is appropriate to use in this case since all data originally in the file was previously stored in the students variable when the program started running and will be re-written into the file. After opening the file, I changed the starter file code to reflect adding data to a JSON file rather than to a csv file. I used the json.dump(students,file) statement to tell the program to "dump" or add the data from the students variable into the opened file. I updated the subsequent print() statement in the for loop to reflect dictionary formatting instead of the list formatting that was present in the starter file.

I added two except blocks to catch errors when writing the file. The first is an IOError, which handles input/output errors such as a problem with the file name or location. I also added the general Exception error in the next except block to handle all generic errors. For each, I added a custom error message followed by the technical error information. Lastly, I added a finally block which closes the file and saves the data if an error was encountered.

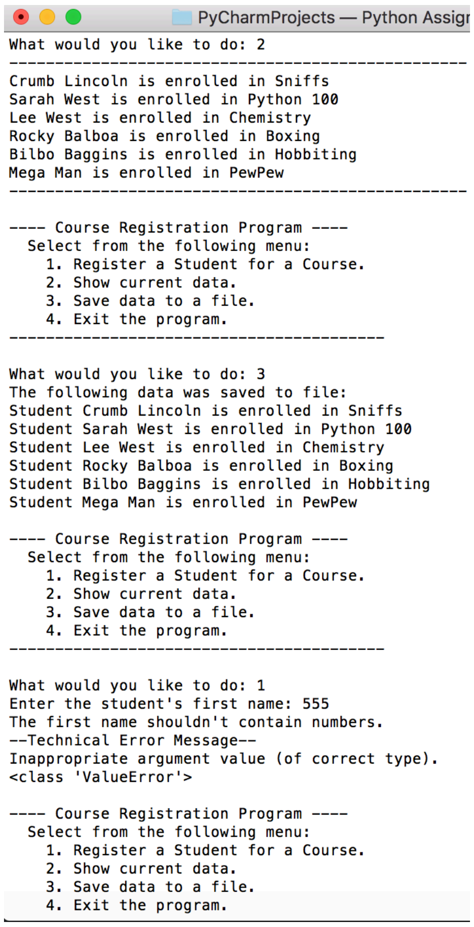
The only thing I edited for menu choice 4 is the print() statement stating to choose options 1,2,3 or 4 (as opposed to options 1,2 or 3 that was originally written).

Testing the Script in PyCharm and MacOS

I tested the script by running it in PyCharm, and it works great! I also tested the error handling by purposefully entering incorrect names, or changing the file name. Note that I used the debug option in PyCharm to test the script throughout the writing instead of using intermediate print statements like I have in the past. After I confirmed the script runs as expected in PyCharm, I confirmed that the script runs successfully from the terminal on my MacOS, which is shown in **Figure 8** below. I also confirmed that the program correctly enters the data into the JSON file, which it does! Note that there are a lot of options to show the output of this script, so I am just showing the use of the script as intended without errors as well as the output of a ValueError in **Figure 8**.



```
Last login: Fri Nov  8 11:25:15 on ttys000
-bash: gau-machine: command not found
-bash: ulimit: open files: cannot modify limit: Invalid argument
MacBook-Pro-372:~ sarahparkhurst$ cd desktop/"Python 100"/PyCharmProjects
MacBook-Pro-372:PyCharmProjects sarahparkhurst$ python3 Assignment05.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.
-----------------------------------------

What would you like to do: 2
--------------------------------------------------
Crumb Lincoln is enrolled in Sniffs
Sarah West is enrolled in Python 100
Lee West is enrolled in Chemistry
Rocky Balboa is enrolled in Boxing
Bilbo Baggins is enrolled in Hobbiting
--------------------------------------------------

---- 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: Mega
Enter the student's last name: Man
Please enter the name of the course: PewPew
You have registered Mega Man for PewPew.

---- 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
--------------------------------------------------
Crumb Lincoln is enrolled in Sniffs
Sarah West is enrolled in Python 100
Lee West is enrolled in Chemistry
Rocky Balboa is enrolled in Boxing
Bilbo Baggins is enrolled in Hobbiting
Mega Man is enrolled in PewPew
--------------------------------------------------

---- 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 Crumb Lincoln is enrolled in Sniffs
Student Sarah West is enrolled in Python 100
Student Lee West is enrolled in Chemistry
Student Rocky Balboa is enrolled in Boxing
Student Bilbo Baggins is enrolled in Hobbiting
Student Mega Man is enrolled in PewPew

---- 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: 555
The first name shouldn't contain numbers.
--Technical Error Message--
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 8. Screenshots showing the output of the script running in MacOS terminal.

Summary

In this assignment, I successfully created a python script that takes a starting JSON file, collects new information inputted by the user, formats the information into a list of dictionaries, and adds that information to the JSON file without deleting the old information. I used the script that was provided to us with preliminary code and pseudo-code and added my code to that script. I also used the debugging tool in PyCharm to test out my new code as I was writing it, and I am happy to report that the script runs as intended!