Bryan Pannell

5/15/2024

IT FDN 110 A

Assignment 05

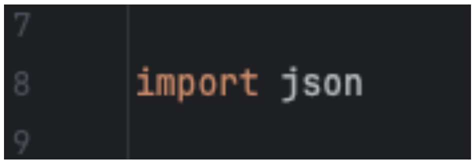
Dictionaries, JSON Files, and Error Handling

Introduction

Things are getting complicated! Last week, we worked with data collections and continued that this week by adding in dictionaries to the mix. Much of the structure of this week's code is transferred from last week's but there are some critical changes. Some pieces of the code compressed a little. By working with the JSON file format, a lot of the file operations were simplified. Python's built-in JSON capabilities made importing and saving the data more streamlined. Core functionality in this week's script is similar to last week's. Dictionaries work much like lists but define the "table" of data with more clarity by identifying the "column headers" using keys. Data is still collected using imported file data and a menu system and user input. Data is then saved in the new JSON format to a file. Error handling to catch specific errors with the file processing and user input is also new this time. The script delivers a data set of student and course registration at the end of the process.

Setting the Stage to Work with JSON files

To be able to work with JSON files, the script must import Python's built-in JSON capabilities. Being able to do this greatly simplifies working with this file type and allows for a wide variety of operations to be performed using the data and files.

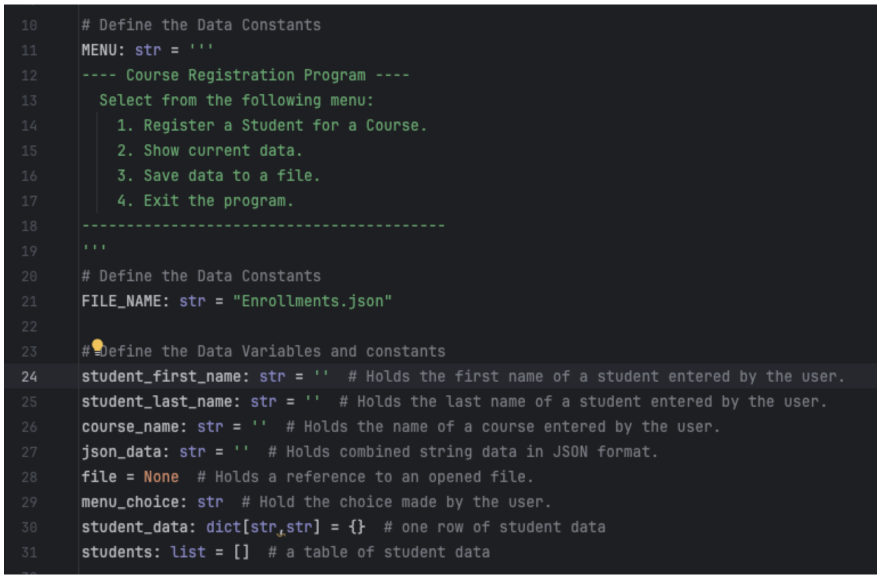


```

import json

```

Figure 1: Simple code to import Python's built-in JSON functionality

Proper Script Structure and Comments

Proper documentation is even more important as the script becomes more complex. We continue to work on script documentation and adding in the appropriate comments and pseudo code. The provided starter files for the course assignments provide a good example of pseudo code and comments that are easy to build from (See Figure 2).



```python
# 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"

# Define the Data Variables and constants
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.
json_data: str = ''  # Holds combined string data in JSON format.
file = None  # Holds a reference to an opened file.
menu_choice: str  # Hold the choice made by the user.
student_data: dict[str,str] = {}  # one row of student data
students: list = []  # a table of student data
```

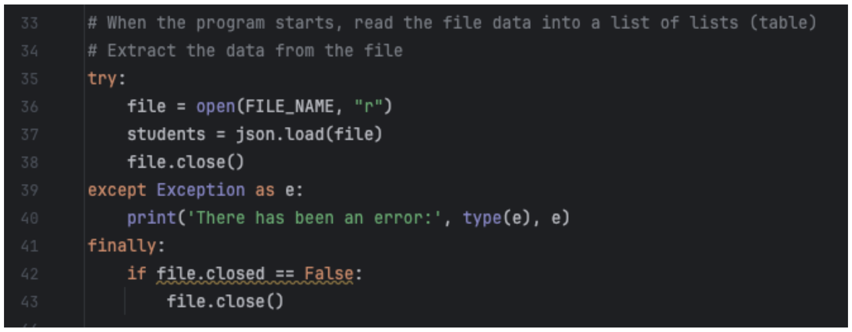
Figure 2: Example of pseudo code and defining constants and variables

Defining Constants and Variables

Per usual procedure, the code starts off with defining the constants and variables needed to perform the functions in the assignment requirements. Doing this upfront makes it easier to see which elements are required and gives better alignment with the logic flow of the script. This week's assignment uses similar constants and variables as last week's script but adds on dictionaries, another type of data collection. See Figure 1 above for the constants and variables defined at the beginning of this week's script.

Accessing data already stored in a JSON file

Just as we did with last's week's CSV file, we accessed and imported data from an existing JSON file. To accomplish this, we must have the script open the file, read the file contents, and then move the file into the working memory. Because JSON capability has been imported for use by the script, the file can simply be load in its current format. To help identify any issues with accessing and reading the file contents into memory, error handling was added to notify the user of the issue, including a message that an error has occurred, the type of error, and the error report. The script then closes the file. Here is the code to make this happen:

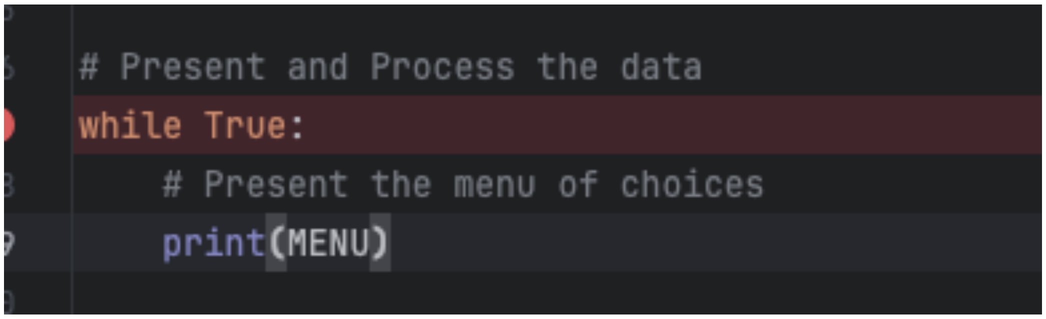


```python
# When the program starts, read the file data into a list of lists (table)
# Extract the data from the file
try:
    file = open(FILE_NAME, "r")
    students = json.load(file)
    file.close()
except Exception as e:
    print('There has been an error:', type(e), e)
finally:
    if file.closed == False:
        file.close()
```

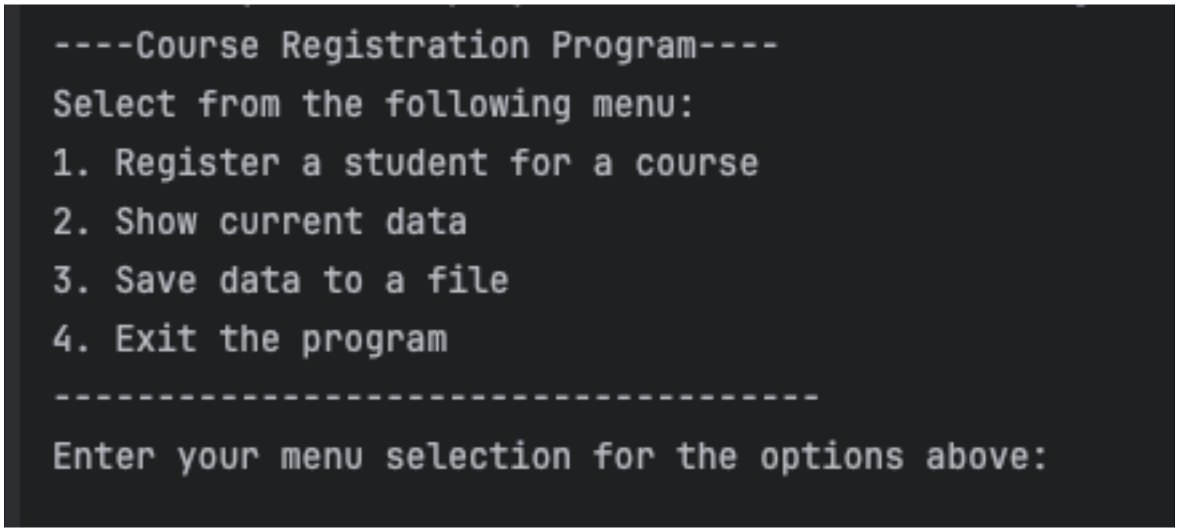
Figure 3: Example of code that opens, reads, and provides access to existing data in a JSON file and delivers error information if an issue occurs in the process

Presenting and Using the Menu

This week's assignment continues use of the menu system from previous scripts to allow the user to select from a variety of menu options to perform various functions: enter input, view the current data, save that input to a file, and exit the script. The contents of the menu are set as a defined constant. The menu is meant to run on a loop until the user opts to exit using option 4. To do this, a "while" loop is set to run until that occurs. The menu is displayed in each iteration of the loop using the print() function. The figure below shows the code for this and what the menu looks like when it is run.



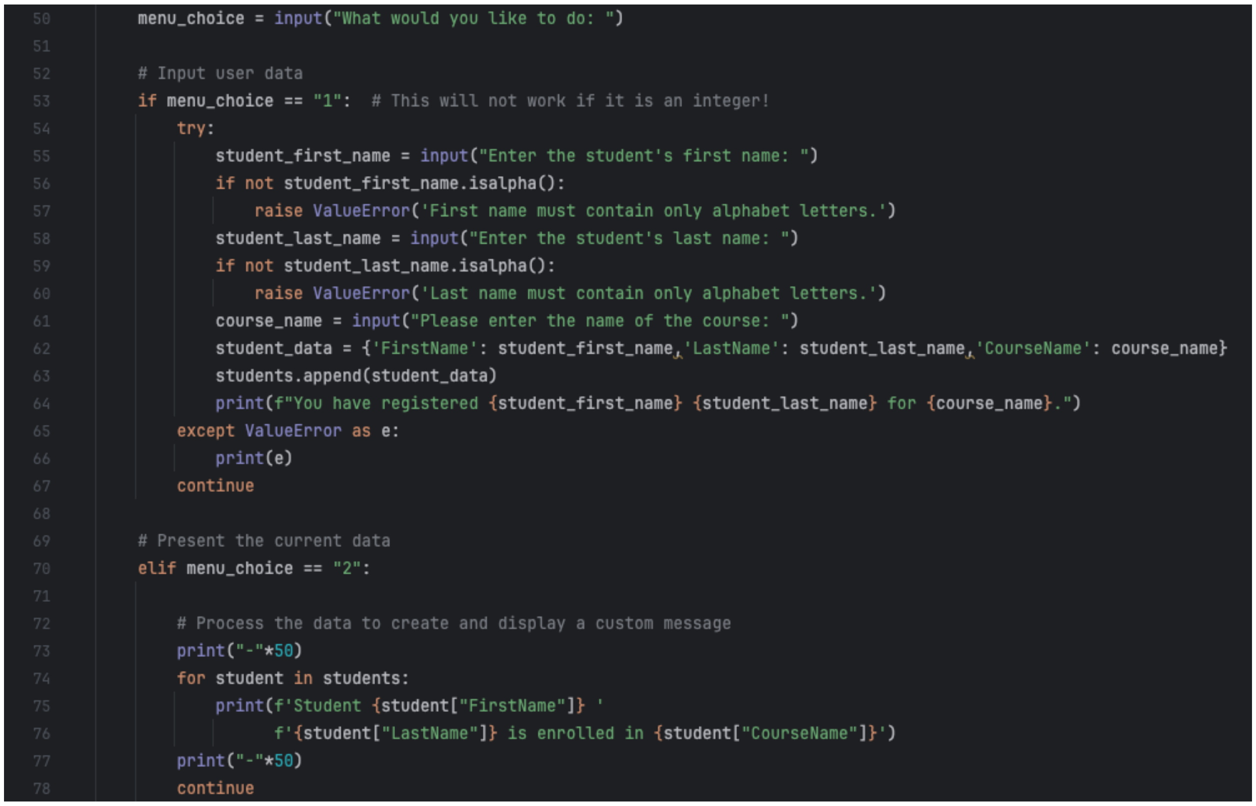
```
# Present and Process the data
while True:
    # Present the menu of choices
    print(MENU)
```



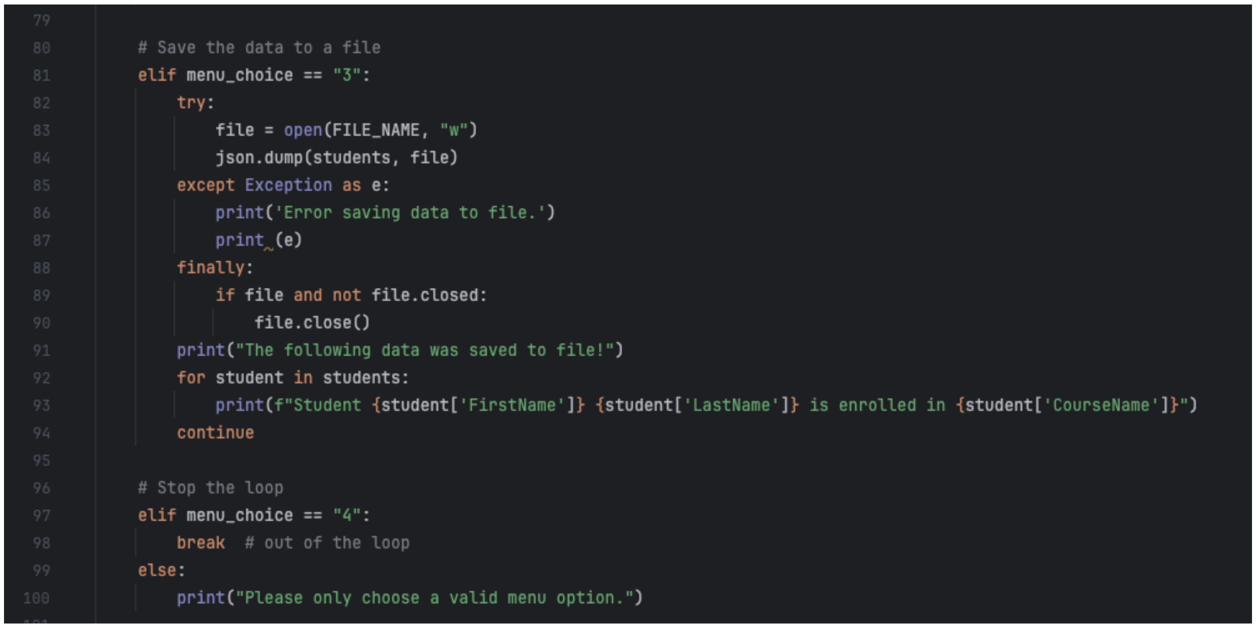
```
----Course Registration Program----
Select from the following menu:
1. Register a student for a course
2. Show current data
3. Save data to a file
4. Exit the program
-----------------------------------
Enter your menu selection for the options above:
```

Figure 4: Python code example to start a "while" loop and display the menu and the displayed menu

The user can enter a menu option on each iteration. Just like last week, menu options are tied to code that defines the function for each option. The user's selection is prompted using the "menu_choice" variable and an input statement. To accomplish this, a set of "if" and "elif" statements are used to guide the code through the process. There is also an "else" statement to catch any invalid selections. The figure below shows this structure:



```python
    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:
            student_first_name = input("Enter the student's first name: ")
            if not student_first_name.isalpha():
                raise ValueError('First name must contain only alphabet letters.')
            student_last_name = input("Enter the student's last name: ")
            if not student_last_name.isalpha():
                raise ValueError('Last name must contain only alphabet letters.')
            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)
            print(f"You have registered {student_first_name} {student_last_name} for {course_name}.")
        except ValueError as e:
            print(e)
        continue

    # Present the current data
    elif menu_choice == "2":

        # Process the data to create and display a custom message
        print("-"*50)
        for student in students:
            print(f'Student {student["FirstName"]} '
                  f'{student["LastName"]} is enrolled in {student["CourseName"]}')
        print("-"*50)
        continue
```



```python

    # Save the data to a file
    elif menu_choice == "3":
        try:
            file = open(FILE_NAME, "w")
            json.dump(students, file)
        except Exception as e:
            print('Error saving data to file.')
            print (e)
        finally:
            if file and not file.closed:
                file.close()
        print("The following data was saved to file!")
        for student in students:
            print(f"Student {student['FirstName']} {student['LastName']} is enrolled in {student['CourseName']}")
        continue

    # Stop the loop
    elif menu_choice == "4":
        break  # out of the loop
    else:
        print("Please only choose a valid menu option.")
```

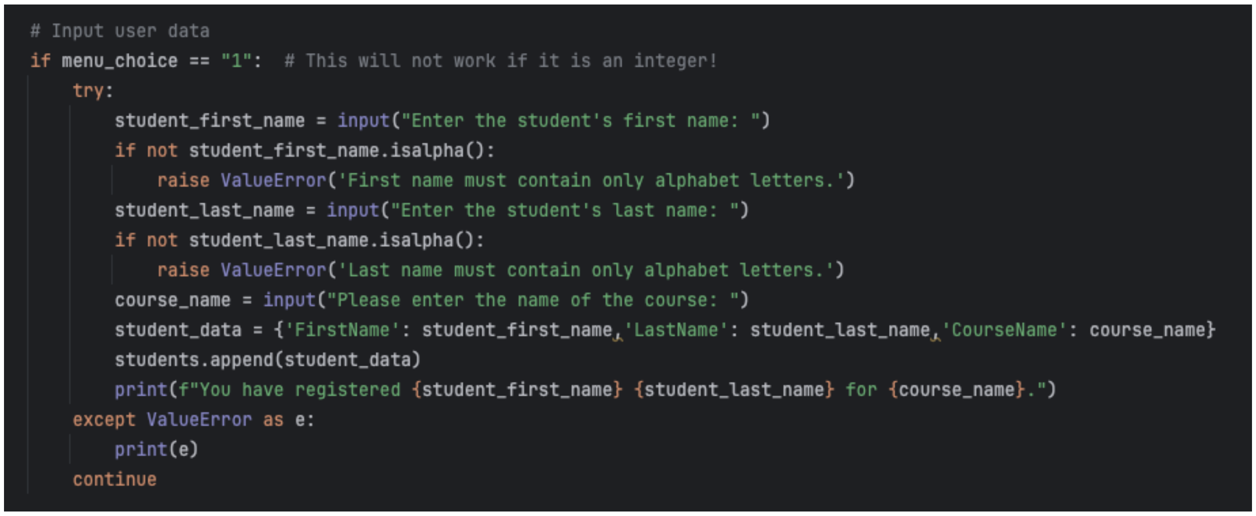
Figure 5: Python code example for menu processing

Menu Options

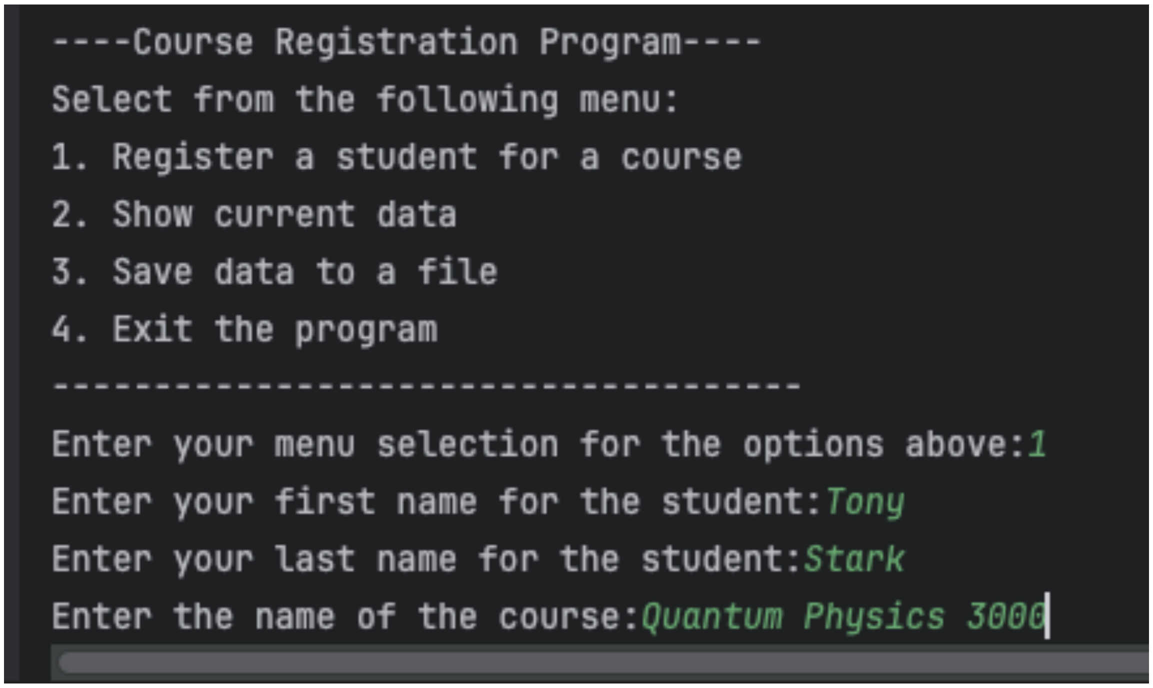
Option 1: Input data

Option 1 allows the user to input three distinct items: student first name, student last name, and the course name. These three data elements comprise a row of data that can be saved. Each row can be viewed as a collection of data elements. These can be stored as lists and dictionaries. When the user selects option 1, three prompts for the items run sequentially. Once the data is collected, the three corresponding variables are stored in the student_data dictionary and then formatted as needed to store the data in a JSON file. The data is also appended to the students list.

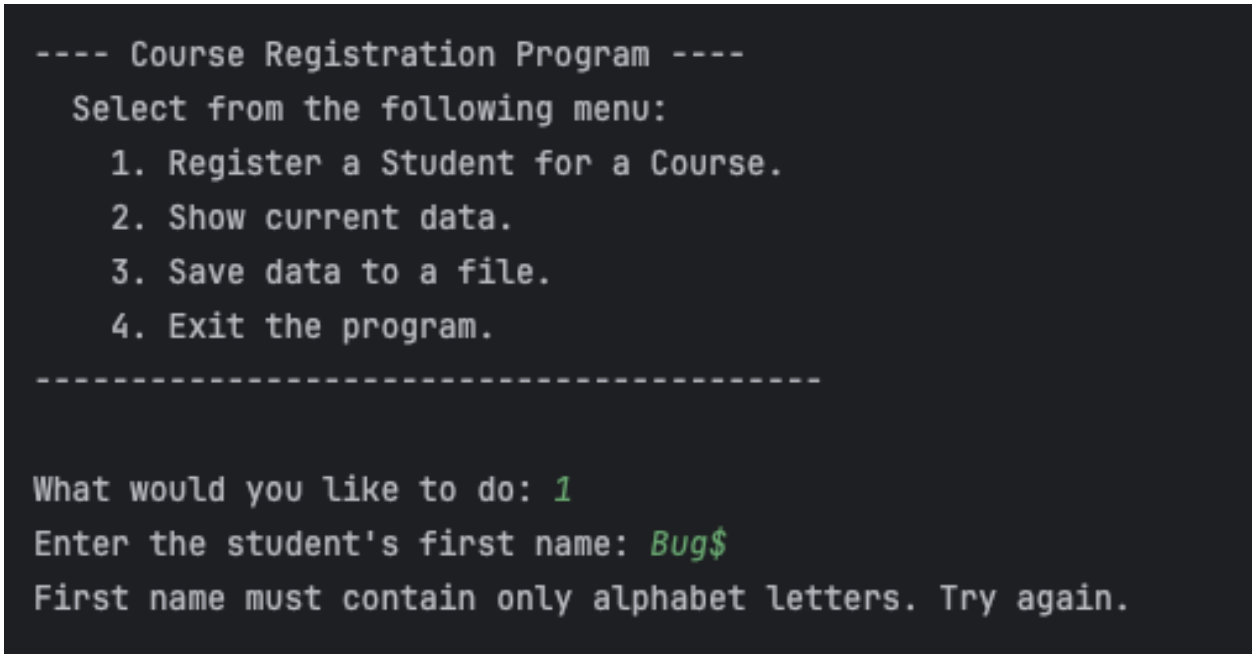
To prevent user input error for the first and last name elements, error handling has been added to ensure that only alphabet letters are used when entering these names. No other character types are allowed. If the user enters any other type of character, an error is generated and displayed onscreen, and the menu is reloaded. Figure 6 shows the code to accomplish this, sample input, and an error message for invalid input for:



```python
# Input user data
if menu_choice == "1":  # This will not work if it is an integer!
    try:
        student_first_name = input("Enter the student's first name: ")
        if not student_first_name.isalpha():
            raise ValueError('First name must contain only alphabet letters.')
        student_last_name = input("Enter the student's last name: ")
        if not student_last_name.isalpha():
            raise ValueError('Last name must contain only alphabet letters.')
        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)
        print(f"You have registered {student_first_name} {student_last_name} for {course_name}.")
    except ValueError as e:
        print(e)
    continue
```



```
----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 selection for the options above:1
Enter your first name for the student:Tony
Enter your last name for the student:Stark
Enter the name of the course:Quantum Physics 3000
```

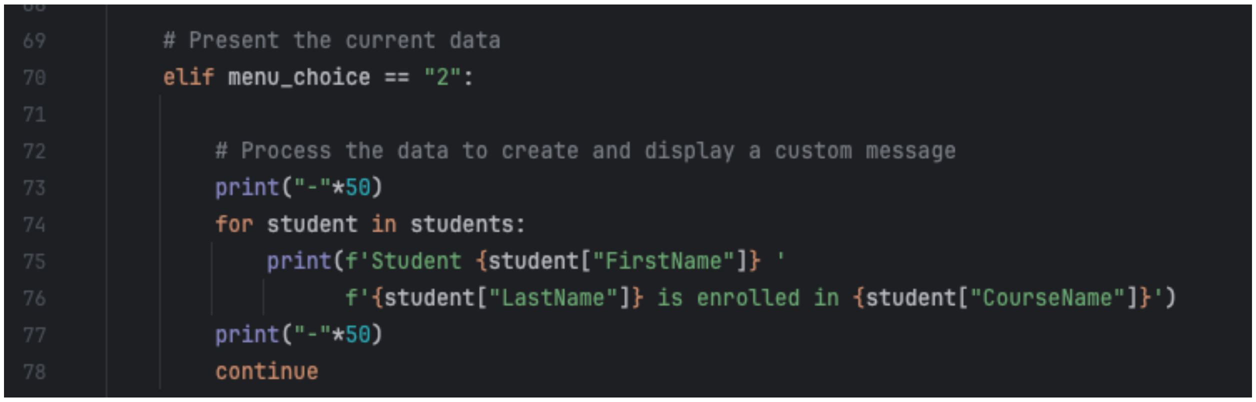


```
---- Course Registration Program ----
  Select from the following menu:
    1. Register a Student for a Course.
    2. Show current data.
    3. Save data to a file.
    4. Exit the program.
-----------------------------------------

What would you like to do: 1
Enter the student's first name: Bug$
First name must contain only alphabet letters. Try again.
```

Figure 6: Python code example for collecting the data from the user, what this looks like onscreen, and an error message for invalid input

Option 2: Display the current data

The functionality of the second menu option is essentially the same as last week. The only difference is how the data is handled. Because we are using a dictionary to store the data, we must display it using the appropriate syntax for that data collection type. The output is formatted to give the user the information from each row in the data table. Just as last week, a formatted string is generated using the print() function to display the data. Figure 7 contains the code for this option and sample output.



```python
    # Present the current data
    elif menu_choice == "2":

        # Process the data to create and display a custom message
        print("-"*50)
        for student in students:
            print(f'Student {student["FirstName"]} '
                  f'{student["LastName"]} is enrolled in {student["CourseName"]}')
        print("-"*50)
        continue
```

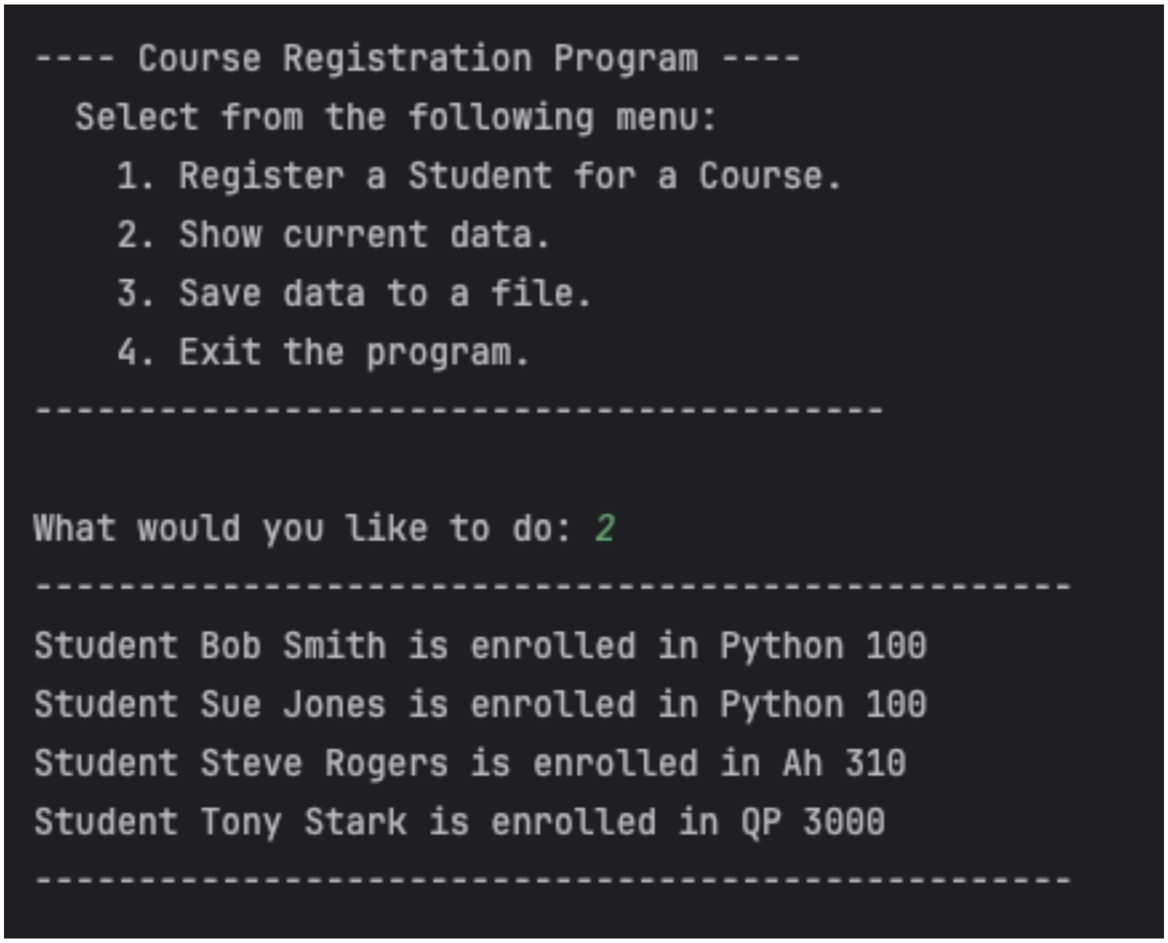
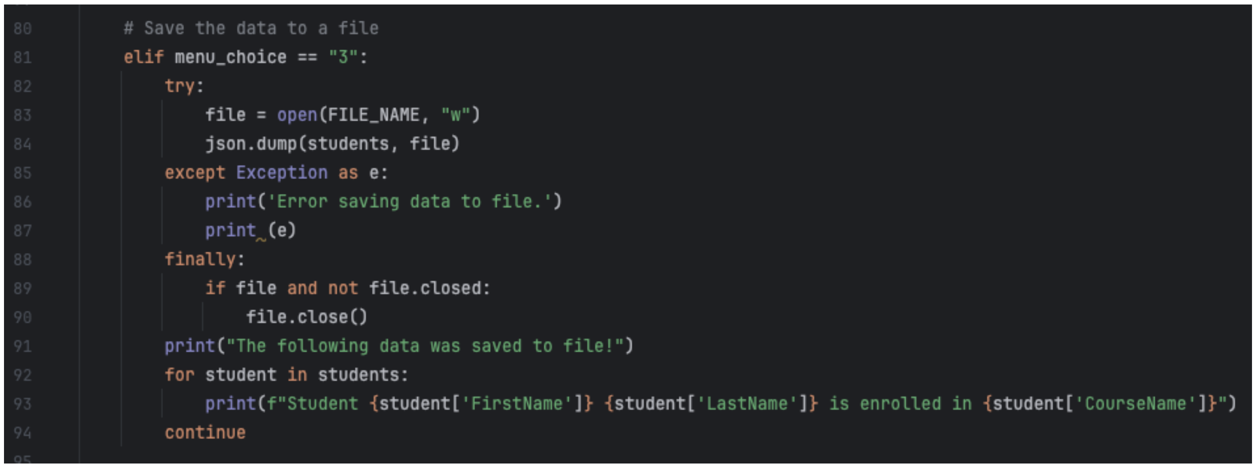

```
---- Course Registration Program ----
  Select from the following menu:
    1. Register a Student for a Course.
    2. Show current data.
    3. Save data to a file.
    4. Exit the program.
-----------------------------------------


What would you like to do: 2
--------------------------------------------------
Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Steve Rogers is enrolled in Ah 310
Student Tony Stark is enrolled in QP 3000
--------------------------------------------------
```

Figure 7: Python code example for displaying the data in a specified format and its corresponding output

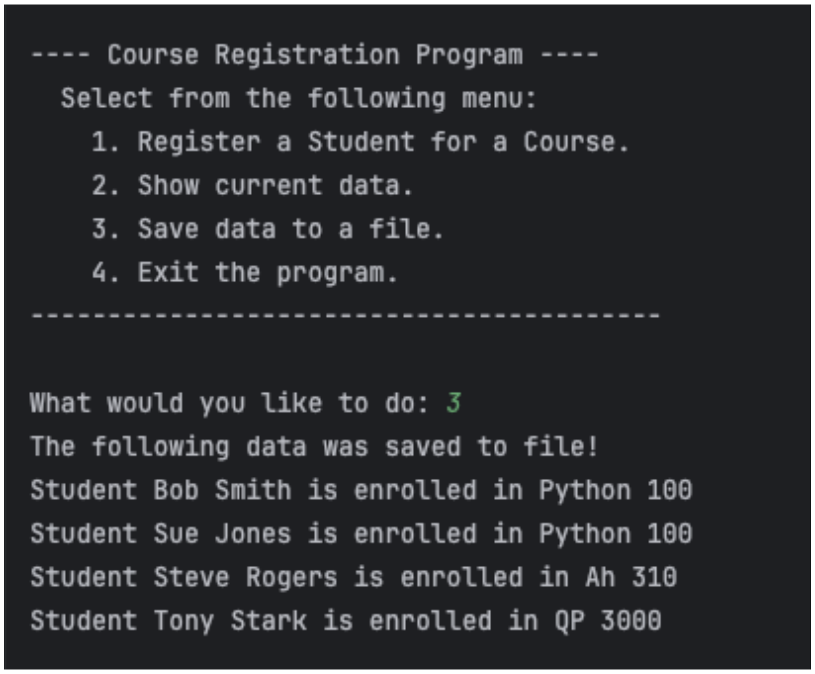
Option 3: Saving the data to a JSON file

Using option 3 in the menu, compared to the CSV coding, this is much simpler. Because of Python's built-in JSON capabilities, selecting option 3 performs four primary instructions (not including error handling). The code opens the file in "write" mode, performs a data dump using the json.dump() function to move the data from memory to the file, closes the file, and finally prints the resulting data to the screen. Error handling was added let the user know if there is a problem with the procedure that results in the file not being updated with the data. The json.dump() function is great because it eliminates the need to have a complete loop structure to move each row of data to the file. It is much more streamlined. The following figures show how this is accomplished and the results.



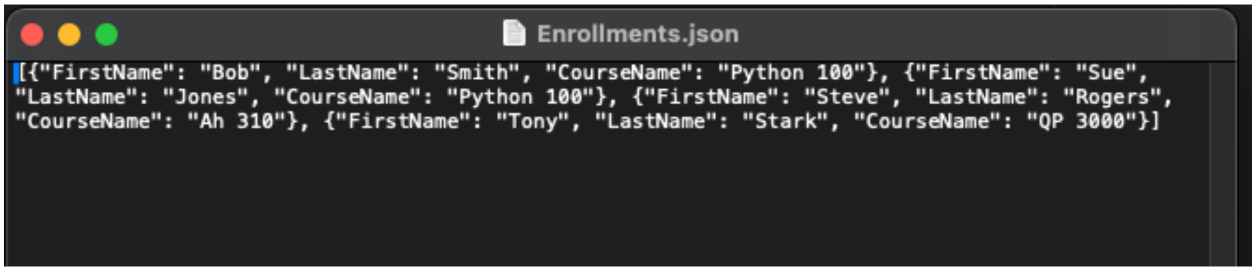
```python
    # Save the data to a file
    elif menu_choice == "3":
        try:
            file = open(FILE_NAME, "w")
            json.dump(students, file)
        except Exception as e:
            print('Error saving data to file.')
            print (e)
        finally:
            if file and not file.closed:
                file.close()
        print("The following data was saved to file!")
        for student in students:
            print(f"Student {student['FirstName']} {student['LastName']} is enrolled in {student['CourseName']}")
        continue
```

Figure 8: Python code example for saving input to a JSON file



```
---- 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 Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Steve Rogers is enrolled in Ah 310
Student Tony Stark is enrolled in QP 3000
```

Figure 9: Script report for saving data to JSON file

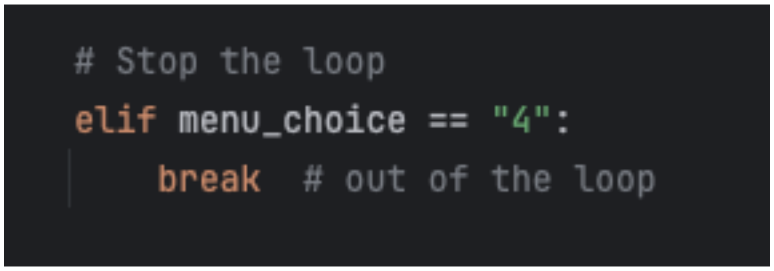


```json
[{"FirstName": "Bob", "LastName": "Smith", "CourseName": "Python 100"}, {"FirstName": "Sue",
"LastName": "Jones", "CourseName": "Python 100"}, {"FirstName": "Steve", "LastName": "Rogers",
"CourseName": "Ah 310"}, {"FirstName": "Tony", "LastName": "Stark", "CourseName": "QP 3000"}]
```

Figure 10: JSON data file sample

Option 4: Exit

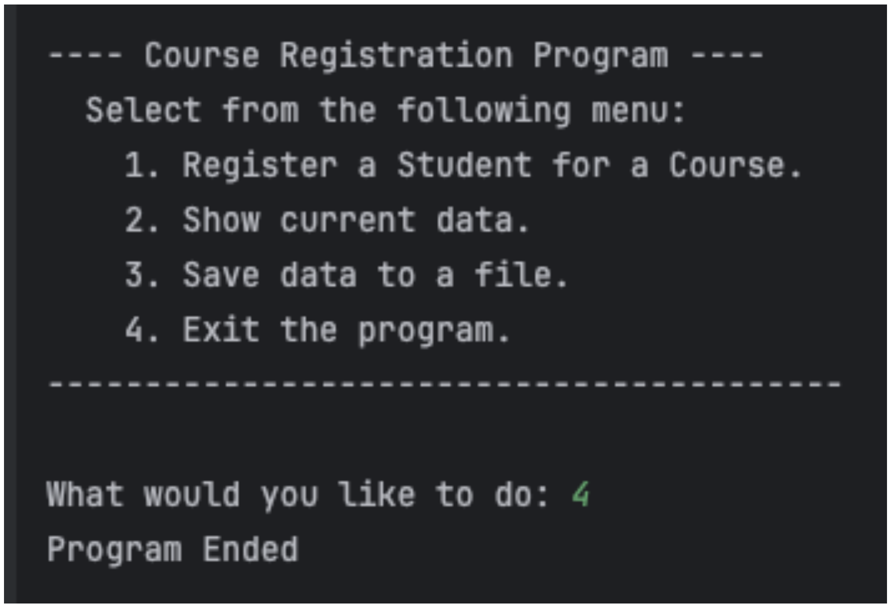
Option 4 is the most basic of the four options (and the only one I didn't have to change!). It simply stops the script by exiting the loop. Here is the code for this function:



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

Figure 11: Sample Python code to end the loop and exit the script

Once the loop is exited, adding a simple print() line with a message such as "Program Ended" lets the user know the session is over.

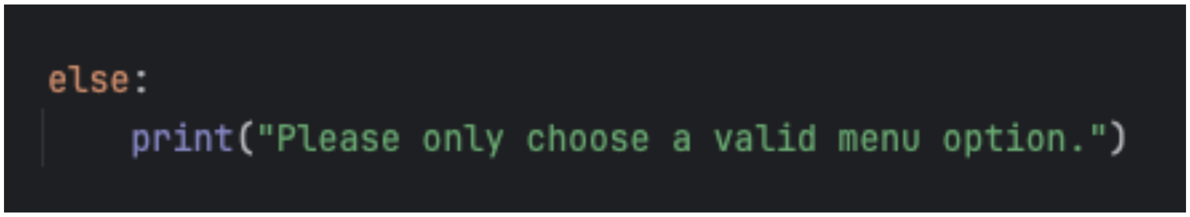


```
---- Course Registration Program ----
  Select from the following menu:
    1. Register a Student for a Course.
    2. Show current data.
    3. Save data to a file.
    4. Exit the program.
-----------------------------------------


What would you like to do: 4
Program Ended
```

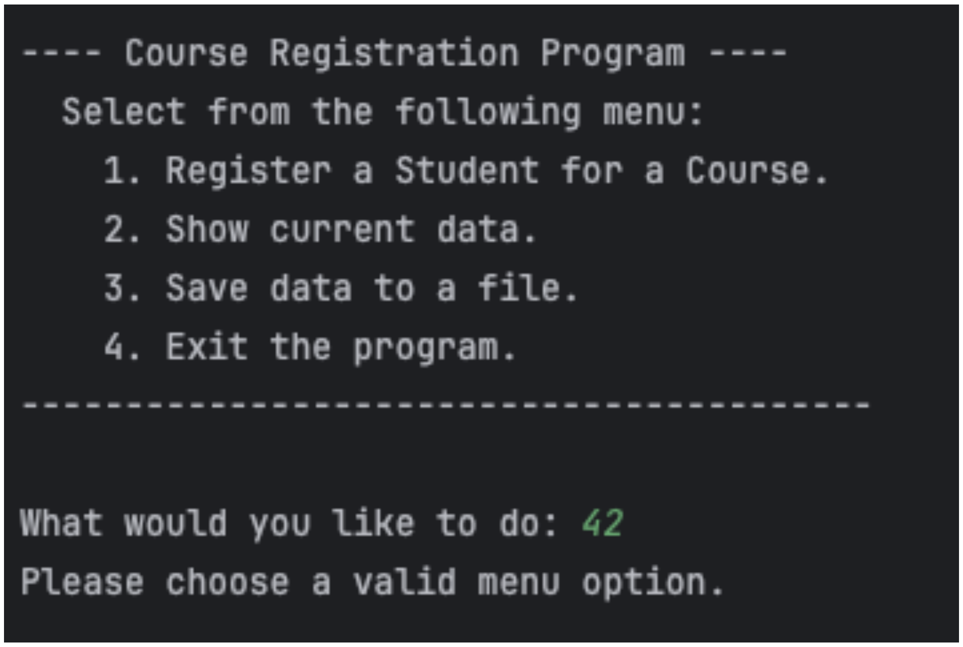
Figure 12: Sample exit message

Making sure that selected menu options are valid

Users can easily make mistakes when providing input. To account for invalid menu entries, the if/elif statement structure can be concluded with an else statement that stipulates that if the user's input does not match one of the defined options, an error message is displayed, and the cycle begins again.



```
else:
    print("Please only choose a valid 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: 42
Please choose a valid menu option.
```

Figure 13: Sample of an error message for an invalid menu selection

Summary

This week's assignment expanded our work with data sets to include dictionaries and JSON files. These additions simplified some file operations for moving data back and forth from a JSON file. Dictionaries also added clarity to how we work with data, if only by helping to imagine that data in a table structure with headers and rows of data. Error handling was introduced this week as well to help identify and possibly prevent errors with file operations and user input. With each new week, the script gets closer to something that could be functional in an operational environment.