Jameson Poth

August 13, 2025

IT FDN 110 A Su 25

Assignment05

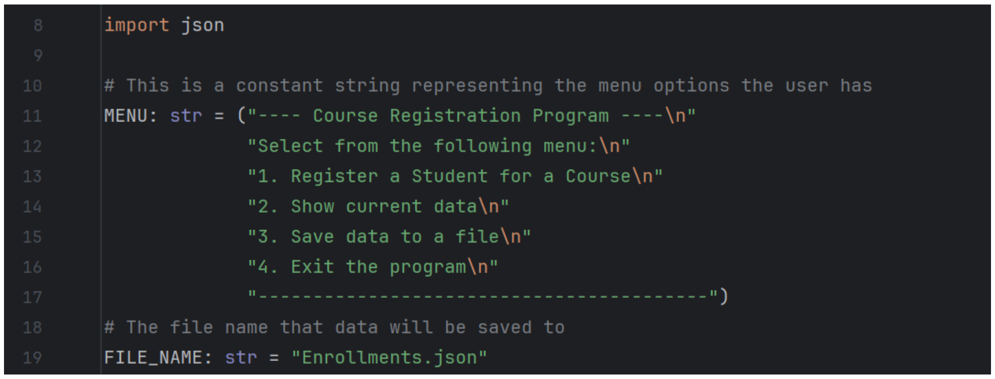
Module 5

Introduction

This module was designed to familiarize the coder with formatting data, reading and writing to JSON data files, and working with dictionaries. The focus was on taking user inputs to create a data string containing the student's name and course name and saving the data to a data file. This module also looked at handling errors and exceptions using a structured format.

Writing the Code

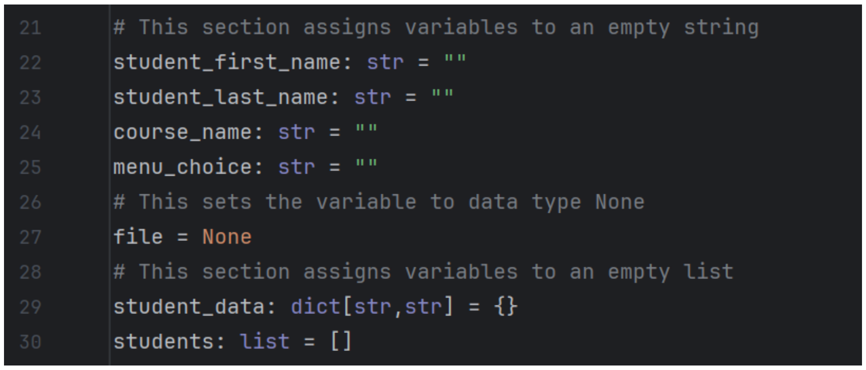
This code was written and developed in the PyCharm Community Edition 2025.1.3.1 Integrated Development Environment (IDE). The code starts out by importing the JSON module which allows for the encoding and decoding of JSON data. JSON stands for JavaScript Object Notation. This code contains two constants with the names "MENU" and "FILE_NAME". Both constants are of datatype string, and this is indicated by adding ": str =" after the constant name. The constant "MENU" is formatted to span across multiple lines, and this is done by using "\n" at the end of each line. Figure 1 shows the definitions of the constants, as well as their data types.



```
import json

# This is a constant string representing the menu options the user has
MENU: str = ("---- Course Registration Program ----\n"
             "Select from the following menu:\n"
             "1. Register a Student for a Course\n"
             "2. Show current data\n"
             "3. Save data to a file\n"
             "4. Exit the program\n"
             "-----------------------------------------")
# The file name that data will be saved to
FILE_NAME: str = "Enrollments.json"
```

Figure 1: Code Showing Constants and Their Data Types

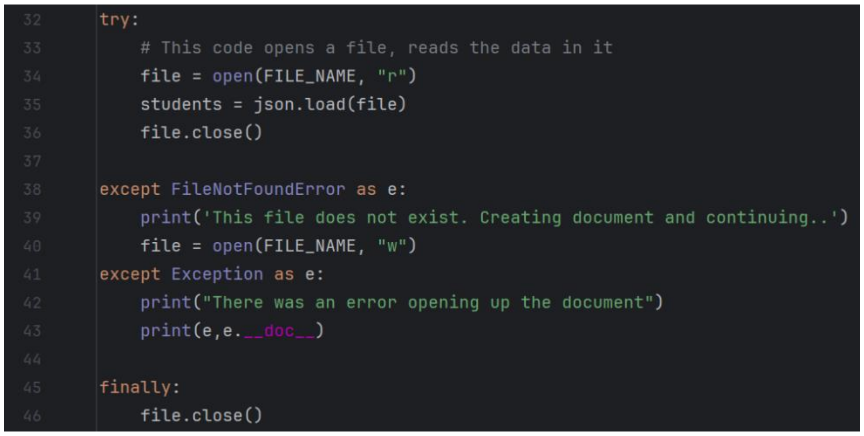
The next part of the script creates string variables that are assigned to an empty string. This is done by setting the variable equal to "". The variables that are defined will be used to store the student's first name, the student's last name, the course name, the data string consisting of the user's inputs, and the menu choice the user chooses. The next variable is called "file" and is set to data type "None". This variable will be used later in the code to open the data file (.json) and read and write the data to it. The final two variables are "student_data" and "students". The "student_data variable is defined as an empty dictionary, while the "students" variable is defined as an empty list. Lists are mutable, which means they can have elements added, modified, or removed. Dictionaries are rows of data that have headers called "Keys". The data in the rows can be called using the "Keys". Figure 2 shows the variables that are created in this portion of the script.



```python
# This section assigns variables to an empty string
student_first_name: str = ""
student_last_name: str = ""
course_name: str = ""
menu_choice: str = ""
# This sets the variable to data type None
file = None
# This section assigns variables to an empty list
student_data: dict[str,str] = {}
students: list = []
```

Figure 2: Code Showing Variables and Their Data Types

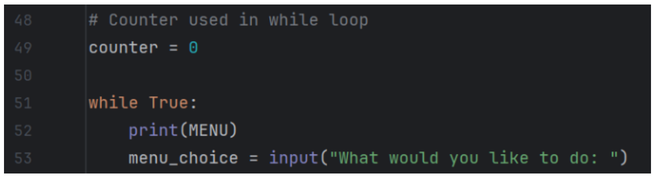
After defining the variables, this script starts by opening the "Enrollments.json" file and reading the data. This is specified by using "r" in the open() function. This is an error prone action, so error handling is implemented. This is done by using "try:" to tell the program to soft try a line a code. If there are no errors, the program continues through the "try:" section and then skips down to the "finally:" section where it closed the file. Currently, two common errors are protected for using the "except:" block. If the opening of the file errors out due to not finding the file, the program will enter the first "except:" block. In this block, a statement is printed warning that the file was not found, and then the file is created. If there is an exception error while opening the document, the code will skip to the second "except:" block. In this block, the error will be printed. After going through the "except:" block(s), the "finally:" block will run, which closes the file. The data from the "Enrollments.json file is stored in the "students" list using the json.load() function. Figure 3 shows the code the opens and reads in the data from the "Enrollments.json" file with error handling.



```python
try:
    # This code opens a file, reads the data in it
    file = open(FILE_NAME, "r")
    students = json.load(file)
    file.close()

except FileNotFoundError as e:
    print('This file does not exist. Creating document and continuing..')
    file = open(FILE_NAME, "w")
except Exception as e:
    print("There was an error opening up the document")
    print(e,e.__doc__)

finally:
    file.close()
```

Figure 3: Code Showing Enrollments.json File Being Opened with Exception Handling

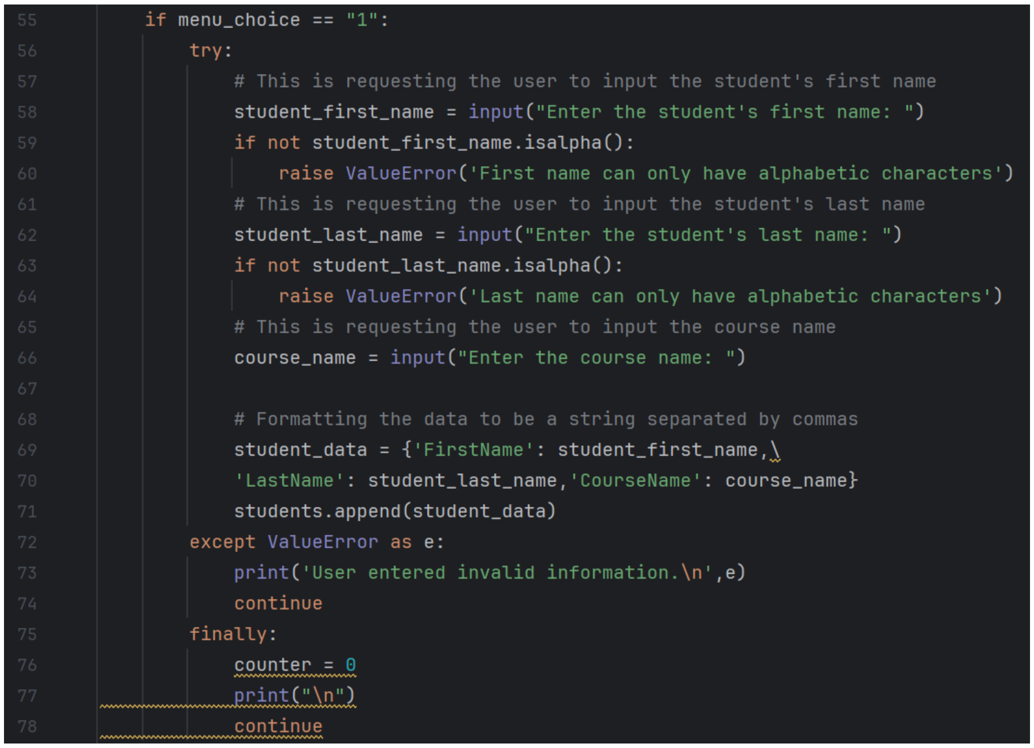
This script uses a while loop to keep the program running until the user ends it. To do this, a while True: statement is used. This will keep the program running indefinitely, until the code reaches a “break” command. To put a portion of code in a while loop, the coder needs to indent the line(s) of script. Lines of code below the while loop that are not indented will be run after the while loop is ended. After the start of the loop, the menu is printed to the console, and the user is asked to input the option in the menu they would like to choose. The print() and input() functions will be printed to the console on every iteration of the loop. Figure 4 shows the start of the while loop. Figure 4 also shows a counter that is used in the loop, which will be touched on later.



```python
# Counter used in while loop
counter = 0

while True:
    print(MENU)
    menu_choice = input("What would you like to do: ")
```

Figure 4: Code Showing the Start of the While Loop

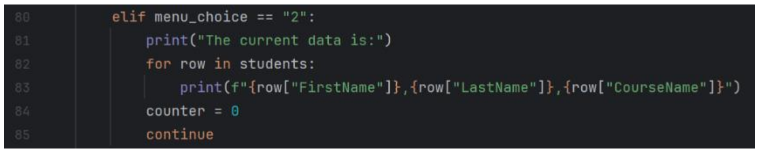
After the user input is requested, a multi-conditional is used to sort the response and complete the proper next steps. The conditional options are shown in Figures 5, 6, 7, 8, and 9. Figure 5 shows the code that is implemented if the user inputs “1”. This code prompts the user to input the student’s first name, the student’s last name, and the course name. This data is stored in the “student_data” dictionary, then appended to the “students” list. The ”student_data” dictionary has ‘FirstName’, ‘LastName’, and ‘CourseName’ as the “Keys” for the data. The input() functions requesting the student’s first and last names have error handling built in. The code has an if not statement built in checking that the user’s input was only alphabetic. If there are special characters or numbers, the code raises a value error and restarts the loop. A line of code is printed telling the user that they must only use alphabetic characters. At the end of the conditional, the counter is set to 0. This is a change from the previous iteration of the code, and will be discussed more in depth later. A print() function is used to add a new line to the console for visual appeal. The "if" conditional is exited in line 78 with the use of a continue command which ends the conditional, and in this case restarts the loop due to no additional code in the loop outside of the multi-conditional.



```
55      if menu_choice == "1":
56          try:
57              # This is requesting the user to input the student's first name
58              student_first_name = input("Enter the student's first name: ")
59              if not student_first_name.isalpha():
60                  raise ValueError('First name can only have alphabetic characters')
61              # This is requesting the user to input the student's last name
62              student_last_name = input("Enter the student's last name: ")
63              if not student_last_name.isalpha():
64                  raise ValueError('Last name can only have alphabetic characters')
65              # This is requesting the user to input the course name
66              course_name = input("Enter the course name: ")
67
68              # Formatting the data to be a string separated by commas
69              student_data = {'FirstName': student_first_name,\
70              'LastName': student_last_name,'CourseName': course_name}
71              students.append(student_data)
72          except ValueError as e:
73              print('User entered invalid information.\n',e)
74              continue
75          finally:
76              counter = 0
77              print("\n")
78              continue
```

Figure 5: Code Showing the "if" Portion of the Multi-Conditional

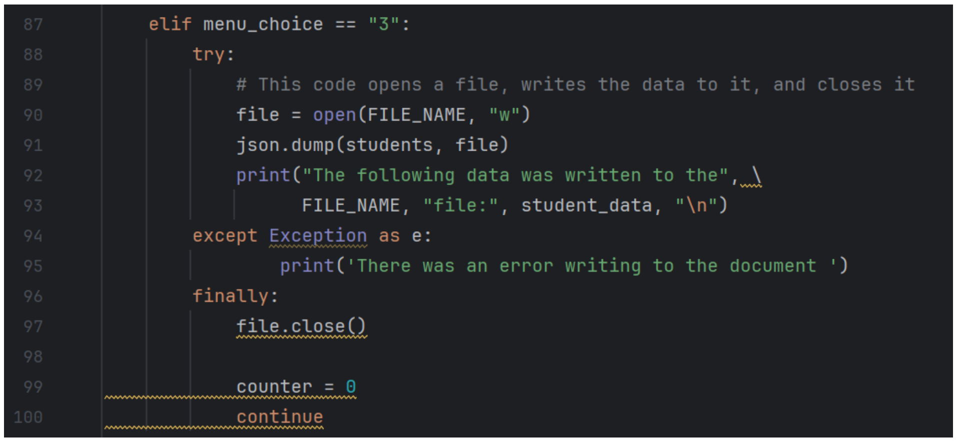
After an "if" conditional in a multi-conditional follows an "elif" conditional, which stands for "else if". If the "if" conditional of inputting a "1" is not met, the code will move onto the first "elif" conditional which is looking for the user to have input a "2". If this condition is met, the code will print the data stored during menu choice 1's code. When the user inputs a "2", the data stored in the "students" list will be displayed. This data is called out and printed using the "Keys" in the "students_data" dictionary. If no data has been input, the code will output the contents of the "Enrollments.json" file since that has been read in and added to the "students" list. If data has been input, the new data will be appended to the end of the contents of the "Enrollments.json" file. This does not guarantee that the input data has been saved to the .json file. The end of this conditional also includes the counter being reset to zero. This will be discussed later on in this document. This section of the multi-conditional is also ended with a "continue" command. Figure 6 shows the script if a "2" is input by the user.



```python
elif menu_choice == "2":
    print("The current data is:")
    for row in students:
        print(f"{row["FirstName"]},{row["LastName"]},{row["CourseName"]}")
    counter = 0
    continue
```

Figure 6: Code Showing the First "elif" Portion of the Multi-Conditional

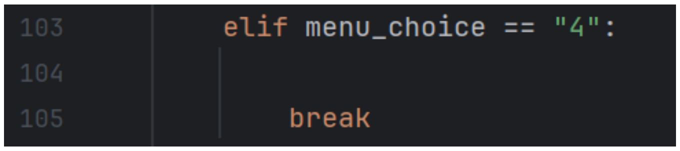
The next part of the multi-conditional is looking for the user to have input a "3". If this condition is met, the code will open the "Enrollments.json" file and print the data stored the last time "1" was input. If this is entered prior to a "1" being input, nothing will be written to the file. The open() command utilizes the append mode, indicated by the "w", which writes over the file each time. This is acceptable since the data is read in and stored in the "students" list. The data is written into the "Enrollments.json" file using the json.dump() command. Since there are errors that may appear while writing the data, error handling is used. If an exception is thrown, the user is alerted to the error occurring while writing to the data file, and then the file is closed and the loop continues. This conditional also resets the counter to zero, which is discussed below. This conditional is also ended with a "continue" command.



```python
elif menu_choice == "3":
    try:
        # This code opens a file, writes the data to it, and closes it
        file = open(FILE_NAME, "w")
        json.dump(students, file)
        print("The following data was written to the",\
              FILE_NAME, "file:", student_data, "\n")
    except Exception as e:
            print('There was an error writing to the document ')
    finally:
        file.close()

        counter = 0
        continue
```

Figure 7: Code Showing the Second "elif" Portion of the Multi-Conditional

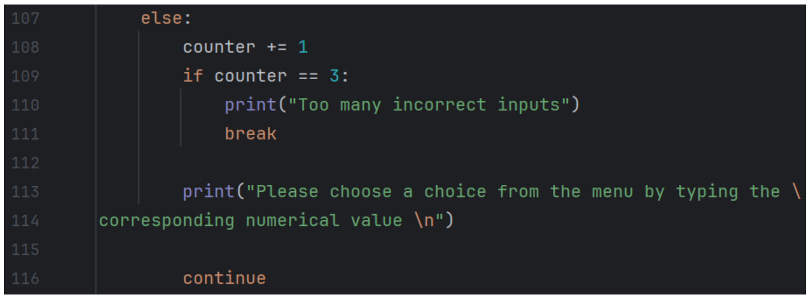
The next portion of the multi-conditional is looking for the user to have input a "4". This menu choice is meant to end the program. To do this, the code uses a "break" command. This ends the conditional, and loop, and kicks the code down to any remaining outside of the while loop.



```python
elif menu_choice == "4":

    break
```

Figure 8: Code Showing the Final "elif" Portion of the Multi-Conditional

The last option in a multi-conditional must be an “else” conditional. This condition will be met if all of the prior conditions are not met. The “else” conditional in this script is used to pick up any erroneous inputs and redirect the user to input the proper value. As mentioned above, there is a counter that was defined prior to the start of the while loop. That counter is used in this portion of the code. Every time the loop enters the “else” conditional, the counter is increased by one. Once the counter reaches a value of 3, the while loop is broken, and the user is told they had too many incorrect inputs. This is done to help keep the code from staying in the while loop indefinitely, and to help guide the user to the desired input. This feature has been updated in this assignment based on a suggestion from the grader for the last assignment. As noted above, the counter is reset to 0 after every correct conditional is entered. The new form of the code requires three consecutive errors from the user prior to ending the program. This doesn’t penalize the user if they have typos/issues when inputting a lot of data using the script.



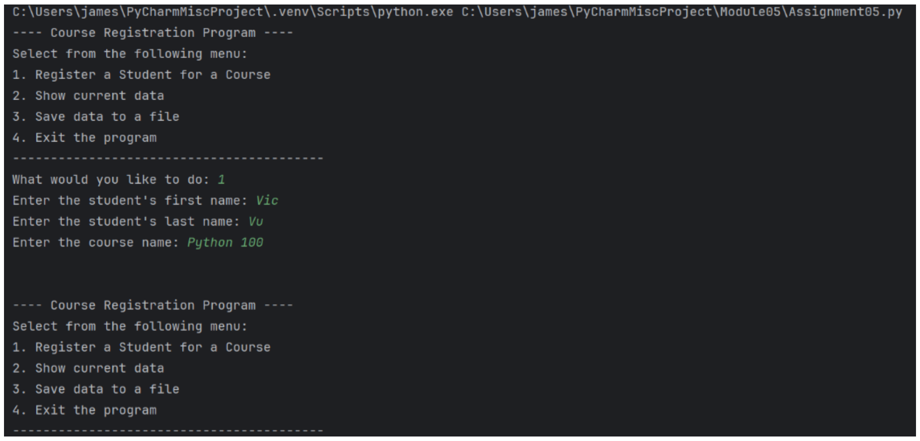
```
        else:
            counter += 1
            if counter == 3:
                print("Too many incorrect inputs")
                break

            print("Please choose a choice from the menu by typing the \
corresponding numerical value \n")

            continue
```

Figure 9: Code Showing the “else” Portion of the Multi-Conditional

The final line of code, not depicted in this document, is a print() function stating that the program has ended. This line is not indented, indicating that it will run once the while loop is broken.

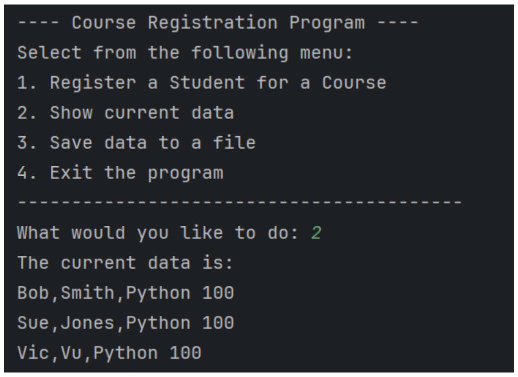
Testing the Code

After the code was written, it was run in PyCharm. It is important to note that the “Enrollments.json” file requires some data in the file prior to the running of the script. Otherwise the script will error on when reading the file. Figure 10 shows the console after the user input “1” and then was prompted to enter the student’s first name, last name, and course. It also shows that the loop returns to printing the menu. Figure 11 shows the console after the user input “2”. Figure 12 shows the console after the user input “3”. Figure 13 shows what happens when the user has an input that is not on the menu. Figure 14 shows how the program ends after a “4” is input by the user. Figure 15 shows how the program ends after the user has three incorrect inputs. Figure 16 shows the error handling if the “Enrollments.json” file does not exist. Figure 17 shows the error handling if the user does not input an alphabetic response.



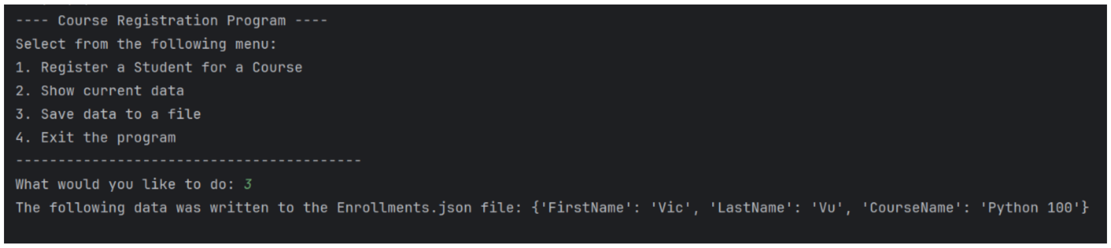
```
C:\Users\james\PyCharmMiscProject\.venv\Scripts\python.exe C:\Users\james\PyCharmMiscProject\Module05\Assignment05.py
---- Course Registration Program ----
Select from the following menu:
1. Register a Student for a Course
2. Show current data
3. Save data to a file
4. Exit the program
----------------------------------------
What would you like to do: 1
Enter the student's first name: Vic
Enter the student's last name: Vu
Enter the course name: Python 100


---- 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 10: Image of PyCharm Console After User Enters "1"



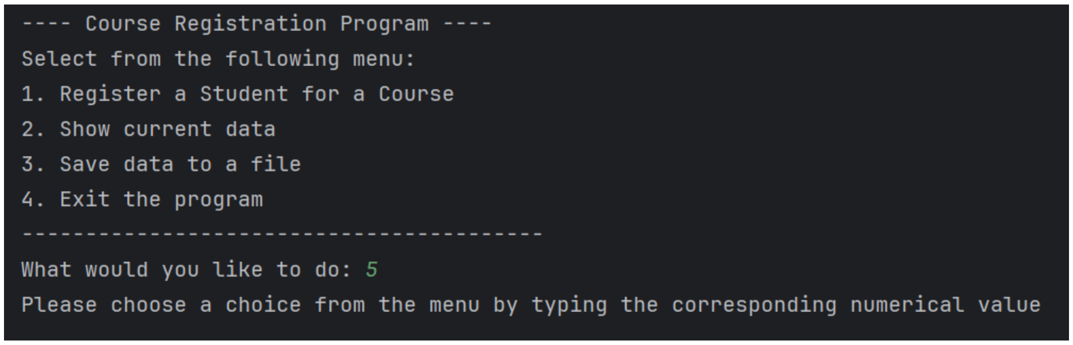
```
---- Course Registration Program ----
Select from the following menu:
1. Register a Student for a Course
2. Show current data
3. Save data to a file
4. Exit the program
-----------------------------------------
What would you like to do: 2
The current data is:
Bob,Smith,Python 100
Sue,Jones,Python 100
Vic,Vu,Python 100
```

Figure 11: Image of PyCharm Console After User Enters "2"



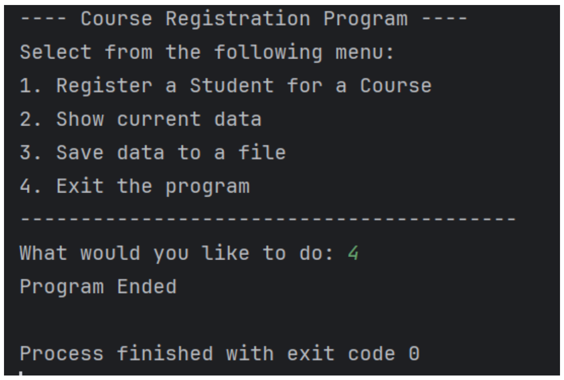
```
---- Course Registration Program ----
Select from the following menu:
1. Register a Student for a Course
2. Show current data
3. Save data to a file
4. Exit the program
----------------------------------------
What would you like to do: 3
The following data was written to the Enrollments.json file: {'FirstName': 'Vic', 'LastName': 'Vu', 'CourseName': 'Python 100'}
```

Figure 12: Image of PyCharm Console After User Enters "3"



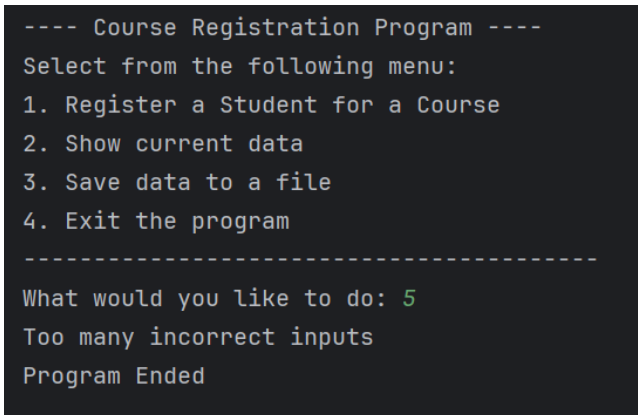
```
---- Course Registration Program ----
Select from the following menu:
1. Register a Student for a Course
2. Show current data
3. Save data to a file
4. Exit the program
-----------------------------------------
What would you like to do: 5
Please choose a choice from the menu by typing the corresponding numerical value
```

Figure 13: Image of PyCharm Console After User Enters Something Incorrect



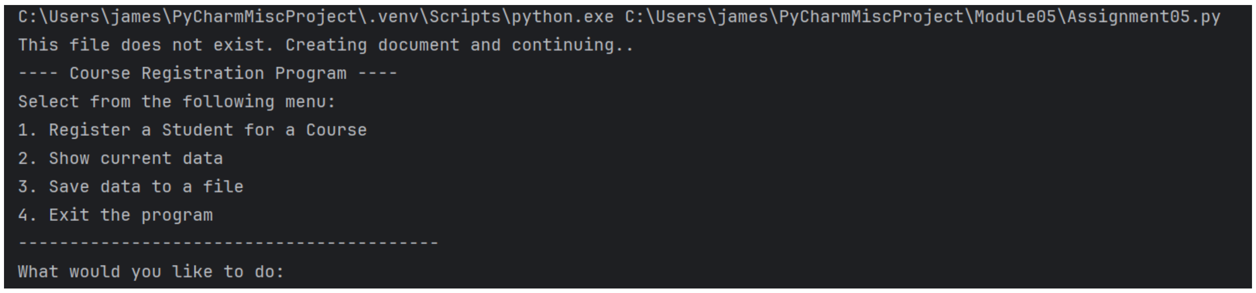
```
---- Course Registration Program ----
Select from the following menu:
1. Register a Student for a Course
2. Show current data
3. Save data to a file
4. Exit the program
-----------------------------------------
What would you like to do: 4
Program Ended

Process finished with exit code 0
```

Figure 14: Image of PyCharm Console After User Enters "4"



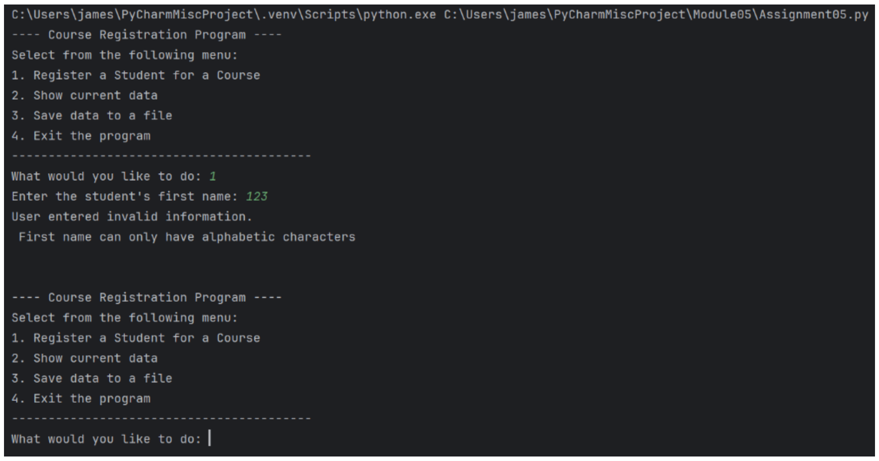
```
---- Course Registration Program ----
Select from the following menu:
1. Register a Student for a Course
2. Show current data
3. Save data to a file
4. Exit the program
-----------------------------------------
What would you like to do: 5
Too many incorrect inputs
Program Ended
```

Figure 15: Image of PyCharm Console After User Enters Three Consecutive Incorrect Inputs



```
C:\Users\james\PyCharmMiscProject\.venv\Scripts\python.exe C:\Users\james\PyCharmMiscProject\Module05\Assignment05.py
This file does not exist. Creating document and continuing..
---- 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:
```

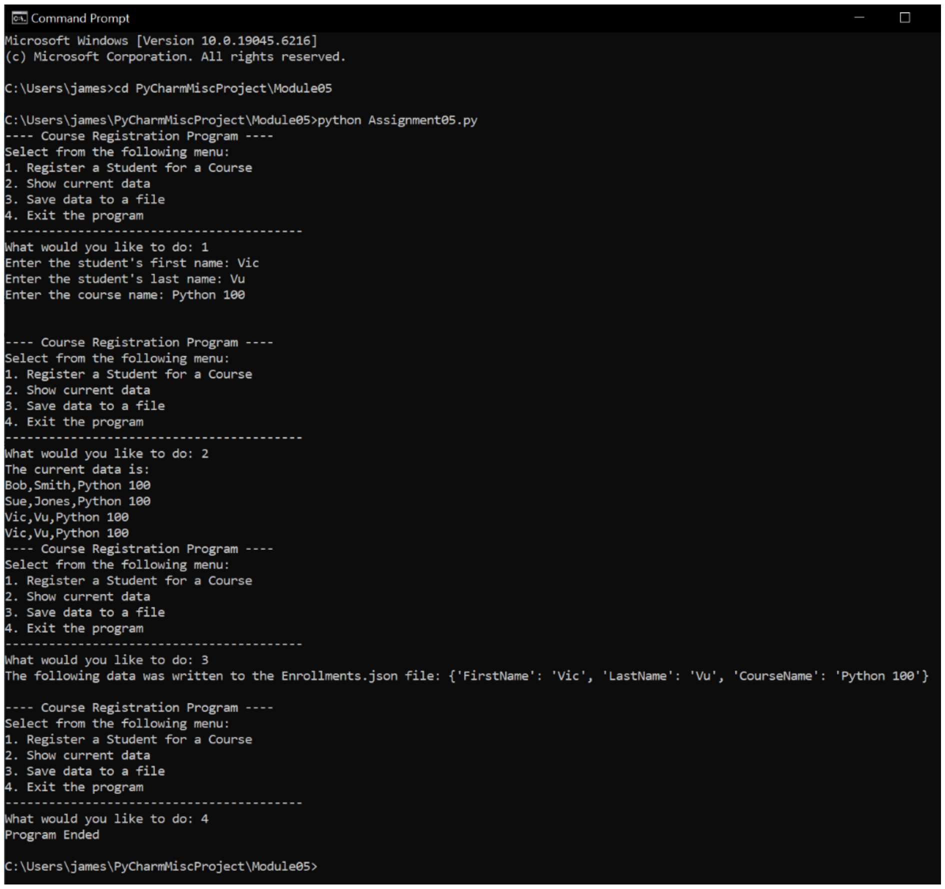
Figure 16: Image of PyCharm Console Showing Error Handling if File Does Not Exist



```
C:\Users\james\PyCharmMiscProject\.venv\Scripts\python.exe C:\Users\james\PyCharmMiscProject\Module05\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: 1
Enter the student's first name: 123
User entered invalid information.
 First name can only have alphabetic characters


---- 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: 
```

Figure 17: Image of Error Handling the Student's Name

Per the Assignment05 instructions, the code needs to run properly in both PyCharm and in the console or terminal. Testing in the terminal required changing the directory to where the Assignment05.py was saved. To do this, the function CD was input, followed by the file path to the folder that Assignment05.py was saved in. To run the python script, type "python Assignment05.py", and press "Enter" on the keyboard. Figure 18 shows the use of the CD command, as well as the output of the code.



```
Command Prompt
Microsoft Windows [Version 10.0.19045.6216]
(c) Microsoft Corporation. All rights reserved.

C:\Users\james>cd PyCharmMiscProject\Module05

C:\Users\james\PyCharmMiscProject\Module05>python 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: 1
Enter the student's first name: Vic
Enter the student's last name: Vu
Enter the course name: Python 100


---- 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
The current data is:
Bob,Smith,Python 100
Sue,Jones,Python 100
Vic,Vu,Python 100
Vic,Vu,Python 100
---- 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 written to the Enrollments.json file: {'FirstName': 'Vic', 'LastName': 'Vu', 'CourseName': 'Python 100'}

---- 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

C:\Users\james\PyCharmMiscProject\Module05>
```

Figure 18: Changing Directory and Running Code Using the Command Prompt

Something to note is that this code creates a file, and therefore that file path can be defined. If the coder wishes, they could define the path to a specific directory. In this case, a directory was not specified, so the file was written into the current working directory. This is called using a relative file path. When creating files, it is important to be cognizant of where the file is being saved to.

The code is stored in a GitHub repository so it can be viewed. The repository can be found using the following link: https://github.com/pothjm/IntroToProg-Python-Mod5

Summary

In this module, the coder carried over lessons learned in the previous modules and added to them. Some of the new topics touched on in this module include reading JSON data files and writing data to them, working with lists and dictionaries, and using the structured error handling to stop the program from being killed. The script that was written for this module was successfully run in PyCharm and the terminal.