Brian Christopherson

28-MAY-2025

Python 100

Assignment06

https://github.com/cb658/IntroToProg-Python-Mod06

A program using classes and functions, with Separation of Concerns patterning

Introduction

This paper will discuss the process used to create Assignment06, which involves writing a Python program that registers new students and adds them to the student enrollment. To do this, the program reads the enrollment file and writes the user-input data back to the enrollments file in the JavaScript Object Notation (JSON) format.

The program presents a menu to the user and uses a *while* loop and conditional logic to control the flow of the program. The program uses constants, variables, dictionaries and a list to store and display the data. Classes and functions are used to organize the program, and the program is further organized in patterns based on the Separation of Concerns (SoC) design principle.

The program also uses the try-except structured error handling method.

This program was written in the PyCharm IDE. Documentation of the program's testing can be found in Appendix A; the full code is in Appendix B.

Writing the Python program

The program creation process began with a review of the acceptance criteria for the program. The acceptance criteria included using a specific name for the program file, a script header and the use of specific constants and variables. The constants were strings presenting a multi-line menu and the name of the file to be used for the read and write operations. For the multi-line menu, a triple-quoted string was used. The menu presented four choices to the user.

The acceptance criteria defined input and output options based on the user choices from the menu:

- The first choice prompts the user via an input() function to enter the student's first and last names, and the name of the course for registration. This information was to be written to the respective variable and added to the **students** list of lists.

- The second menu choice uses the print() function to display the collected data in the **students** variable back to the user in a comma-separated string for each row in the variable.
- The third menu choice opens the Enrollments.json file in write mode using the open() function, writing the data to the file, using the *json.dump()* function, then closing the file to allow future input. The program also uses the students variable to display what was written to the file.
- The fourth menu choice exits the program.

The acceptance criteria required utilizing specific classes and functions. Two classes were to be included, with the descriptive document strings. The classes are:

- FileProcessor
- IO

The acceptance criteria also required specific functions, also with descriptive document strings, calls to function handling error message, and the use of the @staticmethod decorator. The @staticmethod is called because the functions are not trying to change the class or modify objects. The function names and parameters are listed below:

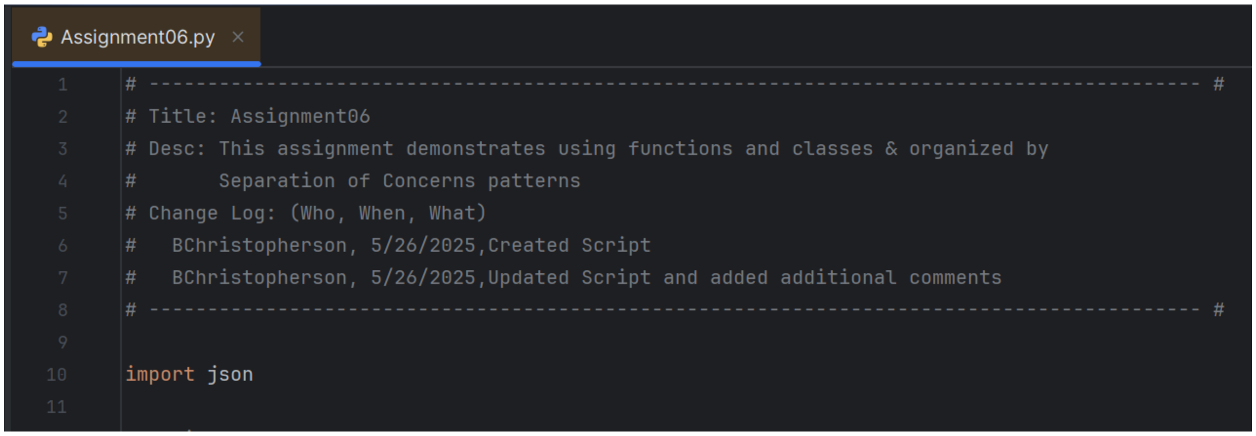
- output_error_messages(message: str, error: Exception = None)
- output_menu(menu: str)
- input_menu_choice()
- output_student_courses(student_data: list)
- input_student_data(student_data: list)
- read_data_from_file(file_name: str, student_data: list):
- write_data_to_file(file_name: str, student_data: list):

The code contained a while statement to loop through the various input choices. The looping allows for another of the acceptance criteria, which was to allow the user to enter multiple student registrations, if desired, and store all those registrations.

The acceptance criteria defined some structured error handling at various points within the program execution: when the file is read in, the user's input of the first name, the user's input of the last name, and lastly when the dictionary rows are written to the file. The last requirements of the acceptance criteria required specific tests of the program input and output functions, as well as confirming the program runs correctly in both PyCharm and the console.

Creating the script file

Creation of the program began by opening the assignment starter file in PyCharm. That file was then saved as Assignment06.py and the change log section of the script header was updated. Figure 1 shows the script header used.



Assignment06.py

```
# ------------------------------------------------------------------------------------------ #
# Title: Assignment06
# Desc: This assignment demonstrates using functions and classes & organized by
#       Separation of Concerns patterns
# Change Log: (Who, When, What)
#   BChristopherson, 5/26/2025,Created Script
#   BChristopherson, 5/26/2025,Updated Script and added additional comments
# ------------------------------------------------------------------------------------------ #

import json

```

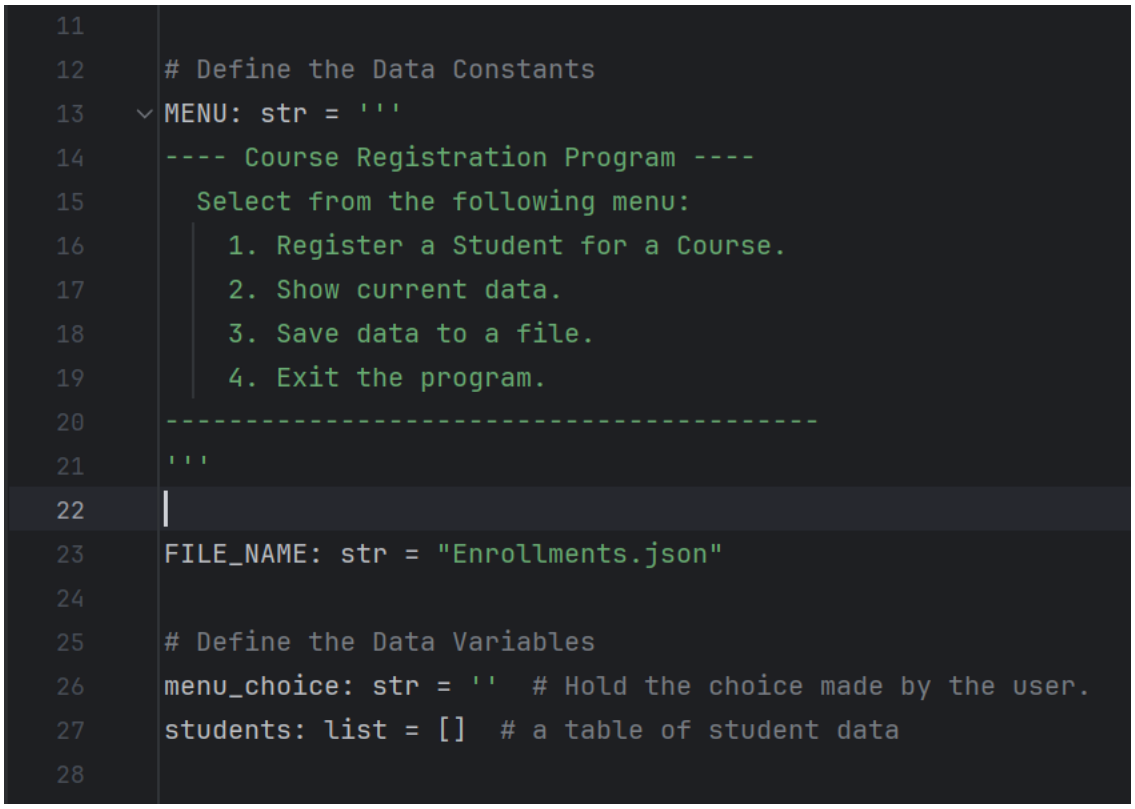
Figure 1 - Script Header

The Program Constants and Variables

The assignment starter file contained simple pseudo-code to guide the creation of the code body. This pseudo-code was leveraged to define the constants and variables, the input and output sections, and the flow of the program. Type hints are used to identify the data types used in the constants and variables.

The constants in the program are the previously mentioned user menu (a string encapsulated in triple quotation marks), and the file name for processing the data. This is also a string with the value of “Enrollments.json”. Both consonant names are written in uppercase, to comply with Python coding standards.

The data types for the variables used in the program are a string and a list for storing the accumulated data on the students. The string is named **menu_choice** for storing the menu choices input by the user; the list is named **students.** These variables are initially defined as empty; this involves using double quotation marks (“”) with no space between them for the string and empty brackets (**[]**) for the lists. The figure below shows the constants and the variable definitions.



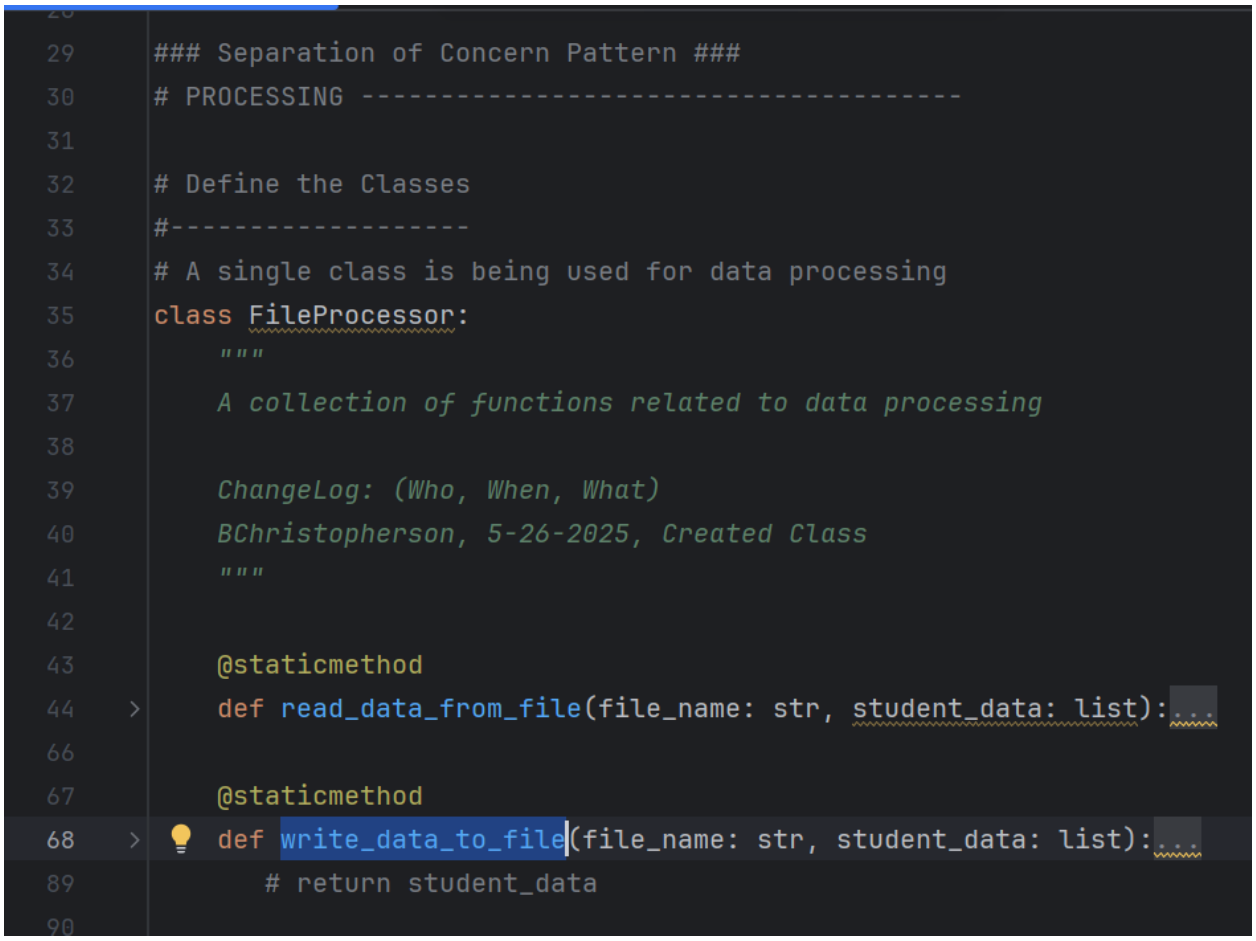
```
# Define the Data Constants
MENU: str = '''
---- Course Registration Program ----
  Select from the following menu:
    1. Register a Student for a Course.
    2. Show current data.
    3. Save data to a file.
    4. Exit the program.
-----------------------------------------
'''

FILE_NAME: str = "Enrollments.json"

# Define the Data Variables
menu_choice: str = ''  # Hold the choice made by the user.
students: list = []  # a table of student data
```

Figure 2- The program's Constants and Variables

Defining the classes and functions –

Processing layer

As defined in the acceptance criteria, and following the Separation of Concerns design principle, the first class is in the "Processing" layer and is named **FileProcessor**. This class contains two functions, named **read_data_from_file** and **write_data_to_file**. The class and the functions have descriptive document strings. The FileProcessor document string and a collapsed view of the two functions are shown in the Figure below.



```
### Separation of Concern Pattern ###
# PROCESSING ------------------------------------

# Define the Classes
#------------------
# A single class is being used for data processing
class FileProcessor:
    """
    A collection of functions related to data processing

    ChangeLog: (Who, When, What)
    BChristopherson, 5-26-2025, Created Class
    """

    @staticmethod
    def read_data_from_file(file_name: str, student_data: list):...

    @staticmethod
    def write_data_to_file(file_name: str, student_data: list):...
        # return student_data
```

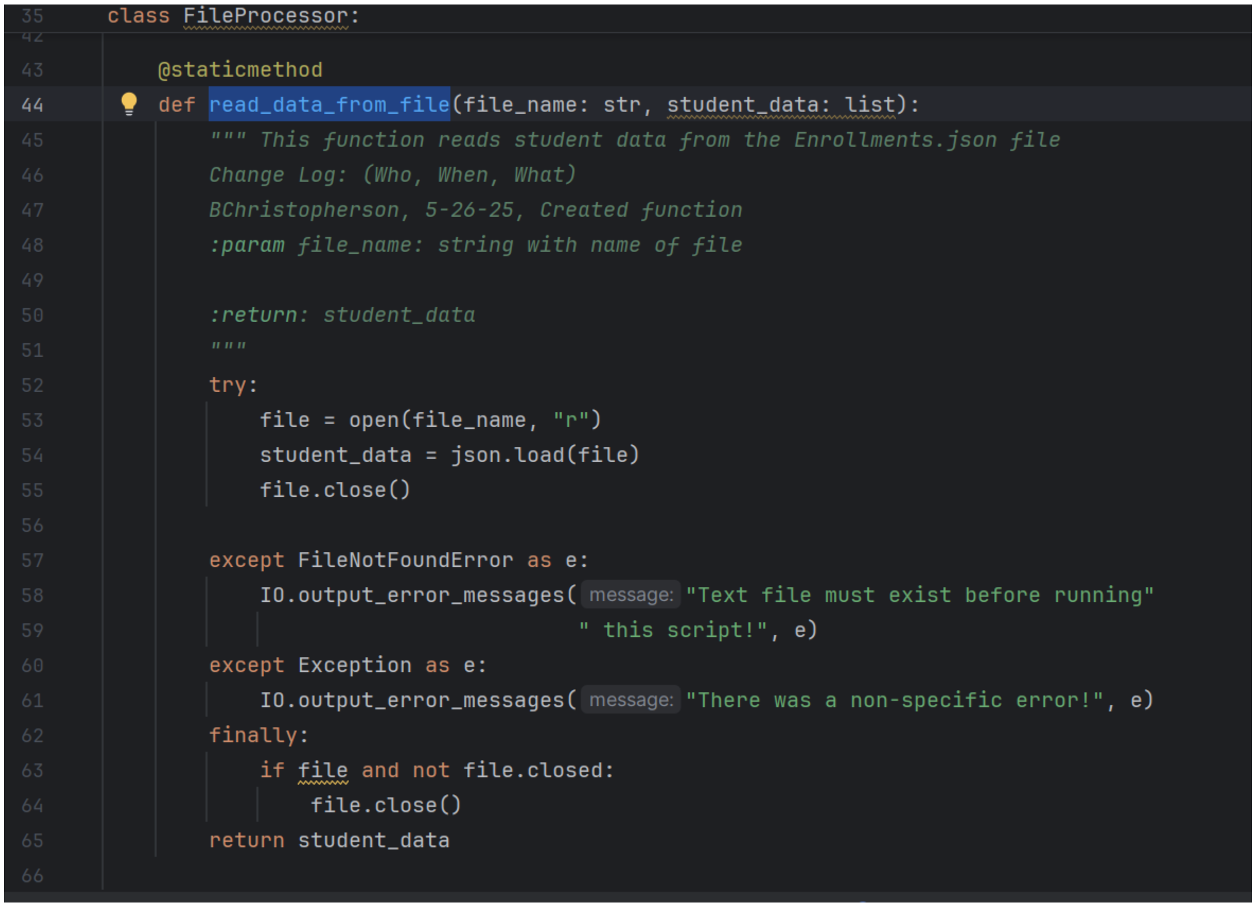
Figure 3 - The FileProcessor class

Reading the json data file using the read_data_from_file function

As defined in the **FILE_NAME** variable, a file named Enrollments.json, is read in using the **read_data_from_file** function. A *try:* block is begun to trap possible errors. As suggested in the acceptance criteria, a list in json format was pre-populated into the file. Within the *try:* block, the code attempts to open the file using the *open()* function with the "r" parameter, accessing the information in read-only mode. If the file is successfully opened, the *json.load() function* is used to read the file and convert it to the students list, then the file is closed using *file.close()*.

The error handling in this section of code begins with an exception block for the *FileNotFoundError* exception. If the file is not found, the **output_error_messages** function in the IO class is called and outputs an error message advising the user that the file needs to exist before running the script. A general exception also calls the **output_error_messages** function with the message "There was a non-specific error!".

At the end of this *Try-Except* block is the *finally:* block. The *if* statement in this block tests whether the file is not closed and if so, closes the file. The Figure below shows the code for the **read_data_from_file** function.

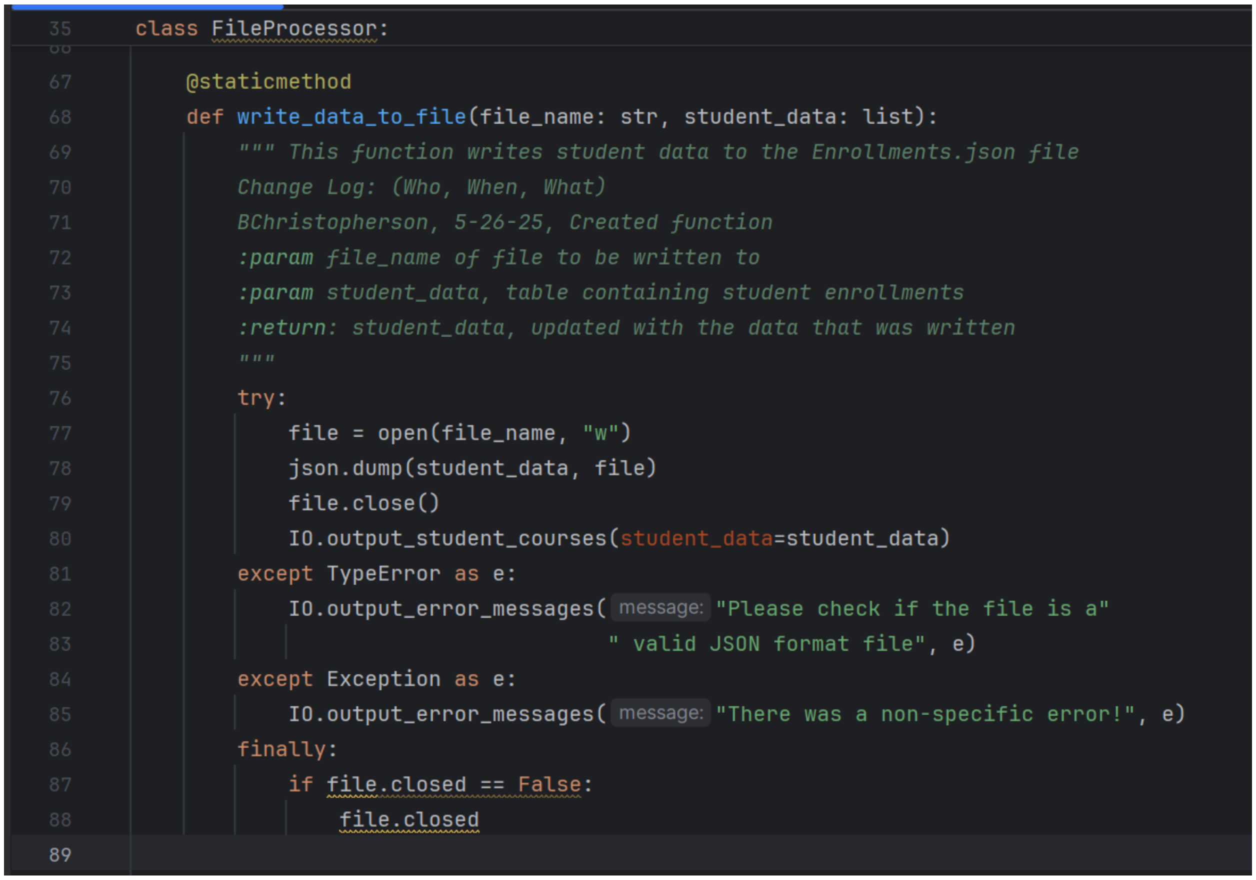


```
class FileProcessor:

    @staticmethod
    def read_data_from_file(file_name: str, student_data: list):
        """ This function reads student data from the Enrollments.json file
        Change Log: (Who, When, What)
        BChristopherson, 5-26-25, Created function
        :param file_name: string with name of file

        :return: student_data
        """
        try:
            file = open(file_name, "r")
            student_data = json.load(file)
            file.close()

        except FileNotFoundError as e:
            IO.output_error_messages( message: "Text file must exist before running"
                                     " this script!", e)
        except Exception as e:
            IO.output_error_messages( message: "There was a non-specific error!", e)
        finally:
            if file and not file.closed:
                file.close()
        return student_data

```

Figure 4 - "read_data_from_file" function

Writing to file, using the write_data_to_file function

The second function in the **FileProcessor** class is named **write_data_to_file**. This function uses the file_name variable to open the Enrollments.json file; it is opened in write mode using the “w” parameter. Using *json.dump()* the student_data list is written to file and then closed. The IO class function **output_student_courses** is included for outputting the student enrollment data. The code above is encased within a *Try-Except* block for capturing any errors. The *TypeError* exception is called in case of issues with the JSON formatting. A general exception, as seen in the **read_data_from_file** function, closes out the error handling for the function. Lastly, the *finally:* clause is used to close out the Try-Except: block and is used to close the file in case it was not closed previously. The code for the read_data_from_file function is shown in the Figure below.



```
class FileProcessor:

    @staticmethod
    def write_data_to_file(file_name: str, student_data: list):
        """ This function writes student data to the Enrollments.json file
        Change Log: (Who, When, What)
        BChristopherson, 5-26-25, Created function
        :param file_name of file to be written to
        :param student_data, table containing student enrollments
        :return: student_data, updated with the data that was written
        """
        try:
            file = open(file_name, "w")
            json.dump(student_data, file)
            file.close()
            IO.output_student_courses(student_data=student_data)
        except TypeError as e:
            IO.output_error_messages( message: "Please check if the file is a"
                                      " valid JSON format file", e)
        except Exception as e:
            IO.output_error_messages( message: "There was a non-specific error!", e)
        finally:
            if file.closed == False:
                file.closed
```

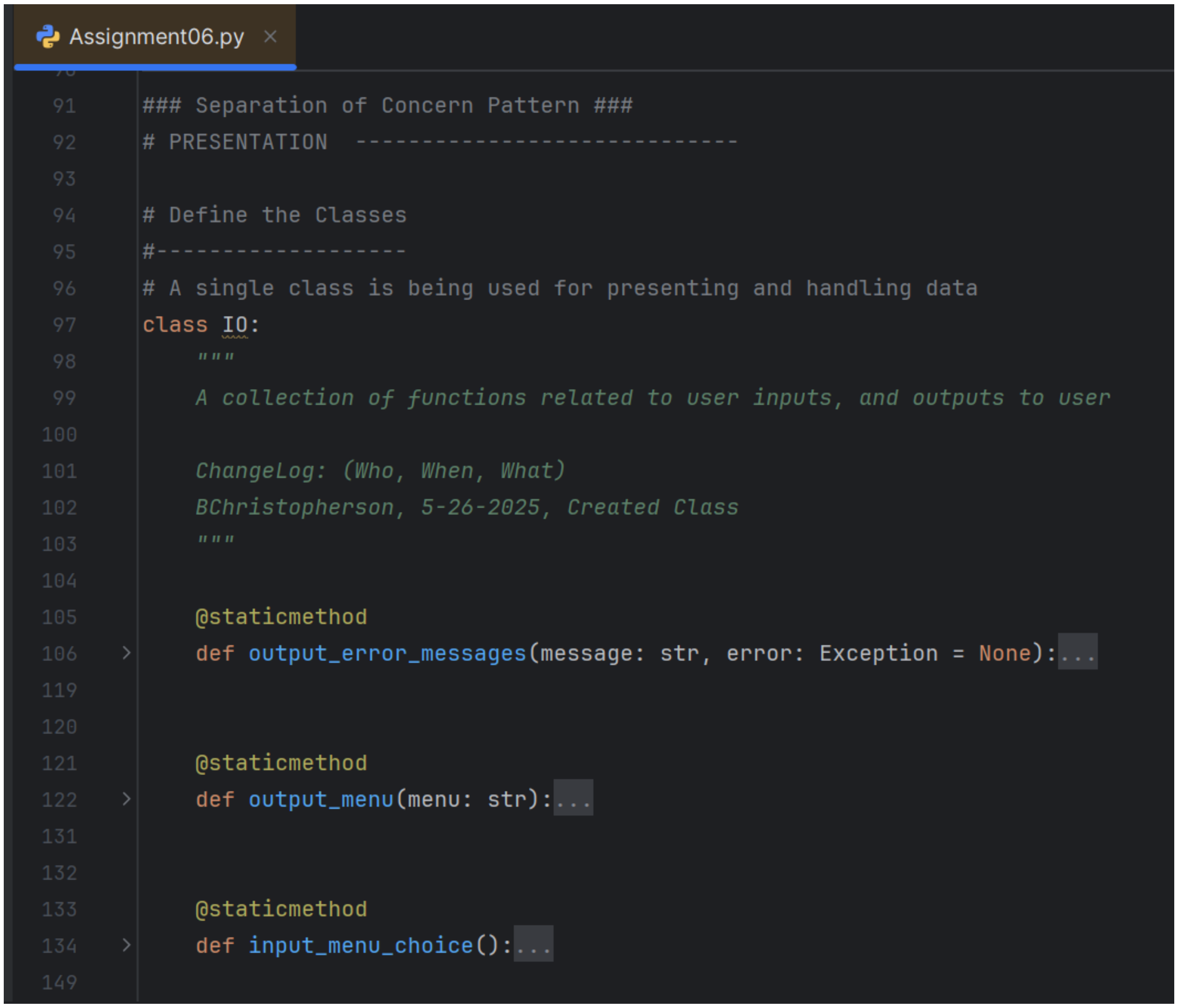
Figure 5 - "write_data_to_file" function

The IO class (in the "*Presentation*" layer)

Following the Separation of Concerns principle, a pattern is defined for the *"Presentation"* layer. This is the layer that defines the User Interface functions such as user prompts, accepting user input and outputting information to the user. Per the acceptance criteria, the class is named "IO" and contains a descriptive document string, along with the functions below:

- output_error_messages
- output_menu
- input_menu_choice
- output_student_courses
- input_student_data

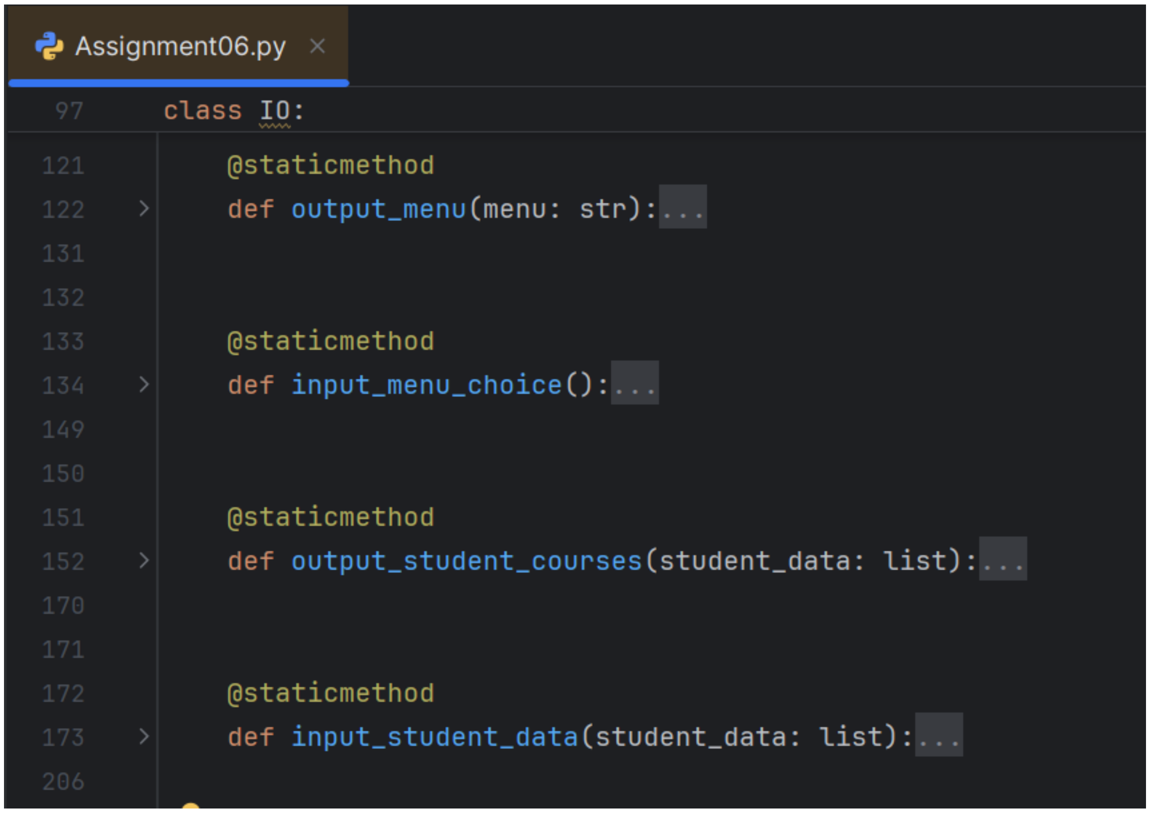
The document string for the class and a collapsed view of the functions is shown in the two figures below.



Assignment06.py

```
### Separation of Concern Pattern ###
# PRESENTATION  ----------------------------

# Define the Classes
#------------------
# A single class is being used for presenting and handling data
class IO:
    """
    A collection of functions related to user inputs, and outputs to user

    ChangeLog: (Who, When, What)
    BChristopherson, 5-26-2025, Created Class
    """

    @staticmethod
    def output_error_messages(message: str, error: Exception = None):...


    @staticmethod
    def output_menu(menu: str):...


    @staticmethod
    def input_menu_choice():...
```

Figure 6 - "IO" class, part 1



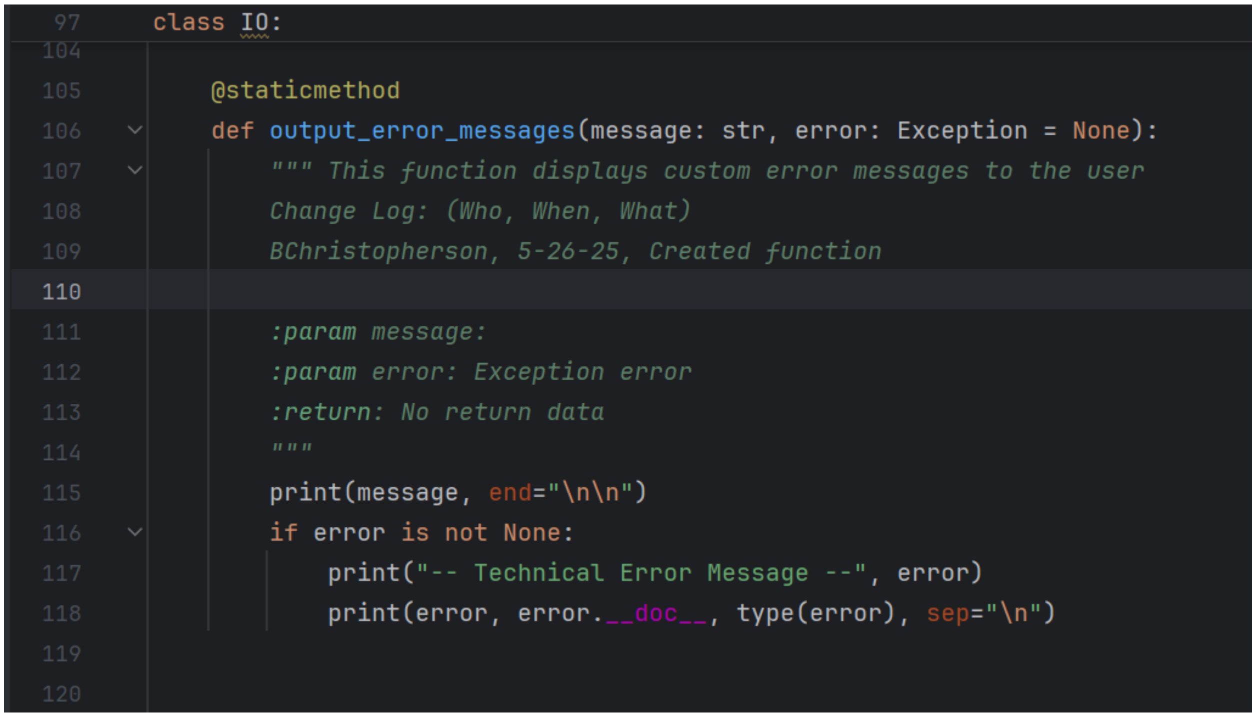
Assignment06.py

```
class IO:
    @staticmethod
    def output_menu(menu: str):...


    @staticmethod
    def input_menu_choice():...


    @staticmethod
    def output_student_courses(student_data: list):...


    @staticmethod
    def input_student_data(student_data: list):...
```

Figure 7 - "IO" class, part 2

The output_error_messages function

This function defines the format of the error messages displayed to the user for general errors, using the *__doc__* syntax to display the document string and error type for the error. The code for this function is seen in the figure below.

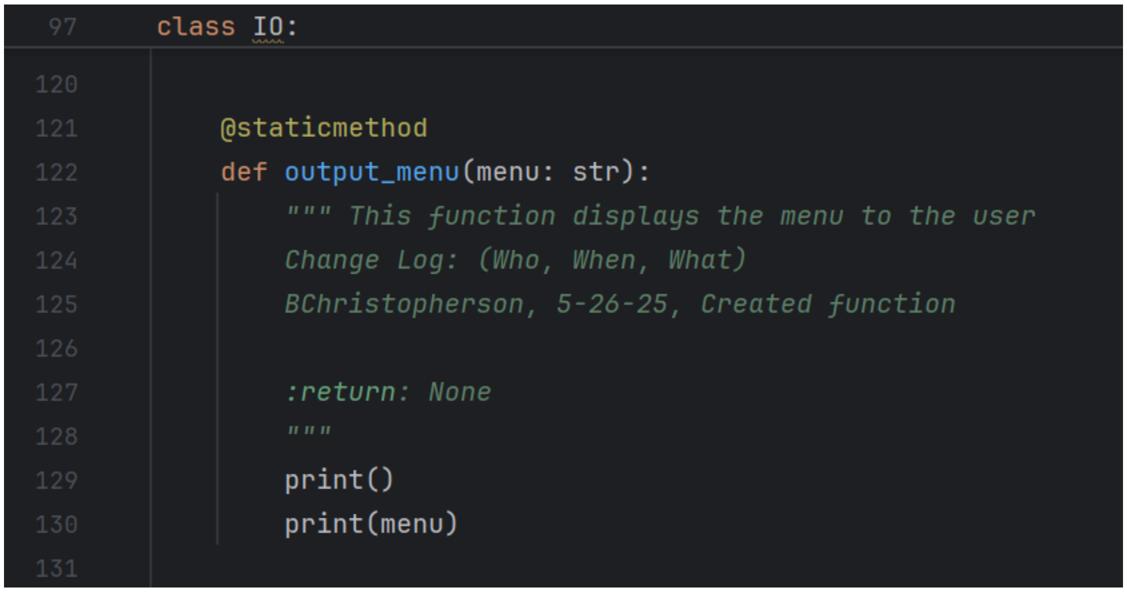


```
class IO:

    @staticmethod
    def output_error_messages(message: str, error: Exception = None):
        """ This function displays custom error messages to the user
        Change Log: (Who, When, What)
        BChristopherson, 5-26-25, Created function

        :param message:
        :param error: Exception error
        :return: No return data
        """
        print(message, end="\n\n")
        if error is not None:
            print("-- Technical Error Message --", error)
            print(error, error.__doc__, type(error), sep="\n")

```

Figure 8 - The "output_error_messages" function

The output_menu function

This is a simple function that displays the content of the **MENU** constant, which is set as an argument to the menu parameter later in the program. The code for this function is shown in the figure below.

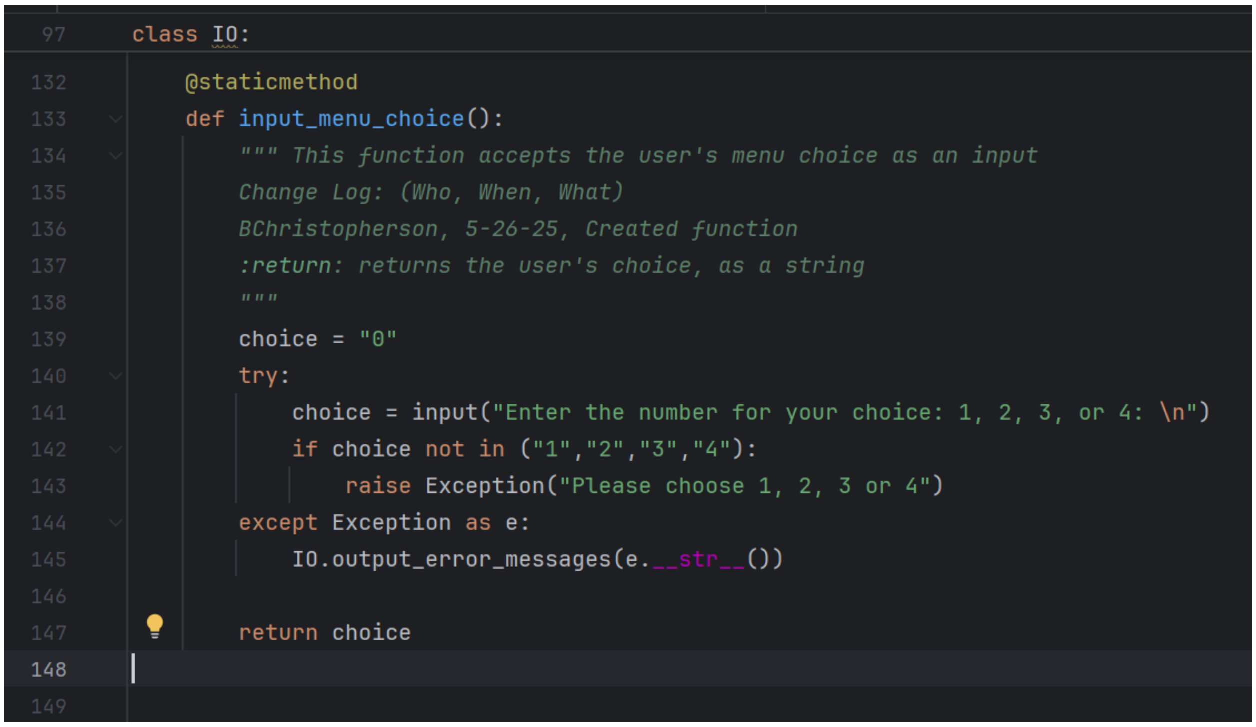


```
class IO:

    @staticmethod
    def output_menu(menu: str):
        """ This function displays the menu to the user
        Change Log: (Who, When, What)
        BChristopherson, 5-26-25, Created function

        :return: None
        """
        print()
        print(menu)

```

Figure 9 - The "output_menu" function

The input_menu_choice function

This function defines the programs assignment of user inputs to the **choice** variable. This function includes error handling to allow for the possibility of the user entering a choice other than the defined options, as well as for a general exception. Both call the output_error_messages function referenced earlier. The choice value is returned using a return statement. The code for the input_menu_choice function can be seen in the figure below.

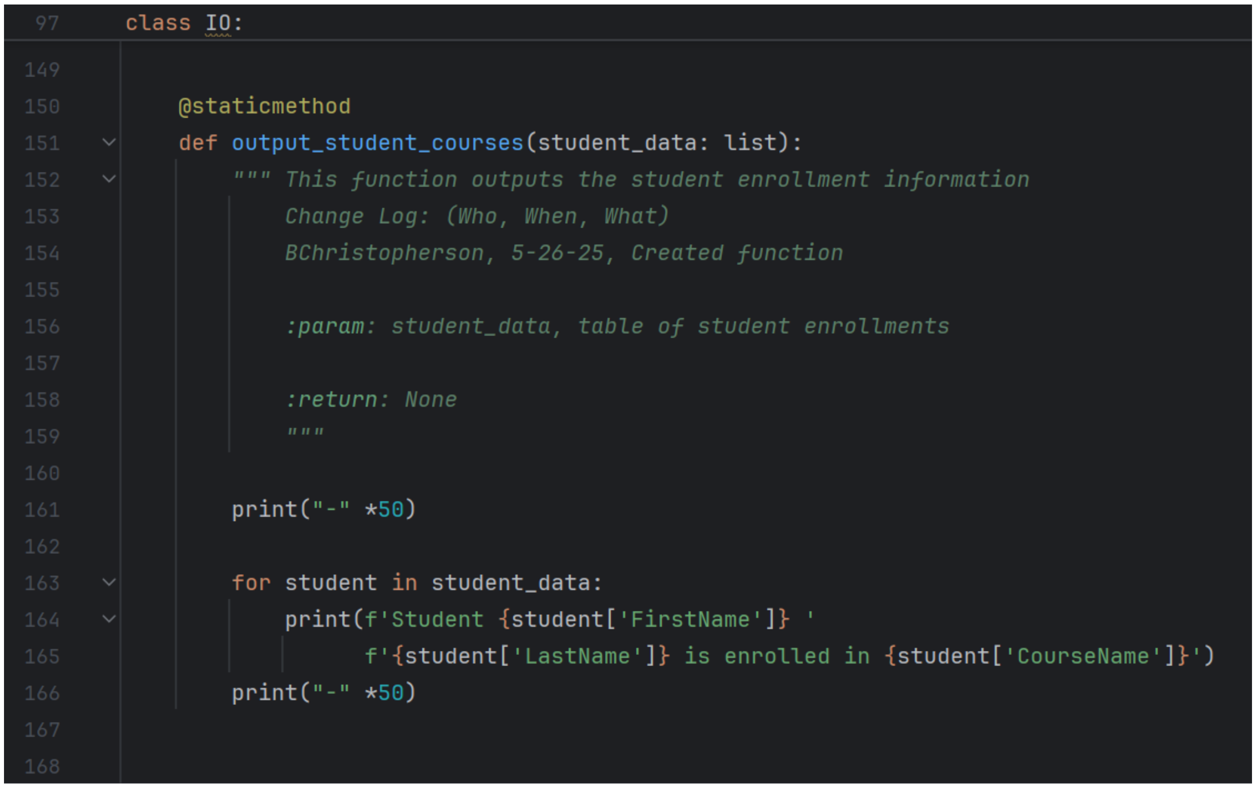


```python
class IO:

    @staticmethod
    def input_menu_choice():
        """ This function accepts the user's menu choice as an input
        Change Log: (Who, When, What)
        BChristopherson, 5-26-25, Created function
        :return: returns the user's choice, as a string
        """
        choice = "0"
        try:
            choice = input("Enter the number for your choice: 1, 2, 3, or 4: \n")
            if choice not in ("1","2","3","4"):
                raise Exception("Please choose 1, 2, 3 or 4")
        except Exception as e:
            IO.output_error_messages(e.__str__())

        return choice
```

Figure 10 – The "input_menu_choice" function

The output_student_courses function

This function outputs the student enrollment information for each student in the **students** list, using the *print()* function to output a formatted string. The code for this function is shown in the figure below.

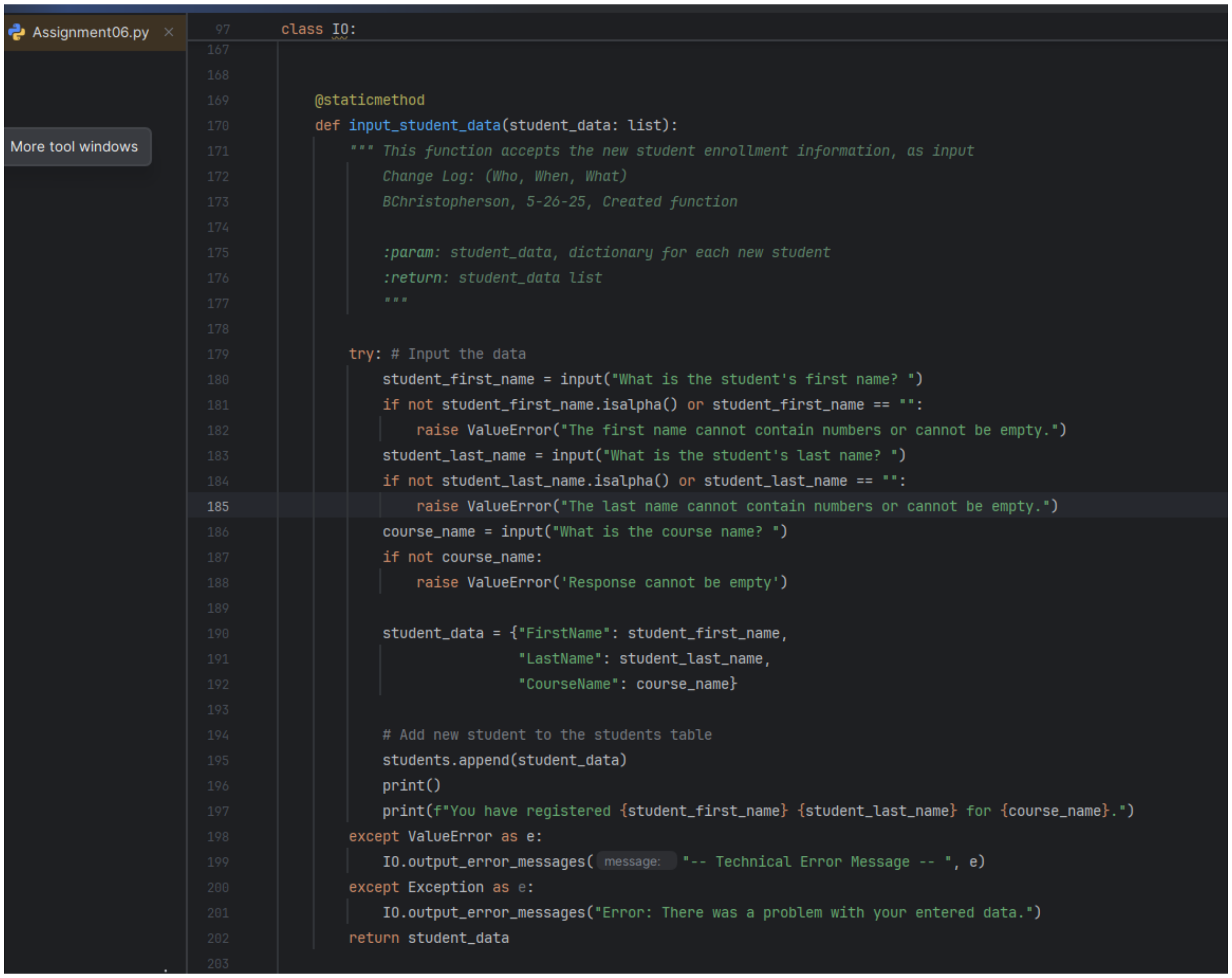


```python
class IO:

    @staticmethod
    def output_student_courses(student_data: list):
        """ This function outputs the student enrollment information
            Change Log: (Who, When, What)
            BChristopherson, 5-26-25, Created function

            :param: student_data, table of student enrollments

            :return: None
            """

        print("-" *50)

        for student in student_data:
            print(f'Student {student['FirstName']} '
                  f'{student['LastName']} is enrolled in {student['CourseName']}')
        print("-" *50)
```

Figure 11 - The "output_student_courses" function

The input_student_data function

This function begins with a Try-Except block for capturing the user input for the student course registration. Due to the potential for input errors, ValueError exceptions are caught if the user inputs numerals in the strings for student first name or last name, or empty values for either name or the course name. If the information is entered in alpha format, the input strings are then added to the **student_data** dictionary in json format. Next, the **student_data** dictionary is then added to the **students** list using the *append()* method. A *print()* statement confirms the student registration to the user. The Except blocks and a return statement returning the student_data list close out the function. The code for the input_student_data function is shown in the figure below.



```
class IO:


    @staticmethod
    def input_student_data(student_data: list):
        """ This function accepts the new student enrollment information, as input
            Change Log: (Who, When, What)
            BChristopherson, 5-26-25, Created function

            :param: student_data, dictionary for each new student
            :return: student_data list
            """

        try: # Input the data
            student_first_name = input("What is the student's first name? ")
            if not student_first_name.isalpha() or student_first_name == "":
                raise ValueError("The first name cannot contain numbers or cannot be empty.")
            student_last_name = input("What is the student's last name? ")
            if not student_last_name.isalpha() or student_last_name == "":
                raise ValueError("The last name cannot contain numbers or cannot be empty.")
            course_name = input("What is the course name? ")
            if not course_name:
                raise ValueError('Response cannot be empty')

            student_data = {"FirstName": student_first_name,
                            "LastName": student_last_name,
                            "CourseName": course_name}

            # Add new student to the students table
            students.append(student_data)
            print()
            print(f"You have registered {student_first_name} {student_last_name} for {course_name}.")
        except ValueError as e:
            IO.output_error_messages( message: "-- Technical Error Message -- ", e)
        except Exception as e:
            IO.output_error_messages("Error: There was a problem with your entered data.")
        return student_data
```

Figure 12 - The "input_student_data" function

Looping through the main body of the program

When the program starts, the file data is read into the **students** list of lists. As part of the acceptance criteria the program allows for the program to potentially be run an infinite number of times, unless the user chooses the option to exit the program. Thus, a *while* loop was used to control the program flow, using "*while (True):*", and conditional *if, elif* and *else* statements.

At the beginning of the *while* loop, the **output_menu** function is called from the **IO** class and displays the **MENU** constant, which asks the user to select a choice from the menu. The input from the user is assigned to the **menu_choice** variable. These input choices were evaluated as strings using if statements and the equals condition, written as "==".

If input choice “1” (Register a Student for a Course)

If the user’s choice is “1”, the **input_student_data** function is called from the **IO** class. This function adds the new student information to **students** list. the **output_menu** function is called from the **IO** class

Else if choice “2” (Show current data)

If the user’s choice is “2”, represented as ***elif menu_choice ==”2”:***, the ***output_student_courses*** function displays each registration to the user” for each student dictionary item in the **students** list. The *continue* keyword is used to continue to the next iteration.

Else if choice “3” (Save data to a file)

If the user’s choice is “3”, represented as ***elif menu_choice ==”3”:***, the **write_data_to_file** function is called from the **FileProcessor** class. This function uses the following parameters and arguments, referencing the constant and variable referenced in the declarations section:

> file_name=**FILE_NAME**
> student_data=**students**

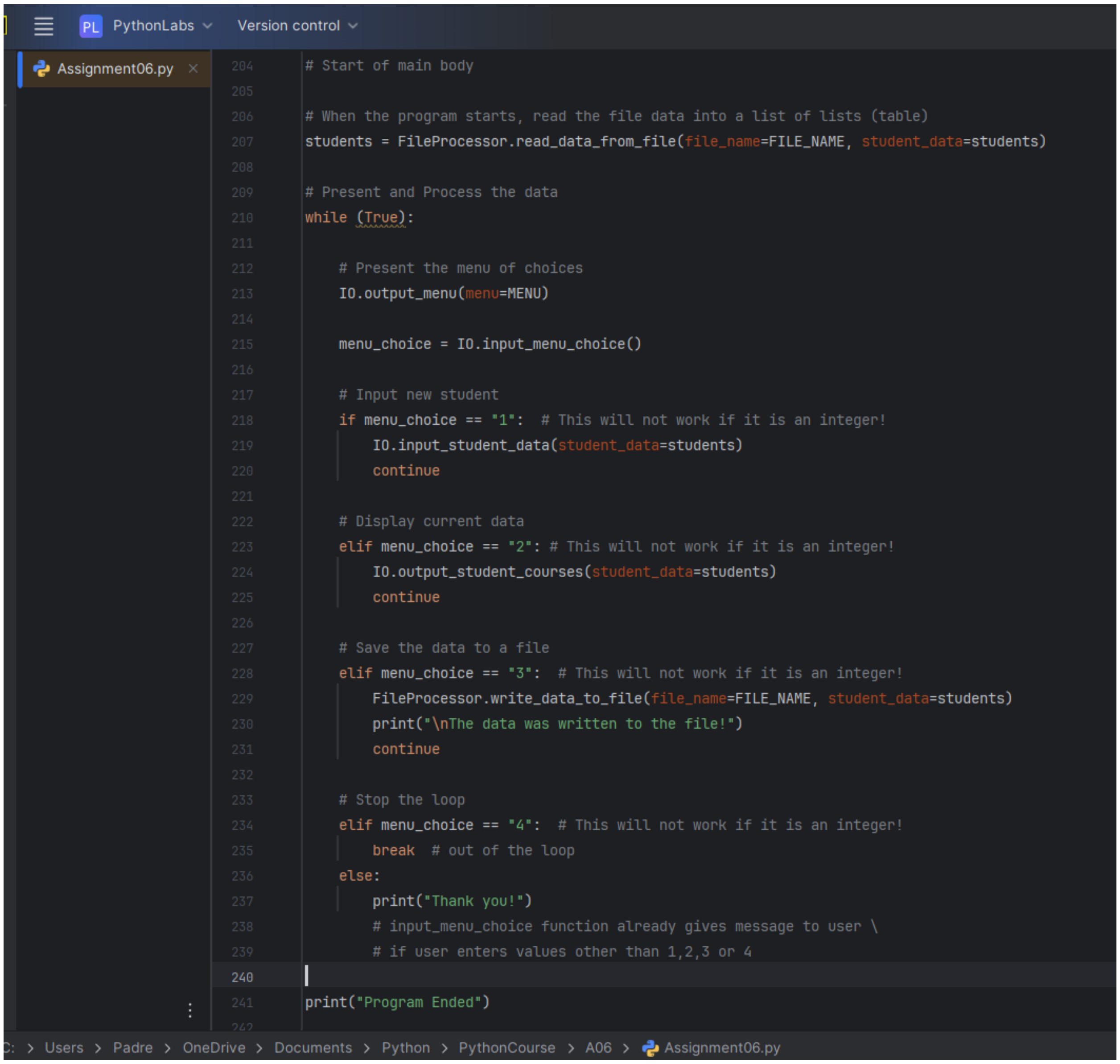
With these parameters and arguments the data in the **students** list is written to the Enrollments.json file. The *continue* keyword is used to continue to the next iteration.

Else if choice “4” (Exit the program

If the user’s choice was “4”, represented as ***elif menu_choice ==”4”:***, the loop was ended using the ***break*** keyword. A final “***else:***” is added to close out the loop and display a print statement to thank the user. In previous iterations of the program this “*else*:” was statement was used to address situations where the user enters something other than “1, 2, 3, or 4”, but this is now covered by the **input_menu_choice** function.

Once the loop has fully ended, a final *print()* statement is presented to the user, indicating the program has ended.

The code for the main body is shown in the figure below.


PythonLabs
Version control
Assignment06.py
C: > Users > Padre > OneDrive > Documents > Python > PythonCourse > A06 > Assignment06.py

```
# Start of main body

# When the program starts, read the file data into a list of lists (table)
students = FileProcessor.read_data_from_file(file_name=FILE_NAME, student_data=students)

# Present and Process the data
while (True):

    # Present the menu of choices
    IO.output_menu(menu=MENU)

    menu_choice = IO.input_menu_choice()

    # Input new student
    if menu_choice == "1":  # This will not work if it is an integer!
        IO.input_student_data(student_data=students)
        continue

    # Display current data
    elif menu_choice == "2": # This will not work if it is an integer!
        IO.output_student_courses(student_data=students)
        continue

    # Save the data to a file
    elif menu_choice == "3":  # This will not work if it is an integer!
        FileProcessor.write_data_to_file(file_name=FILE_NAME, student_data=students)
        print("\nThe data was written to the file!")
        continue

    # Stop the loop
    elif menu_choice == "4":  # This will not work if it is an integer!
        break  # out of the loop
    else:
        print("Thank you!")
        # input_menu_choice function already gives message to user \
        # if user enters values other than 1,2,3 or 4

print("Program Ended")
```

Figure 13 - The main body and end of the program

Testing the program

To test the program within PyCharm, the Run Module (F5) command was invoked. The program presented the menu, accepted the user input, then presented it and the read-in data when the ‘current data’ option was chosen. The program also saved the data to the Enrollments.json file and exited if the ‘exit program’ choice was called, thus meeting the acceptance criteria. The student error handling blocks were tested by changing the name of the Enrollments.json file and entering numerals when entering the student’s first and last names. The program was then tested by opening the Windows console and calling the Assignment06.py file. The program also ran successfully via the console. Screenshots showing the PyCharm and console tests can be found in Appendix A.

Summary

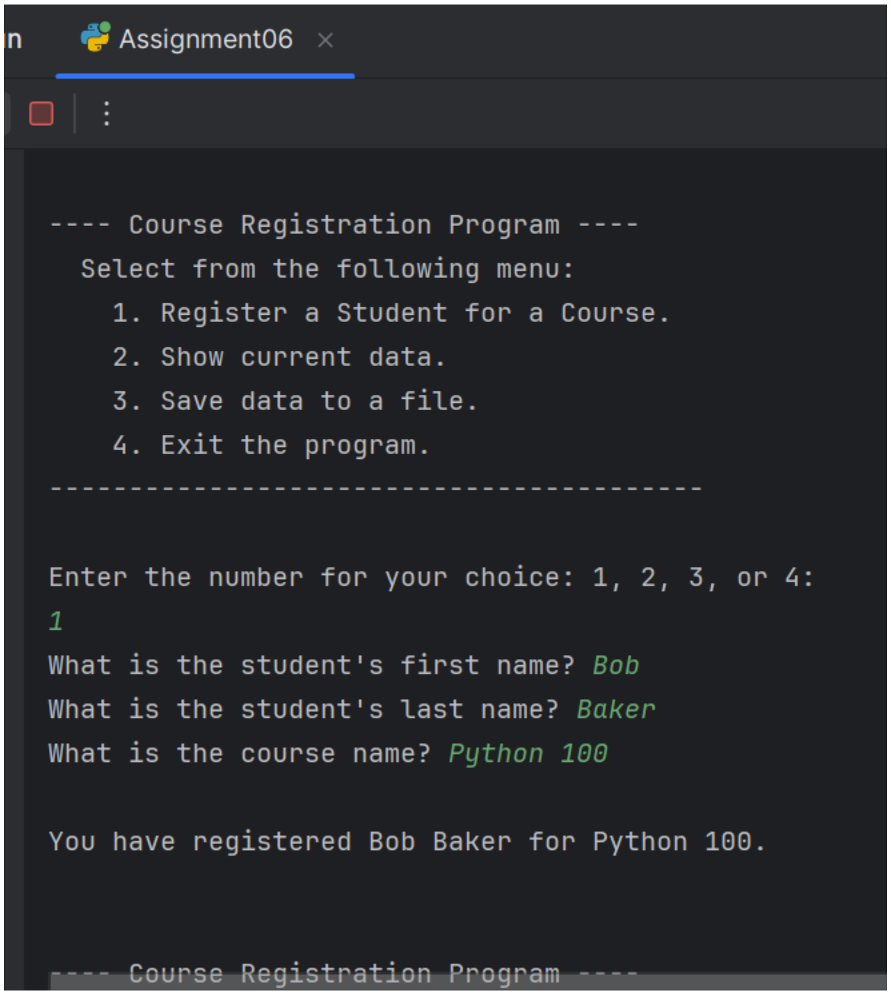
This assignment covered several concepts, such as declaring constants and variables and defining classes and functions. The program used *input()* and *print()* functions; processed data using dictionaries, lists and files in the JSON format. The program also included structured error handling. The program utilized Separation of Concerns (SoC) patterns to organize the classes and functions.

The program successfully accepted input from the program's user, of a student's first name and last name and course for enrollment. That input was then output to the user, then added to the existing "**students**" enrollments list, which was read and assigned memory when the enrollments file was opened. The updated "**students**" enrollments list was then successfully written to the enrollments file. Errors with the file, and errors with the input of the student's first name and last name were addressed within the program. The program was successfully tested in the PyCharm IDE and the Windows Command Prompt console.

The testing of the program is shown in Appendix A and the full code is shown in Appendix B.

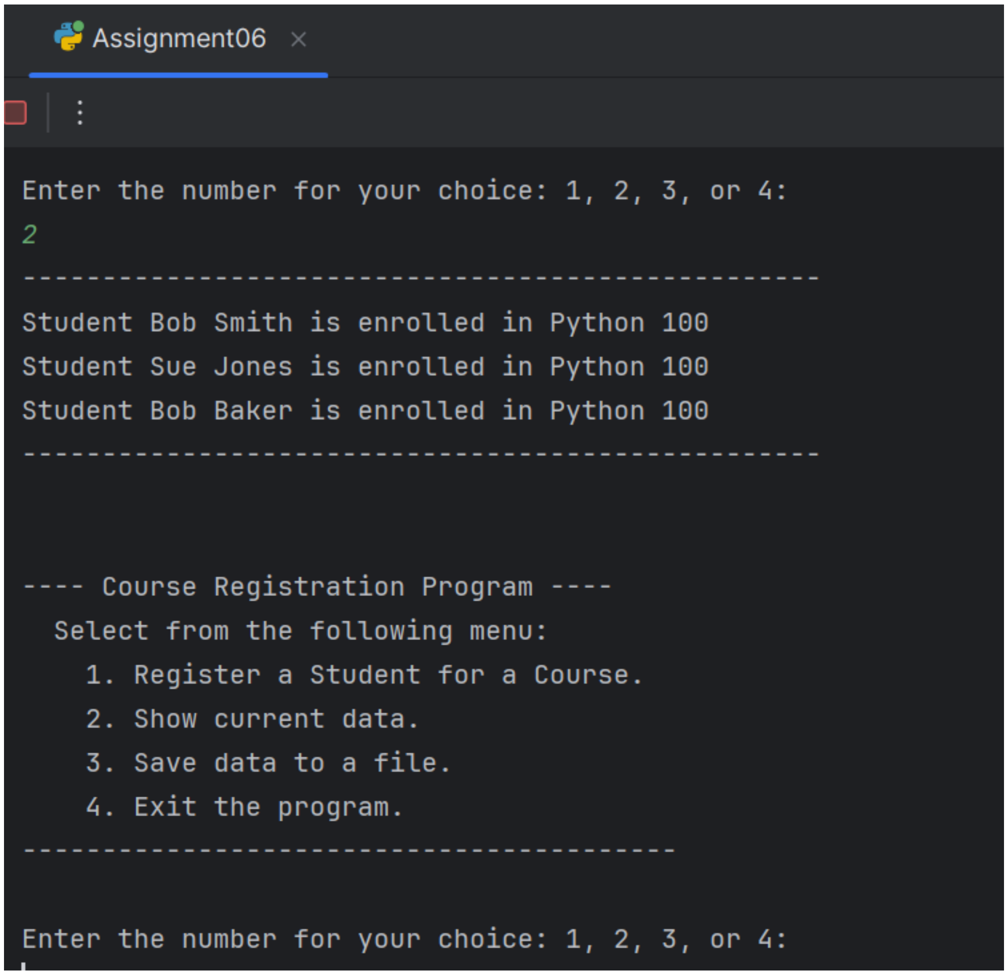
Appendix A – Testing the program

PyCharm testing



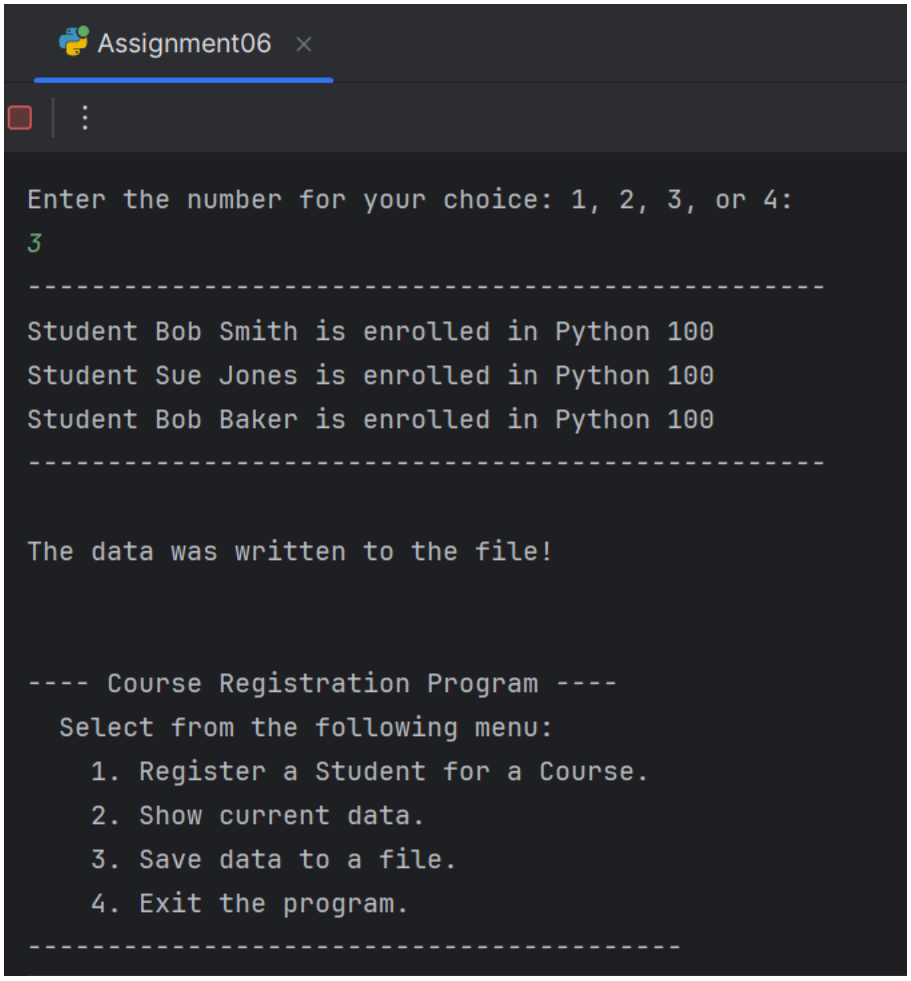
```
---- Course Registration Program ----
  Select from the following menu:
    1. Register a Student for a Course.
    2. Show current data.
    3. Save data to a file.
    4. Exit the program.
-----------------------------------------

Enter the number for your choice: 1, 2, 3, or 4:
1
What is the student's first name? Bob
What is the student's last name? Baker
What is the course name? Python 100

You have registered Bob Baker for Python 100.


---- Course Registration Program ----
```

Figure 14 - Choice 1, Input of student's first & last name & course



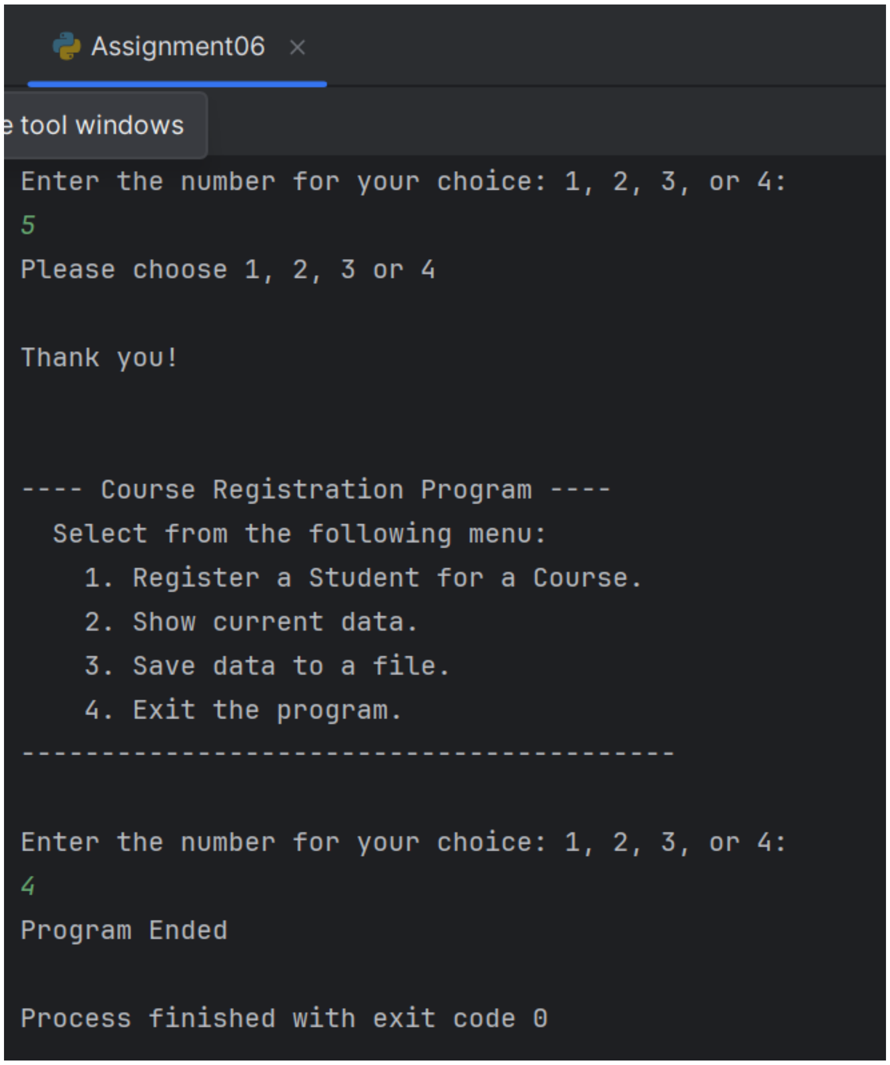
```
Enter the number for your choice: 1, 2, 3, or 4:
2
--------------------------------------------------
Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Bob Baker is enrolled in Python 100
--------------------------------------------------



---- 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.
-----------------------------------------

Enter the number for your choice: 1, 2, 3, or 4:
```

Figure 15 - Choice 2. Display of student's first & last name & course input



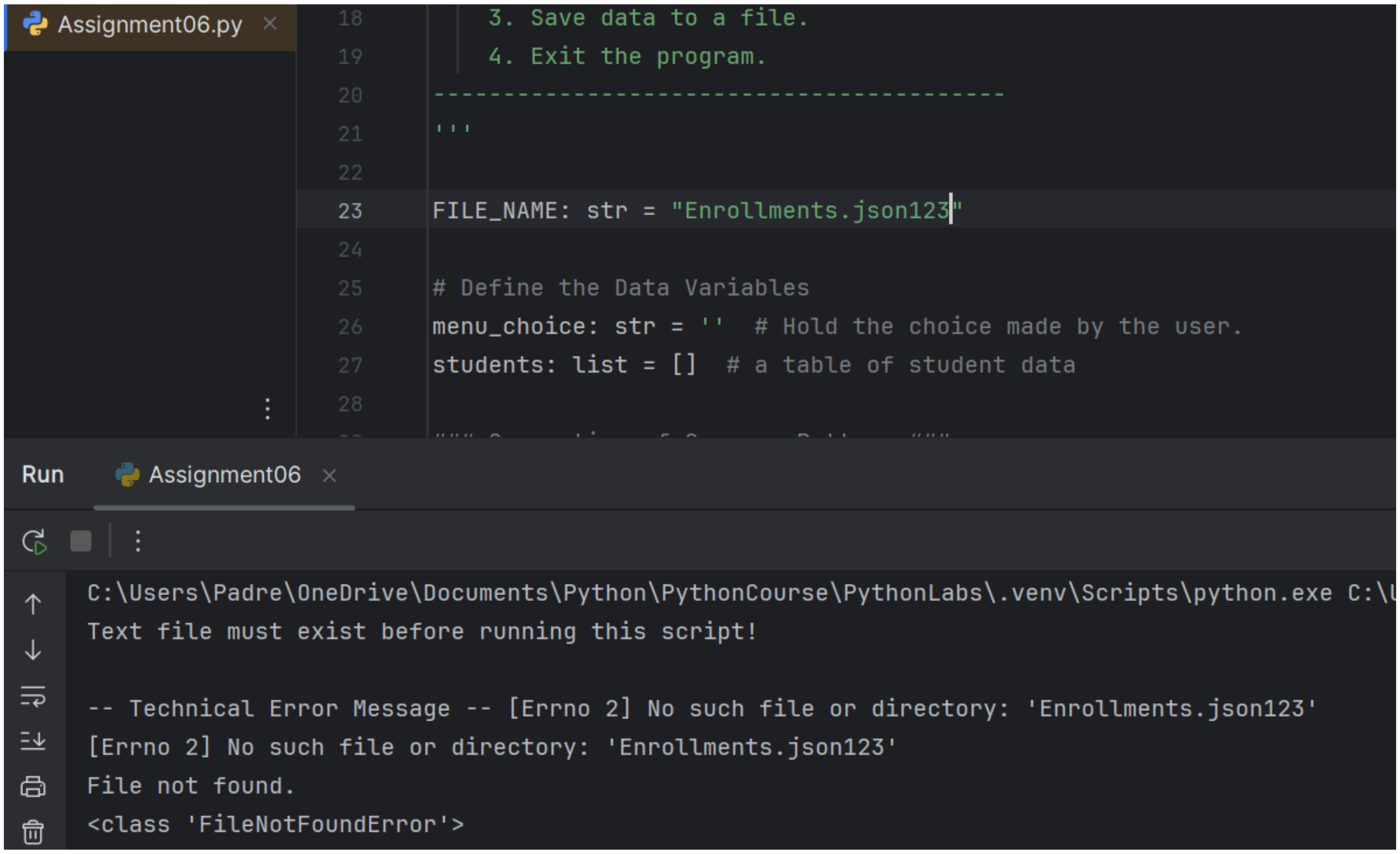
```
Enter the number for your choice: 1, 2, 3, or 4:
3
--------------------------------------------------
Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Bob Baker is enrolled in Python 100
--------------------------------------------------


The data was written to the 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.
-----------------------------------------
```

Figure 16 - Choice 3



```
Enter the number for your choice: 1, 2, 3, or 4:
5
Please choose 1, 2, 3 or 4

Thank you!


---- 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 the number for your choice: 1, 2, 3, or 4:
4
Program Ended

Process finished with exit code 0
```

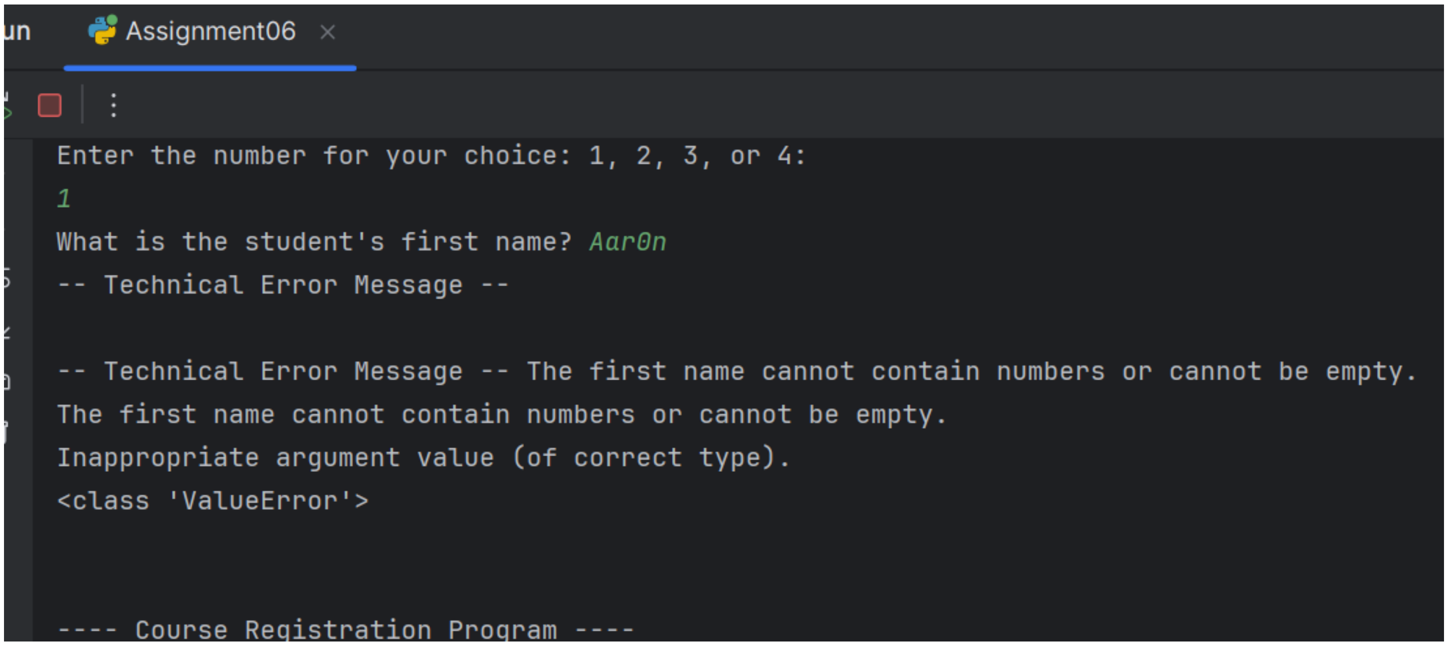
Figure 17 - Undefined choice was made, then Option 4 chosen



```
    3. Save data to a file.
    4. Exit the program.
----------------------------------------
'''

FILE_NAME: str = "Enrollments.json123"

# Define the Data Variables
menu_choice: str = ''  # Hold the choice made by the user.
students: list = []  # a table of student data
```

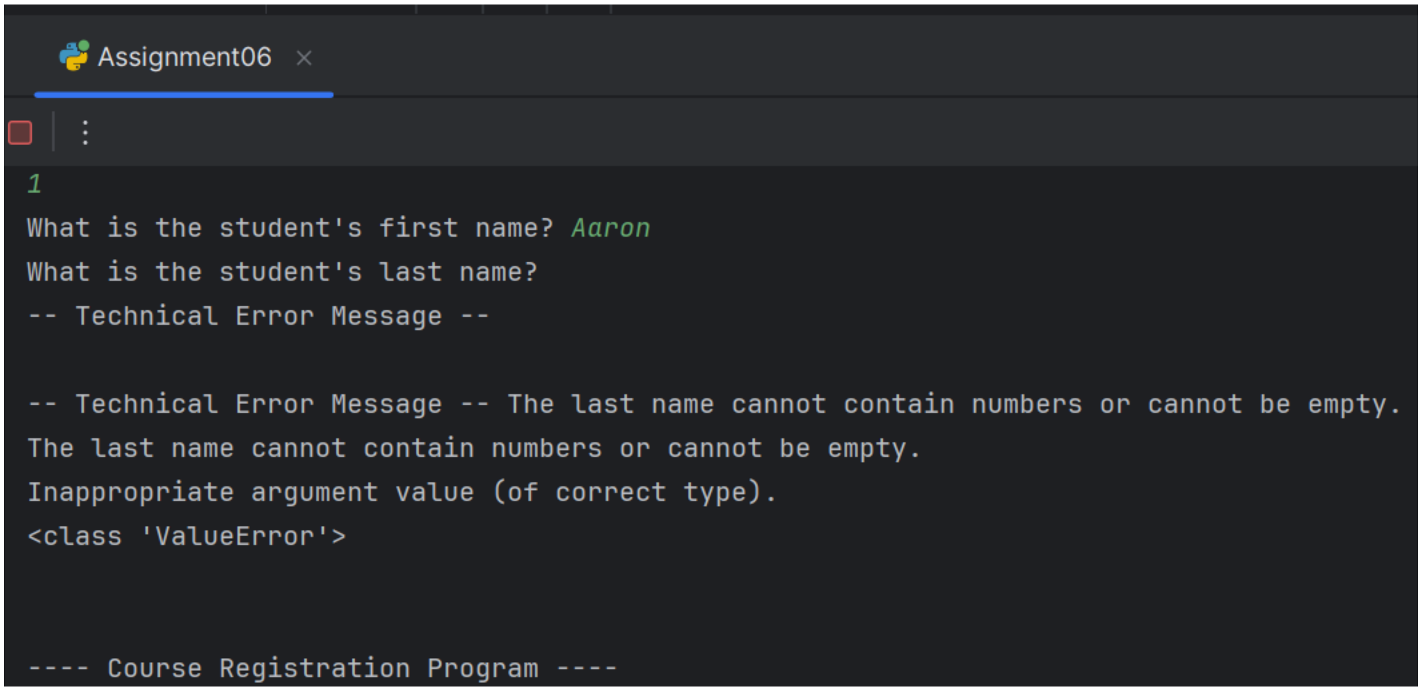
```
C:\Users\Padre\OneDrive\Documents\Python\PythonCourse\PythonLabs\.venv\Scripts\python.exe C:\U
Text file must exist before running this script!

-- Technical Error Message -- [Errno 2] No such file or directory: 'Enrollments.json123'
[Errno 2] No such file or directory: 'Enrollments.json123'
File not found.
<class 'FileNotFoundError'>
```

Figure 18 - Testing the FileNotFoundError



```
Enter the number for your choice: 1, 2, 3, or 4:
1
What is the student's first name? Aar0n
-- Technical Error Message --

-- Technical Error Message -- The first name cannot contain numbers or cannot be empty.
The first name cannot contain numbers or cannot be empty.
Inappropriate argument value (of correct type).
<class 'ValueError'>



---- Course Registration Program ----
```

Figure 19 - Testing student first name error- handling

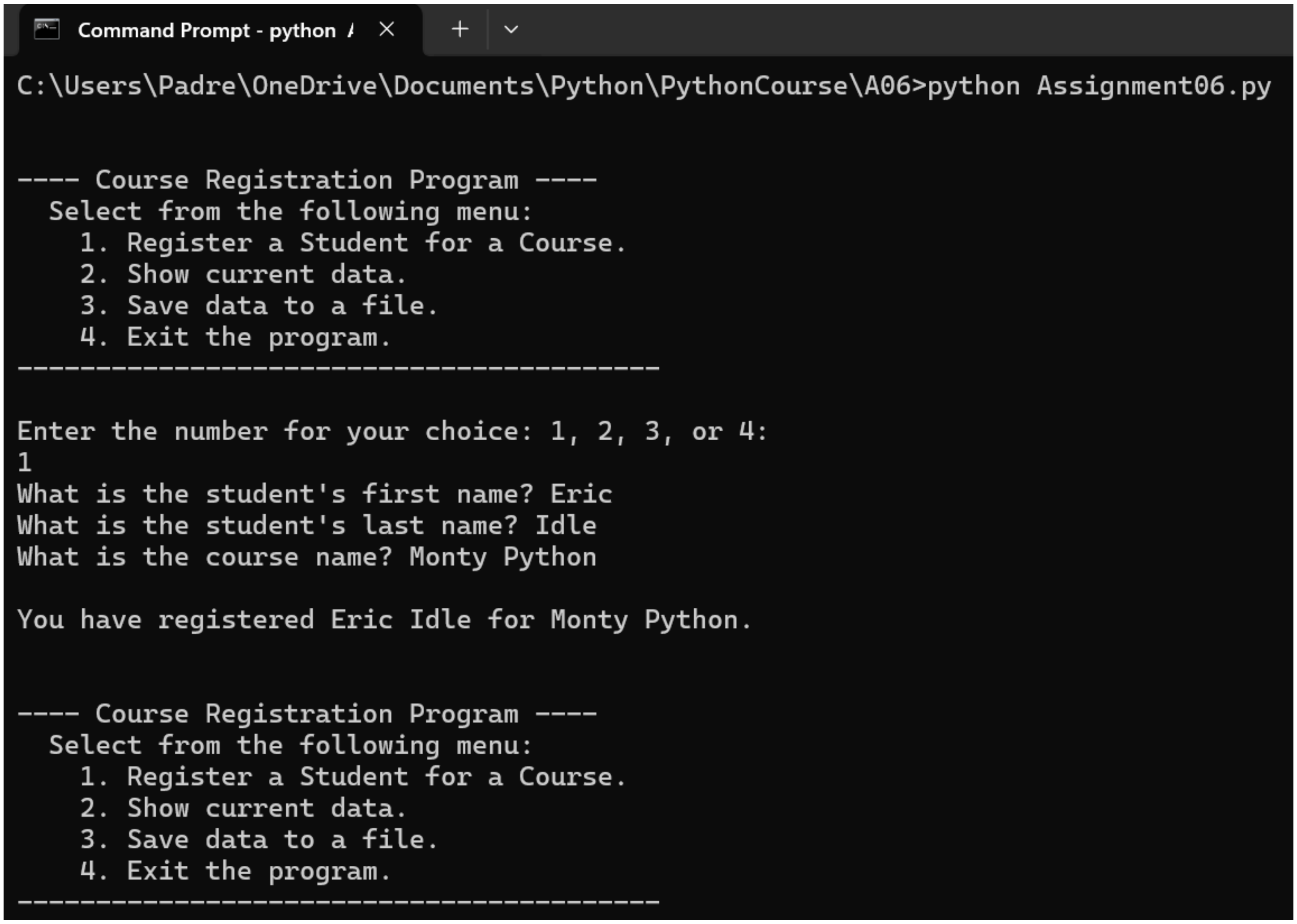


Assignment06

```
1
What is the student's first name? Aaron
What is the student's last name?
-- Technical Error Message --

-- Technical Error Message -- The last name cannot contain numbers or cannot be empty.
The last name cannot contain numbers or cannot be empty.
Inappropriate argument value (of correct type).
<class 'ValueError'>


---- Course Registration Program ----
```

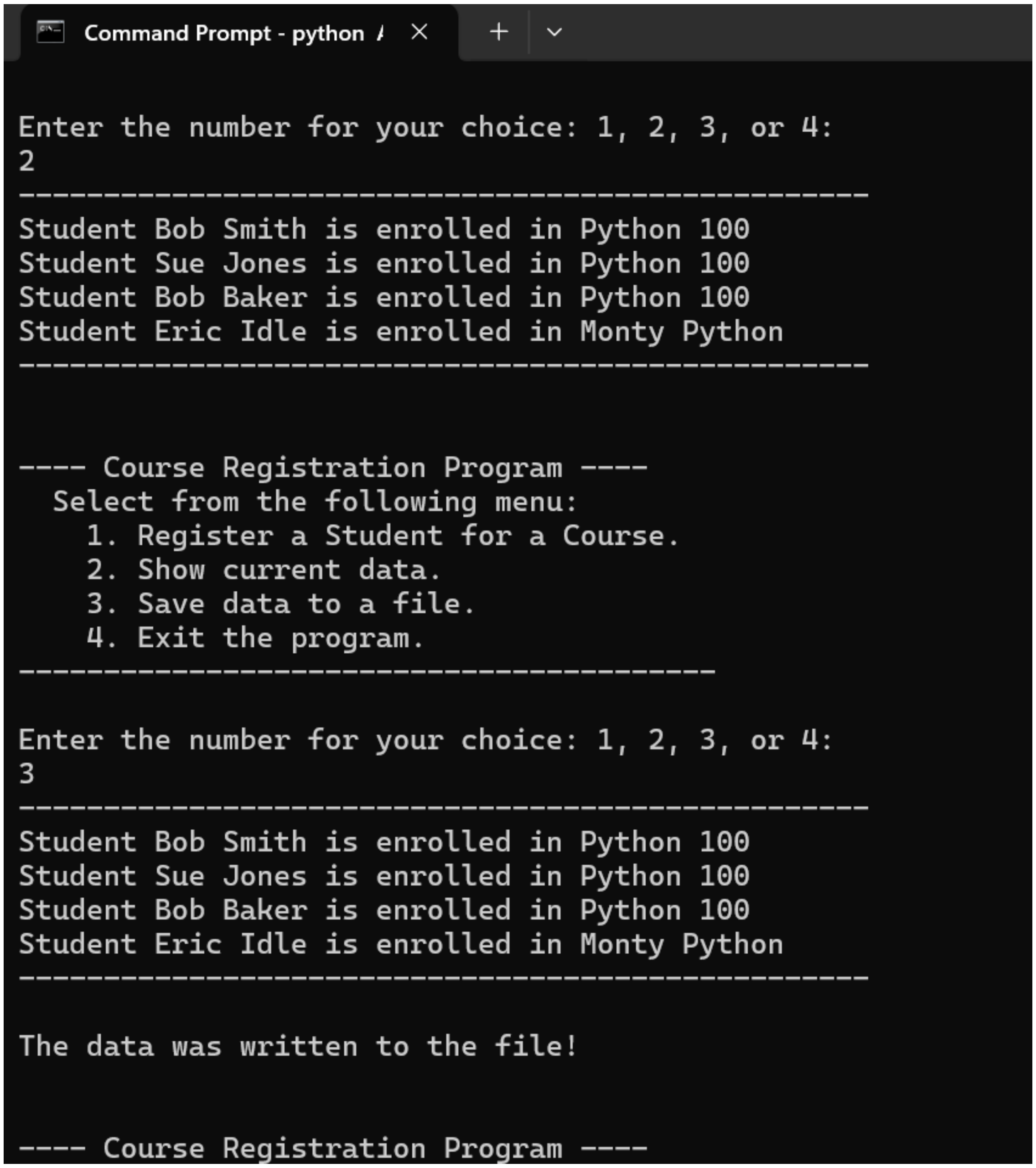
Figure 20 - Testing student last name error-handling

Console testing



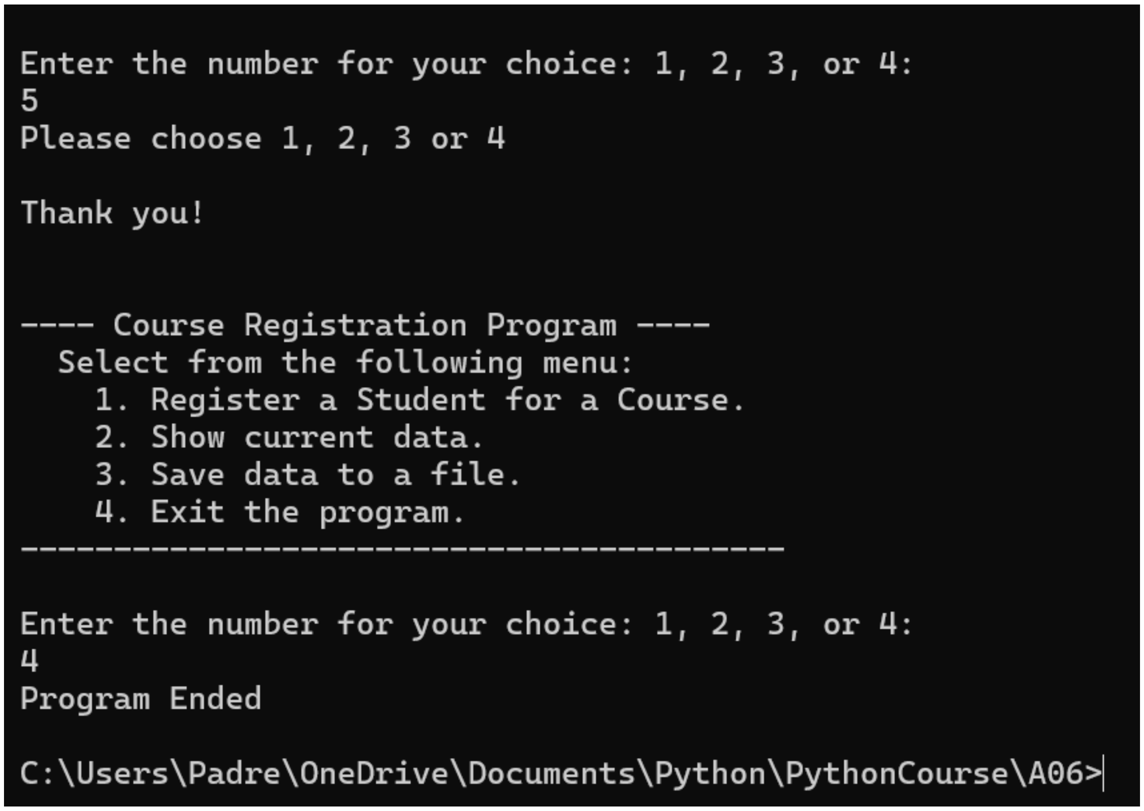
```
C:\Users\Padre\OneDrive\Documents\Python\PythonCourse\A06>python Assignment06.py


---- 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.
-----------------------------------------

Enter the number for your choice: 1, 2, 3, or 4:
1
What is the student's first name? Eric
What is the student's last name? Idle
What is the course name? Monty Python

You have registered Eric Idle for Monty Python.


---- 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 21 - Console testing, choice 1



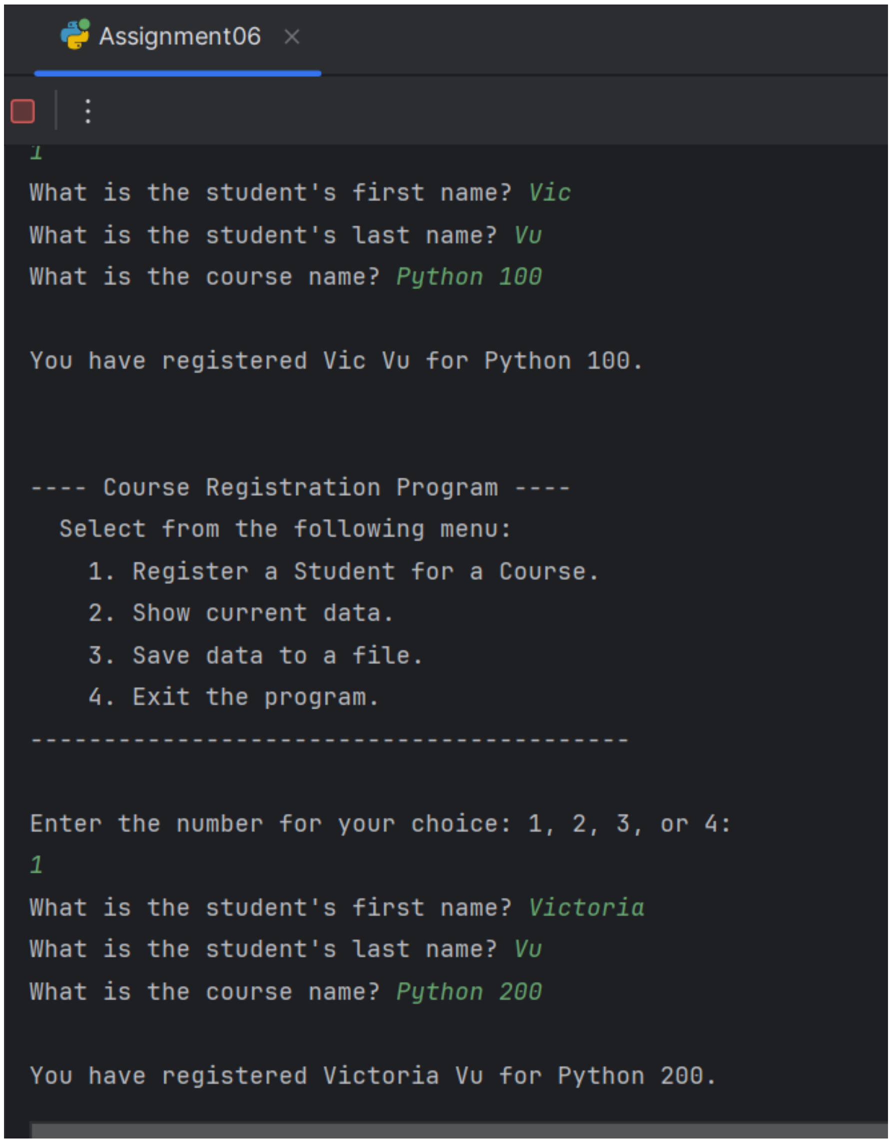
```
Enter the number for your choice: 1, 2, 3, or 4:
2
--------------------------------------------------
Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Bob Baker is enrolled in Python 100
Student Eric Idle is enrolled in Monty Python
--------------------------------------------------



---- 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 the number for your choice: 1, 2, 3, or 4:
3
--------------------------------------------------
Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Bob Baker is enrolled in Python 100
Student Eric Idle is enrolled in Monty Python
--------------------------------------------------

The data was written to the file!


---- Course Registration Program ----
```

Figure 22 - Console testing, choice 2 and 3



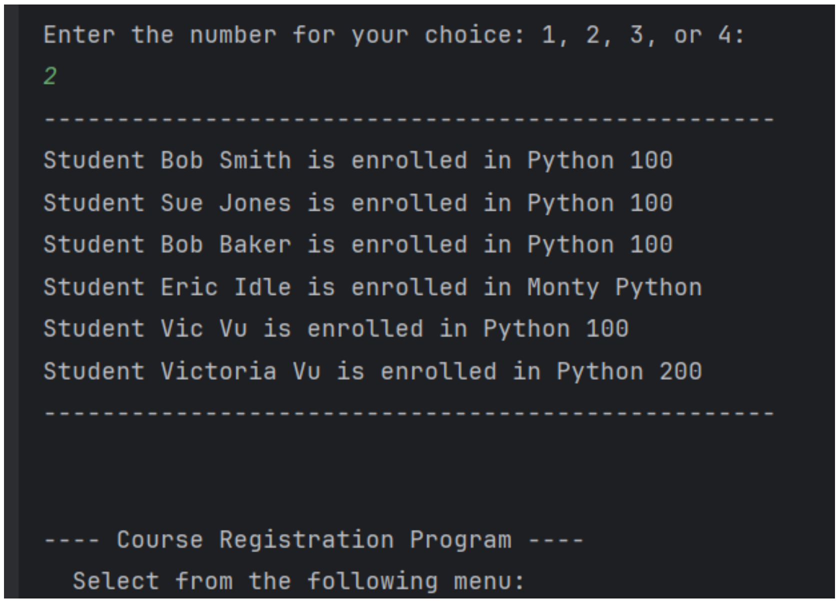
```
Enter the number for your choice: 1, 2, 3, or 4:
5
Please choose 1, 2, 3 or 4

Thank you!


---- 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 the number for your choice: 1, 2, 3, or 4:
4
Program Ended

C:\Users\Padre\OneDrive\Documents\Python\PythonCourse\A06>
```

Figure 23 - "Invalid" choice and choice 4



```
1
What is the student's first name? Vic
What is the student's last name? Vu
What is the course name? Python 100

You have registered Vic Vu for 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.
-----------------------------------------


Enter the number for your choice: 1, 2, 3, or 4:
1
What is the student's first name? Victoria
What is the student's last name? Vu
What is the course name? Python 200

You have registered Victoria Vu for Python 200.
```

Figure 24 - Testing multiple registrations



```
Enter the number for your choice: 1, 2, 3, or 4:
2
--------------------------------------------------
Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Bob Baker is enrolled in Python 100
Student Eric Idle is enrolled in Monty Python
Student Vic Vu is enrolled in Python 100
Student Victoria Vu is enrolled in Python 200
--------------------------------------------------



---- Course Registration Program ----
  Select from the following menu:
```

Figure 25 - Displaying the multiple registrations from previous test

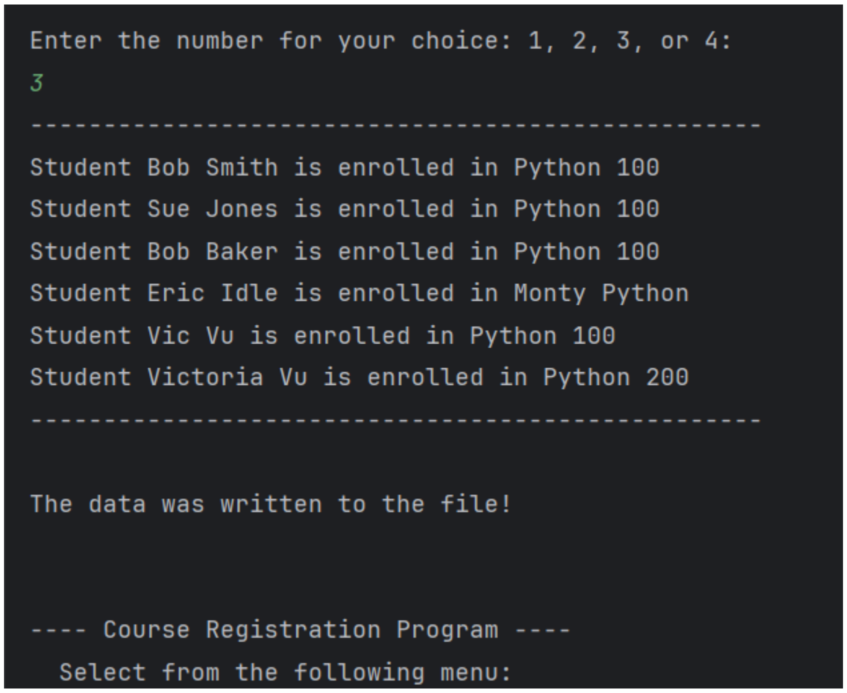
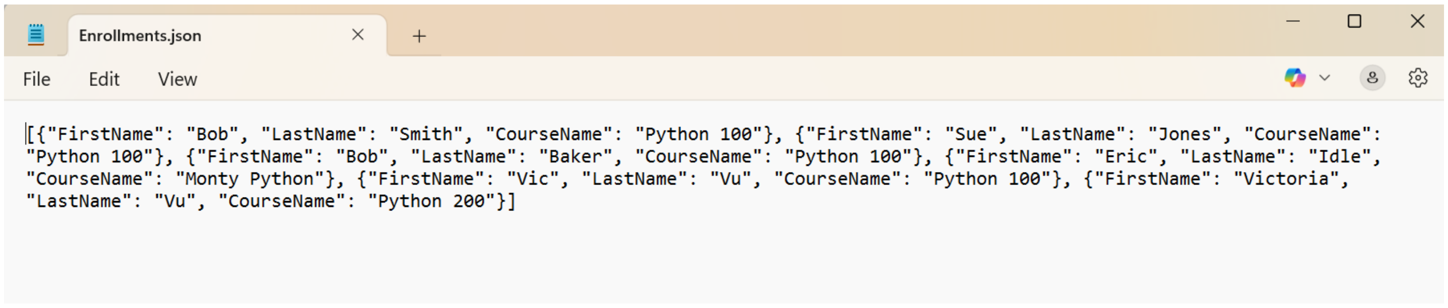
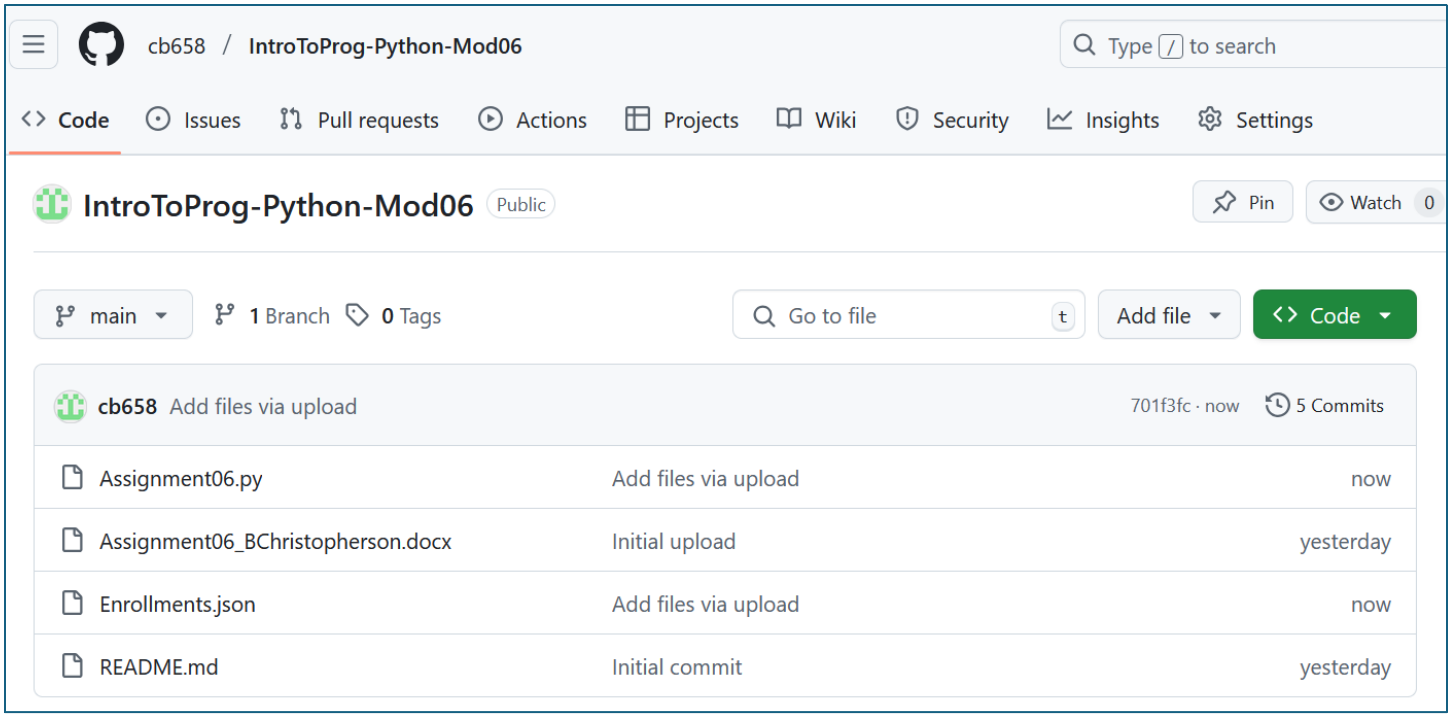

```
Enter the number for your choice: 1, 2, 3, or 4:
3
--------------------------------------------------
Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Bob Baker is enrolled in Python 100
Student Eric Idle is enrolled in Monty Python
Student Vic Vu is enrolled in Python 100
Student Victoria Vu is enrolled in Python 200
--------------------------------------------------


The data was written to the file!



---- Course Registration Program ----
  Select from the following menu:
```

Figure 26 - Writing multiple registrations to file



```
[{"FirstName": "Bob", "LastName": "Smith", "CourseName": "Python 100"}, {"FirstName": "Sue", "LastName": "Jones", "CourseName":
"Python 100"}, {"FirstName": "Bob", "LastName": "Baker", "CourseName": "Python 100"}, {"FirstName": "Eric", "LastName": "Idle",
"CourseName": "Monty Python"}, {"FirstName": "Vic", "LastName": "Vu", "CourseName": "Python 100"}, {"FirstName": "Victoria",
"LastName": "Vu", "CourseName": "Python 200"}]
```

Figure 27 - Review of Enrollments.json after all tests completed


cb658 / IntroToProg-Python-Mod06

cb658 Add files via upload	701f3fc · now	5 Commits
Assignment06.py	Add files via upload	now
Assignment06_BChristopherson.docx	Initial upload	yesterday
Enrollments.json	Add files via upload	now
README.md	Initial commit	yesterday

Figure 28 - Files uploaded to GitHub

Appendix B – Full code

```
# ------------------------------------------------------------------------------------------ #
# Title: Assignment06
# Desc: This assignment demonstrates using functions and classes & organized by
#       Separation of Concerns patterns
# Change Log: (Who, When, What)
#   BChristopherson, 5/26/2025,Created Script
#   BChristopherson, 5/26/2025,Updated Script and added additional comments
# ------------------------------------------------------------------------------------------ #

import json

# 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.
-----------------------------------------
'''

FILE_NAME: str = "Enrollments.json"

# Define the Data Variables
menu_choice: str = ''  # Hold the choice made by the user.
students: list = []  # a table of student data

### Separation of Concern Pattern ###
# PROCESSING --------------------------------------

# Define the Classes
#-------------------
# A single class is being used for data processing
class FileProcessor:
    """
    A collection of functions related to data processing

    ChangeLog: (Who, When, What)
    BChristopherson, 5-26-2025, Created Class
    """

    @staticmethod
    def read_data_from_file(file_name: str, student_data: list):
        """ This function reads student data from the Enrollments.json file
        Change Log: (Who, When, What)
        BChristopherson, 5-26-25, Created function
        :param file_name: string with name of file

        :return: student_data
        """
        try:
            file = open(file_name, "r")
            student_data = json.load(file)
            file.close()

        except FileNotFoundError as e:
            IO.output_error_messages("Text file must exist before running"
                                     " this script!", e)
        except Exception as e:
            IO.output_error_messages("There was a non-specific error!", e)
        finally:
            if file and not file.closed:
                file.close()
        return student_data

    @staticmethod
    def write_data_to_file(file_name: str, student_data: list):
        """ This function writes student data to the Enrollments.json file
        Change Log: (Who, When, What)
        BChristopherson, 5-26-25, Created function
        :param file_name of file to be written to
        :param student_data, table containing student enrollments
        :return: student_data, updated with the data that was written
        """
        try:
            file = open(file_name, "w")
            json.dump(student_data, file)
            file.close()
            IO.output_student_courses(student_data=student_data)
        except TypeError as e:
            IO.output_error_messages("Please check if the file is a"
                                     " valid JSON format file", e)
        except Exception as e:
            IO.output_error_messages("There was a non-specific error!", e)
        finally:
            if file.closed == False:
                file.closed


### Separation of Concern Pattern ###
# PRESENTATION  -----------------------------

# Define the Classes
#-------------------
# A single class is being used for presenting and handling data
class IO:
    """
    A collection of functions related to user inputs, and outputs to user

    ChangeLog: (Who, When, What)
    BChristopherson, 5-26-2025, Created Class
    """

    @staticmethod
    def output_error_messages(message: str, error: Exception = None):
        """ This function displays custom error messages to the user
        Change Log: (Who, When, What)
        BChristopherson, 5-26-25, Created function

        :param message:
        :param error: Exception error
        :return: No return data
        """
        print(message, end="\n\n")
        if error is not None:
            print("-- Technical Error Message --", error)
            print(error, error.__doc__, type(error), sep="\n")


    @staticmethod
    def output_menu(menu: str):
        """ This function displays the menu to the user
        Change Log: (Who, When, What)
        BChristopherson, 5-26-25, Created function

        :return: None
        """
        print()
        print(menu)

    @staticmethod
    def input_menu_choice():
        """ This function accepts the user's menu choice as an input
        Change Log: (Who, When, What)
        BChristopherson, 5-26-25, Created function
        :return: returns the user's choice, as a string
        """
        choice = "0"
        try:
            choice = input("Enter the number for your choice: 1, 2, 3, or 4: \n")
            if choice not in ("1","2","3","4"):
                raise Exception("Please choose 1, 2, 3 or 4")
        except Exception as e:
            IO.output_error_messages(e.__str__())

        return choice


    @staticmethod
    def output_student_courses(student_data: list):
        """ This function outputs the student enrollment information
            Change Log: (Who, When, What)
            BChristopherson, 5-26-25, Created function

            :param: student_data, table of student enrollments

            :return: None
            """

        print("-" *50)

        for student in student_data:
            print(f'Student {student['FirstName']} '
                  f'{student['LastName']} is enrolled in {student['CourseName']}')
        print("-" *50)


    @staticmethod
    def input_student_data(student_data: list):
        """ This function accepts the new student enrollment information, as input
            Change Log: (Who, When, What)
            BChristopherson, 5-26-25, Created function

            :param: student_data, dictionary for each new student
            :return: student_data list
            """

        try: # Input the data
            student_first_name = input("What is the student's first name? ")
            if not student_first_name.isalpha() or student_first_name == "":
                raise ValueError("The first name cannot contain numbers or cannot be empty.")
            student_last_name = input("What is the student's last name? ")
            if not student_last_name.isalpha() or student_last_name == "":
                raise ValueError("The last name cannot contain numbers or cannot be empty.")
            course_name = input("What is the course name? ")
            if not course_name:
                raise ValueError('Response cannot be empty')

            student_data = {"FirstName": student_first_name,
                            "LastName": student_last_name,
                            "CourseName": course_name}

            # Add new student to the students table
            students.append(student_data)
            print()
            print(f"You have registered {student_first_name} {student_last_name} for
{course_name}.")
        except ValueError as e:
            IO.output_error_messages("-- Technical Error Message -- ", e)
        except Exception as e:
            IO.output_error_messages("Error: There was a problem with your entered data.")
        return student_data

# Start of main body

# When the program starts, read the file data into a list of lists (table)
students = FileProcessor.read_data_from_file(file_name=FILE_NAME, student_data=students)

# Present and Process the data
while (True):

    # Present the menu of choices
    IO.output_menu(menu=MENU)

    menu_choice = IO.input_menu_choice()

    # Input new student
    if menu_choice == "1":  # This will not work if it is an integer!
        IO.input_student_data(student_data=students)
        continue

    # Display current data
    elif menu_choice == "2": # This will not work if it is an integer!
        IO.output_student_courses(student_data=students)
        continue

    # Save the data to a file
    elif menu_choice == "3":  # This will not work if it is an integer!
        FileProcessor.write_data_to_file(file_name=FILE_NAME, student_data=students)
        print("\nThe data was written to the file!")
        continue

    # Stop the loop
    elif menu_choice == "4":  # This will not work if it is an integer!
        break  # out of the loop
    else:
        print("Thank you!")
        # input_menu_choice function already gives message to user \
        # if user enters values other than 1,2,3 or 4

print("Program Ended")
```