Kyle Hayward

11/21/2023

Foundations of Programming: Python

Assignment 06

GitHub Link: https://github.com/HaywardUW/IntroToProg-Python-Mod06

Functions, Classes, and Parameters

Introduction

Module 06 introduced functions, classes, and parameters. I learned how to organize my code within those classes and how to implement parameters within the functions.

In this Assignment 06 documentation, I will explain what functions are and how to use parameters. I will also cover what classes are and why they are so important when creating Python programs.

I will then demonstrate how to create a program using the newly learned concepts. Keep in mind that this program is very similar to the Assignment 05 program, however, we are now implementing code grouping to create modular structures as a way to organize the code along with defining our own functions.

Finally, I will launch the program through the PyCharm IDE to validate its initial functionality. I'll complete the demonstration by running the program to completion through the Windows OS Command Prompt.

Classes

Classes are a way to organize your code. It not only makes it easier to maintain, but also makes it easier for other developers to read your code. This is a fundamental concept when creating programs.

In Python, a class is created by first typing 'class', followed by the class name and a colon. For example:

```
Class FileProcessor:
```

After creating the class, a docstring containing a description of the class followed by a change log is added. For example:

```
Class FileProcessor:
    “””
    A set of processing functions the reads from and writes to JSON files.
    ChangeLog: (Who, When, What)
    Kyle Hayward, 11/21/2023, Created Class
```

“””

I will cover more about classes when I demonstrate the details of the program. The most important thing to remember is that classes are used to organize your code. A FileProcessor class might be used when opening and manipulating files. An InputOutput class could be used when presenting data back to the user of the program.

The next section (Functions and Parameters) will cover blocks of code that can be used once a class has been defined.

Functions and Parameters

A function is a reusable block of code. Functions are fundamental concepts in the Python programming language that allow for a program’s modularity and reusability.

There is a plethora of functions in Python. We have been using one of the most widely used Python functions, the print() function. The input() function is another function we have used extensively in this course.

We can also define our own functions by grouping a set of programming statements that we can reference later in the program.

Functions are defined by typing ‘def’ then the name of the function, parenthesis, followed by a colon. For example:

def read_data_from_file():

Note the parenthesis after the function name. These are used to declare parameters which accept arguments. For instance, you can set two parameters within the function. When the function is called, those parameters are completed with arguments that complete the functions task. For example:

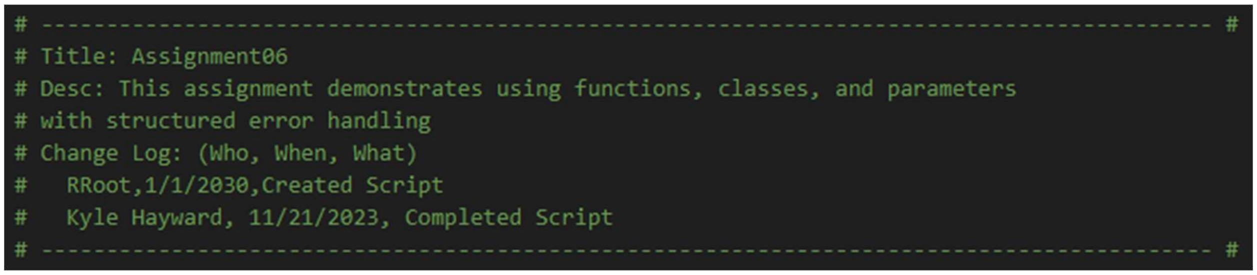
def read_data_from_file(file_name=FILE_NAME, student_data=students):

The above example has two parameters, file_name and student_data. The file-name parameter is completed with FILE_NAME as the argument, while the student_data parameter is completed with the students variable.

I will cover functions, parameters, and arguments in the Creating the Program section of this documentation.

Creating the Program

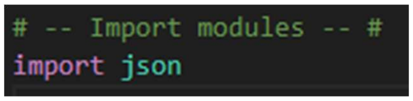
As with all the programs created in this course, I start the program code with a script header. (Figure 1.1)



```
# ------------------------------------------------------------------------------------------ #
# Title: Assignment06
# Desc: This assignment demonstrates using functions, classes, and parameters
# with structured error handling
# Change Log: (Who, When, What)
#   RRoot,1/1/2030,Created Script
#   Kyle Hayward, 11/21/2023, Completed Script
# ------------------------------------------------------------------------------------------ #
```

Figure 1.1: Starting the program code with a script header.

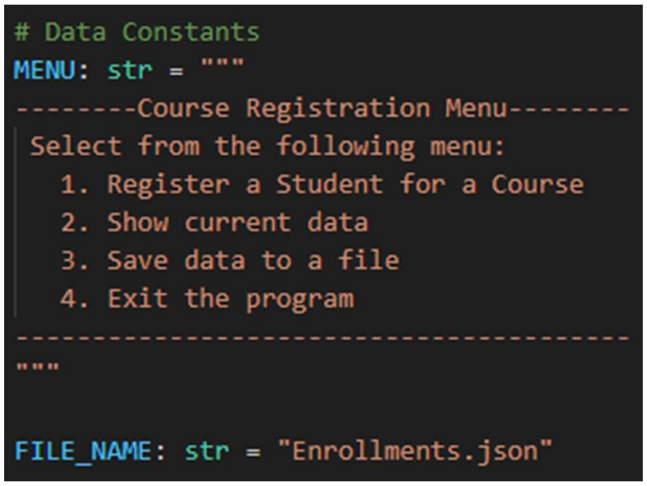
Since this program creates, reads from, and writes to a JSON file, I import the ‘json’ module. (Figure 1.2)



```
# -- Import modules -- #
import json
```

Figure 1.2: Importing the ‘json' module.

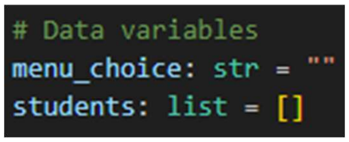
Next, I define the two Data Constants which are MENU and FILE_NAME. MENU contains the Course Registration Menu while FILE_NAME holds the “Enrollments.json” file. (Figure 1.3)



```
# Data Constants
MENU: str = """
--------Course Registration Menu--------
 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"
```

Figure 1.3: Defining the Data Constant variables.

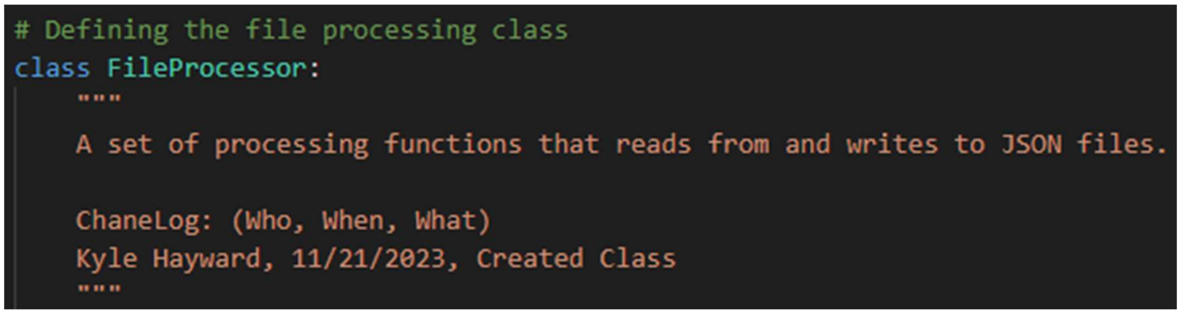
There are only two data variables to define for this program, menu_choice and students. Menu_choice is an empty string, while students is an empty list. (Figure 1.4)



```
# Data variables
menu_choice: str = ""
students: list = []
```

Figure 1.4: Defining the two data variables.

I now start the file processing portion of the program by creating a class called FileProcessor followed by a docstring. (Figure 1.5)



```
# Defining the file processing class
class FileProcessor:
    """
    A set of processing functions that reads from and writes to JSON files.

    ChaneLog: (Who, When, What)
    Kyle Hayward, 11/21/2023, Created Class
    """
```

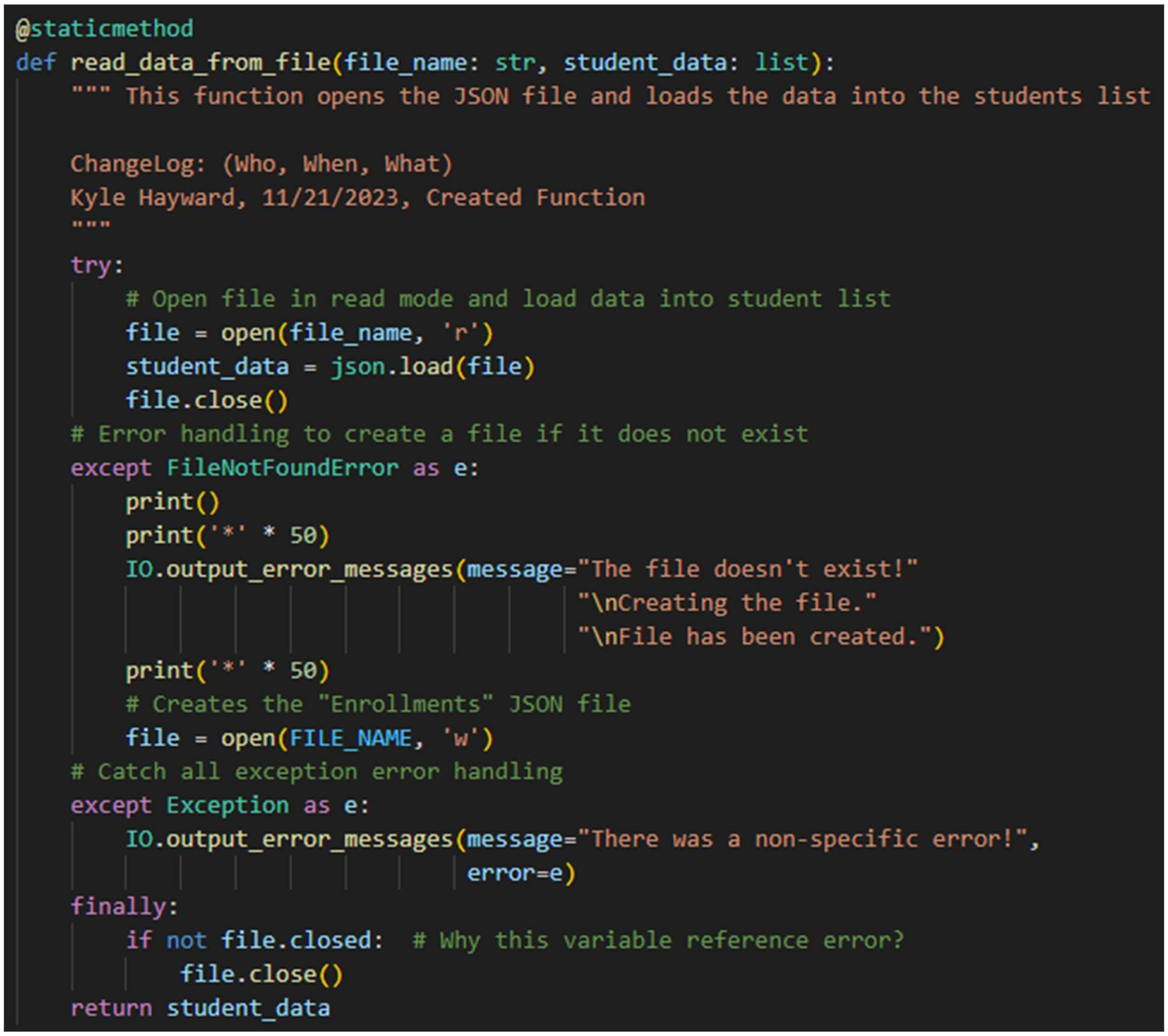
Figure 1.5: Defining the FileProcessor class with a docstring.

Within the FileProcessor class, I define the read_data_from_file function. This function has two parameters, file_name and student_data which will be used later in the program when the function is called.

The read_data_from_file function opens the specified file in read mode and loads the data into a list before closing.

Also included in this function is exception handling that creates the “Enrollments.json” file if it doesn’t exist, displays a generic error to catch other unforeseen exceptions, and closes the file is not already.

Figure 1.6 demonstrates the full read_data_from_file function block.

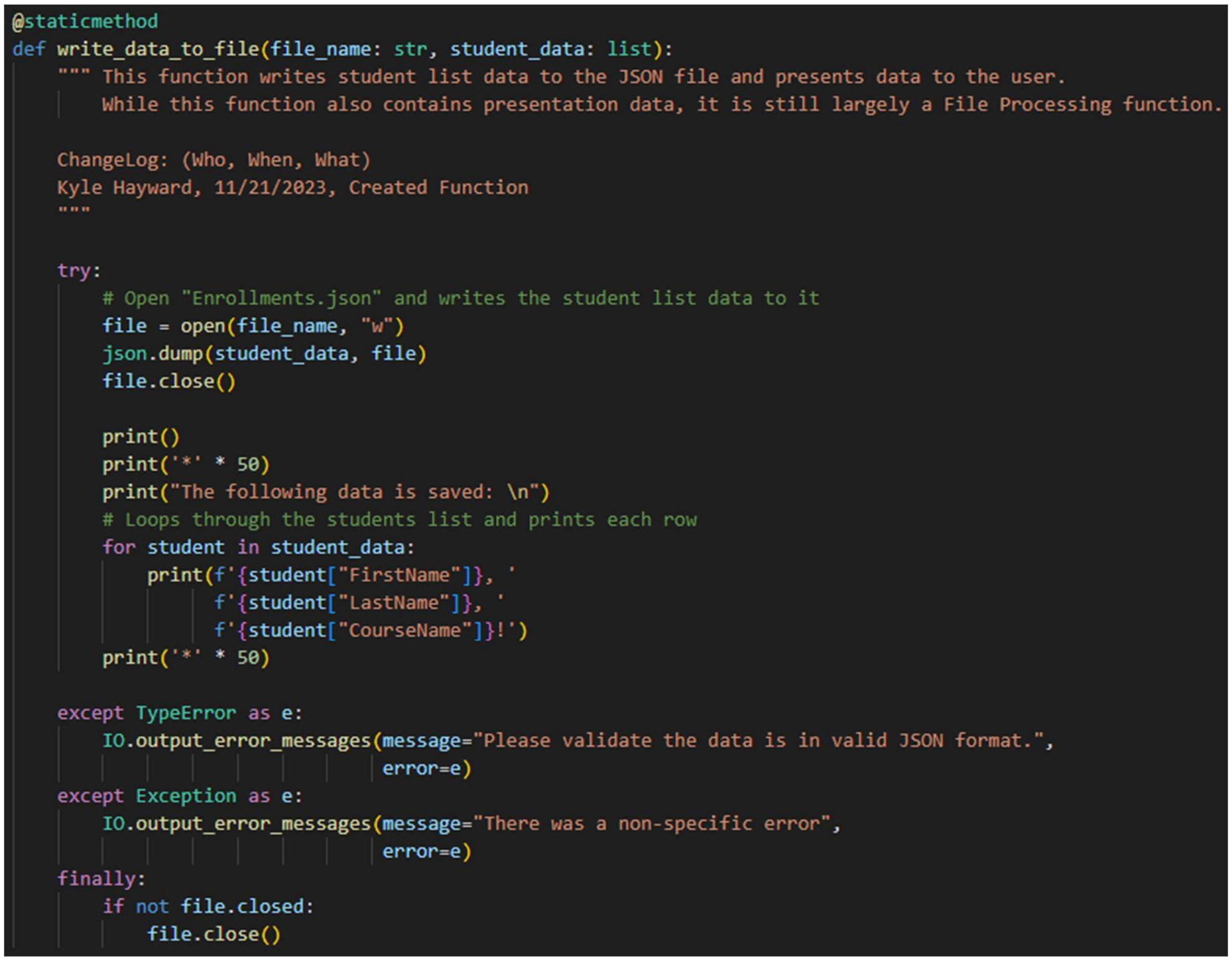


```python
@staticmethod
def read_data_from_file(file_name: str, student_data: list):
    """ This function opens the JSON file and loads the data into the students list

    ChangeLog: (Who, When, What)
    Kyle Hayward, 11/21/2023, Created Function
    """
    try:
        # Open file in read mode and load data into student list
        file = open(file_name, 'r')
        student_data = json.load(file)
        file.close()
    # Error handling to create a file if it does not exist
    except FileNotFoundError as e:
        print()
        print('*' * 50)
        IO.output_error_messages(message="The file doesn't exist!"
                                         "\nCreating the file."
                                         "\nFile has been created.")
        print('*' * 50)
        # Creates the "Enrollments" JSON file
        file = open(FILE_NAME, 'w')
    # Catch all exception error handling
    except Exception as e:
        IO.output_error_messages(message="There was a non-specific error!",
                                 error=e)
    finally:
        if not file.closed:  # Why this variable reference error?
            file.close()
    return student_data
```

Figure 1.6: Defining the read_data_from_file function including exception handling.

The next function under the FileProcessor class is write_data_to_file. This function opens the “Enrollments.json” file in write mode, dumps the data to the student list, and displays the information back to the user. Exception handling is also implemented into this function to validate the file format, a catch all exception and a finally statement to close the file if it not already closed.

Figure 1.7 demonstrates the full write_data_to_file function block.

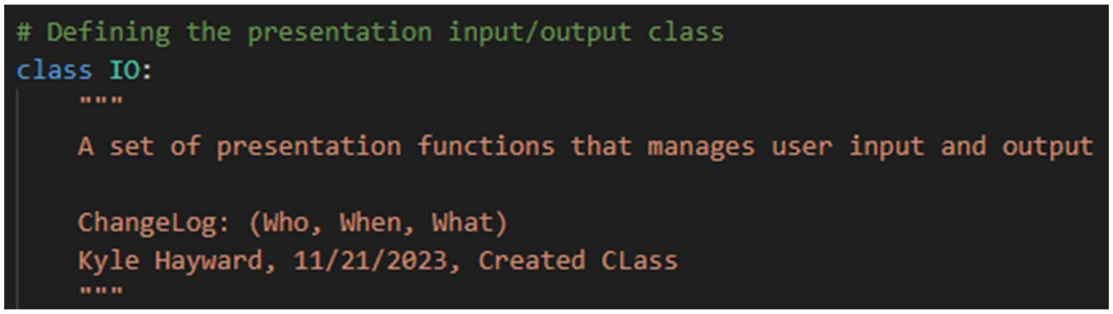


```python
@staticmethod
def write_data_to_file(file_name: str, student_data: list):
    """ This function writes student list data to the JSON file and presents data to the user.
        While this function also contains presentation data, it is still largely a File Processing function.

    ChangeLog: (Who, When, What)
    Kyle Hayward, 11/21/2023, Created Function
    """

    try:
        # Open "Enrollments.json" and writes the student list data to it
        file = open(file_name, "w")
        json.dump(student_data, file)
        file.close()

        print()
        print('*' * 50)
        print("The following data is saved: \n")
        # Loops through the students list and prints each row
        for student in student_data:
            print(f'{student["FirstName"]}, '
                  f'{student["LastName"]}, '
                  f'{student["CourseName"]}!')
        print('*' * 50)

    except TypeError as e:
        IO.output_error_messages(message="Please validate the data is in valid JSON format.",
                                 error=e)
    except Exception as e:
        IO.output_error_messages(message="There was a non-specific error",
                                 error=e)
    finally:
        if not file.closed:
            file.close()
```

Figure 1.7: Defining the write_data_to_file function block with exception handling.

The two functions documented above are the only functions contained in the FileProcessing class. The program now transitions into the presentation or input/output layer.

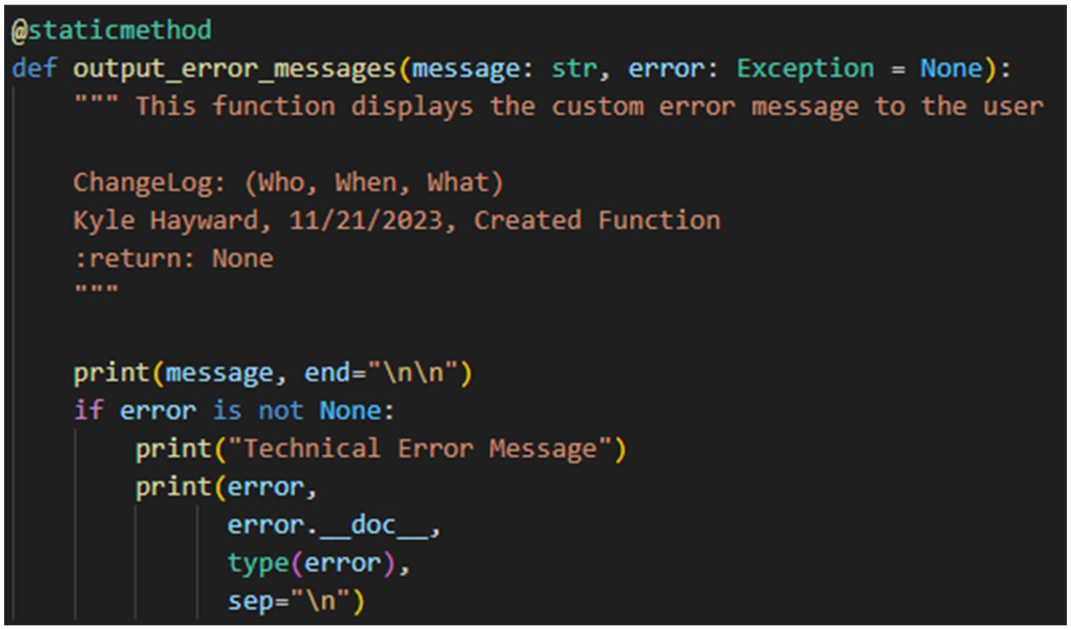
To start the presentation layer, I define a new class named IO followed by a docstring. (Figure 1.8)



```python
# Defining the presentation input/output class
class IO:
    """
    A set of presentation functions that manages user input and output

    ChangeLog: (Who, When, What)
    Kyle Hayward, 11/21/2023, Created CLass
    """
```

Figure 1.8: Defining the IO class to start the presentation layer of the program.

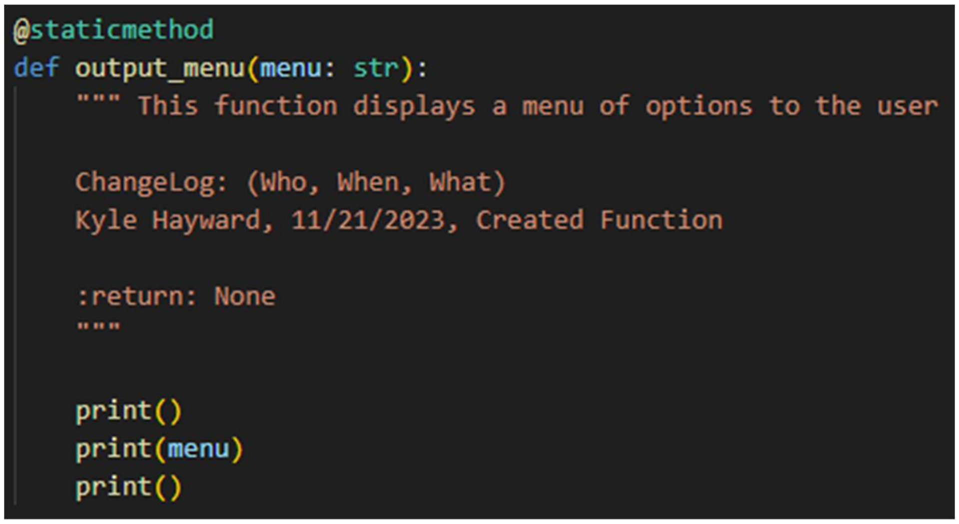
The IO class consists of five functions. The first function being the output_error_messages function. This function contains two parameters, message and Exception. The purpose of this function is to present error messages to the user when a program exception has been triggered. (Figure 1.9)



```python
@staticmethod
def output_error_messages(message: str, error: Exception = None):
    """ This function displays the custom error message to the user

    ChangeLog: (Who, When, What)
    Kyle Hayward, 11/21/2023, Created Function
    :return: None
    """

    print(message, end="\n\n")
    if error is not None:
        print("Technical Error Message")
        print(error,
              error.__doc__,
              type(error),
              sep="\n")
```

Figure 1.9: Defining a function that displays a custom error message to the user.

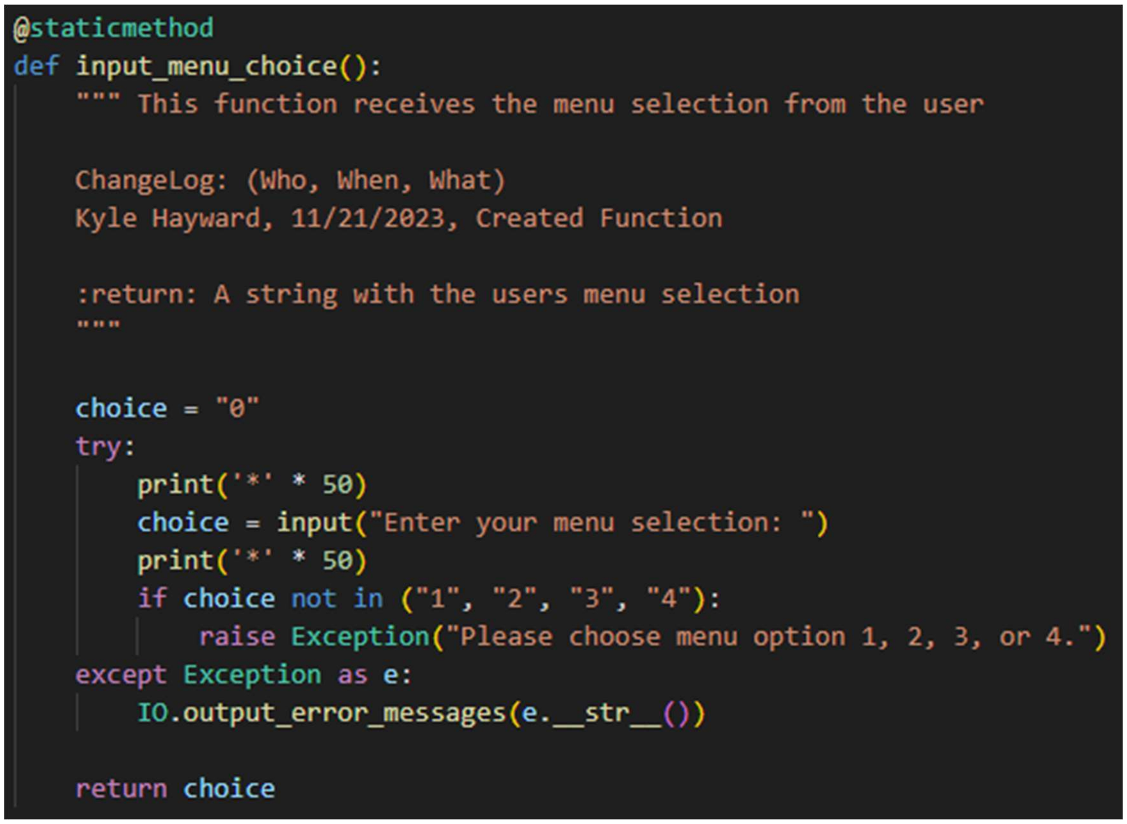
The second function under the IO class is the output_menu function. This function displays the Course Registration Menu to the user. (Figure 1.10)



```python
@staticmethod
def output_menu(menu: str):
    """ This function displays a menu of options to the user

    ChangeLog: (Who, When, What)
    Kyle Hayward, 11/21/2023, Created Function

    :return: None
    """

    print()
    print(menu)
    print()
```

Figure 1.10: Defining the output_menu function to display the Course Registration Menu.

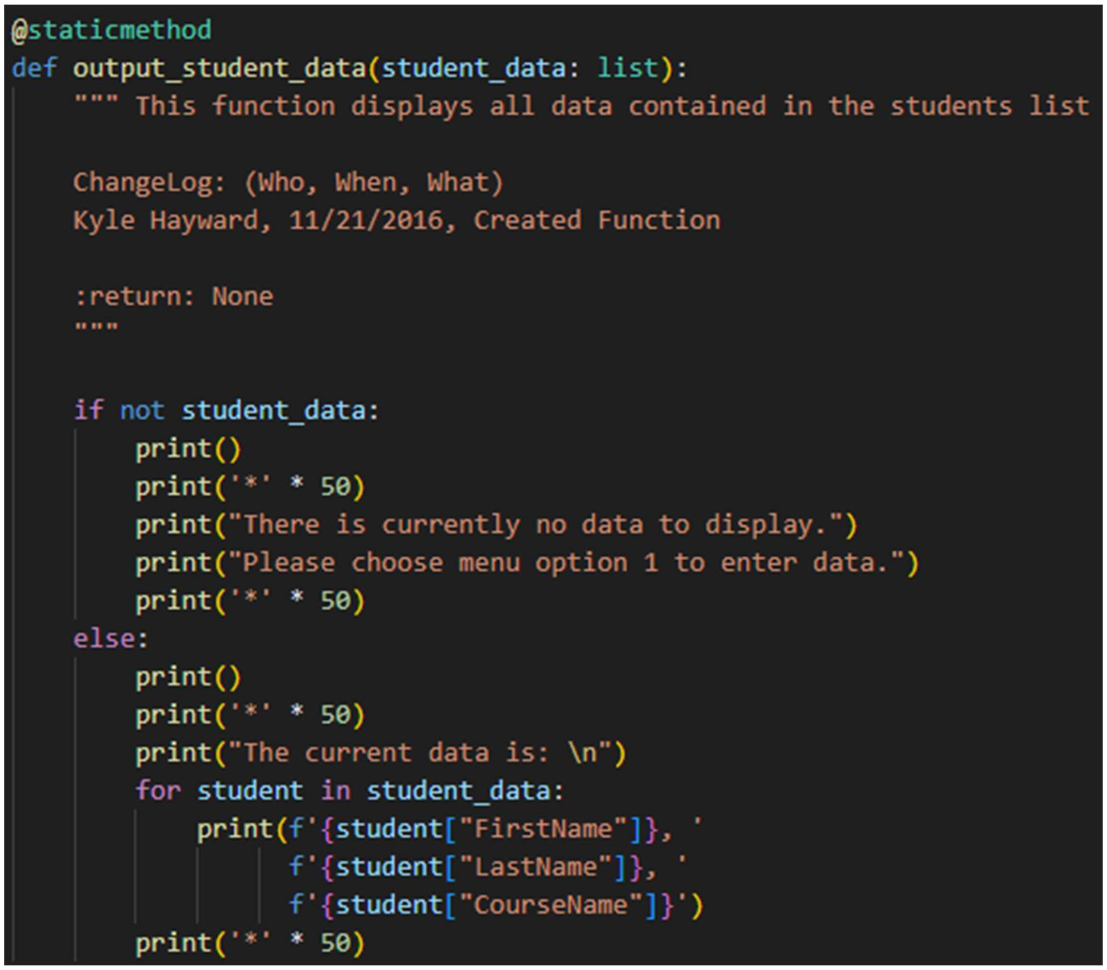
The third function under the IO class is the input_menu_choice function. This function receives the user's menu selection and raises an exception if any other option other than 1, 2, 3, or 4 is chosen. (Figure 1.11)



```python
@staticmethod
def input_menu_choice():
    """ This function receives the menu selection from the user

    ChangeLog: (Who, When, What)
    Kyle Hayward, 11/21/2023, Created Function

    :return: A string with the users menu selection
    """

    choice = "0"
    try:
        print('*' * 50)
        choice = input("Enter your menu selection: ")
        print('*' * 50)
        if choice not in ("1", "2", "3", "4"):
            raise Exception("Please choose menu option 1, 2, 3, or 4.")
    except Exception as e:
        IO.output_error_messages(e.__str__())

    return choice
```

Figure 1.11: Defining the function that receives the menu selection from the user.

The fourth function under the IO class is the output_student_data function which contains the student_data parameter. This function displays all the data that is currently stored in the student list. If there is no data in the list, it will prompt the user to select menu option 1 to enter new data. (Figure 1.12)



```python
@staticmethod
def output_student_data(student_data: list):
    """ This function displays all data contained in the students list

    ChangeLog: (Who, When, What)
    Kyle Hayward, 11/21/2016, Created Function

    :return: None
    """

    if not student_data:
        print()
        print('*' * 50)
        print("There is currently no data to display.")
        print("Please choose menu option 1 to enter data.")
        print('*' * 50)
    else:
        print()
        print('*' * 50)
        print("The current data is: \n")
        for student in student_data:
            print(f'{student["FirstName"]}, '
                  f'{student["LastName"]}, '
                  f'{student["CourseName"]}')
        print('*' * 50)
```

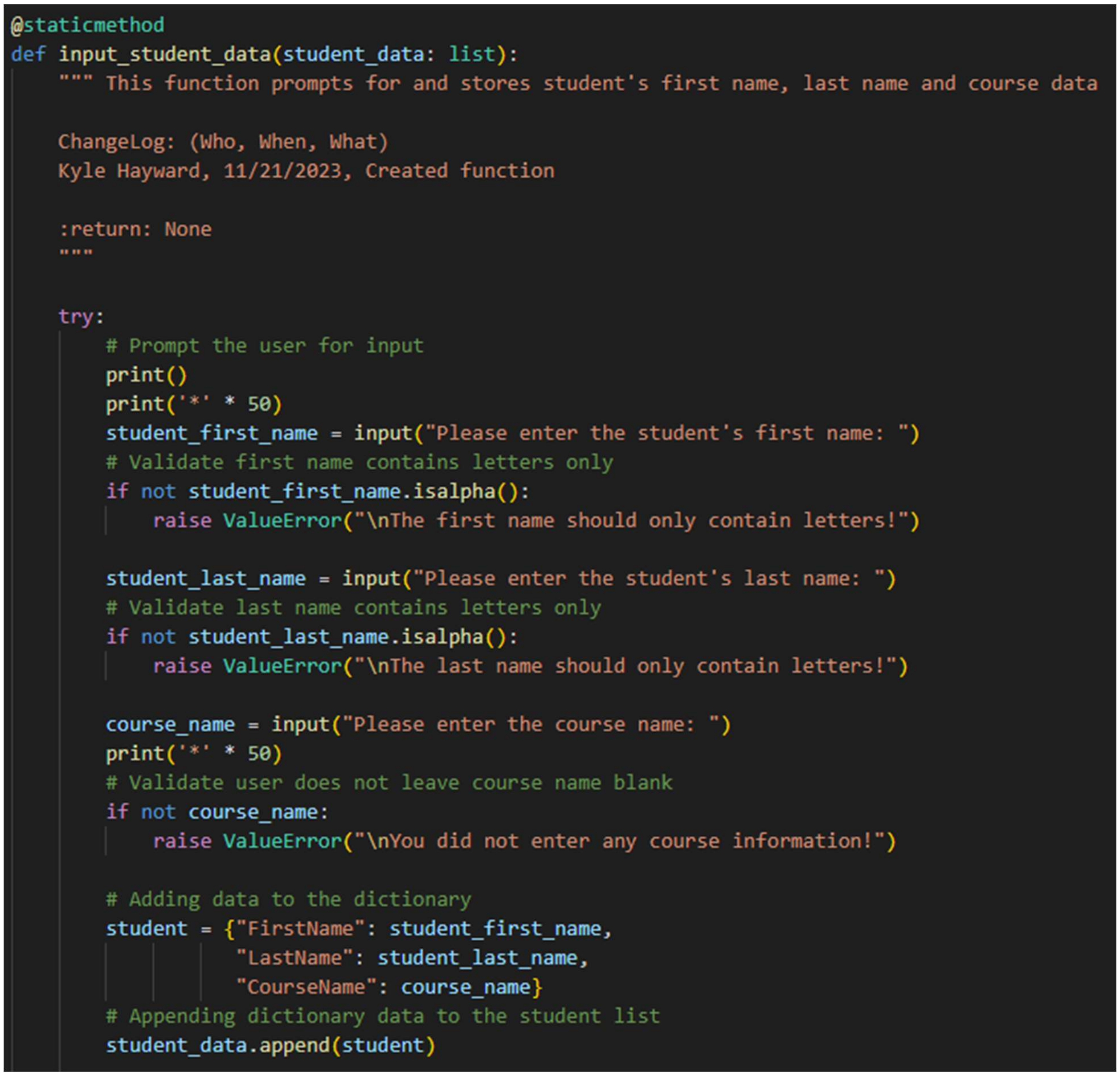
Figure 1.12: Defining the function to display all data contained in the student list.

The fifth and final function under the IO class is the input_student_data function containing one parameter. This function prompts the user to enter the student's first name, the student's last name, and the course to be registered.

Multiple layers of exception handling are implemented here including the isalpha function which ensures the user enters only letters for the first and last names. An exception will be raised if anything other than letters are entered.

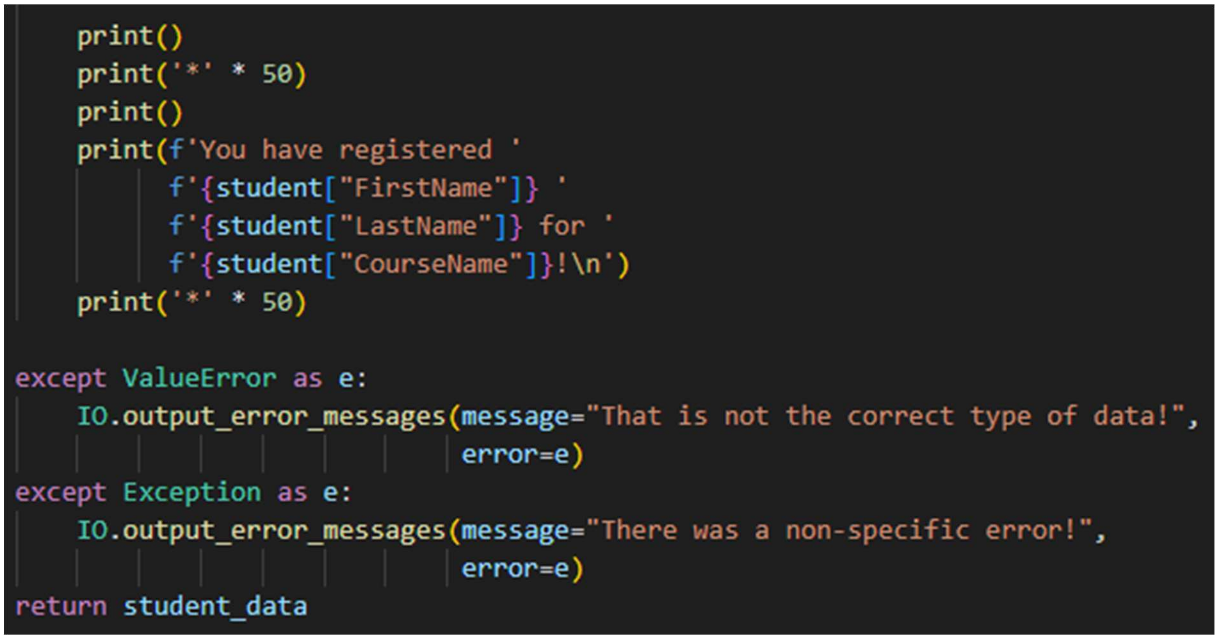
This function also adds the data entered by the user into the students list and displays the data back to the user.

Figures 1.13 and 1.14 demonstrate the full input_student_data function block.



```python
@staticmethod
def input_student_data(student_data: list):
    """ This function prompts for and stores student's first name, last name and course data

    ChangeLog: (Who, When, What)
    Kyle Hayward, 11/21/2023, Created function

    :return: None
    """

    try:
        # Prompt the user for input
        print()
        print('*' * 50)
        student_first_name = input("Please enter the student's first name: ")
        # Validate first name contains letters only
        if not student_first_name.isalpha():
            raise ValueError("\nThe first name should only contain letters!")

        student_last_name = input("Please enter the student's last name: ")
        # Validate last name contains letters only
        if not student_last_name.isalpha():
            raise ValueError("\nThe last name should only contain letters!")

        course_name = input("Please enter the course name: ")
        print('*' * 50)
        # Validate user does not leave course name blank
        if not course_name:
            raise ValueError("\nYou did not enter any course information!")

        # Adding data to the dictionary
        student = {"FirstName": student_first_name,
                   "LastName": student_last_name,
                   "CourseName": course_name}
        # Appending dictionary data to the student list
        student_data.append(student)
```

Figure 1.13: Defining the function that prompts the user to enter data.

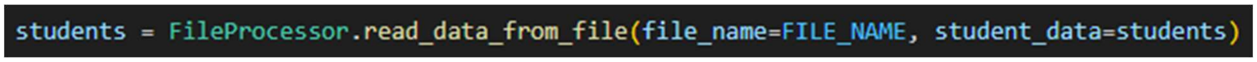


```
    print()
    print('*' * 50)
    print()
    print(f'You have registered '
          f'{student["FirstName"]} '
          f'{student["LastName"]} for '
          f'{student["CourseName"]}!\n')
    print('*' * 50)

except ValueError as e:
    IO.output_error_messages(message="That is not the correct type of data!",
                             error=e)
except Exception as e:
    IO.output_error_messages(message="There was a non-specific error!",
                             error=e)
return student_data
```

Figure 1.14: Completing the input_student_data function.

All classes and functions have now been defined which concludes the processing and presentation layers of the program. It is now time to complete the program by adding the main body of the script.

I accomplish this by having the students variable call the read_data_from_file function within the FileProcessor class. I then set the arguments as FILE_NAME and students. FILE_NAME is the “Enrollments.json” file and students in the list that contains all the data. (Figure 1.15)



```
students = FileProcessor.read_data_from_file(file_name=FILE_NAME, student_data=students)
```

Figure 1.15: Calling the read_data_from_file function with FILE_NAME and student as arguments.

The program then starts a while loop to print the Course Registration Menu, receive the menu choice and execute a particular function based on which menu choice was selected.

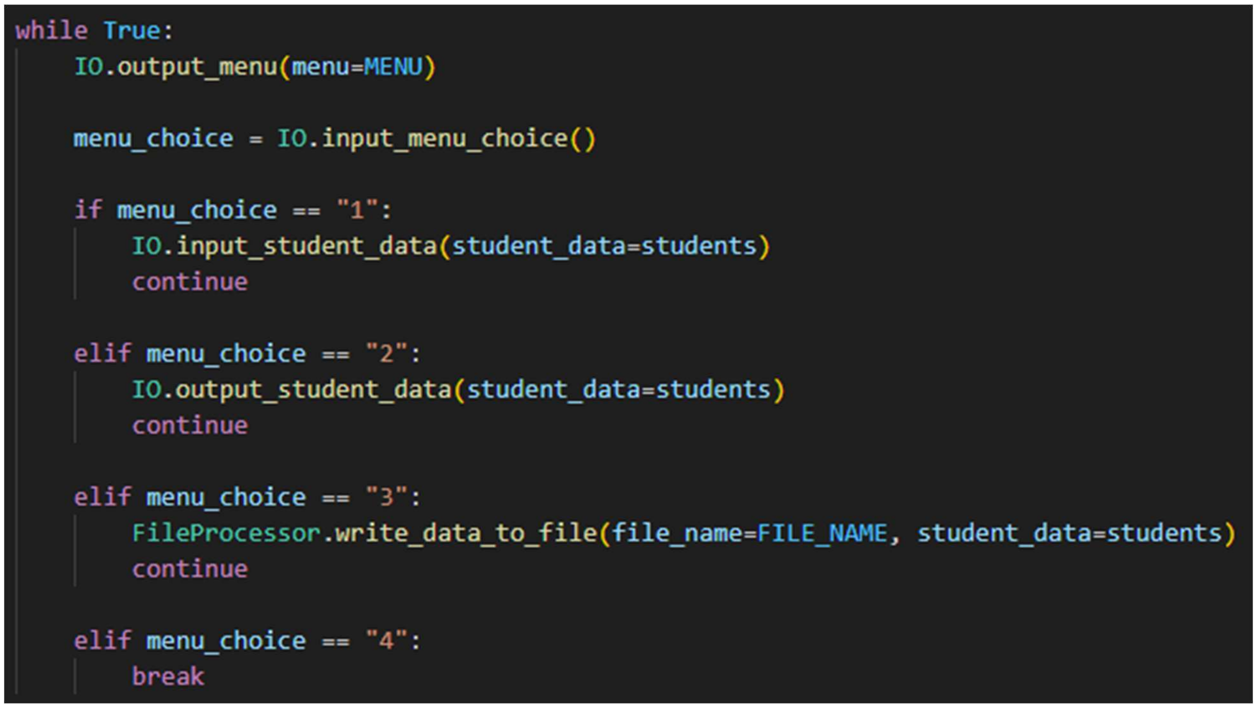
If menu choice 1 is selected, the input_student_data function is called with students as the argument.

If menu choice 2 is selected, the output_student_data function is called with students as the argument as well.

If menu choice 3 is selected, the write_data_to_file function is called with FILE_NAME and students as the arguments.

Finally, if menu choice 4 is selected, break is called effectively ending the program.

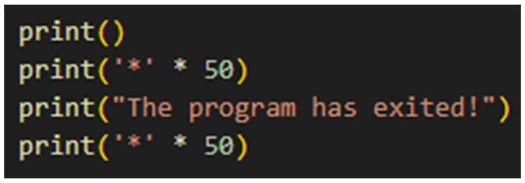
Figure 1.16 demonstrates the while loop within the main body of the script.



```
while True:
    IO.output_menu(menu=MENU)

    menu_choice = IO.input_menu_choice()

    if menu_choice == "1":
        IO.input_student_data(student_data=students)
        continue

    elif menu_choice == "2":
        IO.output_student_data(student_data=students)
        continue

    elif menu_choice == "3":
        FileProcessor.write_data_to_file(file_name=FILE_NAME, student_data=students)
        continue

    elif menu_choice == "4":
        break
```

Figure 1.16: The while loop which prints the menu, receives the user's choice and calls the functions.

I end the program with a simply print statement notifying the user that the program has ended. (Figure 1.17)



```
print()
print('*' * 50)
print("The program has exited!")
print('*' * 50)
```

Figure 1.17: Notifying the user that the program has ended.

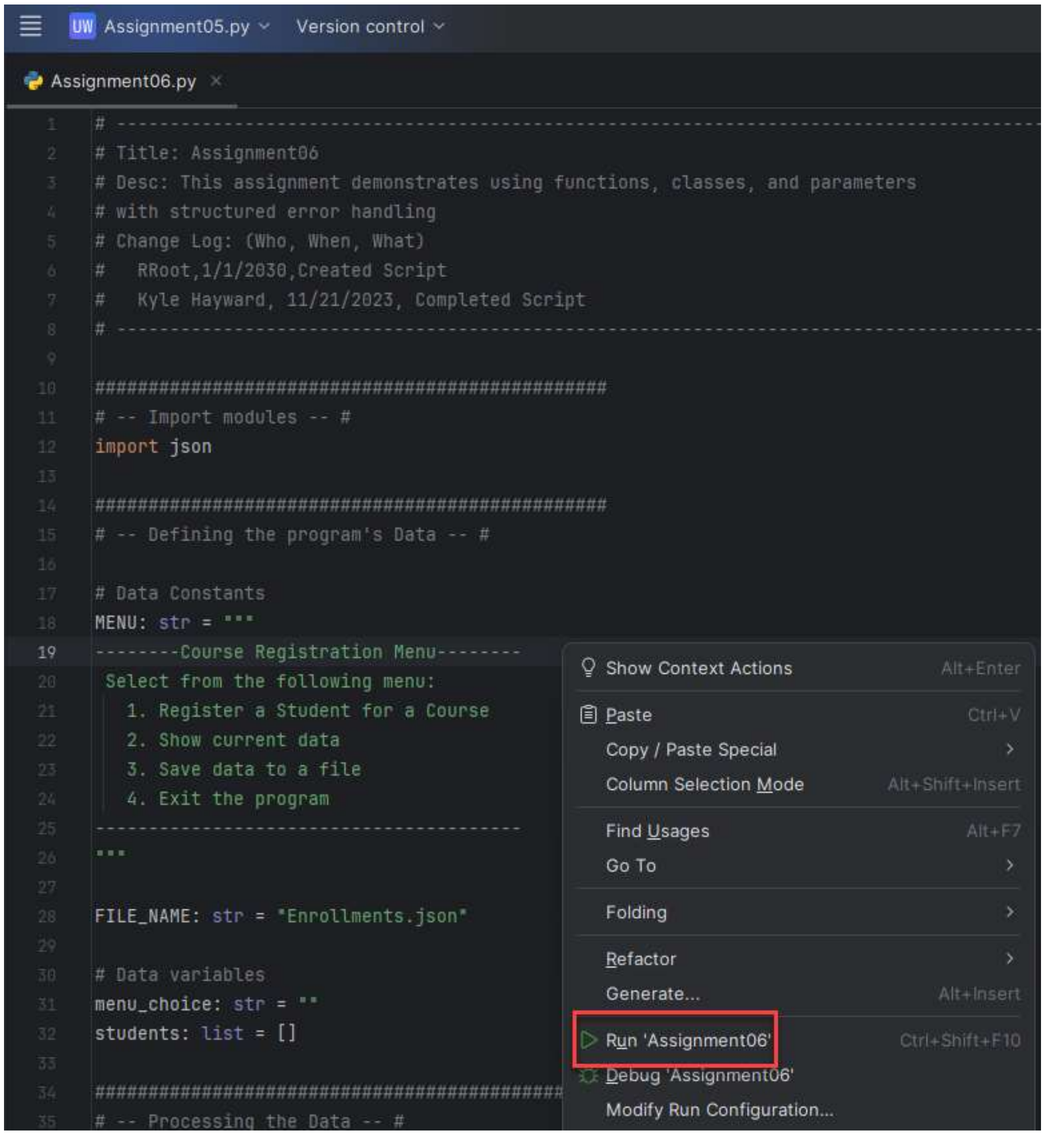
The next sections will demonstrate launching the program in the PyCharm IDE followed by running the program through the Windows OS Command Prompt.

Launching the Program – PyCharm

Now that the program code is written, we'll need to validate its functionality.

To launch the program within the PyCharm IDE, first launch PyCharm and open the Assignment06.py program.

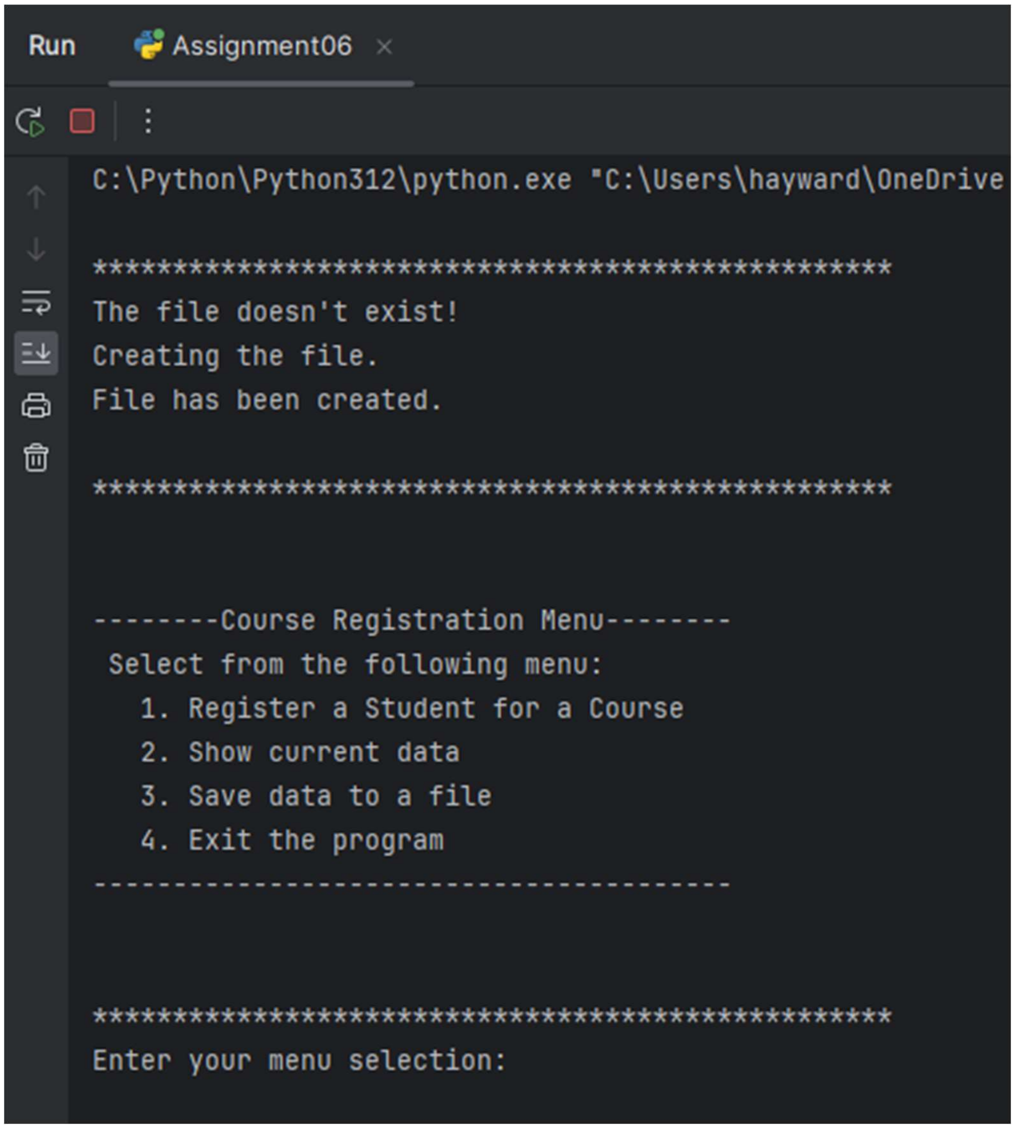
Right-click anywhere within the program code window and click **Run 'Assignment06'.** (Figure 2.1)



```
# ------------------------------------------------------------------------------------------
# Title: Assignment06
# Desc: This assignment demonstrates using functions, classes, and parameters
# with structured error handling
# Change Log: (Who, When, What)
#   RRoot,1/1/2030,Created Script
#   Kyle Hayward, 11/21/2023, Completed Script
# ------------------------------------------------------------------------------------------

################################################
# -- Import modules -- #
import json

################################################
# -- Defining the program's Data -- #

# Data Constants
MENU: str = """
--------Course Registration Menu--------
 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"

# Data variables
menu_choice: str = ""
students: list = []

###############################################
# -- Processing the Data -- #
```

Figure 2.1: Running the 'Assignment06' program within the PyCharm IDE.

The program successfully run within the PyCharm IDE as shown in Figure 2.2.



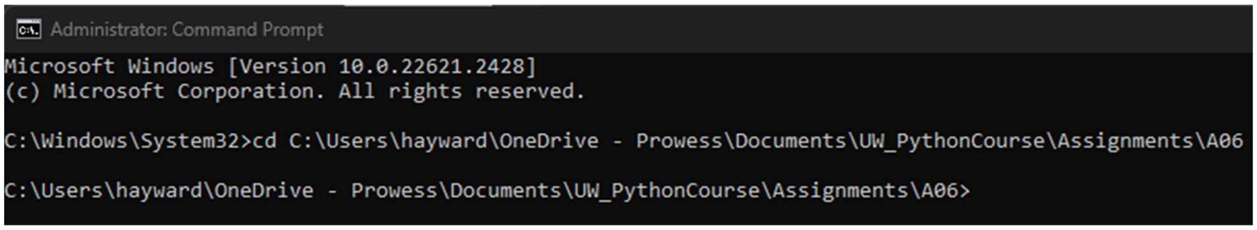
```
C:\Python\Python312\python.exe "C:\Users\hayward\OneDrive

**************************************************
The file doesn't exist!
Creating the file.
File has been created.

**************************************************



--------Course Registration Menu--------
 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:
```

Figure 2.2: The 'Assignment06' program running in the PyCharm IDE.

Now that we have validated the program successfully launches in PyCharm, we will run the program to completion through the Windows OS Command Prompt.

Running the Program – Command Prompt

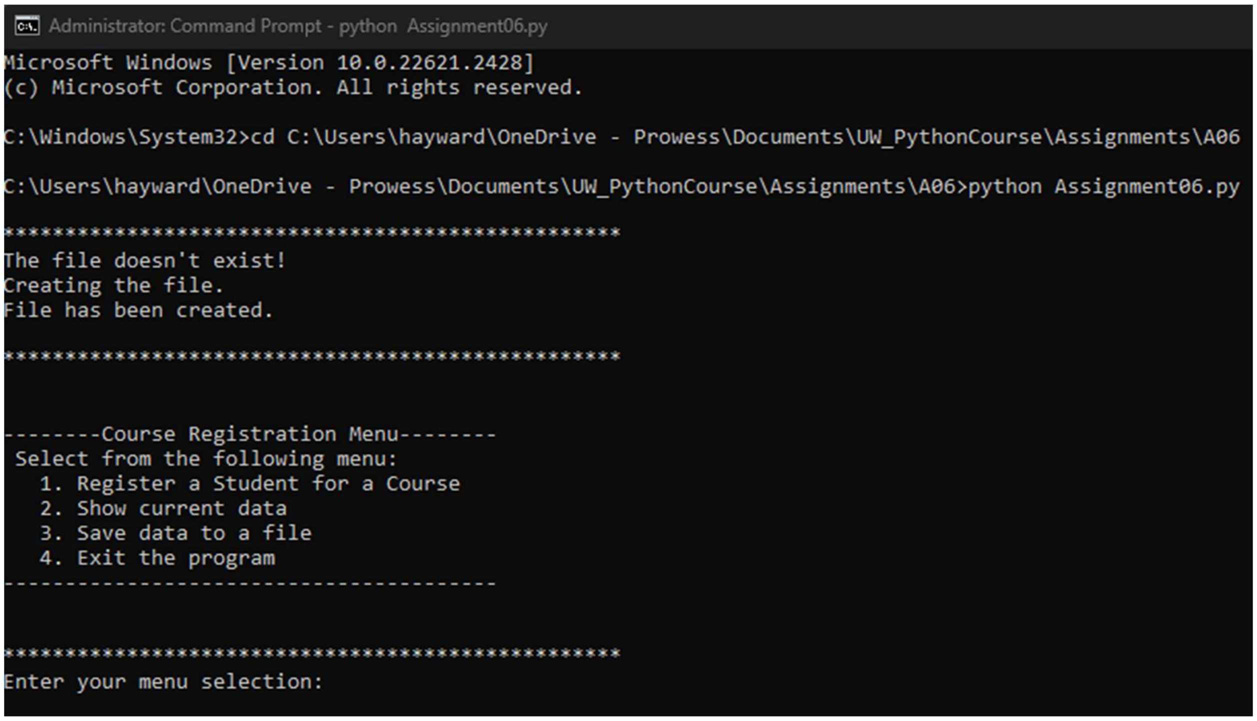
Launch the Command Prompt and navigate to the directory where the 'Assignment06.py' program resides. (Figure 2.3)



```
Microsoft Windows [Version 10.0.22621.2428]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32>cd C:\Users\hayward\OneDrive - Prowess\Documents\UW_PythonCourse\Assignments\A06

C:\Users\hayward\OneDrive - Prowess\Documents\UW_PythonCourse\Assignments\A06>
```

Figure 2.3: Navigating to the directory where the Assignment06.py program is located.

Type ‘python Assignment06.py’ and press Enter on the keyboard to start the program. (Figure 2.4)

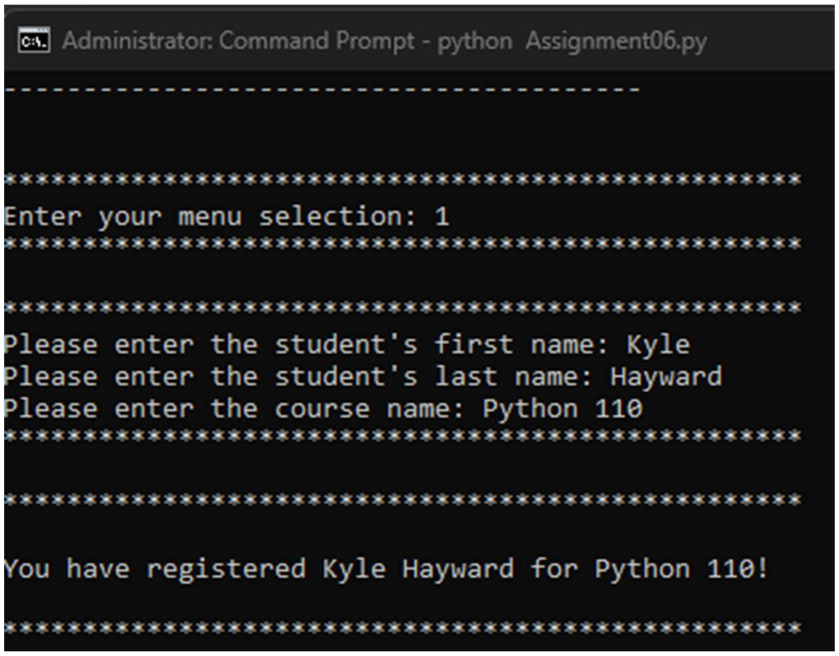


```
Administrator: Command Prompt - python  Assignment06.py
Microsoft Windows [Version 10.0.22621.2428]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32>cd C:\Users\hayward\OneDrive - Prowess\Documents\UW_PythonCourse\Assignments\A06

C:\Users\hayward\OneDrive - Prowess\Documents\UW_PythonCourse\Assignments\A06>python Assignment06.py

**************************************************
The file doesn't exist!
Creating the file.
File has been created.

**************************************************



--------Course Registration Menu--------
 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:
```

Figure 2.4: Launching the program within the Command Prompt.

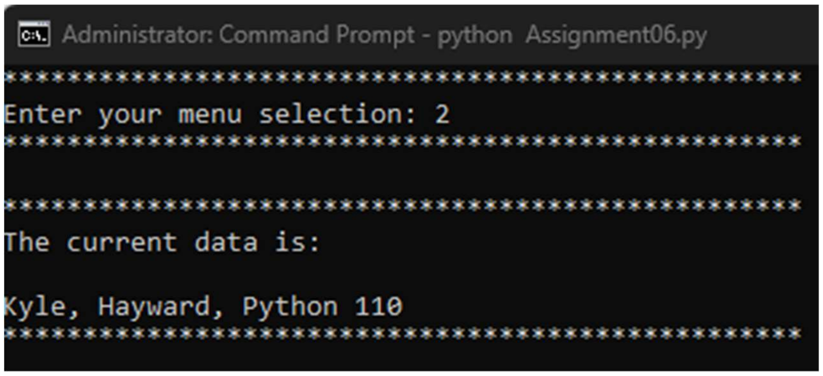
In Figure 2.4, notice how the program notifies the user that the file doesn’t exist. It then creates the file and notifies the user of such.

Figures 2.5 to 2.8 will demonstrate the program running to completion with all menu options being selected in succession.



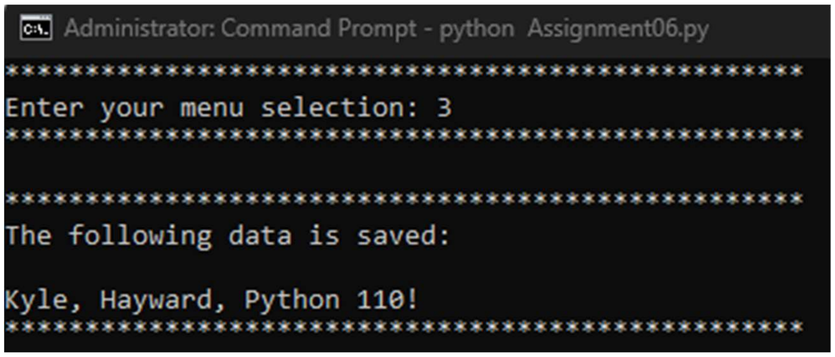
```
Administrator: Command Prompt - python  Assignment06.py
----------------------------------------


**************************************************
Enter your menu selection: 1
**************************************************

**************************************************
Please enter the student's first name: Kyle
Please enter the student's last name: Hayward
Please enter the course name: Python 110
**************************************************

**************************************************


You have registered Kyle Hayward for Python 110!

**************************************************
```

Figure 2.5: Program execution when menu option 1 is selected.



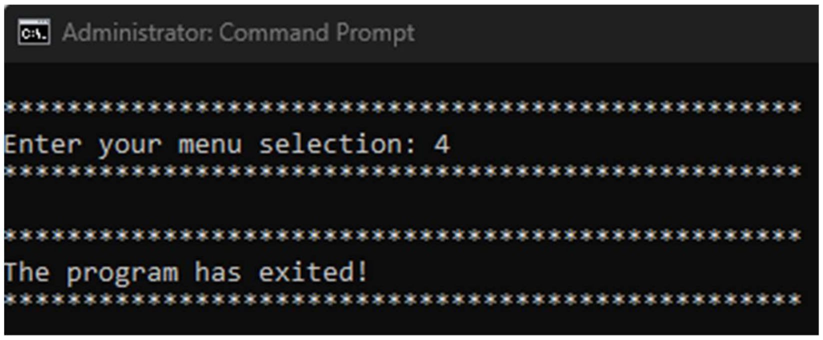
```
Administrator: Command Prompt - python  Assignment06.py
**************************************************
Enter your menu selection: 2
**************************************************

**************************************************
The current data is:

Kyle, Hayward, Python 110
**************************************************
```

Figure 2.6: Program execution when menu option 2 is selected.



```
Administrator: Command Prompt - python  Assignment06.py
**************************************************
Enter your menu selection: 3
**************************************************

**************************************************
The following data is saved:

Kyle, Hayward, Python 110!
**************************************************
```

Figure 2.7: Program execution when menu option 3 is selected.



```
Administrator: Command Prompt

**************************************************
Enter your menu selection: 4
**************************************************

**************************************************
The program has exited!
**************************************************
```

Figure 2.8: Program execution when menu option 4 is selected.

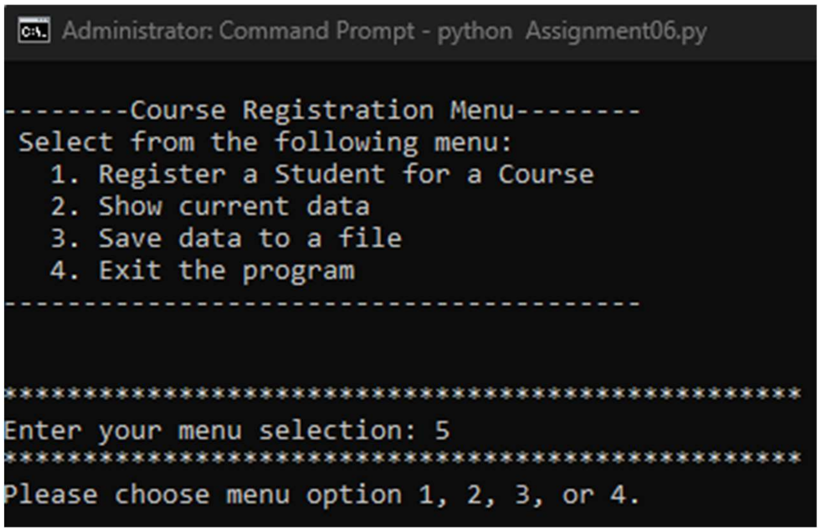
The program has now run to completion and exited gracefully.

In the next section, I will cover some exception handling. For instance, what happens when an invalid menu selection is chosen or if the user enters a name containing numbers?

Validate Exception Handling

Now that we know the program will run if all data entered is of expected type and value. But what happens when a user enters data that we are not expecting? Our program's exception handling should catch such occurrences and continue the program without negatively affecting user experience.

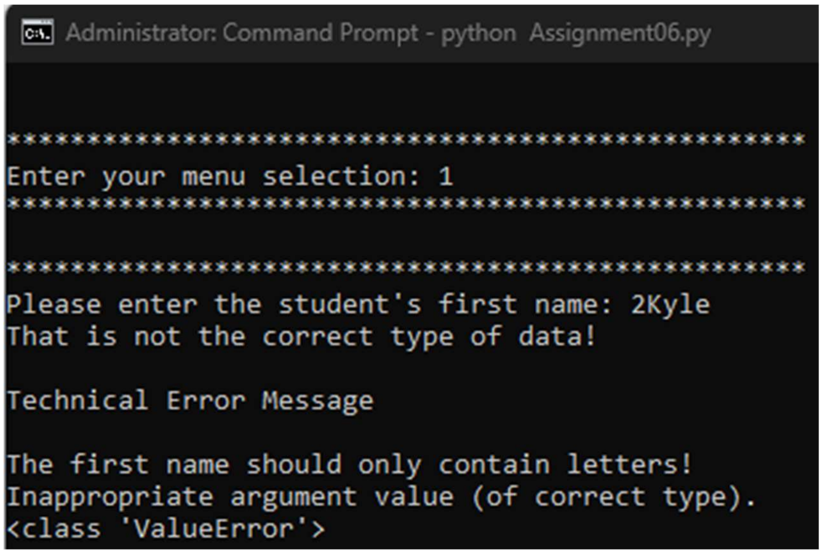
Suppose the user selects menu option 5, which is not a valid option. How does the program handle this? Figure 3.1 will demonstrate the error handling that I have implemented to address this.



```
Administrator: Command Prompt - python  Assignment06.py

--------Course Registration Menu--------
 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: 5
**************************************************
Please choose menu option 1, 2, 3, or 4.
```

Figure 3.1: The program prompts the user to select a menu option 1, 2, 3, or 4.

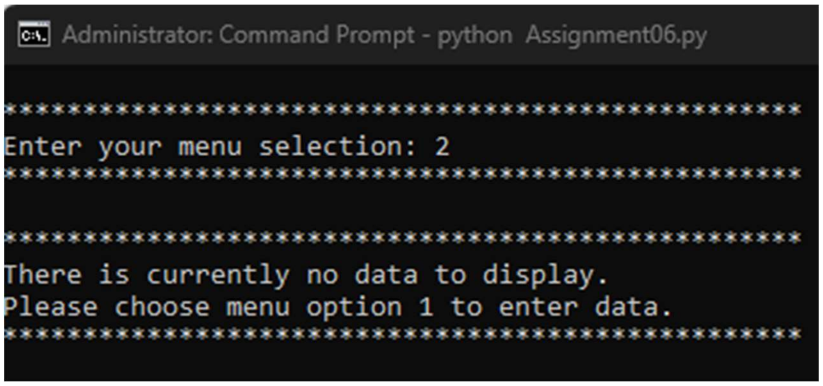
Another scenario could be a user entering invalid name data. For instance, a name containing a number. Figure 3.2 will show how the program handles this scenario.



```
Administrator: Command Prompt - python  Assignment06.py

**************************************************
Enter your menu selection: 1
**************************************************

**************************************************
Please enter the student's first name: 2Kyle
That is not the correct type of data!

Technical Error Message

The first name should only contain letters!
Inappropriate argument value (of correct type).
<class 'ValueError'>
```

Figure 3.2: An error is presented to the user if the input contains anything other than letters.

I have also implemented error handling for menu option 2. If there is no current data in the "Enrollments.json" file, I display a message to the user to choose menu option 1 in order to enter new data. (Figure 3.3)



```
Administrator: Command Prompt - python  Assignment06.py

**************************************************
Enter your menu selection: 2
**************************************************

**************************************************
There is currently no data to display.
Please choose menu option 1 to enter data.
**************************************************
```

Figure 3.3: Displaying a message to the user that there is no current data to display.

This concludes the exception handling portion of the program.

Summary

Module 06 covered the concepts of program organization by implementing classes. It also covered defining new functions and utilizing parameters and arguments within those functions.

I can see how powerful defining your own functions will be as my Python learning continues. The addition of parameters and arguments makes newly defined functions accessible not only throughout the program in which it was defined, but can be utilized in programs yet to be written.

As requested, this program's code has been uploaded to my GitHub repo:
https://github.com/HaywardUW/IntroToProg-Python-Mod06