James Francisco Jabal

20 Aug 2025

IT FDN 110 A Su 25

Assignment 06

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

Module 06 – Functions

Introduction

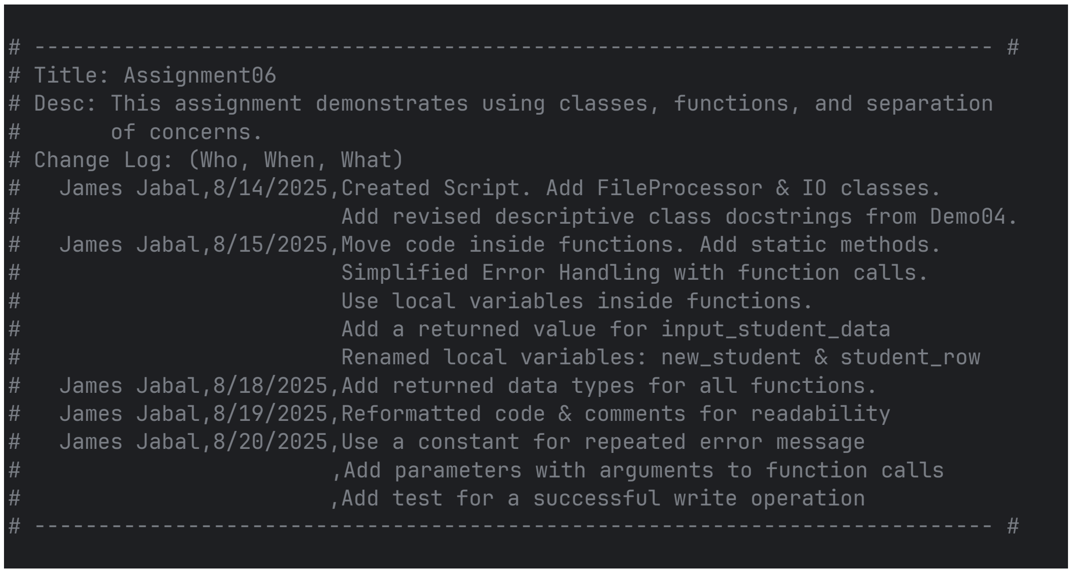
This assignment demonstrates using functions and classes by creating a Course Registration Program. The program would allow the user to read a JSON file of enrolled students, add students to the enrollment list, display the list, and save the list back to a JSON file. We continue our progression by modifying a starter program based on the previous assignment. This time, we will rewrite the program to use functions, classes, and the concept of "Separations of Concerns". We will also save the files on GitHub.

Program

This week, I continued my plan to have running working code as soon as possible before the office hours. I later revised the program based on feedback from the office hours and the course discussion board.

Header

I started with the header from the starter file. Eventually, I made several revisions to the code after the two office hours with the instructor assistants Chris Voit (8/15/2025) and Becky Peltz (8/18/2025), and documented changes in the header (Figure 1).



```
# ------------------------------------------------------------------------ #
# Title: Assignment06
# Desc: This assignment demonstrates using classes, functions, and separation
#       of concerns.
# Change Log: (Who, When, What)
#   James Jabal,8/14/2025,Created Script. Add FileProcessor & IO classes.
#                         Add revised descriptive class docstrings from Demo04.
#   James Jabal,8/15/2025,Move code inside functions. Add static methods.
#                         Simplified Error Handling with function calls.
#                         Use local variables inside functions.
#                         Add a returned value for input_student_data
#                         Renamed local variables: new_student & student_row
#   James Jabal,8/18/2025,Add returned data types for all functions.
#   James Jabal,8/19/2025,Reformatted code & comments for readability
#   James Jabal,8/20/2025,Use a constant for repeated error message
#                        ,Add parameters with arguments to function calls
#                        ,Add test for a successful write operation
# ------------------------------------------------------------------------ #
```

Figure 1. Header with a change log

Imports

I imported the `json` module to simplify working with the JSON file, and the `io` module for using the JSON file with text file wrapper functions (Figure 2).

```
import json
import io as _io
```

Figure 2. Imports

Constants

I defined the global constants: menu, JSON filename, and a commonly repeated error message (Figure 3). Using a constant for common error messages was implemented from feedback in previous assignments.

```
# Define the Data Constants
MENU: str = '''
---- Course Registration Program ----
  Select from the following menu:
    1. Register a Student for a Course.
    2. Show current data.
    3. Save data to a file.
    4. Exit the program.
-----------------------------------------
'''
# Define the Data Constants
FILE_NAME: str = "Enrollments.json"
NON_SPEC_ERROR_MESSAGE: str = "There was a non-specific error!"
```

Figure 3. Global constants

Variables

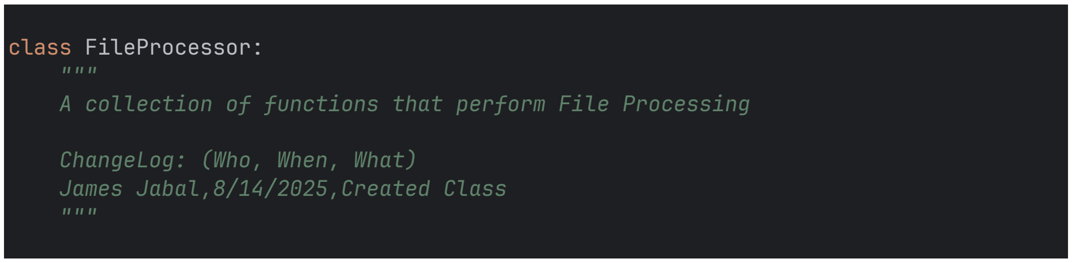
I reduced the number of global variables (Figure 4) and moved local variables to individual functions. Added a Boolean variable to track the status of a successful write operation.

```
# Define the Data Variables
students: list = []  # a table of student data
menu_choice: str = ''  # Hold the choice made by the user.
success: bool = False  # Holds success boolean of file write operation
```

Figure 4. Global variables

Class – File Processor

I created the File Processor class (Figure 5) to separate the concerns of processing the file. It includes two functions of reading data from a file, and writing data to the file. Also added document strings to both classes and functions to clarify how to use them.



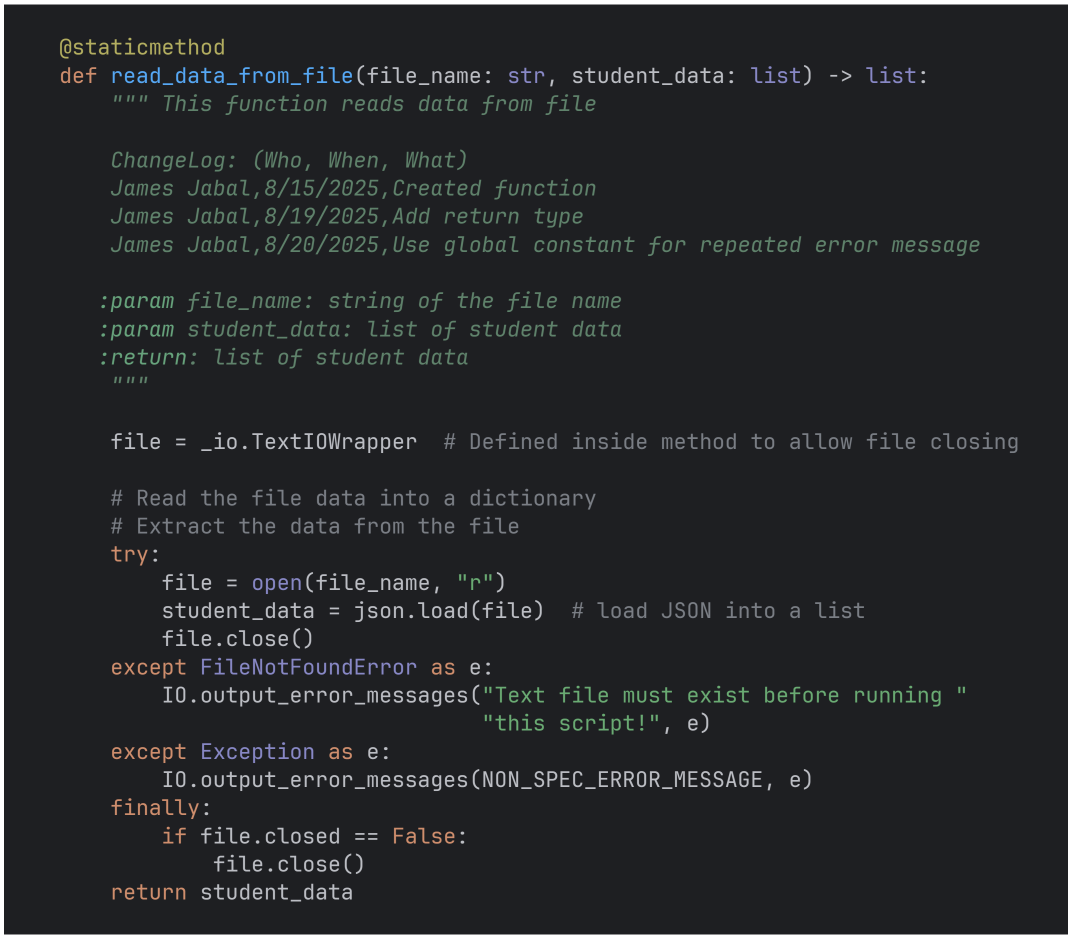
```python
class FileProcessor:
    """
    A collection of functions that perform File Processing

    ChangeLog: (Who, When, What)
    James Jabal,8/14/2025,Created Class
    """
```

Figure 5. File Processor Class

Function – Read Data from File

Within the File Processor class, I created the static method function for reading the data from a file (Figure 6). I included the parameters it would need (file name string & list) and the returned data (list), including the data types. The local variable `file` was initialized, and found necessary to avoid errors when attempting to close the file, and contained within this function.

The file is read with error handling capabilities. Two local variables are created to use the arguments passed to the parameters of this function. In the end, the function returns a list using the contents from the JSON file.



```python
    @staticmethod
    def read_data_from_file(file_name: str, student_data: list) -> list:
        """ This function reads data from file

        ChangeLog: (Who, When, What)
        James Jabal,8/15/2025,Created function
        James Jabal,8/19/2025,Add return type
        James Jabal,8/20/2025,Use global constant for repeated error message

       :param file_name: string of the file name
       :param student_data: list of student data
       :return: list of student data
        """

        file = _io.TextIOWrapper  # Defined inside method to allow file closing

        # Read the file data into a dictionary
        # Extract the data from the file
        try:
            file = open(file_name, "r")
            student_data = json.load(file)  # load JSON into a list
            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(NON_SPEC_ERROR_MESSAGE, e)
        finally:
            if file.closed == False:
                file.close()
        return student_data
```

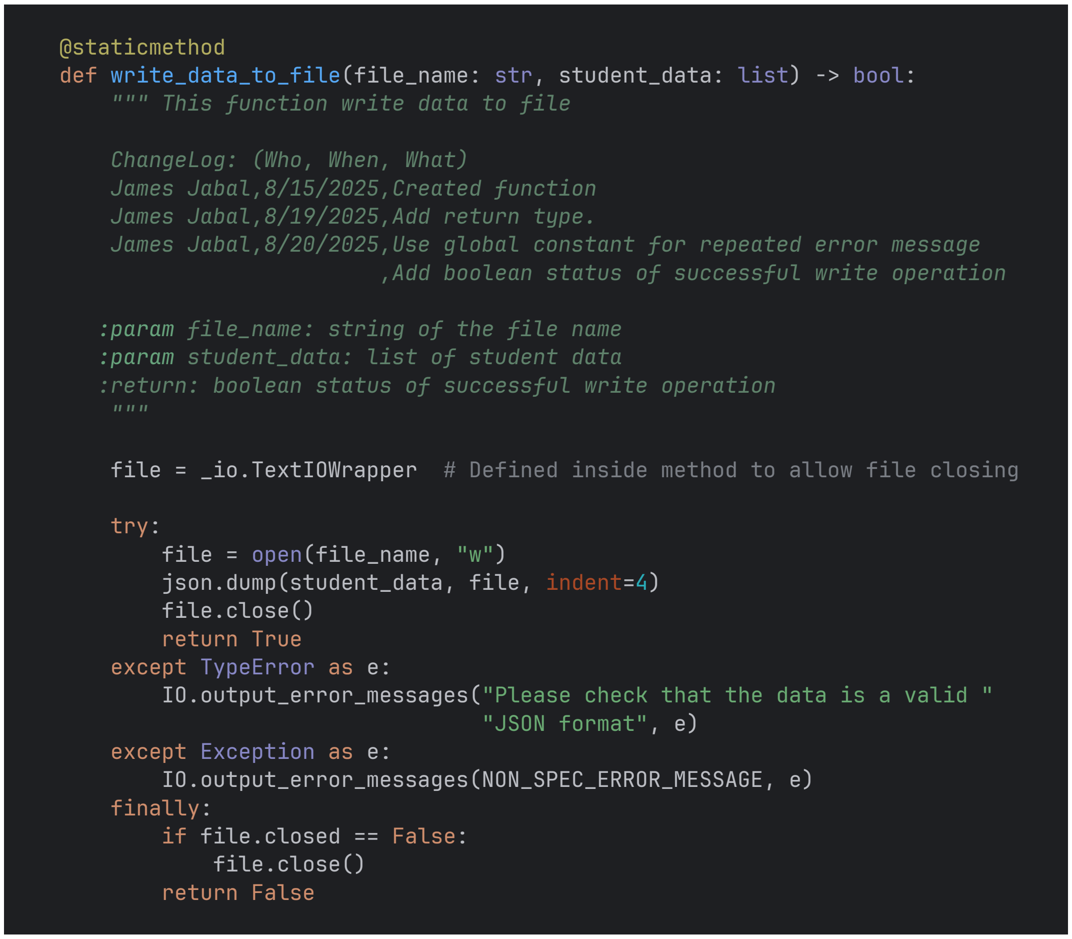
Figure 6. Static method function to read data from a file

Function – Write Data to File

Similarly, I created another static method function within this class for writing files, defining the required parameters, return values, and data types (Figure 7). Also added some error handling functionality.

For writing to a file, I added the argument of "4" to the `indent` parameter of the `json.dump` method. This is to format the JSON file for human readability.

In this case, a return value would be `True` after successfully writing to the file, and return to the program. If there was a write error, this function would return `False` (a successful write won't reach this code).

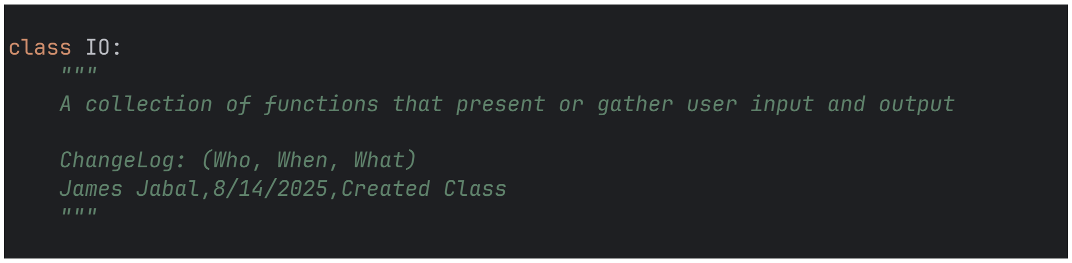


```python
@staticmethod
def write_data_to_file(file_name: str, student_data: list) -> bool:
    """ This function write data to file

    ChangeLog: (Who, When, What)
    James Jabal,8/15/2025,Created function
    James Jabal,8/19/2025,Add return type.
    James Jabal,8/20/2025,Use global constant for repeated error message
                         ,Add boolean status of successful write operation

   :param file_name: string of the file name
   :param student_data: list of student data
   :return: boolean status of successful write operation
    """

    file = _io.TextIOWrapper  # Defined inside method to allow file closing

    try:
        file = open(file_name, "w")
        json.dump(student_data, file, indent=4)
        file.close()
        return True
    except TypeError as e:
        IO.output_error_messages("Please check that the data is a valid "
                                 "JSON format", e)
    except Exception as e:
        IO.output_error_messages(NON_SPEC_ERROR_MESSAGE, e)
    finally:
        if file.closed == False:
            file.close()
        return False
```

Figure 7. Static method function to write data to a file

Class – Input & Output

I also created an Input & Output class (Figure 8) to separate the concerns for presentation. This includes error message output, menu output, menu choice input, student data (name) input, and student course output.



```python
class IO:
    """
    A collection of functions that present or gather user input and output

    ChangeLog: (Who, When, What)
    James Jabal,8/14/2025,Created Class
    """
```

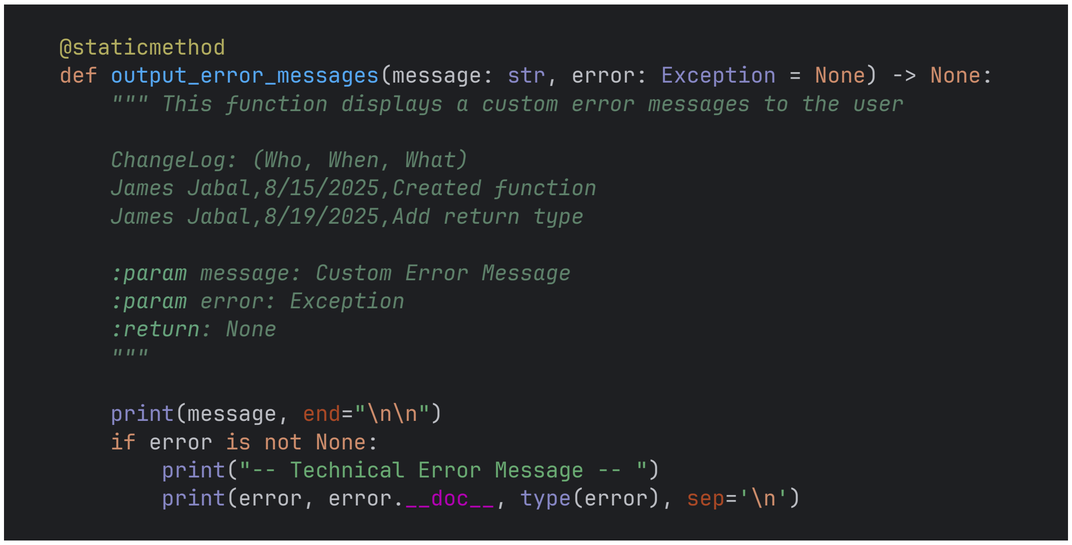
Figure 8. Input & Output Class

Function – Output Error Message

Within the Input & Output class, I created the static method function to output any error messages based on message and error arguments passed to this function (Figure 9).

It prints out custom error messages with a default error on the `None` data type. This is an example of an optional parameter to avoid the technical error message for `ValueError` exception handling.

There is no returned value at the end of the function. Only an error output message is displayed.

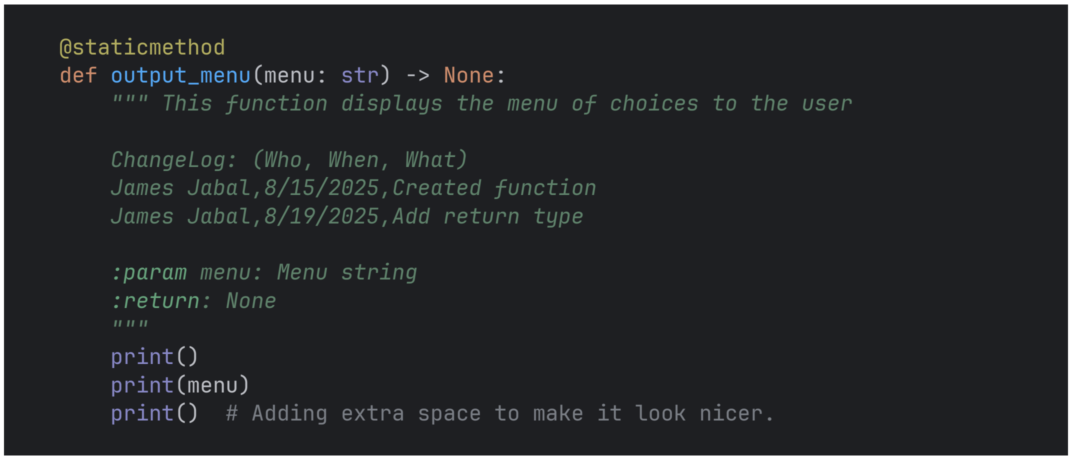


```python
@staticmethod
def output_error_messages(message: str, error: Exception = None) -> None:
    """ This function displays a custom error messages to the user

    ChangeLog: (Who, When, What)
    James Jabal,8/15/2025,Created function
    James Jabal,8/19/2025,Add return type

    :param message: Custom Error Message
    :param error: Exception
    :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 9. Static method function to display error messages

Function – Output Menu

The menu output is now also a separate static method function (Figure 10) that displays the string argument passed to the `menu` parameter. There is no returned value at the end of this function.



```python
@staticmethod
def output_menu(menu: str) -> None:
    """ This function displays the menu of choices to the user

    ChangeLog: (Who, When, What)
    James Jabal,8/15/2025,Created function
    James Jabal,8/19/2025,Add return type

    :param menu: Menu string
    :return: None
    """
    print()
    print(menu)
    print()  # Adding extra space to make it look nicer.
```

Figure 10. Static method function to display the menu

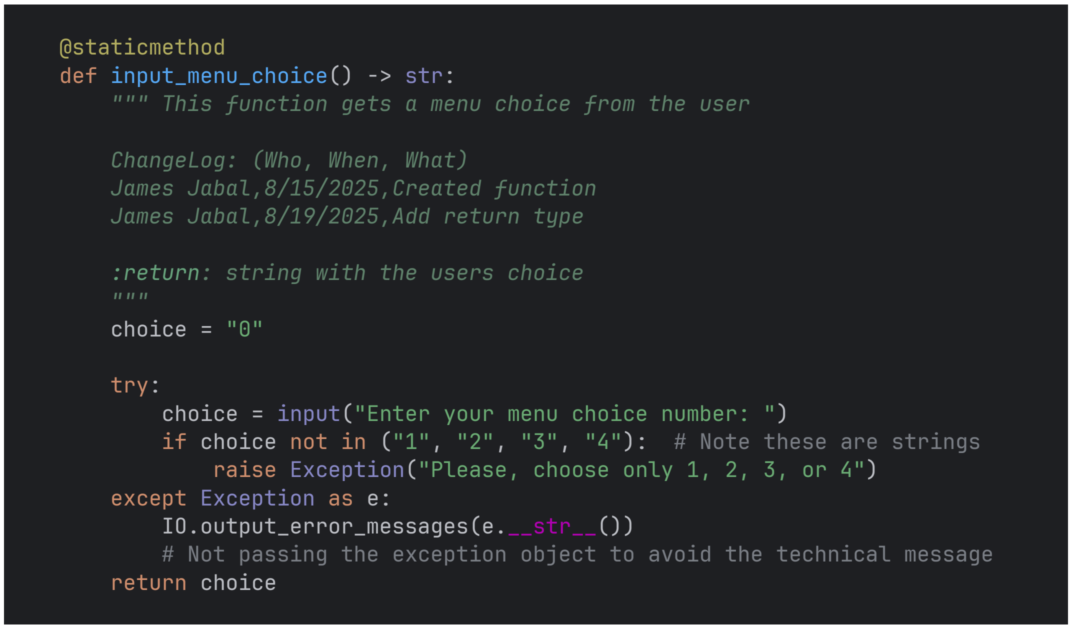
Function – Input Menu Choice

The input for the menu choice is now separated from the output with its own static method function (Figure 11). This is an example where there are no parameters required, and the return value is a string.

After initializing the local variable `choice` (not the global `menu_choice` variable), it starts with checking if the choice is not within a tuple of valid strings, with error handing for any invalid input.

This is also where we see an example of calling the static method function from within the same class to output a custom error message with only one argument being passed to avoid a technical message.

In the end of the function, it returns the string of the actual input.



```python
@staticmethod
def input_menu_choice() -> str:
    """ This function gets a menu choice from the user

    ChangeLog: (Who, When, What)
    James Jabal,8/15/2025,Created function
    James Jabal,8/19/2025,Add return type

    :return: string with the users choice
    """
    choice = "0"

    try:
        choice = input("Enter your menu choice number: ")
        if choice not in ("1", "2", "3", "4"):  # Note these are strings
            raise Exception("Please, choose only 1, 2, 3, or 4")
    except Exception as e:
        IO.output_error_messages(e.__str__())
        # Not passing the exception object to avoid the technical message
    return choice
```

Figure 11. Static method function for menu choice input

Function – Input Student Data

The input for the student data is also a separated function which accepts a list, and returns a modified list with the added student (Figure 12).

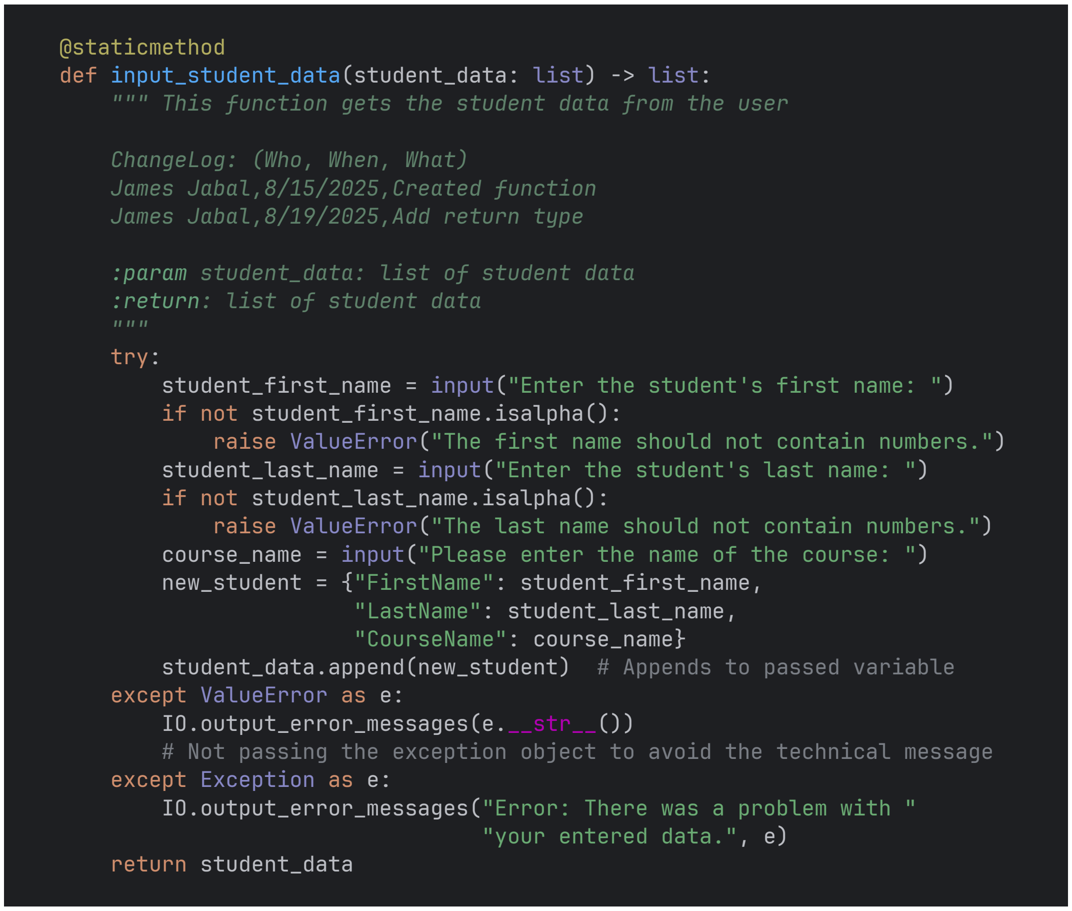
It goes through prompts for the user to input the student's first & last names with error handling if there are any numbers in the name. Then a prompt asks for the course name.

Here, I had to create a dictionary row of data. After some feedback from the office hours, it made sense to create a more descriptive local variable `new_student`. This dictionary row was then appended to the list (passed to this function).

This caused some confusion while debugging, because since the list is appended, the passed list itself is being modified. I tried 3 different versions testing references and mutability of lists. I received feedback from the instructor, Randal Root, on the discussion board, clarifying that none of the versions are wrong. However, it was recommended & made the most sense to modify the list in-place, and return the modified list too. My understanding is that passing a list to a function for adding students implies it will be modified, and returning the modified list makes that change more explicit. This concept is also aligned with PEP 20, "Explicit is better than implicit".

The error handling includes calls to the custom class and error message function using only one argument for a value error, and two arguments for other exceptions.

In the end, the modified list is returned, which references the same contents of the modified passed list.



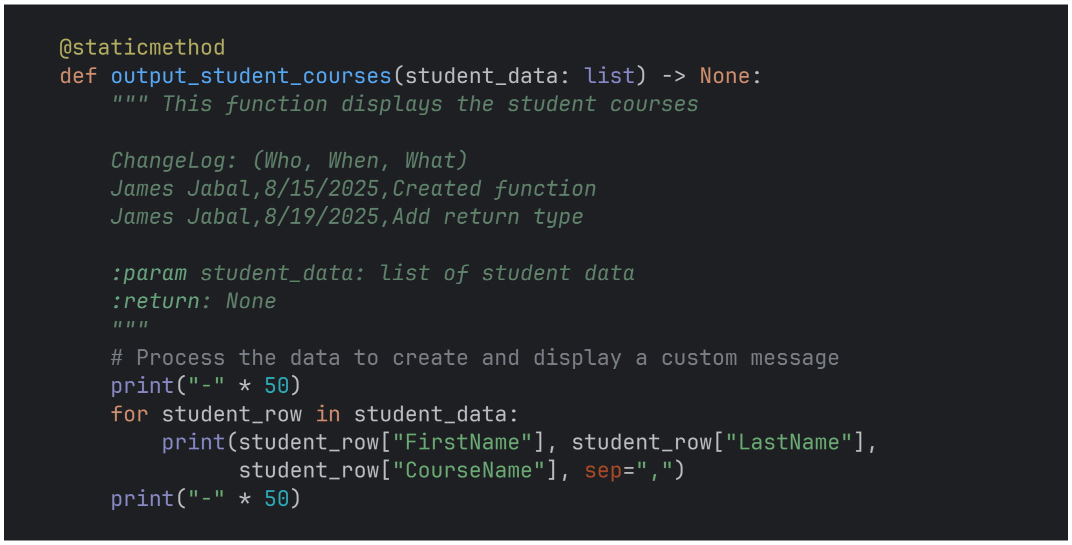
```python
@staticmethod
def input_student_data(student_data: list) -> list:
    """ This function gets the student data from the user

    ChangeLog: (Who, When, What)
    James Jabal,8/15/2025,Created function
    James Jabal,8/19/2025,Add return type

    :param student_data: list of student data
    :return: list of student data
    """
    try:
        student_first_name = input("Enter the student's first name: ")
        if not student_first_name.isalpha():
            raise ValueError("The first name should not contain numbers.")
        student_last_name = input("Enter the student's last name: ")
        if not student_last_name.isalpha():
            raise ValueError("The last name should not contain numbers.")
        course_name = input("Please enter the name of the course: ")
        new_student = {"FirstName": student_first_name,
                       "LastName": student_last_name,
                       "CourseName": course_name}
        student_data.append(new_student)  # Appends to passed variable
    except ValueError as e:
        IO.output_error_messages(e.__str__())
        # Not passing the exception object to avoid the technical message
    except Exception as e:
        IO.output_error_messages("Error: There was a problem with "
                                 "your entered data.", e)
    return student_data
```

Figure 12. Static method function for student data input

Function – Output Student Courses

The output for the student courses was separated into another static method function (Figure 13). It accepts the passed list, and iterates through each row. I implemented feedback to rename the iterator variable for clarity. It displays a CSV string for each row, by passing a "," as an argument to the `sep` parameter for the `print()` function. There is no returned value.

This is also the end of the classes & functions section.



```python
@staticmethod
def output_student_courses(student_data: list) -> None:
    """ This function displays the student courses

    ChangeLog: (Who, When, What)
    James Jabal,8/15/2025,Created function
    James Jabal,8/19/2025,Add return type

    :param student_data: list of student data
    :return: None
    """
    # Process the data to create and display a custom message
    print("-" * 50)
    for student_row in student_data:
        print(student_row["FirstName"], student_row["LastName"],
              student_row["CourseName"], sep=",")
    print("-" * 50)
```

Figure 13. Static method function to display the student course

Main Code

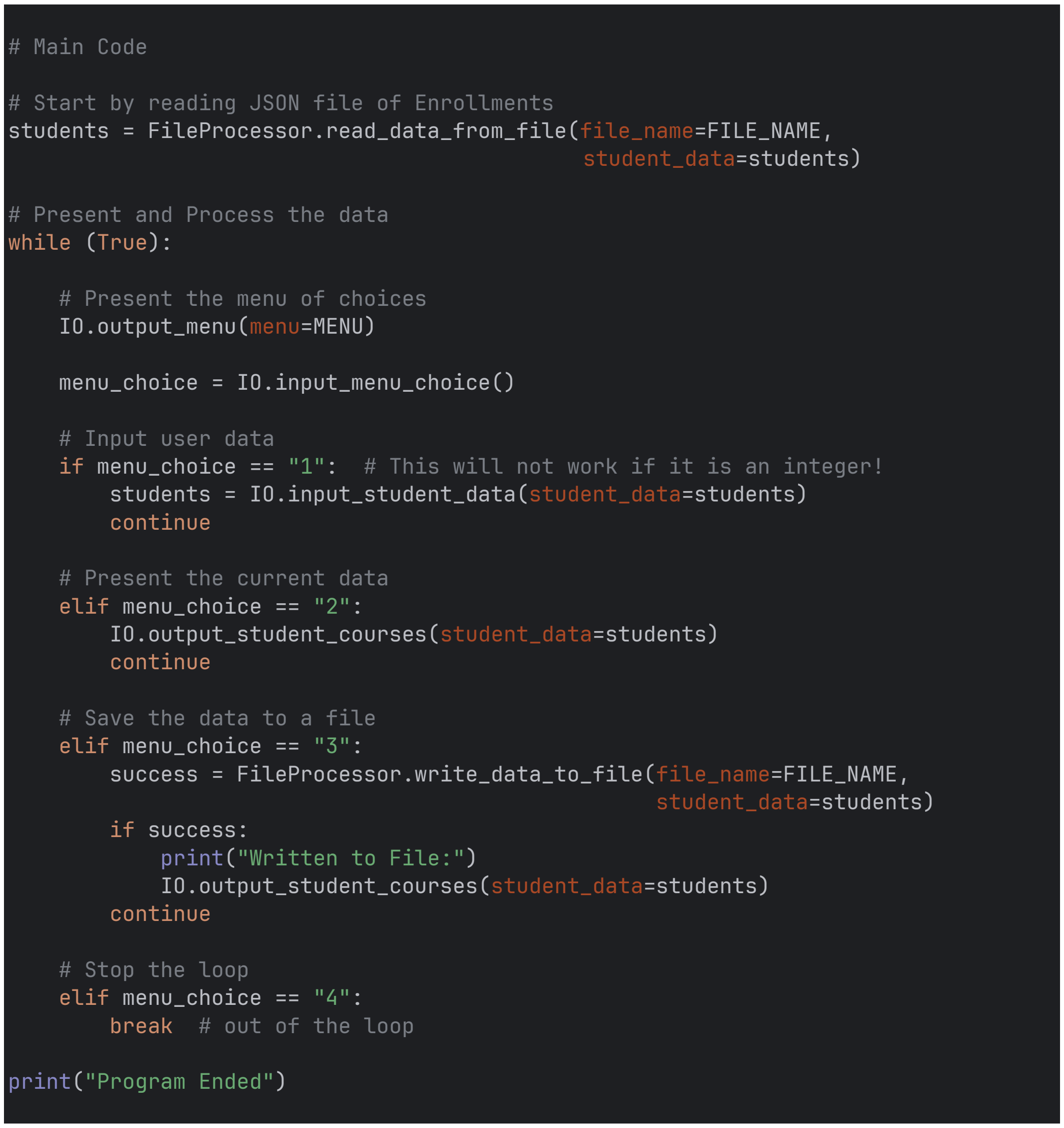
The main code is now simplified with the use of classes, functions, and the separation of concerns (Figure 14). The program now calls functions from each class, passing arguments, and capturing the returned values (if necessary).

It starts with reading the file to update the empty `students` list. Then it goes through the menu loop, displaying the menu, and requesting an input choice.

For each choice, the program calls each specific function to prompt for user data input, present the current data, save the data to a file, and to exit the program.

The choice to save the data includes two functions which separates the concerns for writing to the file (processing) and displaying the student courses (presentation), in two separate classes.

After testing the error handling, I found it necessary to test for a successful write operation. If successful, it would display what was written. If not, the exception handler will display the error message, and return `False`. This would prevent displaying the courses that may erroneously suggest the file data was written.



```python
# Main Code

# Start by reading JSON file of Enrollments
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 user data
    if menu_choice == "1":  # This will not work if it is an integer!
        students = IO.input_student_data(student_data=students)
        continue

    # Present the current data
    elif menu_choice == "2":
        IO.output_student_courses(student_data=students)
        continue

    # Save the data to a file
    elif menu_choice == "3":
        success = FileProcessor.write_data_to_file(file_name=FILE_NAME,
                                                   student_data=students)
        if success:
            print("Written to File:")
            IO.output_student_courses(student_data=students)
        continue

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

print("Program Ended")
```

Figure 14. Main Program

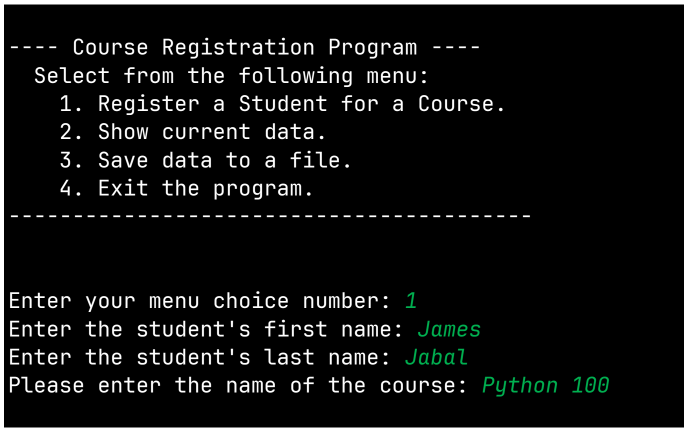
Testing

I tested the program in both PyCharm & Terminal, then checked PyCharm & Terminal for the data in the saved file "Enrollments.json". Testing made sure the program meets the acceptance criteria:

- Takes user input for student registration
- Displays data
- Saves data (Check file in PyCharm & Terminal)
- Allows multiple registrations
- Allows displaying multiple registrations
- Allows saving multiple registrations (Check file in PyCharm & Terminal)
- Runs on both PyCharm & Terminal

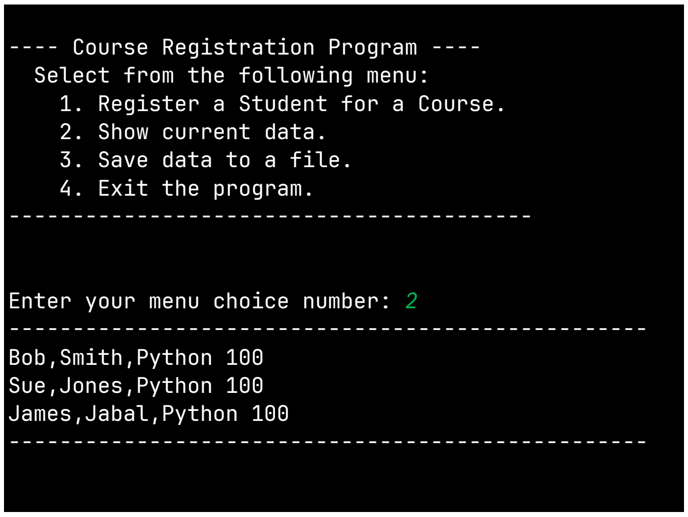
PyCharm

The program was tested in PyCharm using the starter "Enrollments.json", by adding one student to the enrollment list (Figure 15), displaying the enrollment list (Figure 16), saving the list & displaying contents written (Figure 17), then exiting the program (Figure 18).



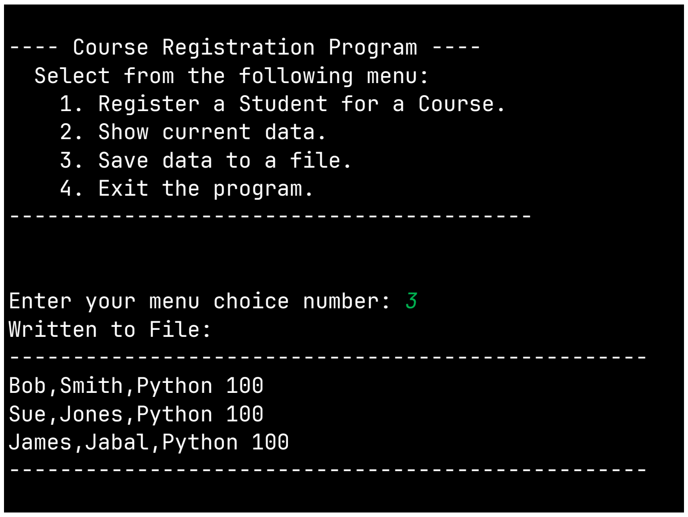
```
---- Course Registration Program ----
  Select from the following menu:
    1. Register a Student for a Course.
    2. Show current data.
    3. Save data to a file.
    4. Exit the program.
-----------------------------------------


Enter your menu choice number: 1
Enter the student's first name: James
Enter the student's last name: Jabal
Please enter the name of the course: Python 100
```

Figure 15. Add Data



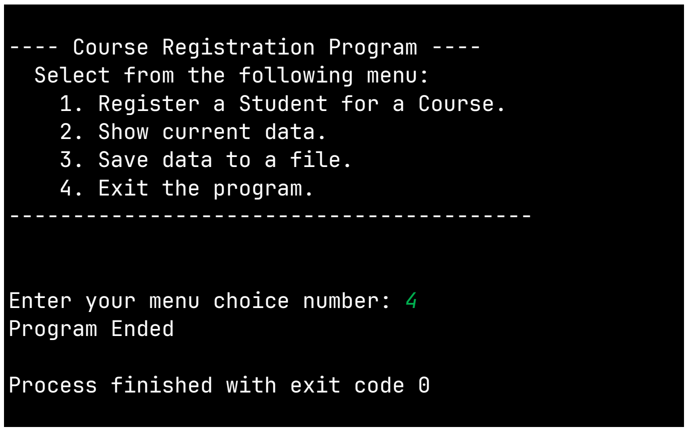
```
---- Course Registration Program ----
  Select from the following menu:
    1. Register a Student for a Course.
    2. Show current data.
    3. Save data to a file.
    4. Exit the program.
-----------------------------------------


Enter your menu choice number: 2
--------------------------------------------------
Bob,Smith,Python 100
Sue,Jones,Python 100
James,Jabal,Python 100
--------------------------------------------------
```

Figure 16. Show Data



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


Enter your menu choice number: 3
Written to File:
--------------------------------------------------
Bob,Smith,Python 100
Sue,Jones,Python 100
James,Jabal,Python 100
--------------------------------------------------
```

Figure 17. Save Data

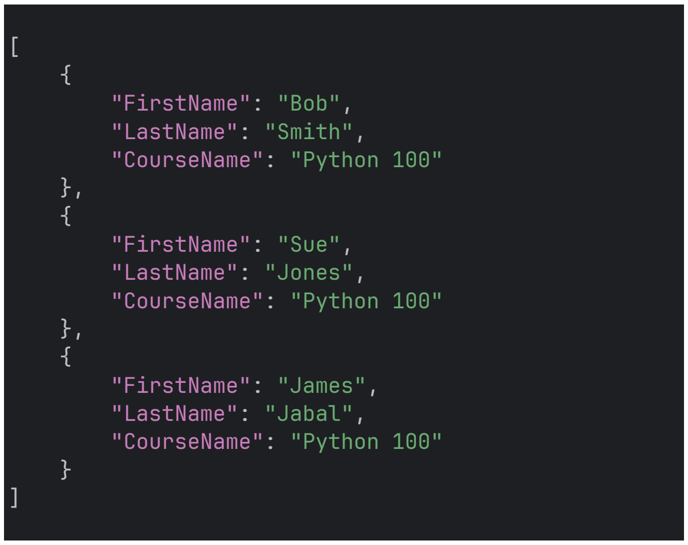


```
---- Course Registration Program ----
  Select from the following menu:
    1. Register a Student for a Course.
    2. Show current data.
    3. Save data to a file.
    4. Exit the program.
-----------------------------------------


Enter your menu choice number: 4
Program Ended

Process finished with exit code 0
```

Figure 18. Exit Program

JSON in PyCharm

The contents of the “Enrollments.json” file saved was checked after testing in PyCharm (Figure 19).

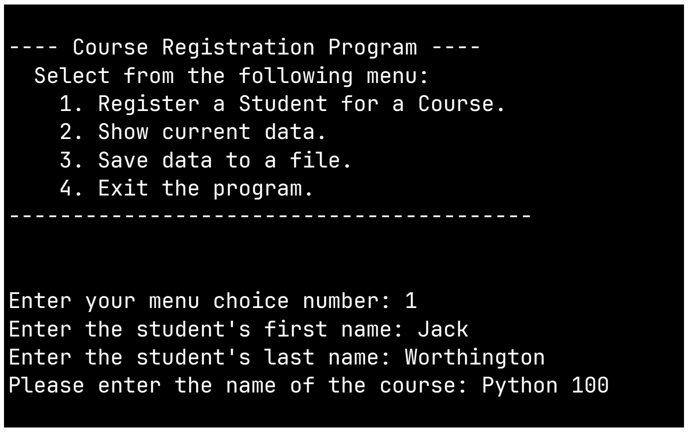


```json
[
    {
        "FirstName": "Bob",
        "LastName": "Smith",
        "CourseName": "Python 100"
    },
    {
        "FirstName": "Sue",
        "LastName": "Jones",
        "CourseName": "Python 100"
    },
    {
        "FirstName": "James",
        "LastName": "Jabal",
        "CourseName": "Python 100"
    }
]
```

Figure 19. “Enrollments.json” file after testing in PyCharm

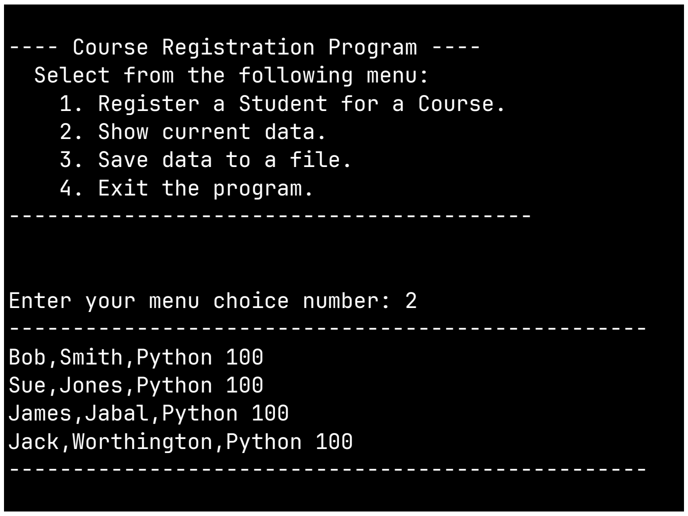
Terminal

The program was tested in the Terminal with similar results (Figures 20 – 23), by adding another student.



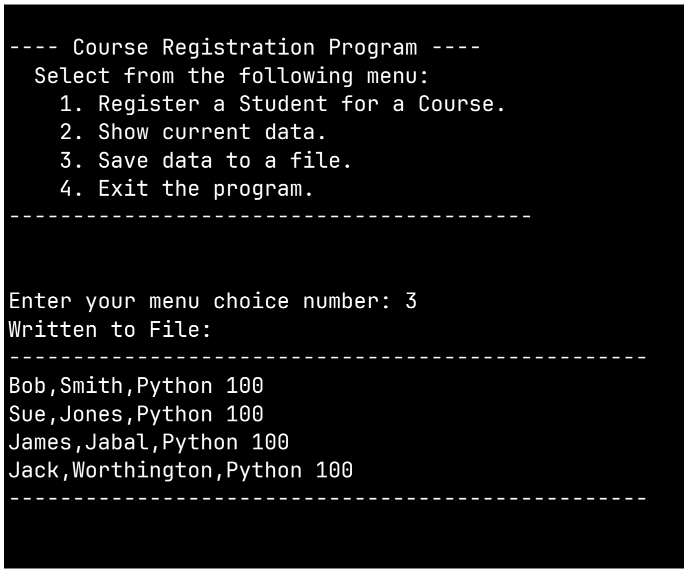
```
---- Course Registration Program ----
  Select from the following menu:
    1. Register a Student for a Course.
    2. Show current data.
    3. Save data to a file.
    4. Exit the program.
-----------------------------------------


Enter your menu choice number: 1
Enter the student's first name: Jack
Enter the student's last name: Worthington
Please enter the name of the course: Python 100
```

Figure 20. Add Data



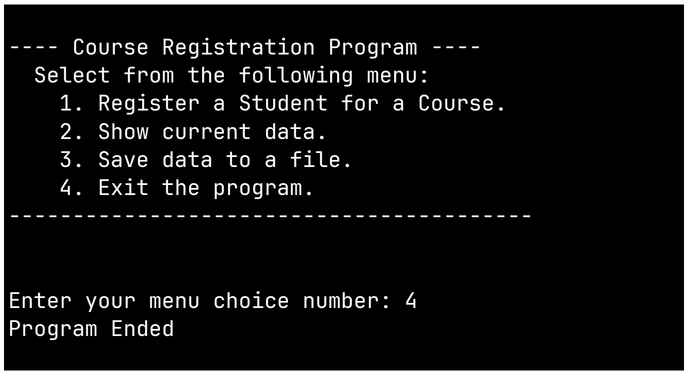
```
---- Course Registration Program ----
  Select from the following menu:
    1. Register a Student for a Course.
    2. Show current data.
    3. Save data to a file.
    4. Exit the program.
-----------------------------------------


Enter your menu choice number: 2
--------------------------------------------------
Bob,Smith,Python 100
Sue,Jones,Python 100
James,Jabal,Python 100
Jack,Worthington,Python 100
--------------------------------------------------
```

Figure 21. Show Data



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


Enter your menu choice number: 3
Written to File:
--------------------------------------------------
Bob,Smith,Python 100
Sue,Jones,Python 100
James,Jabal,Python 100
Jack,Worthington,Python 100
--------------------------------------------------
```

Figure 22. Save Data

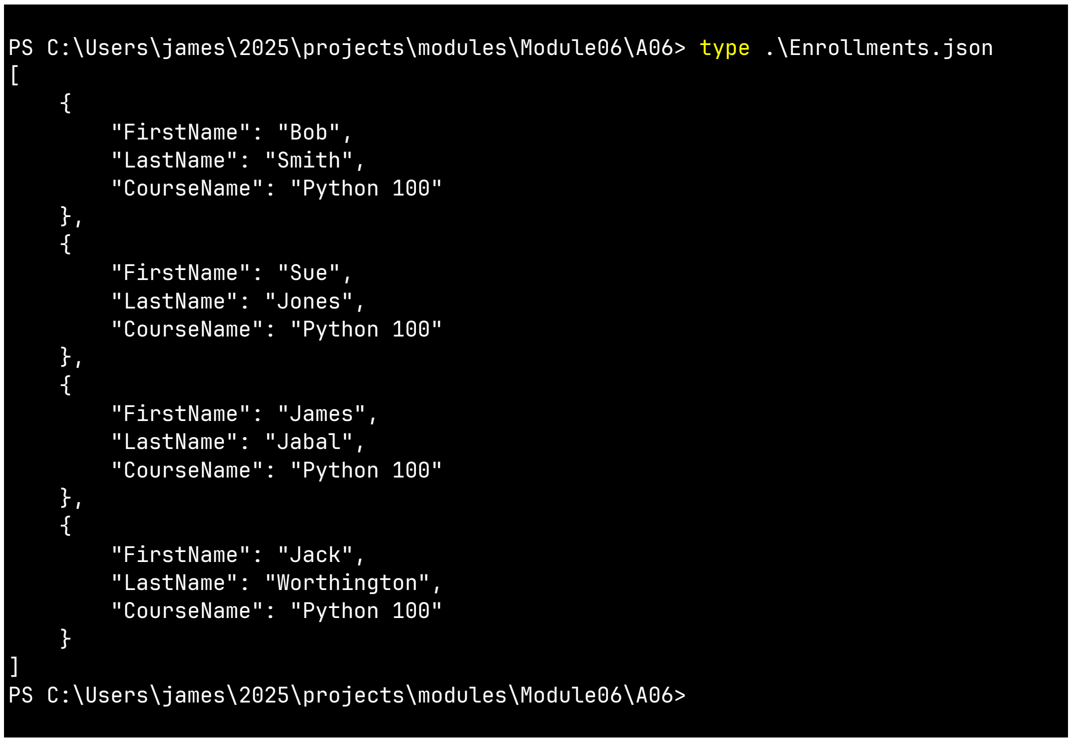


```
---- Course Registration Program ----
  Select from the following menu:
    1. Register a Student for a Course.
    2. Show current data.
    3. Save data to a file.
    4. Exit the program.
-----------------------------------------


Enter your menu choice number: 4
Program Ended
```

Figure 23. Exit Program

JSON in Terminal

The contents of “Enrollments.json” was checked in the Terminal after testing in both PyCharm & Terminal (Figure 24).

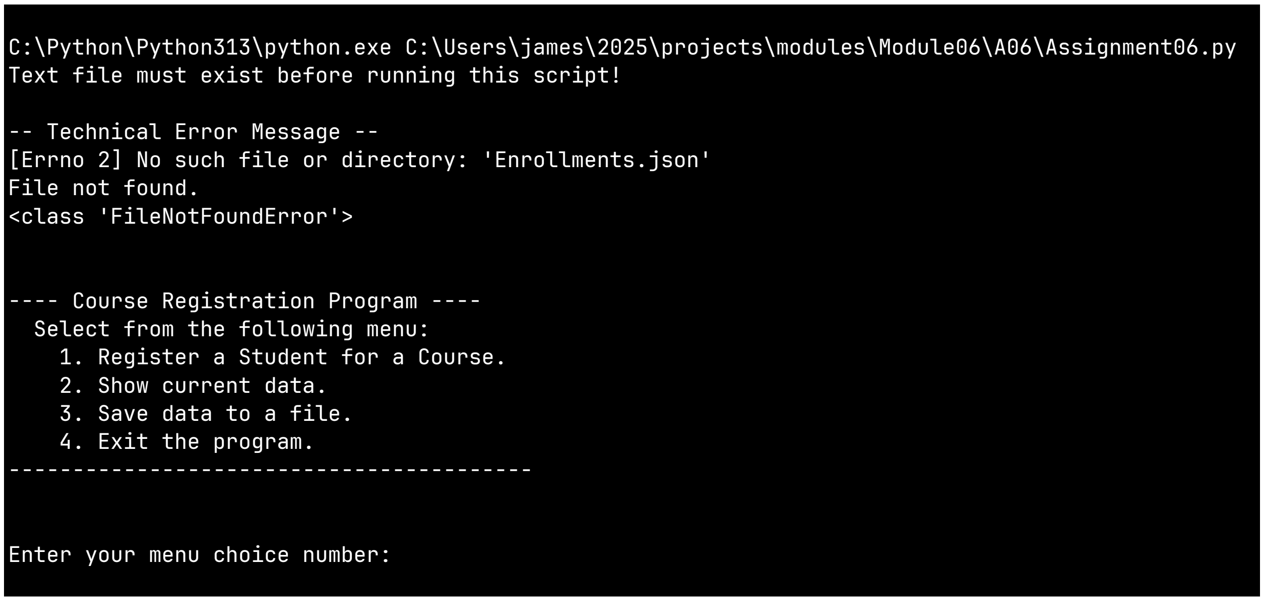


```
PS C:\Users\james\2025\projects\modules\Module06\A06> type .\Enrollments.json
[
    {
        "FirstName": "Bob",
        "LastName": "Smith",
        "CourseName": "Python 100"
    },
    {
        "FirstName": "Sue",
        "LastName": "Jones",
        "CourseName": "Python 100"
    },
    {
        "FirstName": "James",
        "LastName": "Jabal",
        "CourseName": "Python 100"
    },
    {
        "FirstName": "Jack",
        "LastName": "Worthington",
        "CourseName": "Python 100"
    }
]
PS C:\Users\james\2025\projects\modules\Module06\A06>
```

Figure 24. “Enrollments.json” file after testing in PyCharm & Terminal

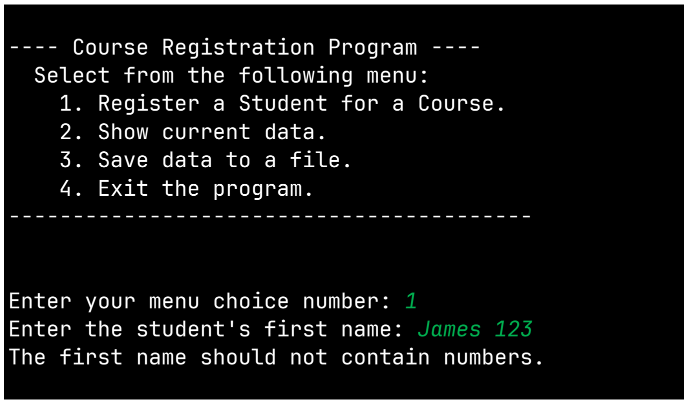
Error Handling

The error handling functions were also tested: reading a non-existing JSON file (Figure 25), entering a numeric student first name(Figure 26) and last name (Figure 27), and writing to a read-only JSON file (Figure 28). The results are as shown below.



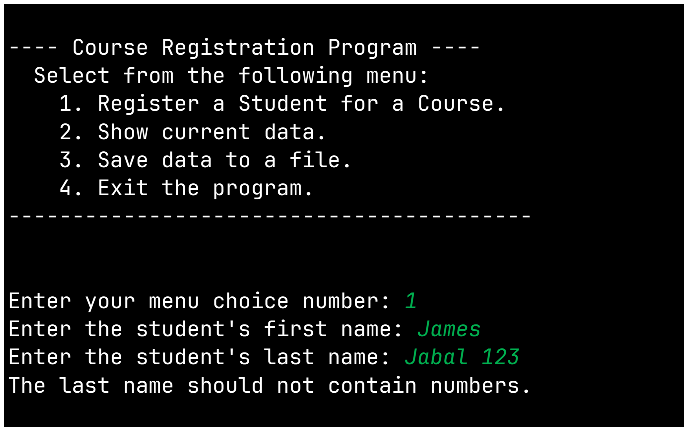
```
C:\Python\Python313\python.exe C:\Users\james\2025\projects\modules\Module06\A06\Assignment06.py
Text file must exist before running this script!

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


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


Enter your menu choice number:
```

Figure 25. FileNotFoundError when opening a non-existing JSON file



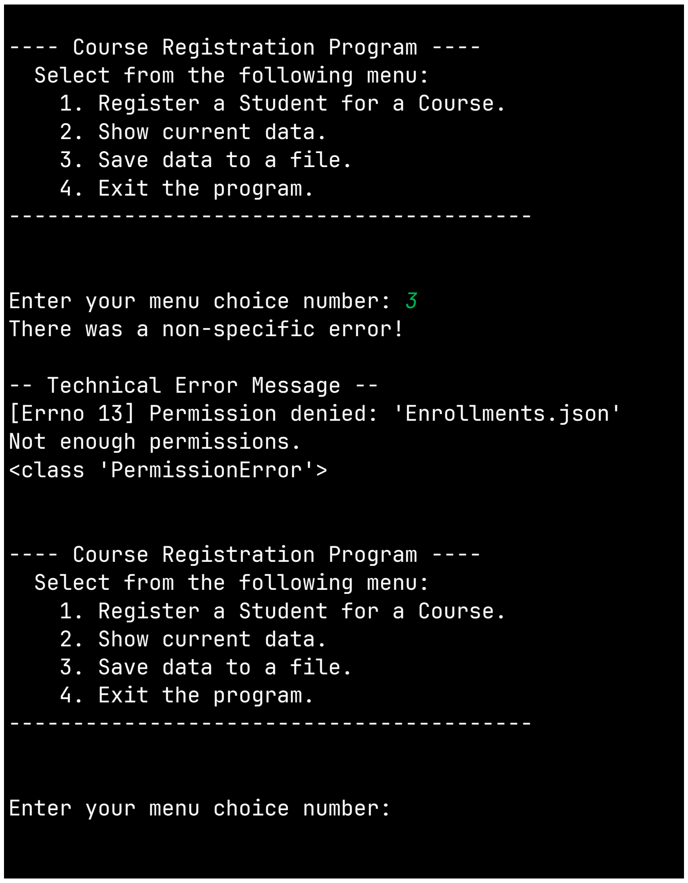
```
---- Course Registration Program ----
  Select from the following menu:
    1. Register a Student for a Course.
    2. Show current data.
    3. Save data to a file.
    4. Exit the program.
-----------------------------------------


Enter your menu choice number: 1
Enter the student's first name: James 123
The first name should not contain numbers.
```

Figure 26. ValueError for a numeric first name



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


Enter your menu choice number: 1
Enter the student's first name: James
Enter the student's last name: Jabal 123
The last name should not contain numbers.
```

Figure 27. ValueError for a numeric last name



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


Enter your menu choice number: 3
There was a non-specific error!

-- Technical Error Message --
[Errno 13] Permission denied: 'Enrollments.json'
Not enough permissions.
<class 'PermissionError'>


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


Enter your menu choice number:
```

Figure 28. Catch-all exception example when writing to a read-only JSON file

Summary

This assignment helped me understand functions, classes, and the separation of concerns. It took effort to modularize the code, by creating two classes, and several functions within those classes. Overall, it made the main code much easier to understand. It also helped me understand and determine which variables must be global or local. After that, I learned how to pass arguments to parameters of the other functions within other classes. Debugging helped the most in understanding the variable contents, modifying & returning values, and what code will be executed next. I also looked for ways to improve readability as the program became more complex. The final steps would be to upload my program code, resulting JSON file, and this document to my repository in GitHub.