Matthew Young

Introduction to Programming: Python

03/05/2025

GitHub URL: https://github.com/myoung1010/IntroToProg-Python-Mod05

Functions and Separation of Concerns

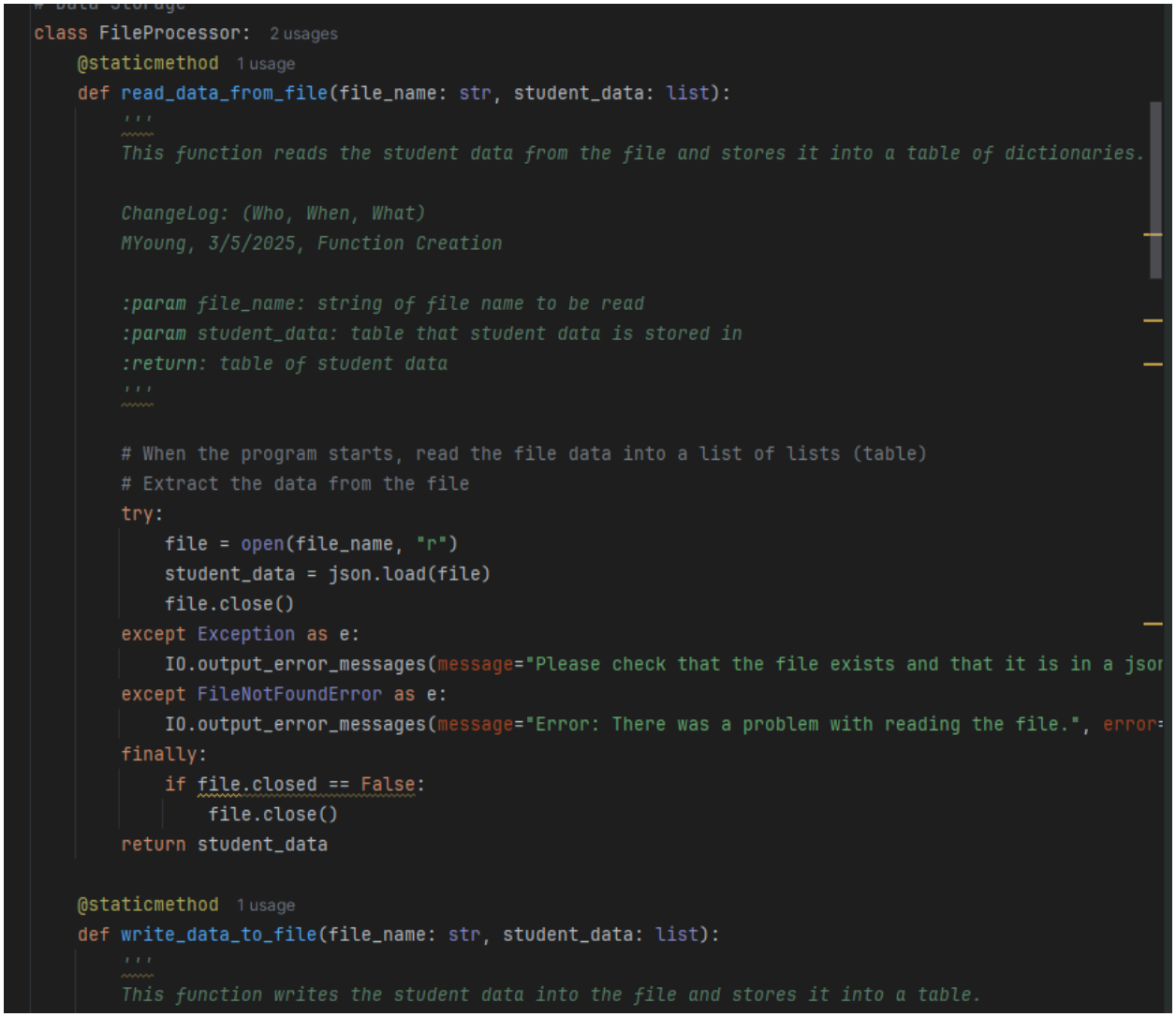
Assignment 06

Introduction

This assignment focused on structure and modularity of the program that we have been building up these past weeks. Several concepts such as functions, classes, document strings, and separation of concerns were incorporated into the script to make it easier to read and follow along. These changes removed a lot of the repeated statements that were previously used to perform tasks multiple times and expanded the scope of the assignment beyond just the task of creating a database of registered student information.

Functional Modularity

As we worked through and expanded on the capabilities of the student registration program with displaying student data and structured error handling when processing file and user data, a lot of repeated statements were used to perform what was essentially the same task multiple times. Times like these are were defining functions come in handy. Functions are reusable blocks of code that performs a specific task or set of tasks. By breaking up all the program's tasks into modules to read data from the file, write data to the file, input student data, display student data, and more, the program is divided into smaller sections that makes managing the code easier by allowing the developer to focus on specific tasks for maintenance and debugging purposes as seen in Figure 1.



```
class FileProcessor:  2 usages
    @staticmethod  1 usage
    def read_data_from_file(file_name: str, student_data: list):
        '''
        This function reads the student data from the file and stores it into a table of dictionaries.

        ChangeLog: (Who, When, What)
        MYoung, 3/5/2025, Function Creation

        :param file_name: string of file name to be read
        :param student_data: table that student data is stored in
        :return: table of student data
        '''

        # When the program starts, read the file data into a list of lists (table)
        # Extract the data from the file
        try:
            file = open(file_name, "r")
            student_data = json.load(file)
            file.close()
        except Exception as e:
            IO.output_error_messages(message="Please check that the file exists and that it is in a jso
        except FileNotFoundError as e:
            IO.output_error_messages(message="Error: There was a problem with reading the file.", error=
        finally:
            if file.closed == False:
                file.close()
        return student_data

    @staticmethod  1 usage
    def write_data_to_file(file_name: str, student_data: list):
        '''
        This function writes the student data into the file and stores it into a table.
```

Figure 1: Defined functions to read and write data to file.

Figure 1 also defined a “class” called “FileProcessor”. Classes are essentially grouped functions that served related purposes like reading and write data to a file. Another class called “IO” was also created for the program to handle all of the input and output functions to handle the menu options presented in the script. This provides a natural way to organize the code to further hone the theme of modularity in this assignment. Another concept that was explored in this figure are static classes indicated by the @staticmethod decorator on the lines preceding the function definitions. Static classes are functions whose code never changes and allows the use of class functions directly without having to create an object first. Finally, the long, multiple line comment string that appears before the function’s code is called a document string (or Docstring). These serve as the header at the beginning of a class or function that developers use to include addition notes and additional information about the class or function as needed. The above figure uses the Docstring to show the change log and a brief explanation of the purpose of each requested parameter and the return values. With the use of classes and functions, the main

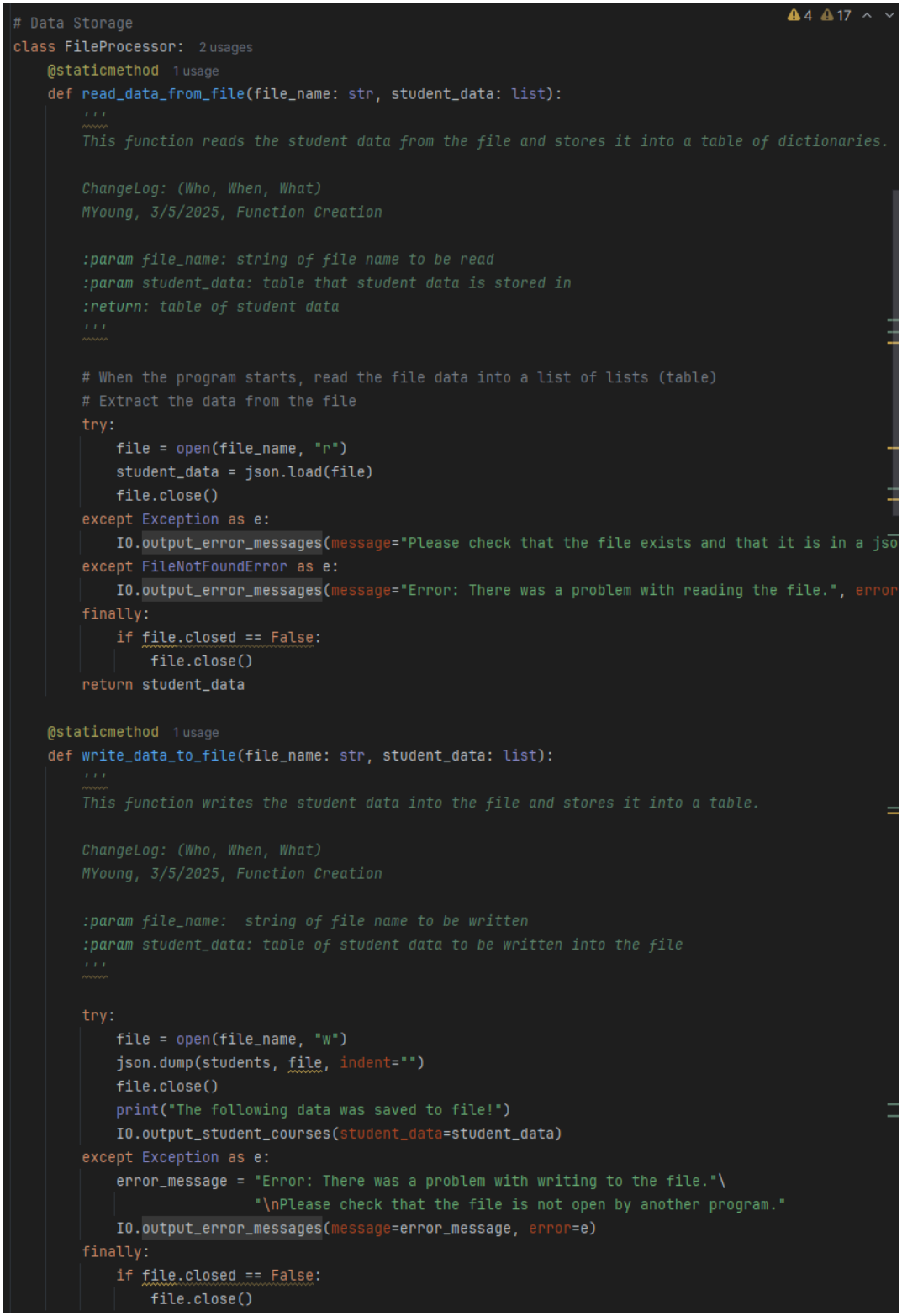
body of the script was able to be reduced to only a few lines as shown in Figure 2 below, where it is much easier to follow along the logic of the script.

```
# Processing Layer
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!
        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":
        FileProcessor.write_data_to_file(file_name=FILE_NAME, student_data=students)
        continue

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

print("Program Ended")
```

Figure 2: Reduced main body that showcases the overall logic flow of the program.

Global and Local Variables

Another concept that was exercised in this assignment was separation of concerns. This is one of the largest fundamental software design principles that aims to enhance the maintainability, scalability, and readability of code by breaking it down into self-contained components, following the theme of modularity. “Concerns” are specific aspects or responsibilities of a program’s functionality. These can be divided into three layers: the Data Layer, the Processing Layer, and the Presentation Layer. All three of these layers were established to organize the code based on specific concerns. The Data Layer, shown in Figure 3, shows how the program handled the data storage and retrieval processes of the code.



```python
# Data Storage
class FileProcessor:  2 usages
    @staticmethod  1 usage
    def read_data_from_file(file_name: str, student_data: list):
        '''
        This function reads the student data from the file and stores it into a table of dictionaries.

        ChangeLog: (Who, When, What)
        MYoung, 3/5/2025, Function Creation

        :param file_name: string of file name to be read
        :param student_data: table that student data is stored in
        :return: table of student data
        '''

        # When the program starts, read the file data into a list of lists (table)
        # Extract the data from the file
        try:
            file = open(file_name, "r")
            student_data = json.load(file)
            file.close()
        except Exception as e:
            IO.output_error_messages(message="Please check that the file exists and that it is in a jso
        except FileNotFoundError as e:
            IO.output_error_messages(message="Error: There was a problem with reading the file.", error
        finally:
            if file.closed == False:
                file.close()
        return student_data

    @staticmethod  1 usage
    def write_data_to_file(file_name: str, student_data: list):
        '''
        This function writes the student data into the file and stores it into a table.

        ChangeLog: (Who, When, What)
        MYoung, 3/5/2025, Function Creation

        :param file_name:  string of file name to be written
        :param student_data: table of student data to be written into the file
        '''

        try:
            file = open(file_name, "w")
            json.dump(students, file, indent="")
            file.close()
            print("The following data was saved to file!")
            IO.output_student_courses(student_data=student_data)
        except Exception as e:
            error_message = "Error: There was a problem with writing to the file."\
                            "\nPlease check that the file is not open by another program."
            IO.output_error_messages(message=error_message, error=e)
        finally:
            if file.closed == False:
                file.close()
```

Figure 3: Data Layer of the script

The Presentation Layer, shown in Figure 4, shows how the code handled the presentation of data to the user. Each task to be executed by the script was assigned its own function under the IO class. This is seen with the output_error_messages() function, output_menu() function, and the input_menu_choice() functions, which handles the error handling, displaying of the menu options, and the collection of what the user wants to do. What is not shown are the output_student_courses() and input_student_data() functions, which handled the student data collection and display of the gathered student data. The error messages that are typically displayed with the established error handling is now covered by the output_error_message() function.

```python
# Presentation Layer
class IO:  11 usages
    @staticmethod  6 usages
    def output_error_messages(message: str, error: Exception = None):
        '''
        This function outputs error messages when an error occurs.

        ChangeLog: (Who, When, What)
        MYoung, 3/5/2025, Function Creation

        :param message: string of error message to be displayed
        :param error: error type
        '''

        print(message)
        if error is not None:
            # Prints the custom message
            print("-- Technical Error Message -- ")
            print(error.__doc__)
            print(error.__str__())

    @staticmethod  1 usage
    def output_menu(menu: str):
        '''
        This function displays menu options for the user to choose from.

        ChangeLog: (Who, When, What)
        MYoung, 3/5/2025, Function Creation

        :param menu: string of menu options to be selected from
        '''

        print(menu)

    @staticmethod  1 usage
    def input_menu_choice():
        '''
        This function collects user choice from menu options

        ChangeLog: (Who, When, What)
        MYoung, 3/5/2025, Function Creation

        :return: string of user's menu choice
        '''

        try:
            choice = input("What would you like to do: ")
            if choice not in ["1", "2", "3", "4"]:
                raise Exception("Please only choose option 1, 2, or 3")
        except Exception as e:
            IO.output_error_messages(message="Please only choose option 1, 2, or 3", error=e)
        return choice
```

Figure 4: Excerpt of the Presentation Layer of the code

Finally, the Processing Layer, shown in Figure 5, once again shows the main body of the script which handles the processing of user collecting data and information. This layer is also known as the "Business Logic Layer" or the "Application layer" which handles the core functionality and business roles of the application. This layer processes and transforms the data acquired in menu choice 1 by collecting the student first and last name and course name and collects in into the cumulative student data list, all comprised within the IO.input_student_data() function under the IO class. The display of registered student data from menu choice 2 is completely handled by the IO.output_student_courses() function and part of the write_data_to_file() function in menu choice 3.

```
# 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!
        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":
        FileProcessor.write_data_to_file(file_name=FILE_NAME, student_data=students)
        continue

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

print("Program Ended")
```

Figure 5: The Processing Layer of the script that comprises the main body of the script.

Conclusion

This assignment focused on the structure and organization of the various tasks that the code that we've been building up over the past weeks into separate functions and classes based on separation of concerns. This makes the code more modular which makes it easier to manage

and troubleshoot in the case of necessary debugging. This also reduced the amount of repetitive code that needed to be written to perform the same task at several points in the code.