Hannah Brown
5/25/2025
IT FDN 110 A
Assignment 06
https://github.com/hkbrown42/IntroToProg-Python-Mod06

Assignment 06 - Functions and Classes

Introduction

This week we learned about functions and classes, and how they can be used to both organize and streamline a longer or more complex program. We also addressed the concept of separation of concern, and how implementing this concept in code helps improve the efficiency, maintainability, and readability of a program. This document describes the process of how I modified the starter file (equivalent to last week's assignment) to include classes and functions, and to follow the principles of separation of concern.

Initial Setup

The first thing I did to set up the program was to make sure the json module was imported at the top of the program. I then added comments to organize the code into three layers: Data (where the constants and variables will be defined), Processing (where the functions to read and write from the JSON file will be defined), and Presentation (where all the input and output functions will be defined). Next, I verified that the MENU and FILE_NAME constants were defined correctly. I then removed most of the global variable definitions, since I will be using them locally within functions instead; the only remaining global variables are menu_choice and students, which are initialized as an empty string and an empty list respectively.

```
8   import json
9
10  # -- Data -- #
11  # Define the data constants
12  MENU: str = """
13  ---- Course Registration Program ----
14    Select from the following menu:
15      1. Register a student for a course.
16      2. Show current data.
17      3. Save data to a file.
18      4. Exit the program.
19  ----------------------------------------
20  """
21  FILE_NAME: str = "Enrollments.json"
22
23  # Define the data variables
24  menu_choice: str = ""       # holds the menu selection made by the user
25  students: list = []         # a table of student data
```

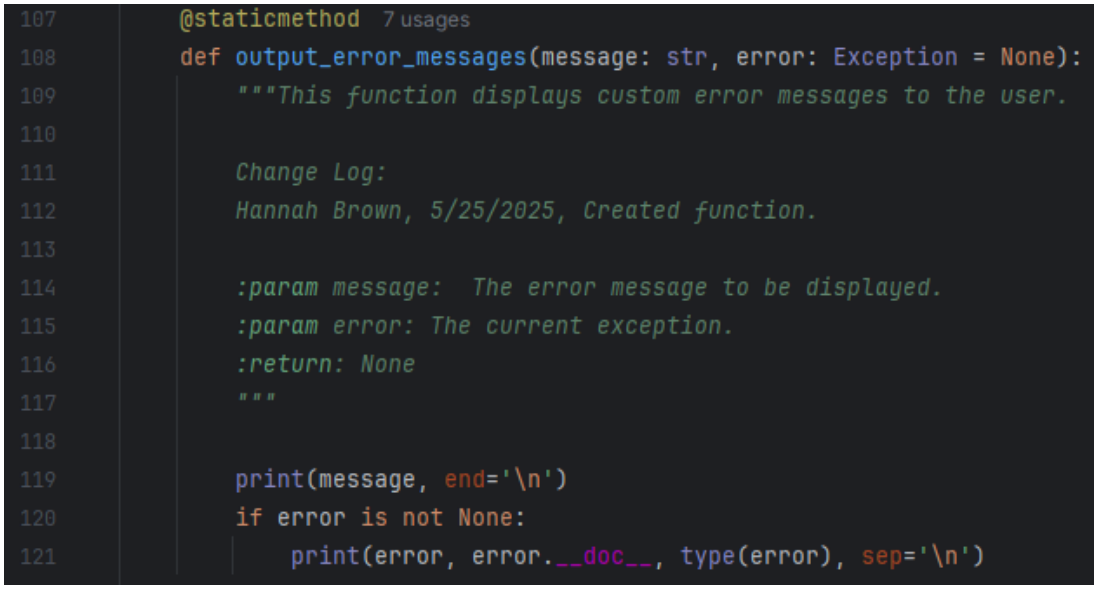
Figure 1: Module importing and definition of constants and variables.

I also copied the provided “Enrollments.json” file into the A06 project, so that there will be starting data to work with.

Modifying and Testing the Program

Overall, creating this program required very little new code - it was predominantly a matter of creating new classes and functions, then sorting and organizing the existing code into these. I started out by creating two new classes - FileProcessor and IO - and adding descriptive document strings. Once the classes were created, I could start defining functions one by one, using the @staticmethod decorator for each function - this will allow me to call these functions without first creating an instance of the class. I also made sure to add descriptive document strings to each function as I defined it.

I began by defining the output_error_messages() function in the IO class - it was important to get this function set up first, so that I could test out the structured error handling on each subsequent function as I create them. This function is defined with two parameters - message, which will take the argument of a string of a custom error message, and error, which will take the argument of an exception being raised.



```
107     @staticmethod  7 usages
108     def output_error_messages(message: str, error: Exception = None):
109         """This function displays custom error messages to the user.
110
111         Change Log:
112         Hannah Brown, 5/25/2025, Created function.
113
114         :param message:  The error message to be displayed.
115         :param error: The current exception.
116         :return: None
117         """
118
119         print(message, end='\n')
120         if error is not None:
121             print(error, error.__doc__, type(error), sep='\n')
```

Figure 2: Definition of output_error_messages() function within class IO.

From here, I moved to the start of the program and began moving code into functions as I went through the flow of the program. When the program starts, the first thing it does is reads the contents of the “Enrollments.json” file and stores it in a list of dictionaries. I took this block of code that performs this operation - including the structured error handling - and moved it into the FileProcessor class, then indented it to define the new read_data_from_file() function. This function calls for two parameters - file_name, which will take the argument of the name of the JSON file, and student_data, which will take the argument of a list where the JSON data will be stored. I also changed the code to call the parameters instead of the hardcoded values that

were there previously, changed the structured error handling code to call the newly-created output_error_messages() function, and added a return value at the end.

```
38      @staticmethod  1 usage
39      def read_data_from_file(file_name: str, student_data: list):
40          """This function loads the existing JSON data and stores
41          it in a list of dictionaries.
42
43          Change Log:
44          Hannah Brown, 5/25/2025, Created function.
45
46          :return: currently saved data (list)
47          """
48
49          try:
50              file = open(file_name, 'r')
51              student_data = json.load(file)
52              file.close()
53          except FileNotFoundError as e:
54              IO.output_error_messages(
55                   message: "Please make sure the file you are trying to open exists!", e)
56          except Exception as e:
57              IO.output_error_messages(
58                   message: "Unspecified error. Please try again.", e)
59          finally:
60              if not file.closed:
61                  file.close()
62          return student_data
```

Figure 3: Definition of read_data_from_file() function within class FileProcessor.

Finally, at the start of the program's main body, I added code calling the function and passed the correct arguments - the FILE_NAME constant, and the students list.

```
217     # Start of program's main body
218     # When the program starts, read the file data into a list of lists (table)
219     students = FileProcessor.read_data_from_file(file_name=FILE_NAME,
220                                                  student_data=students)
```

Figure 4: Start of program calling the function, with arguments.

At this point, I temporarily added a line of code underneath that was just "print(students)" - I then ran the program and verified that the file data was loaded correctly. I also temporarily changed the name in the FILE_NAME constant in order to force a FileNotFoundError and make sure that error handling worked. Once I verified the function was working properly and the error handling was performing as expected, I changed the FILE_NAME constant back to the correct name, and removed the print statement.

```
"C:\Users\hkbro\OneDrive\Documents\Python\Foundations of Python Programming\As
Please make sure the file you are trying to open exists!
[Errno 2] No such file or directory: 'Enrollment.json'
File not found.
<class 'FileNotFoundError'>
```

Figure 4: Forcing a FileNotFoundError as a test.

```
"C:\Users\hkbro\OneDrive\Documents\Python\Foundations of Python Programming\Assignments\A06\.venv\Sc
[{'FirstName': 'Bob', 'LastName': 'Smith', 'CourseName': 'Python 100'}, {'FirstName': 'Sue', 'LastNa
```

Figure 5: Printed out students variable to test that JSON data was loaded correctly.

After the program has loaded the JSON data, it needs to display the menu of options to the user, then ask for the user's input. I navigated to the IO class and defined two new functions - output_menu(), to display the menu, and input_menu_choice(), to gather user input. Defining output_menu() was very straightforward, since it simply takes the passed argument of the MENU constant and prints it, but previously menu_choice was just a single line of code asking for input, and I needed to flesh this function out further, including structured error handling to call the output_error_messages() function if the user enters in a choice that is not one of the valid options.

```
@staticmethod  1 usage
def output_menu(menu: str):
    """This function displays the menu to the user.

    Change Log:
    Hannah Brown, 5/25/2025, Created function.

    :param menu: The menu to be displayed.
    :return: None
    """

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

```
@staticmethod  1 usage
def input_menu_choice():
    """This function prompts the user to select from the menu
    and stores the user's selection.

    Change Log:
    Hannah Brown, 5/25/2025, Created function.

    :return: user's menu selection (str)
    """

    choice = ""
    try:
        choice = input("Please enter your menu selection: ")
        if choice not in ("1","2","3","4"):
            raise Exception("Please choose a valid option!")
    except Exception as e:
        IO.output_error_messages(e.__str__())
    return choice
```

Figures 6-7: Definition of output_menu() and input_menu_choice() functions within class IO.

Returning to the program's logic, within the while loop, I added function calls with the necessary arguments. I then ran the program and made sure that the menu displayed correctly, and that it prompted me for input. I tried entering in an invalid option to make sure the error handling performed correctly, and it displayed the expected error message before returning me to the menu.

```
222     while True:
223         # Present the menu of choices
224         IO.output_menu(menu=MENU)
225         menu_choice = IO.input_menu_choice()
```

Figure 8: Presenting the menu and asking for input

```
---- 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.
----------------------------------------


Please enter your menu selection: 5
Please choose a valid option!


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

Figure 9: Menu displayed correctly, and forcing an invalid option exception.

Now I can move on to the code used for each menu option. For menu option 1, I defined the new function input_student_data() in the IO class with one parameter (student_data, a list of dictionaries that will hold the student registrations), and moved the code for this menu option into the function definition. To avoid confusion, I changed the name of one of the local variables here to student_registration. I also updated the structured error handling code to call the output_error_messages() function, and added a return value at the end.

Returning to the program's logic, within the if statement for menu option 1, I added the call to this new function with the appropriate argument. Here I also added a temporary print statement to make sure that the new student registration had indeed been appended to the students list. I then ran the program and tested to make sure the error handling performed as expected if I entered an invalid first or last name. Once I was satisfied this portion was working as expected, I removed the print statement.

```python
@staticmethod  1 usage
def input_student_data(student_data: list):
    """This function registers a student for a course by taking in user
    input of a student's name and the course name and storing it in a
    table.

    Change Log:
    Hannah Brown, 5/25/2025, Created function

    :return: students' names and courses (str)
    """

    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: ")
        student_registration = {"FirstName": student_first_name,
                                "LastName": student_last_name,
                                "CourseName": course_name}
        students.append(student_registration)
        print()
        print(f"You have registered {student_first_name} "
              f"{student_last_name} for {course_name}.")
    except ValueError as e:
        IO.output_error_messages(
            message: "Please make sure you have entered the correct information.", e)
    except Exception as e:
        IO.output_error_messages(
            message: "Unspecified error. Please try again.", e)
    return student_data
```

Figure 10: Definition of input_student_data() function within class IO.

```
227         # Input user data
228         if menu_choice == "1":
229             IO.input_student_data(student_data=students)
230             continue
```

Figure 11: Program logic calling new function.

```
Please enter your menu selection: 1
Enter the student's first name: 12345
Please make sure you have entered the correct information.
The first name should not contain numbers.
Inappropriate argument value (of correct type).
<class 'ValueError'>
```

Figure 12: Forcing a ValueError.

Moving on to menu option 2, I returned to the IO class and defined the new function output_student_courses() with one parameter (student_data, the list of dictionaries that holds the student registrations), and moved the code for this menu option into the function definition. This simply prints out the contents of the list, and does not require structured error handling here.

```
199     @staticmethod  1 usage
200     def output_student_courses(student_data: list):
201         """This function displays the current student registrations.
202
203         Change Log:
204         Hannah Brown, 5/25/2025, Created function.
205
206         :return: None
207         """
208
209         print("-" * 50)
210         for student in student_data:
211             print(f"Student {student["FirstName"]} {student["LastName"]} "
212                   f"is enrolled in {student["CourseName"]}")
213         print("-" * 50)
```

Figure 13: Definition of output_student_courses() function within class IO.

Returning to the program's logic, within the if statement for menu option 2, I added the call to this new function with the appropriate argument. I then ran the program and tested to make sure the data was displayed as I expected.

```
232     # Present the current data
233     elif menu_choice == "2":
234         IO.output_student_courses(student_data=students)
235         continue
```

Figure 14: Program logic calling new function.

```
Please enter your menu selection: 2
-------------------------------------------------
Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
-------------------------------------------------
```

Figure 15: Current data displayed correctly.

Moving onto menu option 3, I returned to the FileProcessor class and defined the new function write_data_to_file() with two parameters (file_name, which takes the argument of the name of the file where the data will be written, and student_data, which takes the argument of the previously-used list of student registrations) and moved the code for this menu option into the function definition. I also changed the code to call the parameters instead of the hardcoded

values that were there previously and updated the structured error handling code to call the output_error_messages() function.

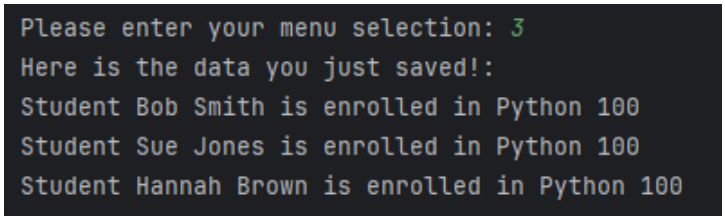
```
65      @staticmethod  1 usage
66      def write_data_to_file(file_name: str, student_data: list):
67          """This function takes the course registration data currently stored
68          and writes it to a designated JSON file.
69
70          Change Log:
71          Hannah Brown, 5/25/2025, Created function.
72
73          :return: None
74          """
75
76          try:
77              file = open(file_name, 'w')
78              json.dump(student_data, file, indent=2)
79              file.close()
80              print("Here is the data you just saved!:")
81              for student in student_data:
82                  print(f"Student {student["FirstName"]} "
83                        f"{student["LastName"]} is enrolled in {student["CourseName"]}")
84          except TypeError as e:
85              IO.output_error_messages(
86                  message: "Please make sure the data is a valid JSON format!", e)
87          except Exception as e:
88              IO.output_error_messages(
89                  message: "Unspecified error. Please try again.", e)
90          finally:
91              if not file.closed:
92                  file.close()
```

Figure 16: Definition of write_data_to_file() function within class FileProcessor.

Returning to the program's logic, within the if statement for menu option 3, I added the call to this new function with the appropriate arguments. I then ran the program, ran through menu option 1 to add a student registration, then ran through menu option 3 to make sure the new data was saved correctly. I verified this by checking that the "Enrollments.json" file contained the correct new data. Here I was unable to successfully force a TypeError, so I can't be entirely sure that the error handling will work exactly as planned.

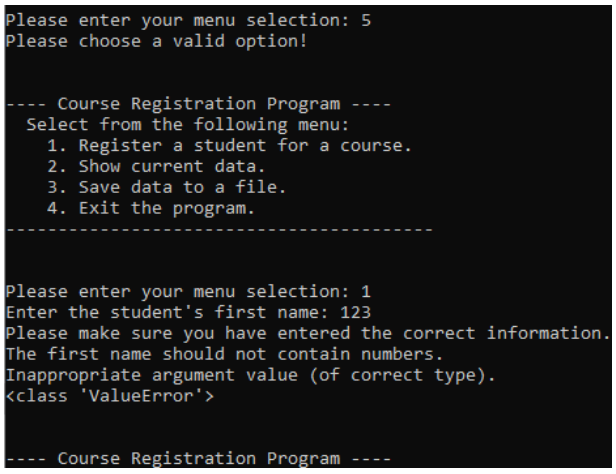
```
237     # Save the data to a file
238     elif menu_choice == "3":
239         FileProcessor.write_data_to_file(file_name=FILE_NAME,
240                                          student_data=students)
241         continue
```

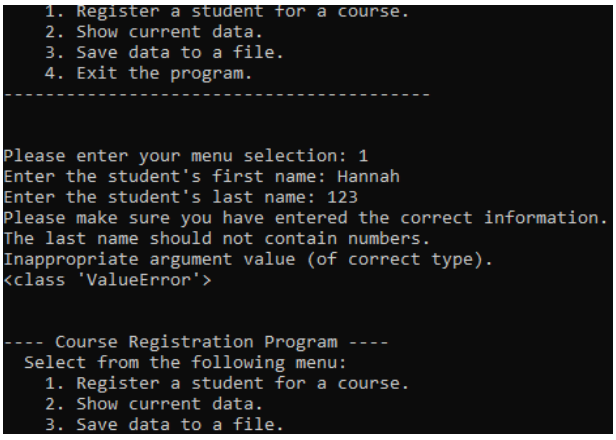
Figure 17: Program logic calling new function.



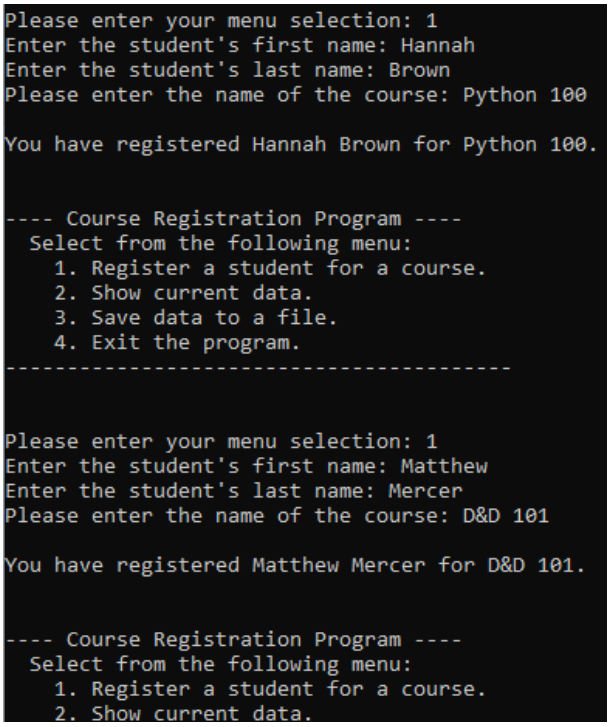
Figures 18-19: New data saved to file, and verified in JSON file.

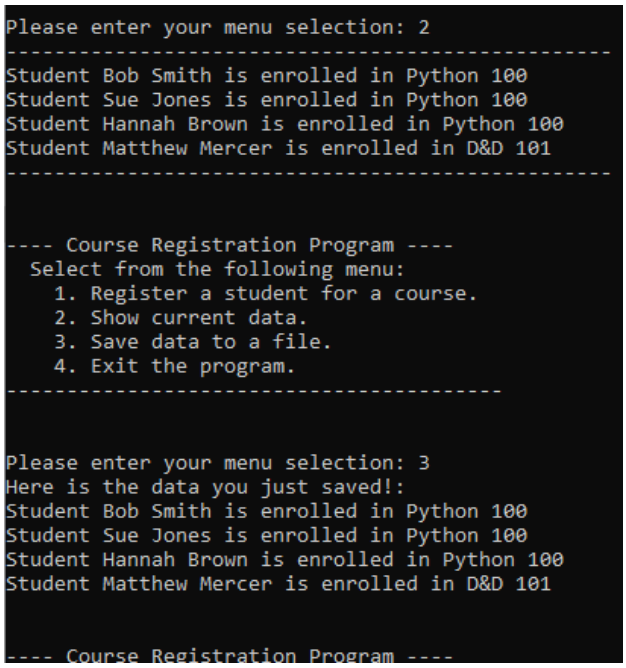
Finally, I removed the else statement at the end of the program - the structured error handling for the new input_menu_choice() function makes this redundant. At this point I was satisfied that the program ran smoothly and as expected within PyCharm, as I've thoroughly tested each part of it as I went along. I then ran the program in the command prompt and went through each menu option again, making sure it performed as expected. Here I went through menu option 1 several times in order to add multiple registrations. Once again, I checked the "Enrollments.json" file afterwards to verify that the multiple registrations had all been saved to the file correctly. I also tried forcing any errors I could in order to verify the error handling.





Figures 20-21: Forcing errors in command prompt.





Figures 22-23: Program running in command prompt with multiple registrations.

Summary

I found this week's assignment trickier than most of the previous ones - I felt like I understood everything conceptually, but I found myself getting a little bogged down while trying to sort the existing code into functions. I also found it a little bit confusing keeping the variables and parameters with similar names straight (the students list global variable, the student_data local variable, the student_data parameter, etc.). Overall I feel like I've grasped the concepts, and I'm looking forward to eventually writing a more complicated program like this from scratch in the future.