Edwin Kintu-Lubowa

November 19, 2024

Foundations of Programming: Python

Assignment06

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

Functions, Classes, Error Handling, and Separation of Concerns (SoC)

Introduction

This week, I learned about functions in Python: how to define the functions, how to define global variables within the function, how to use parameters, and pass arguments to those parameters. Alongside functions, I also learned about Classes and the concept of Separation of Concerns (SoC) and how classes can organize functions, and how SoC organizes the placement of those functions into separate classes.

"In this module, we looked at how functions work and are used to organize your code. We also looked that how classes can organize functions, and how the separation of concerns pattern can organize the placement of those functions into separate classes." (Randal Root, 2024, Introduction to Programming with Python: Module 6 - Functions, Page 43)

In this knowledge document, I will chronicle the concepts I learned and the process I employed to successfully complete this assignment.

The Transition

This week's focus was on the introduction of Functions and Classes into the script, and organizing the functions based on the specific tasks they performed.
I started the assignment by opening the starter file provided by the instructor. The starter file, itself, was the answer to Assignment05 and this emphasized a sense of continuity. The first task I performed was update the file header, then run the file and ensure it run as expected — which it did —then I saved the file as "Assignment06.py"

The next task was to comment out all the data variables and constants in the script, as the Assignment called for the use of only two variables: "menu_choice" a string set to empty, and "students" a list of dictionaries, also set to empty. (Figure 1)

```
# 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.csv"
FILE_NAME: str = "Enrollments.json"

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

# commented out all prior programs variables; will be using the above listed variables for
# Assignment06; and the rest will be used as local variables within the functions
# # Define the Data Variables and constants
# student_first_name: str = ''   # Holds the first name of a student entered by the user.
# student_last_name: str = ''    # Holds the last name of a student entered by the user.
# course_name: str = ''          # Holds the name of a course entered by the user.
# student_data: dict = {}        # one row of student data
# students: list = []            # a table of student data
# csv_data: str = ''             # Holds combined string data separated by a comma.
# json_data: str = ''            # Holds combined string data in a json format.
# file = None                    # Holds a reference to an opened file.
# menu_choice: str               # Hold the choice made by the user.

```

Figure 1: Commenting out all data variables and constants

Using code structure similar to Assignment05

The transition from the prior assignment's approach and structure to this current assignment was marked by a few stages. In Assignment05, I coded the processing of whatever task the script demanded in place: the conditional logic process, user interaction, and error handling shared the same space in order to meet the program's demands.

Below is a snippet, from Assignment05, for processing user input. In this snippet, I started with the logic processing, if the user entered "option 1", the option to register a student; in the program, I started by matching and verifying that the option the user entered was "1". Then, I use the try-except structure to ensure that the user does not enter inappropriate data. So, the presentation, processing and error handling all shared the same block of data — in this example, I used twenty two lines of code to correctly process the user input. (Figure 2)

```python

    # Input user data
    if menu_choice == "1":  # This will not work if it is an integer!
        try:  # trap user input errors
            student_first_name = input("Enter the student's first name: ")
            if not student_first_name.isalpha():
                raise ValueError("First name should only contain alphabetic characters!")
            student_last_name = input("Enter the student's last name: ")
            if not student_last_name.isalpha():
                raise ValueError("Last name should only contain alphabetic characters!")
            course_name = input("Please enter the name of the course: ")
            student_data = {"FirstName": student_first_name,
                            "LastName": student_last_name,
                            "CourseName": course_name}
            students.append(student_data)
            print(f"You have registered {student_data["FirstName"]} {student_data["LastName"]} for {student_data["CourseName"]}.")
        except ValueError as e:
            print(e)  # Prints custom message
            print("-- Technical Error Message --")
            print(e.__doc__)
            print(e.__str__())
        except Exception as e:
            print("There was a non-specific error!")
            print("-- Technical Error Message --")
            print(e, e.__doc__, type(e), sep='\n')
            print("-" * 50)

```

Figure 2 - inputting user data in Assignment05

Functions using Global Variables

In this week's Lab01, I learned how to code using Functions. I, essentially, coded the script to address the same challenges: present data to user; glean input from user; process user input; check for user input errors — but this time around, I used a function. However, the function itself got a little bigger than the original code from Assignment05, mostly owing to the definition of global variables at the head of the function. The first function in the Lab used global variables. Below is a snippet of this code showing the global variables used. (Figure 3)

```
91
92  def input_student_data():  1 usage
93      global student_first_name
94      global student_last_name
95      global student_gpa
96      global student_data
97      global students
98
99      try:
100         # Input the data
101         student_first_name = input("What is the student's first name? ")
102         if not student_first_name.isalpha():
103             raise ValueError("The first name should not contain numbers.")
104
105         student_last_name = input("What is the student's last name? ")
106         if not student_last_name.isalpha():
107             raise ValueError("The last name should not contain numbers.")
108
109         try:  # using a nested try block to capture when an input cannot be changed to a float
110             student_gpa = float(input("What is the student's GPA? "))
111         except ValueError:
112             raise ValueError("GPA must be a numeric value.")
113
```

Figure 3 - Function showing global variables

The function, including the global variables, user input handling, processing, and error handling within the same function got a little bigger than the code from Assignment05. Below is the bottom section of the function. The code, to essentially address the same challenge, had grown to thirty four lines, however, it was cleaner and the variables I was using were clearly defined at the beginning of the function. (Figure 4)

```
111         except ValueError:
112             raise ValueError("GPA must be a numeric value.")
113
114         student_data = {"FirstName": student_first_name,
115                         "LastName": student_last_name,
116                         "GPA": float(student_gpa)}
117         students.append(student_data)
118     except ValueError as e:
119         print(e)  # Prints the custom message
120         print("-- Technical Error Message -- ")
121         print(e.__doc__)
122         print(e.__str__())
123     except Exception as e:
124         print("There was a non-specific error!\n")
125         print("-- Technical Error Message -- ")
126         print(e, e.__doc__, type(e), sep='\n')
127
```

Figure 4 - the bottom section of the code

Functions using Parameters

In Lab02, I learned how to code using parameters in place of global variables. The use of parameters, in functions, is preferred over the use of global variables for a myriad of reasons: (External Site) (https://chat.chatbotapp.ai/chats/-OC7MEGpmc0DxcHgAvHP?model=gpt-3.5 , 2024) (Figure 5)

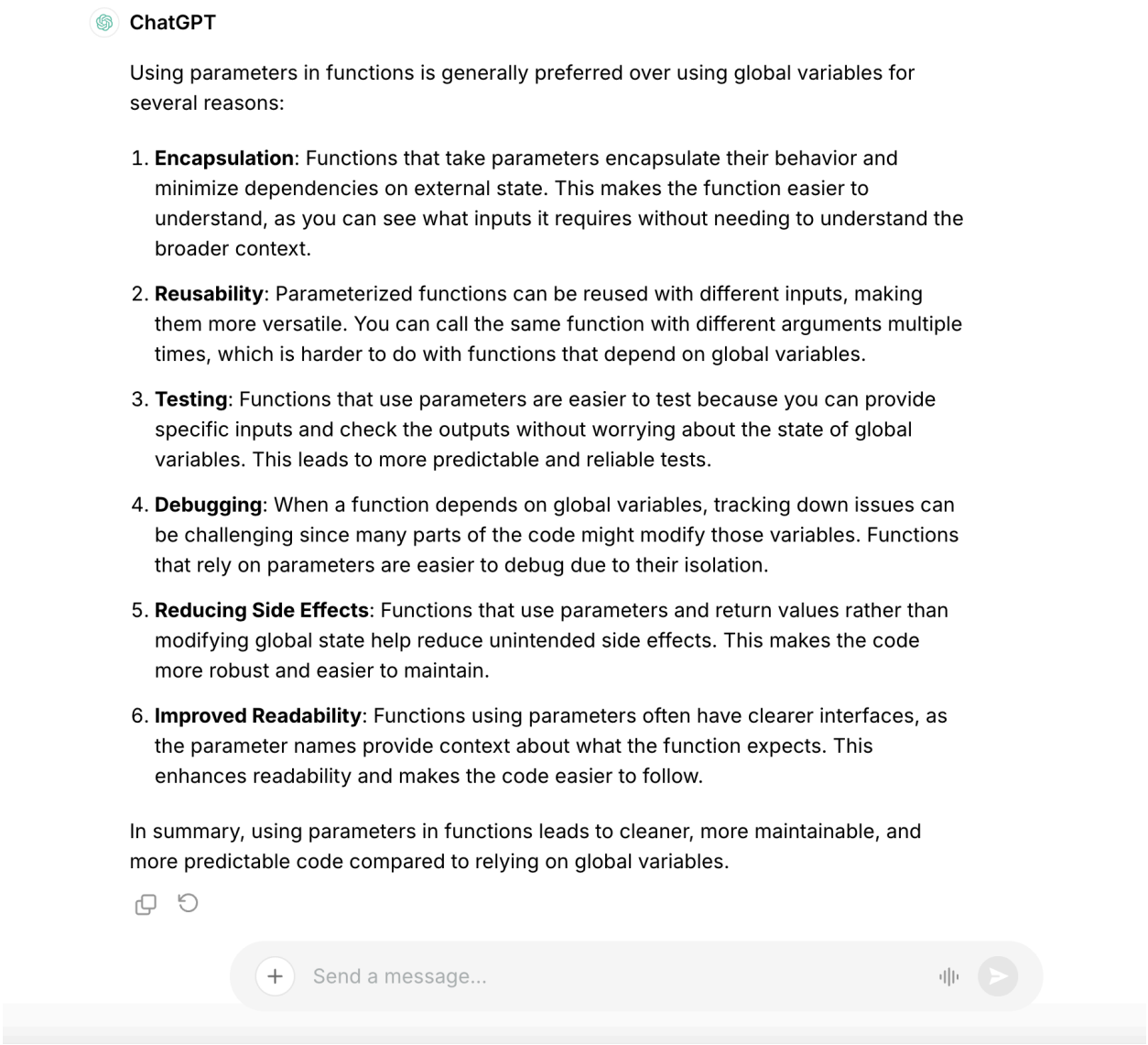
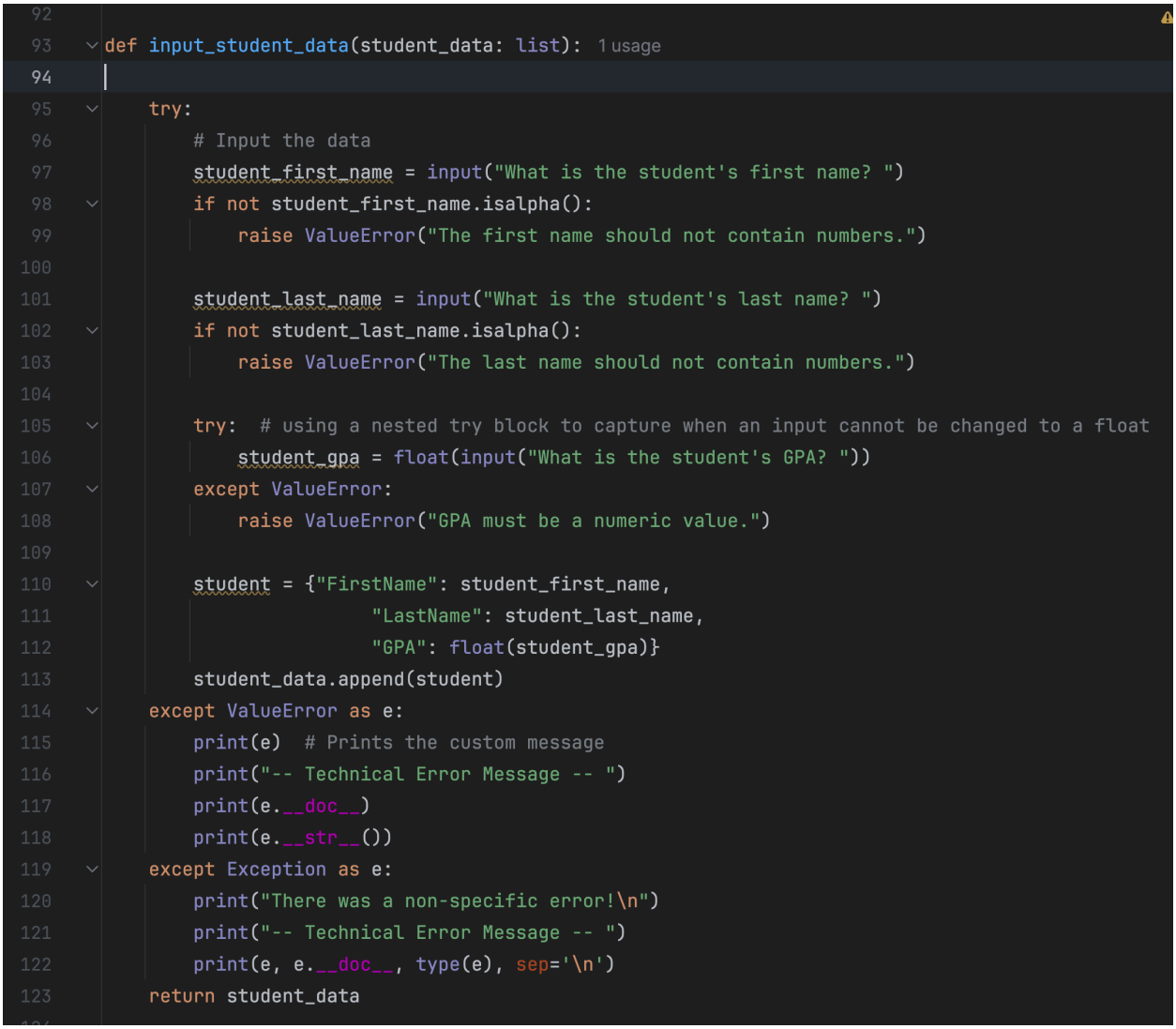


Figure 5 - why is a function using parameters better than one using global variables?

In Lab02, I learned how to code, the same function, but this time using parameters in place of global variables. The function, this time, just depended on one parameter. The most obvious aspect was the non-dependency on externals. The lack of dependency on externals, now, meant that I could reuse it and not have to rewrite the code. Learning how to write and use functions, this week, gave me the ability to think beyond the present task and start to think about the common themes: am I processing something, am I presenting something, or am I managing storage. Below, is the same function, now, using parameters in place of global variables.



```
92
93  def input_student_data(student_data: list):  1 usage
94
95      try:
96          # Input the data
97          student_first_name = input("What is the student's first name? ")
98          if not student_first_name.isalpha():
99              raise ValueError("The first name should not contain numbers.")
100
101         student_last_name = input("What is the student's last name? ")
102         if not student_last_name.isalpha():
103             raise ValueError("The last name should not contain numbers.")
104
105         try:  # using a nested try block to capture when an input cannot be changed to a float
106             student_gpa = float(input("What is the student's GPA? "))
107         except ValueError:
108             raise ValueError("GPA must be a numeric value.")
109
110         student = {"FirstName": student_first_name,
111                    "LastName": student_last_name,
112                    "GPA": float(student_gpa)}
113         student_data.append(student)
114     except ValueError as e:
115         print(e)  # Prints the custom message
116         print("-- Technical Error Message -- ")
117         print(e.__doc__)
118         print(e.__str__())
119     except Exception as e:
120         print("There was a non-specific error!\n")
121         print("-- Technical Error Message -- ")
122         print(e, e.__doc__, type(e), sep='\n')
123     return student_data
```

(Figure 6)

Figure 6 -Function using parameter in place of global variables

Classes and Separation of Concerns(SoC)

In Lab03, I learned how to use classes and the Separation of Concerns pattern. I set up two classes: FileProcessor and IO. In FileProcessor, I included two functions, both dealing with reading data from and writing or saving data to a file. The two functions were aptly named “read_data_from_file()” and “write_data_to_file()”. I defined the functions under a “static method” decorator. (External Site) (https://chat.chatbotapp.ai/chats/-OC7MEGpmc0DxcHgAvHP?model=gpt-3.5, 2024) (Figure 7)

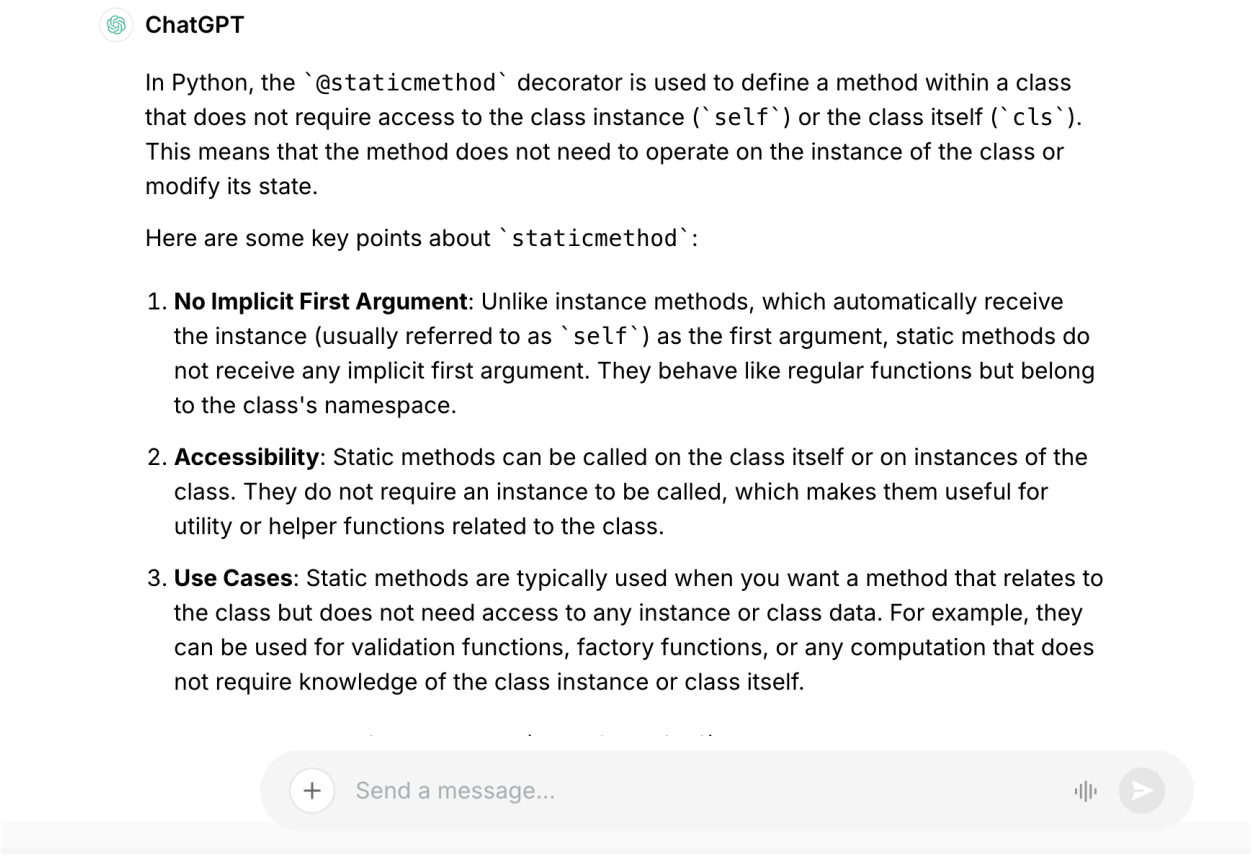


Figure 7 - what is a static method decorator?

The Separation of Concerns (SoC)

“*The Separation of Concerns (SoC) is a fundamental software design principle that aims to enhance the maintainability, scalability, and readability of code by breaking it down into distinct, self-contained components, each responsible for a specific aspect of the application's functionality. This pattern encourages modularity, reduces code complexity, and makes it easier*

to manage and extend software systems." (Randal Root, 2024, Introduction to Programming with Python: Module 6 - Functions, Page 33)
This week, I learned how to use a pattern called Separation of Concerns to organize the placement of functions within the classes. In lab03, I set up two classes, FileProcessor, and IO. In FileProcessor class, I placed my file processing functions: "read_data_from_file()" and "write_data_to_file()". Immediately below the file definition line, I included a document string — docstring, then the processing block of function code afterwards.
In the IO class, I included the rest of the other functions, which included all the presentation functions as well as the error handling function.

The concepts that helped me successfully complete this assignment are chronicled in the about heading named "The Transition". Although, I did not use global variables in this assignment, the Labs I worked on during the week kept introducing better methods and procedures usurping each previously introduced method.
In last weeks, assignment and labs, I was introduced to the try-except error handling structure. This week, I learned and started to use custom functions in place of the program structure I used last week. Within the realm of the functions, this week, I also transitioned from using global variables to using parameters and arguments to parameters; to using classes and grouping functions under classes; to understanding the pattern of Separation of Concerns (SoC) and these are the concepts I learned, that helped me successfully complete Assignment06.

Writing the Script

The FileProcessor class

Doing the Labs and reading the module notes significantly helped me in successfully completing the assignment. After running the program and finding no issues with it, I then commented out both the data variables and constants, as well as the rest of the body of the script that performed the processing and presentation.

The next task was to define the classes and functions. Below is the code snippet showing the task of defining the classes and functions. The snippet shows the FileProcessor class with a document string below the class definition. I temporarily uncommented the data variables and

constants in order to spot the type hints, but aware that I was not going to use them. I separated the “Processing” comment by two lines from the variables and constants definition. (Figure 8)

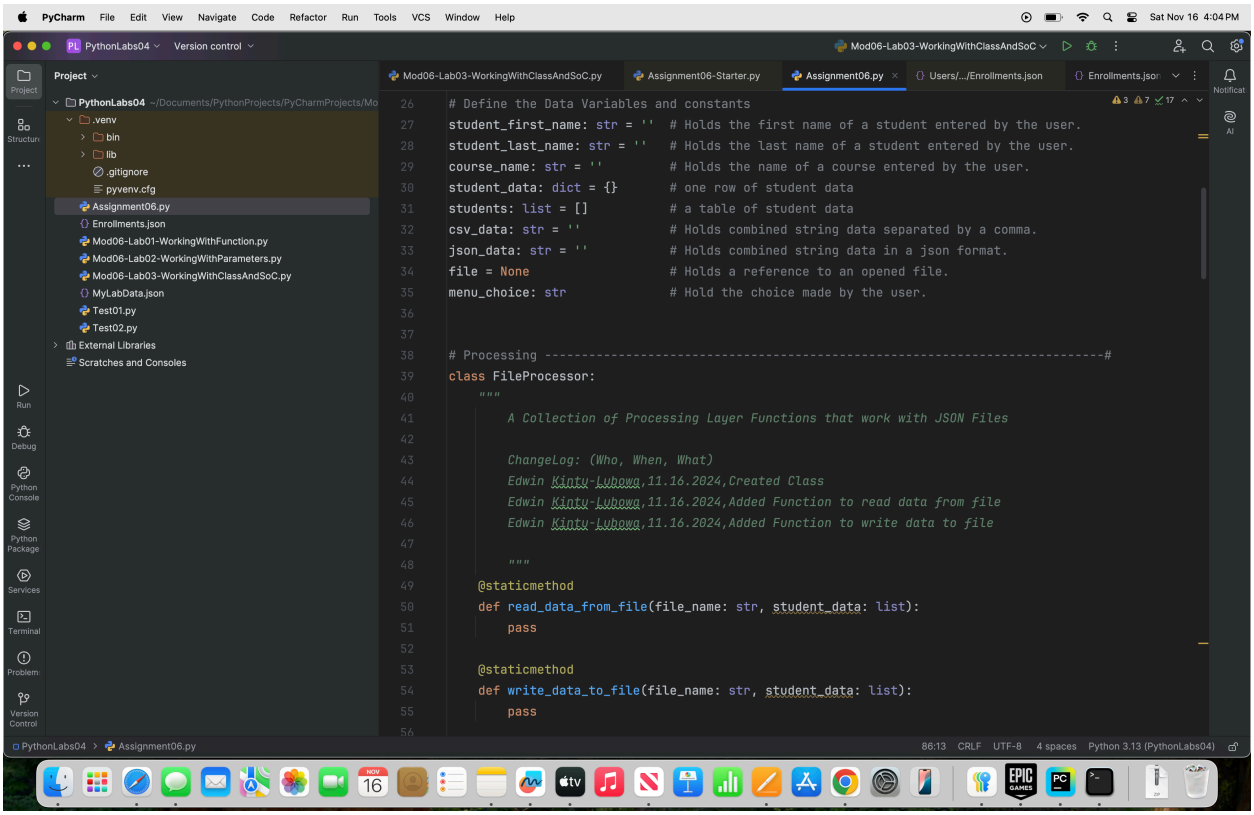


Figure 8 - Defining the FileProcessor class and file processing functions

---

The IO class

In the IO class, two more functions were introduced: one to process user input, and the other to output error messages. This was a significant development from the prior assignment’s code, because, now, I could reuse the functions and I did not have to include and extra “else” statement to process invalid menu choices, and I didn’t have to include extra lines of code to process error handling messages wherever I used the try-except structure.

To start the process, I defined all the functions within their respective classes, included the parameter names and each function preceded by the “@staticmethod” decorator. I included a brief description in the document string. Below, is a screen shot of the IO class alongside the docstring and the five presentation functions. (Figure 9)

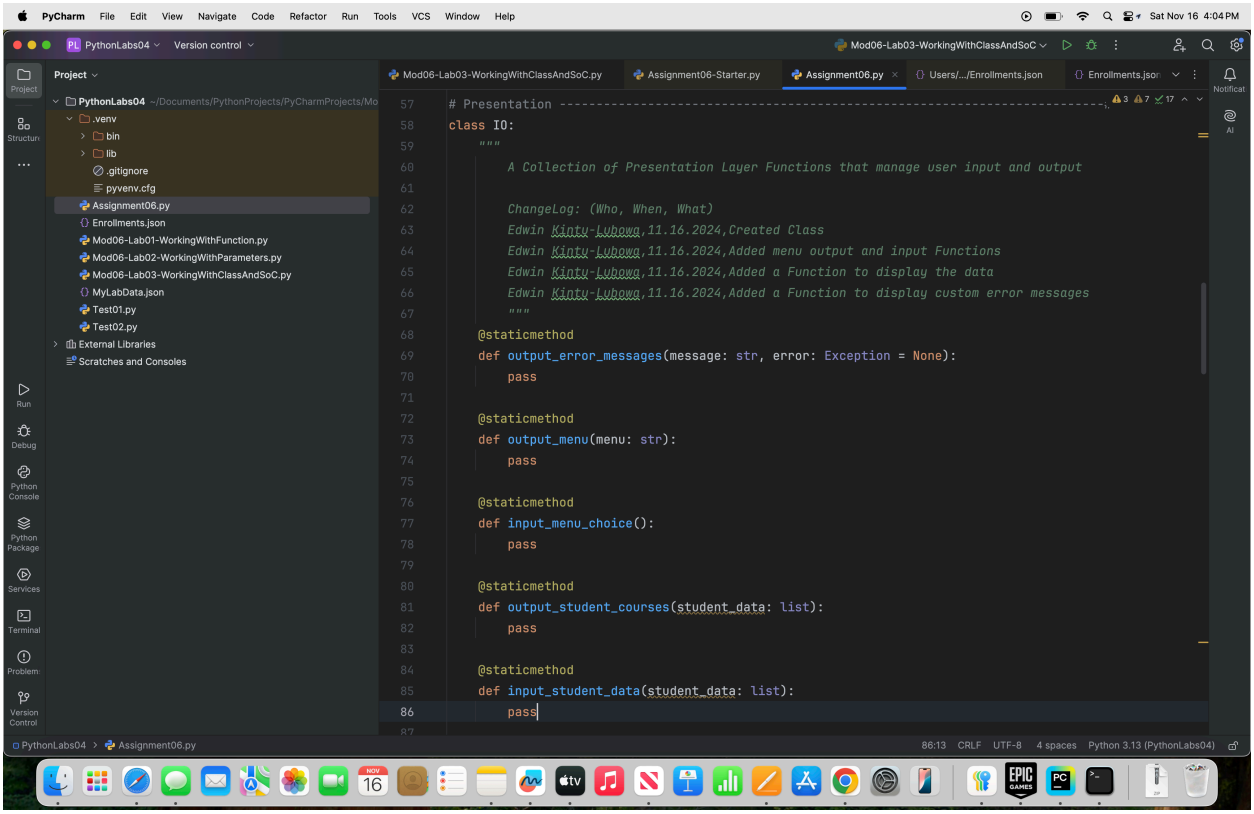


Figure 9 - Defining the IO class and the presentation functions

One function at a time

I used the same process for each function:

1. Copy the code from its section in the script and paste it into the function, with due attention to indentation.
2. Checked whether and how I needed to use the parameter(s) in the body of the function
3. Wrote a function call and then run a test
4. Removed extra lines of error checking code that were being handled by the “output_error_messages()” function.
5. After a successful run, I deleted the commented out code that I copied into the function.

Challenges

One of the many challenges I faced while working on the assignment was one of the parameter names “student_data” was defined as a dictionary in the “data and constants definitions” but was going to be used as a list in the assignment. This led to many a confused moment for me during the process; I updated the parameter name to “student_list”.

Once I commented out the variables and constants definitions that the prior assignment used, then I was able to use the rightful parameter name. I updated all the instances of “student_list” and run the program successfully. However, one review, when I hovered over the parameter name in one of the docstrings, I was pleasantly surprised that doctoring caught the error; I had updated the parameter name in the processing code sections, but not in the docstrings. Below is an error, at the top of the docstring, alerting me that function does not have a parameter “student_list” (Figure 10)

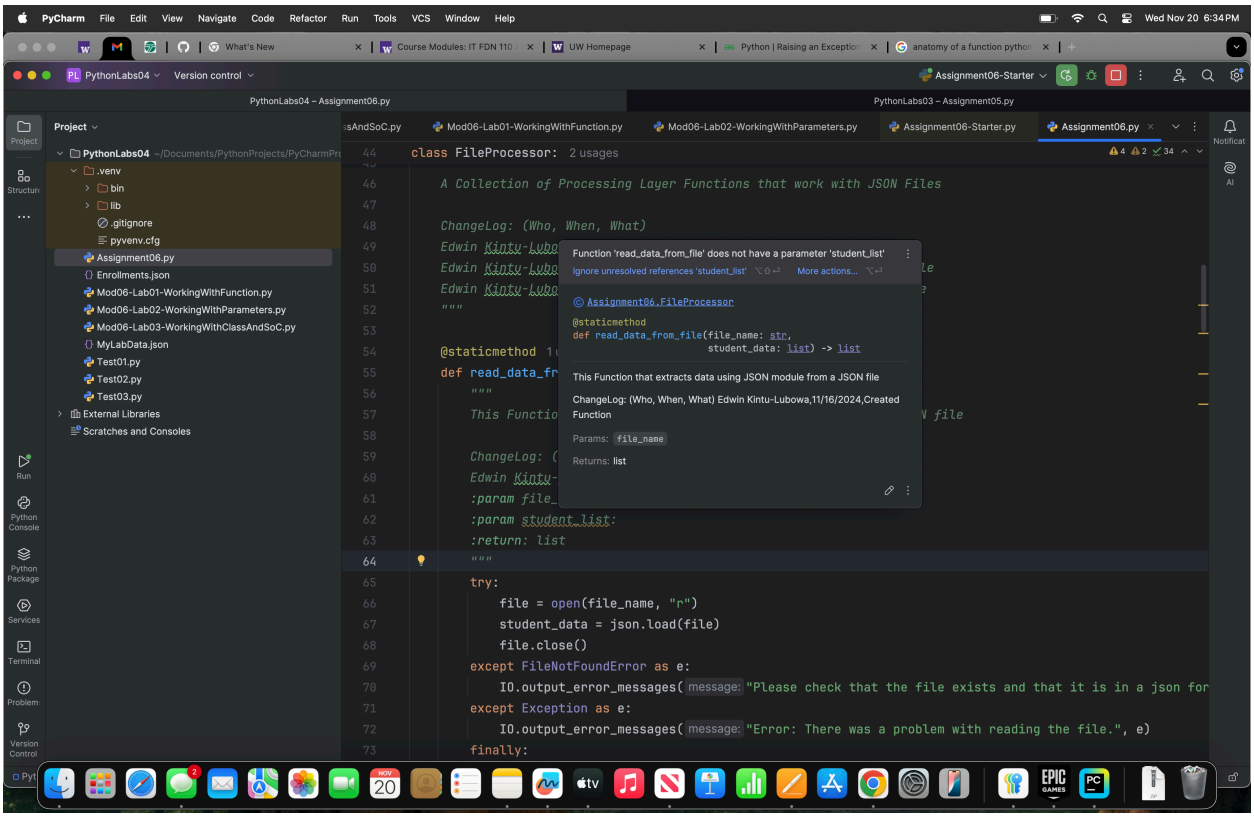


Figure 10 - ‘docstring’ message on parameter name mismatch

Summary

In this week’s assignment, I learned better ways to do everything compared to the prior weeks’ assignments. In the prior weeks, I learned how to take input from the user; process and display data. I learned how to use conditional logic statements to make the program more interactive with the user. In last week’s assignment, I learned how to handle errors to prevent both a catastrophic program crash, and enhance user experience. This week, I learned how to define and use functions; the advantages of using parameters in place of global variables. Using

classes and functions has given me a much better feel for how I think of approaching the task of writing an application.