Sid Tolbert
11/22/2024
IT FDN 110 A
Assignment06
https://github.com/Sid-Tolbert/IntroToProg-Python-Mod06

Assignment 6 – Functions

Introduction

Assignment 6 focused on improving code organization using functions and classes. The goal was to break down programs into smaller, reusable parts that are easier to understand and maintain. It emphasized the Separation of Concerns principle by dividing the code into layers for data handling, processing, and user interaction.

The assignment also introduced classes to group related functions and data, making the code more modular and structured. Key concepts like using parameters, return values, and error handling were included to create more flexible and reliable programs.

Doing the Assignment

To complete Assignment 6, I relied heavily on the guidance provided in Mod06_Notes**,** particularly the section on Lab 03: Working with Classes and the Separation of Concerns Pattern. This part of the notes was incredibly helpful for organizing my code into clear sections and ensuring it followed best practices.

I started by reviewing the Lab 03 instructions, which focused on breaking the program into three main layers: Data, Processing, and Presentation. Using this structure helped me think about how to divide my code logically.

```
# Title: Assignment06
# Desc: This assignment demonstrates using functions
# with structured error handling
# Change Log: (Who, When, What)
#   Sid Tolbert,11.24.2024,Created Script
#   <Sid Tolbert>,<11.24.2024>,<Functions>
# ------------------------------------------------------------------------------ #


#_____Import_Libraries_____
# Import required libraries
import json


#_____Constants_and_Variables_____
# 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.
--------------------------------------
'''

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

For example, I set up a FileProcessor class to handle all the file-related tasks, like reading and writing data, and an IO class for managing input and output. The notes provided a sample docstring format for classes, which I used to document what each class and its functions were supposed to do. This made my code more readable and organized.


```
#_____Processing_____#
2 usages
class FileProcessor:
    """

    A class of functions to process data input and output from files.

    ChangeLog: (Who, When, What)
    Sid Tolbert, 11.24.2024, FileProcessor class created


    """

    # When the program starts, read the file data into table
    # Extract the data from the file
    # Read from the Json file


    1 usage
```

Figure 2: FileProcessor Class creation

I liked how the editor had the built-in docstring function that auto-populated the function parameter descriptions and what was being returned.

The example in Lab 03 about separating functions into the right layers was particularly helpful when I was refactoring my code. I moved the read_data_from_file function into the FileProcessor class and used the @staticmethod decorator as shown in the notes. I also updated the error handling in this function to call a custom output_error_messages method from the IO class. This step really reinforced how separating concerns makes it easier to troubleshoot and maintain code. Instead of having error messages scattered throughout the script, they were centralized in one place.

```python
class FileProcessor:
    1 usage
    @staticmethod
    def read_data_from_file(file_name: str, student_data: list):
        """
        A function that reads the content of a JSON file into a list.
        There is error handling in case there isn't a JSON file in the working directory.

        ChangeLog: (Who, When, What)
        Sid Tolbert, 11.24.2024, read_data_from_file function created

        :parameter file_name: str = This string is the name of the file to be read from
        :parameter student_data: list = list that file will be read into
        :return: student_data
        """

        try:
            file = open(file_name, "r")
            student_data = json.load(file)
            file.close()
        except FileNotFoundError as e:
            IO.output_error_messages( message: "File not found!", e)
        except Exception as e:
            IO.output_error_messages( message: "An error occurred!", e)
        except Exception as e:
            print("Error: There was a problem with reading the file.")
            print("Please check that the file exists and that it is in a json format.")
            print("-- Technical Error Message -- ")
            print(e.__doc__)
            print(e.__str__())
        finally:
            if not file.closed:
                file.close()
        return student_data
```
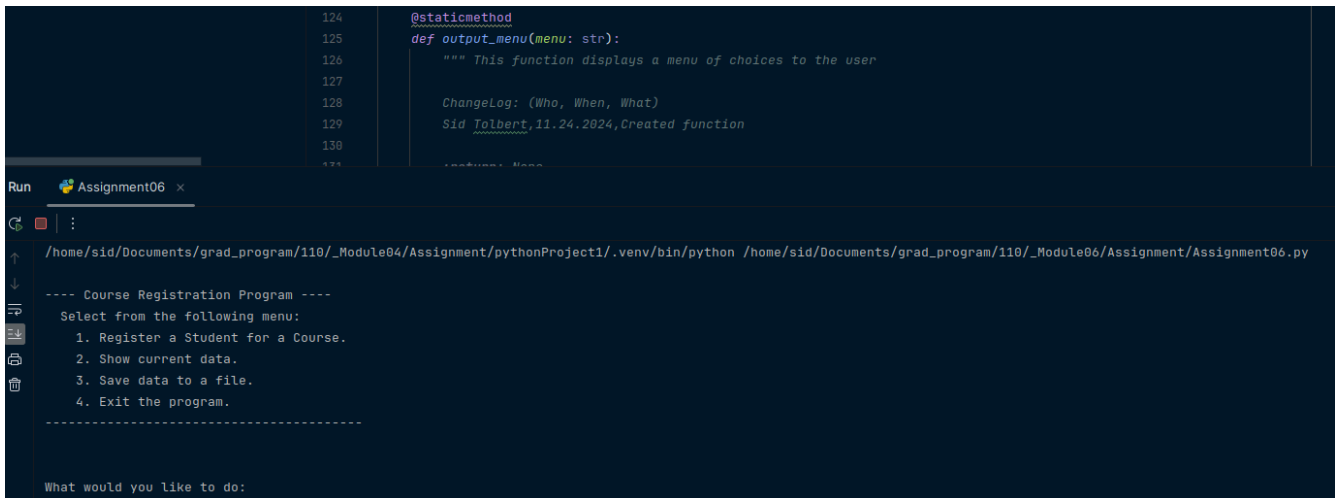
Figure 3: FileProcessor function with doscstring

I also followed the notes' advice on testing the code after each change. For instance, after moving output_menu into the IO class, I tested it to ensure it worked the same way as before. This approach of "one step at a time" made debugging much easier and prevented me from making too many changes at once. The lab also emphasized replacing global variables with parameters or local variables, which helped me avoid potential bugs.

Figure 4: Testing output_menu

The final piece of Lab 03 was running the program in a command-line terminal to ensure everything worked outside of PyCharm. This was a good reminder from the notes, as it's easy to forget that not all users will run the script in an IDE. I made sure to organize the final script so it could run smoothly in any environment.

Figure 5: Testing from the console

Conclusion

Assignment 6 focused on using functions and classes to create organized and efficient Python programs. Breaking the code into layers and applying the Separation of Concerns principle made the

scripts easier to read and debug. Using classes to group related functionality added structure and made the code more reusable.

This assignment reinforced the importance of clear code organization and introduced practices that are essential for writing scalable and professional programs.