Shannon Brown

February 18, 2024

IT FDN 110 A

Assignment 6

Functions, Classes, and SoC Pattern

Introduction

This week, we learned to create more concise, organized code with functions, classes, and the separation of concern (SoC) pattern. We expanded on our previous work which most recently focused on the use of dictionaries, lists, and error handling. We reviewed video modules, readings, and labs on these topics. The purpose of this paper is to outline how we expand on our previous program from Assignment05, but with the use of functions, classes, and the SoC pattern.

Creating a Python Script

The aim of Assignment 06 was to expand on script from Assignment 05 and reorganized our code into a more organized and user-friendly format with functions, classes, and the SoC pattern. The overall goal was to still allow users to select from a menu and accomplish the below goals:

(1) Prompt user to enter the student's first name, last name, and course name using input()
(2) Present a coma-separated string by formatting the collected data using print()
(3) Open and write the above collected data in a JSON file.
(4) End the program.

I opened Assignment06-Starter.py in PyCharm IDE, updated the header, and resaved the file as Assignment06.py. Next, I made sure the required constants (e.g., MENU, FILE_NAME) and variables (e.g., menu_choice) were defined, and I removed any unnecessary or redundant variables and constants. (Figure 1). Before proceeding, I also double-checked that JSON module build-in associated with python was at the top of my code (import json).

```
# Define the Data Constants
FILE_NAME: str = "Enrollments.json"
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 program's data
students: list = []
menu_choice: str = ''
```

Figure 1. Verify and adding data constants and variables needed for Assignment06

Next, the data file would need to be opened and written in, so I created the class FileProcessor. I employed the SoC pattern to keep my script outlined and organized, and I added details about my code purpose and structure (Figure 2). Since this code to open my file would be static, I used a @staticmethod decorator and declared a function that would read my data from my file. Functions allow you to break down large programs into smaller, more manageable pieces. After the function in the parenthetical, I set the type of my parameters, which are used to make the functions reusable. Lastly, to finalize this function, I copy and pasted the try-except code from Assignment05 which opened the JSON file. I went through and updated students to the parameter (student_data) and reduced the complexity of the error handling with an IO function, which I would be writing in a forthcoming step (Figure 2).

```
class FileProcessor:
    """
    A collection of processing layer functions that with JSON files

    ChangeLog: (Who, When, What)
    Shannon Brown, 2/18/2024, Created Class
    """
    @staticmethod
    def read_data_from_file(file_name: str, student_data: list):
        try:
            file = open(file_name, "r")
            student_data = json.load(file)
            file.close()
        except FileNotFoundError as e:
            IO.output_error_messages( message: "text file must exist before running this script!", e)
        except Exception as e:
            IO.output_error_messages( message: "There was a non-specific error!", e)
        finally:
            if file.closed == False:
                file.close()
        return student_data
```

Figure 2. Utilizing classes, functions, and the SoC pattern to open a JSON file

Within the class FileProcessor, I also added a function to write data into the file. I shuffle up the code which already existed in Assignment06-Starter.py, added parameters, and made sure they were updated throughout the function. I also simplified the error handling with an IO function (Figure 3).

```
@staticmethod
def write_data_to_file(file_name: str, student_data: list):
    try:
        file = open(file_name, "w")
        json.dump(student_data, file)
        file.close()
    except TypeError as e:
        IO.output_error_messages( message: "Please check the date is a valid JSON format", e)
    except Exception as e:
        IO.output_error_messages( message: "There was a non-specific error!", e)
    finally:
        if file.closed == False:
            file.close()
```

Figure 3. Adding a write data to file function to the class FileProcessor

A class IO was added to fulfil the remaining script criteria. I created the class and providing detail on the code using the SoC pattern (Figure 4). As I added functions, I added more detail to this background section.

```
class IO:
    """
    A collection of presentation layer functions that manage user input and output

    ChangeLog: (Who, When, What)
    Shannon Brown, 2/18/2024, Created Class
```

Figure 4. Creating a class IO and adding detail about the code using SoC pattern

Next, since I'd already references an error function in class FileProcessor, I wrote a function for printing error messages. All functions in this assignment are static so they began with @staticmethod. I relied on code written for Module06-Lab03 to draft the script for error handling. When IO.output_error_messages is now called, the message provided in the parenthetical will print (Figure 5).

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

    ChangeLog: (Who, When, What)
    Shannon Brown, 2/18/2024, Created function

    :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 5. Using a function within class IO to simply error handling

The following functions I added to class IO were created to open a menu and interpret the output entry. The code for the output_menu function was already in the Assignment06-Starter.py and was migrated with a parameter change. For input_menu_choice function, I utilized a try-except statement to ask the user for an input and then report an error if their input did not meet the established criteria (Figure 6).

The input_student_data function was created with code already in the document. No changes were made besides adding a parameter and changing the error messages to IO.output_error_messages (Figure 7). Lastly, in order to output the provided data, as requested by menu option #2, I created a final function under class IO called output_student_course, which used the print() function to display the student data including their first name, last name, and course name.

```python
@staticmethod
def output_menu(menu: str):
    """ This function displays the menu of choices to the user

        ChangeLog: (Who, When, What)
        Shannon Brown,2/18/2024,Created function

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

@staticmethod
def input_menu_choice():
    """ This function gets a menu choice from the user

        :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 e to avoid the technical message
    return choice
```

Figure 6. Adding functions to class IO to aid with menu display and menu choice

```python
@staticmethod
def input_student_data(student_data: list):
    """
    This function collects the first name, last name, and course name from the user

    ChangeLog: (Who, When, What)
    Shannon Brown, 2/18/2024,Created function

    :return: None
    """
    try:
        student_first_name = input("Enter the student's first name: ")
        if not student_first_name.isalpha():
            raise ValueError("The last 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_data = {"FirstName": student_first_name,
                        "LastName": student_last_name,
                        "CourseName": course_name}
        students.append(student_data)
        # print(f"You have registered {student_first_name} {student_last_name} for {course_name}.")
    except ValueError as e:
        IO.output_error_messages( message: "That value is not the correct type of data!", e)
    except Exception as e:
        IO.output_error_messages( message: "There was a non-specific error!", e)
    return student_data
```

Figure 7. Adding functions to class IO to aid with inputting student data

After utilizing classes and functions to organize a clear file processing layer and input/output layer, I wrote out a simplified version of my script referencing these tools. A while loop and if-elif function were still utilized, but the modular, maintainable main script remained clear (Figure 8).

```
# Beginning of the main body of this script
students = FileProcessor.read_data_from_file(file_name = FILE_NAME, student_data = students)

while True:
    IO.output_menu(menu = MENU)

    menu_choice = IO.input_menu_choice()

    if menu_choice == "1":
        IO.input_student_data(student_data = students)
        continue

    elif menu_choice == "2":
        IO.output_student_course(student_data = students)
        continue

    elif menu_choice == "3":
        FileProcessor.write_data_to_file(file_name = FILE_NAME, student_data = students)
        continue

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

Figure 8. Simplified main script which pulls directly from predefined classes and functions

Confirming Functionality of the Script

On PyCharm, I right clicked and then selected “Run Assignment06” to test my code throughout, and utilized debugging tools to fix any small errors, including missing syntax (e.g., quotes, parentheticals). Once the code was tested on PyCharm, I opened the command prompt to test the program in the terminal. To use python to run my created program, I typed “python.exe” followed by the Assignment06.py, after relocating to the program’s home directory. I confirmed the code was run correctly on the terminal, checked the JSON file for accuracy, and ensured the script met all parameters set by the acceptance criteria.

Summary

Assignment 06 employed previous programming tools such as dictionaries, lists, and error handling; however, we expanded on this work using functions, classes, and the SoC pattern to create more concise, reproducible code. All information required for the assignment was available via the module videos, readings, or labs. These tools will be further employed when I work with larger data sets in the future.