Erin Fox
11/20/2024
IT FDN 110
Assignment06
https://github.com/emfox55/IntroToProg-Python-Mod06

Functions

Introduction

In this assignment we were asked to expand on our prior work on JSON files, dictionaries, and exception handling by layering in functions and separation of concerns. In this document I will detail the steps I took to complete this task. All assignment instructions and informational materials that informed my approach can be found in the UW IT FDN 110 “Module06 - Functions” course materials.

Notes on Previous Documentation

The main functionality of this code (read data from a file, ask user for additional data, show user currently stored data, save all data back to file, JSON files and error handling) was established in Assignments 04 and 05, so I focus primarily on the new methods here. Please refer to the previous assignment documents for more detail. All development and testing was completed using the PyCharm IDE.

Functions

To achieve optimal modularity and reusability, all parts of the code that perform a task, especially repeat tasks like error handling/throwing exceptions, were organized into functions. This allows us to “call” the function using the name of the function, and to pass data (arguments) to it using the parameters required or optional to run the code (***Mod06 Notes - Functions & Parameters***). Docstrings are used to help any reader understand the intention of each function, and how to use it (***Mod06 Notes - Docstrings***).

Figure 1: Docstrings.

```
120     @staticmethod  1 usage
121     def input_menu_choice(menu: str) -> str:
122         """ This function gets a menu choice from the user
123
124         ChangeLog: (Who, When, What)
125         EFox,11/19/2024,Created function
126
127         :return: string with the user's choice
128         """
129         menu_choice = input("What would you like to do: ")
130         while menu_choice not in ['1','2','3','4']:
131             IO.output_error_messages('Please enter a number between 1 and 4.')
132             menu_choice = input("What would you like to do: ")
133         return menu_choice
134
```

Not only is it more modular and reusable, it improves the readability of the code later when the different functions are called based on different menu options.

Figure 2: Improved readability using functions.

```
# Present and Process the data
while True:
    # Present the menu of choices
    print(MENU)
    # Store user menu choice
    menu_choice = IO.input_menu_choice(menu = MENU)

    if menu_choice == '1':
        # Input user data
        IO.input_student_data(student_data = students)
        continue

    elif menu_choice == '2':
        # Present the current data
        IO.output_student_courses(student_data = students)
        continue

    elif menu_choice == '3':
        # Save the data to a file
        FileProcessor.write_data_to_file(file_name = FILE_NAME, student_data = students)
        continue
```

Separation of Concerns

Separation of concerns is a method of organizing code into three sections (Data, Processing, and Presentation), for optimal comprehension and maintainability. To make sure the code follows this concept, I organized the functions responsible for reading data from and writing data to the JSON file into a class called “FileProcessor” (Processing), and all the functions governing input from or output to the user in a class called “IO” (Presentation). The Data section houses the constant and variable definitions (***Mod06 Notes - Organizing code based on Concerns***). As with functions, all classes are documented using docstrings.

Error Handling Update on Assignment05 Logic

In the error handling from Assignment05, I added an Exception that uses the built-in isalpha() method to try and disallow users from entering numerical values in the first or last name variables. However this time, when I tried testing a valid first name with a space in it, Mary Ann, it threw an error because a space is not an alpha character. I decided not to use isalpha(), and find something more specific to numeric characters instead. There is no built-in method I could find that checks if *any* character in a string is numeric, so I created a custom function that loops through each character in the user entry and applies the isnumeric() method to check each one.

Figure 3: New function to disallow numeric characters

```python
    @staticmethod  2 usages
    def has_numeric(input_string: str):
        """ This function loops through each character in a string and returns a boolean value of True
        if any character is numeric

        ChangeLog: (Who, When, What)
        EFox,11/19/2024,Created Function

        :return: Boolean
        """
        for char in input_string:
            if char.isnumeric():
                return True
        return False
```

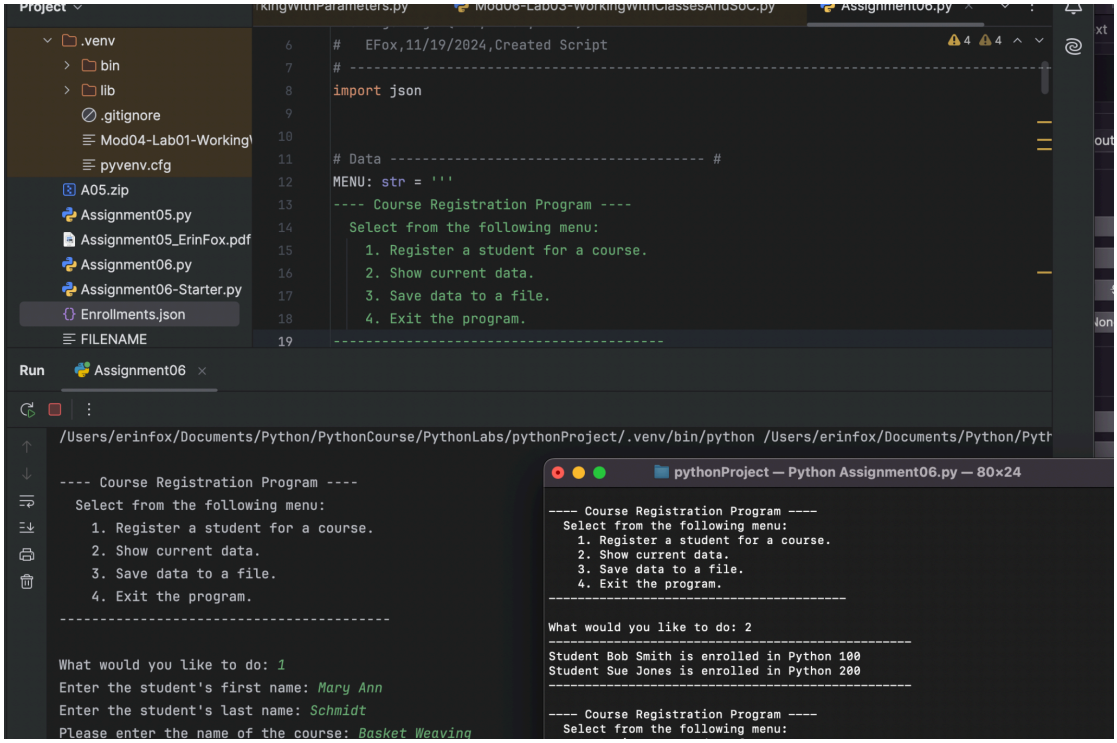
```python
try:
    student_first_name = input("Enter the student's first name: ")
    if IO.has_numeric(student_first_name):
        raise ValueError("The first name should not contain numbers.")
    student_last_name = input("Enter the student's last name: ")
    if IO.has_numeric(student_last_name):
        raise ValueError("The last name should not contain numbers.")
```

I used ChatGPT to research the best way to approach this.

Testing

I was able to successfully run this code in both PyCharm and Terminal

Figure 4: Testing code



Summary

Using the documents and videos provided in the Module06 course materials, I was able to understand and display how to use functions and classes to achieve modularity, reusability and separation of concerns. I was able to validate the accuracy of my script by running it in both IDLE and Mac Terminal.