Richard Eidswick

11/17/2025

Foundations of Programming: Python - IT FDN 110 A

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

Assignment 06

Introduction

For this week's assignment, I continued to build on the Python program that created an interactive registration menu that outputs information to a .json file. The key difference between the previous version was the introduction of working with classes, functions, and the "Separation of Concerns". After getting comfortable with these new topics, I used them to develop the program and test it in both PyCharm and the command prompt.

New Topics

Classes and Functions

This assignment introduced Functions and Classes. Functions are repeatable sets of code. They can be standalone, take in inputs, and return outputs. Classes are used to create objects, group functions, and help build modular code that can be reused. Both functions and classes increase the readability and functionality of a program. An important part of functions and classes are the Document Strings or Docstrings. Docstrings are comments that help the programmer communicate the intended use of the function or class, the variables, the output, and the changelog.

Separation of Concerns

Separation of Concerns is a design principle to group similar aspects of code. This helps make code more modular and readable. It breaks down into three common types: Presentation concern, Logic concern, and Data Storage concern.

Program

This assignment used an existing starter file that had most of the necessary variables, including the "MENU" global constant. With the introduction of methods and functions, some of the variables became unnecessary and were consumed internal to functions. This helped minimize the amount of variable called (Figure 1).

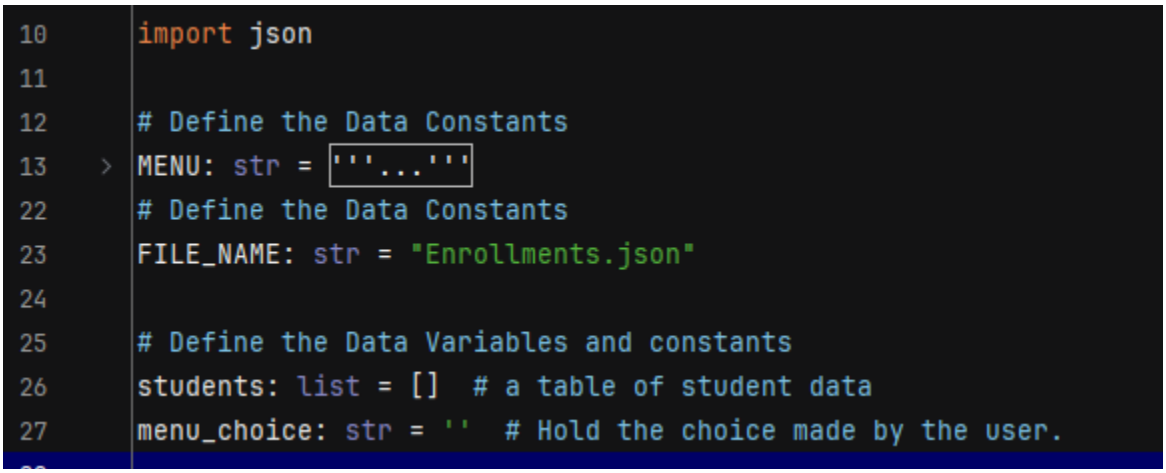

Figure 1: Globals and Variables.

The next part was to build two classes – one for the file processing, FileProcessor, and one for managing user inputs and outputs, IO. The FileProcessor class had the functions for reading from a file and writing to a file (Figure 2).

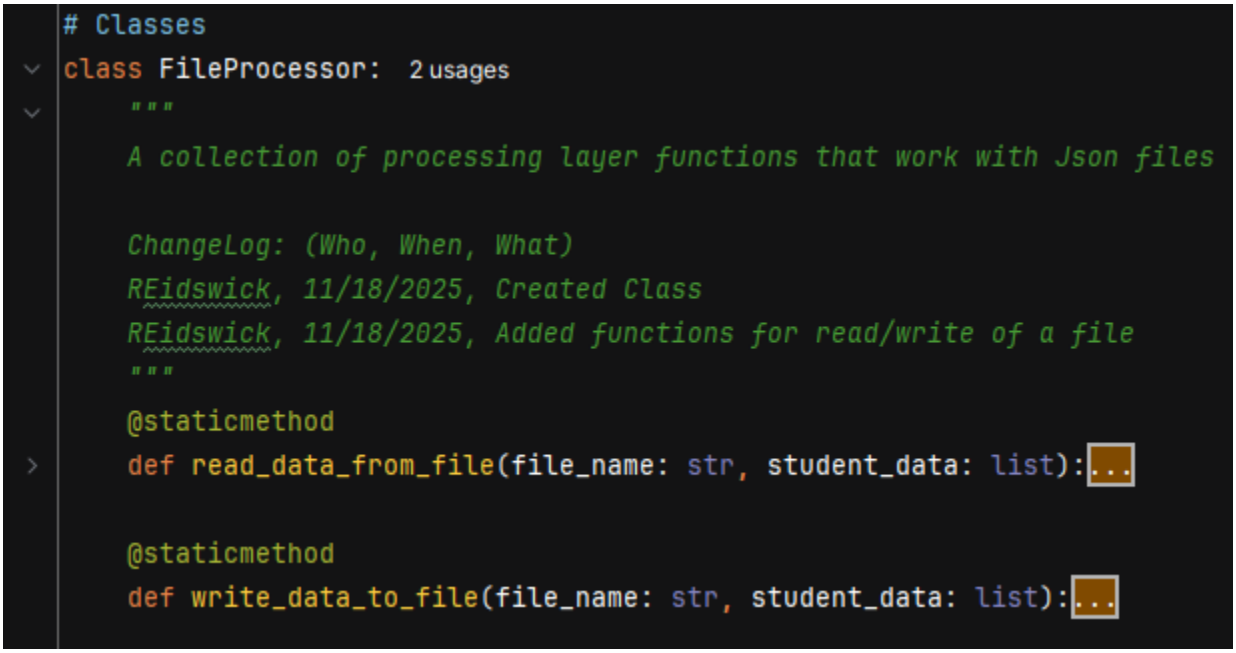

Figure 2: FileProcessor functions.

Both of these functions took in the *file_name* and *student_data* parameters that were necessary to pass information into the functions. The read function had a return for the student data that was held in the file to display it during user interaction and build on it to later write it in the file. The write function does not need a return since it is writing information to the .json file and no additional information was needed to pull out for later use. (Figure 3).

```python
    @staticmethod
    def read_data_from_file(file_name: str, student_data: list):
        """ Function that reads data from json files

        :param file_name: name of the json file
        :param student_data: empty list
        :return: student first name, student last name, and course name from json file
        """
        try:
            file = open(file_name, "r")
            student_data = json.load(file)
        except FileNotFoundError as e:
            IO.output_error_message( message: "File must be created first. Creating the file...", e)
            file = open(file_name, "w")
            json.dump(student_data, file, indent=4)
        except Exception as e:
            IO.output_error_message( message: "There was a non-specific error when reading the file.", e)
            pass
        finally:
            if file is not None:
                file.close()

        return student_data
```

Figure 3: Read function.

The IO class held five functions: outputting error messages, outputting the menu selection table, the input selection from the user, the output of student data, and the user input for student data (Figure 4).

```python
class IO:  11 usages
    """
    A  collection of presentation layer functions that manage user
    input and output

    ChangeLog: (Who, When, What)
    REidswick, 11.18.2025, Created Class
    REidswick, 11.18.2025, Added menu output and input functions
    REidswick, 11.18.2025, Added a function to display the data
    REidswick, 11.18.2025, Added a function to display custom error messages
    """
    @staticmethod  7 usages
    def output_error_message(message: str, error: Exception = None):...

    @staticmethod  1 usage
    def output_menu(menu: str):...

    @staticmethod  1 usage
    def input_menu_choice():...

    @staticmethod  1 usage
    def output_student_courses(student_data: list):...

    @staticmethod  1 usage
    def input_student_data(student_data: list):...
```

Figure 4: IO Class

The function to output error messages took in the message to the user and the exception. It printed out the error message with the technical error message (Figure 5).

```
92      @staticmethod  7 usages
93      def output_error_message(message: str, error: Exception = None):
94          """ Creates an error message to handle exceptions.
95
96          :param message: message to display to the user
97          :param error: Exception type
98          :return: None
99          """
100         print("="*50)
101         print(message, end="\n\n")
102         if error is not None:
103             print("-- Technical Error Message -- ")
104             print(error, error.__doc__, type(error), sep='\n')
```

Figure 5: Output Error Messages

The function for displaying the menu took in a string parameter that used the “MENU” global variable. This was to make it easy to change the menu if needed (Figure 6).

```
106     @staticmethod  1 usage
107     def output_menu(menu: str):
108         """Print menu for user to see
109
110         :param menu: menu to print
111         :return: None
112         """
113         print()
114         print(menu)
115         print()
```

Figure 6: Display menu function

The function to control the input from the user for the menu selection did not take in a parameter but returned the selection choice into the main body of the code. This function also used error handling to simplify the “if” options in the while loop (Figure 7).

```python
    @staticmethod  1 usage
    def input_menu_choice():
        """ Function that prompts the user to select from the menu.

        :return: menu choice
        """
        choice = "0"
        try:
            choice = input("Please select from 1,2,3, or 4: ")
            if choice not in ("1", "2", "3", "4"):
                raise Exception("Please select from 1,2,3, or 4")
        except Exception as e:
            IO.output_error_message(e.__str__())
        return choice
```

Figure 7: User input for menu selection.

Next, the function that handled the display of current student courses took in the current list of student data and printed it (Figure 8).

```python
    @staticmethod  1 usage
    def output_student_courses(student_data: list):
        """ Function that prints student courses

        :param student_data: list of student first name, last name, and course
        :return: None
        """

        print("-" * 50)
        for student in student_data:
            print(f'Student {student["FirstName"]} '
                  f'{student["LastName"]} is enrolled in {student["CourseName"]}')
        print("-" * 50)
```

Figure 8: Output current student data.

The last function handled the user inputs when the user needed to add new student data. It took in the current list, appended to the list, and returned it. It also implemented error handling to avoid any non-alphabetic characters in first or last names (Figure 9).

```python
    @staticmethod  1 usage
    def input_student_data(student_data: list):
        """ Function that adds new student data to the list.

        :param student_data: current student data
        :return: updated student data
        """
        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: ")
            new_student_data = {"FirstName": student_first_name,
                                "LastName": student_last_name,
                                "CourseName": course_name}
            student_data.append(new_student_data)
            print(f"You have registered {student_first_name} {student_last_name} for {course_name}.")
        except ValueError as e:
            IO.output_error_message( message: "There was a ValueError exception while reading the data.", e)
        except Exception as e:
            IO.output_error_message( message: "There was a ValueError exception while reading the data.", e)
        return student_data
```

Figure 9: User inputs for student data.

With all the classes and functions defined, it made the main body of the code simple and easy to read (Figure 10).

```python
# Main
students = FileProcessor.read_data_from_file(FILE_NAME, students)

# Present and Process the data
while (True):

    # Present the menu of choices
    IO.output_menu(MENU)
    menu_choice = IO.input_menu_choice()

    # Input user data
    if menu_choice == "1":  # This will not work if it is an integer!
        students = IO.input_student_data(students)
        continue

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

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

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

print("Program Ended")
```

Figure 10: Main body of the code.

The last part was to test the code. The test consisted of running the code -> showing current data -> adding a new student -> saving to a file -> displaying the .json file (Figures 11-13).

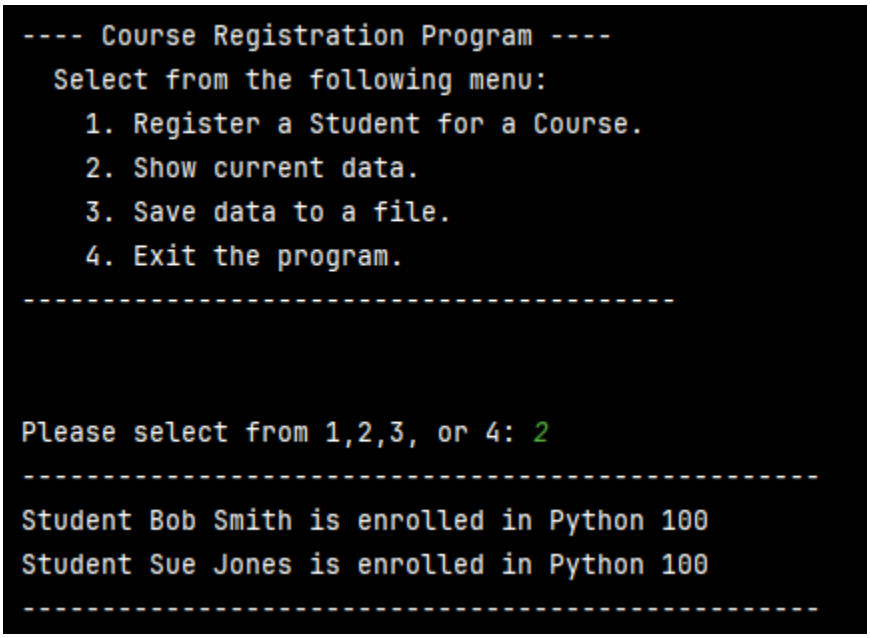

Figure 11: User selection 2.

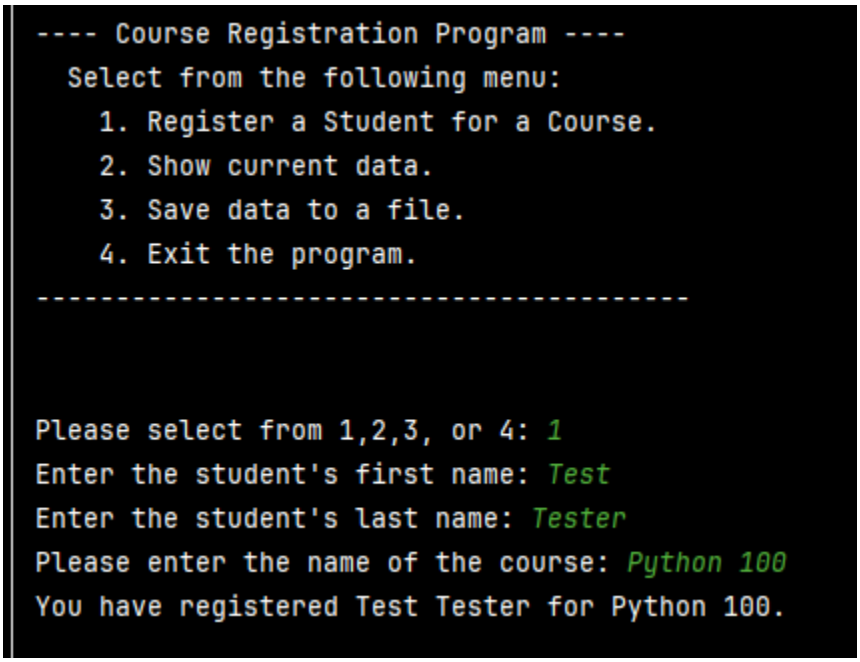

Figure 12: Adding in a new user.

M6_L3.py Assignment06.py Enrollments.json

```
[
    {
        "FirstName": "Bob",
        "LastName": "Smith",
        "CourseName": "Python 100"
    },
    {
        "FirstName": "Sue",
        "LastName": "Jones",
        "CourseName": "Python 100"
    },
    {
        "FirstName": "Test",
        "LastName": "Tester",
        "CourseName": "Python 100"
    }
]
```

Figure 13: Reviewing the .json file after saving.

Following this, I tested to verify it worked in the command prompt (Figure 14).

```
C:\Users\reids\Documents\Career\School\Python 110\Module 6>Python Assignment06.py


---- 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 select from 1,2,3, or 4: 2
-------------------------------------------------
Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Test Tester is enrolled in Python 100
-------------------------------------------------


---- 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 select from 1,2,3, or 4: 4
Program Ended

C:\Users\reids\Documents\Career\School\Python 110\Module 6>
```

Figure 14: Testing in Command Prompt.

Summary

In this assignment, I went through the process of adding classes and functions into the program to enhance modularity, readability, and scalability. I combined this with the previous error handling to make a robust program It ran successfully in both PyCharm and command prompt.