Walden Marcus

03/04/2025

IT FDN 110 A

Assignment 06

“Assignment06.py”

Link to Github Repository

Introduction

“Assignment06.py” is a program designed to read student registration information from a JSON file, receive more information, and write it back to the same file. In comparison to previous assignments, “Assignment06.py” utilizes classes and functions to organize its structure. As well, implements error handling and considers separation of concerns.

Packages, Constants, and Variables

There is one package requisite to the function of “Assignment06.py”: json, which must be imported for proper manipulation of “Enrollments.JSON”. There are two data constants: *MENU*, which contains the strings to display options to the user, and *FILE_NAME*, which holds the name of “Enrollments.json”. There are two global variables in “Assignment06.py”, as other variables are contained locally within functions; They are *menu_choice* (which holds user input) and *students* (which holds student data in tabular format).

```
#PACKAGES -------
import json
# 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"
# VARIABLES --------
menu_choice: str  #holds user choice from MENU
students: list = [] #list of dictionaries containing student registration information
```

Fig. 1 The packages, constants, and global variables of “Assignment06.py”.

Presentation

The methods associated with user experience and program presentation together make the IO class. There are five methods contained within the IO class: *output_error_messages(), output_menu(), input_menu_choice(), output_student_courses(),* and *input_student_data()*. *Output_menu()* prints the data passed through its single parameter. *Input_menu_choice()* returns user input to the global variable *menu_choice*. *Input_student_data()* requests and receives student registration information from the user, then appends it to the global variable *students*. Lastly, *output_error_messages()* adds a technical error message when called upon during error handling.

```
#PRESENTATION --------

class IO:

    @staticmethod

    def output_error_messages(message: str, error: Exception = None):

        """

        This function displays a technical error message if an exception is raised.

        Change Log:

        WMarcus, 3/2/25, Created Function

        """

        print(message, end="\n\n")

        if error is not None:

            print("-- Technical Error Message -- ")

            print(error, error.__doc__, type(error), sep='\n')
```

Fig. 2 The method *IO.output_error_messages*. This method displays a technical error message, and is called by other methods in this program. Notice how this method, like all other methods in this program, utilizes @staticmethod to be called without being assigned to an object.

```
    @staticmethod

    def output_menu(menu: str):

        """

        This function prints the data passed through its parameter.

        Change Log:

        WMarcus, 3/2/25, Created Function

        """

        print(menu)

    @staticmethod

    def input_menu_choice():

        """

        This function receives the user's choice from MENU, and addresses invalid choices.

        return: string with user choice
```

```
        Change Log:
        WMarcus, 3/2/25, Created Function
        """
        global menu_choice
        try:
            menu_choice = input("Enter your menu choice number: ")
            if menu_choice not in ("1", "2", "3", "4"):
                raise Exception("Please, choose only 1, 2, 3, or 4")
        except Exception as e:
            IO.output_error_messages(e.__str__())
        return menu_choice
```

Fig. 3 *IO.output_menu() and IO.input_menu_choice* both utilize global variables, which are declared after the docustrings.

```
    @staticmethod
    def output_student_courses(student_data: list):
        """
         This function displays all the student data, whether saved to file or awaiting writing to
file.
        Change Log:
        WMarcus, 3/2/25, Created Function
        """
        print("-" * 50)
        for student in student_data:
            print(f'Student {student["FirstName"]} '
                f'{student["LastName"]} is enrolled in {student["CourseName"]}')
            print("-" * 50)
    @staticmethod
    def input_student_data(student_data:list):
        """
        This function requests and receives user input about student registration.
        return: A nested list containing user input
        Change Log:
        WMarcus, 3/2/25, Created Function
        """
        student_first_name: str = ''
        student_last_name: str = ''
        course_name: str = ''
```

```
        student_data: dict = {}
        global students
        try:
            student_first_name = input("Enter the student's first name: ")
            if not student_first_name.isalpha():
                raise ValueError()
            student_last_name = input("Enter the student's last name: ")
            if not student_last_name.isalpha():
                raise ValueError()
            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("Your entry has been rejected. \
            Student names may only contain letters.", e)
        except Exception as e:
            IO.output_error_messages("Your entry has been rejected. \
            Student names may only contain letters.", e)
        return student_data
```

Fig. 4 *IO.output_student_courses* and *IO.input_student_data* respectively print and collect data contained in a list of dictionaries. *IO.input_student_data* raises errors when entered information contains a non-letter entry, like a number or a character.

Data Storage

The methods associated with file manipulation and data storage together make the FileProcessor class. Two methods are contained with the FileProcessor class: *read_data_from_file()* and *write_data_to_file()*. *Read_data_from_file()* reads from the designated JSON file into a two-dimensional list of dictionaries. *Write_data_to_file()* writes a two-dimensional list of dictionaries to a JSON file. Both of these methods depend on features of the *json* package.

```
#DATA STORAGE -------
class FileProcessor:
    @staticmethod
    def read_data_from_file(file_name: str, student_data: list):
        """
        This function reads data from the JSON file into a list of dictionary rows.
```

```
        return: A list of dictionaries
        Change Log:
        WMarcus, 3/2/25, Created Function
        """
        global students
        try:
            file = open(file_name, "r")
            student_data = json.load(file)
            file.close()
        except FileNotFoundError as e:
            IO.output_error_messages("Text file must exist before running this script!", e)
        except Exception as e:
            IO.output_error_messages("There was a non-specific error!", e)
        finally:
            if not file.closed:
                file.close()
        return student_data
    @staticmethod
    def write_data_to_file(file_name:str, student_data:list):
        """
        This function writes a list of dictionaries to a JSON, then prints the data that was written.
        Change Log:
        WMarcus, 3/2/25, Created Function
        """
        file = None  # Holds a reference to an opened file.
        global students
        try:
            file = open(file_name, "w")
            json.dump(student_data, file)
            file.close()
            print("The following data was saved to file!")
            for student in student_data:
                    print(f'Student {student["FirstName"]} {student["LastName"]} is enrolled in
{student["CourseName"]}')
        except FileNotFoundError as e:
                IO.output_error_messages("Text file must exist before running this script!", e)
        except Exception as e:
```

```
            IO.output_error_messages("There was a non-specific error!", e)
        finally:
            if not file.closed:
                file.close()
```

Fig. 5 *FileProcessor.read_data_from_file* and *FileProcessor.write_data_to_file* interact with a JSON file. *FileProcessor.read_data_from_file* is called when "Assignment06.py" is initialized, whereas *FileProcessor.write_data_to_file* is called based on user input (see "Logic").

Logic

The logic of "Assignment06.py" is centered on a while loop that calls various functions depending on user input to *IO.input_menu_choice()*. Take heed to how parameters are specified within the logic of the program: *MENU* is passed into *IO.output_menu*, and *students* is passed into *FileProcessor.read_data_from_file(), IO.input_student_data*, *IO.output_student_courses,* and *FileProcessor.write_data_to_file().* In *FileProcessor.write_data_to_*file, the global constant *FILE_*NAME is passed into the function. Before the loop begins, "Assignment06" reads "Enrollments.json" and writes the data to *students*.

```
#LOGIC---------
students = FileProcessor.read_data_from_file(file_name=FILE_NAME, student_data=students)
while True:
    IO.output_menu(menu = MENU)
    IO.input_menu_choice()
            # Input user data
    if menu_choice == "1":
        IO.input_student_data(student_data = students)
        continue
            # Present the current data
    elif menu_choice == "2":
        IO.output_student_courses(student_data = students)
        continue
            # Save the data to a file
    elif menu_choice == "3":
        FileProcessor.write_data_to_file(file_name = FILE_NAME, student_data = students)
        continue
            # Stop the loop
    elif menu_choice == "4":
            break
print("Program Ended")
```

Fig. 6 The logic of "Assignment06.py" relies on a while loop. Notice how the loop-level activity, such as *continue* and *break* exist within the concern of logic.

Summary

"Assignment06.py", under expected use cases, reads from a JSON file, and can write multiple additional entries to the same file. Multiple opportunities of error handling ensure proper data entry and offer minor troubleshooting. Future directions of development for this program include data editing, row deletion, and JSON renaming options.