Ryan Seng

5/20/2024

Python 100

Assignment 06

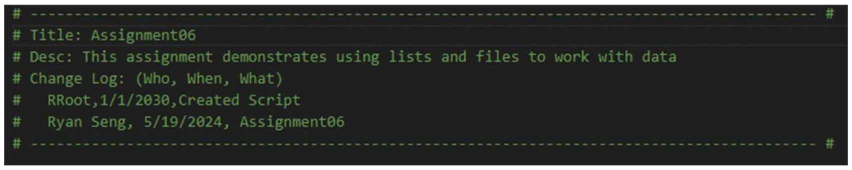
Assignment 06 – Functions

Introduction

This document will outline how I created the assignment06.py script to turn in for the assignment. To create this code, I referenced the lessons learned in the class on 5/15/2024, Assignment06-Starter.py, and the Lab answer scripts for Lab 3. The methodology section below goes into further detail.

Methodology

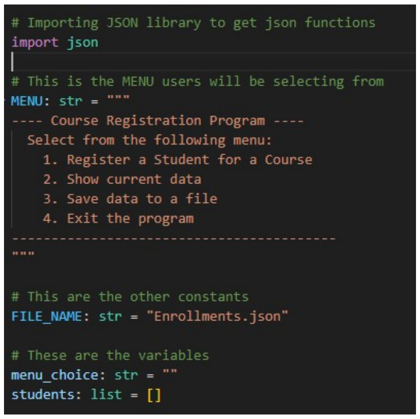
The comments for the assignment were taken straight from my submission for Assignment 05 but I changed <Your Name Here>, <Date>, and Assignment to my name, Ryan Seng, 5/19/2024, and Assignment06 which was when I initially started working on the assignment. The snip below contains the text I was referring to:



```
# ------------------------------------------------------------------------------------------ #
# Title: Assignment06
# Desc: This assignment demonstrates using lists and files to work with data
# Change Log: (Who, When, What)
#   RRoot,1/1/2030,Created Script
#   Ryan Seng, 5/19/2024, Assignment06
# ------------------------------------------------------------------------------------------ #
```

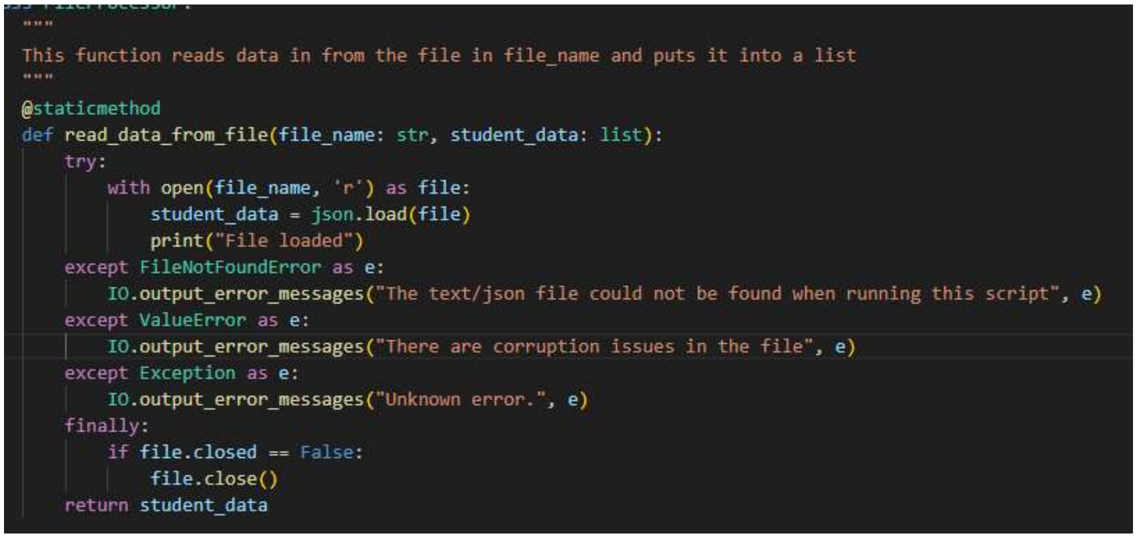
Since the assignment required that the “Enrollments” file I worked with be a .json file, I imported the functions from the json library using “import json”

For the constants and variables needed outside of each class or function, there is only 2 constants and 2 variables. The first constant is MENU which is a string with text resembling a menu for users to give inputs for. The second constant is FILE_NAME which is also a string set equal to “Enrollments.json”. For the variables, there is menu_choice which is set as a blank string and students which is a blank list. Below is a snip with the code as described:



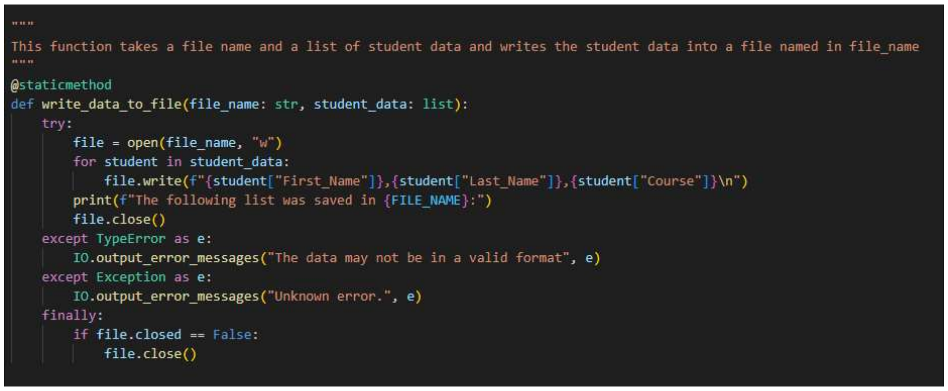
```
# Importing JSON library to get json functions
import json

# This is the MENU users will be selecting from
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
-----------------------------------------
"""

# This are the other constants
FILE_NAME: str = "Enrollments.json"

# These are the variables
menu_choice: str = ""
students: list = []
```

For the next part of the assignment, I create a bunch of classes which would contain functions that can be used throughout the script. The first class is the FileProcessor class which contains two functions with static method decorators. The first is the read_data_from_file function which takes a file_name and student_data as arguments. For this function, it is the same as the similar function in Assignment 5. It would open a file using the name from file_name and read the data into the student_data variable passed in the argument. Then it would return the student_data variable. This was slightly changed from Assignment 5's version which used .open() and .closed while the Assignment 6 uses with open(). There are some exceptions in case the file name could not be found, if there were any issues with the file itself or if there was some error I did not name directly. These exceptions would then cause another function in another class to be run. Lastly, to be safe, I included a clause in the code from the Assignment06-Starter.py to ensure the file was closed in case the file was somehow open. Below is the code as described.



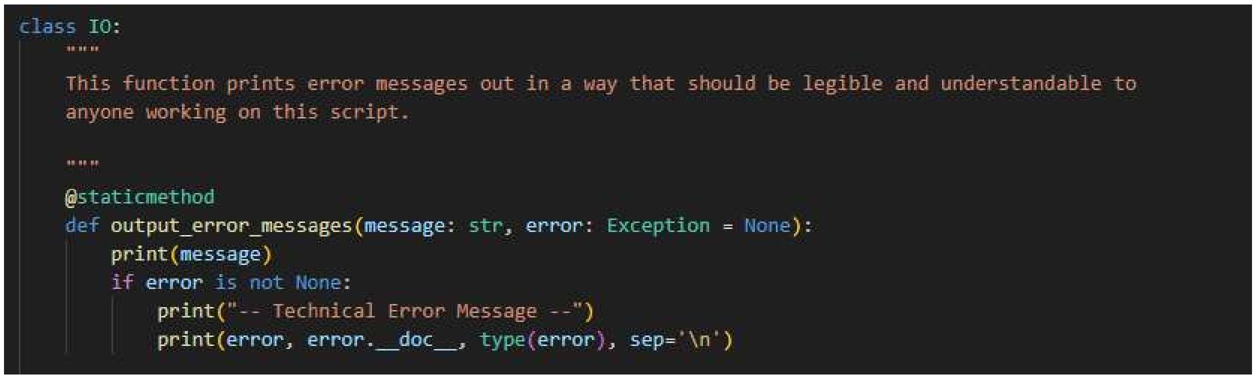
```
    """
    This function reads data in from the file in file_name and puts it into a list
    """
    @staticmethod
    def read_data_from_file(file_name: str, student_data: list):
        try:
            with open(file_name, 'r') as file:
                student_data = json.load(file)
                print("File loaded")
        except FileNotFoundError as e:
            IO.output_error_messages("The text/json file could not be found when running this script", e)
        except ValueError as e:
            IO.output_error_messages("There are corruption issues in the file", e)
        except Exception as e:
            IO.output_error_messages("Unknown error.", e)
        finally:
            if file.closed == False:
                file.close()
        return student_data
```

The other function in the FileProcessor class is write_data_to_file. This function would take a file_name and student_data passed in as arguments. Then it would try to open a file, write the student_data into that file, print the student data written into the file, then close the file. If there were any errors with the data format or an error I didn't anticipate, the code would run IO.output_error_messages(). I didn't know think about the case that a TypeError could occur until I looked at Lab 3 for Module 6 so I took the code from that script. Lastly like with the previous group of code, I included code from Assignment06-Starter.py that covered the case that if the file was still open, it would close. Below is the code as described.



```
"""
This function takes a file name and a list of student data and writes the student data into a file named in file_name
"""
@staticmethod
def write_data_to_file(file_name: str, student_data: list):
    try:
        file = open(file_name, "w")
        for student in student_data:
            file.write(f"{student["First_Name"]},{student["Last_Name"]},{student["Course"]}\n")
        print(f"The following list was saved in {FILE_NAME}:")
        file.close()
    except TypeError as e:
        IO.output_error_messages("The data may not be in a valid format", e)
    except Exception as e:
        IO.output_error_messages("Unknown error.", e)
    finally:
        if file.closed == False:
            file.close()
```

The next chunk of code for the assignment pertains to the IO class and its functions. All functions have the static method decorator. The first function is the output_error_messages function which takes in a message as a string, and an error. If no error is passed into the function, then the default is None. The function would print the message, then if the error is not None, it'd print details for the error. The "if" portion of the code was taken from Lab 3. Below is the code as described.

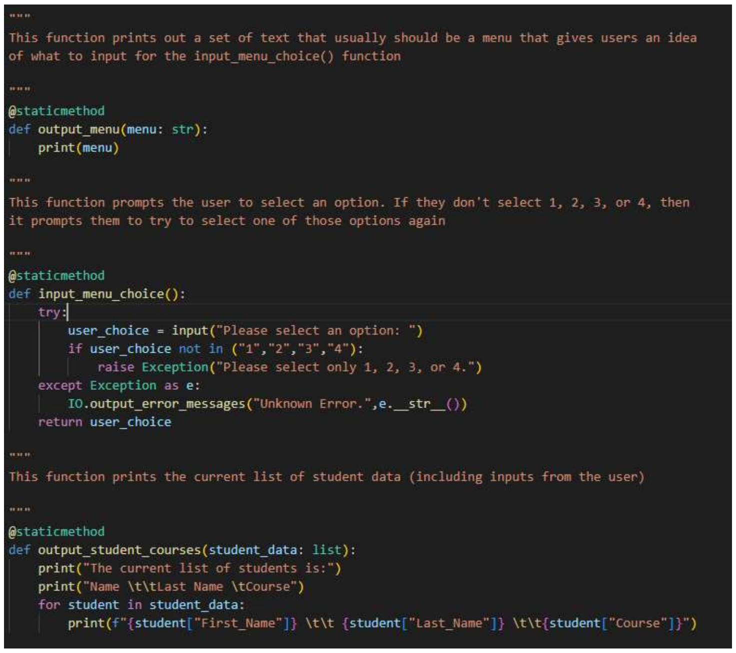


```python
class IO:
    """
    This function prints error messages out in a way that should be legible and understandable to
    anyone working on this script.

    """
    @staticmethod
    def output_error_messages(message: str, error: Exception = None):
        print(message)
        if error is not None:
            print("-- Technical Error Message --")
            print(error, error.__doc__, type(error), sep='\n')
```

For the second function, it is named output_menu which takes a menu variable as an argument and prints that menu.

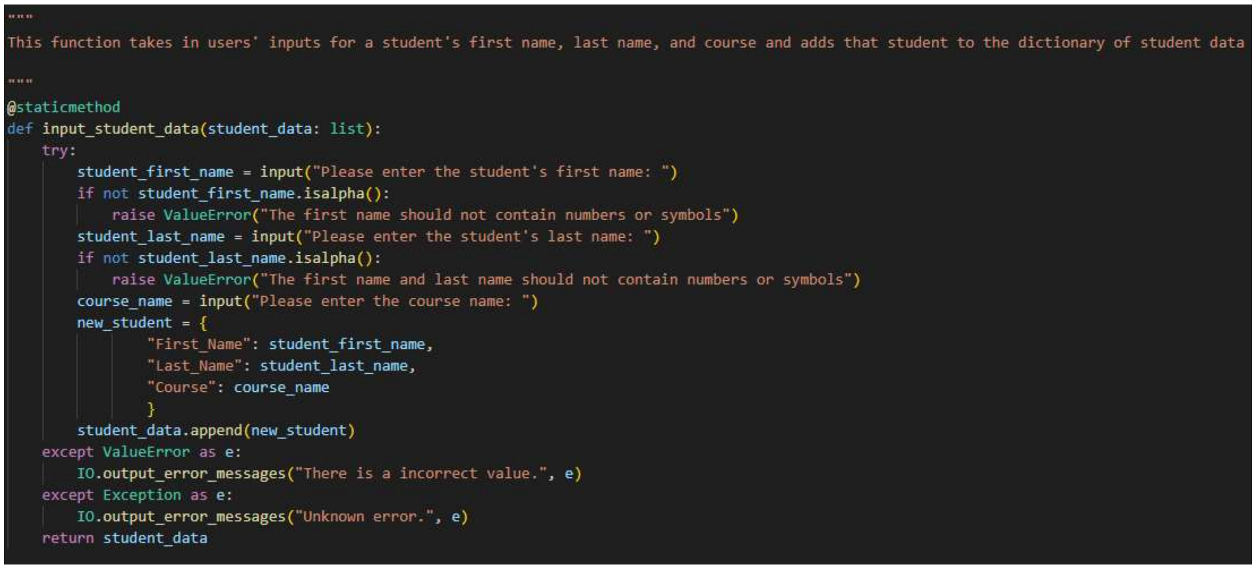
For the third function, it is called input_menu_choice. The function prompts the user to select an option by giving an input. If the input is no "1", "2", "3", or "4", then it would raise an error. Additionally, if there was some error that the user entered that I did not anticipate, then it would also raise an error and call the output_error_messages function in the IO class. Lastly, it would return the user's input.

The fourth function is the output_student_courses function which takes in a list of student_data as the argument. The function would then print the message "The current list of students is:" and the table of student_data passed in. Below is the code for both the second, third, and fourth functions as described.



```python
"""
This function prints out a set of text that usually should be a menu that gives users an idea
of what to input for the input_menu_choice() function

"""
@staticmethod
def output_menu(menu: str):
    print(menu)

"""
This function prompts the user to select an option. If they don't select 1, 2, 3, or 4, then
it prompts them to try to select one of those options again

"""
@staticmethod
def input_menu_choice():
    try:
        user_choice = input("Please select an option: ")
        if user_choice not in ("1","2","3","4"):
            raise Exception("Please select only 1, 2, 3, or 4.")
    except Exception as e:
        IO.output_error_messages("Unknown Error.",e.__str__())
    return user_choice

"""
This function prints the current list of student data (including inputs from the user)

"""
@staticmethod
def output_student_courses(student_data: list):
    print("The current list of students is:")
    print("Name \t\tLast Name \tCourse")
    for student in student_data:
        print(f"{student["First_Name"]} \t\t {student["Last_Name"]} \t\t{student["Course"]}")
```

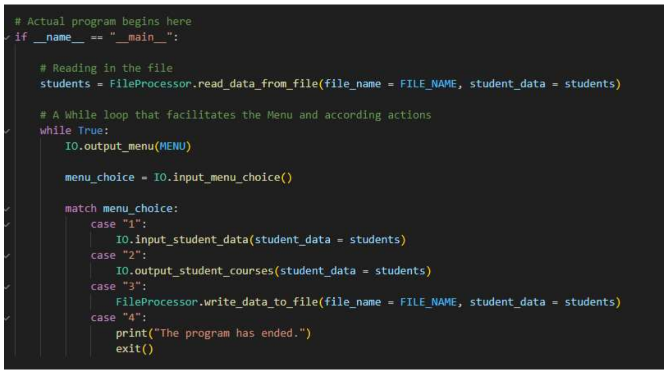
The final function in the IO class is input_student_data with a static method decorator and it takes a list of student_data as an argument. The code would prompt the user to try to enter a student's first name, last name, and course. Then it would combine the first name, last name, and course into one variable, new_student, and append it to the student_data list. Then the function would return the student_list variable. If there were symbols or numbers entered in the student's name, then it would raise an error. This error handling was taken from Lab 3 as I did not know the .isalpha() function existed. Below is the code as described.



```
"""
This function takes in users' inputs for a student's first name, last name, and course and adds that student to the dictionary of student data

"""
@staticmethod
def input_student_data(student_data: list):
    try:
        student_first_name = input("Please enter the student's first name: ")
        if not student_first_name.isalpha():
            raise ValueError("The first name should not contain numbers or symbols")
        student_last_name = input("Please enter the student's last name: ")
        if not student_last_name.isalpha():
            raise ValueError("The first name and last name should not contain numbers or symbols")
        course_name = input("Please enter the course name: ")
        new_student = {
                "First_Name": student_first_name,
                "Last_Name": student_last_name,
                "Course": course_name
                }
        student_data.append(new_student)
    except ValueError as e:
        IO.output_error_messages("There is a incorrect value.", e)
    except Exception as e:
        IO.output_error_messages("Unknown error.", e)
    return student_data
```

The last remainder of the code is the main program itself. The script is only set to run if the script is the main program due to the "if __name__ == "__main__"". If the script is the main program, then it will load the student data using the FileProcessor class' read_data_from_file function with FILE_NAME and students as the arguments. Then it stores the returned data in the students variable. Then the script prints the menu using the IO.output_menu with the MENU constant as the argument. Then it runs IO.input_menu_choice() and stores the result in the menu_choice variable.

If the menu_choice variable is 1, then the script runs the IO.input_student_data function with the students variable as the argument. If menu_choice is 2, then it runs the IO.output_student_courses function with students as the argument. If the menu_choice is 3, then it runs the FileProcessor.write_data_to_file function with FILE_NAME and students as the arguments. Lastly, if menu_choice is 4, then it prints "the program has ended" and exits the program. Below is the code as described in the last 2 paragraphs.



```
# Actual program begins here
if __name__ == "__main__":

    # Reading in the file
    students = FileProcessor.read_data_from_file(file_name = FILE_NAME, student_data = students)

    # A While loop that facilitates the Menu and according actions
    while True:
        IO.output_menu(MENU)

        menu_choice = IO.input_menu_choice()

        match menu_choice:
            case "1":
                IO.input_student_data(student_data = students)
            case "2":
                IO.output_student_courses(student_data = students)
            case "3":
                FileProcessor.write_data_to_file(file_name = FILE_NAME, student_data = students)
            case "4":
                print("The program has ended.")
                exit()
```

By doing all of this, I was able to complete Assignment 06. I then named the script “assignment06.py”.

Summary

Through lessons learned in class and from Lab 3, I was able load in data from a csv file into a list to create a dictionary of lists. Then I prompted the user to follow a menu with a variety of options including registering data, displaying the current data, saving the data, and exiting the program. All of this included structured error handling detailing what went wrong if there was an input from Enrollment.csv or from the user that cause issues. Potential repetitiveness in the code was reduced through the usage of classes and functions that was called throughout the script.

Link to GitHub

Link to GitHub: https://github.com/RyanS39/IntroToProg-Python-Mod6