Nasmil Valera Cuevas
February 19th, 2024
Foundations of Programming, Python
Assignment 06
https://github.com/nasmilvc/IntroToProg-Python-Mod06

Functions

Introduction

In this assignment, I will go over how I created a Python program that demonstrates the use of three common techniques for improving your scripts: functions, classes, and the separation of concerns programming pattern. I also used things we have learned on past modules, such as data processing using lists and dictionaries, usage of JavaScript Object Notation (JSON) files and Structured Error Handling. This work was completed with the source material provided by Professor Randal Root, helpful demonstrations by Anubhaw Arya and a handful of external videos linked throughout the module.

Drafting the Code

I began by opening and reviewing the starter file Assignment06-Starter.py, and since I knew I would be working with JSON data, I imported the built-in JSON module onto the script. I also made sure the 'Enrollments.json' file had data.

Defining the Data Layer

When defining the data items, I defined 'FILE_NAME' and 'MENU' as constants and 'students' and 'menu_choice' as global variables. Defining these as global variables allows them to be accessed and modified from anywhere within the script.

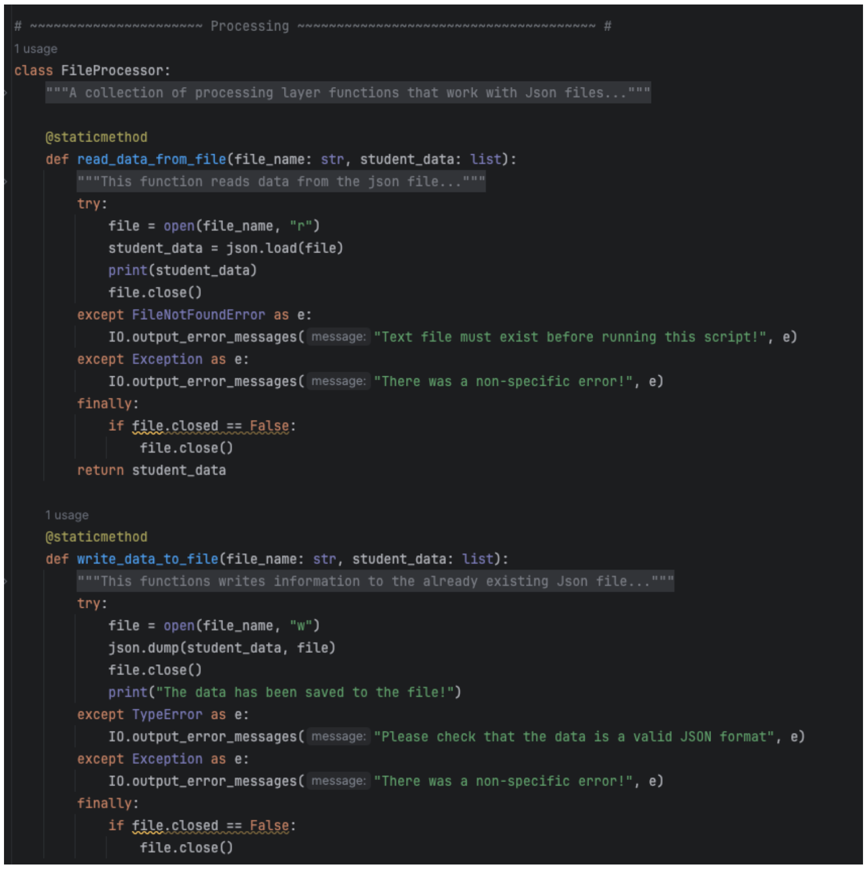
Defining the Processing Layer

I created a class named 'FileProcessor' with a docstring with additional notes describing it as a collection of functions that work with Json files. I defined the 'read_data_from_file' function with the '@staticmethod' decorator and established local variables 'file_name' and 'student_data' as parameters. When we use the '@staticmethod' decorator Python allows us to use the functions in a class without first creating an object. In this assignment, I added this decorator to every function. I added a docstring to specify that this function reads data from Json files. I used previous code that reads information, closes the file, and uses a try-except construct for when the file I am asking the program to read cannot be found or when a non-specific error occurs.
Then, I defined the 'write_data_to_file' function using the same local variables as parameters like the previous function, 'file_name' and 'student_data'. I added a docstring to specify that this function writes information to already existing Json files. I used previous code that uses the

open() function to open a file in write mode, uses the json.dump() function to add information to Json files and closed the file to save the changes made. I also added a print statement to notify the user that the data had been saved to the file. This code also had the try-except construct for if the user attempts to add data that is not in a valid JSON format and or when a non-specific error occurs. This layer is not complete until we get to the presentation layer. Refer to Figure 1 for the finished product.

Defining the Presentation Layer

I created a class named 'IO' with a docstring with additional notes describing it as a collection of functions that manage user input and output. I defined the 'output_error_messages' function with the parameter 'message: st'r and the argument 'Exception = None.' I added a docstring to specify that the function displays a custom error message to the user. If an error occurs, the function will print the error, the error documentation, and the error type. Once I checked the function was running well, I went back to the 'FileProcessing' class to add this function as IO. output_error_messages() to the try-except constructs of functions 'read_data_from_file' and 'write_data_to_file' (Figure 1).



```
# ~~~~~~~~~~~~~~~~~~~~~~~ Processing ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ #
1 usage
class FileProcessor:
    """A collection of processing layer functions that work with Json files..."""

    @staticmethod
    def read_data_from_file(file_name: str, student_data: list):
        """This function reads data from the json file..."""
        try:
            file = open(file_name, "r")
            student_data = json.load(file)
            print(student_data)
            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

    1 usage
    @staticmethod
    def write_data_to_file(file_name: str, student_data: list):
        """This functions writes information to the already existing Json file..."""
        try:
            file = open(file_name, "w")
            json.dump(student_data, file)
            file.close()
            print("The data has been saved to the file!")
        except TypeError as e:
            IO.output_error_messages( message: "Please check that the data 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 1. Screenshot of Processing layer, including class FileProcessor, and functions to read data from the file and write data to the file.

I defined function ‘output_menu’ with parameter ‘menu:str’ and a docstring describing it displays the menu of choices to the user. I also added two print statements, one above and one below the menu to create more space and make the code look nice when it’s ran.

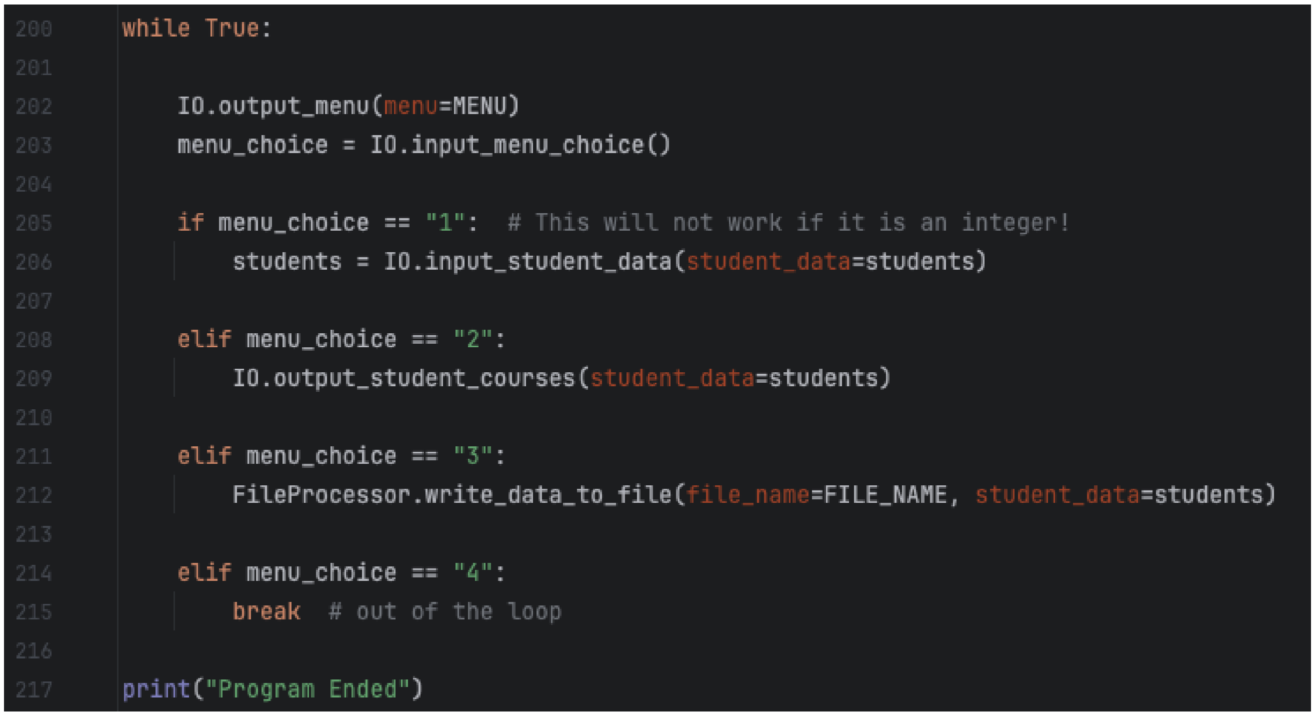
I defined the ‘input_menu_choice’ function and added a docstring describing what the function does. I used the input() function to prompt the user to enter their choice and a try-except construct with the output_error_message() function to show the user the error. I also used a return statement with ‘choice’ to return the value to the calling function.

I defined the ‘input_student_data’ function, adding ‘student_data’ as a parameter and a docstring. I used code we’ve used in the past to prompt the user to input the student’s first name, last name, and course they are signing up for. I used a structured error handling when the user inputs a first name and a last name by adding the .isalpha() function, which ensures that all the entered values are alphabetical. I also used a try-except construct with exception portions that include the IO.output_error_messages() function with the respective statements I want printed for the user. I created a dictionary with the user input called ‘student’ and used the .append() function to add the new ‘student’ information to the ‘student_data’ list. Then, I formatted a string to display the student information and the course they just enrolled in.
I defined the ‘output_student_courses’ function with parameter ‘student_data’ and a corresponding docstring. I included a string to display that the following would be the information that they have thus far. Then, I used a for loop to print all the data that is currently stored in the list.

Beginning of the Main Body of Script

When the program starts, the contents of the "Enrollments.json" file have to be automatically displayed in a two-dimensional list table so I called the read_from_file() function as students = ‘FileProcessor.read_data_from_file(file_name=FILE_NAME, student_data=students)’. I added the class name before the function so that Python knows where to find the function. This also adds value to the ‘students’ variable that I will continue to use throughout the program.

The starter code provided the structure needed to plug in the functions where the code that defined them existed. Once I located the while loop, I added the output_menu() function from the IO class to display the menu to the user after every selection and defined the ‘menu_choice’ variable with the input_menu_choice() function from the IO class. Then, I continued to replace the existing code for the if and elif statements for menu_choice 1 through 3 with their respective functions (Figure 2). Menu choice 4 breaks the user out of the loop and after the loop is closed, the program notifies the user that the program has ended.

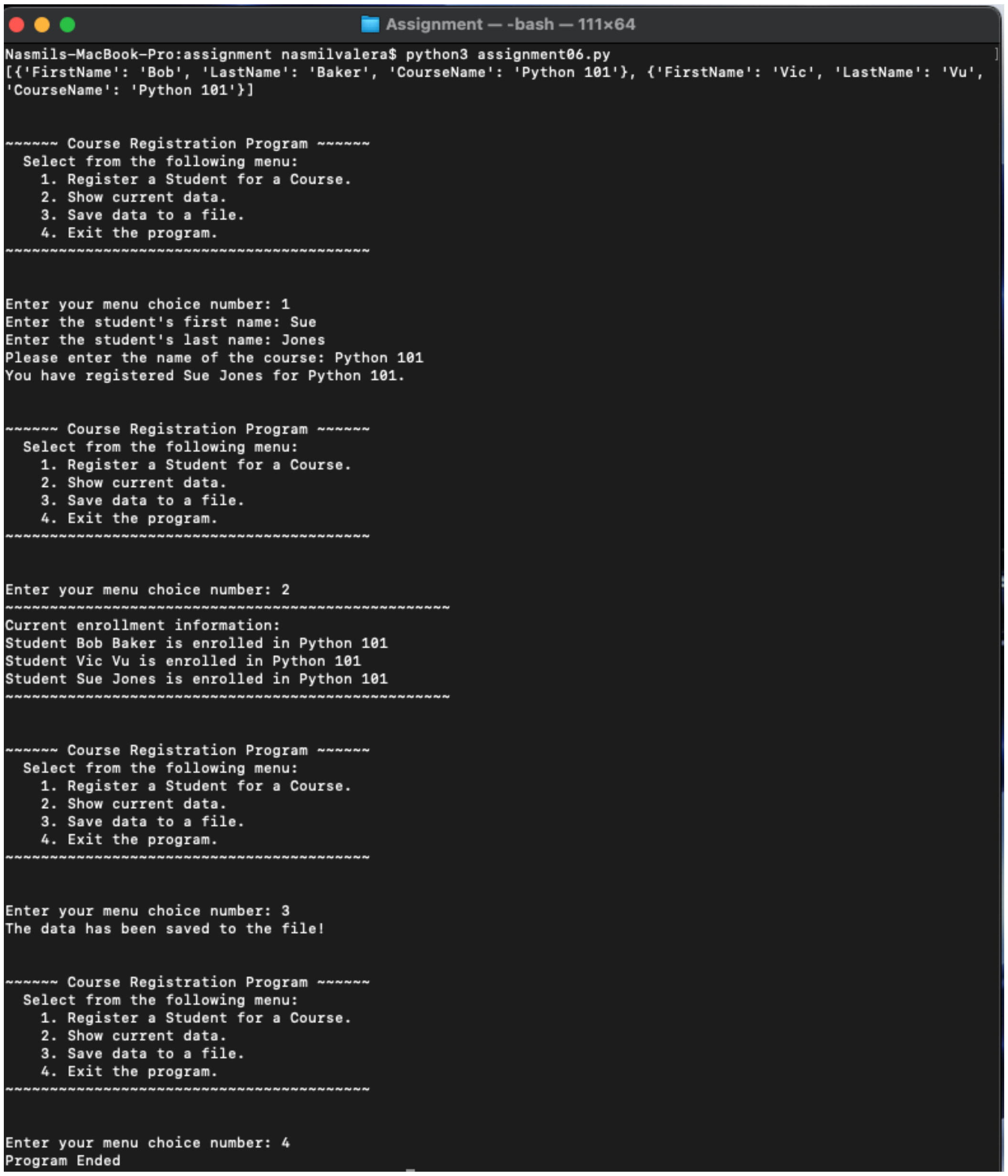


```
while True:

    IO.output_menu(menu=MENU)
    menu_choice = IO.input_menu_choice()

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

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

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

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

print("Program Ended")
```

Figure 2: Screenshot of while loop with recalled functions from classes FileProcessing and IO.

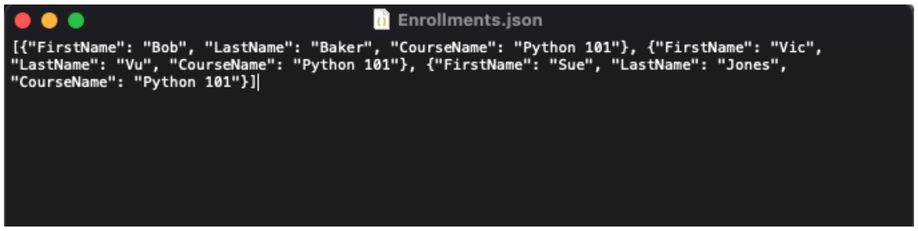
Running the Script in OS Command

I opened the Terminal console on my computer, navigated across directories and used the python3 command with file Assignment06.py to run my script. I followed the user prompts, and the script ran as expected (Figure 4), which was confirmed by the contents of the json file manipulated in this program (Figure 5).



```
Nasmils-MacBook-Pro:assignment nasmilvalera$ python3 assignment06.py
[{'FirstName': 'Bob', 'LastName': 'Baker', 'CourseName': 'Python 101'}, {'FirstName': 'Vic', 'LastName': 'Vu',
'CourseName': 'Python 101'}]


~~~~~~ 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.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


Enter your menu choice number: 1
Enter the student's first name: Sue
Enter the student's last name: Jones
Please enter the name of the course: Python 101
You have registered Sue Jones for Python 101.


~~~~~~ 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.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


Enter your menu choice number: 2
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Current enrollment information:
Student Bob Baker is enrolled in Python 101
Student Vic Vu is enrolled in Python 101
Student Sue Jones is enrolled in Python 101
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


~~~~~~ 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.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


Enter your menu choice number: 3
The data has been saved to the file!


~~~~~~ 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.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


Enter your menu choice number: 4
Program Ended
```

Figure 4: Screenshot of program Assignment06.py running currently in the MacOS Terminal



```
[{"FirstName": "Bob", "LastName": "Baker", "CourseName": "Python 101"}, {"FirstName": "Vic",
"LastName": "Vu", "CourseName": "Python 101"}, {"FirstName": "Sue", "LastName": "Jones",
"CourseName": "Python 101"}]
```

Figure 5. Screenshot of file "Enrollments.json" text file with data from the program.

Summary

I was able to successfully create a Python program that demonstrated the use of three common techniques that are commonly used to improve scripts, which are functions, classes, and the separation of concerns programming pattern. I also learned about return values and the value of encapsulating logic within functions. I used an array of functions streamline the code and push errors to guide code users to follow prompts according to the format.