Andrea Martinez

November 21, 2024

Python 100

Assignment 06

GitHub URL: https://github.com/martia72/IntroToProg-Python-Mod06.git

Assignment 06 – Programming with Python Functions and Classes

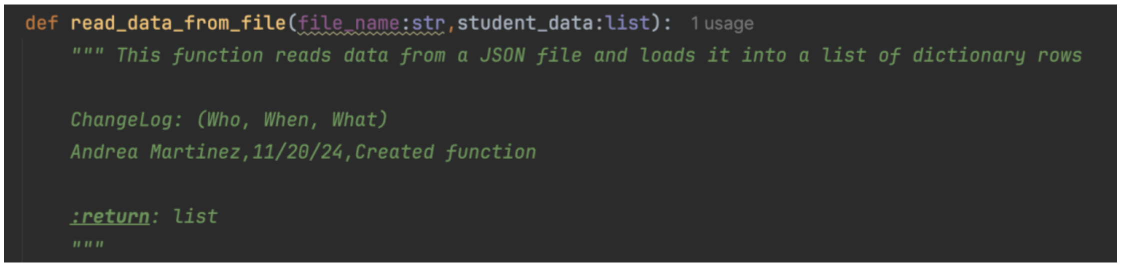
Introduction

In module 06 of this course, one of the main objectives was to use functions with structured error handling. In this document, I'll briefly describe my initial thoughts about using functions as well as the differences from using classes. In addition, I'll describe the steps taken to manage errors within a script and how the try, except, and finally can be helpful while using functions. I'll also show how scripts can be organized based on concerns and divided into three types of concerns. Finally, I will mention how implementing functions can be a powerful tool for reusability for large programs.

The FileProcessor Class

In this module, I learned that classes are a way to group functions, and it helps maintain a modular structure in the code. This in turn, helps the script keep a good structure, which makes it easier to read. This is usueful in areas where several developers contribute to the same code, so dividing code into classes and functions can help contributors debug codes and add additional utilities to the code.

In the FileProcessor Class, I added two functions, the read_data_from_file and the write_data_to_file to be able to open, read, and write data to a file. In both of the class and the two the functions I wrote, I included a doctring, which is a literal that occurs as the first statement in a module, function, class, or method definition. It's used to document the code and explain its purpose, parameters, and return values according to the Python documentation (https://peps.python.org/pep0257/#:~:text=A%20docstring%20is%20a%20string,module%20should%20also%20have%20docstrings, 2024). Figure 1 contains a screenshot of the doctring used in the read_data_from_file function.



```
def read_data_from_file(file_name:str,student_data:list):  1 usage
    """ This function reads data from a JSON file and loads it into a list of dictionary rows

    ChangeLog: (Who, When, What)
    Andrea Martinez,11/20/24,Created function

    :return: list
    """
```

Figure 1: Docstring string literal under a python function

Writing the two functions under the class FileProcessor was very straightforward because I had already written the execution code for reading and writing JSON files during module 05. So, all that I had to do for reading a file was to copy the code that pertained to the read JSON data and paste it under the read_data_from_file function, which contained the important load() statement that is used to process JSON files. Similarly, I also copied the code that included the dump() function for writing into a JSON file and saved it under the newly created write_data_to_file function.

While writing this class, I was also able to maintain the same try-except statements that I made in Module 05 that raise flags or errors where there are problems reading or writing data to a file.

The IO Class

The IO class, or the Input-Output class is a collection of functions that manage the user's input and output. For example, in this class, I defined the functions that display the programming menu, collect the user's choices, present the data in the file, and handle any input errors.

The first function defined in the IO class is the output_error_messages(), which allows the developer to include custom error messages to the user. For instance, this function was called under the read_data_from_file and write_data_to_file functions to raise any errors when there were any issues reading or wiritng the files. So, I included a custom message in the form of a string as the argument of the output_error_messages() function. Writing this script helped me understand the difference between parameters and arguments. A parameter is a variable listed inside the parentheses in the function definition, which acts as a placeholder for the values that will be passed to the function when it is called. On the other hand, the argument is the actual value that is passed to a function when it is called as stated in W3Schools.com (https://www.w3schools.com/python/gloss_python_function_arguments.asp#:~:text=Parameters%20or%20Arguments?,function%20when%20it%20is%20called, 2024).

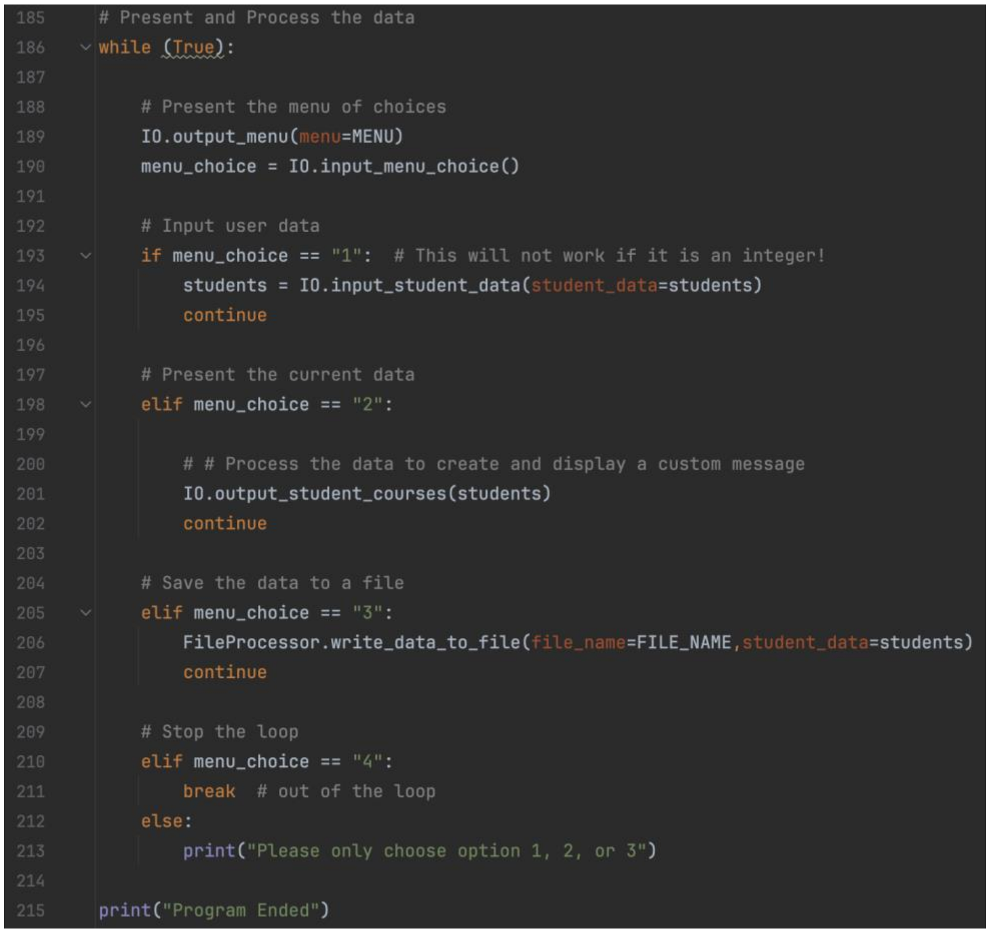
Other functions in the IO class included the output_menu() function which takes the menu string as the parameter and displays it to the user. The input_menu_choice() choice function uses Python's input() function to collect the user's choice, and it also includes two error handling statements in case there are issues collecting the right data. The output_student_courses() displays the students registered and their courses to the user by using print() statements and the existing data saved in the dictionaries. Lastly, the input_student_data() function is used when the user chooses to register a student in the programming menu. This function also contains error handling that alerts the user when the entered data was invalid. The ValueError exception is used to show flag the program.

Calling Functions in the Presentation Layer

Now that all the functions and classes were defined, we were ready to execute the code. Up until this point, the classes or functions do not execute unless they are called in the script. So, the first thing I did was call the read_data_from_file() function, which reads the file data into a list of lists (table), and extracts the data from the file. Then, I called the output_menu() function to display the menu of choices, and the input_menu_choice() to store the choice in a variable. If choice #1 was selected from the menu, I used the input_student_data() function to register a new student to a course. If menu choice #2 was selected, I called the output_student_courses() function to simply output the current list of students registered to a course. If choice #3 was selected, I called the write_data_to_file() function to save the current list of students to a JSON file. If selection #4 was made, no other operation would be made, and I would just end the program with the break command. Now, even though, this was not a part of the script requirement, I added an “else” statement for any other answer that was not either one of the choices in the menu (1-4). This else statement printed an error message, indicating the selection was not valid and a valid answer must be entered again.

Testing the Program

As a developer, I had to test my code to make sure that it was doing what it was supposed to by meeting all the requirements set by the instructor. Writing this code was very interesting because I learned that defining functions and classes at the start of the code do not get executed unless I call them later in the script. This is a very powerful way to script because I can reuse functions in other scripts as long as I import them or call them in the code. Avoiding repeated lines of code can help save a lot of time and memory space in a computer when running long scripts. Figure 2 shows how the execution portion of the code can be shorten to roughly 30 lines when it used to be approximately 80 in the module 05 assignment. It also looks far cleaner and more organized to have a concise execution code that calls functions that are created earlier in the script.



```python
# Present and Process the data
while (True):

    # Present the menu of choices
    IO.output_menu(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(student_data=students)
        continue

    # Present the current data
    elif menu_choice == "2":

        # # Process the data to create and display a custom message
        IO.output_student_courses(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  # out of the loop
    else:
        print("Please only choose option 1, 2, or 3")

print("Program Ended")
```

Figure 2: Execution code

Separation of Concerns

After writing this code in different stages (classes, functions, calling functions), I can now attest to the versatility of Separation of Concerns (SoC) in scripting. SoC involves dividing code into distinct sections, each addressing a specific concern or task. This approach improves readability, reduces complexity, enhances maintainability, facilitates reuse. The first part of the code contained the Data Layer, which focused on creating all the right constants and global variables to be used by the script. Then came the Processing Layer, which was managed by the FileProcessor class. Finally, the IO class represented the Presentation Layer of the script, which contained all the user-input handling and data display.

Summary

In conclusion, the planning, writing, and execution of this code taught me how to properly use classes, functions, and separation of concerns. I strengthened my skills reading and writing data to a JSON file. I learned to define functions and classes to reference them in the presentation layer of a code. While writing functions, I also learned the differences between arguments and parameters, which are oftentimes used interchangeably but have different meanings. Additionally, I learned how to use global and local variables in a code. This code only contained two global variables, which were ‘students’ and ‘menu_choice’, which could be technically used in other scripts as long as I import them. Lastly, I continued to apply good practices in coding such as using GitHub for uploading version-controlled python files.