Tom Lowder

11/19/2024

IT FDN 110 A

Assignment06

GitHub Repository: https://github.com/tlowder24/IntroToProg-Python-Mod06

Functions

Introduction

Module06 explored the building blocks of programming, functions and classes. The module delved into the distinction between global vs local variables, parameters, return values, and separation of concerns pattern.

Creating the program

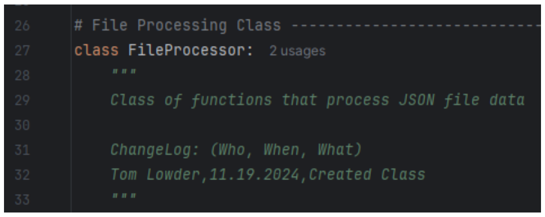
The objective of Assignment06 was to modularize the Assignment06-Starter program to include classes and functions. The remainder of the paper describes modfications made to the program to achieve the task.

First off, the program header was modified for the Assginment06-Starter program.

Then the buidling blocks of the new program, the class names, functions names, and parameters used in each function, were added. The following structure was added to the program after the defined variables.

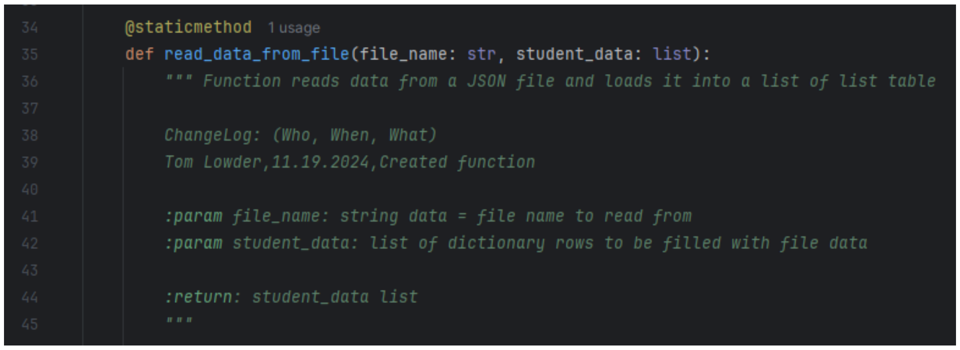
```
Class FileProcessor:
        read_data_from_file(file_name: str, student_data: list):
        write_data_to_file(file_name: str, student_data: list):
Class IO:
        output_error_messages(message: str, error: Exception = None):
        output_menu(menu: str):
        input_menu_choice():
        output_student_courses(student_data: list):
        input_student_data(student_data: list):
```

Since it is common practice to include a docstring (header) at the begginning of each class or function, these were included in the program. Information in the docstring inlcuded a description of the class or function, changelog, parameter description (for functions) and return value (for functions). An example of a class and function docstring used in the program are shown in Figure 1 and figure 2.



```
# File Processing Class ---------------------------
class FileProcessor:  2 usages
    """
    Class of functions that process JSON file data

    ChangeLog: (Who, When, What)
    Tom Lowder,11.19.2024,Created Class
    """
```

Figure 1. Class docstring

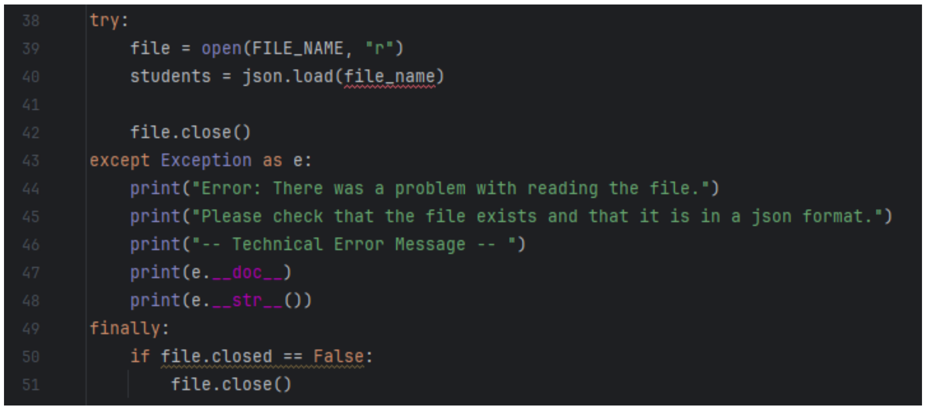


```python
    @staticmethod  1 usage
    def read_data_from_file(file_name: str, student_data: list):
        """ Function reads data from a JSON file and loads it into a list of list table

        ChangeLog: (Who, When, What)
        Tom Lowder,11.19.2024,Created function

        :param file_name: string data = file name to read from
        :param student_data: list of dictionary rows to be filled with file data

        :return: student_data list
        """
```

Figure 2. Function docstring and @staticmethod

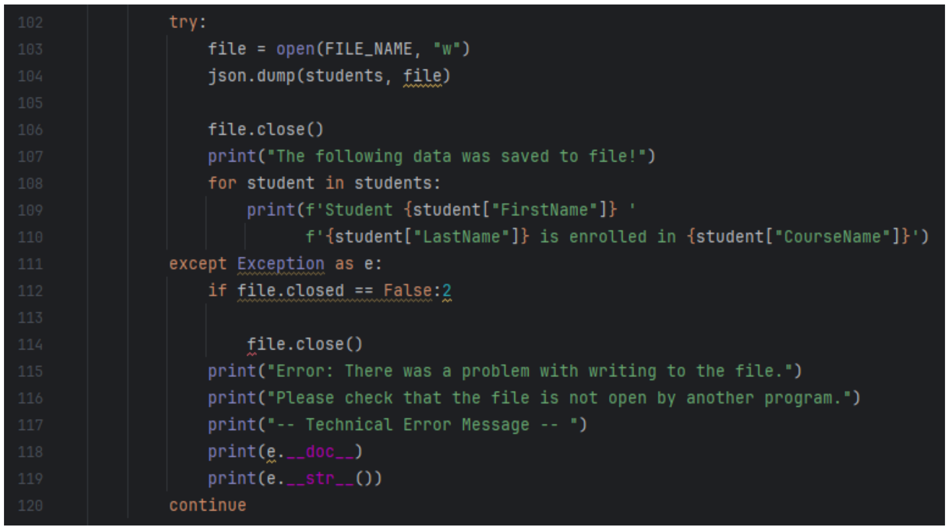
Additionally, all function code in this program never change or are "static", so the @staticmethod decorator is used and placed above each function. This allows the use of class functions directly without having to create an operator. This is shown in Figure 2.

After that, the following starter code was cut and pasted into the respective functions.



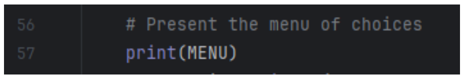
```python
try:
    file = open(FILE_NAME, "r")
    students = json.load(file_name)

    file.close()
except Exception as e:
    print("Error: There was a problem with reading the file.")
    print("Please check that the file exists and that it is in a json format.")
    print("-- Technical Error Message -- ")
    print(e.__doc__)
    print(e.__str__())
finally:
    if file.closed == False:
        file.close()
```

Figure 3. For read_data_from_file(file_name: str, student_data: list):



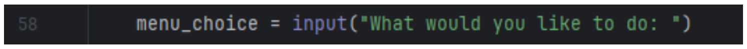
```python
        try:
            file = open(FILE_NAME, "w")
            json.dump(students, file)

            file.close()
            print("The following data was saved to file!")
            for student in students:
                print(f'Student {student["FirstName"]} '
                      f'{student["LastName"]} is enrolled in {student["CourseName"]}')
        except Exception as e:
            if file.closed == False:2

                file.close()
            print("Error: There was a problem with writing to the file.")
            print("Please check that the file is not open by another program.")
            print("-- Technical Error Message -- ")
            print(e.__doc__)
            print(e.__str__())
        continue
```

Figure 4. For write_data_to_file(file_name: str, student_data: list):



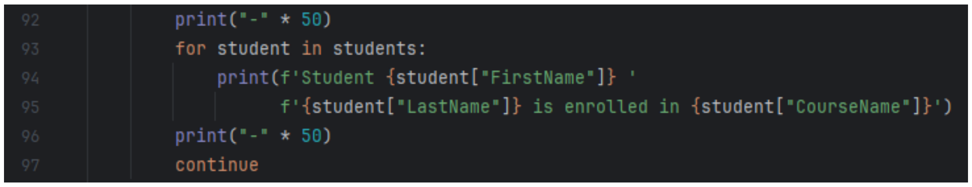
```python
    # Present the menu of choices
    print(MENU)
```

Figure 5. For output_menu(menu: str):



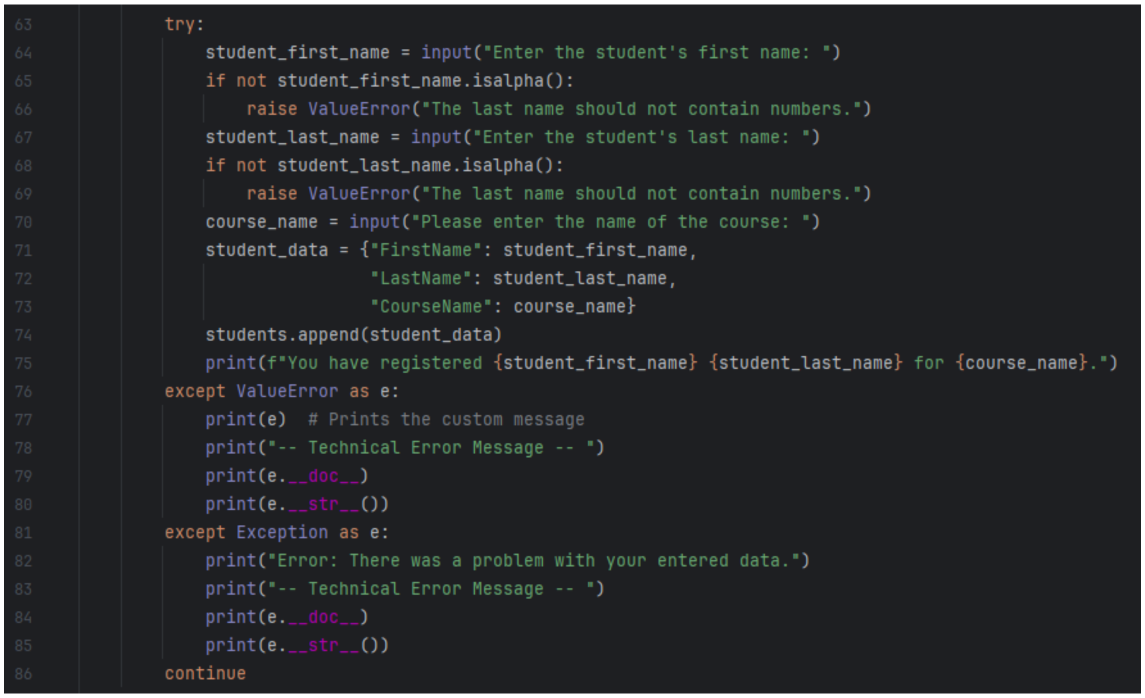
```python
    menu_choice = input("What would you like to do: ")
```

Figure 6. For input_menu_choice():



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

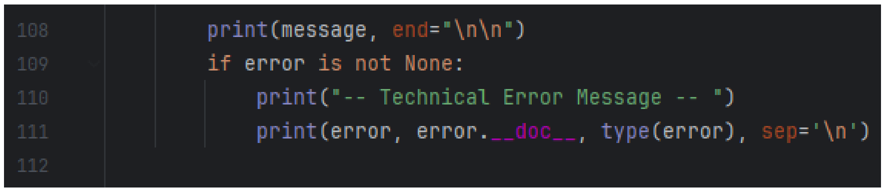
Figure 7. For output_student_courses(student_data: list):



```python
        try:
            student_first_name = input("Enter the student's first name: ")
            if not student_first_name.isalpha():
                raise ValueError("The last 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: ")
            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:
            print(e)  # Prints the custom message
            print("-- Technical Error Message -- ")
            print(e.__doc__)
            print(e.__str__())
        except Exception as e:
            print("Error: There was a problem with your entered data.")
            print("-- Technical Error Message -- ")
            print(e.__doc__)
            print(e.__str__())
        continue
```

Figure 8. For input_student_data(student_data: list):

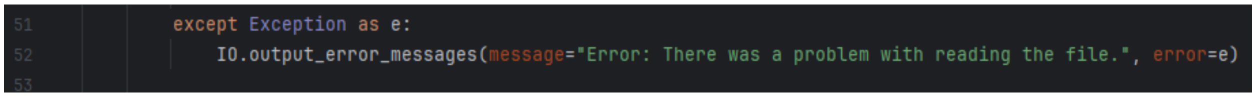
Output_error_messages(message: str, error: Exception = None): function was then written, Figure 9, to be used to present exception error messages to user.



```
        print(message, end="\n\n")
        if error is not None:
            print("-- Technical Error Message -- ")
            print(error, error.__doc__, type(error), sep='\n')

```

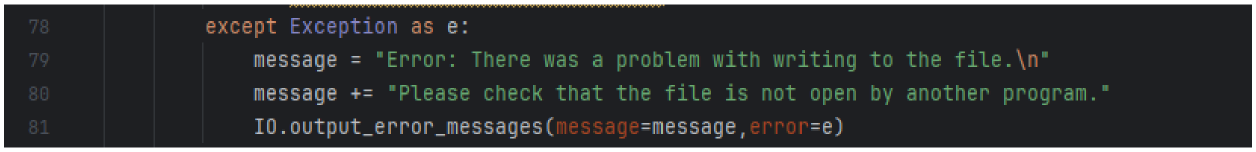
Figure 9. Output_error_messages function code

Next, the error handling code in each function was changed to use the output_error_messages function.



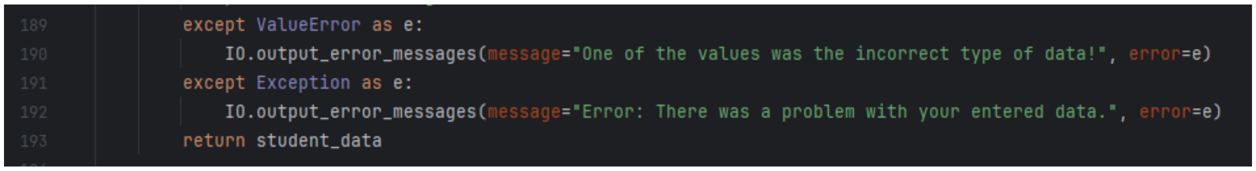
```
        except Exception as e:
            IO.output_error_messages(message="Error: There was a problem with reading the file.", error=e)
```

Figure 10. For read_data_from_file(file_name: str, student_data: list):



```
        except Exception as e:
            message = "Error: There was a problem with writing to the file.\n"
            message += "Please check that the file is not open by another program."
            IO.output_error_messages(message=message,error=e)
```

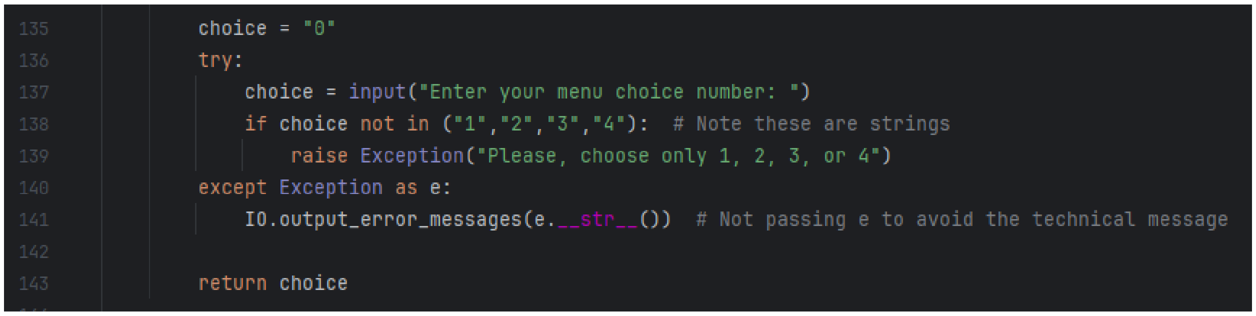
Figure 11. For write_data_to_file(file_name: str, student_data: list):



```
        except ValueError as e:
            IO.output_error_messages(message="One of the values was the incorrect type of data!", error=e)
        except Exception as e:
            IO.output_error_messages(message="Error: There was a problem with your entered data.", error=e)
        return student_data
```

Figure 12. For input_student_data(student_data: list):

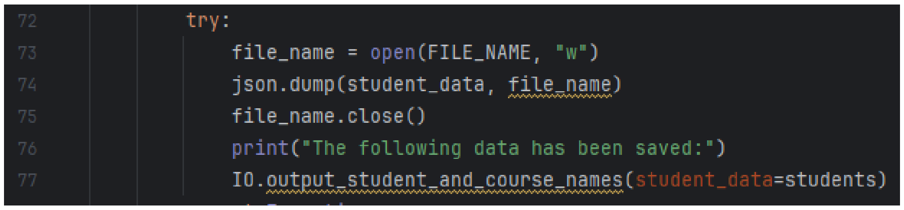
For input_menu_choice(): function, the else state from the starter file was incorporated as a raise exception in the function. Additionally, e.__string__() was used in the ouput_error_message function to not pass e and avoid a technical error message.



```
        choice = "0"
        try:
            choice = input("Enter your menu choice number: ")
            if choice not in ("1","2","3","4"):  # Note these are strings
                raise Exception("Please, choose only 1, 2, 3, or 4")
        except Exception as e:
            IO.output_error_messages(e.__str__())  # Not passing e to avoid the technical message

        return choice
```

Figure 13. Input_menu_choice(): function

After that, the for loop in the For write_data_to_file(file_name: str, student_data: list): function was replaced with the output_student_courses function as shown in Figure 14.

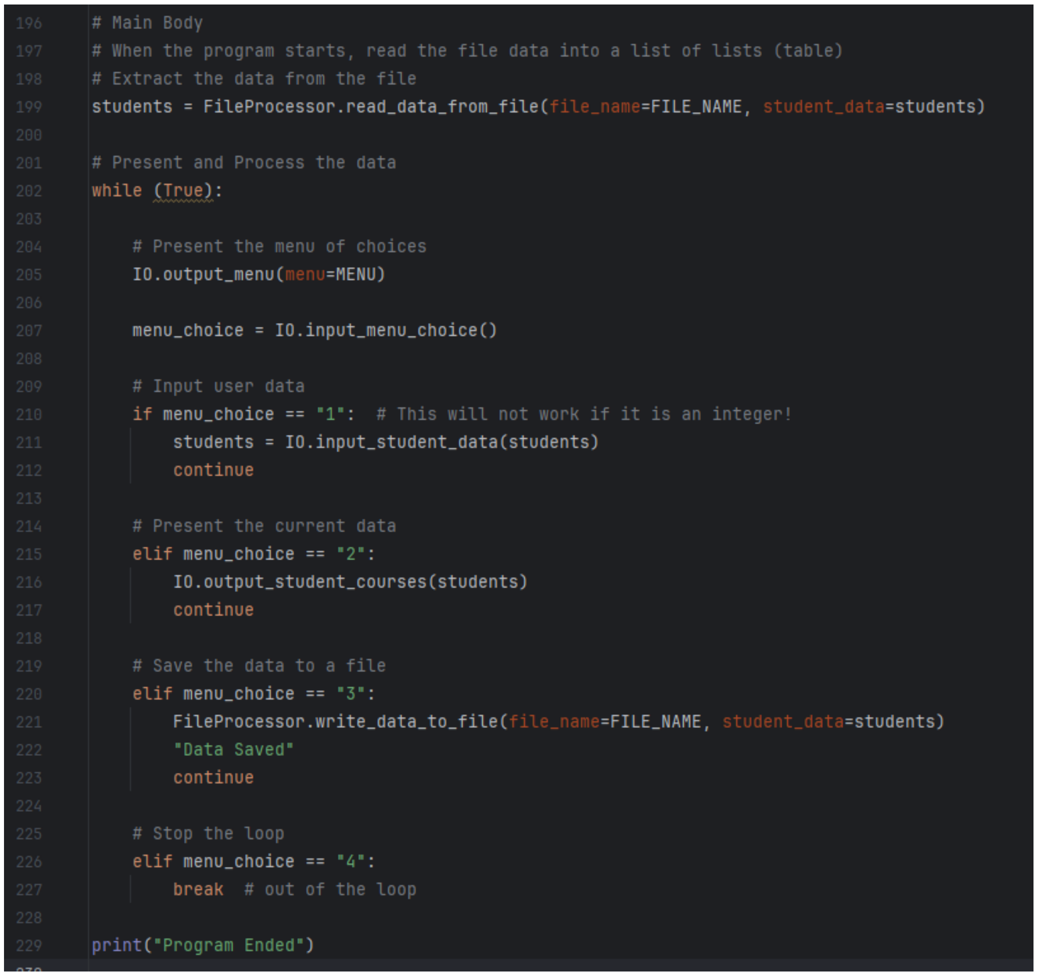


```
        try:
            file_name = open(FILE_NAME, "w")
            json.dump(student_data, file_name)
            file_name.close()
            print("The following data has been saved:")
            IO.output_student_and_course_names(student_data=students)
```

Figure 14. Output_student_courses function used in write_data_to_file function.

The next step was to clean up the formatting of the function data and variables. This included changing the old variables pasted in the function to the new parameters, removing variables not used, and adding extra space to make the program output neatly.

Lastly the main body of the code was simplified by including the functions.



```
# Main Body
# When the program starts, read the file data into a list of lists (table)
# Extract the data from the file
students = FileProcessor.read_data_from_file(file_name=FILE_NAME, student_data=students)

# 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(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=FILE_NAME, student_data=students)
        "Data Saved"
        continue

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

print("Program Ended")
```

Figure 15. Main body of program with functions.

Testing the program

The program was tested in PyCharm and the command prompt. Tests included:

- The program takes the user's input for a student's first, last name, and course name.
- The program displays the user's input for a student's first, last name, and course name.
- The program saves the user's input for a student's first, last name, and course name to a coma-separated string file.
- The program allows users to enter multiple registrations (first name, last name, course name).
- The program allows users to display multiple registrations (first name, last name, course name).
- The program allows users to save multiple registrations to a file (first name, last name, course name).

The program successfully ran and saved to Enrollments.json in both PyCharm and the command prompt.

I saved the program and this assignment to a new GitHub repository for module06 so others can review my work. The link to the repository is in the header of this document.

Summary

With the resources provided in module 06 I was able to create the program. The program demonstrates my new understanding of how to use classes and functions to modularize code.