Caitlyn McFarland

March 5, 2025

IT FDN 110 A

Assignment 06

GitHub Link: https://github.com/CaitMcF/IntroToProg-Python-Mod06

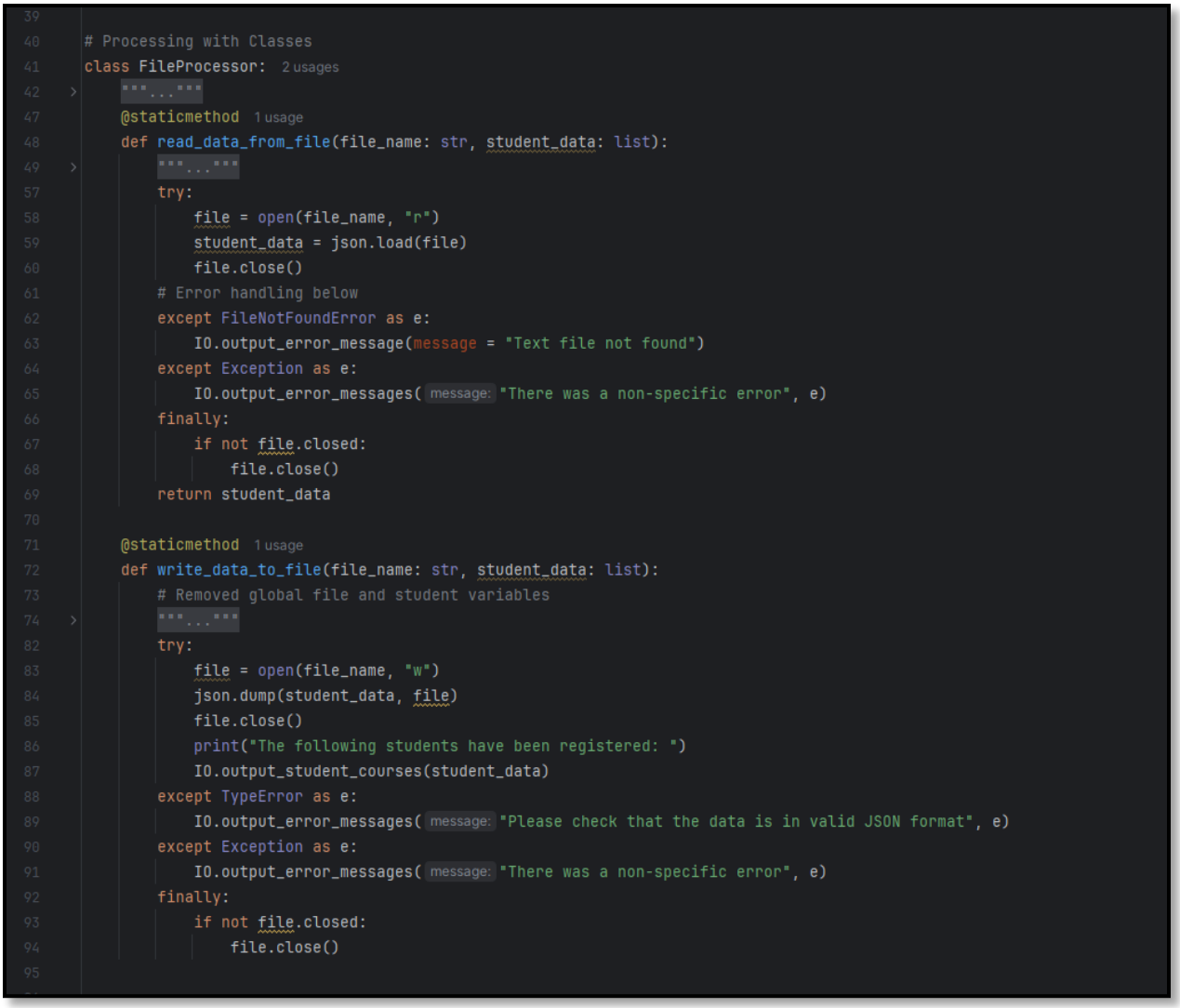
Classes

Intro

This week's assignment focused on using classes and functions. These tools are useful in structuring a program with logic blocks and organizing data for ease of use. This allows for a more organized and efficient code.

Creating Classes and Functions

In this code I used two different classes. The classes help organize methods and data by grouping functions, constants, and variables by the class name. This is useful in larger projects by creating more structured and readable code. Figures 1, 2.1, and 2.2 show the two classes used in the program. The FileProcessor (Figure 1) class houses the processing functions (like opening the file, writing, reading, and closing the file) while the IO (Figures 2.1 and 2.2) class houses all the input and output functions that the user interacts with. The main functions in the FileProcessor class allow the user to open the file and read the data that is currently in the file. If there is not a file an error is thrown. If successfully opened, the next function writes data to the file. For the IO class functions, the menu options are defined and the input from the user is guided by a function that asks for specific information using error handling.



```python

# Processing with Classes
class FileProcessor:  2 usages
    """..."""
    @staticmethod  1 usage
    def read_data_from_file(file_name: str, student_data: list):
        """..."""
        try:
            file = open(file_name, "r")
            student_data = json.load(file)
            file.close()
        # Error handling below
        except FileNotFoundError as e:
            IO.output_error_message(message = "Text file not found")
        except Exception as e:
            IO.output_error_messages( message: "There was a non-specific error", e)
        finally:
            if not file.closed:
                file.close()
        return student_data

    @staticmethod  1 usage
    def write_data_to_file(file_name: str, student_data: list):
        # Removed global file and student variables
        """..."""
        try:
            file = open(file_name, "w")
            json.dump(student_data, file)
            file.close()
            print("The following students have been registered: ")
            IO.output_student_courses(student_data)
        except TypeError as e:
            IO.output_error_messages( message: "Please check that the data is in valid JSON format", e)
        except Exception as e:
            IO.output_error_messages( message: "There was a non-specific error", e)
        finally:
            if not file.closed:
                file.close()

```

Figure 1: FileProcessor Class. This is the class that holds processing functions.

```python
class IO:  12 usages
    """..."""

    @staticmethod  6 usages
    def output_error_messages(message :str, exception: Exception = None):
        """..."""
        print(message)
        if exception is not None:
            print("-- Technical Error Message -- ")
            print(exception, exception.__doc__, type(exception), sep='\n')

    @staticmethod  1 usage
    def output_menu(menu: str):
        """..."""
        print() # Adding extra space to make it look nicer.
        print(menu)
        print()  # Adding extra space to make it look nicer.

    @staticmethod  1 usage
    def input_menu_choice():
        """..."""
        try:
            menu_choice = input("What would you like to do: ")
            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
```

Figure 2.1: IO Class. This is the class that holds the input and output functions and data. The functions listed above handle errors and allow the user to input what menu option they would like to choose.

```python
@staticmethod  2 usages
def output_student_courses(student_data: list):
    """..."""
    print("-" * 50)
    for student in student_data:
        print(f'Student {student["FirstName"]} '
              f'{student["LastName"]} is enrolled in {student["Course"]}')
    print("-" * 50)

@staticmethod  1 usage
def input_student_data(student_data: list):
    """..."""

    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,
                        "Course": 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(message="Invalid Data Type")
    except Exception as e:
        IO.output_error_messages(message="Error: There was a problem with your entered data.")
    return student_data
```

Figure 2.2: IO Class. This is the class that holds the input and output functions and data. The functions in this portion of the code allow the user to see the data in the file and add information to it.

Simplifying the Data Processing

By using classes and functions you can minimize the code that is used to perform the tasks the code is set to accomplish. Instead of a long code that the user must sift through, the result is much cleaner and more user friendly. Figure 3 shows a simplified version of data processing. In previous assignments our functions would be large blocks of code that were difficult to sort through for handling errors. In this manner, the functioning portion of the code that completes the task is simpler so you can troubleshoot using the class functions and not mess with the structure of the code below.

```python
# 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!
        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  # out of the loop

print("Program Ended")
```

Figure 3: Simpler data processing by using classes and embedded functions

Testing the code in PyCharm and the command line

Figures 4 and 5 show the script working in PyCharm and in the command line. The output for both creates a compatible format for JSON files. You can achieve the same result as previous assignments with classes and functions making your script more easily navigable.

```
[{"FirstName": "Sam", "LastName": "Smith", "Course": "Python 100"},
 {"FirstName": "Teri", "LastName": "James", "Course": "Python 200"},
 {"FirstName": "Susie", "LastName": "Q", "Course": "Python 500"},
 {"FirstName": "Hailey", "LastName": "Mae", "Course": "Python 100"}]
```

Figure 4. PyCharm output in .json file format

Figure 5: Running in the command line.

Summary

Using classes and functions makes code easier to read. Blocks of code can be put into functions that can then be stored in classes. This allows the calling of functions in a more efficient way. A takeaway for me in this week's assignment was to watch out for key errors. My .json file had a different name using 'Course' instead of 'CourseName', but I was able to figure out eventually how to fix that and my code ran smoothly. Using the error message from PyCharm I was able to work backwards to figure this out and swap to match the code in my script to what was listed in my file.