Keegan McGuire

8/19/2025

Foundations of Python Programming

Mod06 - Assignment06

https://github.com/Keegan-McGuire/IntroToProg-Python-Mod06

Functions, Parameters, and Classes

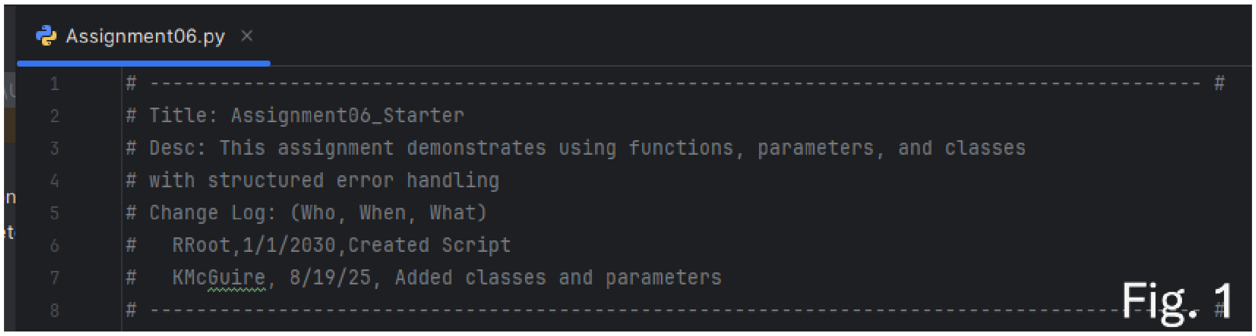
Creating a more modular structure with Functions and Classes

This week, our continuing class registration program stayed largely the same from an outside perspective, although the internals of it would end up looking quite changed by the end of the task. That's because this week we would transition from a structure that felt mostly linear into one that defined many of its important aspects early on, then executed those in a set of commands that would end up much cleaner.

Establishing this structure would depend on converting much of the code into functions, adding parameters to it, and then establishing those items in classes. Once those steps were completed, the classes could then be deployed into the code structure we had already built.

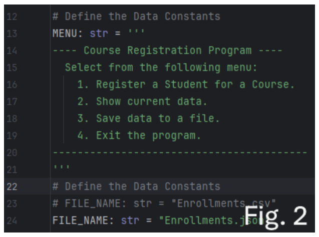
Establishing the framework

Just like last week, Professor Root emphasized the importance of upgrading established code instead of starting from scratch to meet the assignment criteria, so I started by opening up the Assignment06Starter.py file and resaving it as Assignment06.py (Fig 1.). I filled out the header and began to work my way through the assignment criteria.



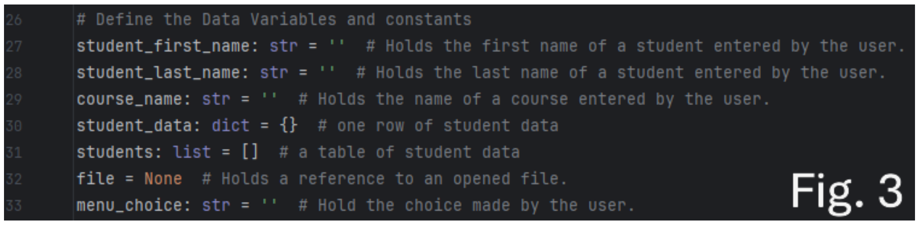
```
# ---------------------------------------------------------------------------------------- #
# Title: Assignment06_Starter
# Desc: This assignment demonstrates using functions, parameters, and classes
# with structured error handling
# Change Log: (Who, When, What)
#   RRoot,1/1/2030,Created Script
#   KMcGuire, 8/19/25, Added classes and parameters
# ---------------------------------------------------------------------------------------- #
```

By this point, much of our early criteria weren't going to change much, so a quick glance at these elements (Fig. 2, Fig 3.) told me I could probably move on to what was going to be the biggest hurdle for this project – building the classes.



```
# Define the Data Constants
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.
-----------------------------------------
'''
# Define the Data Constants
# FILE_NAME: str = "Enrollments.csv"
FILE_NAME: str = "Enrollments.json"
```

Fig. 2



```
# Define the Data Variables and constants
student_first_name: str = ''  # Holds the first name of a student entered by the user.
student_last_name: str = ''  # Holds the last name of a student entered by the user.
course_name: str = ''  # Holds the name of a course entered by the user.
student_data: dict = {}  # one row of student data
students: list = []  # a table of student data
file = None  # Holds a reference to an opened file.
menu_choice: str = ''  # Hold the choice made by the user.
```

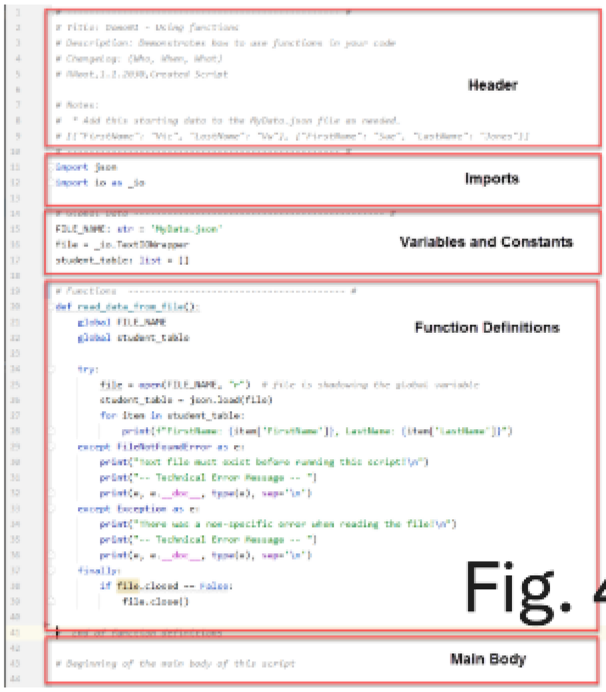
Fig. 3

Creating Classes

From the way the notes and the labs progressed, I knew that I ultimately wanted to modify some of the specific operations I had built in previous assignments to do the same tasks as before, but with increased modularity after being converted first into functions, assigned parameters, and then established as classes.

How I was going to accomplish this was at the time unclear to me. Following the outline provided by the sequence of the labs seemed like the simplest approach, but the time investment involved with completing all three had been high, and the assignment criteria ended with the suggestion that the process of completing this assignment was similar to the process of completing Lab03.

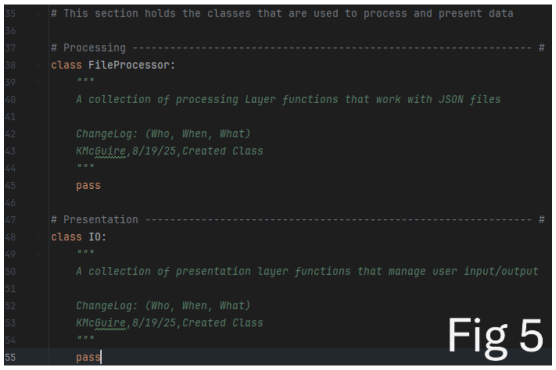
However, the classes were the first new item introduced by the assignment criteria, and I knew I wanted something that would eventually resemble the visual outline provided at the beginning of the notes (Fig. 4), so I decided to build some passed through class types at the very least.



```
# ------------------------------------------------------------------ #
# Title: Demo03 - Using functions
# Description: Demonstrates how to use functions in your code
# ChangeLog: (Who, When, What)
# RRoot,1.1.2030,Created Script

# Notes:
#  * Add this starting data to the MyData.json file as needed.
# [{"FirstName": "Vic", "LastName": "Vu"}, {"FirstName": "Sue", "LastName": "Jones"}]
# ------------------------------------------------------------------ #
import json
import io as _io

# Global Data --------------------------------------------------------- #
FILE_NAME: str = 'MyData.json'
file = _io.TextIOWrapper
student_table: list = []

# Functions ----------------------------------------------- #
def read_data_from_file():
    global FILE_NAME
    global student_table

    try:
        file = open(FILE_NAME, "r")  # file is shadowing the global variable
        student_table = json.load(file)
        for item in student_table:
            print(f"FirstName: {item['FirstName']}, LastName: {item['LastName']}")
    except FileNotFoundError as e:
        print("Text file must exist before running this script!\n")
        print("-- Technical Error Message -- ")
        print(e, e.__doc__, type(e), sep='\n')
    except Exception as e:
        print("There was a non-specific error when reading the file!\n")
        print("-- Technical Error Message -- ")
        print(e, e.__doc__, type(e), sep='\n')
    finally:
        if file.closed == False:
            file.close()

# End of function definitions

# Beginning of the main body of this script
```

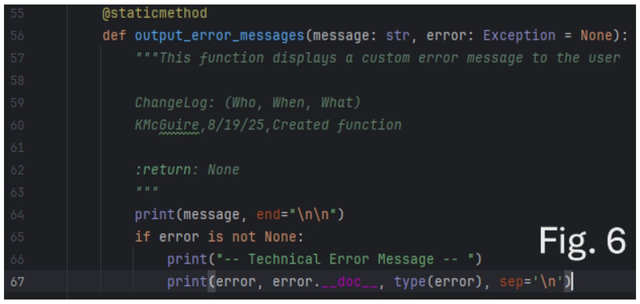
Fig. 4

Using the examples provided in Lab03, I built out two classes (Fig. 5) that I could add functionality to as I continued to work through the assignment. These would be the FileProcessor class and the IO class. The FileProcessor class would work with my JSON files, and the IO class would be responsible for monitoring user inputs and displaying information to the user. Once that was completed, I decided to remove the pass function from the IO file and replace it with a custom error message (Fig 6), just as we had done in Lab03.



```python
# This section holds the classes that are used to process and present data

# Processing --------------------------------------------------------------- #
class FileProcessor:
    """
    A collection of processing Layer functions that work with JSON files

    ChangeLog: (Who, When, What)
    KMcGuire,8/19/25,Created Class
    """
    pass

# Presentation ------------------------------------------------------------- #
class IO:
    """
    A collection of presentation layer functions that manage user input/output

    ChangeLog: (Who, When, What)
    KMcGuire,8/19/25,Created Class
    """
    pass
```

Fig 5

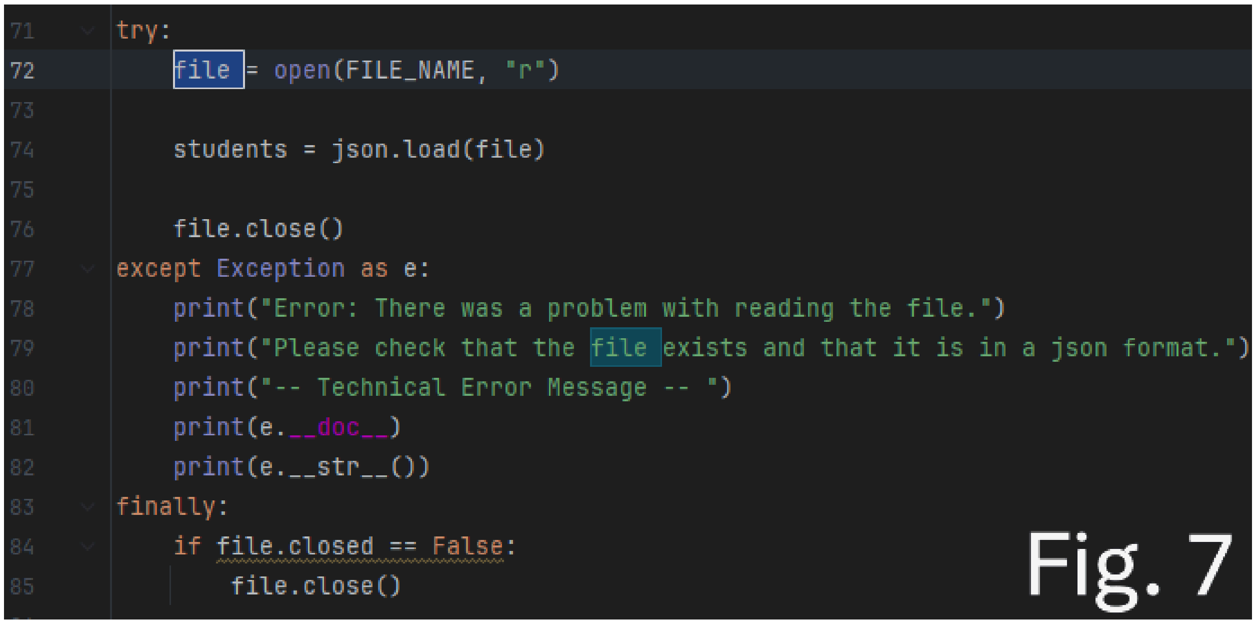


```python
    @staticmethod
    def output_error_messages(message: str, error: Exception = None):
        """This function displays a custom error message to the user

        ChangeLog: (Who, When, What)
        KMcGuire,8/19/25,Created function

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

Fig. 6

Preparing Function for Classes

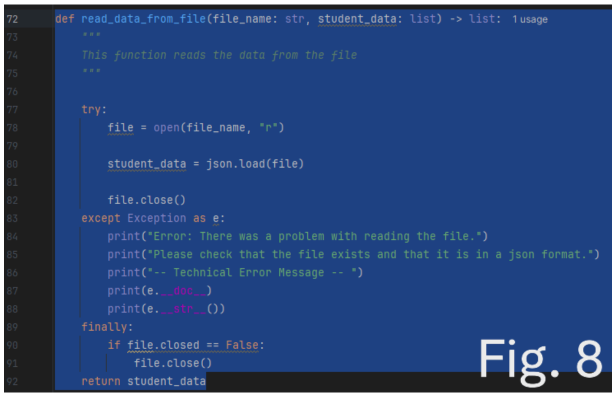
Now I knew it was time to start taking the sections of code that we had worked so hard on for the last five weeks and start moving them to their new home – the classes created above. I was feeling nervous about disassembling my menu items, so I decided to start with something that seemed a little simpler, the code I already had that read my JSON file (Fig. 7).



```python
try:
    file = open(FILE_NAME, "r")

    students = json.load(file)

    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()
```

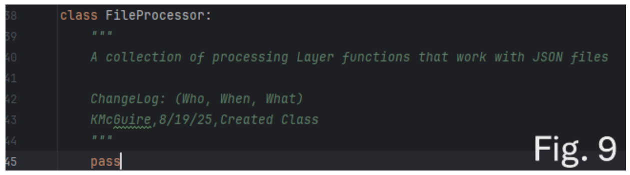
Fig. 7

I added the code that I needed to define that as a function, wrote a description, and indented what was already there to convert it to its final form. With that out of the way, I copied what I had (Fig. 8), went up to my FileProcessor class and pasted it over the temporary pass command (Fig. 9). I also knew I would need to add code to call that function, so I built that into the main body of my script (Fig. 10).



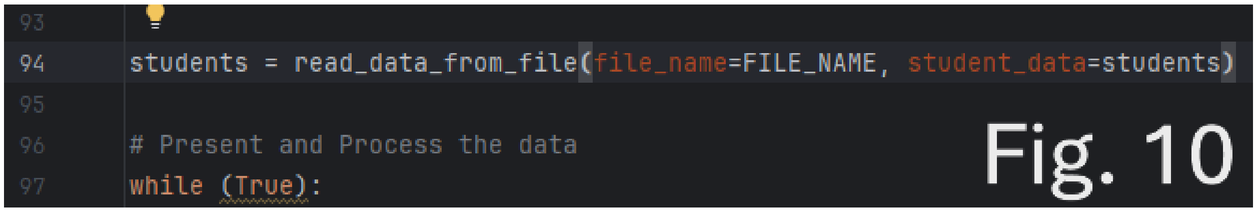
```python
def read_data_from_file(file_name: str, student_data: list) -> list:  1 usage
    """
    This function reads the data from the file
    """

    try:
        file = open(file_name, "r")

        student_data = json.load(file)

        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()
    return student_data
```

Fig. 8



```python
class FileProcessor:
    """
    A collection of processing Layer functions that work with JSON files

    ChangeLog: (Who, When, What)
    KMcGuire,8/19/25,Created Class
    """
    pass
```

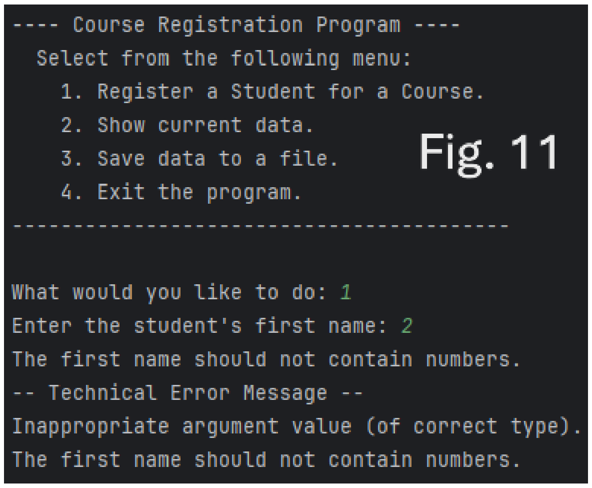
Fig. 9



```python

students = read_data_from_file(file_name=FILE_NAME, student_data=students)

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

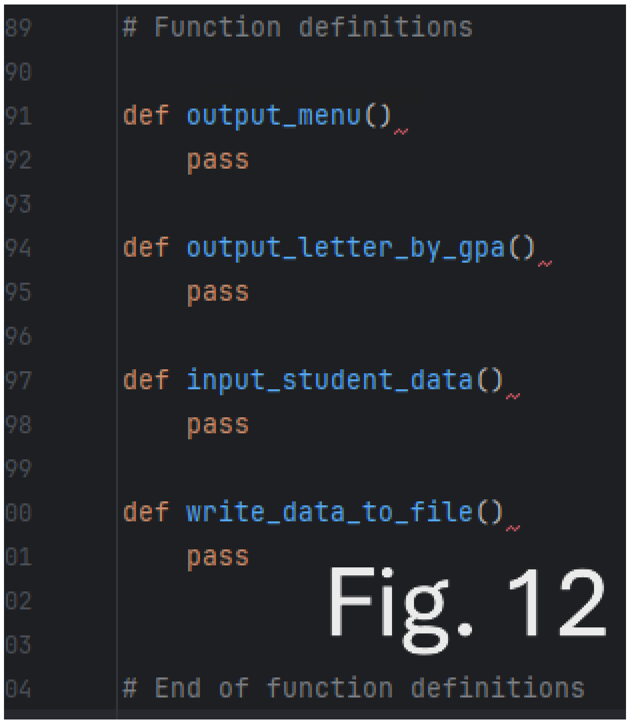
Fig. 10

At this point, I wasn't entirely sure if the many modifications I had made to the code were going to cooperate, but I ran it and sure enough – it seemed to be working (Fig. 11).



```
---- 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.
-----------------------------------------

What would you like to do: 1
Enter the student's first name: 2
The first name should not contain numbers.
-- Technical Error Message --
Inappropriate argument value (of correct type).
The first name should not contain numbers.
```

Fig. 11



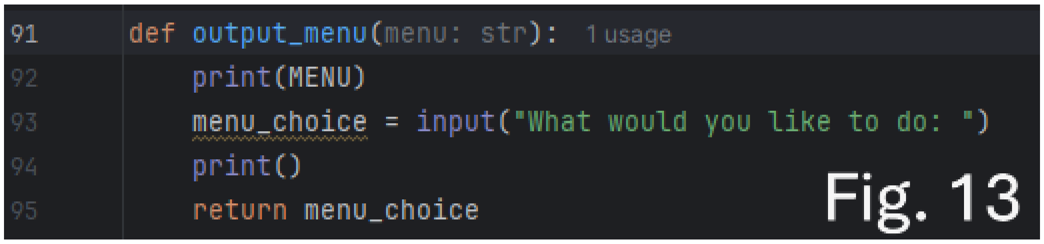
```python
# Function definitions

def output_menu()
    pass

def output_letter_by_gpa()
    pass

def input_student_data()
    pass

def write_data_to_file()
    pass



# End of function definitions
```

Fig. 12

Now it was time to finally start moving my menu items up to the IO class. I wanted to stage this as we'd done in the labs, so I began by creating some empty definitions for the functions that I would then pass to my classes (Fig. 12). I created these by copying the examples we had worked through in Lab01, which were close but not exactly correct, which is something I would have to correct at the end of the assignment.

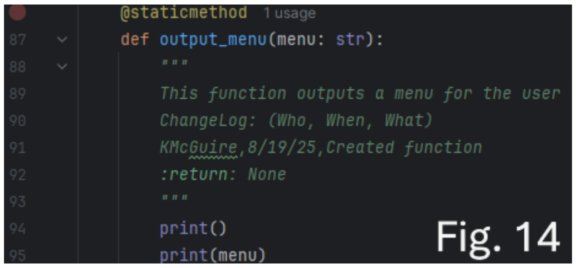
First I moved my output menu instructions up to create a function. I had intended to move everything up one by one to build up my functions in their entirety and then plug them in to the classes at the end. However, I quickly started to feel confused and lost jumping from one section of the notes to the others, and decided to simply stage them and build them out into the final format that I had from Lab03. This likely wasn't the most correct way to accomplish this, but I felt like it ultimately reduced the opportunity to introduce errors.

To that end, I simply moved my output_menu commands (Fig. 13) up to the function framework I had created, then went ahead and plugged it into the section of the IO class (Fig. 14). After that was complete, I returned to the body of the code and instructed it to use the class to run the program (Fig. 15).



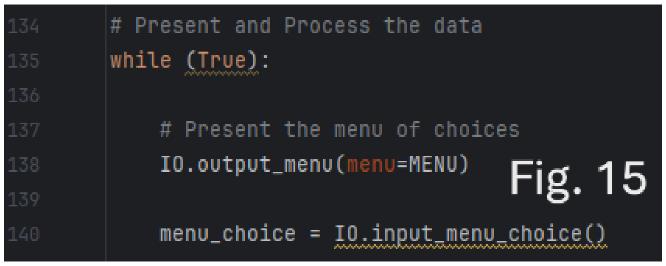
```python
def output_menu(menu: str):  1 usage
    print(MENU)
    menu_choice = input("What would you like to do: ")
    print()
    return menu_choice
```

Fig. 13



```python
@staticmethod  1 usage
def output_menu(menu: str):
    """
    This function outputs a menu for the user
    ChangeLog: (Who, When, What)
    KMcGuire,8/19/25,Created function
    :return: None
    """
    print()
    print(menu)
```

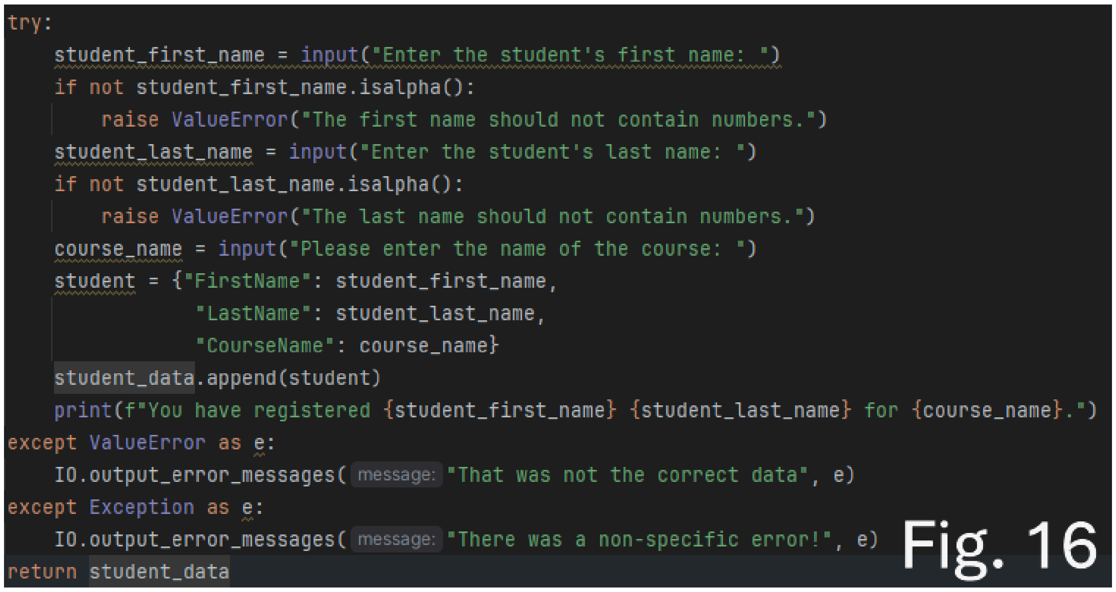
Fig. 14



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

    # Present the menu of choices
    IO.output_menu(menu=MENU)

    menu_choice = IO.input_menu_choice()
```

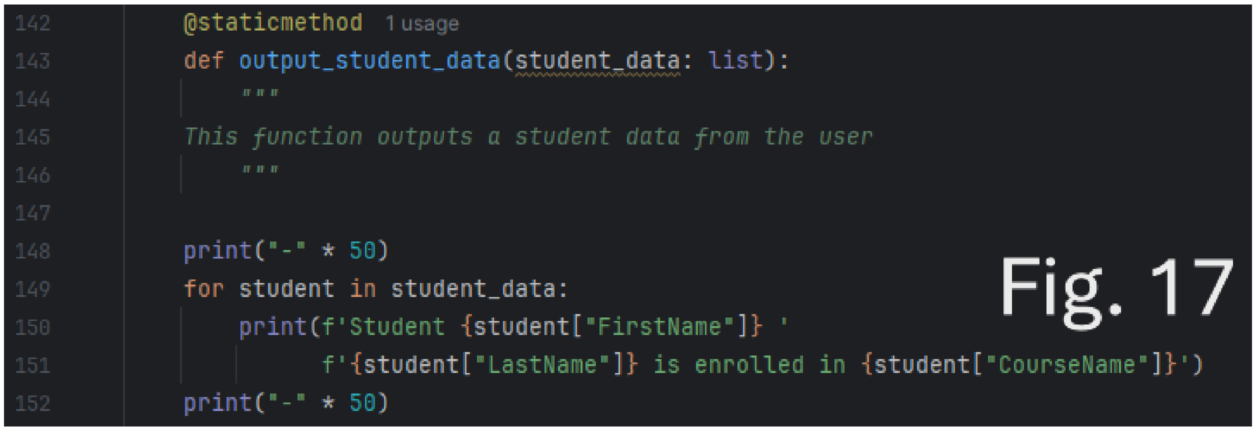
Fig. 15

This was only the beginning, as I would perform this same operation for each of the menu items except for the always amenable Menu item 4. This meant that I would plug in my code to input student data (Fig. 16), output student data (Fig. 17), and write the data to the file (Fig. 18).



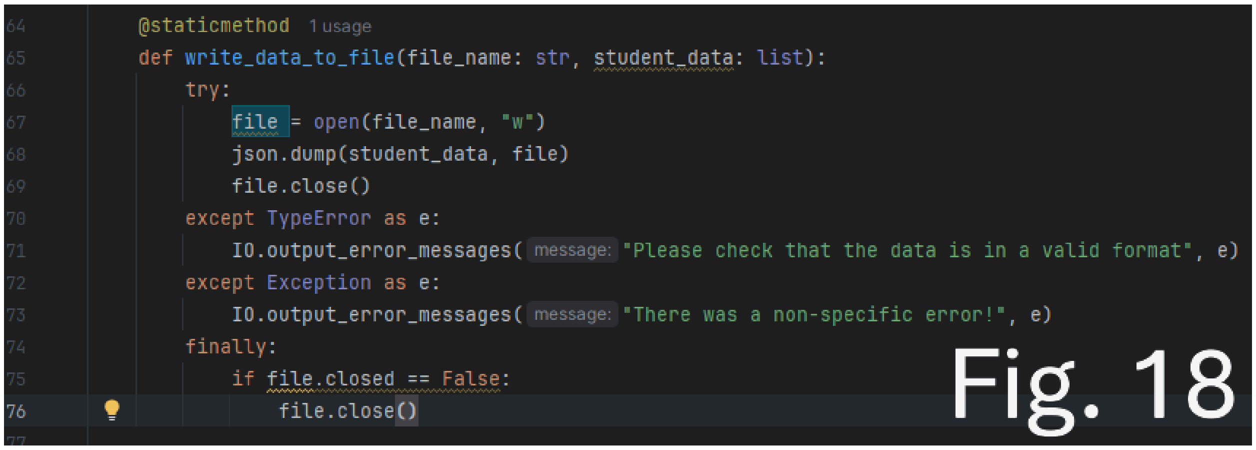
```python
try:
    student_first_name = input("Enter the student's first name: ")
    if not student_first_name.isalpha():
        raise ValueError("The first name should not contain numbers.")
    student_last_name = input("Enter the student's last name: ")
    if not student_last_name.isalpha():
        raise ValueError("The last name should not contain numbers.")
    course_name = input("Please enter the name of the course: ")
    student = {"FirstName": student_first_name,
               "LastName": student_last_name,
               "CourseName": course_name}
    student_data.append(student)
    print(f"You have registered {student_first_name} {student_last_name} for {course_name}.")
except ValueError as e:
    IO.output_error_messages( message: "That was not the correct data", e)
except Exception as e:
    IO.output_error_messages( message: "There was a non-specific error!", e)
return student_data
```

Fig. 16



```
    @staticmethod  1 usage
    def output_student_data(student_data: list):
        """
    This function outputs a student data from the user
        """

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

Fig. 17



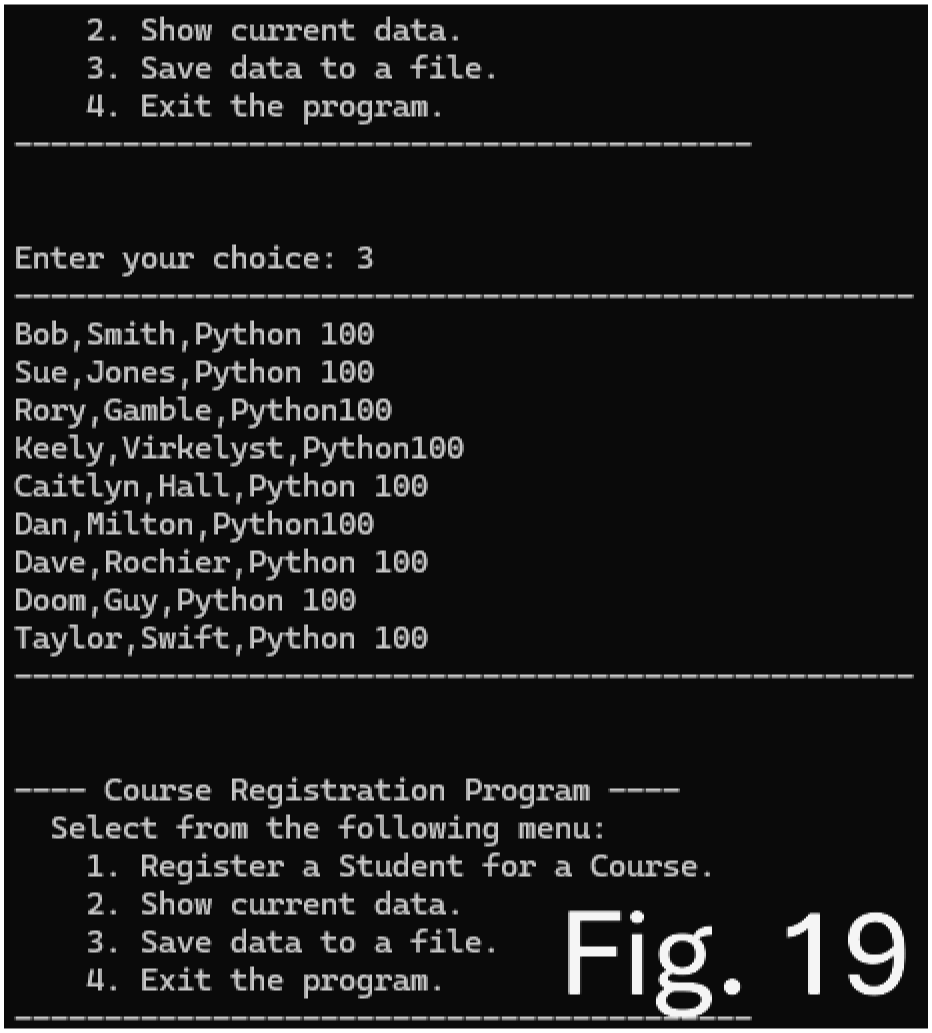
```
    @staticmethod  1 usage
    def write_data_to_file(file_name: str, student_data: list):
        try:
            file = open(file_name, "w")
            json.dump(student_data, file)
            file.close()
        except TypeError as e:
            IO.output_error_messages( message: "Please check that the data is in a valid 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()
```

Fig. 18

Testing the Program

To a certain extent, I feel like I'm not doing this entire process justice by not explaining it step by step, but the reality is that it was similar for each element. It – and all the troubleshooting that went along with it – didn't always feel that way, but it went pretty smoothly once I had figured out a method to get all the pieces where they needed to be and talk to one another.

However, it was working – everything seemed to be communicating, saving new data to a file, and displaying what it was supposed to. It even functioned without any issues in the command prompt (Fig. 19), which always feels like the true test of a new program.



```
    2. Show current data.
    3. Save data to a file.
    4. Exit the program.
-----------------------------------------



Enter your choice: 3
--------------------------------------------------
Bob,Smith,Python 100
Sue,Jones,Python 100
Rory,Gamble,Python100
Keely,Virkelyst,Python100
Caitlyn,Hall,Python 100
Dan,Milton,Python100
Dave,Rochier,Python 100
Doom,Guy,Python 100
Taylor,Swift,Python 100
--------------------------------------------------



---- 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.
-----------------------------------------
```

Fig. 19

Summary

While this assignment introduced a lot of new concepts, in the end we had the same program functioning in a much more modular, and possibly elegant, way. The functions inside the classes interacted with the necessary parameters to deliver necessary operations in a modular deployment.

The process of refactoring this code was somewhat lengthy, but once things clicked it did at least make sense how each portion interacted with one another. This did seem like something of a leap from the steps we were learning before, but I am excited to see how they will set us up for the remainder of the course.