Rob Armstrong

August 14, 2024

IT FDN 110 A

Assignment07

https://github.com/robstrong517/IntroToProg-Python-Mod07

Intro to Programming with Python

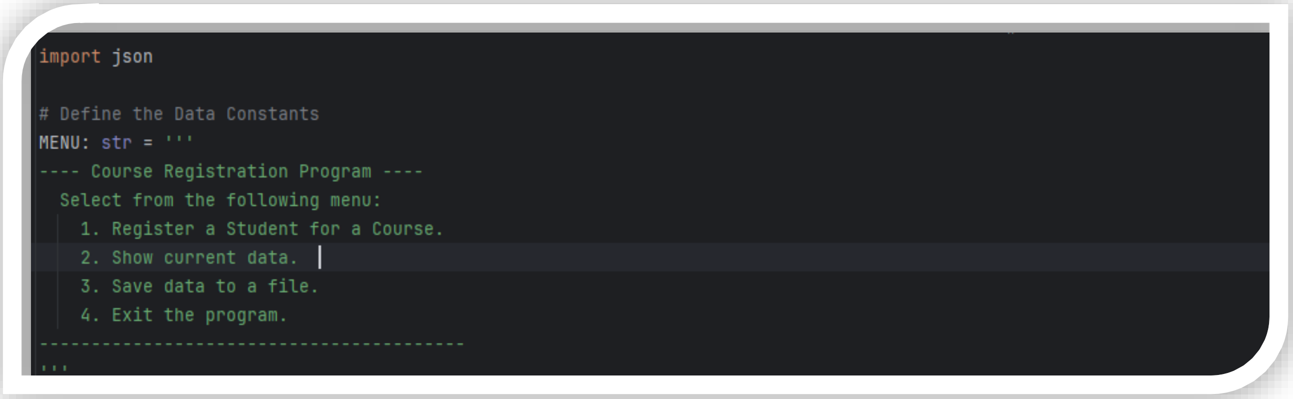
Module 07 – Classes and Objects

Introduction

Since we have been learning to reference functions in the main body of our script, which allows for reiterative use of a specific line of code, and also keeps the script more organized, this assignment took these reference concepts a step further by introducing classes and objects.

Writing the Code

As it has been every week, compounding more and more with each new idea learned, this assignment required operational definitions for every class type. These class are described more below.

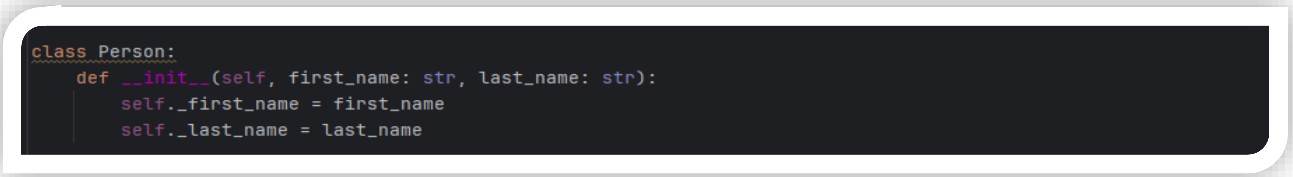


```
import json

# 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.
-----------------------------------------
'''
```

FIGURE 1: CONSTANTS USED IN ASSIGNMENT 07.

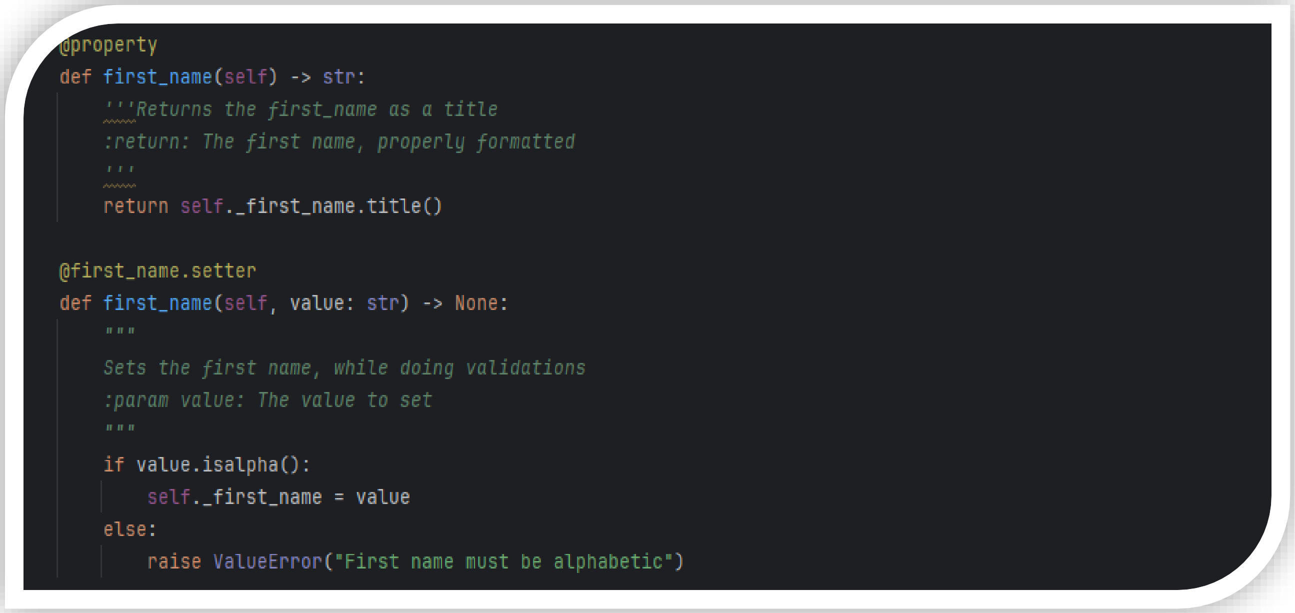
Data Classes

As mentioned above in the section for Writing the Code, constructors are an important part of data classes. This allows for objects to be created out of a parent class. In this assignment, the parent class was “Person”. A secondary class, derived from the parent class, had been labeled as “Student”. Student has the same properties as the “Person” class, however, it amends the data and adds an additional variable to each Person. That additional variable is Course Name, which makes “Student” a role for a “Person”.



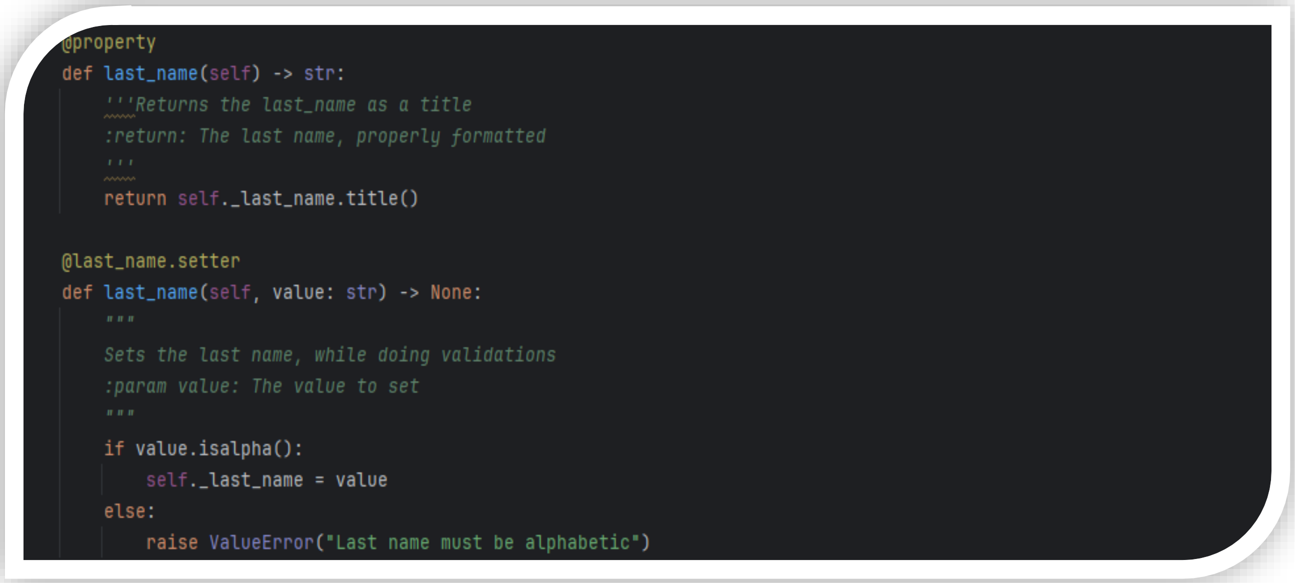
```
class Person:
    def __init__(self, first_name: str, last_name: str):
        self._first_name = first_name
        self._last_name = last_name
```

FIGURE 2: DEFINITION OF FIRST CLASS TO BE USED, “PERSON”.



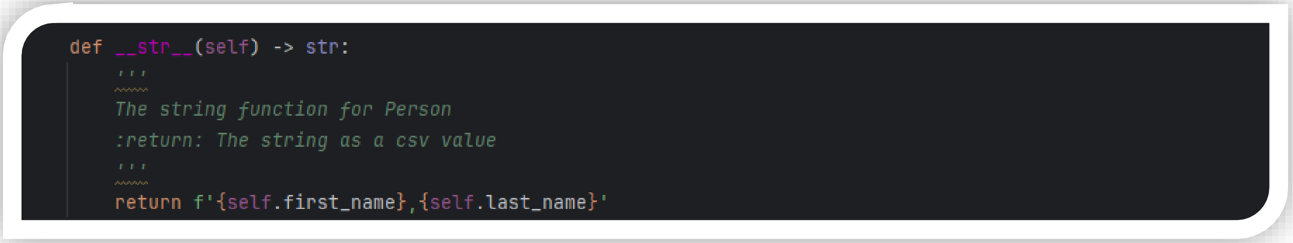
```python
@property
def first_name(self) -> str:
    '''Returns the first_name as a title
    :return: The first name, properly formatted
    '''
    return self._first_name.title()

@first_name.setter
def first_name(self, value: str) -> None:
    """
    Sets the first name, while doing validations
    :param value: The value to set
    """
    if value.isalpha():
        self._first_name = value
    else:
        raise ValueError("First name must be alphabetic")
```

FIGURE 3: FIRST NAME PROPERTIES AND SETTER FOR CLASS "PERSON".



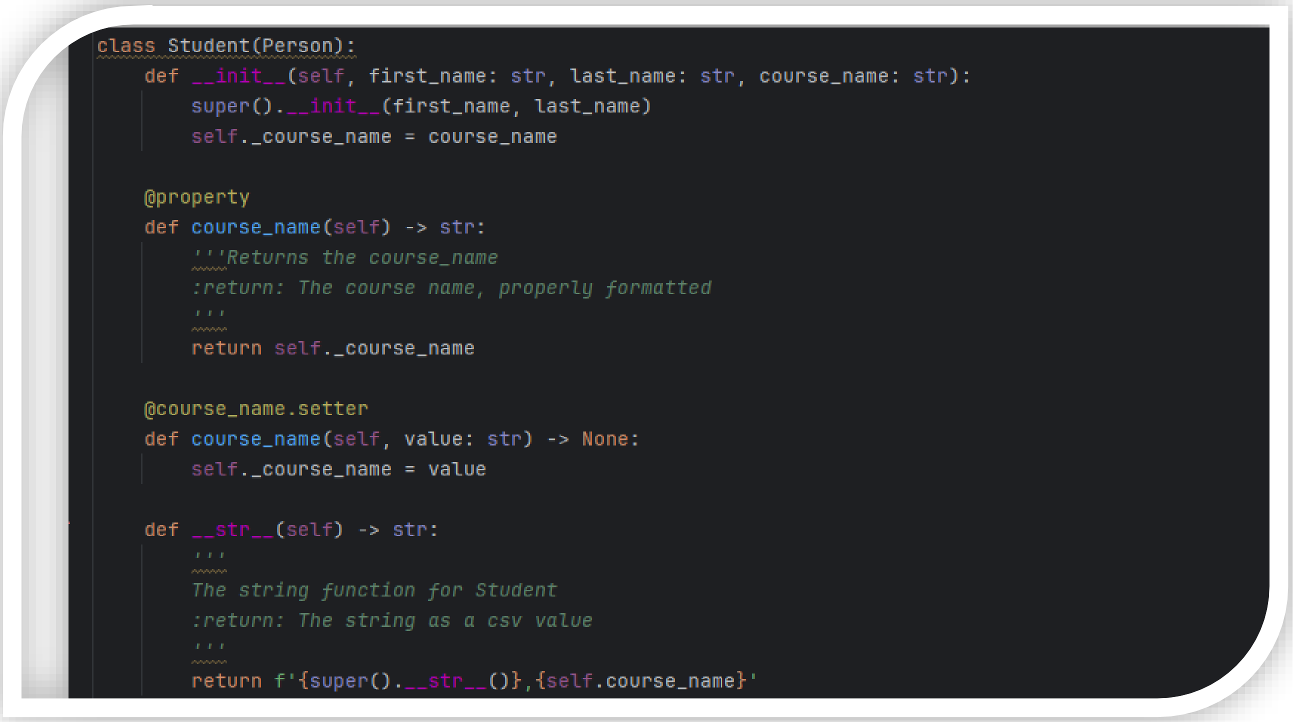
```python
@property
def last_name(self) -> str:
    '''Returns the last_name as a title
    :return: The last name, properly formatted
    '''
    return self._last_name.title()

@last_name.setter
def last_name(self, value: str) -> None:
    """
    Sets the last name, while doing validations
    :param value: The value to set
    """
    if value.isalpha():
        self._last_name = value
    else:
        raise ValueError("Last name must be alphabetic")
```

FIGURE 4: LAST NAME PROPERTIES AND SETTER FOR CLASS "PERSON".



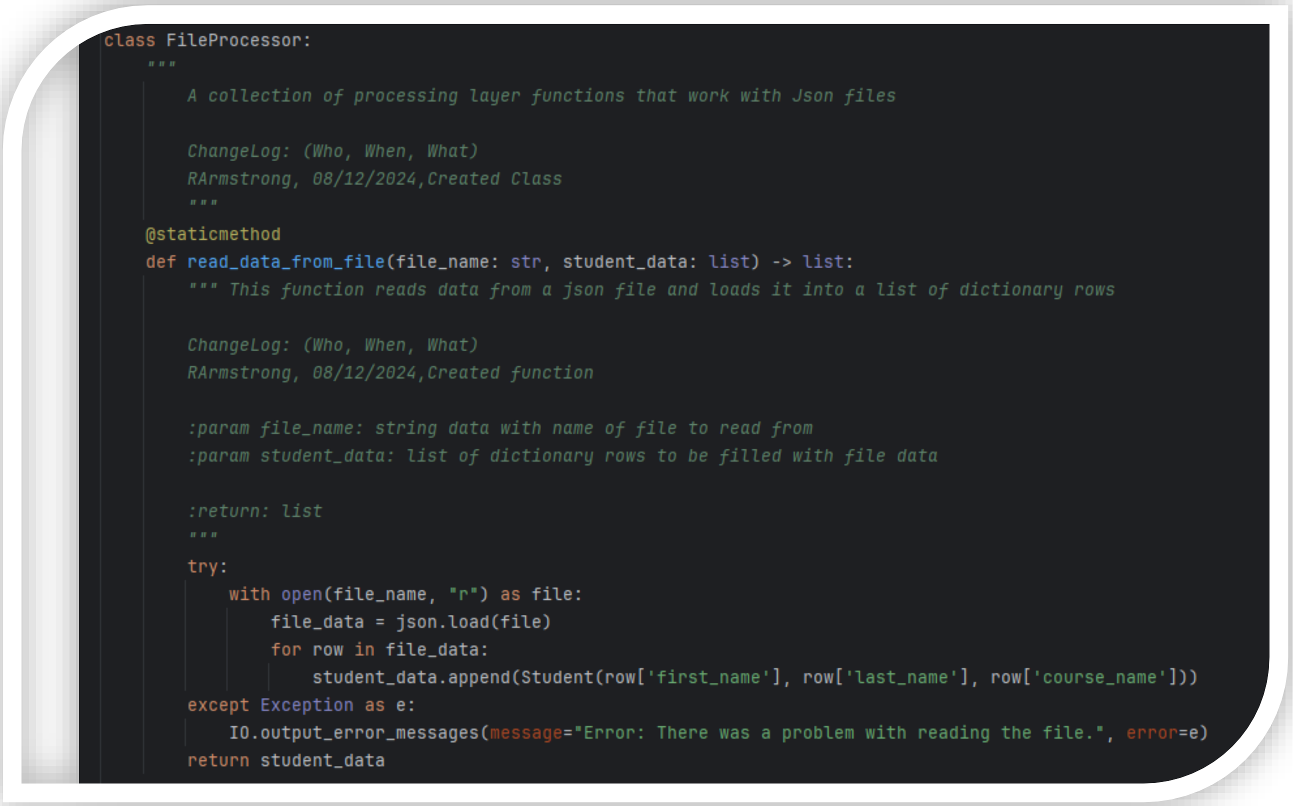
```python
def __str__(self) -> str:
    '''
    The string function for Person
    :return: The string as a csv value
    '''
    return f'{self.first_name},{self.last_name}'
```

FIGURE 5: FUNCTION TO RETURN THE STRING AS CSV.



```python
class Student(Person):
    def __init__(self, first_name: str, last_name: str, course_name: str):
        super().__init__(first_name, last_name)
        self._course_name = course_name

    @property
    def course_name(self) -> str:
        '''Returns the course_name
        :return: The course name, properly formatted
        '''
        return self._course_name

    @course_name.setter
    def course_name(self, value: str) -> None:
        self._course_name = value

    def __str__(self) -> str:
        '''
        The string function for Student
        :return: The string as a csv value
        '''
        return f'{super().__str__()},{self.course_name}'
```

FIGURE 6: CLASS OF "STUDENT" WITH DATA APPENDED FROM "PERSON" AND AUGMENTED.

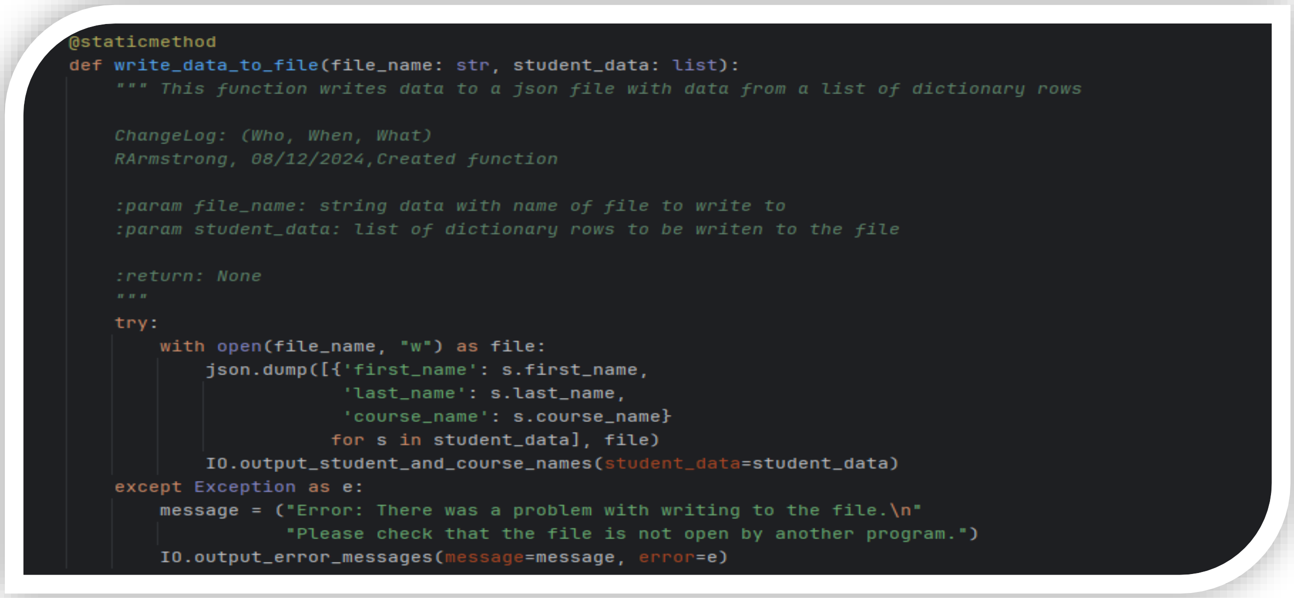


```python
class FileProcessor:
    """
    A collection of processing layer functions that work with Json files

    ChangeLog: (Who, When, What)
    RArmstrong, 08/12/2024,Created Class
    """
    @staticmethod
    def read_data_from_file(file_name: str, student_data: list) -> list:
        """ This function reads data from a json file and loads it into a list of dictionary rows

        ChangeLog: (Who, When, What)
        RArmstrong, 08/12/2024,Created function

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

        :return: list
        """
        try:
            with open(file_name, "r") as file:
                file_data = json.load(file)
                for row in file_data:
                    student_data.append(Student(row['first_name'], row['last_name'], row['course_name']))
        except Exception as e:
            IO.output_error_messages(message="Error: There was a problem with reading the file.", error=e)
        return student_data
```

FIGURE 7: CLASS FOR FILEPROCESSOR, TO INCLUDE READ_DATA COMMANDS AND ERROR HANDLING.

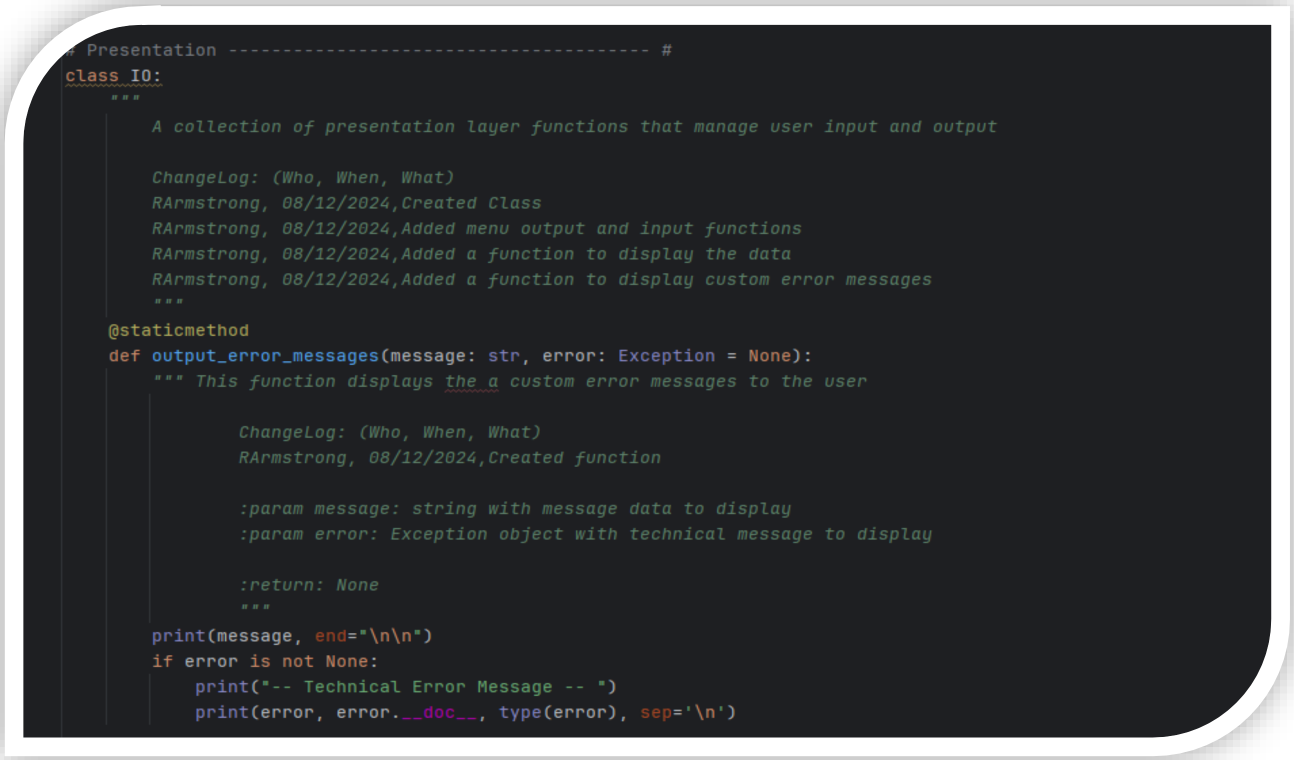
Processing and Presentation Classes

Similarly as with the Data Class section, processing and presentation classes require operational definitions as well. This was not done much differently than it was in Assignment 06. Each of these subcategories starts with the definition of the command, provides the item type, and then includes a small section of code that is unique to the function. This is where the commands for menu output, menu selection, and the respective menu options are housed, to name a few.



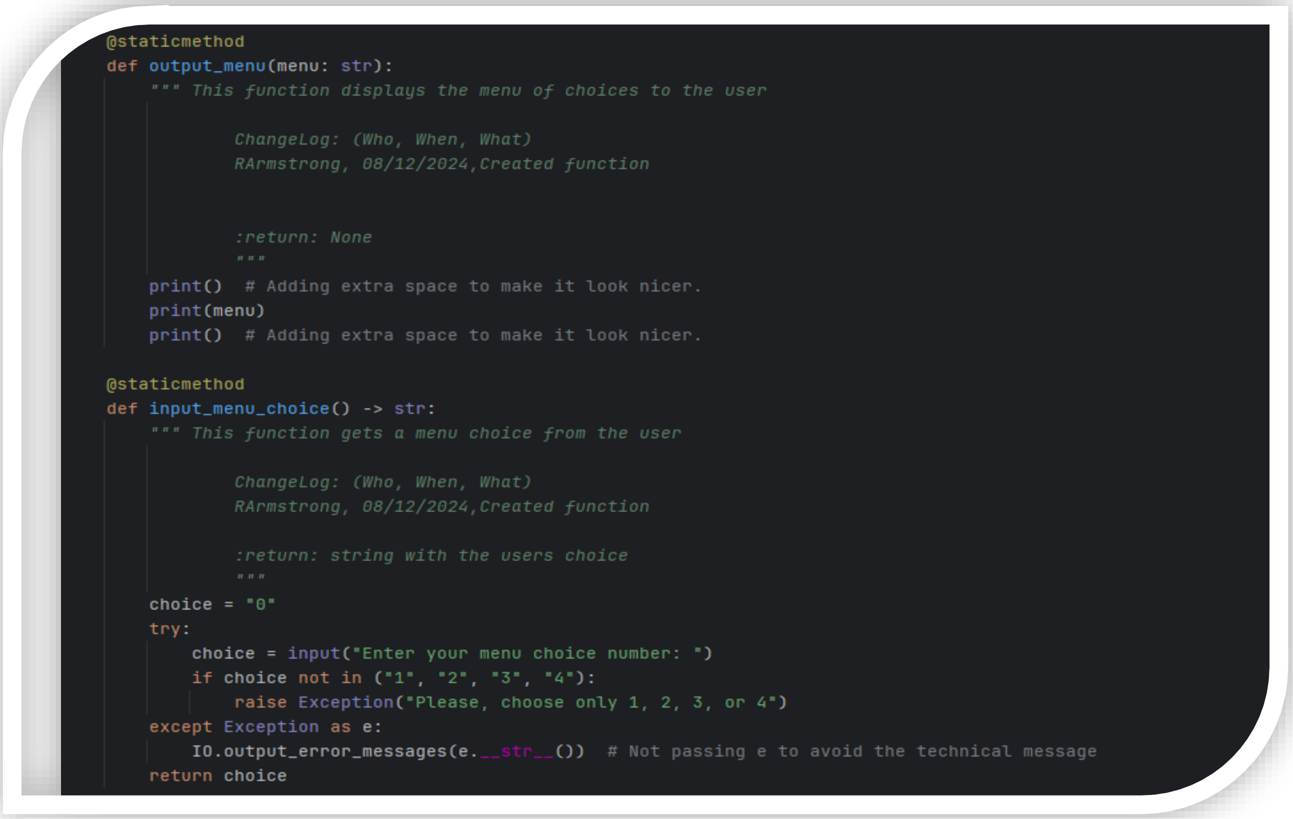
```python
@staticmethod
def write_data_to_file(file_name: str, student_data: list):
    """ This function writes data to a json file with data from a list of dictionary rows

    ChangeLog: (Who, When, What)
    RArmstrong, 08/12/2024,Created function

    :param file_name: string data with name of file to write to
    :param student_data: list of dictionary rows to be writen to the file

    :return: None
    """
    try:
        with open(file_name, "w") as file:
            json.dump([{'first_name': s.first_name,
                        'last_name': s.last_name,
                        'course_name': s.course_name}
                       for s in student_data], file)
            IO.output_student_and_course_names(student_data=student_data)
    except Exception as e:
        message = ("Error: There was a problem with writing to the file.\n"
                   "Please check that the file is not open by another program.")
        IO.output_error_messages(message=message, error=e)
```

FIGURE 8: DEFINITION FOR WRITE_DATA FUNCTION AND ERROR HANDLING.



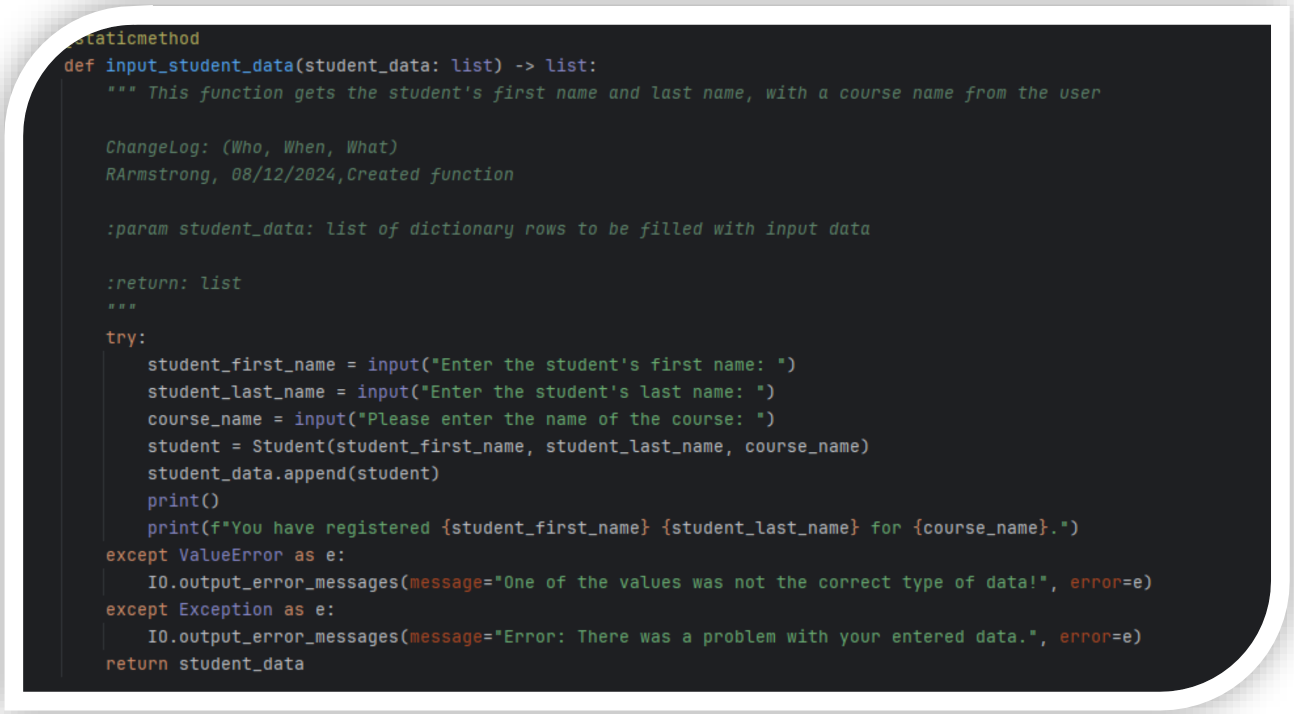
```python
# Presentation --------------------------------------- #
class IO:
    """
    A collection of presentation layer functions that manage user input and output

    ChangeLog: (Who, When, What)
    RArmstrong, 08/12/2024,Created Class
    RArmstrong, 08/12/2024,Added menu output and input functions
    RArmstrong, 08/12/2024,Added a function to display the data
    RArmstrong, 08/12/2024,Added a function to display custom error messages
    """
    @staticmethod
    def output_error_messages(message: str, error: Exception = None):
        """ This function displays the a custom error messages to the user

                ChangeLog: (Who, When, What)
                RArmstrong, 08/12/2024,Created function

                :param message: string with message data to display
                :param error: Exception object with technical message to display

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

FIGURE 9: PRESENTATION CLASS FOR DISPLAYING ERROR MESSAGES.



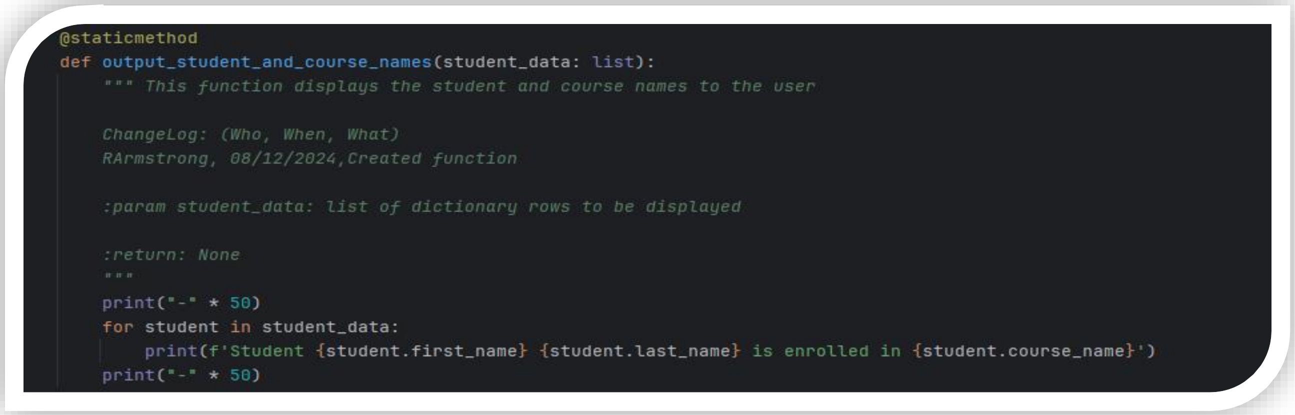
```
@staticmethod
def output_menu(menu: str):
    """ This function displays the menu of choices to the user

            ChangeLog: (Who, When, What)
            RArmstrong, 08/12/2024,Created function


            :return: None
            """
    print()  # Adding extra space to make it look nicer.
    print(menu)
    print()  # Adding extra space to make it look nicer.

@staticmethod
def input_menu_choice() -> str:
    """ This function gets a menu choice from the user

            ChangeLog: (Who, When, What)
            RArmstrong, 08/12/2024,Created function

            :return: string with the users choice
            """
    choice = "0"
    try:
        choice = input("Enter your menu choice number: ")
        if 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__())  # Not passing e to avoid the technical message
    return choice
```

FIGURE 10: PRESENTATION CLASS FOR MENU DISPLAY AND MENU SELECTION DISPLAY.



```
 staticmethod
def input_student_data(student_data: list) -> list:
    """ This function gets the student's first name and last name, with a course name from the user

    ChangeLog: (Who, When, What)
    RArmstrong, 08/12/2024,Created function

    :param student_data: list of dictionary rows to be filled with input data

    :return: list
    """
    try:
        student_first_name = input("Enter the student's first name: ")
        student_last_name = input("Enter the student's last name: ")
        course_name = input("Please enter the name of the course: ")
        student = Student(student_first_name, student_last_name, course_name)
        student_data.append(student)
        print()
        print(f"You have registered {student_first_name} {student_last_name} for {course_name}.")
    except ValueError as e:
        IO.output_error_messages(message="One of the values was not the correct 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 11: PRESENTATION CLASS FOR INPUTTING STUDENT DATA, MENU CHOICE 1.

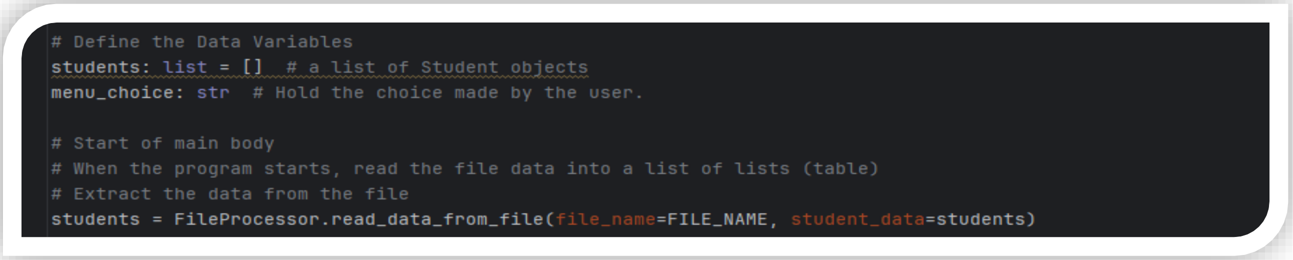


```python
@staticmethod
def output_student_and_course_names(student_data: list):
    """ This function displays the student and course names to the user

    ChangeLog: (Who, When, What)
    RArmstrong, 08/12/2024,Created function

    :param student_data: list of dictionary rows to be displayed

    :return: None
    """
    print("-" * 50)
    for student in student_data:
        print(f'Student {student.first_name} {student.last_name} is enrolled in {student.course_name}')
    print("-" * 50)
```

FIGURE 12: PRESENTATION CLASS FOR DISPLAYING CURRENT STUDENT DATA, MENU CHOICE 2.

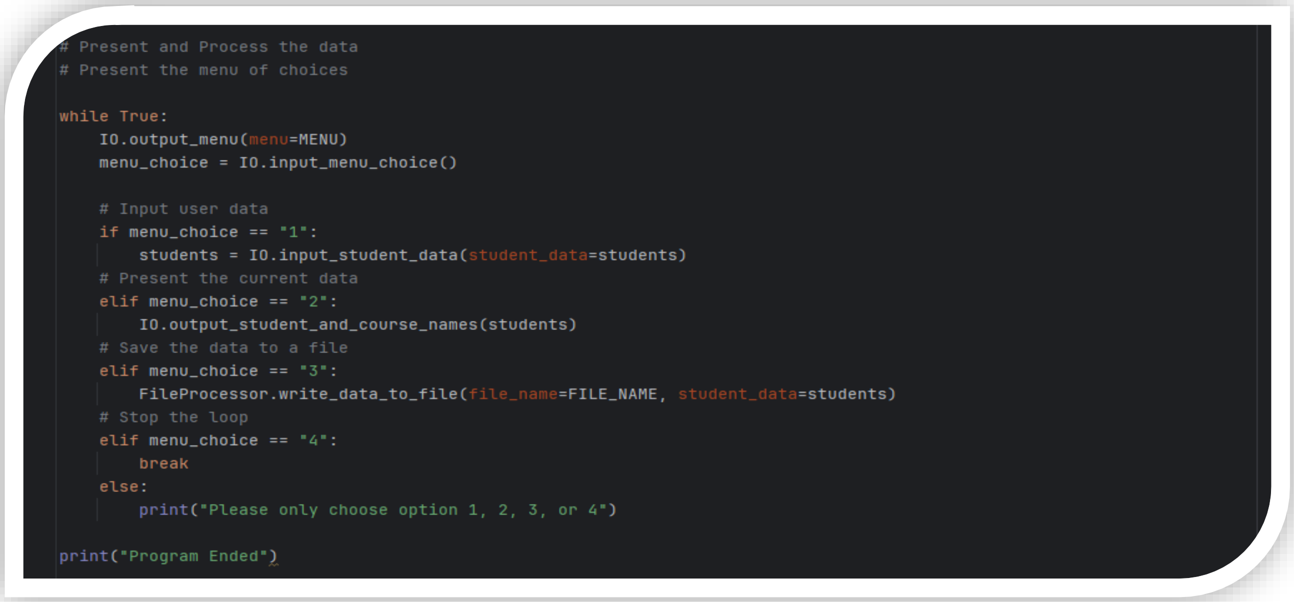
Main Body

Now that the classes have all been defined with all required input/output features, the main body of the script can be written. One last thing to be done, however, is to define the variables. Normally, the variables are near the beginning of the script. The classes and functions used above directly contribute to the building of the target variables. Defining these variables earlier in the script would yield errors, because they had not yet been created.



```python
# Define the Data Variables
students: list = []  # a list of Student objects
menu_choice: str  # Hold the choice made by the user.

# Start of 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)
```

FIGURE 13: DATA VARIABLES, LISTED JUST PRIOR TO BODY BECAUSE IT REFERENCES CLASSES IN PREVIOUS FIGURES.



```python
# Present and Process the data
# Present the menu of choices

while True:
    IO.output_menu(menu=MENU)
    menu_choice = IO.input_menu_choice()

    # Input user data
    if menu_choice == "1":
        students = IO.input_student_data(student_data=students)
    # Present the current data
    elif menu_choice == "2":
        IO.output_student_and_course_names(students)
    # Save the data to a file
    elif menu_choice == "3":
        FileProcessor.write_data_to_file(file_name=FILE_NAME, student_data=students)
    # Stop the loop
    elif menu_choice == "4":
        break
    else:
        print("Please only choose option 1, 2, 3, or 4")

print("Program Ended")
```

FIGURE 14: MAIN BODY SCRIPT.

With these variables working correctly, the main organized body of the script can be built to tie all these different classes, functions, and variables together.

Summary

This was another difficult assignment because the ideas are more abstract than in previous modules. While the basic ideas like read and write to file have become familiar, it took more time to be able to wrap my head around constructors and getting the script to return information correctly. They are still somewhat foreign to me, but in time I'm confident that will improve. A lot of work went into these definitions and classes and relatively little work went into constructing the main body. The idea of a complex code or program stresses me out, given that it would certainly be much more difficult than this practice.