Rachel Hostetler

05/28/2024

Foundations of Programming: Python

Assignment07

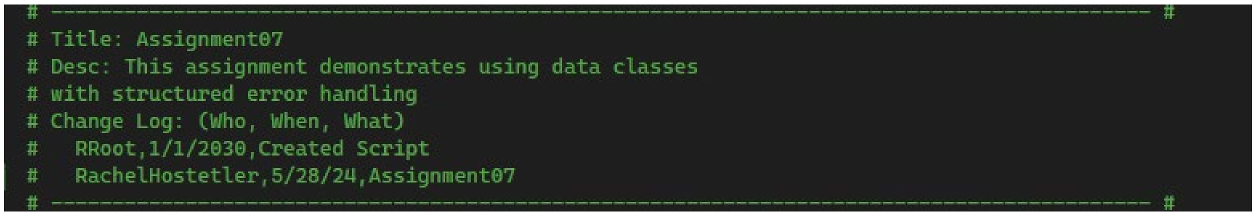
https://github.com/rachelhostetler/IntroToProg-Python

Assignment07

Introduction

This assignment is similar to 06 in that it uses classes to execute functions and menu options, however additional functions are added to the classes, such as getter and setter for each of their properties.

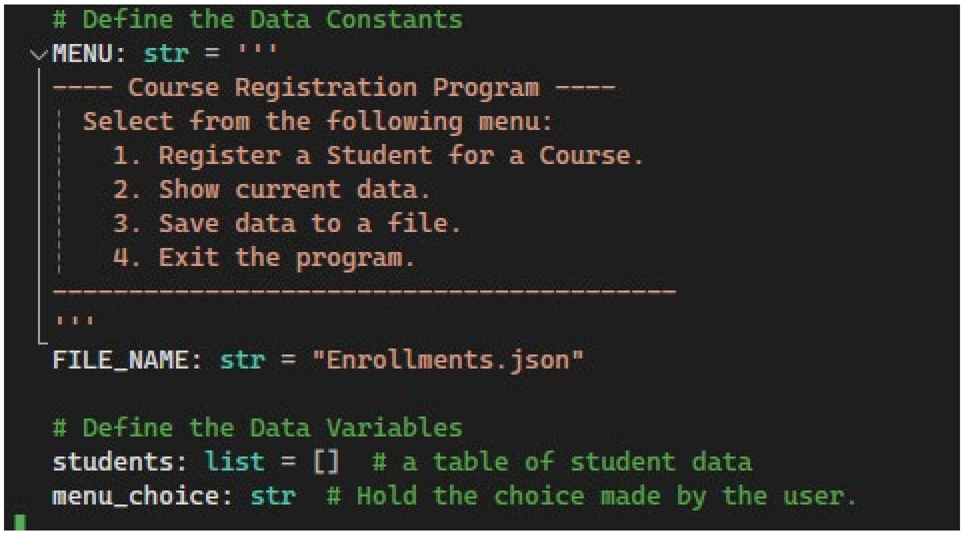
First I opened Visual Studio, then opened “Assignment07-Starter.py,” where I added my information to the change log at the top (Figure 1).



```
# ------------------------------------------------------------------------------------------ #
# Title: Assignment07
# Desc: This assignment demonstrates using data classes
# with structured error handling
# Change Log: (Who, When, What)
#   RRoot,1/1/2030,Created Script
#   RachelHostetler,5/28/24,Assignment07
# ------------------------------------------------------------------------------------------ #
```

Figure.7_Same.header.as.before

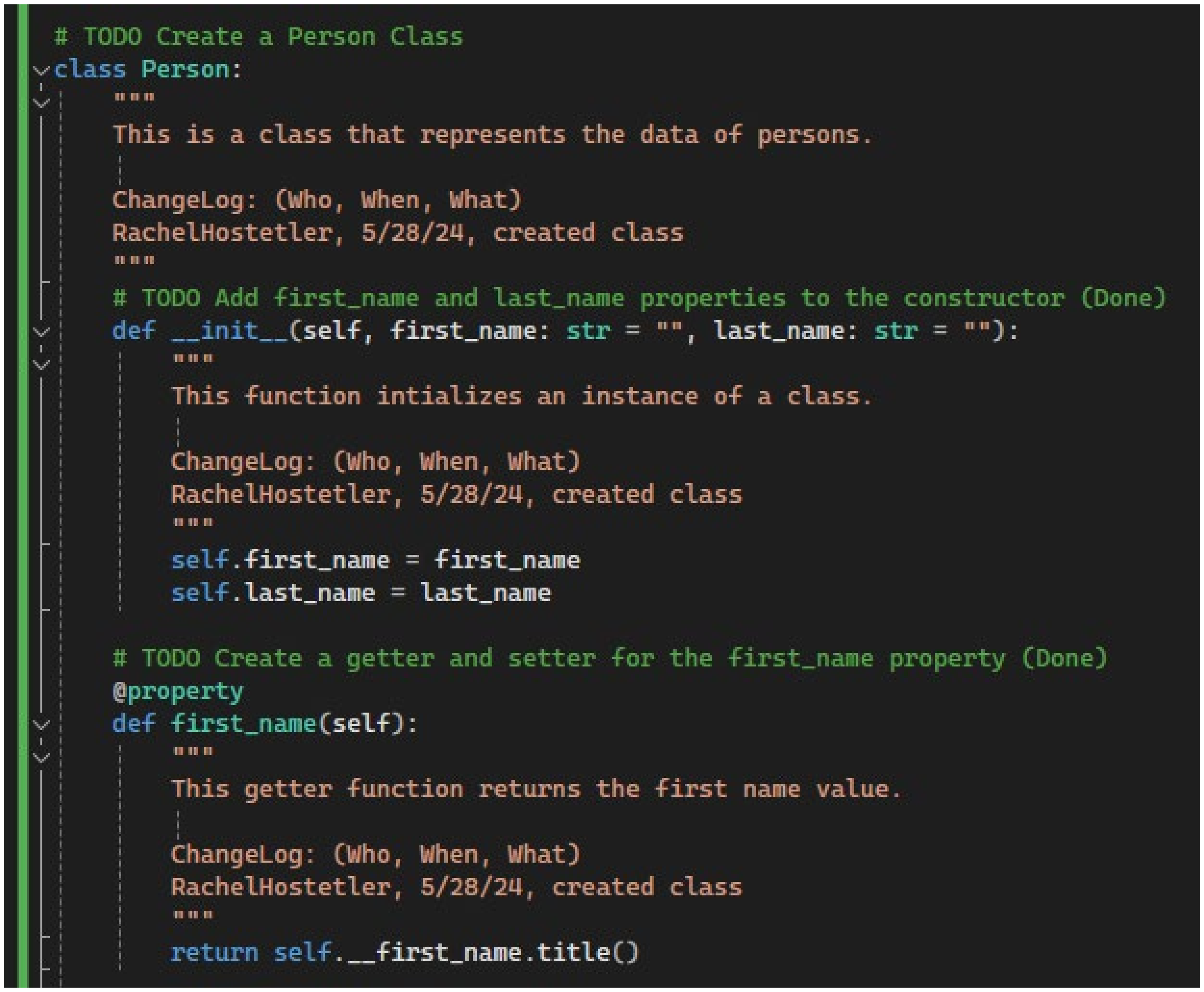
Afterwards I define my data constants and variables based on the assignment and the information from the starter file (Figure 2).



```
# 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.
-----------------------------------------
'''
FILE_NAME: str = "Enrollments.json"

# Define the Data Variables
students: list = []  # a table of student data
menu_choice: str  # Hold the choice made by the user.
```

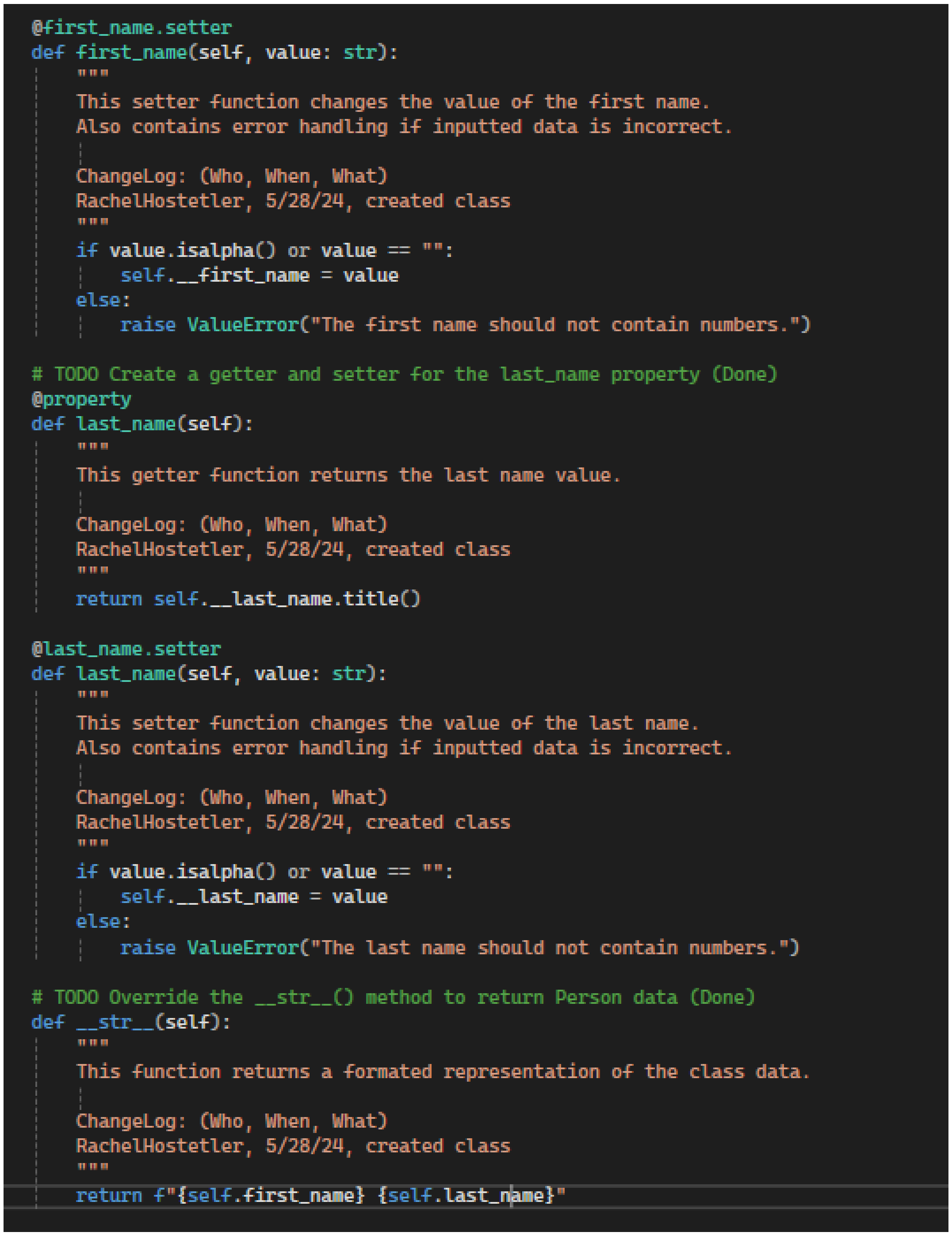
Figure.8_Same.menu.printout.as.before

The next step was to define the class “Person,” which represents the data of persons. This class uses getter and setter functions to save data inputted about a person (Figure 3).



```python
# TODO Create a Person Class
class Person:
    """
    This is a class that represents the data of persons.

    ChangeLog: (Who, When, What)
    RachelHostetler, 5/28/24, created class
    """
    # TODO Add first_name and last_name properties to the constructor (Done)
    def __init__(self, first_name: str = "", last_name: str = ""):
        """
        This function intializes an instance of a class.

        ChangeLog: (Who, When, What)
        RachelHostetler, 5/28/24, created class
        """
        self.first_name = first_name
        self.last_name = last_name

    # TODO Create a getter and setter for the first_name property (Done)
    @property
    def first_name(self):
        """
        This getter function returns the first name value.

        ChangeLog: (Who, When, What)
        RachelHostetler, 5/28/24, created class
        """
        return self.__first_name.title()
```

Figure.9._.Person.Class?its.constructor.and.the.getter.for.first.name



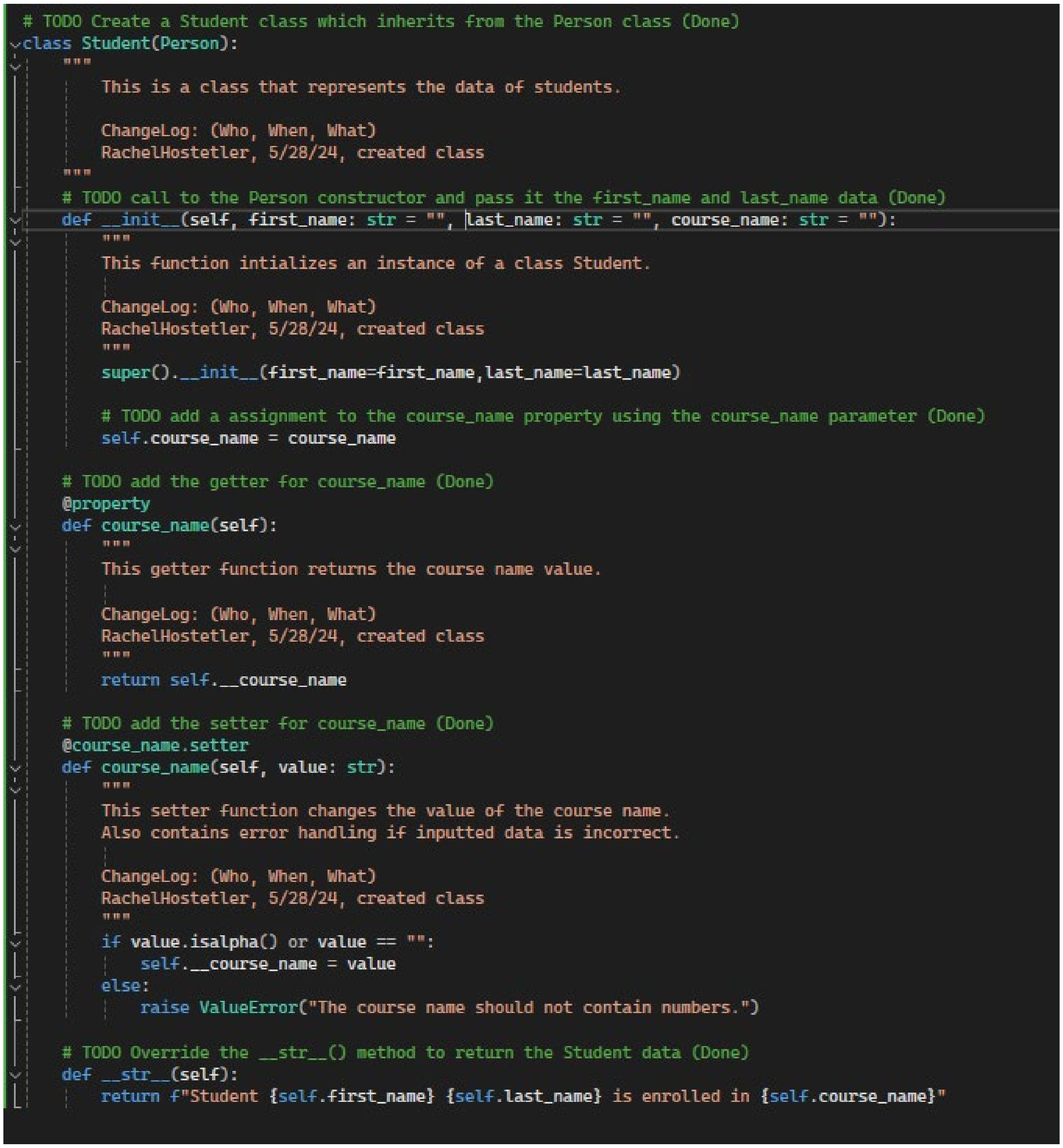
```
@first_name.setter
def first_name(self, value: str):
    """
    This setter function changes the value of the first name.
    Also contains error handling if inputted data is incorrect.

    ChangeLog: (Who, When, What)
    RachelHostetler, 5/28/24, created class
    """
    if value.isalpha() or value == "":
        self.__first_name = value
    else:
        raise ValueError("The first name should not contain numbers.")

# TODO Create a getter and setter for the last_name property (Done)
@property
def last_name(self):
    """
    This getter function returns the last name value.

    ChangeLog: (Who, When, What)
    RachelHostetler, 5/28/24, created class
    """
    return self.__last_name.title()

@last_name.setter
def last_name(self, value: str):
    """
    This setter function changes the value of the last name.
    Also contains error handling if inputted data is incorrect.

    ChangeLog: (Who, When, What)
    RachelHostetler, 5/28/24, created class
    """
    if value.isalpha() or value == "":
        self.__last_name = value
    else:
        raise ValueError("The last name should not contain numbers.")

# TODO Override the __str__() method to return Person data (Done)
def __str__(self):
    """
    This function returns a formated representation of the class data.

    ChangeLog: (Who, When, What)
    RachelHostetler, 5/28/24, created class
    """
    return f"{self.first_name} {self.last_name}"
```

Figure.0_Person.class¡.First.name.setter?Last.Name.getter.and.setter.and.string.method

In sequence, I added a class Student, which inherits from the class Person the first name and last name properties and corresponding getter/setter functions.

The inheritance also changes how to write the constructor (__init__) method, because I had to add the super().__init__ to capture the constructor details for the properties of Person.

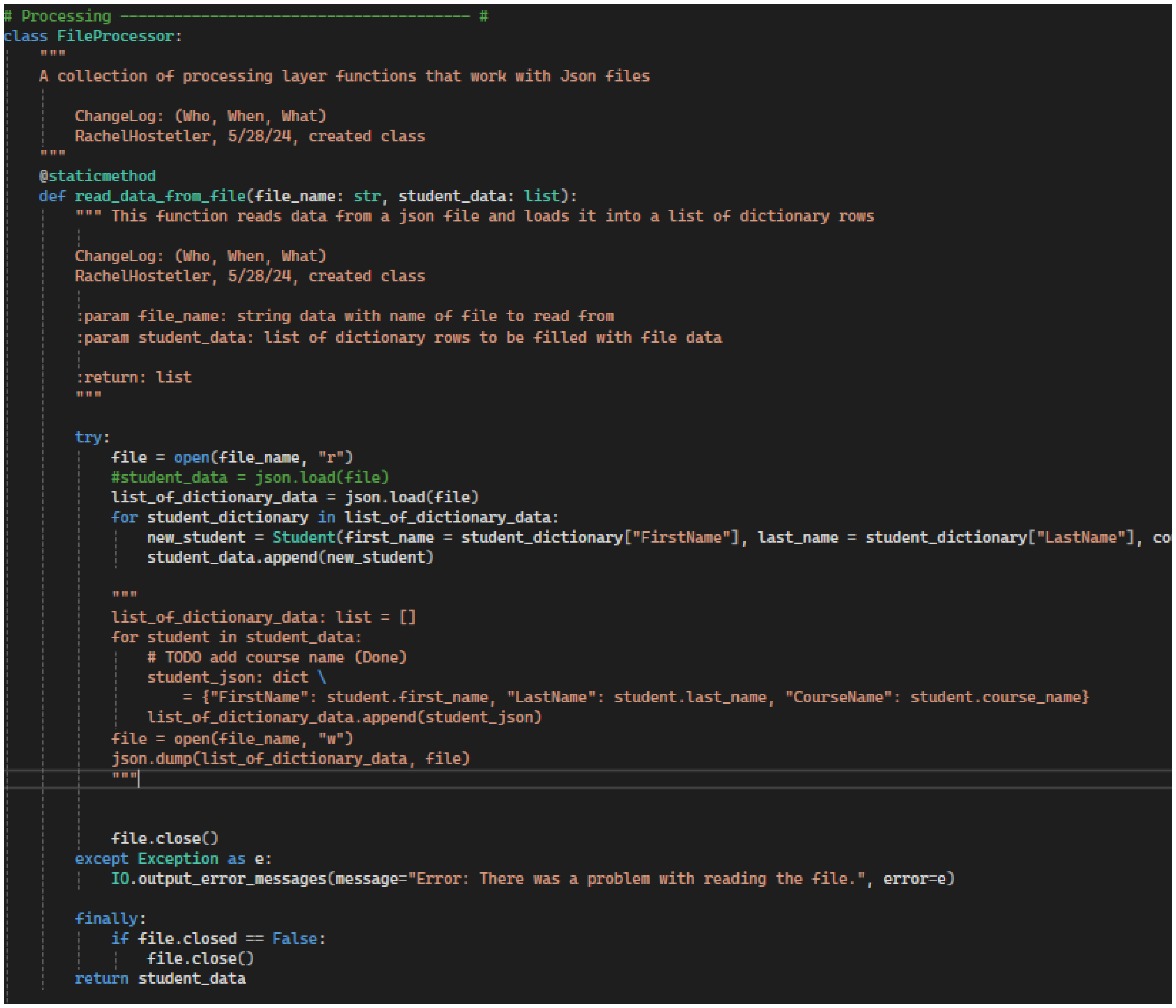
But differently from class where the Student class has GPA, I added a course name property with corresponding getter and setter functions.



```
# TODO Create a Student class which inherits from the Person class (Done)
class Student(Person):
    """
        This is a class that represents the data of students.

        ChangeLog: (Who, When, What)
        RachelHostetler, 5/28/24, created class
    """
    # TODO call to the Person constructor and pass it the first_name and last_name data (Done)
    def __init__(self, first_name: str = "", last_name: str = "", course_name: str = ""):
        """
        This function intializes an instance of a class Student.

        ChangeLog: (Who, When, What)
        RachelHostetler, 5/28/24, created class
        """
        super().__init__(first_name=first_name,last_name=last_name)

        # TODO add a assignment to the course_name property using the course_name parameter (Done)
        self.course_name = course_name

    # TODO add the getter for course_name (Done)
    @property
    def course_name(self):
        """
        This getter function returns the course name value.

        ChangeLog: (Who, When, What)
        RachelHostetler, 5/28/24, created class
        """
        return self.__course_name

    # TODO add the setter for course_name (Done)
    @course_name.setter
    def course_name(self, value: str):
        """
        This setter function changes the value of the course name.
        Also contains error handling if inputted data is incorrect.

        ChangeLog: (Who, When, What)
        RachelHostetler, 5/28/24, created class
        """
        if value.isalpha() or value == "":
            self.__course_name = value
        else:
            raise ValueError("The course name should not contain numbers.")

    # TODO Override the __str__() method to return the Student data (Done)
    def __str__(self):
        return f"Student {self.first_name} {self.last_name} is enrolled in {self.course_name}"
```

Figure.❶_Student.Class.with.constructor?getter.and.setter.and.string.method

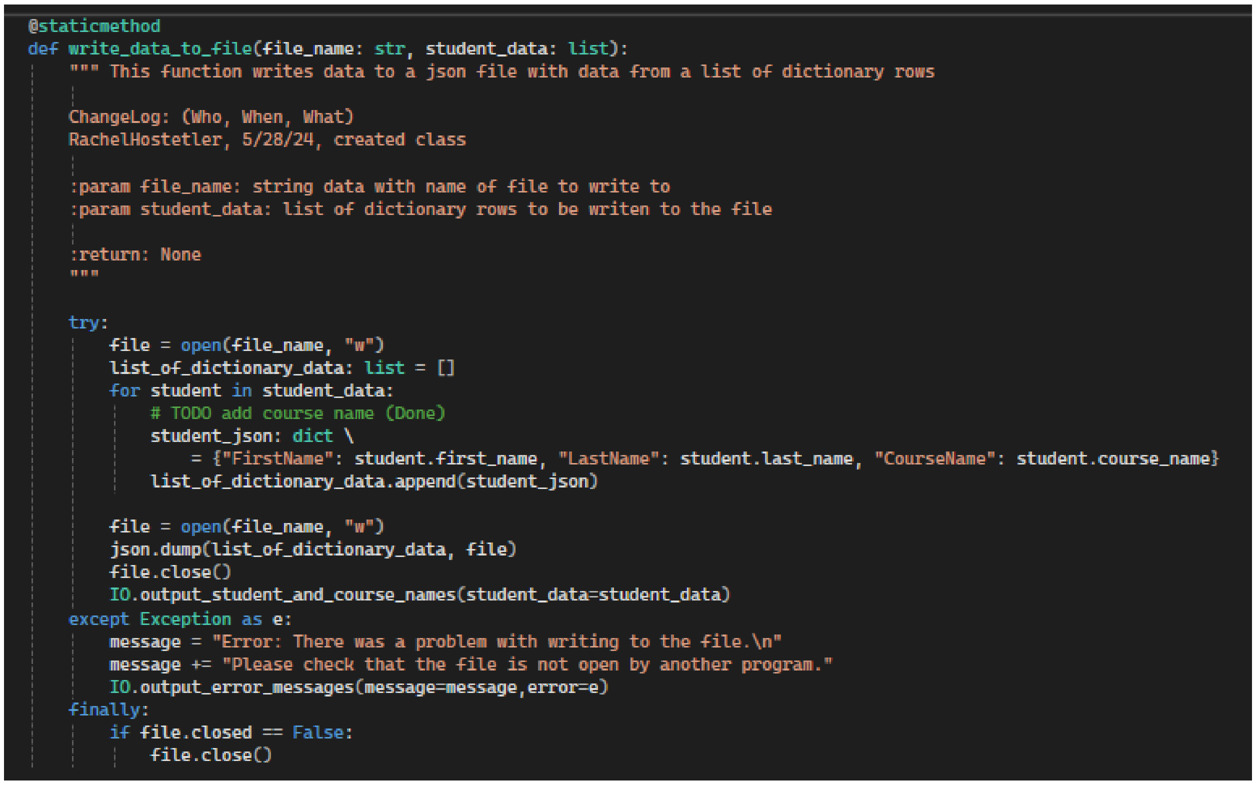
I had to modify the FileProcessor class to read json files and add to student_data list but now using the Student class. The read method reads into a list of dictionaries. Then an object is create for each item on the list and appended to student_data



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

        ChangeLog: (Who, When, What)
        RachelHostetler, 5/28/24, created class
    """
    @staticmethod
    def read_data_from_file(file_name: str, student_data: list):
        """ This function reads data from a json file and loads it into a list of dictionary rows

        ChangeLog: (Who, When, What)
        RachelHostetler, 5/28/24, created class

        :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:
            file = open(file_name, "r")
            #student_data = json.load(file)
            list_of_dictionary_data = json.load(file)
            for student_dictionary in list_of_dictionary_data:
                new_student = Student(first_name = student_dictionary["FirstName"], last_name = student_dictionary["LastName"], co
                student_data.append(new_student)

            """
            list_of_dictionary_data: list = []
            for student in student_data:
                # TODO add course name (Done)
                student_json: dict \
                    = {"FirstName": student.first_name, "LastName": student.last_name, "CourseName": student.course_name}
                list_of_dictionary_data.append(student_json)
            file = open(file_name, "w")
            json.dump(list_of_dictionary_data, file)
            """


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

        finally:
            if file.closed == False:
                file.close()
        return student_data
```

Figure.❷_.Read.method.from.FileProcessor

I changed the write_to_data_file method to save the student_data to a json file. For that I had to create a student_json dictionary for each Student element on the list

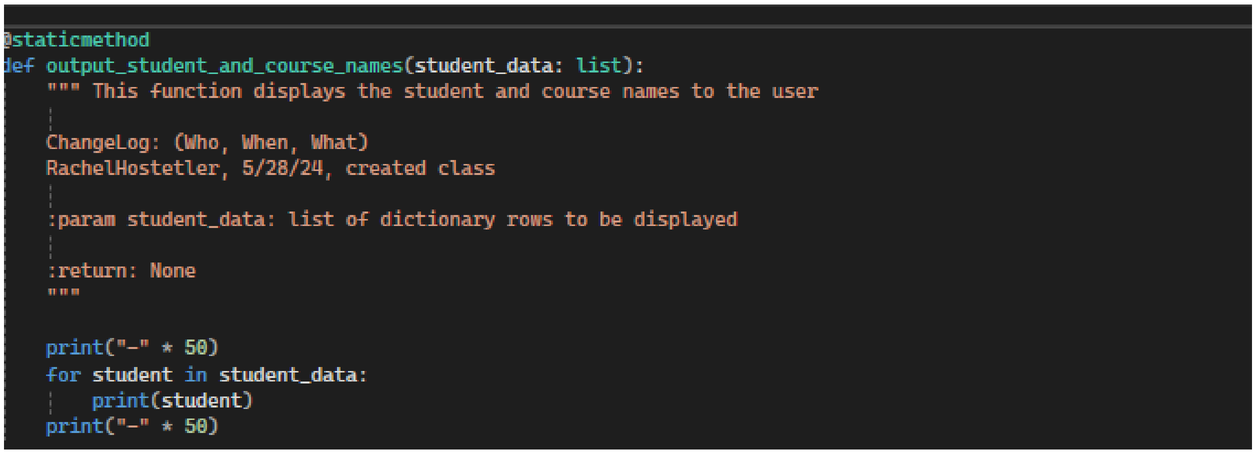


```
@staticmethod
def write_data_to_file(file_name: str, student_data: list):
    """ This function writes data to a json file with data from a list of dictionary rows

    ChangeLog: (Who, When, What)
    RachelHostetler, 5/28/24, created class

    :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:
        file = open(file_name, "w")
        list_of_dictionary_data: list = []
        for student in student_data:
            # TODO add course name (Done)
            student_json: dict \
                = {"FirstName": student.first_name, "LastName": student.last_name, "CourseName": student.course_name}
            list_of_dictionary_data.append(student_json)

        file = open(file_name, "w")
        json.dump(list_of_dictionary_data, file)
        file.close()
        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"
        message += "Please check that the file is not open by another program."
        IO.output_error_messages(message=message,error=e)
    finally:
        if file.closed == False:
            file.close()
```

Figure.❸_Write.data.to.json.file

For the IO class, I only had to change how to print and input data.

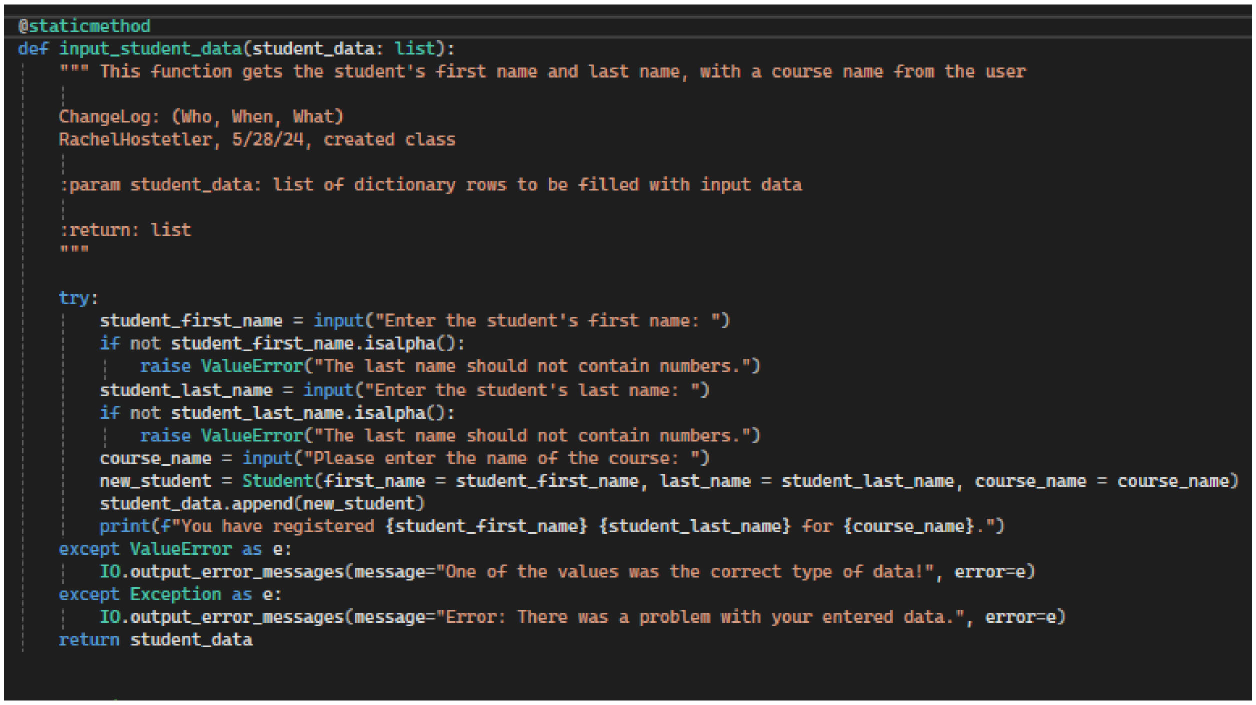
To print, it was easy. I just needed to print each student object since the __str__ method made the formatting for me:



```
@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)
    RachelHostetler, 5/28/24, created class

    :param student_data: list of dictionary rows to be displayed

    :return: None
    """

    print("-" * 50)
    for student in student_data:
        print(student)
    print("-" * 50)
```

Figure.❹_Output.student.data.using.((str((.method.of.the.class

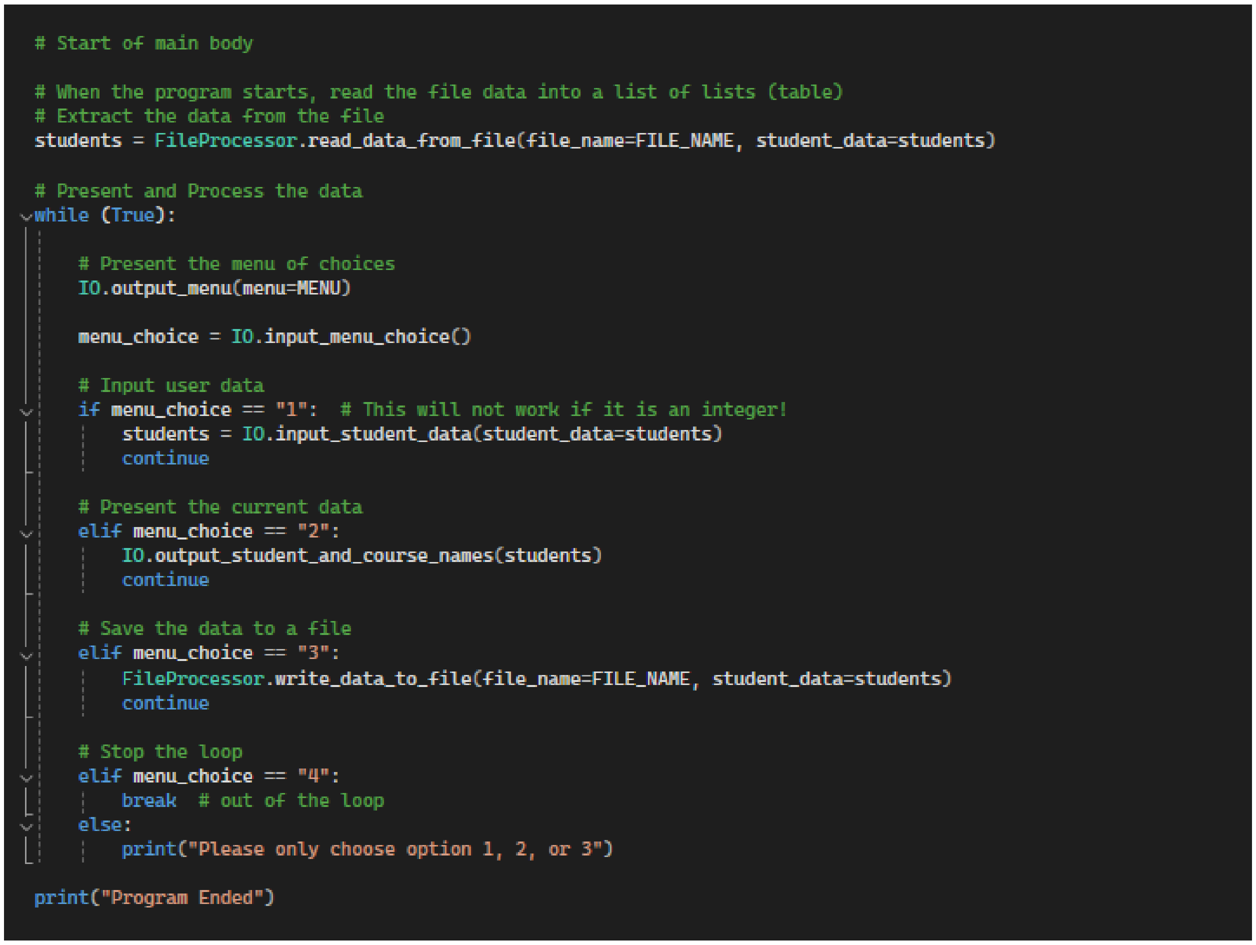
To input the data, I just needed to create an object using the data that was obtained from input() function. Very similar to when I read the json file



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

    ChangeLog: (Who, When, What)
    RachelHostetler, 5/28/24, created class

    :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: ")
        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: ")
        new_student = Student(first_name = student_first_name, last_name = student_last_name, course_name = course_name)
        student_data.append(new_student)
        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 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.❺ Input.student.data

The rest of the code didn’t have to be modified



```
# 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)

# 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":
        IO.output_student_and_course_names(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.76_Options.and.while.loop

Summary

This assignment builds on the previous assignments by adding the class student to store the information of each of the students on the list. I had to adapt how to read and write to json files and understand how __init__ and __str__ methods work.