Name: Lorena Sainz-Maza Lecanda
Date: 11/27/2024
Course: Introduction to Programing with Python
GitHub: https://github.com/lorenasml/IntroToProg-Python-Mod07

Assignment 07 – Classes and Objects

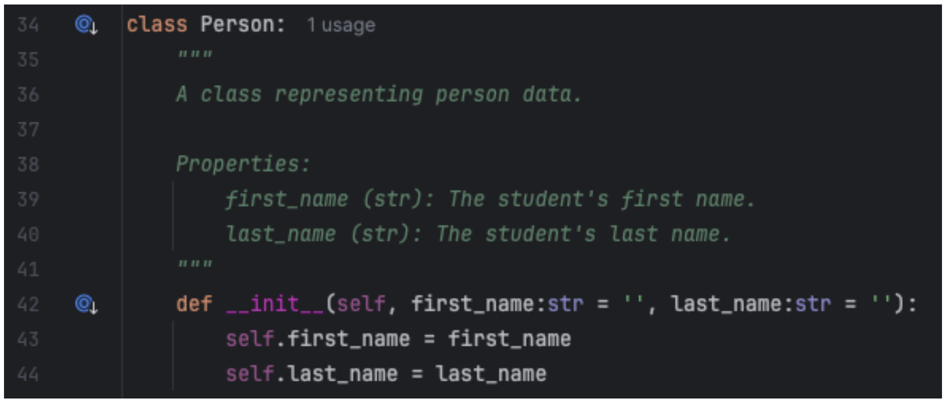
Introduction

In this assignment, I use PyCharm IDE to write a Python script that allows us to register multiple students, process data using dictionaries and save the data in a CSV file. In this program, I introduce the use of data classes and the concept of inheritance. In this essay, I review the steps I took to define my data classes and run the program successfully. It also includes short observations about my performance (i.e. where I got stuck and how I resolved it).

Steps & Observations

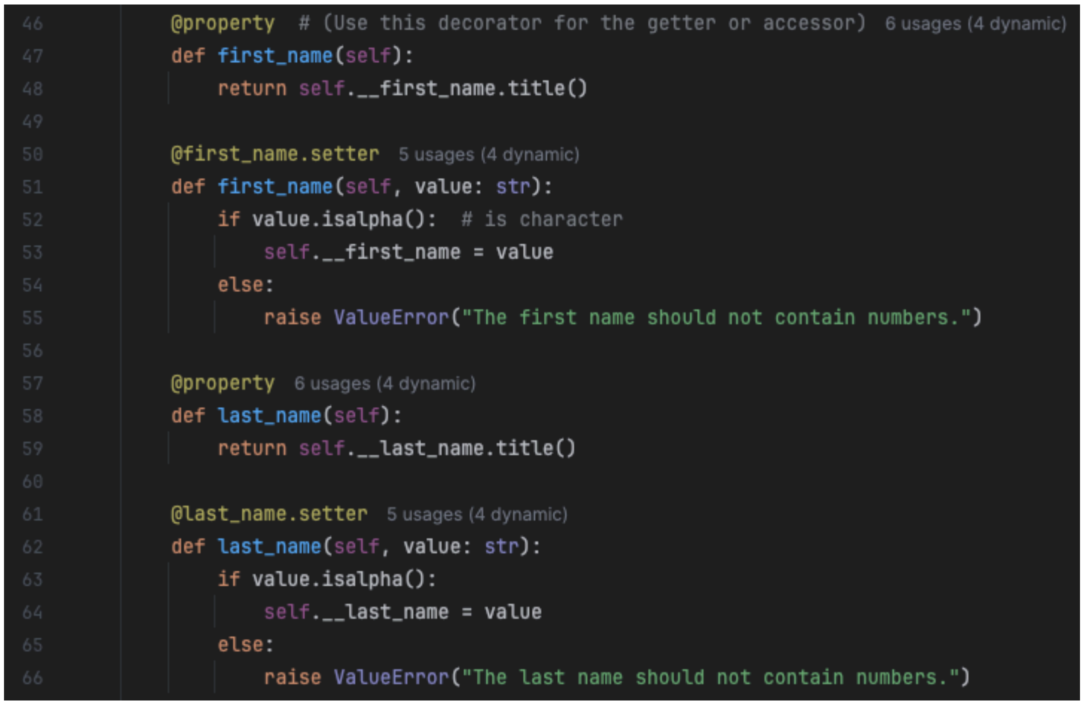
1. Defining functions

First, I started by creating my **Person** `data class`. I initialize my variables (attributes), using `__init__`, in the constructor, without declaring as class-level variables. I used the keyword `self` in the constructor method, which is used to refer to data or functions found in an object instance.



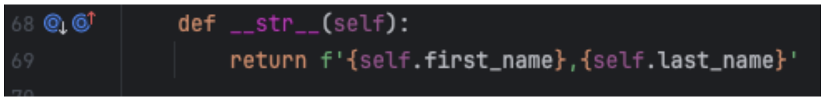
```python
class Person:  1 usage
    """
    A class representing person data.

    Properties:
        first_name (str): The student's first name.
        last_name (str): The student's last name.
    """
    def __init__(self, first_name:str = '', last_name:str = ''):
        self.first_name = first_name
        self.last_name = last_name
```

Once I added my properties to manage data attributes. Specifically, I created 2 properties, one for “getting” the data and one for “setting” the data. To let Python know I need a Getter property function, I use the `@property` decorator. I also added a Setter property function to add data validation and error handling to my code, using the `@name_of_property.setter` decorator.



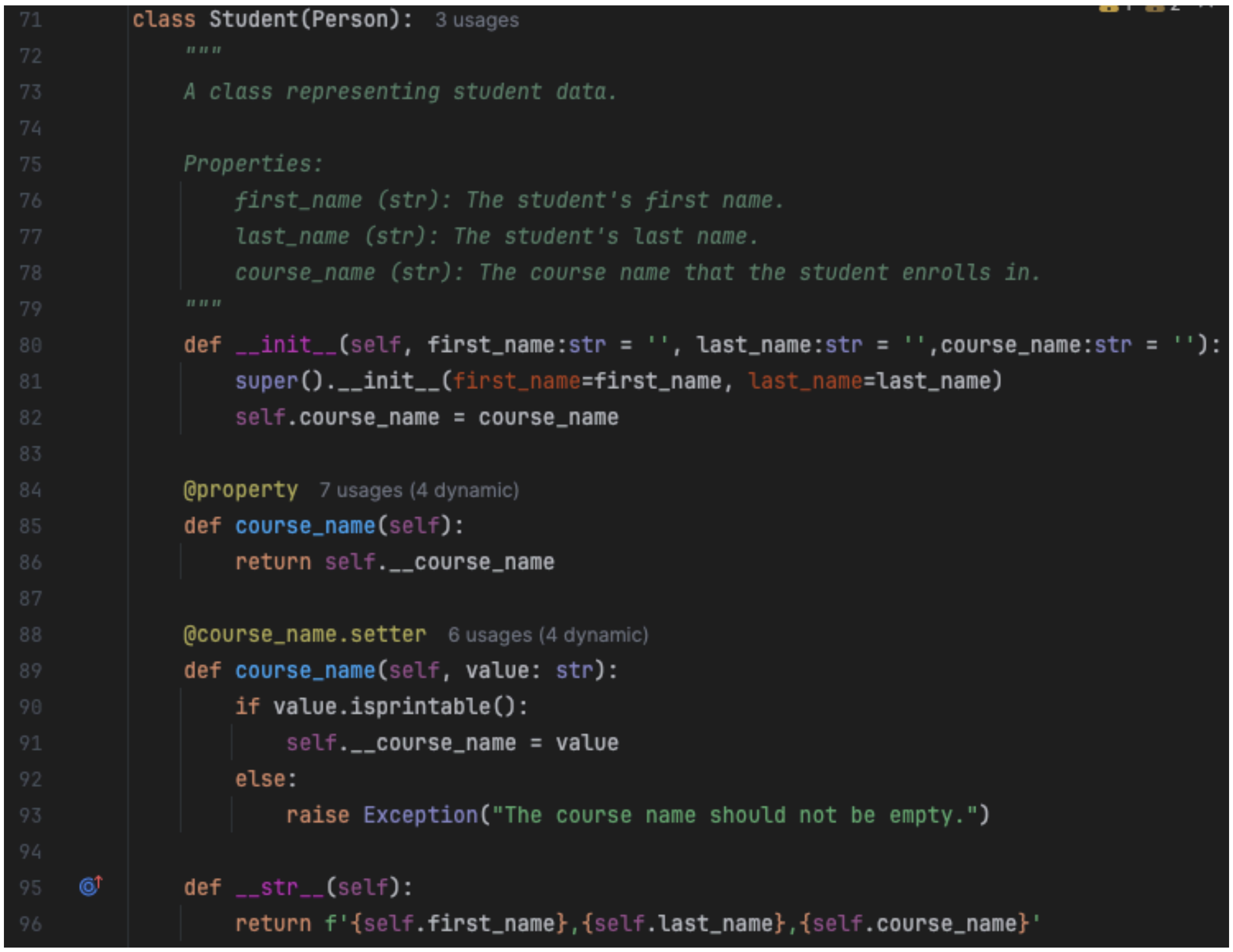
```
    @property  # (Use this decorator for the getter or accessor)  6 usages (4 dynamic)
    def first_name(self):
        return self.__first_name.title()

    @first_name.setter  5 usages (4 dynamic)
    def first_name(self, value: str):
        if value.isalpha():  # is character
            self.__first_name = value
        else:
            raise ValueError("The first name should not contain numbers.")

    @property  6 usages (4 dynamic)
    def last_name(self):
        return self.__last_name.title()

    @last_name.setter  5 usages (4 dynamic)
    def last_name(self, value: str):
        if value.isalpha():
            self.__last_name = value
        else:
            raise ValueError("The last name should not contain numbers.")
```

In this code, I also implemented the "magic" method, `__str__()` method, in order to return a customized human-readable string representation of the object.



```
    def __str__(self):
        return f'{self.first_name},{self.last_name}'
```

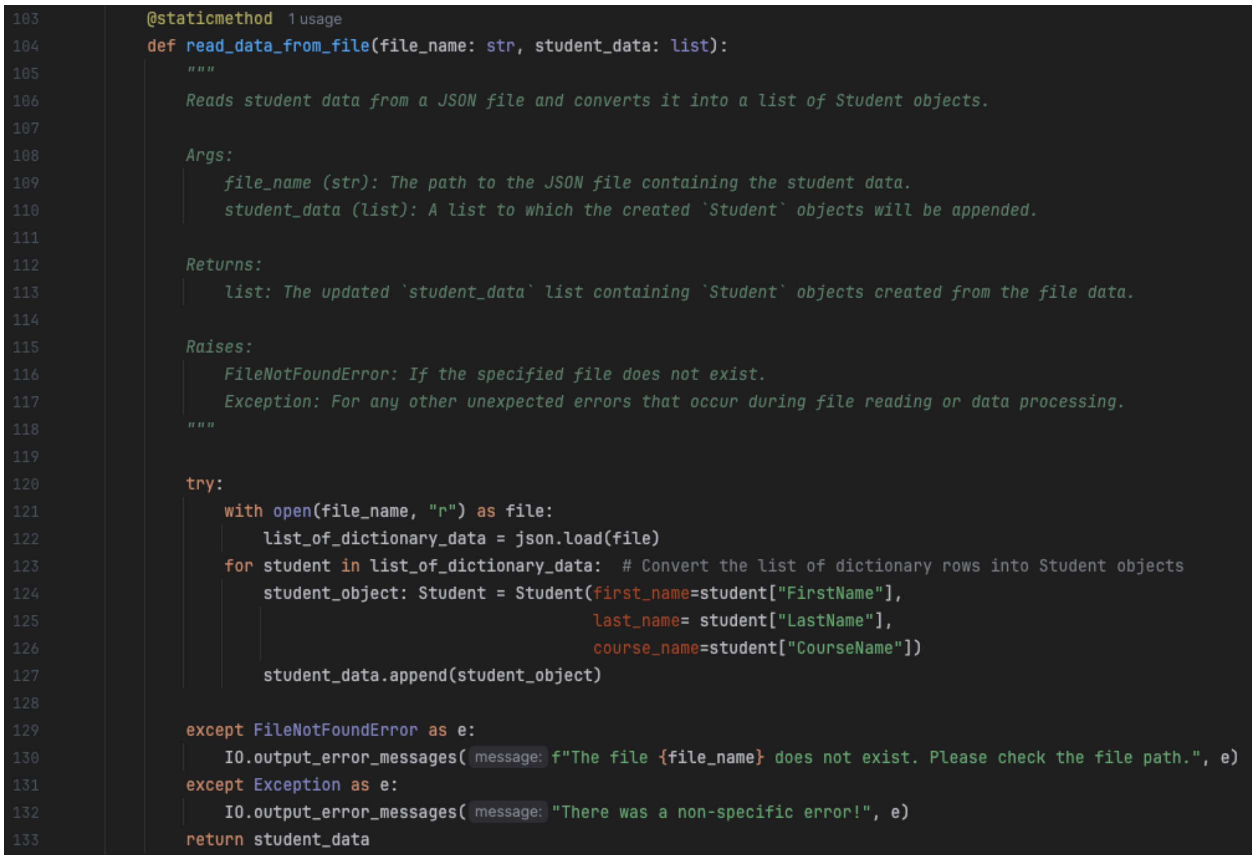
After creating my Person data class, I also created **Student** `data class` inherits code from its parent class or superclass **Student**, resulting in the form `class` **`Student(Person)`**. For this piece of code, I also I initialize my variables using `__init__`, in the constructor, but I add `super()` to let Python know that I'm inheriting `first_name` and `last_name` from the superclass Student. Unlike Student, the Person data class introduces a new attribute, `course_name`, in the constructor. Here, too, I use the Getter and Setter property functions introduced by their respective decorators.



```python
class Student(Person):  3 usages
    """
    A class representing student data.

    Properties:
        first_name (str): The student's first name.
        last_name (str): The student's last name.
        course_name (str): The course name that the student enrolls in.
    """
    def __init__(self, first_name:str = '', last_name:str = '',course_name:str = ''):
        super().__init__(first_name=first_name, last_name=last_name)
        self.course_name = course_name

    @property  7 usages (4 dynamic)
    def course_name(self):
        return self.__course_name

    @course_name.setter  6 usages (4 dynamic)
    def course_name(self, value: str):
        if value.isprintable():
            self.__course_name = value
        else:
            raise Exception("The course name should not be empty.")

    def __str__(self):
        return f'{self.first_name},{self.last_name},{self.course_name}'
```

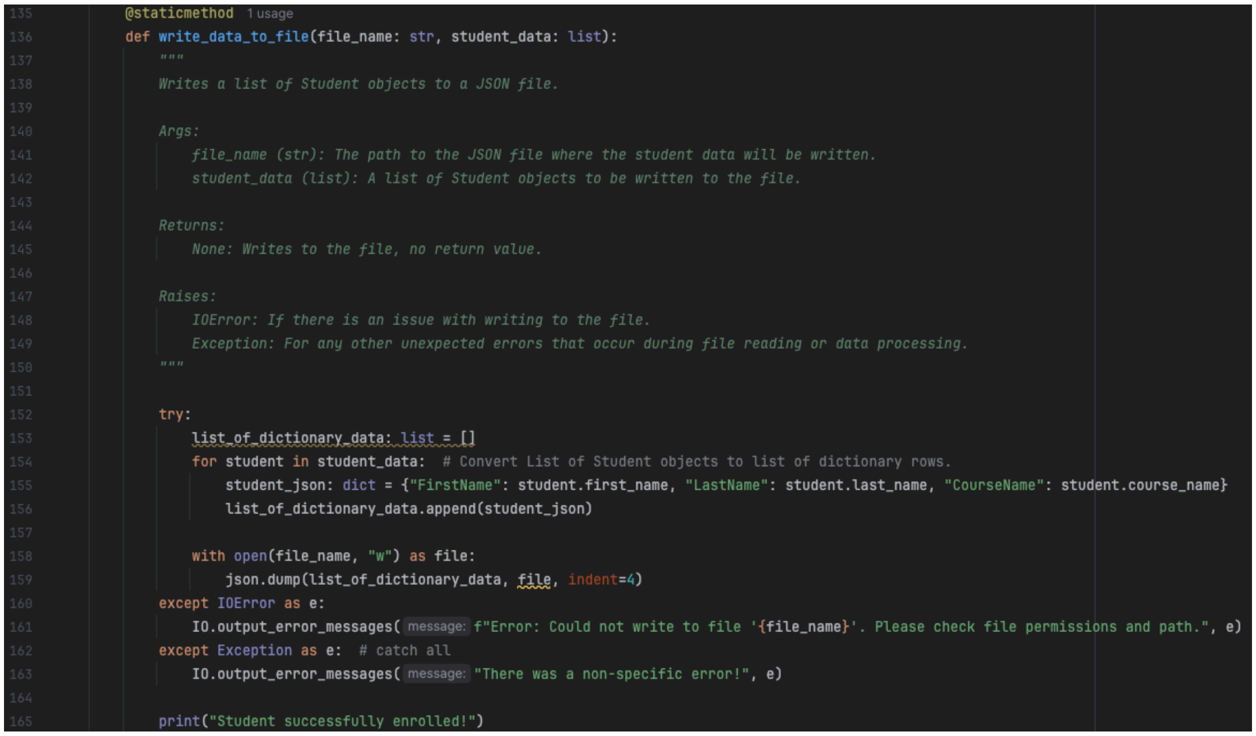
Since I'm dealing with Student objects, I had to make modifications to my `read_data_from_file` and `write_data_to_file` functions inside the **FileProcessor** class.

- **`read_data_from_file:`** This function reads student data from a JSON file and converts it into a list of `Student` objects. To do that, I open the file in "read" mode using the `with` statement (it will automatically close the file). I use a `for` loop to iterate over the list of dictionary rows and convert them into Student objects.



```python
    @staticmethod  1 usage
    def read_data_from_file(file_name: str, student_data: list):
        """
        Reads student data from a JSON file and converts it into a list of Student objects.

        Args:
            file_name (str): The path to the JSON file containing the student data.
            student_data (list): A list to which the created `Student` objects will be appended.

        Returns:
            list: The updated `student_data` list containing `Student` objects created from the file data.

        Raises:
            FileNotFoundError: If the specified file does not exist.
            Exception: For any other unexpected errors that occur during file reading or data processing.
        """

        try:
            with open(file_name, "r") as file:
                list_of_dictionary_data = json.load(file)
            for student in list_of_dictionary_data:  # Convert the list of dictionary rows into Student objects
                student_object: Student = Student(first_name=student["FirstName"],
                                                  last_name= student["LastName"],
                                                  course_name=student["CourseName"])
                student_data.append(student_object)

        except FileNotFoundError as e:
            IO.output_error_messages( message: f"The file {file_name} does not exist. Please check the file path.", e)
        except Exception as e:
            IO.output_error_messages( message: "There was a non-specific error!", e)
        return student_data
```

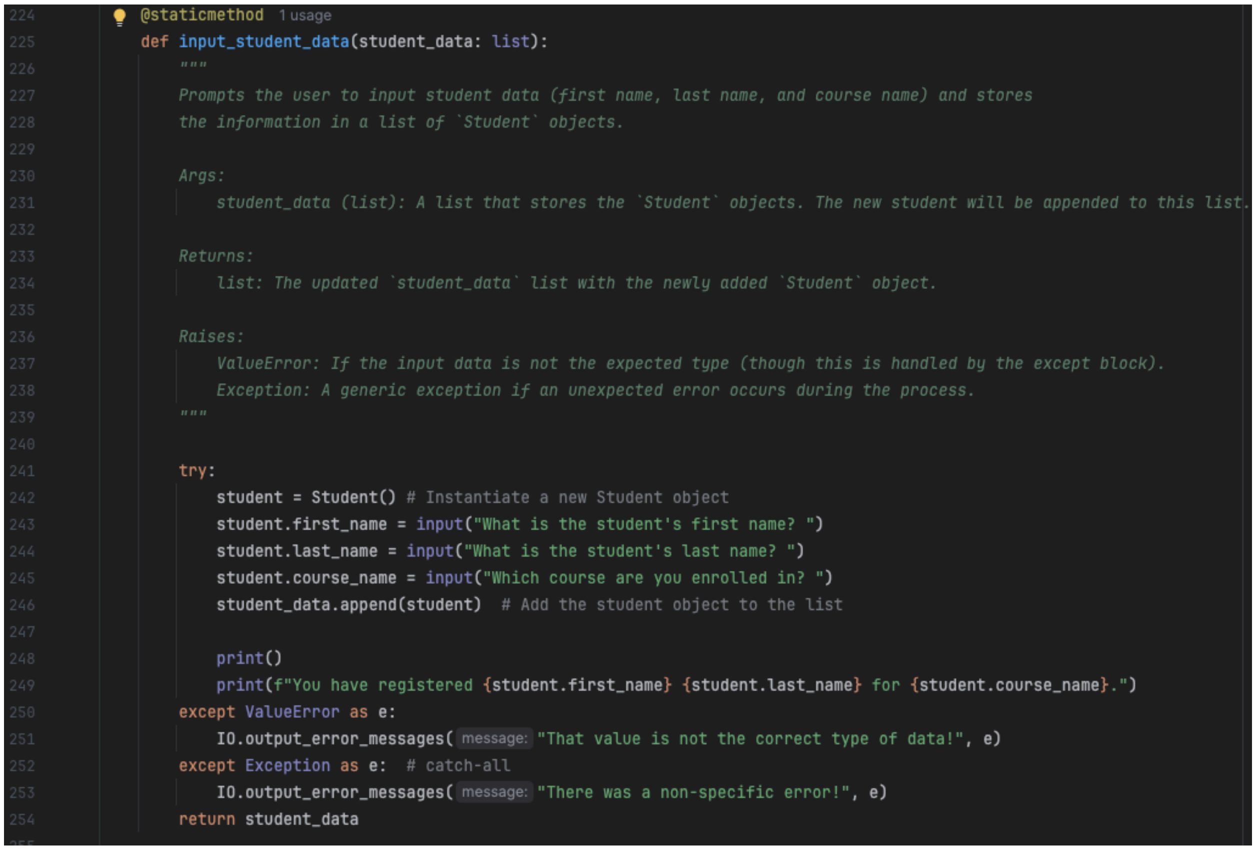
- **`write_data_to_file`**: This function writes a list of Student objects to the JSON file. First, I create an empty list to hold the student data in dictionary form under `list_of_dictionary_data`. The code loops over each **Student** object in `student_data`, and for each **Student** object, it creates a dictionary `student_json` containing first name, last name and course name information. Lastly, I open the file in “write” mode and, using `json.dump`, the code takes the `list_of_dictionary_data` (which contains the student data in dictionary form) and writes it to the file.



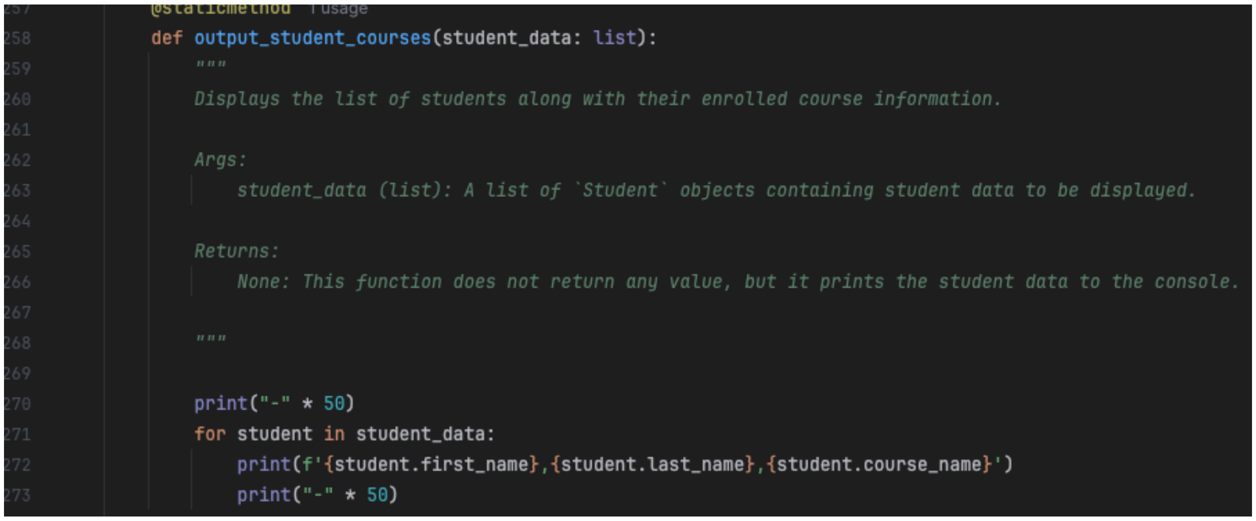
```python
    @staticmethod  1 usage
    def write_data_to_file(file_name: str, student_data: list):
        """
        Writes a list of Student objects to a JSON file.

        Args:
            file_name (str): The path to the JSON file where the student data will be written.
            student_data (list): A list of Student objects to be written to the file.

        Returns:
            None: Writes to the file, no return value.

        Raises:
            IOError: If there is an issue with writing to the file.
            Exception: For any other unexpected errors that occur during file reading or data processing.
        """

        try:
            list_of_dictionary_data: list = []
            for student in student_data:  # Convert List of Student objects to list of dictionary rows.
                student_json: dict = {"FirstName": student.first_name, "LastName": student.last_name, "CourseName": student.course_name}
                list_of_dictionary_data.append(student_json)

            with open(file_name, "w") as file:
                json.dump(list_of_dictionary_data, file, indent=4)
        except IOError as e:
            IO.output_error_messages( message: f"Error: Could not write to file '{file_name}'. Please check file permissions and path.", e)
        except Exception as e:  # catch all
            IO.output_error_messages( message: "There was a non-specific error!", e)

        print("Student successfully enrolled!")
```

Before running the script, I had to update the `input_student_data` and `output_student_courses` fuctions as well.

- `input_student_data`: The code had to instantiate a **Student** object. It initialized the **Student** object and gets user input to enter the first name, last name and course name. Using `append()`, I add the new **Student** object to `student_data` lists that stores all the student objects.
- `Output_student_courses`: Here, I had to update the format of the `f-string` in order to print the `first_name, last_name`, and `course_name` attributes of each **Student** object, separated by commas.



```python
    @staticmethod  1 usage
    def input_student_data(student_data: list):
        """
        Prompts the user to input student data (first name, last name, and course name) and stores
        the information in a list of `Student` objects.

        Args:
            student_data (list): A list that stores the `Student` objects. The new student will be appended to this list.

        Returns:
            list: The updated `student_data` list with the newly added `Student` object.

        Raises:
            ValueError: If the input data is not the expected type (though this is handled by the except block).
            Exception: A generic exception if an unexpected error occurs during the process.
        """

        try:
            student = Student() # Instantiate a new Student object
            student.first_name = input("What is the student's first name? ")
            student.last_name = input("What is the student's last name? ")
            student.course_name = input("Which course are you enrolled in? ")
            student_data.append(student)  # Add the student object to the list

            print()
            print(f"You have registered {student.first_name} {student.last_name} for {student.course_name}.")
        except ValueError as e:
            IO.output_error_messages( message: "That value is not the correct type of data!", e)
        except Exception as e:  # catch-all
            IO.output_error_messages( message: "There was a non-specific error!", e)
        return student_data
```



```python
    @staticmethod  1 usage
    def output_student_courses(student_data: list):
        """
        Displays the list of students along with their enrolled course information.

        Args:
            student_data (list): A list of `Student` objects containing student data to be displayed.

        Returns:
            None: This function does not return any value, but it prints the student data to the console.

        """

        print("-" * 50)
        for student in student_data:
            print(f'{student.first_name},{student.last_name},{student.course_name}')
            print("-" * 50)
```

2. The program script

Note: I've uploaded a video of the program running successfully in PyCharm and Terminal.

The program script remains identical to Assignment 06. The program starts by reading student data from "Enrollments.json" and loading it into the `students` list. The `read_data_from_file` function reads student data from the file and returns a list of **Student** objects, which is assigned to the `students` variable. After that, I introduce the `while` loop, which ensures that the program keeps running until the user chooses option 4 (Exit). The `IO.output_menu(menu=MENU)` function displays the menu, and `IO.input_menu_choice()` prompts the user for their selection. The program iterates through 4 different menu options:

- **Option 1**: Adds a new student to the students list and uses the `input_student_data` function.
- **Option 2**: Displays the current student courses and uses the `output_student_courses` function.
- **Option 3**: Saves the student data back to a file with the file processing function `write_data_to_file`.
- **Option 4**: Exits the loop and terminates the program.

Summary & Reflection

To summarize, Task 7 took me approximately 5 hours. I challenged myself to use my previous assignment code rather than the starter. Mod07-Lab3 was extremely helpful to figure out how to write the new data classes with inheritance. The most challenging part for me was understanding how to convert the student data to a list of student objects in the `read_data_from_file` function. To better understand this part, I ran snippets of Mod07-Lab3 through chatGPT and asked it to explain the code to me. I also used chatGPT to help me generate the docstrings.