Saima Ahmed

November 26th, 2023

Foundation of Programming: Python

Assignment 07

https://github.com/Saima-87/IntroToProg-Python-Mod07

Creating a Python Script

Introduction:

In this module, we delve into the core concepts of programming and version control. We begin by clarifying the distinctions between statements, functions, and classes. Next, we explore the world of classes in more detail, dissecting data classes, presentation classes, and processing classes, and understanding their unique purposes. We also learn about constructors, attributes, and properties. We also study the concept of class inheritance and the power it brings to code reuse and extensibility, including the concept of overridden methods.

Function:

In programming, a function is a reusable block of code that performs a specific task or a set of tasks. Functions are a fundamental concept in most programming languages and serve several important purposes.

Parameters:

Parameters allow you to make your functions more flexible and reusable because you can customize their behavior by providing different values when you call them. The parameters act as placeholders for the values that you want to pass to the function when you call it. The values you pass into parameters are called arguments.

Classes and Functions:

Another way to organize your code is by using classes. Classes are a way of grouping functions, variables, and constants by the name of the class. Using classes to group functions is a fundamental concept in programming. Grouping functions within classes creates a modular structure, making it easier to manage and maintain code. Classes provide a natural way to organize code by grouping functions and data needed by those functions, making code more structured and readable, especially in larger projects.

The Separation of Concerns (SoC):

The Separation of Concerns (SoC) is a fundamental software design principle that aims to enhance the maintainability, scalability, and readability of code by breaking it down into distinct, self-contained components, each responsible for a specific aspect of the application's functionality. This pattern encourages modularity, reduces code complexity, and makes it easier to manage and extend software systems.

Attributes:

In programming, an attribute is a piece of data, or a characteristic associated with an object. Attributes describe or store information about the object they belong to. Depending on the programming context, attributes are also referred by developers as fields.

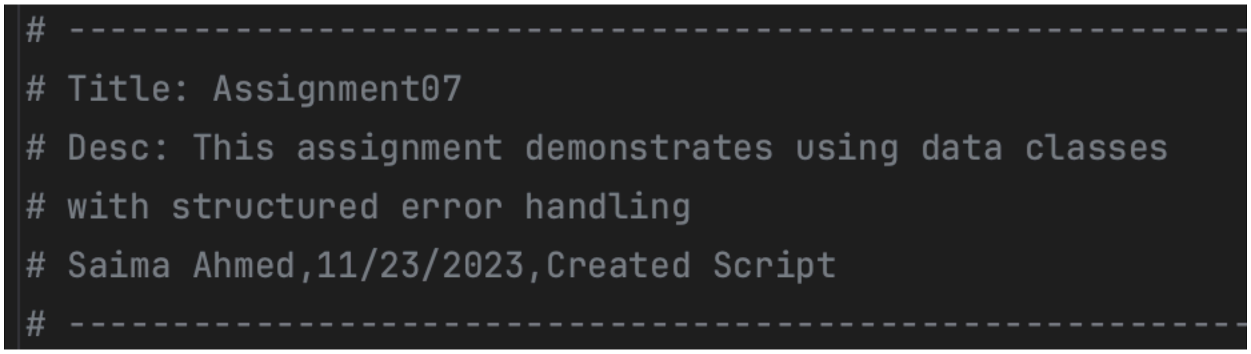
Constructor:

A constructor is a special method (function) that is automatically called when an object of a class is created. Its primary purpose is to set an object's attributes values when it is created. Constructors are sometime called an initializer because it sets up any necessary initial data (known as state) for the object.

Creating the Program:

The objective of this week was to create a Python program that demonstrates using constants, variables, and print statements to display a message about a student's registration for a Python course with **the use of functions, classes, and using the separation of concerns pattern.**

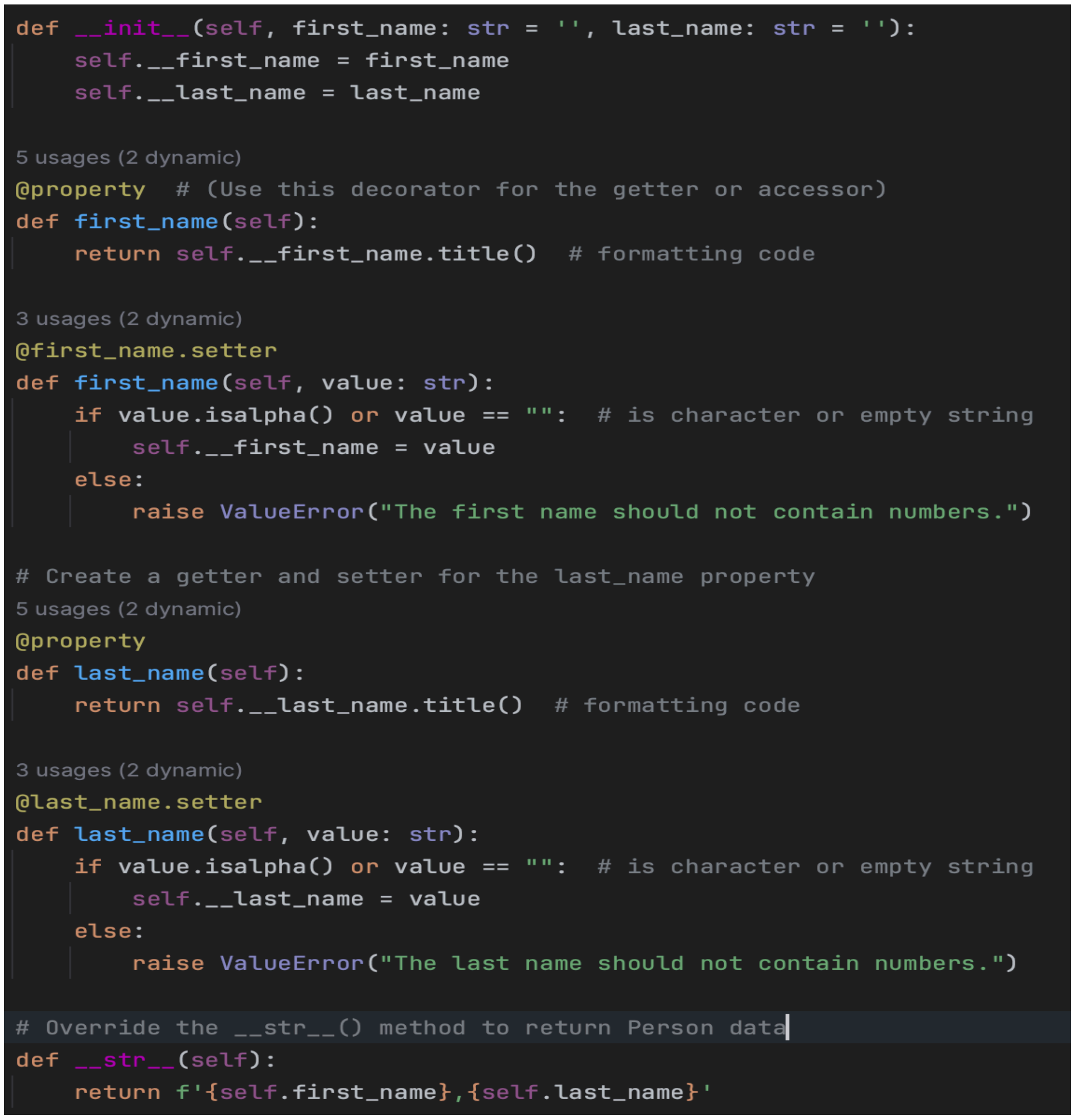
- I started the program by first including the **Script Header** that shows Title and description about the program along with my name and current date as shown in Fig 1



```
# -----------------------------------------------------
# Title: Assignment07
# Desc: This assignment demonstrates using data classes
# with structured error handling
# Saima Ahmed,11/23/2023,Created Script
# -----------------------------------------------------
```

Fig 1: Creating Script Header

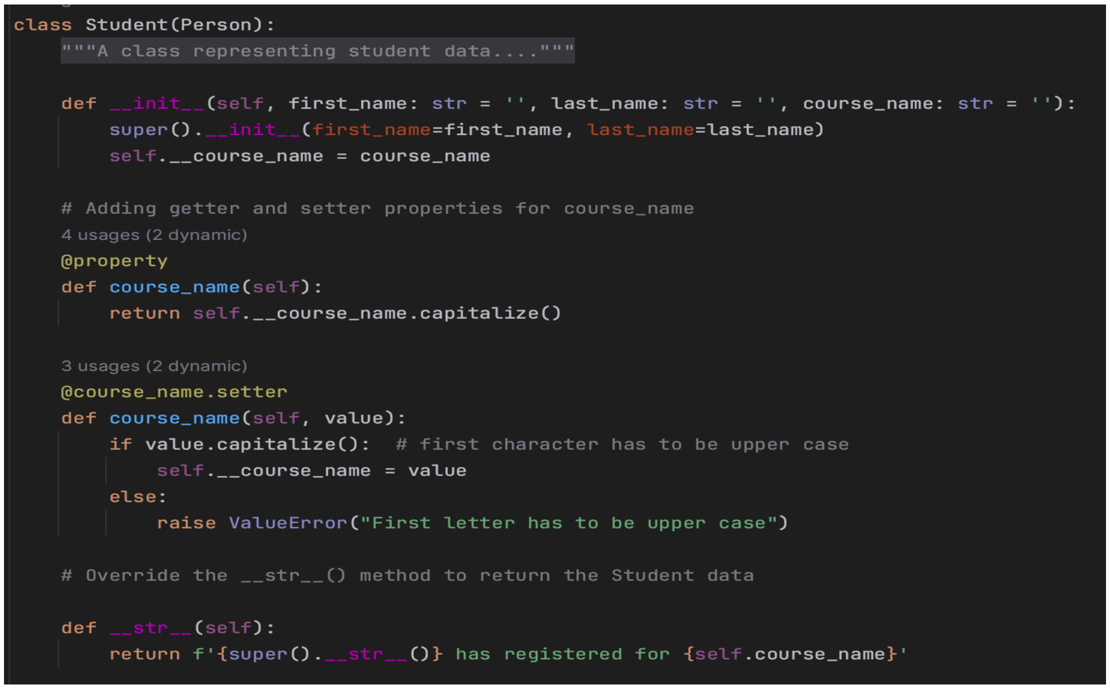
- Using Separation of concerns (SoC) pattern, I organized my program into three main classes called 'Person', 'FileProcessor' and 'IO'.
- The class 'Person' has a sub-class called 'Student'. Both these classes belong to Data classes whose main objective is to organize data about the students. While processing and presentation classes just have methods, Data classes typically have Attributes, Constructor, Properties in addition to Methods.
- The class File Processor has mainly all the functions that handles the main operation of the code. In this code, 'read_data_from_file' and 'write_data_to_file' were included in the class FileProcessor.
- Class IO has mainly all the functions that handles taking inputs from the user and displaying outputs to the user. In this code I used the functions output_error_message, out_menu, input_menu_choice, output_student_courses and input_student_data.
- The class 'Student' has an __init__ function that handles the attributes first_name and last_name of the student. I have also used getter and setter properties on these attributes and over ride the __str()__ method to return Person data as shown in the Fig 2



```python
def __init__(self, first_name: str = '', last_name: str = ''):
    self.__first_name = first_name
    self.__last_name = last_name

5 usages (2 dynamic)
@property  # (Use this decorator for the getter or accessor)
def first_name(self):
    return self.__first_name.title()  # formatting code

3 usages (2 dynamic)
@first_name.setter
def first_name(self, value: str):
    if value.isalpha() or value == "":  # is character or empty string
        self.__first_name = value
    else:
        raise ValueError("The first name should not contain numbers.")

# Create a getter and setter for the last_name property
5 usages (2 dynamic)
@property
def last_name(self):
    return self.__last_name.title()  # formatting code

3 usages (2 dynamic)
@last_name.setter
def last_name(self, value: str):
    if value.isalpha() or value == "":  # is character or empty string
        self.__last_name = value
    else:
        raise ValueError("The last name should not contain numbers.")

# Override the __str__() method to return Person data
def __str__(self):
    return f'{self.first_name},{self.last_name}'
```

Fig 2: Main structure of class Person

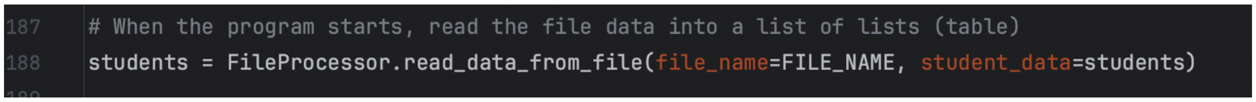
- Person class has a sub class called ‘Student’ which manages an extra attribute called ‘course_name”. I have again used getter and setter properties on course_name and override the __str()__ method to return student data as shown in Fig 3



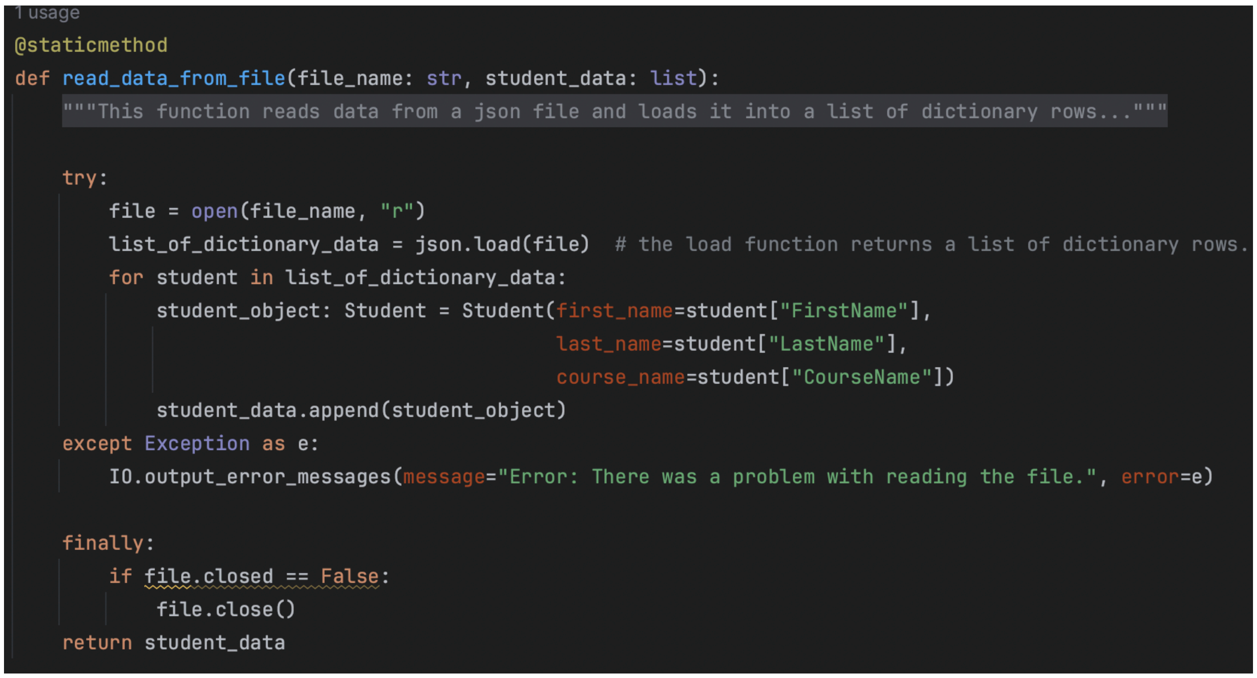
```python
class Student(Person):
    """A class representing student data...."""

    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

    # Adding getter and setter properties for course_name
    4 usages (2 dynamic)
    @property
    def course_name(self):
        return self.__course_name.capitalize()

    3 usages (2 dynamic)
    @course_name.setter
    def course_name(self, value):
        if value.capitalize():  # first character has to be upper case
            self.__course_name = value
        else:
            raise ValueError("First letter has to be upper case")

    # Override the __str__() method to return the Student data

    def __str__(self):
        return f'{super().__str__()} has registered for {self.course_name}'
```

Fig 3: Main structure of student class

- The program starts by calling the function read_data_from_file to read data form the json file Enrollments.json as shown in Fig3



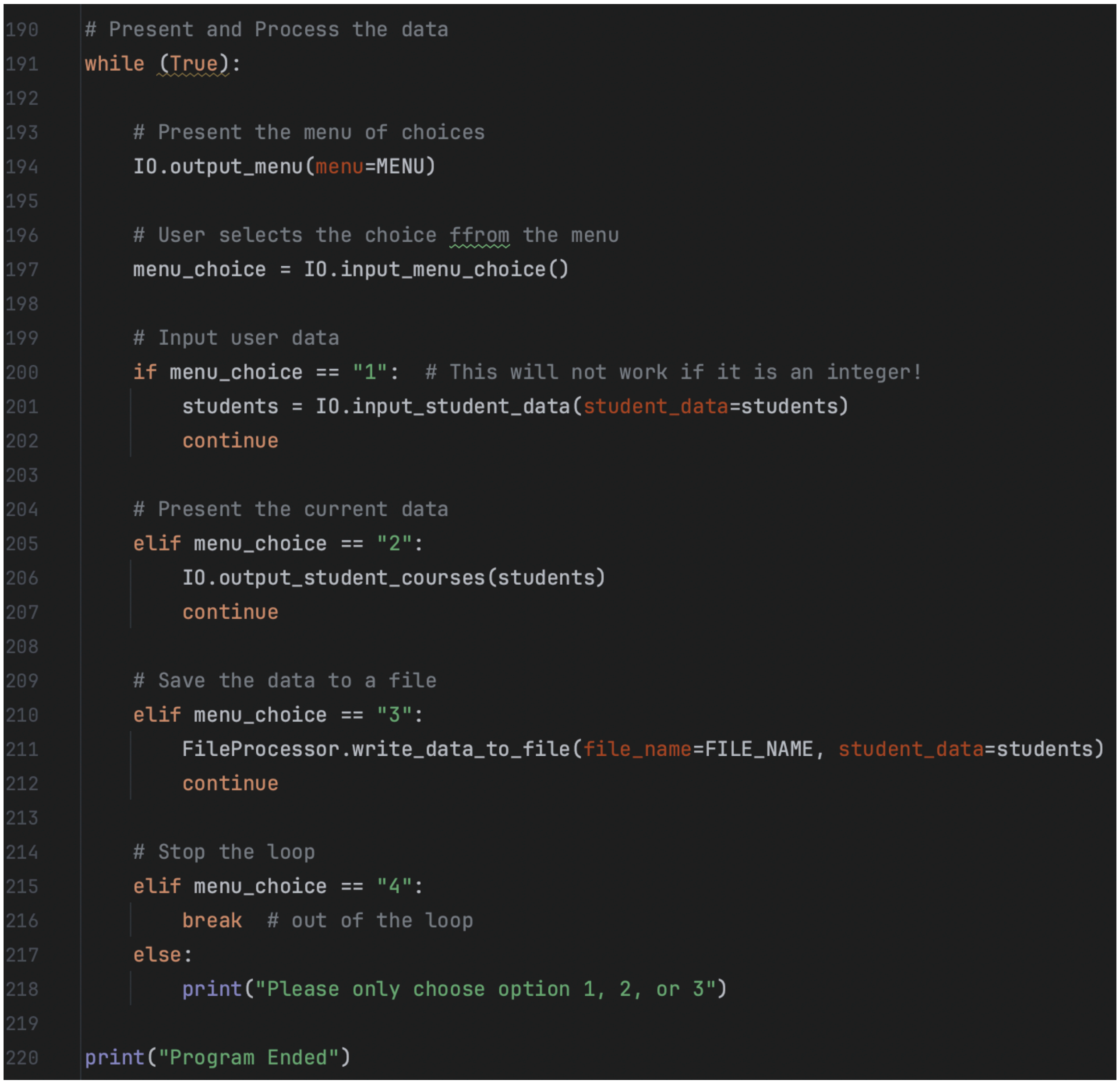
```python
# When the program starts, read the file data into a list of lists (table)
students = FileProcessor.read_data_from_file(file_name=FILE_NAME, student_data=students)
```



```
1 usage
@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..."""

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

Fig 3: Opening the json file in read mode

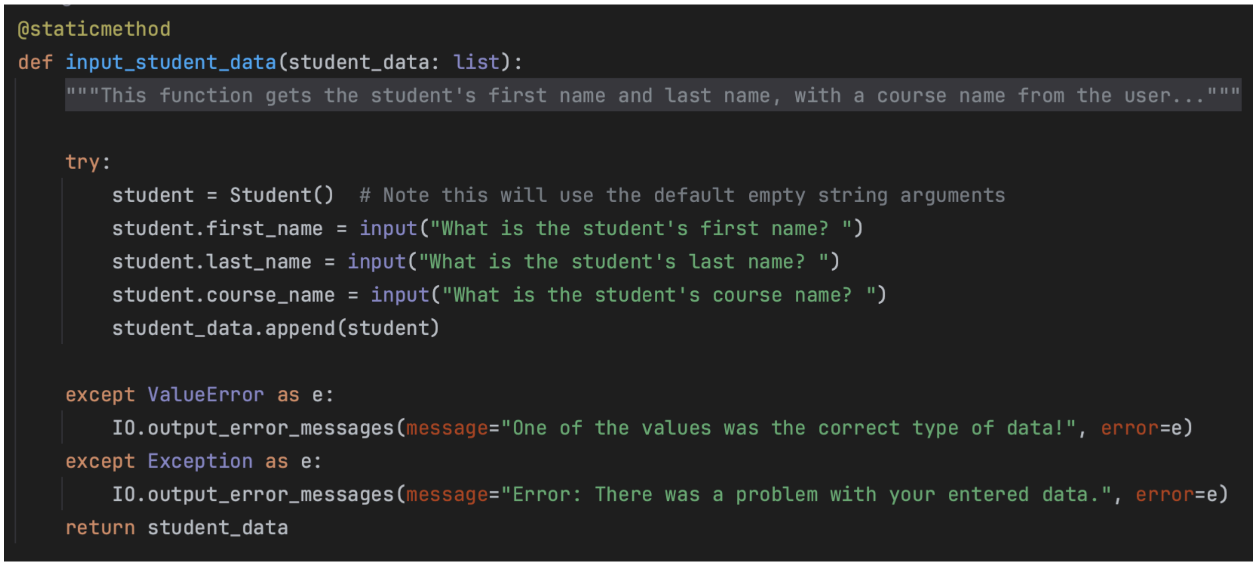
- Then I used the While loop because I wanted to keep iterating the program for taking inputs, showing the output and saving data into a json file. I again used try-except exception technique to make sure user's mistakes while entering the inputs could be caught in time as shown in Fig 4. The functions called were shown in the Fig4.



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

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

    # User selects the choice ffrom the 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_courses(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")
```

Fig 4: Processing data and exception handling.

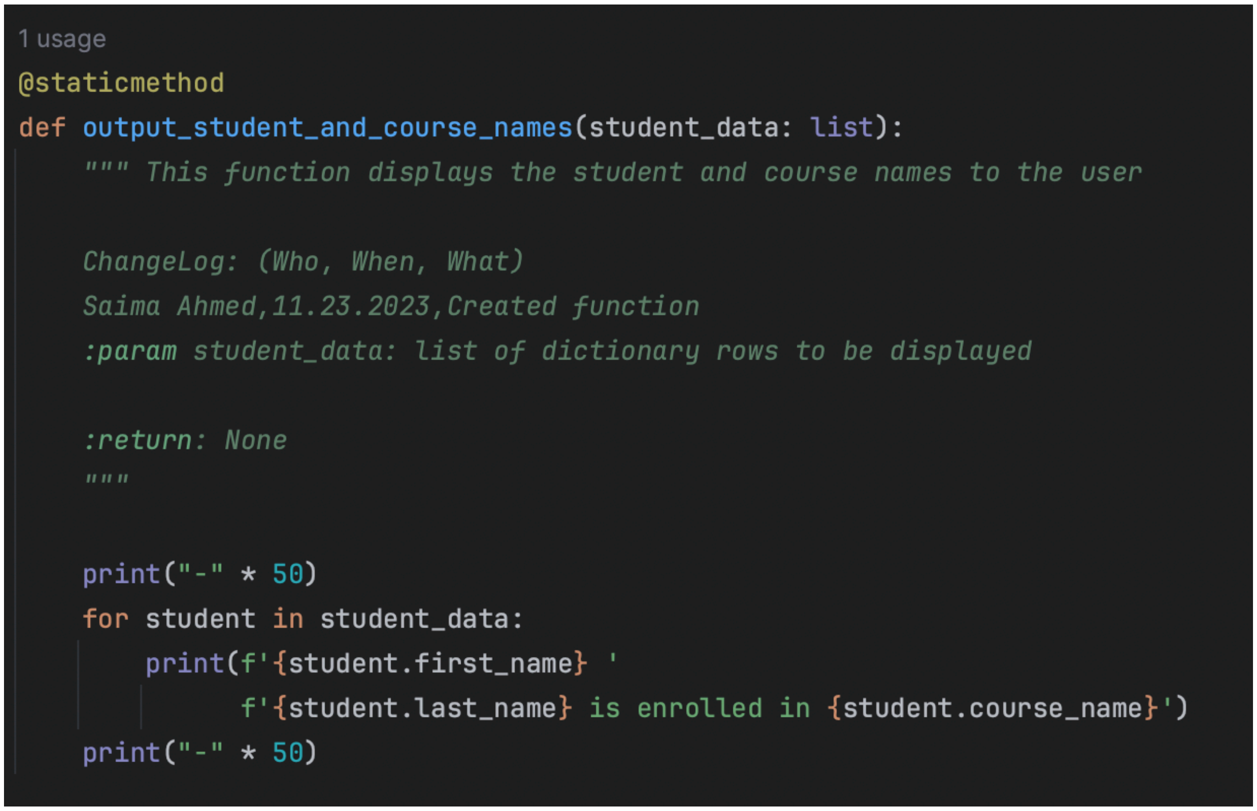
- For menu choice ‘1’ which was to register student for the course I used the function input_student_data as shown in Fig 5



```python
@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..."""

    try:
        student = Student()  # Note this will use the default empty string arguments
        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("What is the student's course name? ")
        student_data.append(student)

    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
```

Fig 5: Function input_student_data

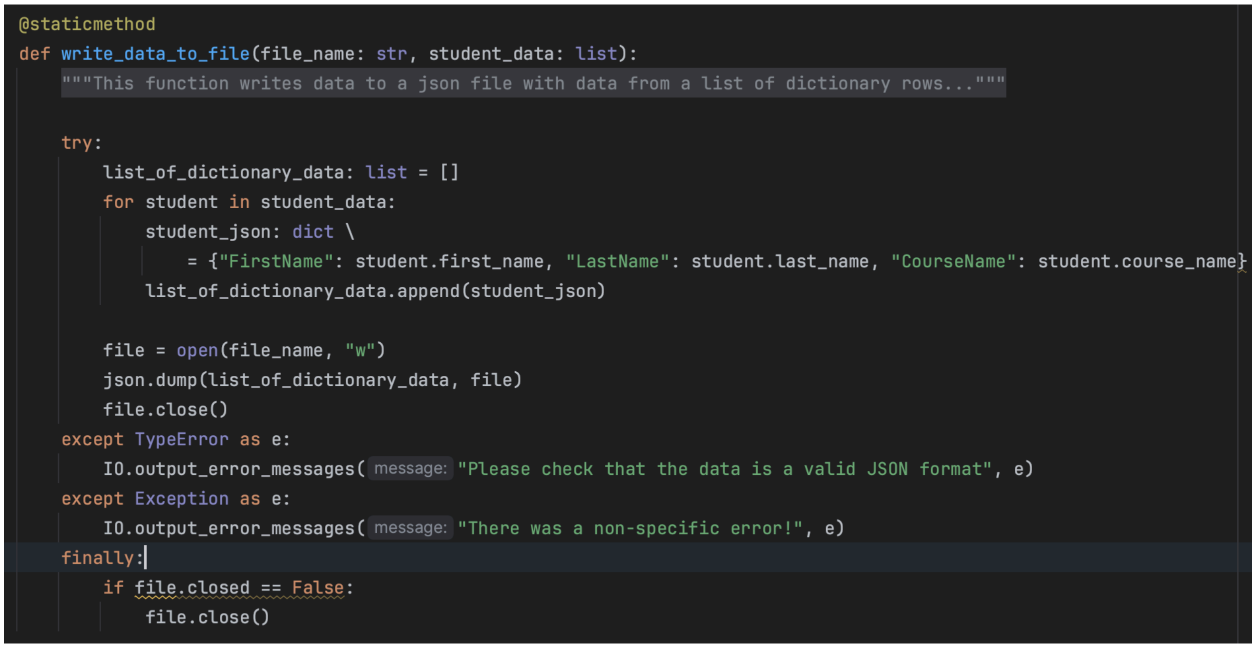
- For menu choice '2' which was to display the current data, I used the function output_student_courses as shown in Fig 6



```python
1 usage
@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)
    Saima Ahmed,11.23.2023,Created function
    :param student_data: list of dictionary rows to be displayed

    :return: None
    """

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

Fig 6: Function output_student_courses

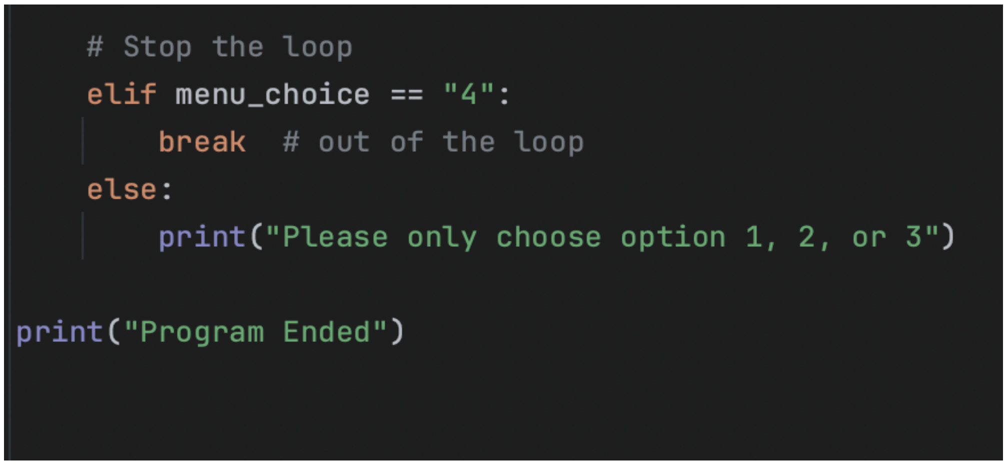
- For menu choice ' 3' which is to save the data into the file I used the function write_data_to_file



```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..."""

    try:
        list_of_dictionary_data: list = []
        for student in student_data:
            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 TypeError as e:
        IO.output_error_messages( message: "Please check that the data is a valid JSON 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 7 Function write_data_to_file

- For menu choice '4' I used the 'break' command to break the while loop and end the program as shown in Fig 8

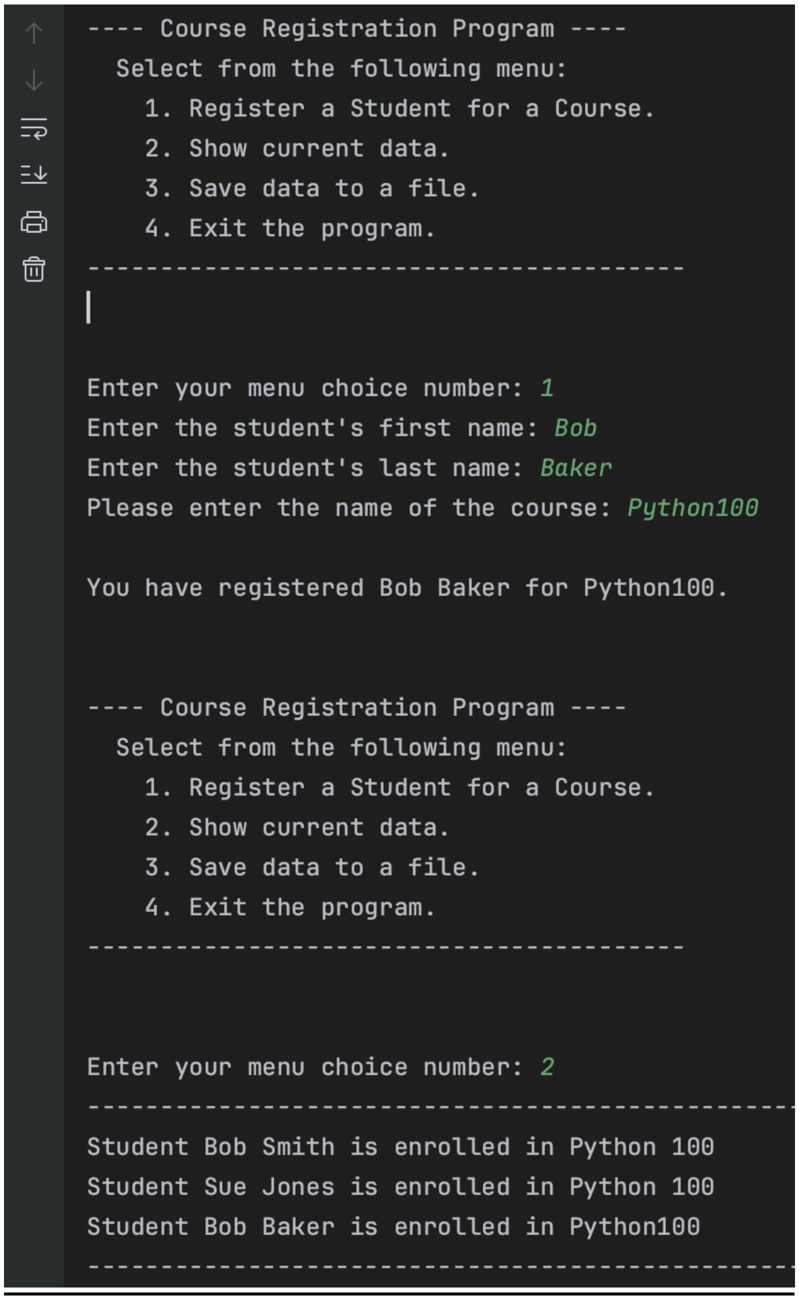


```python
    # Stop the loop
    elif menu_choice == "4":
        break  # out of the loop
    else:
        print("Please only choose option 1, 2, or 3")

print("Program Ended")
```

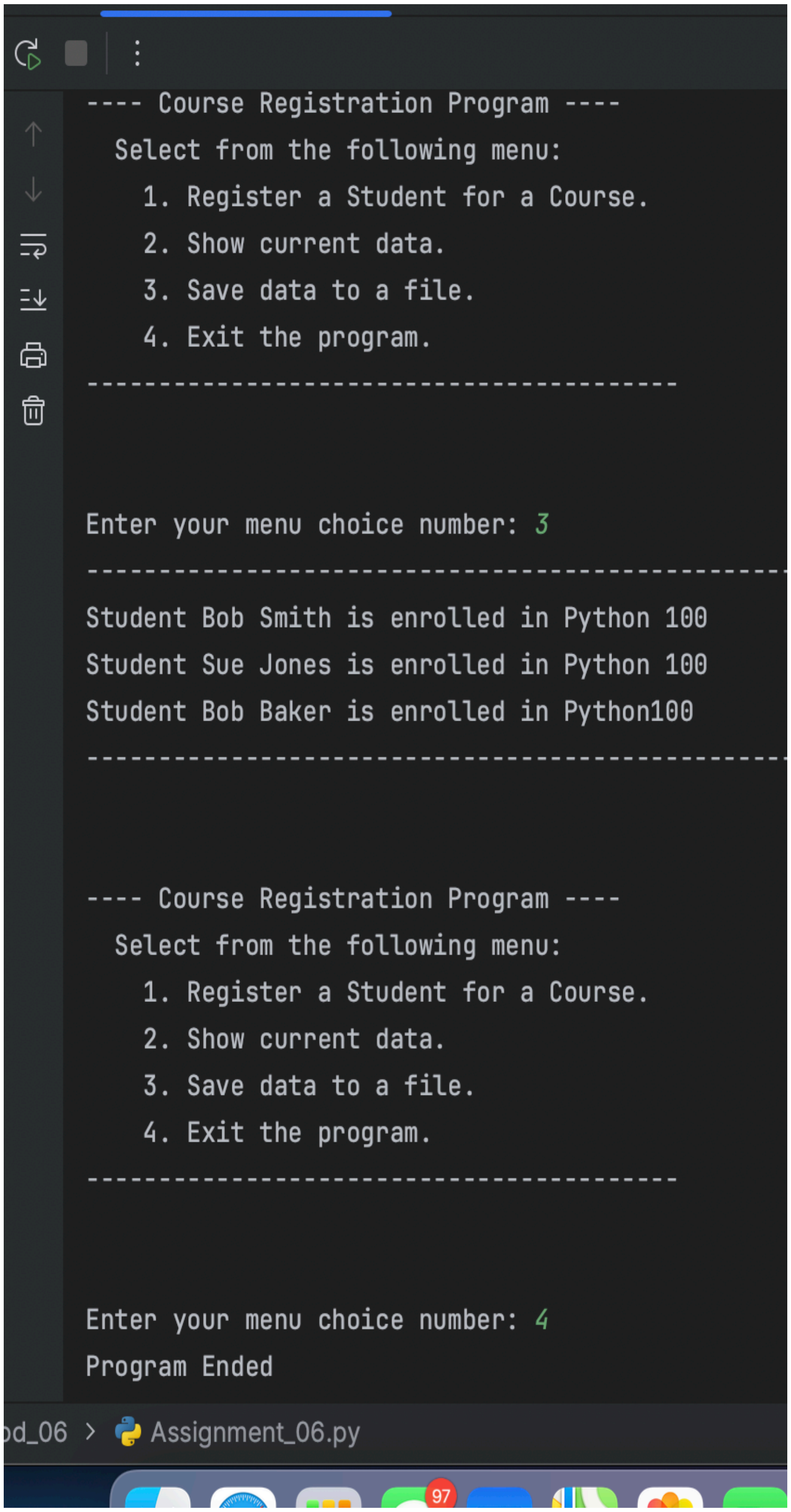
Fig 8: Ending the program

Running the program:

First I made a json file called Enrollments.json and saved some data in it. Next I ran the program and took some more inputs from the user using input command. When I ran the program on PyCharm console, below is the output shown in Fig 9:



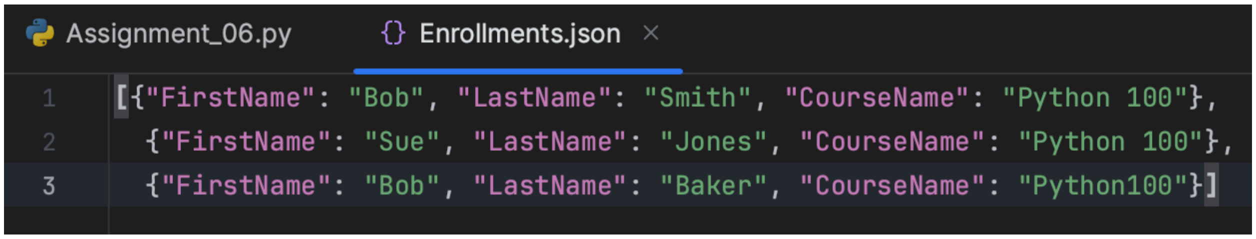
```
---- Course Registration Program ----
  Select from the following menu:
    1. Register a Student for a Course.
    2. Show current data.
    3. Save data to a file.
    4. Exit the program.
-----------------------------------------


Enter your menu choice number: 1
Enter the student's first name: Bob
Enter the student's last name: Baker
Please enter the name of the course: Python100

You have registered Bob Baker for Python100.


---- 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.
-----------------------------------------



Enter your menu choice number: 2
------------------------------------------------
Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Bob Baker is enrolled in Python100
------------------------------------------------
```



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



Enter your menu choice number: 3
------------------------------------------------------
Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Bob Baker is enrolled in Python100
------------------------------------------------------



---- 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.
-----------------------------------------



Enter your menu choice number: 4
Program Ended
```

Fig 9: Output of the program on PyCharm console

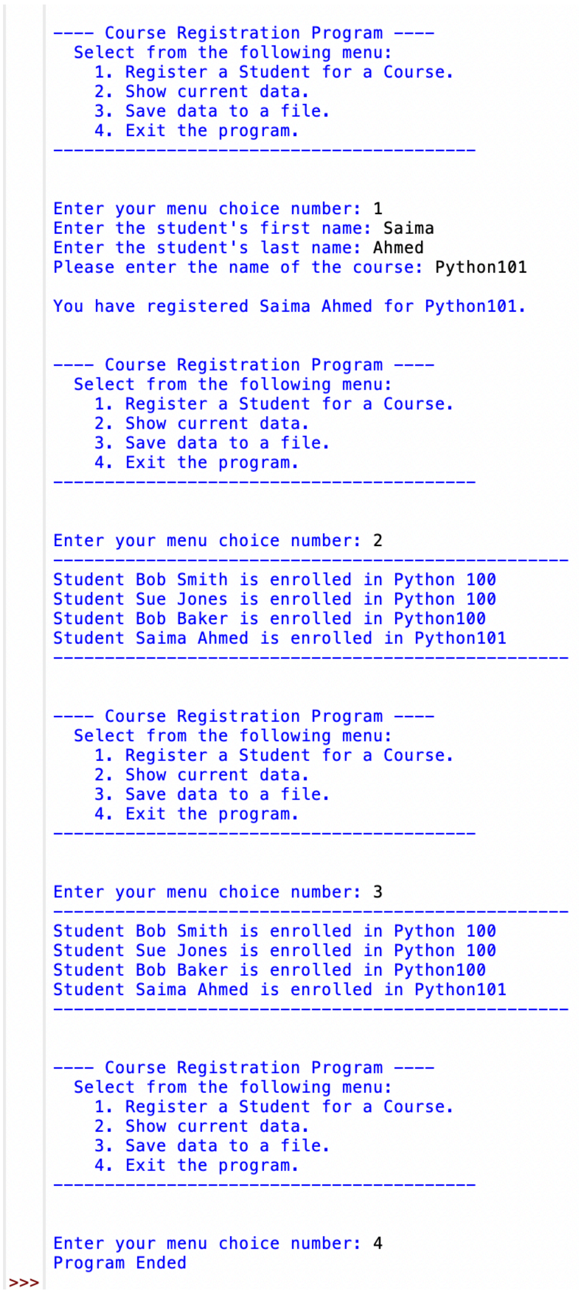
Following is my Enrollments.json file which has all the stored data in Fig 10


Assignment_06.py Enrollments.json

```
[{"FirstName": "Bob", "LastName": "Smith", "CourseName": "Python 100"},
  {"FirstName": "Sue", "LastName": "Jones", "CourseName": "Python 100"},
  {"FirstName": "Bob", "LastName": "Baker", "CourseName": "Python100"}]
```

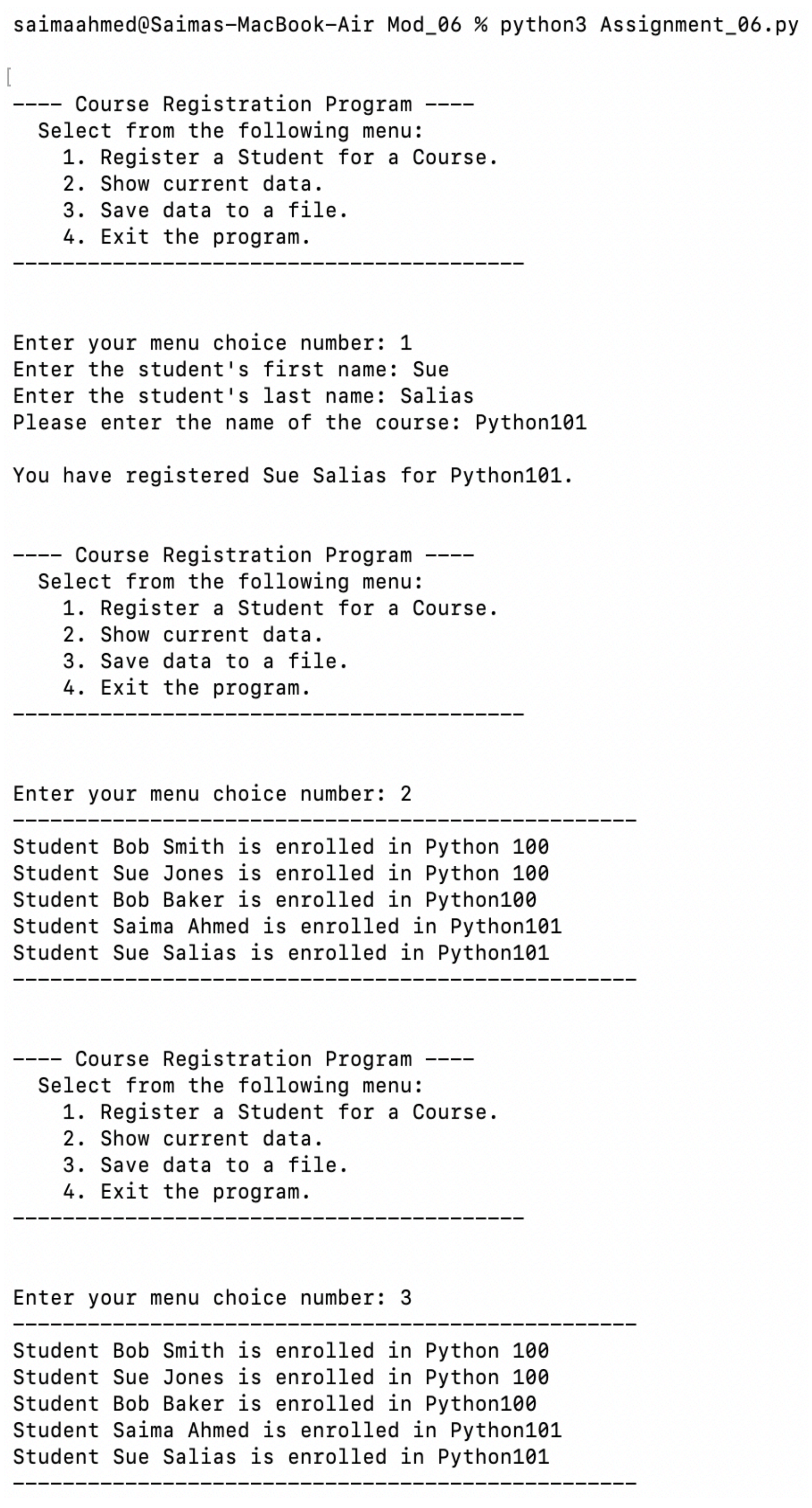
Fig 10: Json file

When I ran the program on IDLE and Terminal, below are the outputs shown in Fig 11 and Fig 12 respectively.



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


Enter your menu choice number: 1
Enter the student's first name: Saima
Enter the student's last name: Ahmed
Please enter the name of the course: Python101

You have registered Saima Ahmed for Python101.


---- 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.
-----------------------------------------


Enter your menu choice number: 2
--------------------------------------------------
Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Bob Baker is enrolled in Python100
Student Saima Ahmed is enrolled in Python101
--------------------------------------------------


---- 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.
-----------------------------------------


Enter your menu choice number: 3
--------------------------------------------------
Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Bob Baker is enrolled in Python100
Student Saima Ahmed is enrolled in Python101
--------------------------------------------------


---- 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.
-----------------------------------------


Enter your menu choice number: 4
Program Ended
>>>
```

Fig11: Output of the program on IDLE



```
saimaahmed@Saimas-MacBook-Air Mod_06 % python3 Assignment_06.py

---- 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.
-----------------------------------------


Enter your menu choice number: 1
Enter the student's first name: Sue
Enter the student's last name: Salias
Please enter the name of the course: Python101

You have registered Sue Salias for Python101.


---- 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.
-----------------------------------------


Enter your menu choice number: 2
--------------------------------------------------
Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Bob Baker is enrolled in Python100
Student Saima Ahmed is enrolled in Python101
Student Sue Salias is enrolled in Python101
--------------------------------------------------


---- 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.
-----------------------------------------


Enter your menu choice number: 3
--------------------------------------------------
Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Bob Baker is enrolled in Python100
Student Saima Ahmed is enrolled in Python101
Student Sue Salias is enrolled in Python101
--------------------------------------------------
```

Fig 12: Output of the program on Terminal

Summary:

In this module, we looked at how to work with functions and classes. With the knowledge of While loop and conditional statements, we are able to make user interactive programs that can be run as many times as we want. We also got introduced to exception handling techniques using try-except.