Traci McGrew

Randall Root

IT FDN 110 A

Assignment 07 due 5/29/24

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

Classes and Objects

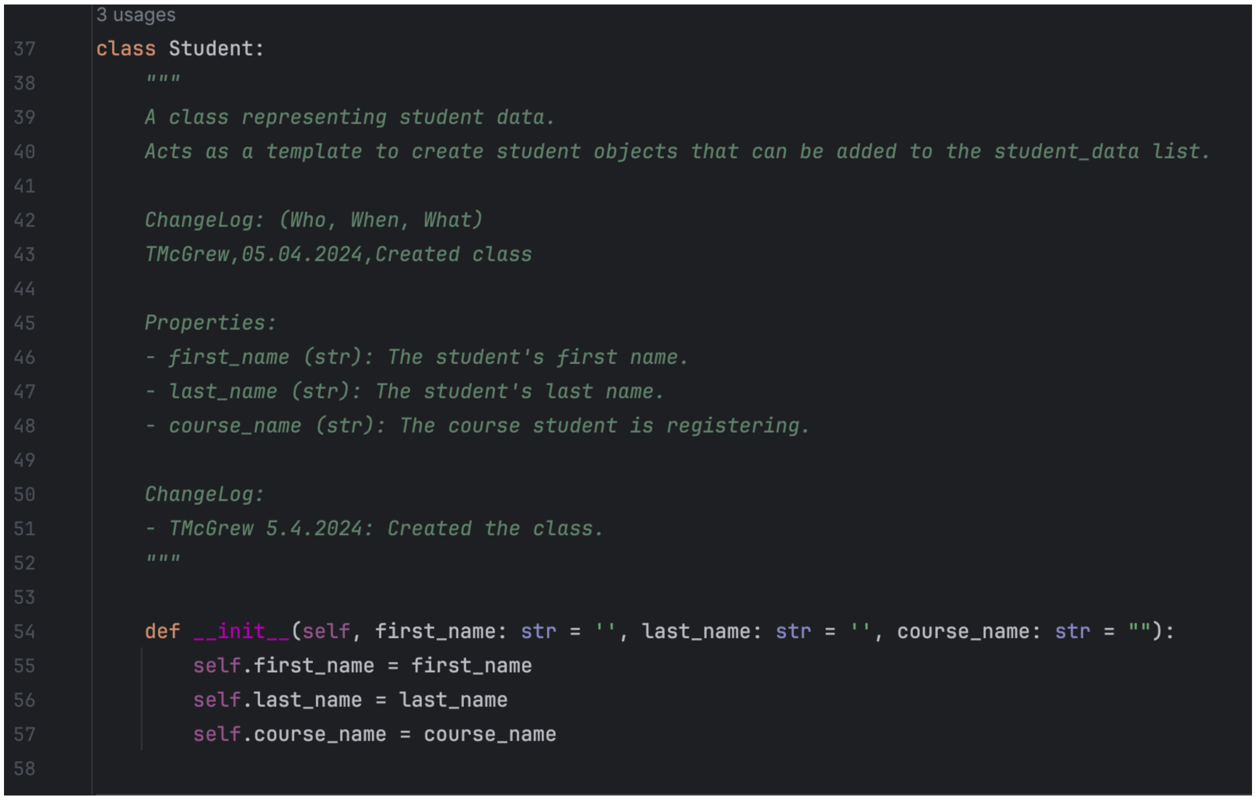
Introduction

In this paper I will go over how I completed creating a Python program like the program in Assignment06, but adds a set of data classes. I will also touch on some concepts from the module, and my observations while doing the assignments in the module.

Program Assignment 07

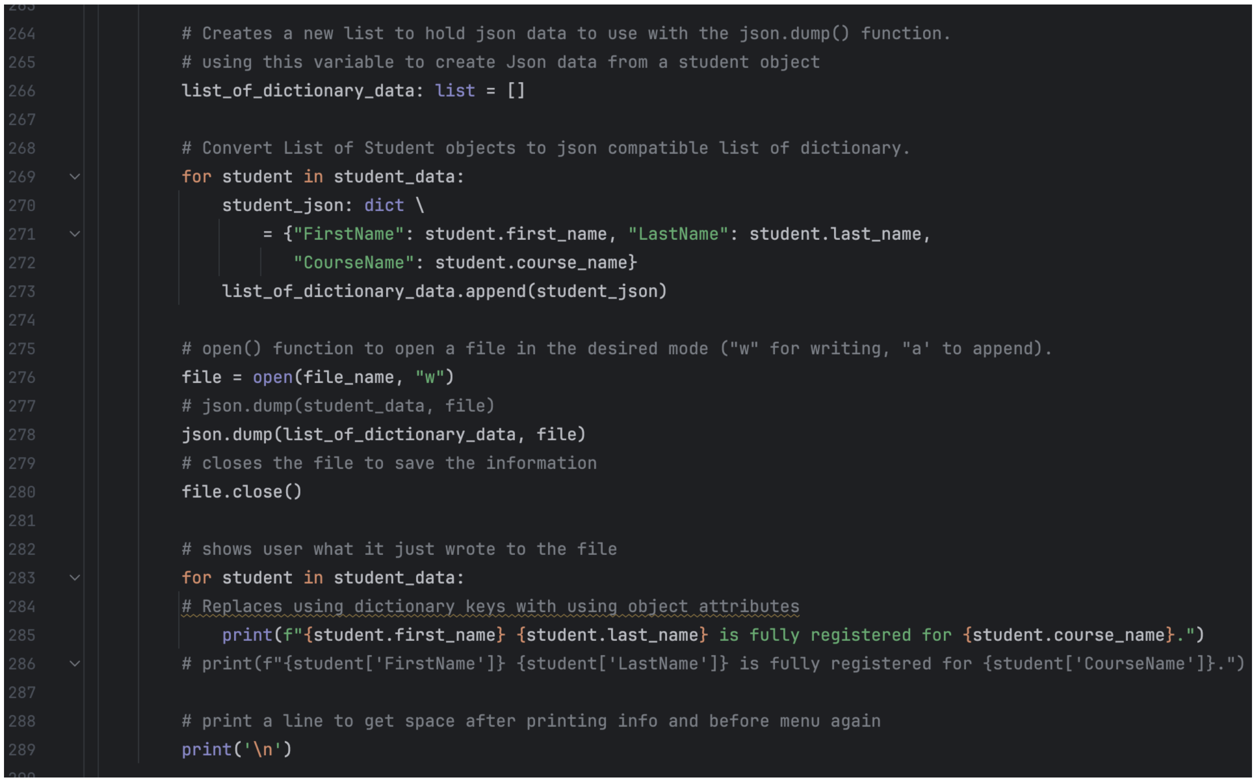
I created a program that was like the program in Assignment06, but builds on those concepts to also demonstrate using a set of data classes.

The program functions the same as in Assignment06. The code was just refactored to include data classes. I first made a Student class as shown in figure 1. Then I changed the write/read from file methods to use the list of Student objects. This involved changing each student object into a dictionary row to write to a file using *json.dump( )* and back from a dictionary row into a Student object again when reading from a file as json can't work with the object directly. This is shown in figures 2 and 3.



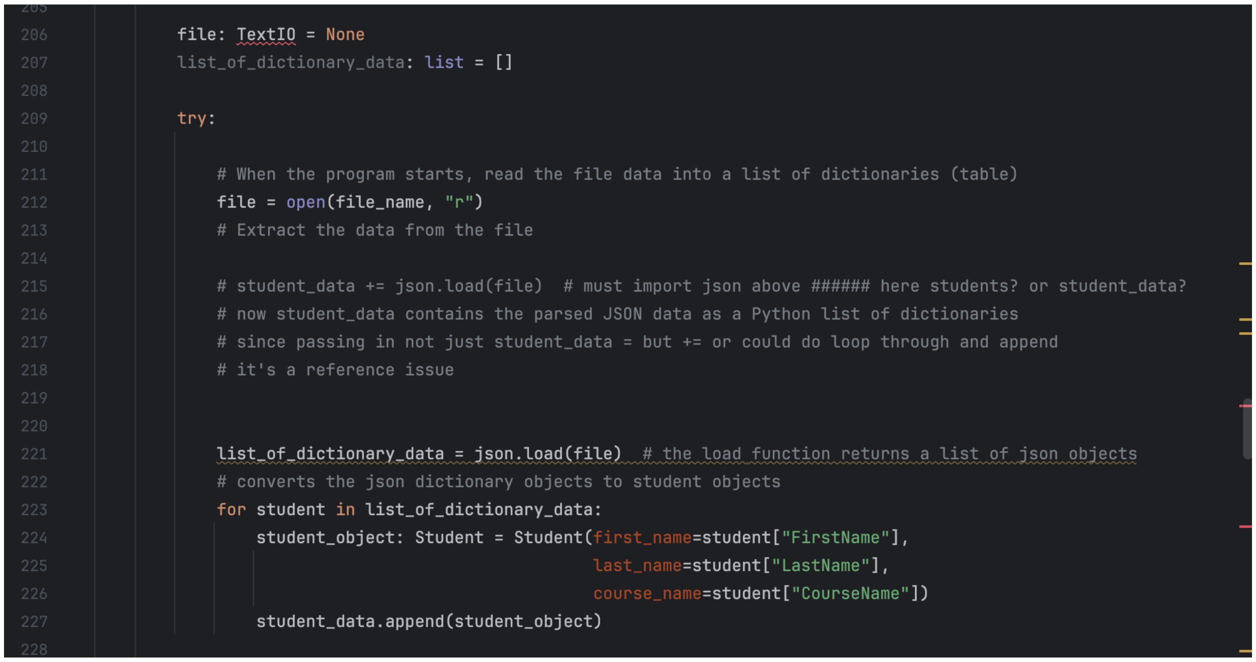
```
3 usages
class Student:
    """
    A class representing student data.
    Acts as a template to create student objects that can be added to the student_data list.

    ChangeLog: (Who, When, What)
    TMcGrew,05.04.2024,Created class

    Properties:
    - first_name (str): The student's first name.
    - last_name (str): The student's last name.
    - course_name (str): The course student is registering.

    ChangeLog:
    - TMcGrew 5.4.2024: Created the class.
    """

    def __init__(self, first_name: str = '', last_name: str = '', course_name: str = ""):
        self.first_name = first_name
        self.last_name = last_name
        self.course_name = course_name

```

Figure 1 Shows the new Student class



```
        # Creates a new list to hold json data to use with the json.dump() function.
        # using this variable to create Json data from a student object
        list_of_dictionary_data: list = []

        # Convert List of Student objects to json compatible list of dictionary.
        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)

        # open() function to open a file in the desired mode ("w" for writing, "a' to append).
        file = open(file_name, "w")
        # json.dump(student_data, file)
        json.dump(list_of_dictionary_data, file)
        # closes the file to save the information
        file.close()

        # shows user what it just wrote to the file
        for student in student_data:
        # Replaces using dictionary keys with using object attributes
            print(f"{student.first_name} {student.last_name} is fully registered for {student.course_name}.")
        # print(f"{student['FirstName']} {student['LastName']} is fully registered for {student['CourseName']}.")

        # print a line to get space after printing info and before menu again
        print('\n')
```

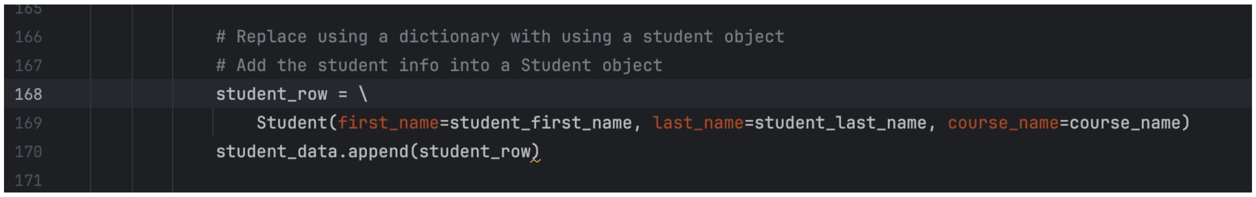
Figure 2 Shows the changes to the writing to a file had to use list_of_dictionary_data variable to convert it to a list of dictionaries



```
        file: TextIO = None
        list_of_dictionary_data: list = []

        try:

            # When the program starts, read the file data into a list of dictionaries (table)
            file = open(file_name, "r")
            # Extract the data from the file

            # student_data += json.load(file)  # must import json above ###### here students? or student_data?
            # now student_data contains the parsed JSON data as a Python list of dictionaries
            # since passing in not just student_data = but += or could do loop through and append
            # it's a reference issue


            list_of_dictionary_data = json.load(file)  # the load function returns a list of json objects
            # converts the json dictionary objects to student objects
            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)
```

Figure 3 Shows the changes to the reading from a file had to use list_of_dictionary_data variable to get it and then loop through to put into list of student objects

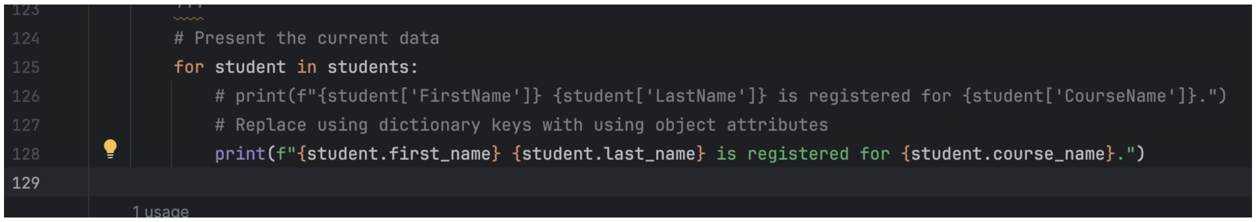
I then changed several functions to now work with the list of objects instead of a list of dictionaries. Figure 4 shows where I changed *input_student_data(student_data= list)* method to now make a Student object instead of a dictionary row and adds it to the list that was passed in.



```
        # Replace using a dictionary with using a student object
        # Add the student info into a Student object
        student_row = \
            Student(first_name=student_first_name, last_name=student_last_name, course_name=course_name)
        student_data.append(student_row)
```

Figure 4 Shows the changes to the input_student_data(student_data= list) method where it now makes an object row instead of a dictionary row and adds it to the table (list of objects) passed in

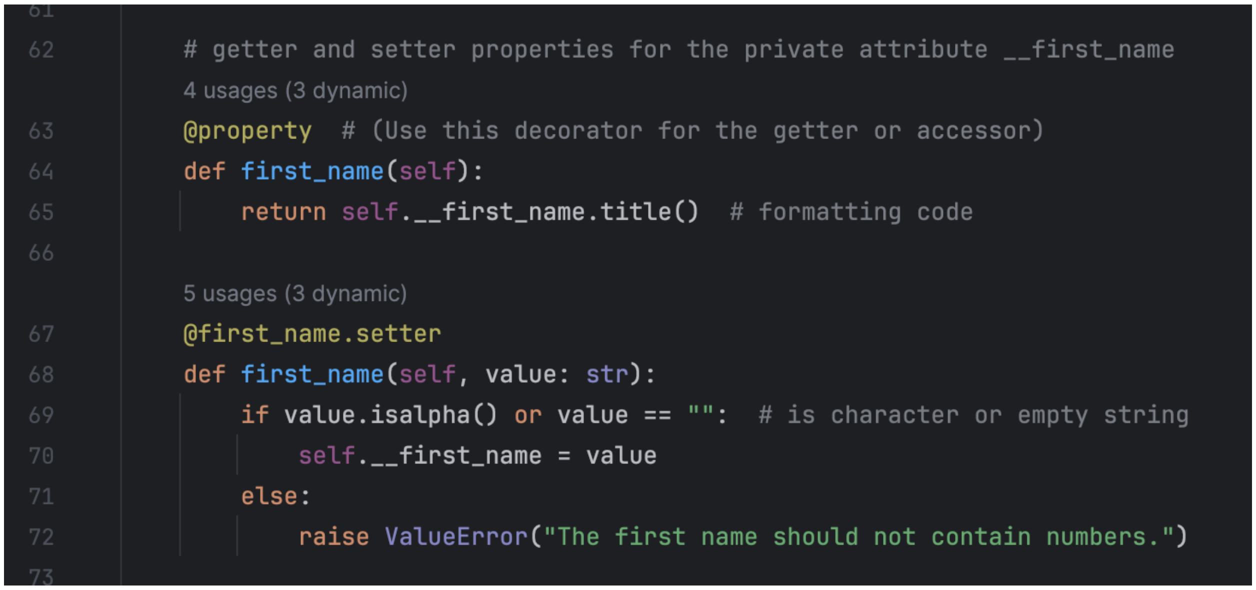
Several other places where it used to use the dictionary were also changed to use the object now instead. This included the *output_student_courses(student_data= list)* method as shown in figure 5.



```
    # Present the current data
    for student in students:
        # print(f"{student['FirstName']} {student['LastName']} is registered for {student['CourseName']}.")
        # Replace using dictionary keys with using object attributes
        print(f"{student.first_name} {student.last_name} is registered for {student.course_name}.")

    1 usage
```

Figure 5 Shows the changes to the output_student_courses(student_data= list) method where it makes a f-string now with the object row instead

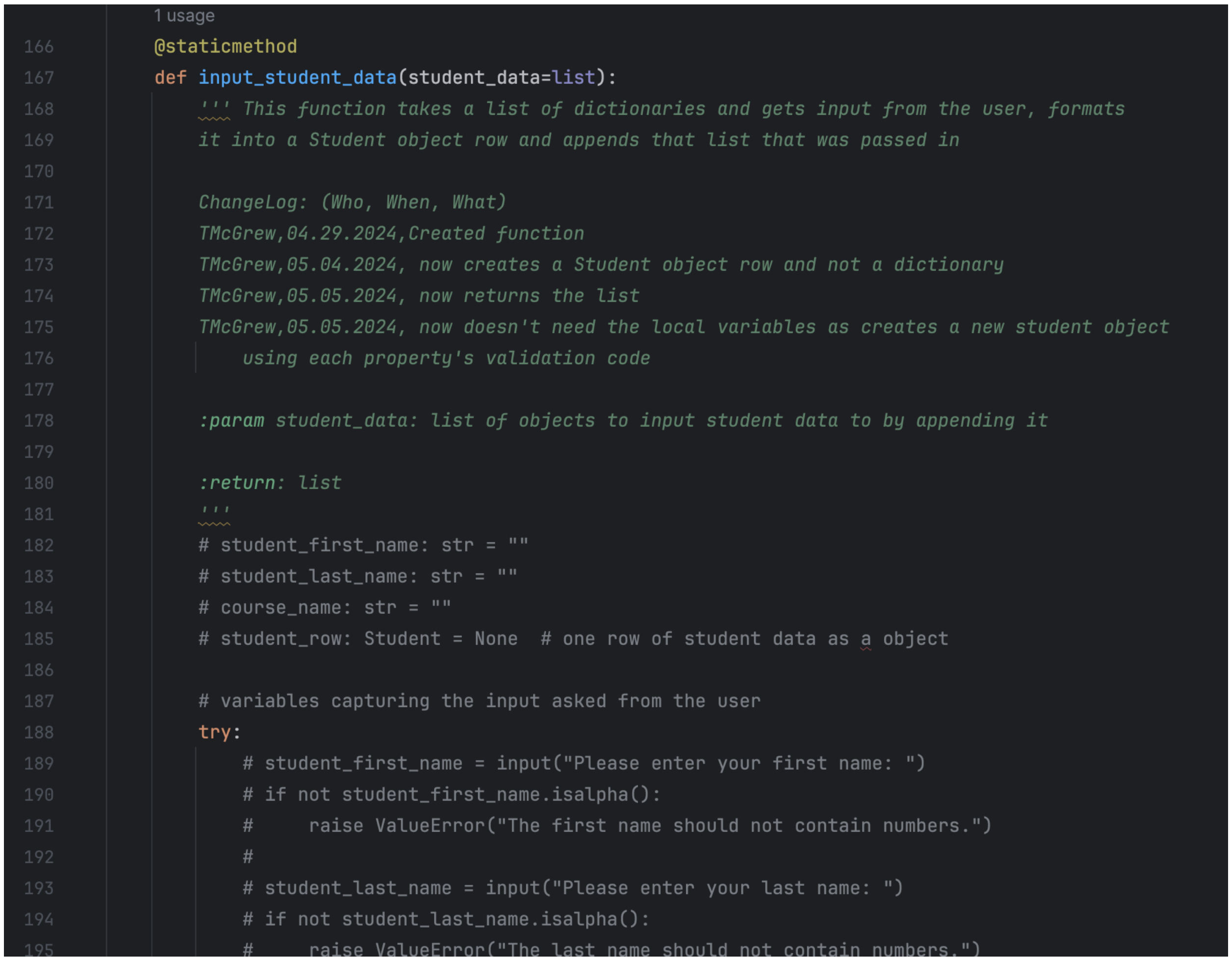
I then added properties and private attributes to the Student class. This involved getter and setter methods for the first name, last name, and course name. This also involved changing where it checked if the user entered alphabetic characters or not from the *input_student_data(student_data_list)* method to the new getter and setters for the first and last student name. This can be shown in figure 6.



```
    # getter and setter properties for the private attribute __first_name
    4 usages (3 dynamic)
    @property  # (Use this decorator for the getter or accessor)
    def first_name(self):
        return self.__first_name.title()  # formatting code

    5 usages (3 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.")
```

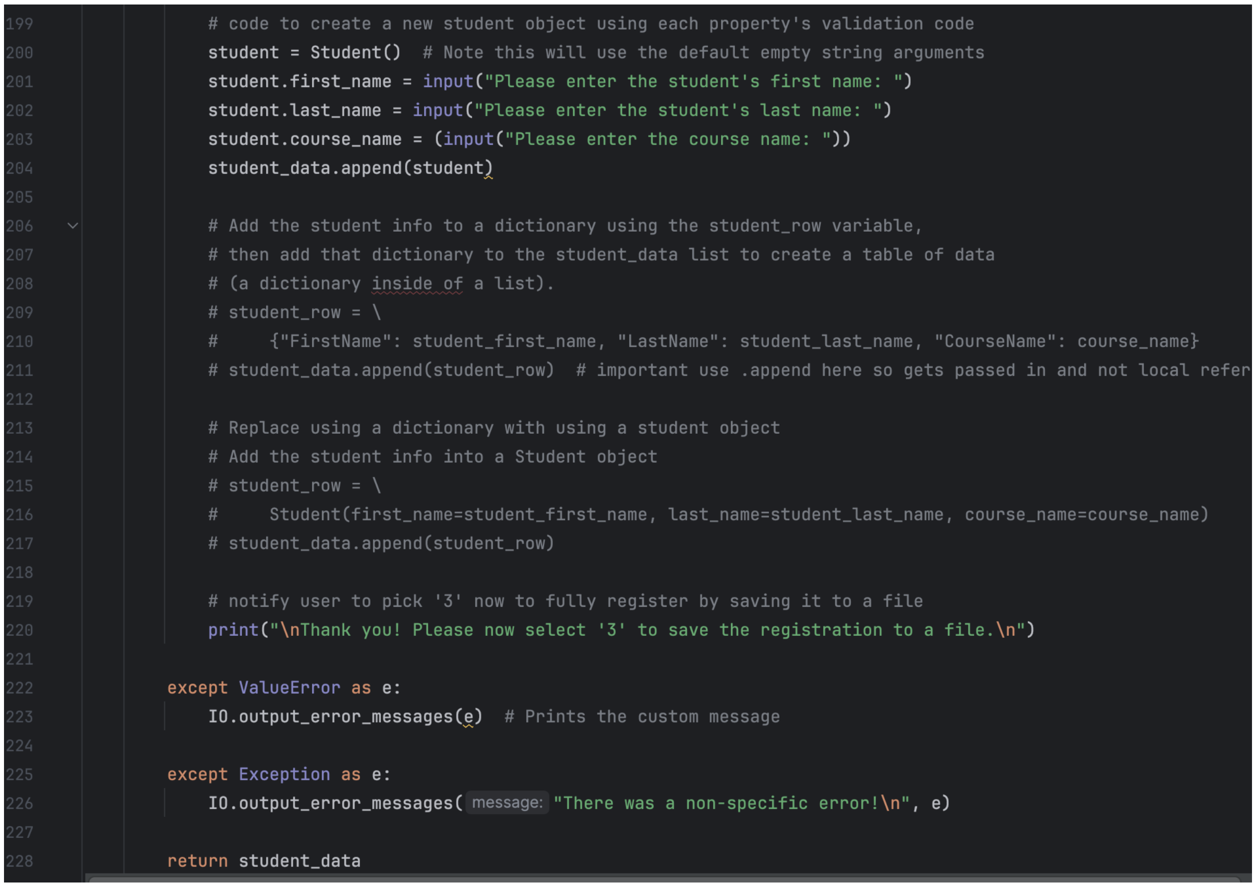
Figure 6 Shows the new getter and setter for __first_name

Because of the new getter and setter methods, the code could be reworked and the local variables for the *input_student_data(student_data_list)* method could be removed. I also changed that function to now return the list it takes in and changed the calling code to be equal to the list so it could use the returned list. Figure 7 and 8 shows the changes to the method and figure 9 shows where I changed where I called it in menu option 1.



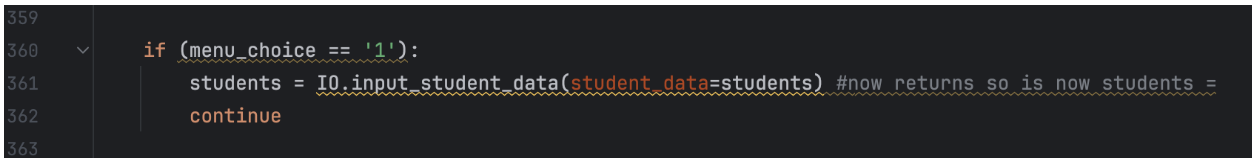
```
        1 usage
        @staticmethod
        def input_student_data(student_data=list):
            ''' This function takes a list of dictionaries and gets input from the user, formats
            it into a Student object row and appends that list that was passed in

            ChangeLog: (Who, When, What)
            TMcGrew,04.29.2024,Created function
            TMcGrew,05.04.2024, now creates a Student object row and not a dictionary
            TMcGrew,05.05.2024, now returns the list
            TMcGrew,05.05.2024, now doesn't need the local variables as creates a new student object
                using each property's validation code

            :param student_data: list of objects to input student data to by appending it

            :return: list
            '''
            # student_first_name: str = ""
            # student_last_name: str = ""
            # course_name: str = ""
            # student_row: Student = None  # one row of student data as a object

            # variables capturing the input asked from the user
            try:
                # student_first_name = input("Please enter your first name: ")
                # if not student_first_name.isalpha():
                #     raise ValueError("The first name should not contain numbers.")
                #
                # student_last_name = input("Please enter your last name: ")
                # if not student_last_name.isalpha():
                #     raise ValueError("The last name should not contain numbers.")
```

Figure 7 Shows the reworked input_student_data(student_data_list) -> list method



```
        # code to create a new student object using each property's validation code
        student = Student()  # Note this will use the default empty string arguments
        student.first_name = input("Please enter the student's first name: ")
        student.last_name = input("Please enter the student's last name: ")
        student.course_name = (input("Please enter the course name: "))
        student_data.append(student)

        # Add the student info to a dictionary using the student_row variable,
        # then add that dictionary to the student_data list to create a table of data
        # (a dictionary inside of a list).
        # student_row = \
        #     {"FirstName": student_first_name, "LastName": student_last_name, "CourseName": course_name}
        # student_data.append(student_row)  # important use .append here so gets passed in and not local refer

        # Replace using a dictionary with using a student object
        # Add the student info into a Student object
        # student_row = \
        #     Student(first_name=student_first_name, last_name=student_last_name, course_name=course_name)
        # student_data.append(student_row)

        # notify user to pick '3' now to fully register by saving it to a file
        print("\nThank you! Please now select '3' to save the registration to a file.\n")

    except ValueError as e:
        IO.output_error_messages(e)  # Prints the custom message

    except Exception as e:
        IO.output_error_messages( message: "There was a non-specific error!\n", e)

    return student_data
```

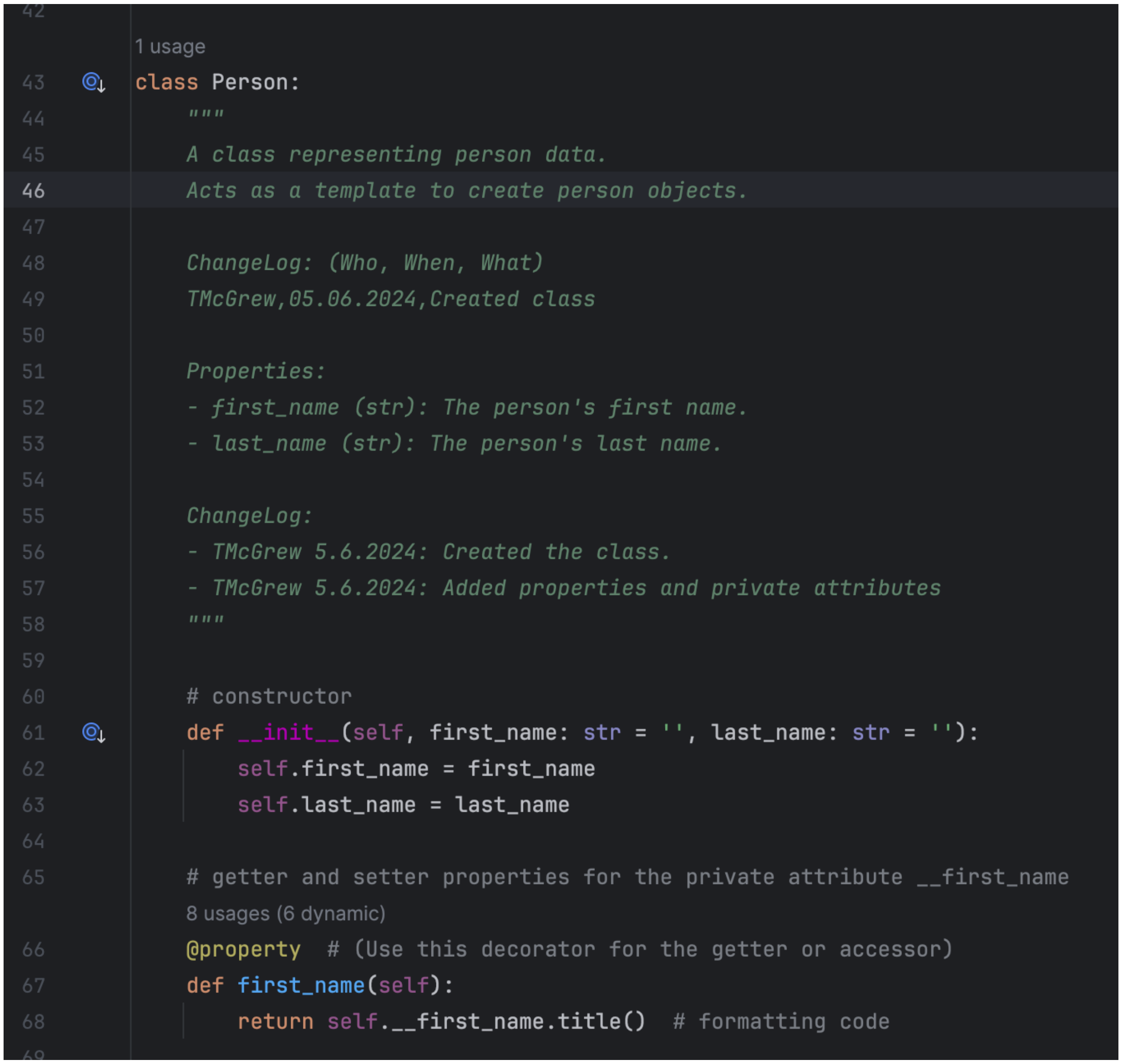
Figure 8 Continuation of figure 7



```
    if (menu_choice == '1'):
        students = IO.input_student_data(student_data=students) #now returns so is now students =
        continue
```

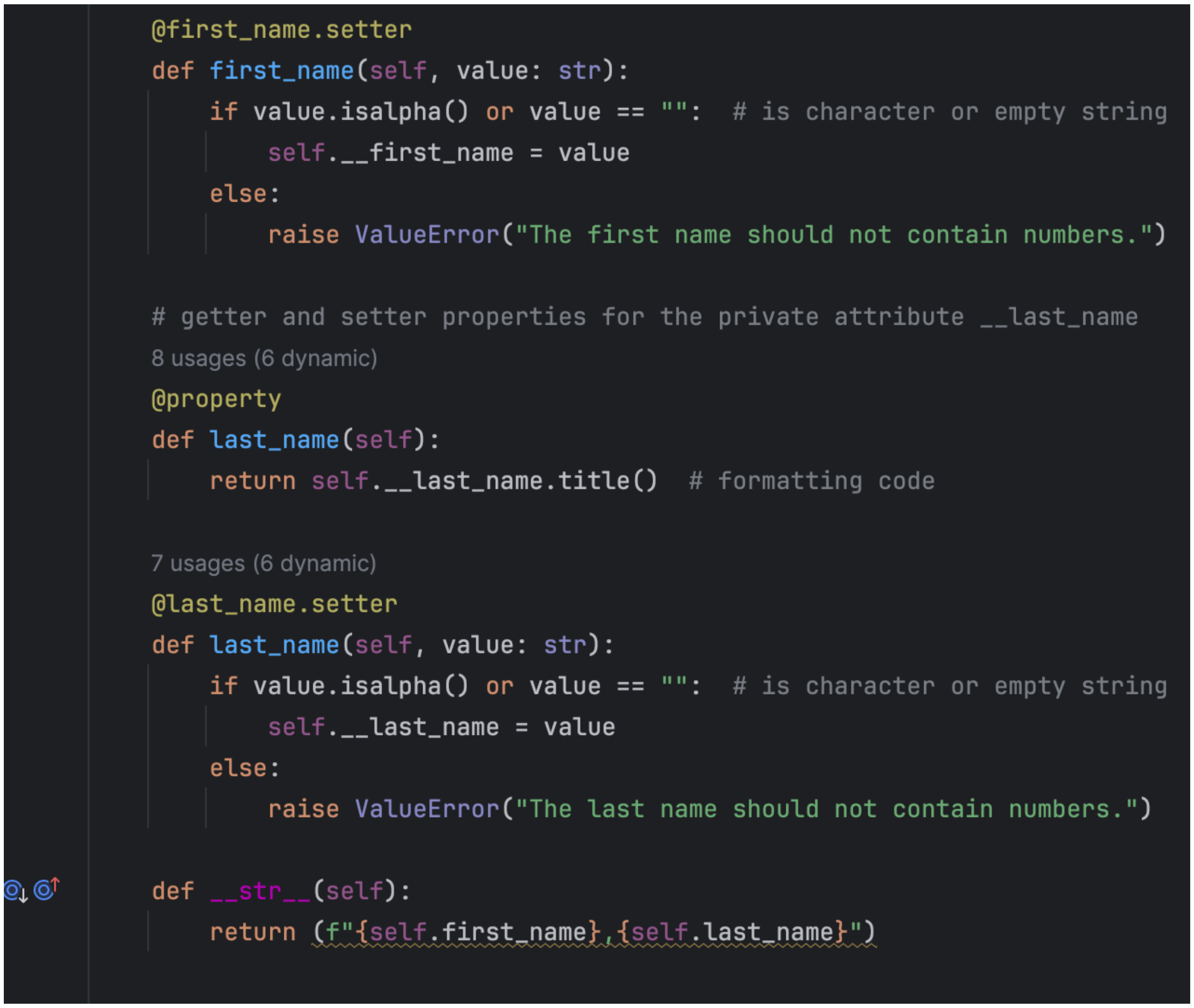
Figure 9 Shows the function call is now set to a list as one is returned now from the function

Next, I added a Person class from which the Student class can inherit. The Person class now handles the first and last name and the Student class extends that to also have a course name. This involved creating the Person class and moving the getter and setter code for the first name and last name attributes up to the Person class. The Person class is shown in figures 10 and 11. Figure 12 shows the changes of the Student class with it now inheriting from the Person class.



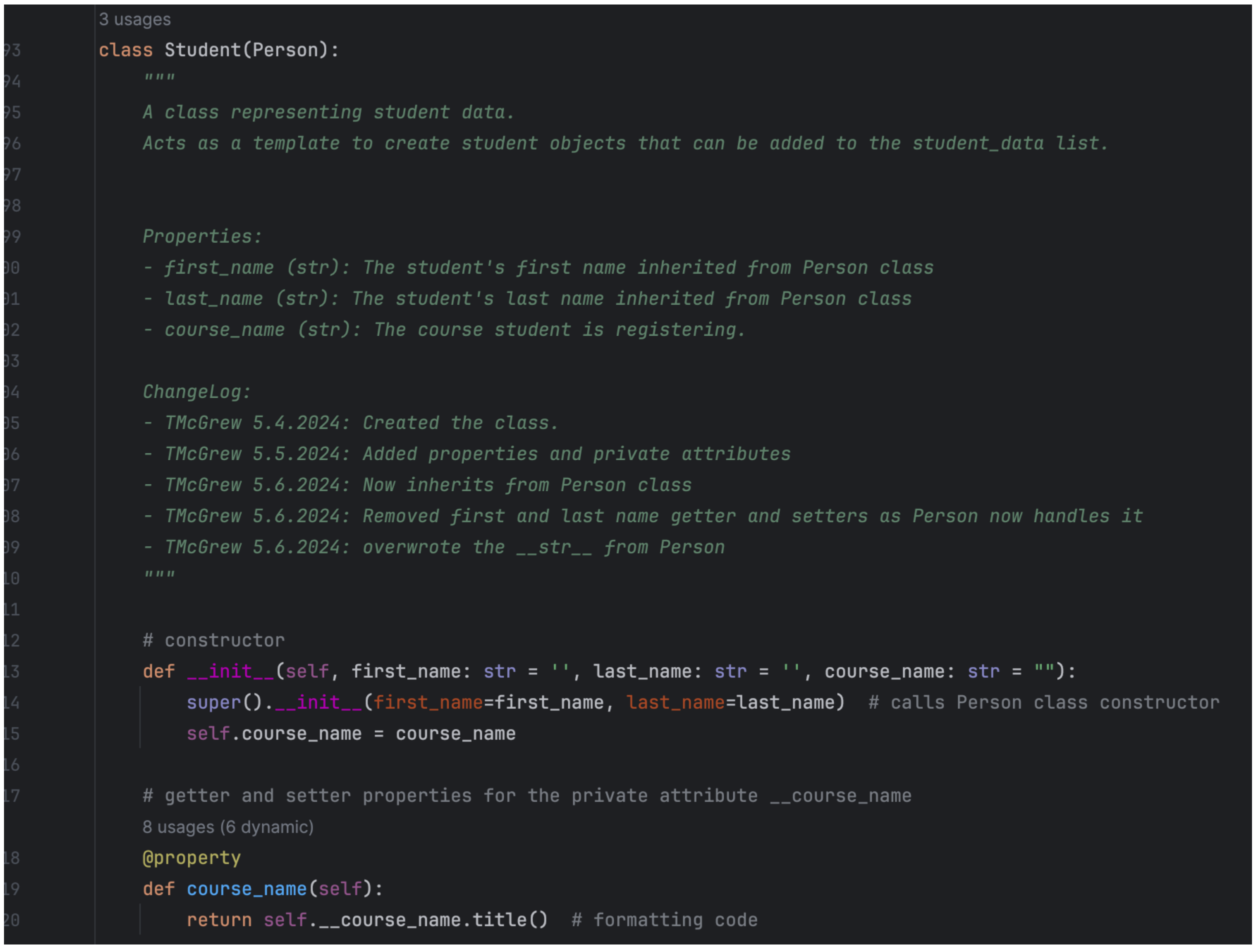
```
1 usage
class Person:
    """
    A class representing person data.
    Acts as a template to create person objects.

    ChangeLog: (Who, When, What)
    TMcGrew,05.06.2024,Created class

    Properties:
    - first_name (str): The person's first name.
    - last_name (str): The person's last name.

    ChangeLog:
    - TMcGrew 5.6.2024: Created the class.
    - TMcGrew 5.6.2024: Added properties and private attributes
    """

    # constructor
    def __init__(self, first_name: str = '', last_name: str = ''):
        self.first_name = first_name
        self.last_name = last_name

    # getter and setter properties for the private attribute __first_name
    8 usages (6 dynamic)
    @property  # (Use this decorator for the getter or accessor)
    def first_name(self):
        return self.__first_name.title()  # formatting code
```

Figure 10 Person class



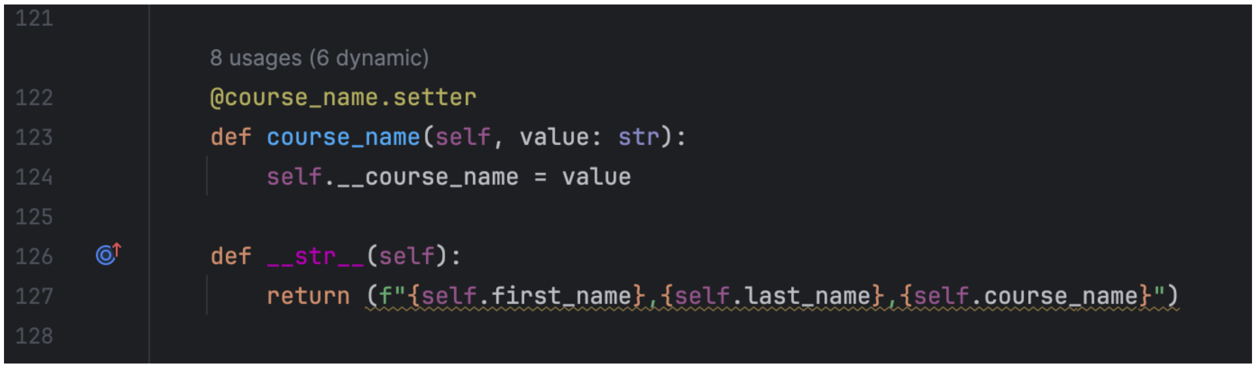
```
    @first_name.setter
    def first_name(self, value: str):
        if value.isalpha() or value == "":  # is character or empty string
            self.__first_name = value
        else:
            raise ValueError("The first name should not contain numbers.")

    # getter and setter properties for the private attribute __last_name
    8 usages (6 dynamic)
    @property
    def last_name(self):
        return self.__last_name.title()  # formatting code

    7 usages (6 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.")

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

Figure 11 Continuation of figure 10



```
3 usages
class Student(Person):
    """
    A class representing student data.
    Acts as a template to create student objects that can be added to the student_data list.


    Properties:
    - first_name (str): The student's first name inherited from Person class
    - last_name (str): The student's last name inherited from Person class
    - course_name (str): The course student is registering.

    ChangeLog:
    - TMcGrew 5.4.2024: Created the class.
    - TMcGrew 5.5.2024: Added properties and private attributes
    - TMcGrew 5.6.2024: Now inherits from Person class
    - TMcGrew 5.6.2024: Removed first and last name getter and setters as Person now handles it
    - TMcGrew 5.6.2024: overwrote the __str__ from Person
    """

    # constructor
    def __init__(self, first_name: str = '', last_name: str = '', course_name: str = ""):
        super().__init__(first_name=first_name, last_name=last_name)  # calls Person class constructor
        self.course_name = course_name

    # getter and setter properties for the private attribute __course_name
    8 usages (6 dynamic)
    @property
    def course_name(self):
        return self.__course_name.title()  # formatting code
```

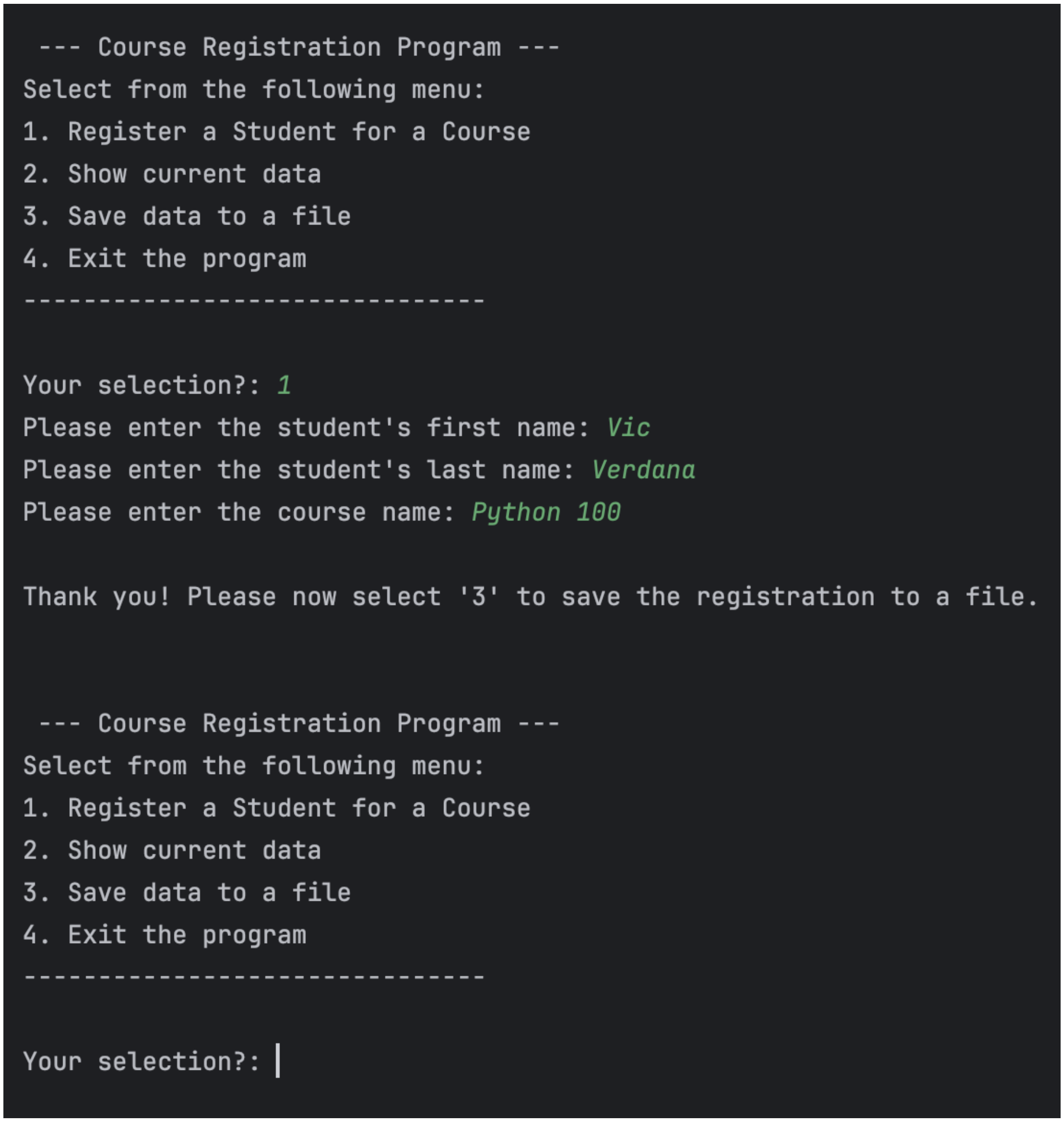
Figure 12 Student class inheriting from the Person class



```

    8 usages (6 dynamic)
    @course_name.setter
    def course_name(self, value: str):
        self.__course_name = value

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

```

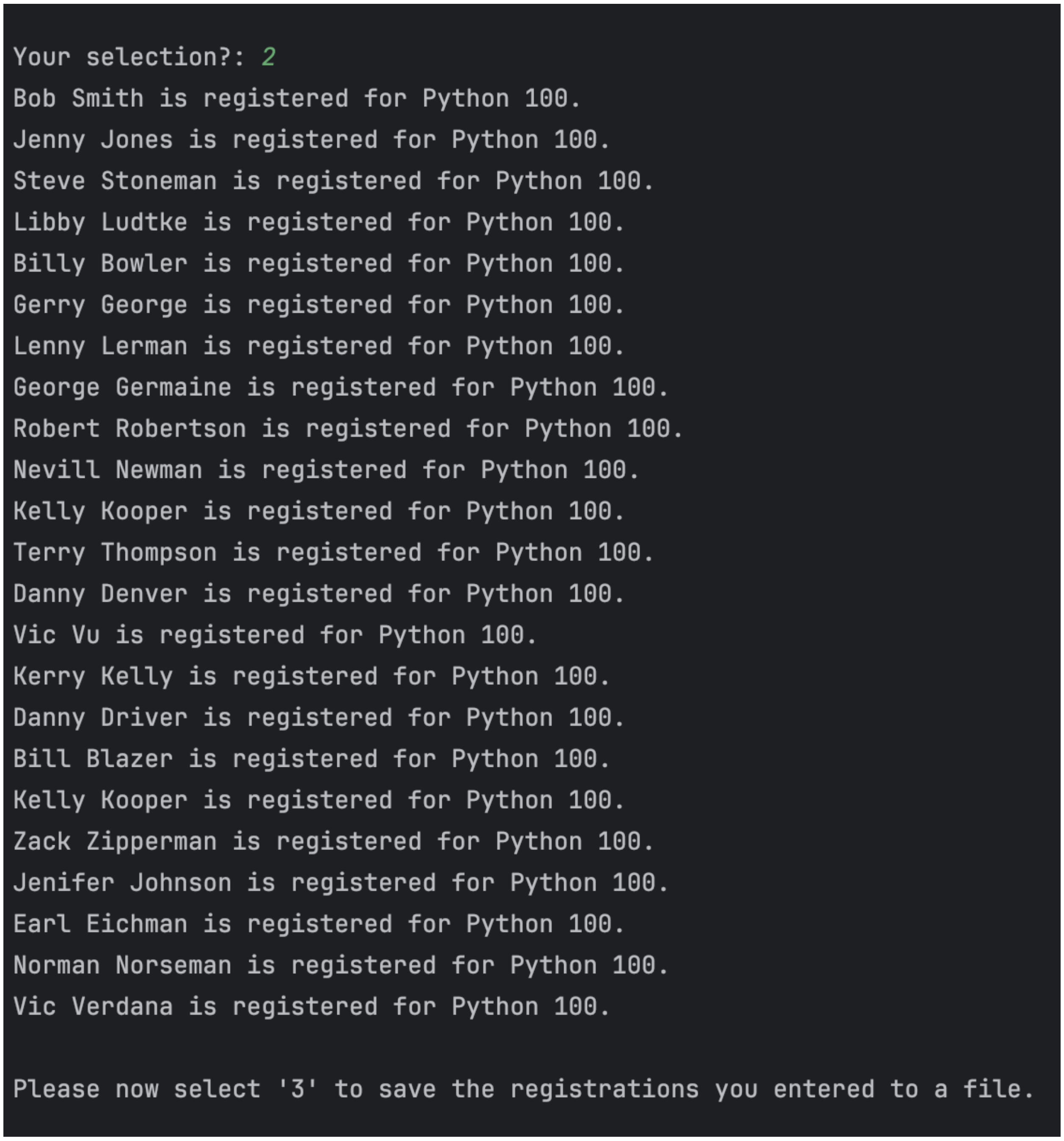
Figure 13 Continuation of figure 12

The program runs from a user's perspective the exact same as it did in previous iterations. The following figures show it running in both PyCharm and Terminal on Mac OS and the contents of the Enrollments.json file.



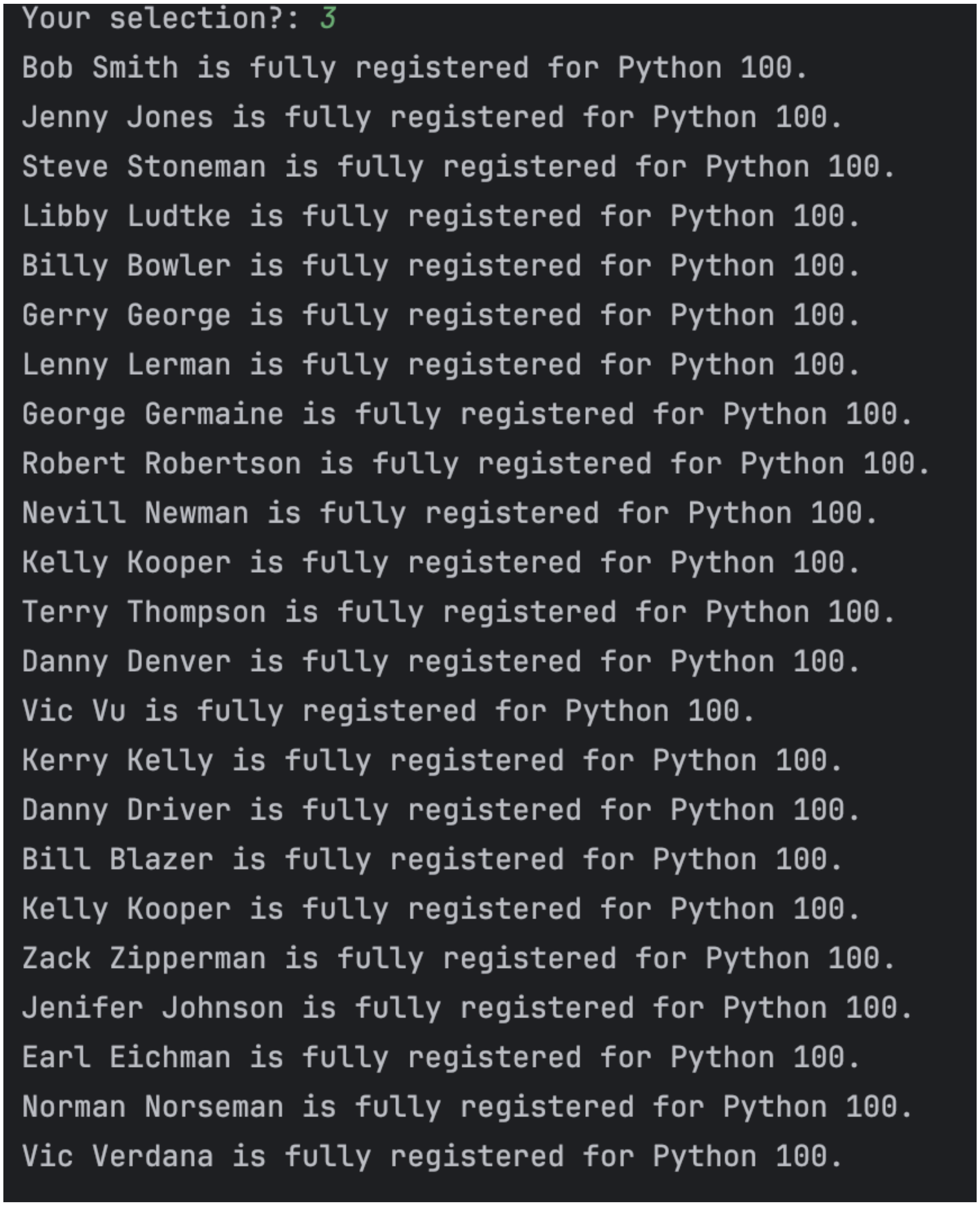
```
 --- Course Registration Program ---
Select from the following menu:
1. Register a Student for a Course
2. Show current data
3. Save data to a file
4. Exit the program
------------------------------

Your selection?: 1
Please enter the student's first name: Vic
Please enter the student's last name: Verdana
Please enter the course name: Python 100

Thank you! Please now select '3' to save the registration to a file.


 --- 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
------------------------------

Your selection?: 
```

Figure 14 Module running in PyCharm



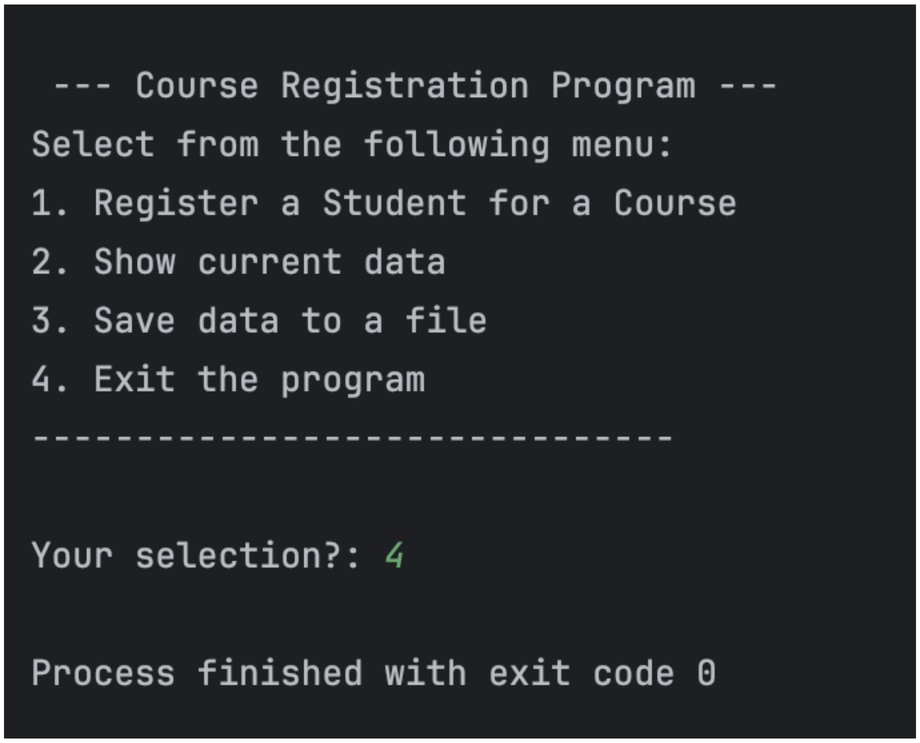
```
Your selection?: 2
Bob Smith is registered for Python 100.
Jenny Jones is registered for Python 100.
Steve Stoneman is registered for Python 100.
Libby Ludtke is registered for Python 100.
Billy Bowler is registered for Python 100.
Gerry George is registered for Python 100.
Lenny Lerman is registered for Python 100.
George Germaine is registered for Python 100.
Robert Robertson is registered for Python 100.
Nevill Newman is registered for Python 100.
Kelly Kooper is registered for Python 100.
Terry Thompson is registered for Python 100.
Danny Denver is registered for Python 100.
Vic Vu is registered for Python 100.
Kerry Kelly is registered for Python 100.
Danny Driver is registered for Python 100.
Bill Blazer is registered for Python 100.
Kelly Kooper is registered for Python 100.
Zack Zipperman is registered for Python 100.
Jenifer Johnson is registered for Python 100.
Earl Eichman is registered for Python 100.
Norman Norseman is registered for Python 100.
Vic Verdana is registered for Python 100.

Please now select '3' to save the registrations you entered to a file.
```

Figure 15 Continuation of figure 14



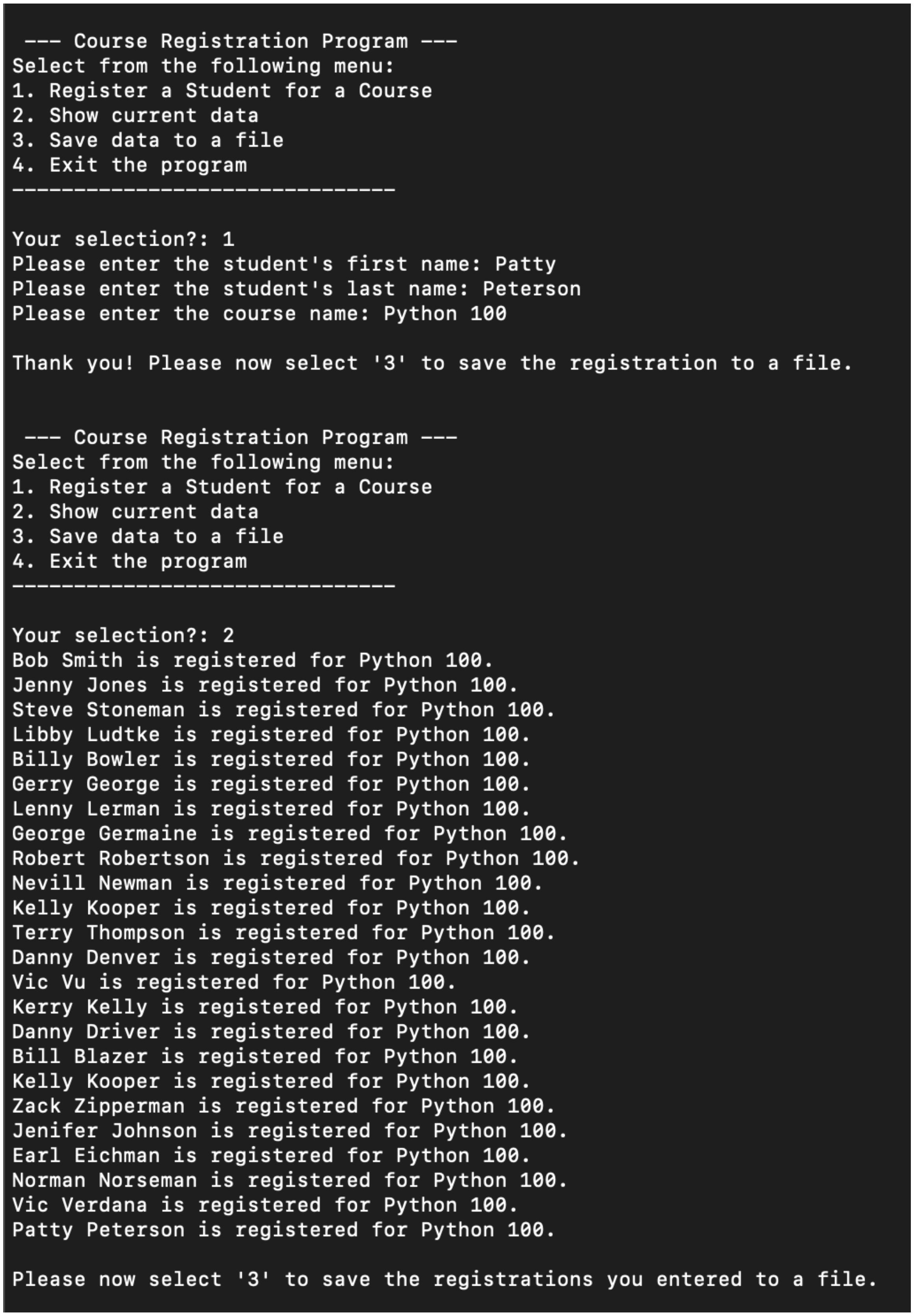
```
Your selection?: 3
Bob Smith is fully registered for Python 100.
Jenny Jones is fully registered for Python 100.
Steve Stoneman is fully registered for Python 100.
Libby Ludtke is fully registered for Python 100.
Billy Bowler is fully registered for Python 100.
Gerry George is fully registered for Python 100.
Lenny Lerman is fully registered for Python 100.
George Germaine is fully registered for Python 100.
Robert Robertson is fully registered for Python 100.
Nevill Newman is fully registered for Python 100.
Kelly Kooper is fully registered for Python 100.
Terry Thompson is fully registered for Python 100.
Danny Denver is fully registered for Python 100.
Vic Vu is fully registered for Python 100.
Kerry Kelly is fully registered for Python 100.
Danny Driver is fully registered for Python 100.
Bill Blazer is fully registered for Python 100.
Kelly Kooper is fully registered for Python 100.
Zack Zipperman is fully registered for Python 100.
Jenifer Johnson is fully registered for Python 100.
Earl Eichman is fully registered for Python 100.
Norman Norseman is fully registered for Python 100.
Vic Verdana is fully registered for Python 100.
```

Figure 16 Continuation of figure 15



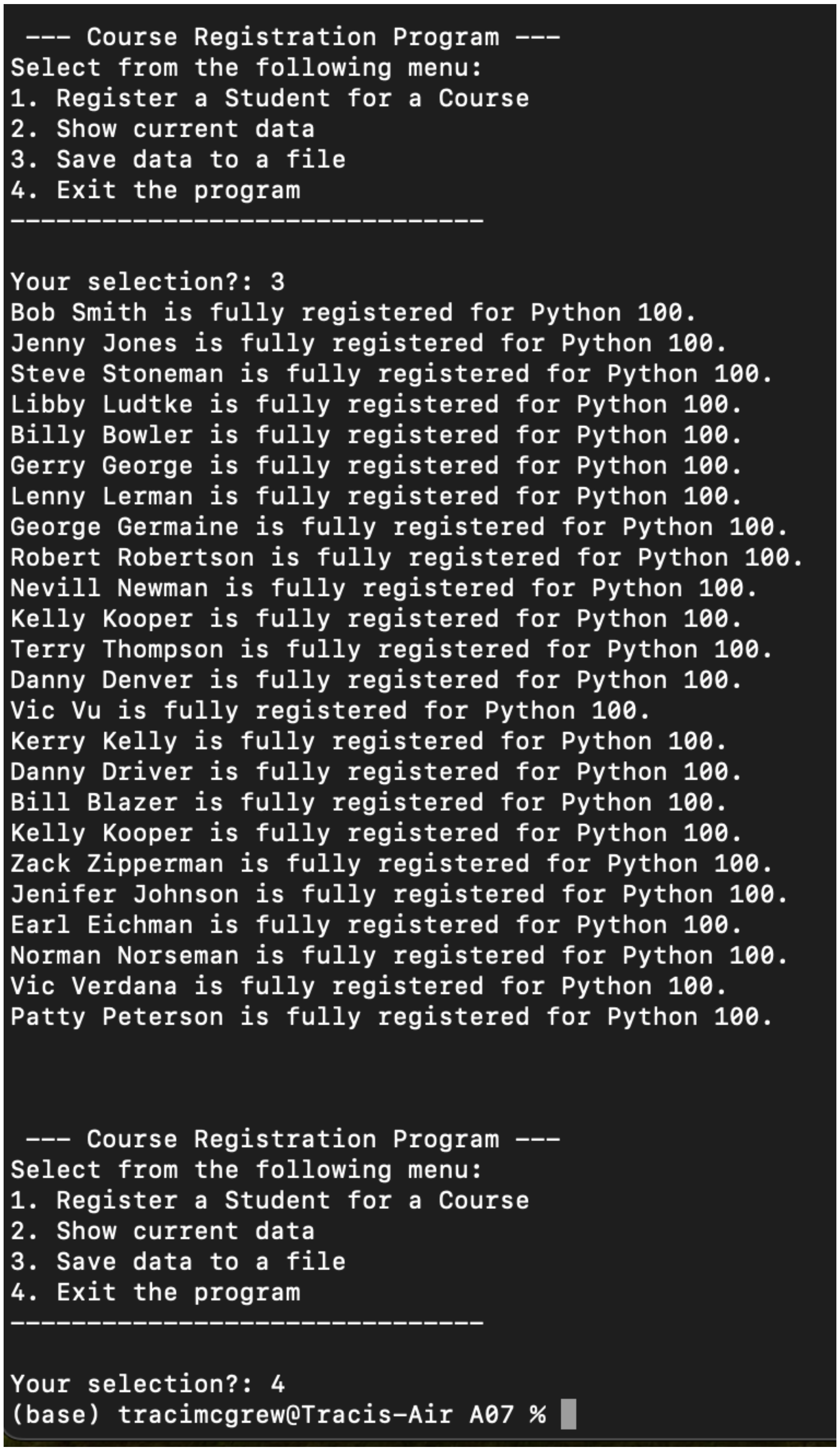
```
 --- Course Registration Program ---
Select from the following menu:
1. Register a Student for a Course
2. Show current data
3. Save data to a file
4. Exit the program
-------------------------------

Your selection?: 4

Process finished with exit code 0
```

Figure 17 Continuation of figure 16



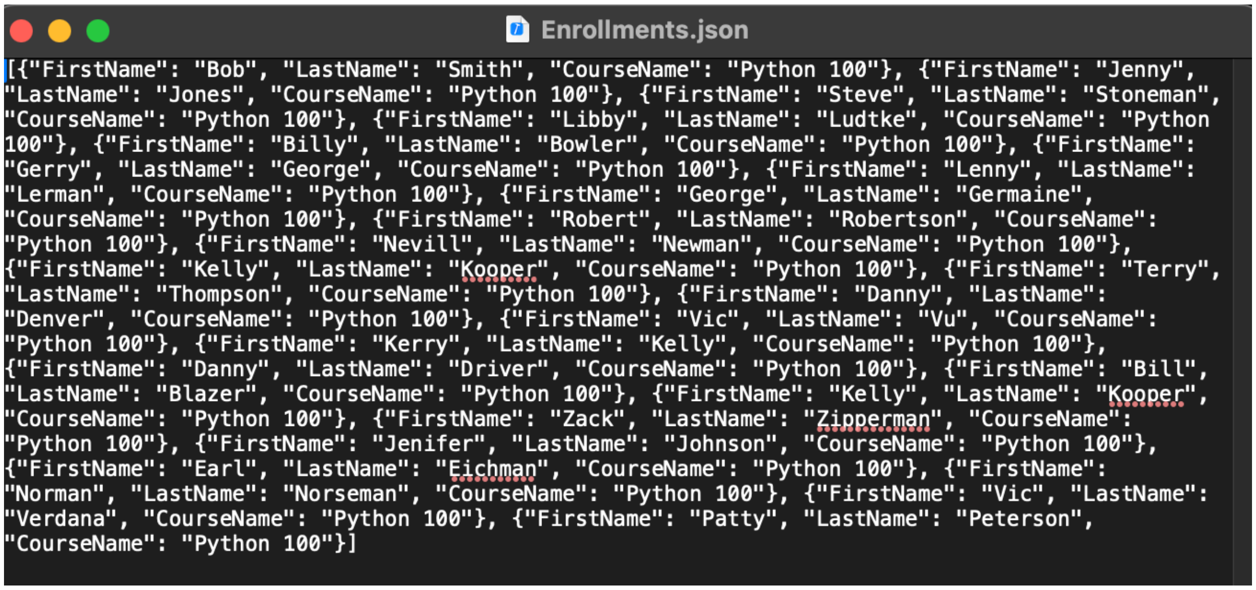
```
 --- Course Registration Program ---
Select from the following menu:
1. Register a Student for a Course
2. Show current data
3. Save data to a file
4. Exit the program
-------------------------------

Your selection?: 1
Please enter the student's first name: Patty
Please enter the student's last name: Peterson
Please enter the course name: Python 100

Thank you! Please now select '3' to save the registration to a file.


 --- 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
-------------------------------

Your selection?: 2
Bob Smith is registered for Python 100.
Jenny Jones is registered for Python 100.
Steve Stoneman is registered for Python 100.
Libby Ludtke is registered for Python 100.
Billy Bowler is registered for Python 100.
Gerry George is registered for Python 100.
Lenny Lerman is registered for Python 100.
George Germaine is registered for Python 100.
Robert Robertson is registered for Python 100.
Nevill Newman is registered for Python 100.
Kelly Kooper is registered for Python 100.
Terry Thompson is registered for Python 100.
Danny Denver is registered for Python 100.
Vic Vu is registered for Python 100.
Kerry Kelly is registered for Python 100.
Danny Driver is registered for Python 100.
Bill Blazer is registered for Python 100.
Kelly Kooper is registered for Python 100.
Zack Zipperman is registered for Python 100.
Jenifer Johnson is registered for Python 100.
Earl Eichman is registered for Python 100.
Norman Norseman is registered for Python 100.
Vic Verdana is registered for Python 100.
Patty Peterson is registered for Python 100.

Please now select '3' to save the registrations you entered to a file.
```

Figure 18 Module running in Terminal on Mac OS



```
 --- 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
-------------------------------

Your selection?: 3
Bob Smith is fully registered for Python 100.
Jenny Jones is fully registered for Python 100.
Steve Stoneman is fully registered for Python 100.
Libby Ludtke is fully registered for Python 100.
Billy Bowler is fully registered for Python 100.
Gerry George is fully registered for Python 100.
Lenny Lerman is fully registered for Python 100.
George Germaine is fully registered for Python 100.
Robert Robertson is fully registered for Python 100.
Nevill Newman is fully registered for Python 100.
Kelly Kooper is fully registered for Python 100.
Terry Thompson is fully registered for Python 100.
Danny Denver is fully registered for Python 100.
Vic Vu is fully registered for Python 100.
Kerry Kelly is fully registered for Python 100.
Danny Driver is fully registered for Python 100.
Bill Blazer is fully registered for Python 100.
Kelly Kooper is fully registered for Python 100.
Zack Zipperman is fully registered for Python 100.
Jenifer Johnson is fully registered for Python 100.
Earl Eichman is fully registered for Python 100.
Norman Norseman is fully registered for Python 100.
Vic Verdana is fully registered for Python 100.
Patty Peterson is fully registered for 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
-------------------------------

Your selection?: 4
(base) tracimcgrew@Tracis-Air A07 %
```

Figure 19 Continuation of figure 18



Enrollments.json

```
[{"FirstName": "Bob", "LastName": "Smith", "CourseName": "Python 100"}, {"FirstName": "Jenny",
"LastName": "Jones", "CourseName": "Python 100"}, {"FirstName": "Steve", "LastName": "Stoneman",
"CourseName": "Python 100"}, {"FirstName": "Libby", "LastName": "Ludtke", "CourseName": "Python
100"}, {"FirstName": "Billy", "LastName": "Bowler", "CourseName": "Python 100"}, {"FirstName":
"Gerry", "LastName": "George", "CourseName": "Python 100"}, {"FirstName": "Lenny", "LastName":
"Lerman", "CourseName": "Python 100"}, {"FirstName": "George", "LastName": "Germaine",
"CourseName": "Python 100"}, {"FirstName": "Robert", "LastName": "Robertson", "CourseName":
"Python 100"}, {"FirstName": "Nevill", "LastName": "Newman", "CourseName": "Python 100"},
{"FirstName": "Kelly", "LastName": "Kooper", "CourseName": "Python 100"}, {"FirstName": "Terry",
"LastName": "Thompson", "CourseName": "Python 100"}, {"FirstName": "Danny", "LastName":
"Denver", "CourseName": "Python 100"}, {"FirstName": "Vic", "LastName": "Vu", "CourseName":
"Python 100"}, {"FirstName": "Kerry", "LastName": "Kelly", "CourseName": "Python 100"},
{"FirstName": "Danny", "LastName": "Driver", "CourseName": "Python 100"}, {"FirstName": "Bill",
"LastName": "Blazer", "CourseName": "Python 100"}, {"FirstName": "Kelly", "LastName": "Kooper",
"CourseName": "Python 100"}, {"FirstName": "Zack", "LastName": "Zipperman", "CourseName":
"Python 100"}, {"FirstName": "Jenifer", "LastName": "Johnson", "CourseName": "Python 100"},
{"FirstName": "Earl", "LastName": "Eichman", "CourseName": "Python 100"}, {"FirstName":
"Norman", "LastName": "Norseman", "CourseName": "Python 100"}, {"FirstName": "Vic", "LastName":
"Verdana", "CourseName": "Python 100"}, {"FirstName": "Patty", "LastName": "Peterson",
"CourseName": "Python 100"}]
```

Figure 20 Showing the contents of the JSON file Enrollments.json

Summary

I went over some core concepts from the module, and then showed a Python program I created that was like the program in Assignment06, but builds on those concepts to also demonstrate using a set of data classes Person and Student. The code for this iteration of the module can be found in my Github repository https://github.com/tmcgrew/IntroToProg-Python-Mod07