

Keegan McGuire

8/26/2025

Foundations of Python Programming

Mod07 - Assignment07

<https://github.com/Keegan-McGuire/IntroToProg-Python-Mod07>

## Adding Parent Classes, Objects, and Constructors

In this assignment, we are continuing to add complexity to our student registration program to illustrate the power of parent classes, inheritance, and objects. To do this, we will add new lines of code to our program, as well as creating entirely new classes, Student and Person, which would handle how data was received by the program, saved to a file, and presented to the user.

### A New Approach?

Since our program would be adding new layers on top of the many existing layers we had already encountered, I wanted to try a new way of organizing my approach towards the task. Up until maybe the last Assignment or two, I had pretty much been going down the list of acceptance criteria to make sure I wrote each aspect in, but the acceptance criteria had become dense with things that wouldn't necessarily need to change, and may already be included in the starter file from previous assignments.

To that end, I started by going through the acceptance criteria and trying to color code the individual requirements. Things I thought were already established in the code I marked with a red highlighter, things that I thought needed to be added were marked in green, and things that seemed like rules that needed to be double checked as I worked through the assignment were underlined with blue. (Fig 1) I then went through the code and tried to see if I could sketch out a plan for how I would work with different sections (Fig. 2).

#### Classes:

- The program includes a class named FileProcessor.
- The program includes a class named IO.
- The program includes a class named Person.
- The program includes a class named Student.
- All classes include descriptive document strings.

Fig. 1

```
28 # TODO Create a Person Class
29 # TODO Add first_name and last_name properties to the constructor
30 # TODO Create a getter and setter for the first_name property
31 # TODO Create a getter and setter for the last_name property
32 # TODO Override the __str__() method to return Person data
33
34 # TODO Create a Student class the inherits from the Person class
35 # TODO call to the Person constructor and pass it the first_name and last_name data
36 # TODO add a assignment to the course_name property using the course_name parameter
37 # TODO add the getter for course_name
38 # TODO add the setter for course_name
39 # TODO Override the __str__() method to return the Student data
```

Fig 2

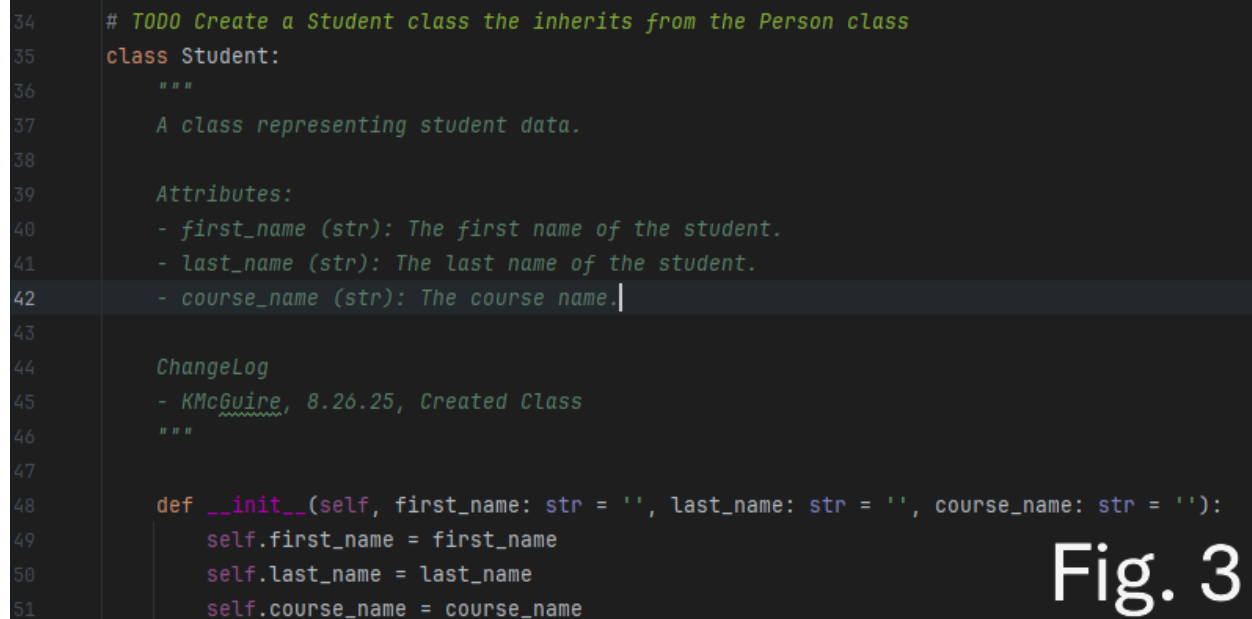
I had high hopes that my preparation would yield a more organized workflow, and maybe it did – this assignment felt like it fell into place much more fluidly than some of the previous assignments did. However, for all the scaffolding that I attempted to put into place, it ultimately seemed like there was another element that guided me much more effectively

## Figuring out what TODO

While I had spent some time sketching out an angle of approach for this assignment, I quickly fell back into what had become something of a bad habit – trying to follow the labs chronologically instead of applying all the lessons learned at the same time.

To that end, I started out by trying to build the Student class first, because that was what was featured in Lab01. I did this in spite of the planning I had done and the structure of the TODO notes in the starter file (Fig. 3).

```
34 # TODO Create a Student class the inherits from the Person class
35 class Student:
36     """
37     A class representing student data.
38
39     Attributes:
40     - first_name (str): The first name of the student.
41     - last_name (str): The last name of the student.
42     - course_name (str): The course name.
43
44     ChangeLog
45     - KMcGuire, 8.26.25, Created Class
46     """
47
48     def __init__(self, first_name: str = '', last_name: str = '', course_name: str = ''):
49         self.first_name = first_name
50         self.last_name = last_name
51         self.course_name = course_name
```



However, the further I got in this process, the more I started to feel like I was adding steps that would simply have to be refactored later, especially since it seemed like the Student class would end up inheriting aspects of the Person class. In addition, as I worked through the Student class, I noticed that if I wanted to follow the sort of pseudocode structure of the TODO instructions, it would be more helpful to start breaking the steps up, instead of doing them all in one go.

This turned out to be a critical juncture, because at this point I started paying much more attention to the placement and instructions of the TODO notes to try to achieve those goals, then cross referencing them with the assignment criteria, to make sure things like the rules I had underlined were being followed. I'm not sure if the TODO instructions were more detailed in this lab or if I had simply not been paying attention enough to them in previous labs, but this method seemed to provide a much clearer pathway to building a functional program. I started working through the Person class, breaking up the individual steps, getters, setters, and all of that into chunks of code neatly separated by the TODO instructions (Fig. 4)

```

93     # TODO call to the Person constructor and pass it the first_name and last_name data
94
95     def __init__(self, first_name: str = '', last_name: str = '', course_name: str = ''):
96         super().__init__(first_name=first_name, last_name=last_name)
97         # TODO add a assignment to the course_name property using the course_name parameter
98         self.course_name = course_name
99
100    # TODO add the getter for course_name
101
102    @property 3 usages (1 dynamic)
103    def course_name(self) -> str:
104        return self.__course_name
105
106    # TODO add the setter for course_name
107
108    @course_name.setter 2 usages (1 dynamic)
109    def course_name(self, value: str):
110        if not isinstance(value, str):
111            raise ValueError("Course name should be text.")
112        self.__course_name = value.strip()

```

Fig. 4

## Making the Code Work With the New Classes

It wasn't too long before I had my Person and Student classes added, which meant it was time to start working through the body of the code to make sure that everything cooperated with these new elements. Thankfully, I was once again aided by the helpful signposts provided by the TODO instructions, so the next thing I started working on was getting my code to convert dictionary data to student data (Fig. 5, Fig. 6).

```

147     # Convert the list of dictionary rows into a list of Student objects
148     student_objects = []
149     # TODO replace this line of code to convert dictionary data to Student data
150     student_objects = json_students

```

Fig. 5

```

149     # TODO replace this line of code to convert dictionary data to Student data
150     for row in json_students:
151         student = Student(first_name=row["FirstName"],
152                           last_name=row["LastName"],
153                           course_name=row["CourseName"])
154     student_objects.append(student)

```

Fig. 6

My next task was to make sure my code was converting student objects into dictionaries (Fig. 7). After that I needed to add code that would access the student data as opposed to the dictionary data (Fig. 8), and then I needed to replace some code so that my program would access the student objects instead of the dictionaries' (Fig. 9).

```

181     try:
182         # TODO Add code to convert Student objects into dictionaries (Done)
183         list_of_dictionary_data: list = []
184         for student in student_data:
185             student_json: dict = {
186                 "FirstName": student.first_name,
187                 "LastName": student.last_name,
188                 "CourseName": student.course_name
189             }
190             list_of_dictionary_data.append(student_json)
191
192             file = open(file_name, "w")
193             json.dump(list_of_dictionary_data, file)
194             file.close()
195             IO.output_student_and_course_names(student_data=student_data)
196         except Exception as e:
197             message = "Error: There was a problem with writing to the file.\n"
198             message += "Please check that the file is not open by another program."
199             IO.output_error_messages(message=message, error=e)
200         finally:
201             if file.closed == False:
202                 file.close()

```

Fig. 7

```

279     print("-" * 50)
280     for student in student_data:
281
282         # TODO Add code to access Student object data instead of dictionary data
283         print(f'Student {student.first_name} '
284               f'{student.last_name} is enrolled in {student.course_name}')
285
286     print("-" * 50)

```

Fig. 8

```

309     # TODO Replace this code to use a Student objects instead of a dictionary objects
310     student = {"FirstName": student_first_name,
311               "LastName": student_last_name,
312               "CourseName": course_name}
313
314     student_data.append(student)

```

Fig. 9

From there it was just a process of double checking all the assignment criteria, making sure the code was functional and getting ready to test everything.

## Testing the New Program

Once I had everything in place, it was time to start running through the list of tests included in the assignment criteria. I ran it a couple of times in both PyCharm (Fig. 10) and the command terminal (Fig. 11), then opened up the Enrollments file to make sure that things were successfully saved.

Once I had those steps taken care of, I started ticking them off on the assignment criteria (Fig. 12), and got ready to upload my program to GitHub and package everything to be turned in.

```
---- 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 Vic Vu is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Rory Gamble is enrolled in Python 100
-----
```

Fig. 10

```
Enter your menu choice number: 3
-----
Vic,Vu,Python 100
Sue,Jones,Python 100
Rory,Gamble,Python 100
Caitlyn,Hall,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.
-----
```

Fig. 11

#### Test:

- The program takes the user's input for a student's first, last name, and course name. ✓
- The program displays the user's input for a student's first, last name, and course name. ✓
- The program saves the user's input for a student's first, last name, and course name to a JSON file. (check this in PyCharm or a simple text editor like Notepad or TextEdit.) ✓
- The program allows users to enter multiple registrations (first name, last name, course name). ✓
- The program allows users to display multiple registrations (first name, last name, course name). ✓
- The program allows users to save multiple registrations to a file (first name, last name, course name). ✓
- The program runs correctly in both **PyCharm** and from the **console or terminal**. ✓

#### Source Control:

- The script file and the knowledge document are hosted on a GitHub repository.

- A link to the repository is included in the knowledge document.
- A link to the repository is included in the GitHub links forum.

Fig. 12

**NOTE:** The process and code needed to complete this assignment task is very similar to Modul07-Lab03!

## Summary

Overall, this project went much smoother than I expected. Everything worked much the same on the user's end as it did before, which was what I was expecting, and all the different components seemed to play nicely with one another. The new classes for Person and Student slotted in pretty seamlessly to the existing code, and overall I was impressed with how everything came together in the end.