Lisa Robinson

November 28, 2023 (**2nd Late Assignment of Term**)

Foundations of Programming: Python

Assignment 07

https://github.com/Lisa-jr-batista/IntroToProg-Python-Mod07

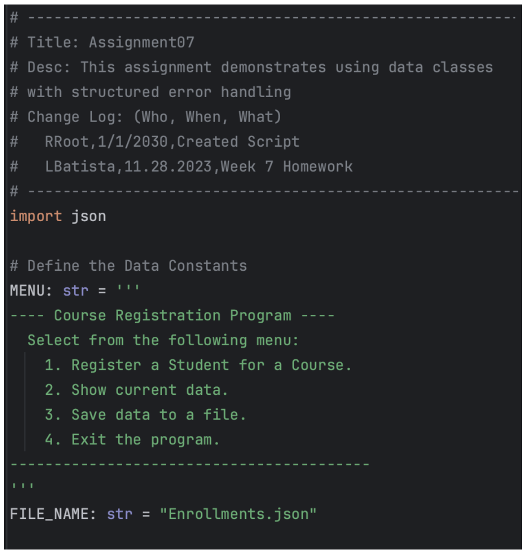
Classes and Objects

Introduction

This week the focus was on delving deeper into classes and objects. We started by revisiting statements, functions, and classes. These three components are essential to organize a more complex program and make the program easier to manage. New information I will need to internalize is that once in a class, functions become methods. Another aspect that was introduced this week are attributes (sometimes referred to as fields). An attribute is a variable that holds data specific to an object and might include characteristics of an object. This week we started assigning attributes while working with the same student input program as earlier assignments. Along with attributes we started to explore abstraction (what an object does) and encapsulation (how the object does it). Next, we finished this week by learning about constructors and setter/getter code. Finally, we started working with desktop Github.

Writing the Program

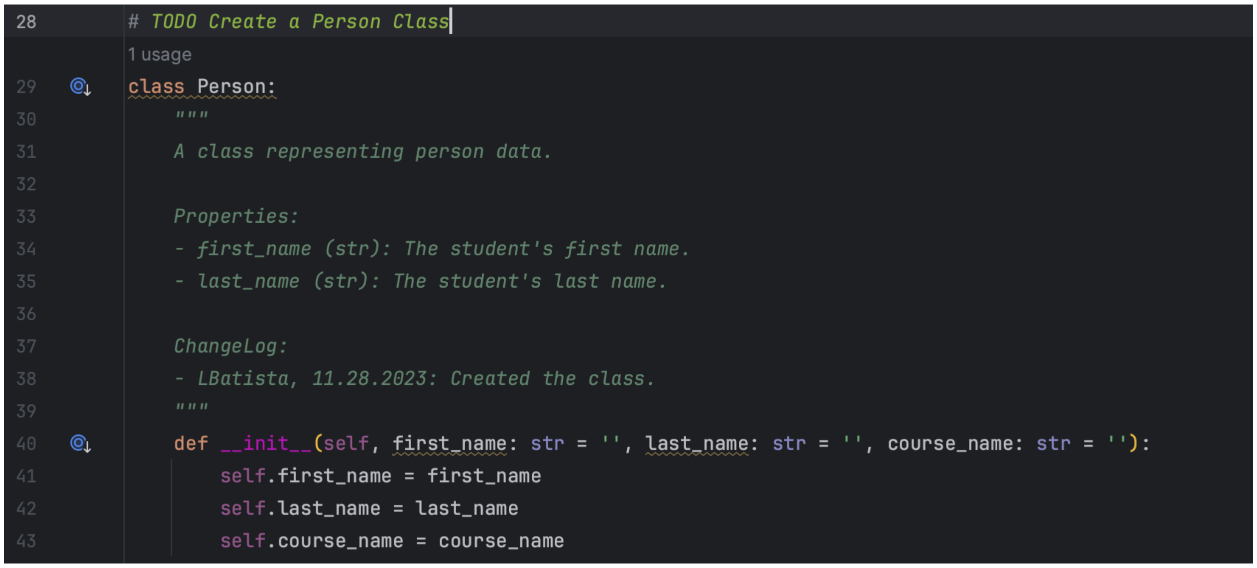
After copy and pasting the week 7 starter file, I updated my header and double checked the variables and constants. I noticed in the starter .py only JSON was imported. When I looked through the code I could not find any errors that needed the JSON Decode Error import or the TextIO so I left those off.



```
# ------------------------------------------------------------
# Title: Assignment07
# Desc: This assignment demonstrates using data classes
# with structured error handling
# Change Log: (Who, When, What)
#   RRoot,1/1/2030,Created Script
#   LBatista,11.28.2023,Week 7 Homework
# ------------------------------------------------------------
import json

# 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"
```

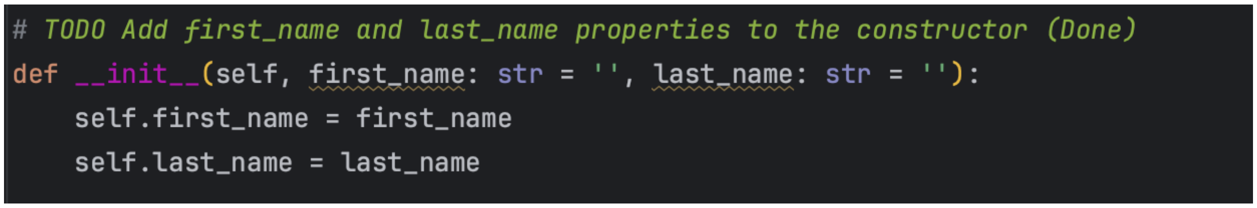
Figure 1 - Script header and constants.

For the sake of time, I decided to use the "enrollments.json" dataset I created with last week's assignment and created a copy to use for this assignment. I then started working through the assignment's "todos". Figure 2 shows the person class I created. I pulled the code from Lab 03 and edited it so the GPA spot would accurately show the course name the student was registering for. I needed to also change the float to a string value.



```python
# TODO Create a Person Class
1 usage
class Person:
    """
    A class representing person data.

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

    ChangeLog:
    - LBatista, 11.28.2023: 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
```

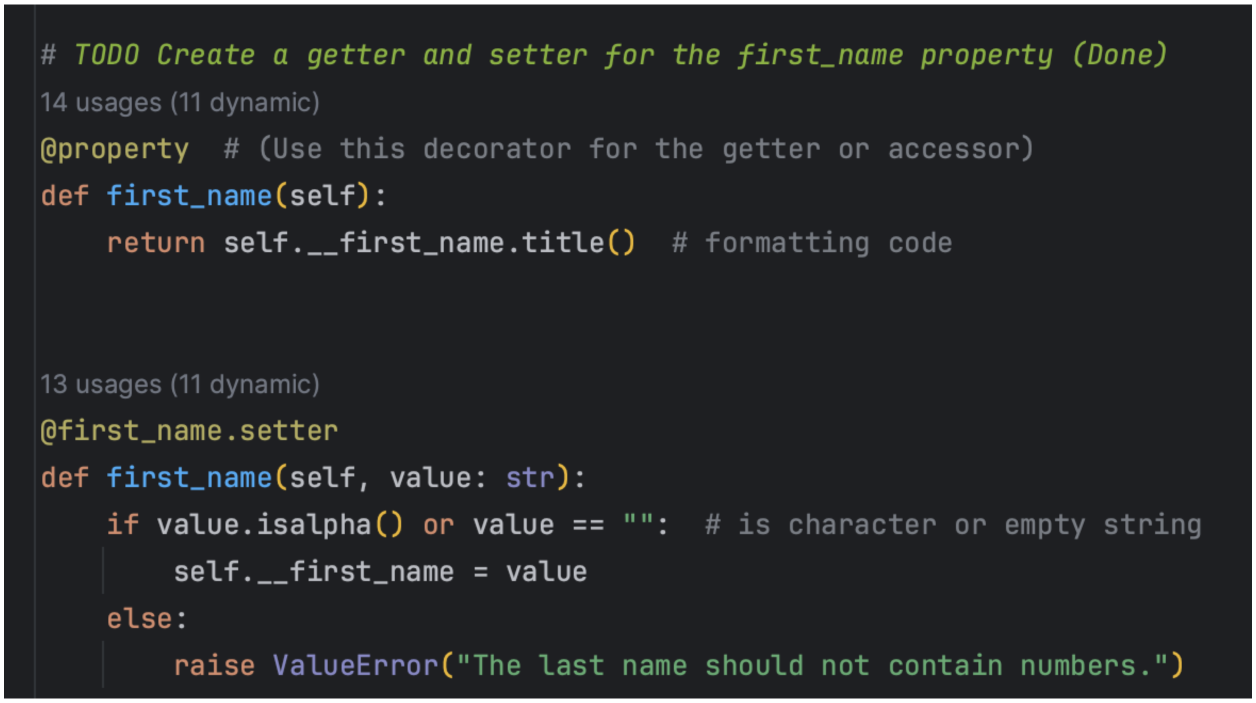
Figure 2 - Creating a "person" class



```python
# TODO Add first_name and last_name properties to the constructor (Done)
def __init__(self, first_name: str = '', last_name: str = ''):
    self.first_name = first_name
    self.last_name = last_name
```

Figure 3 - Added name properties to the constructor

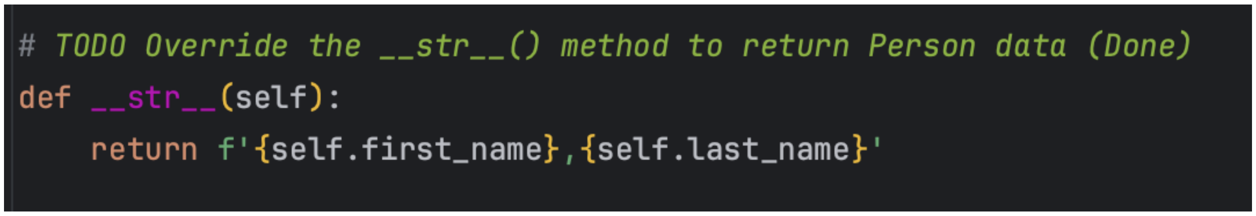
Next I added the first and last name properties to the constructor for the person class. Biggest difference here is that the course name (which will be unique to the "student" subclass) is not included. Figure 4 shows the getter and setter for the first name and I followed that with nearly identical code for the last name. Because I reused code from Lab 03, I also had to change the float value to a "str" here as well.



```
# TODO Create a getter and setter for the first_name property (Done)
14 usages (11 dynamic)
@property  # (Use this decorator for the getter or accessor)
def first_name(self):
    return self.__first_name.title()  # formatting code


13 usages (11 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 last name should not contain numbers.")
```

Figure 4 - getter and setter for the first name property.

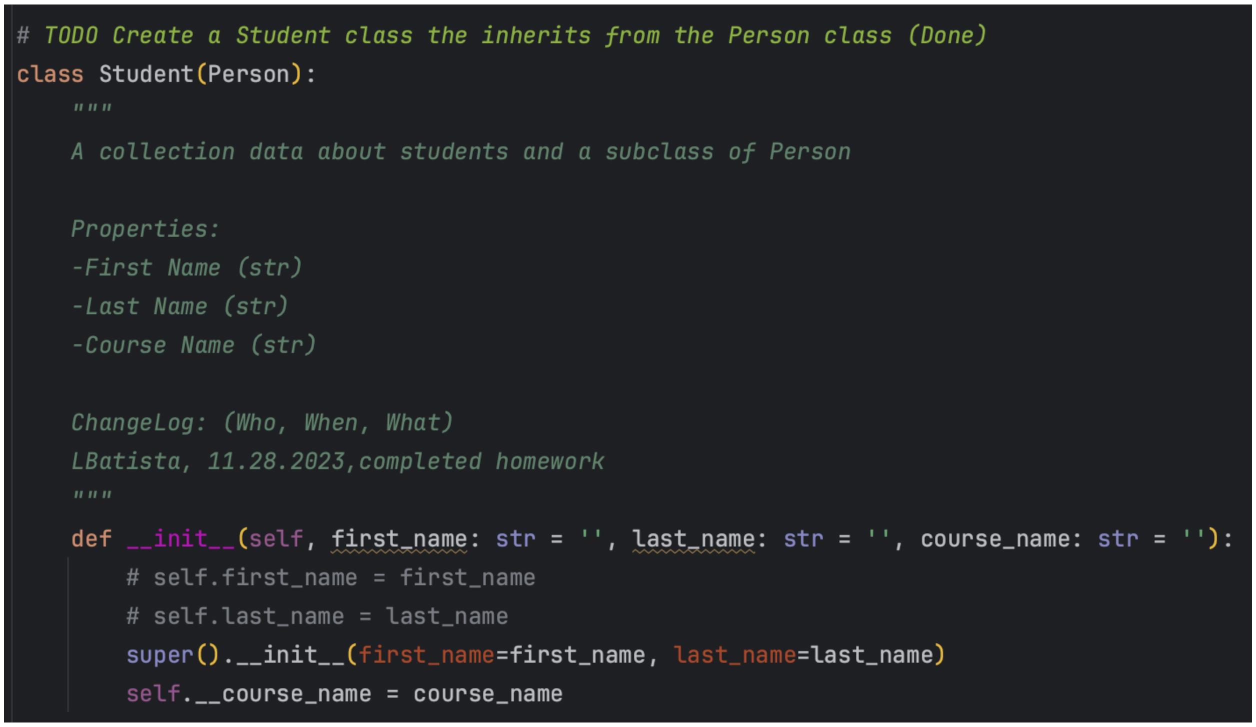
Next I had the program return the student data – this code is shown in figure 5.



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

Figure 5 - Return command

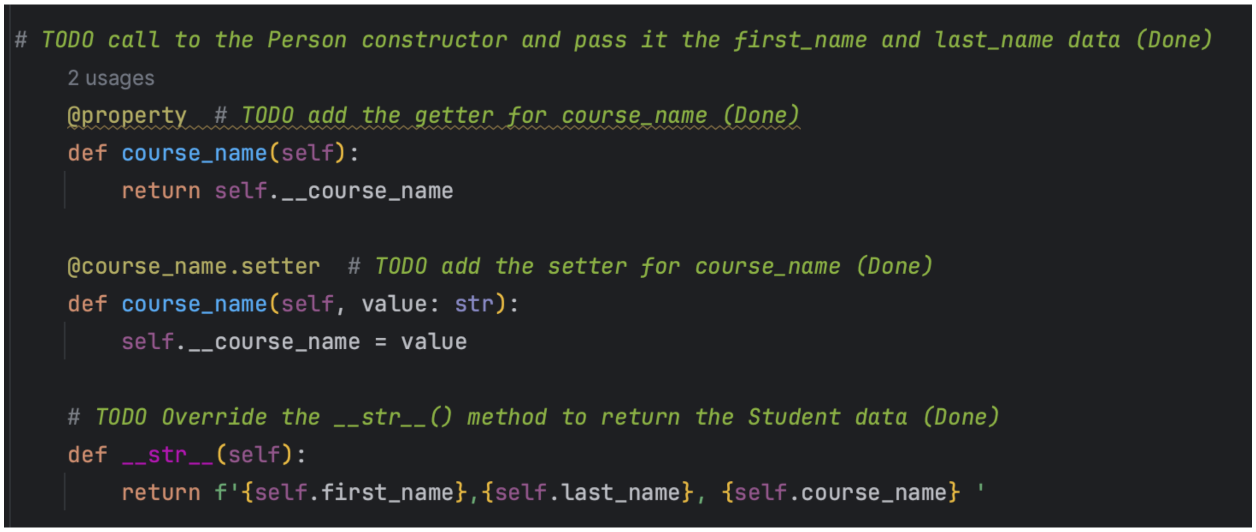
I needed to repeat the above steps for the “student” class, but to ensure it was a subclass of “Person,” I added: “class Student (Person)” when I defined the class. I also added in “super” to the attributes the subclass would inherit from the Person class. This command tells the program to use the same data as the earlier person class (Figure 6).



```python
# TODO Create a Student class the inherits from the Person class (Done)
class Student(Person):
    """
    A collection data about students and a subclass of Person

    Properties:
    -First Name (str)
    -Last Name (str)
    -Course Name (str)

    ChangeLog: (Who, When, What)
    LBatista, 11.28.2023,completed homework
    """
    def __init__(self, first_name: str = '', last_name: str = '', course_name: str = ''):
        # self.first_name = first_name
        # self.last_name = last_name
        super().__init__(first_name=first_name, last_name=last_name)
        self.__course_name = course_name
```

Figure 6 - Student subclass

Finally I completed the last few new steps and called the Person constructor and created a getter and setter for the course_name attribute. Lastly I asked the code to return the variables in a comma separated list. This code is shown in figure 7.

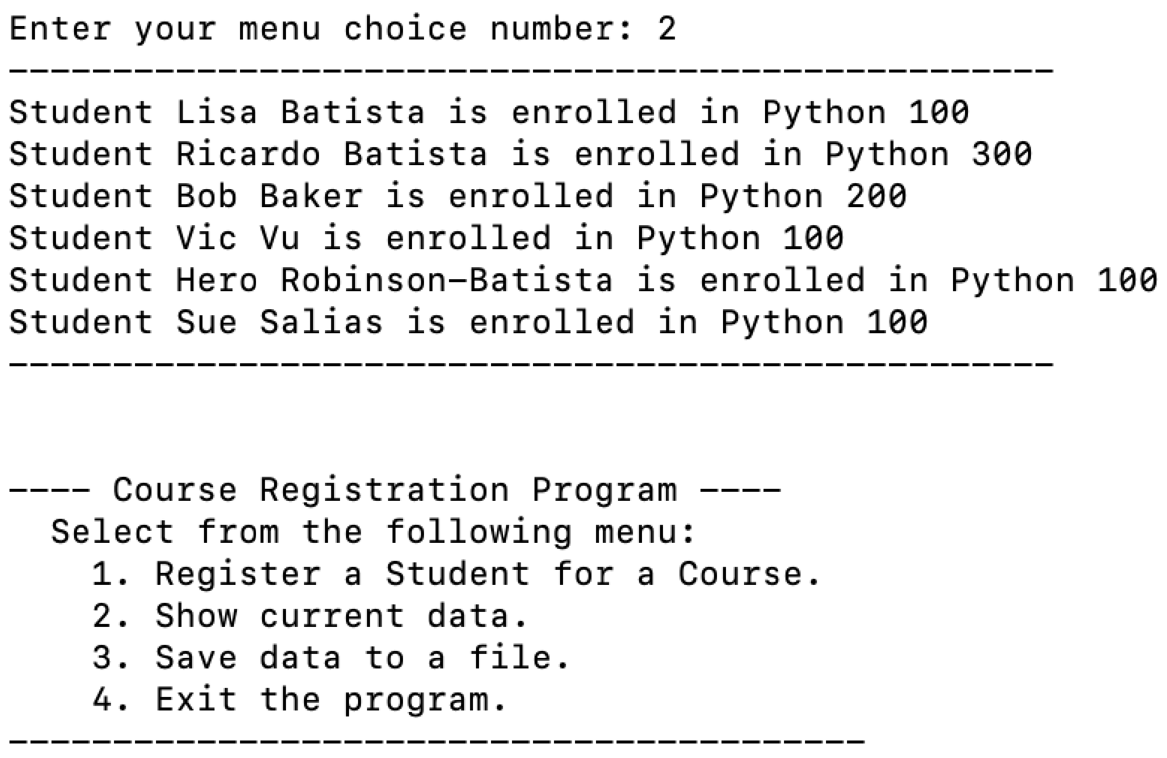


```python
# TODO call to the Person constructor and pass it the first_name and last_name data (Done)
    2 usages
    @property  # TODO add the getter for course_name (Done)
    def course_name(self):
        return self.__course_name

    @course_name.setter  # TODO add the setter for course_name (Done)
    def course_name(self, value: str):
        self.__course_name = value

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

Figure 7 - getter and setter for course_name

Next I changed the copied code in the IO class for inputting student data to allow for special characters (something I started doing last week). Even if it is small, I like being able to put that personal touch on it.

After finishing the code for the assignment, I tested it in terminal (figure 8) and uploaded my files to GIT through my desktop.



```
Enter your menu choice number: 2
--------------------------------------------------
Student Lisa Batista is enrolled in Python 100
Student Ricardo Batista is enrolled in Python 300
Student Bob Baker is enrolled in Python 200
Student Vic Vu is enrolled in Python 100
Student Hero Robinson-Batista is enrolled in Python 100
Student Sue Salias is enrolled in 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.
-----------------------------------------
```

Figure 8 - Running in Terminal

Summary

This week was a tough one–not because there was too much (I actually thought it was a pretty light week and after finishing Lab03, the homework felt like I was missing something due to how quickly I finished it)--but because of the holiday finding the time and focus to stay on top of completing the demos and labs. I am still a little lost when it comes to constructors (I feel pretty good about classes) and the particular code needed to create/define them, but after talking it over with family, they pointed out that python really is a LANGUAGE and languages have rules that do not always make sense. This is something I will need to remember and continue working on when I am frustrated about how to write a bit of code with a lot of punctuation or specific calls. In this case I think on-going practice will always help.