Jeremy Peters
November 19, 2024
IT FDN 110 B
Assignment 07
Files in GitHub

Instance Methods, Class Methods And Version Control via GitHub

Introduction

In this lesson, we have expanded upon the classes and functions structure in our code, which was introduced in the last lesson, by reorganizing some of our functions into new classes, namely the *Person* and *Student* classes, and by adding instance and class methods. Additionally, while we've been using GitHub as a storage medium for our code, during this lesson, we started using GitHub for version control by committing code changes to GitHub directly within the PyCharm (or other) IDE.

IDE Version Control via GitHub

We've been using GitHub to store our assignment files, but we hadn't yet integrated that into our IDE. This assignment had us enable GitHub version control within the IDE of our choosing. I'm using PyCharm as my IDE, which makes version control via GitHub quick to set up and easy to manage going forward, though I imagine other IDE platforms are similarly easy to use.

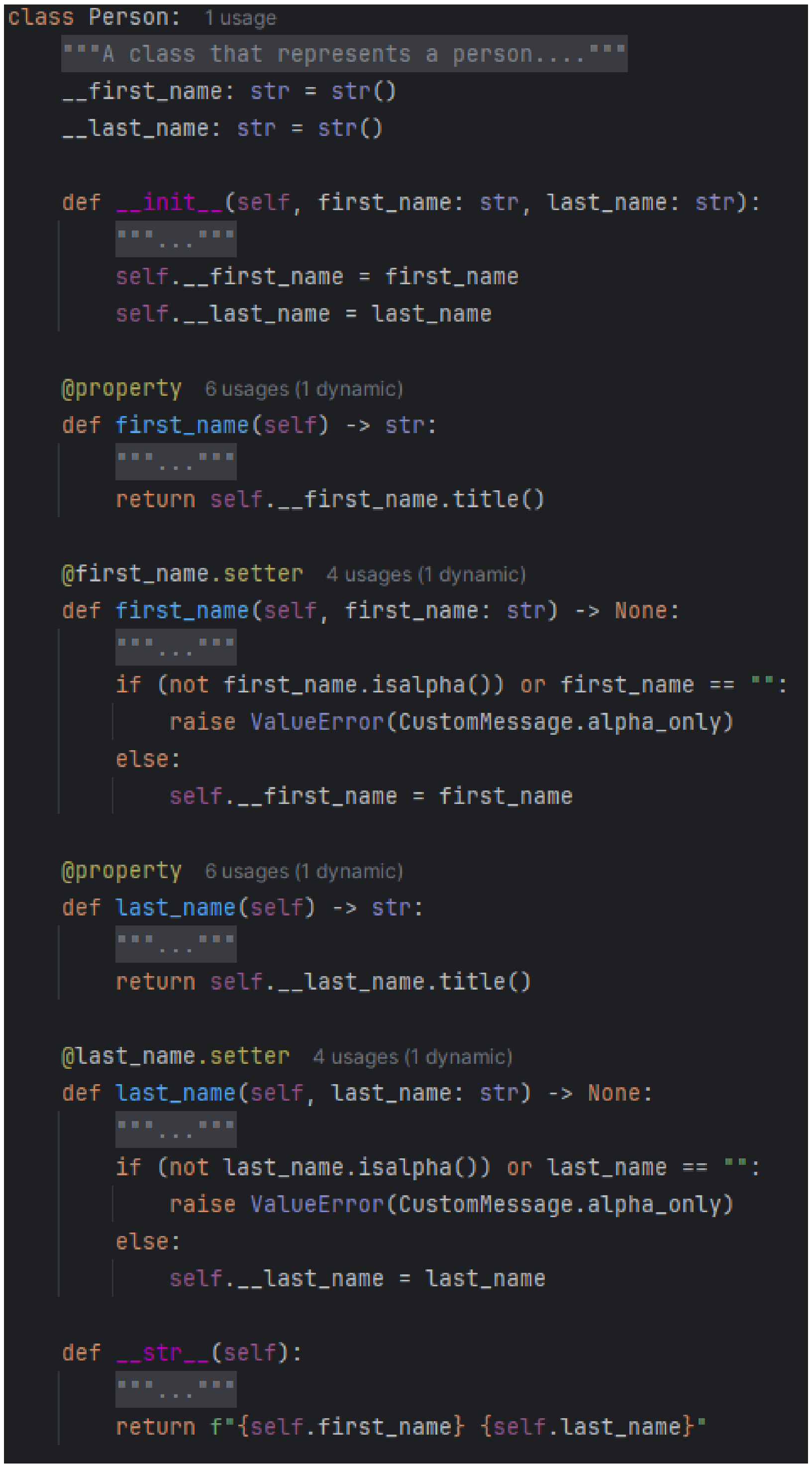
In PyCharm, I didn't have the Git menu discussed during the lesson. I first had to navigate to the *VCS* menu, select *Get from Version Control*, and then select GitHub from the option on the left. This then prompted me to log in and select a repository.

One of my favorite features of this integration is viewing the version history of the files committed. In PyCharm, if you right-click a file that has already had changes committed to GitHub, you can select *Compare with Revision…* from the *Git…* submenu and compare your local file to a previous commit version of the file. I found the changes between my initial file and the one submitted enlightening. I can see how it would be beneficial for troubleshooting or when needing to revert after multiple changes that don't work out.

Additional features of version control include forking repositories, which would be helpful for multiple people contributing to a single project without overwriting each other's changes, as well as creating branches, which are useful for freezing code for a release or if you're working on a feature and don't want to "muddy up" the main branch with changes that may or may not be included.

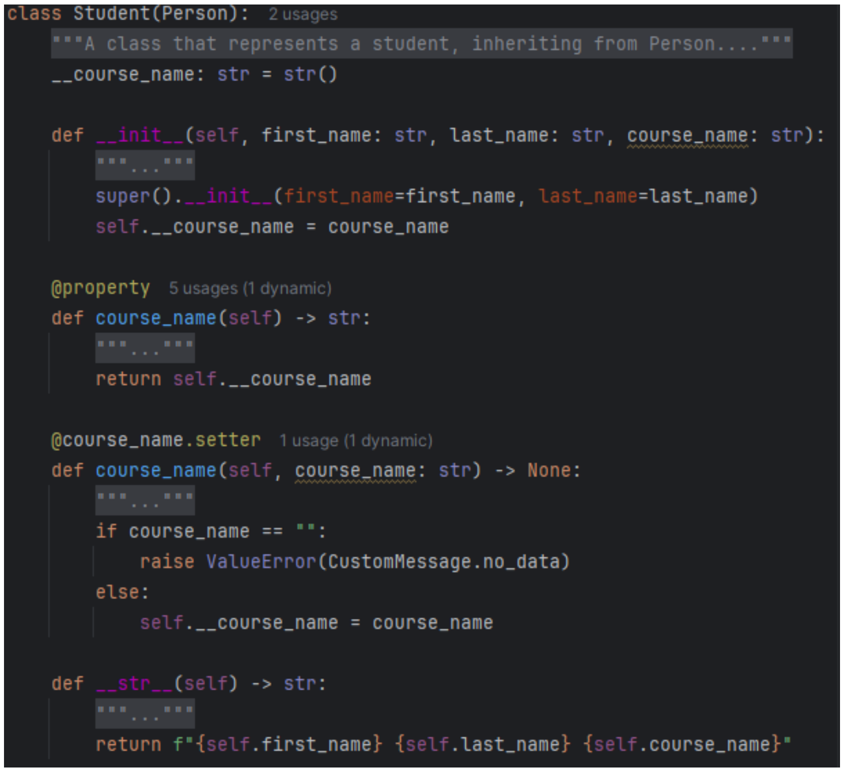
New Classes

Reading through the requirements for this assignment, what immediately stood out to me was the addition of the *Person* and *Student* classes. Using the examples from our lesson, I knew that these classes would have a Parent-Child relationship because both a "Person" and a "Student" would have a first and last name. The *Student* class inherits the *first_name* and *last_name* properties from the *Person* class.



```
class Person:  1 usage
    """A class that represents a person...."""
    __first_name: str = str()
    __last_name: str = str()

    def __init__(self, first_name: str, last_name: str):
        """..."""
        self.__first_name = first_name
        self.__last_name = last_name

    @property  6 usages (1 dynamic)
    def first_name(self) -> str:
        """..."""
        return self.__first_name.title()

    @first_name.setter  4 usages (1 dynamic)
    def first_name(self, first_name: str) -> None:
        """..."""
        if (not first_name.isalpha()) or first_name == "":
            raise ValueError(CustomMessage.alpha_only)
        else:
            self.__first_name = first_name

    @property  6 usages (1 dynamic)
    def last_name(self) -> str:
        """..."""
        return self.__last_name.title()

    @last_name.setter  4 usages (1 dynamic)
    def last_name(self, last_name: str) -> None:
        """..."""
        if (not last_name.isalpha()) or last_name == "":
            raise ValueError(CustomMessage.alpha_only)
        else:
            self.__last_name = last_name

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

To establish the parent-child hierarchy, or "inheritance," we declare **class Student(Person)** as the opening statement for the class. Here, **(Parent)** is added as a parameter to establish the relationship between the two. Then, we overload the properties by adding the *course_name* to those from the parent class and return all three when the *Student* class is called.



```
class Student(Person):  2 usages
    """A class that represents a student, inheriting from Person...."""
    __course_name: str = str()

    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  5 usages (1 dynamic)
    def course_name(self) -> str:
        """..."""
        return self.__course_name

    @course_name.setter  1 usage (1 dynamic)
    def course_name(self, course_name: str) -> None:
        """..."""
        if course_name == "":
            raise ValueError(CustomMessage.no_data)
        else:
            self.__course_name = course_name

    def __str__(self) -> str:
        """..."""
        return f"{self.first_name} {self.last_name} {self.course_name}"
```

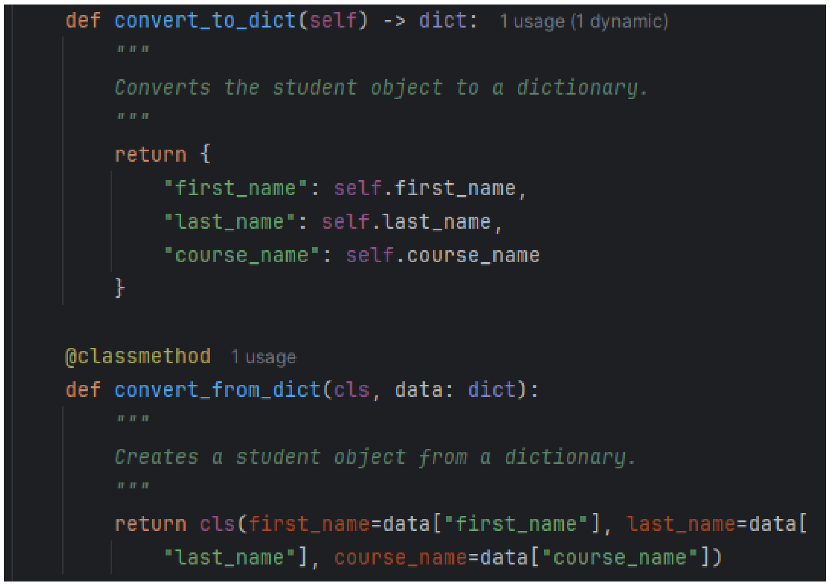
In each of these, we first initialize the class using the "dunder" **__init__()** method. The *self* keyword in the method identifies this as an "instance method" and binds the attributes with the arguments passed (for the *Person* class, we're using *first_name* and *last_name* properties, and for the *Student* class, we're using the *course_name* property, as arguments).

We then identify the properties for each class using the **@property** decorator (a.k.a. "getter"), which gets the value from the argument passed to it. This also allows me to use the *.title()* method to transform the data input so it conforms to my requirement of ensuring that names use a capital for the first letter in the first, last, and course names entered. This functionality was present in my previous assignments' *input()* statements.

We then create the property object using the **.setter** method for each property added, which binds (or "sets") the arguments passed to the class. I've also added a catch for any *ValueError* responses if the user inputs a name with anything other than alphabetic characters or nothing at all, using the **.isalpha()** method in an *if* statement. The **def __str__(self)** function tells Python that we want a string response when we **return** the property objects in our response.

(De)Serializing from/to JSON Format

I added two additional functions to my **Student** class, which were not part of the assignment, to better handle serialization and deserialization of the data objects from/to JSON format. I hit some problems when I tried to reuse the code from the last assignment. When *json.load()* and *json.dump()* attempted to iterate over the data in the **Student** variable. However, the program failed because those methods could not convert objects into dictionaries when saving and could not convert them back into **Student** objects when reading.

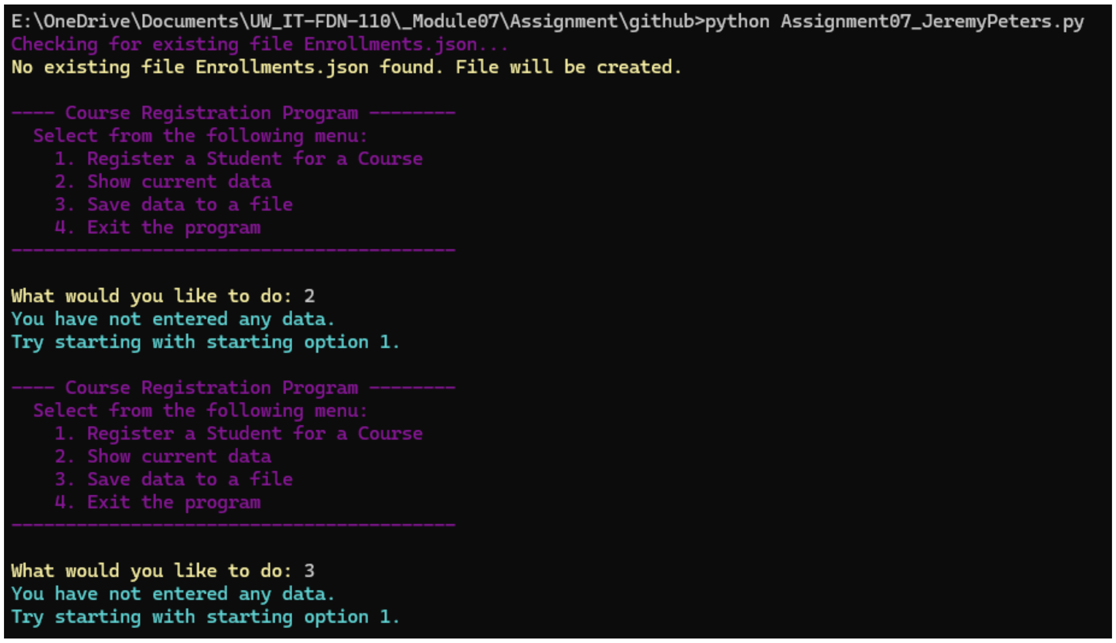


```
    def convert_to_dict(self) -> dict:  1 usage (1 dynamic)
        """
        Converts the student object to a dictionary.
        """
        return {
            "first_name": self.first_name,
            "last_name": self.last_name,
            "course_name": self.course_name
        }

    @classmethod  1 usage
    def convert_from_dict(cls, data: dict):
        """
        Creates a student object from a dictionary.
        """
        return cls(first_name=data["first_name"], last_name=data[
            "last_name"], course_name=data["course_name"])
```

The **convert_from_dict** function was added as an *@classmethod*, which wasn't required for this assignment. I needed to use it to leverage its functionality with data already present in the main program.

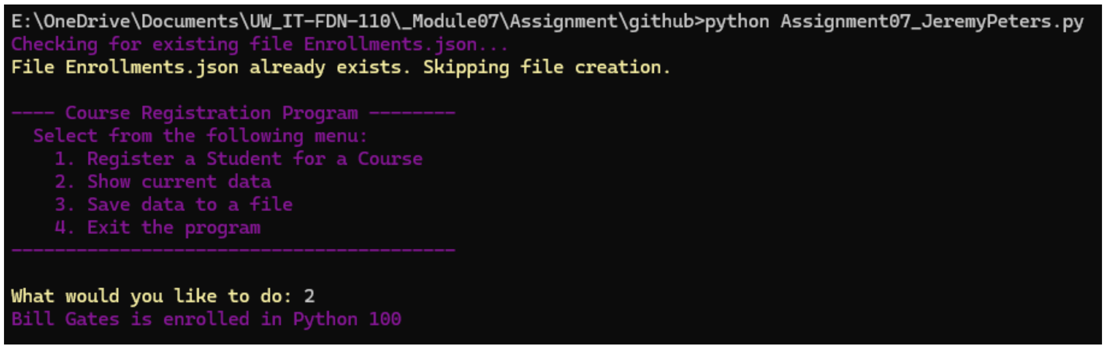
Updates to Error Handling

Based on the feedback from the last assignment submission, I've updated the functionality, and now the code only produces a friendly error if there was no file present when the program started or if the JSON file was empty (but still had the "[]" brackets for a valid JSON file). Now, if the user proceeds to option 2 or 3, and an empty JSON file is found, I direct the user to try option "1".



```
E:\OneDrive\Documents\UW_IT-FDN-110\_Module07\Assignment\github>python Assignment07_JeremyPeters.py
Checking for existing file Enrollments.json...
No existing file Enrollments.json found. File will be created.

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

What would you like to do: 2
You have not entered any data.
Try starting with starting option 1.

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

What would you like to do: 3
You have not entered any data.
Try starting with starting option 1.
```

If the JSON file contains existing and valid data and the user proceeds directly to option "2," I present the file's contents.



```
E:\OneDrive\Documents\UW_IT-FDN-110\_Module07\Assignment\github>python Assignment07_JeremyPeters.py
Checking for existing file Enrollments.json...
File Enrollments.json already exists. Skipping file creation.

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

What would you like to do: 2
Bill Gates is enrolled in Python 100
```

To accomplish this, I added a **file_status: bool = False** variable to my variable declarations, which will hold the True/False value when the program attempts to read the *Enrollments.json* file for the first time. If the file isn't present or is an empty JSON file, the bool flag is set to **True**, causing the program to return the friendly error when they choose option 2 or 3 from the menu. If the *Enrollments.json* file has data, menu options 2 and 3 operate normally, displaying or writing the data to the file, respectively.

CustomMessage Class

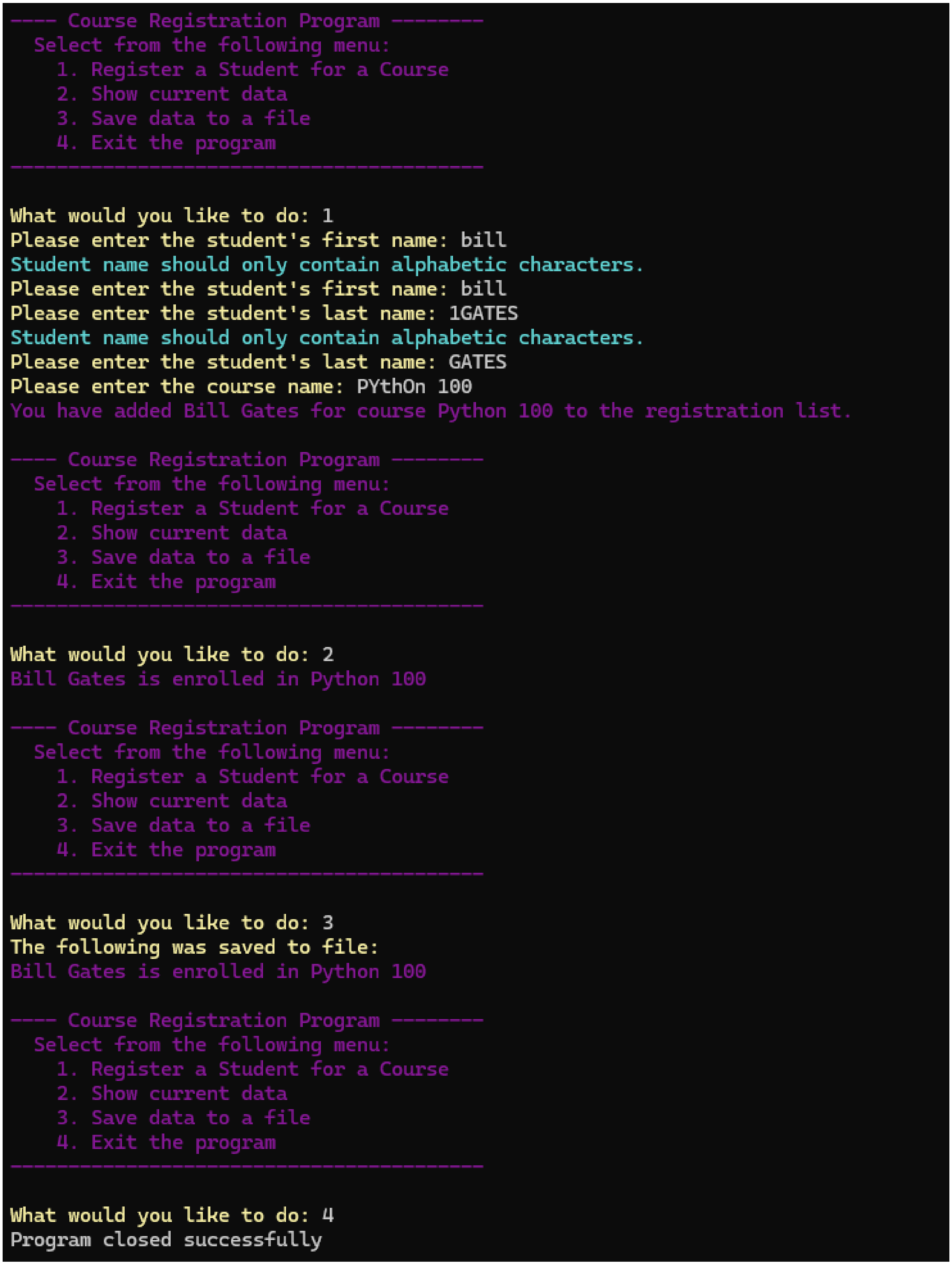
In my previous assignment files, I slowly added and iterated over some custom message variables to hold messages in a central location within the code. It was simple enough to encapsulate those into a class of their own, so I created the **CustomMessage** class and moved all my variables under it. In the future, it may make more sense to create a dedicated library file for these and import it like we do for the *json* or other libraries.

Summary

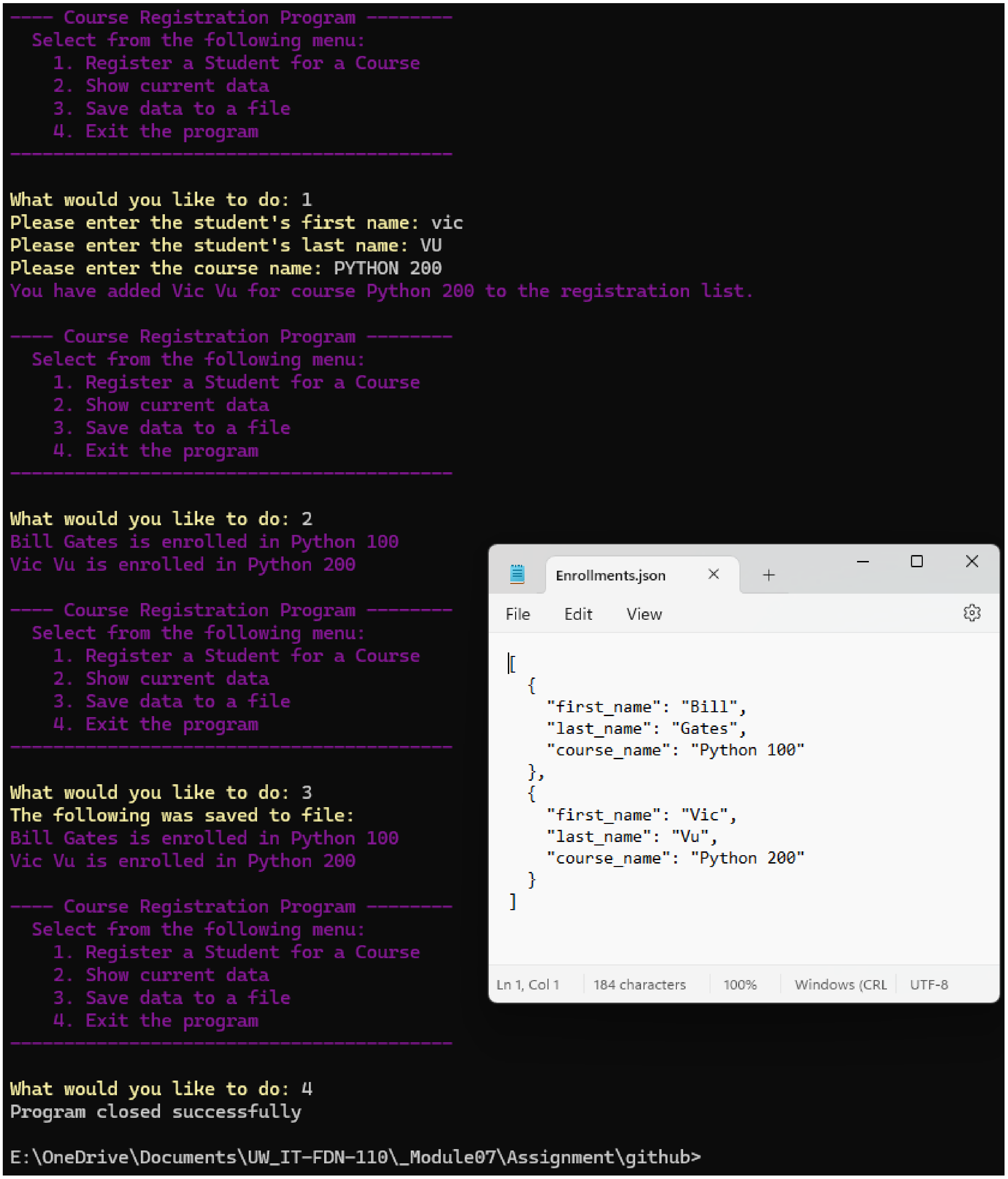
While we’re not utilizing the version control features on GitHub to its fullest, I can see its usefulness if you collaborate with other developers. Even without using many features, committing changes regularly allows us to review previous versions and revert to them if needed.

The additional classes for **Person** and **Student** continued work performed in previous assignments. However, we expanded on the concept of classes by using instance and class methods to define better the properties expected and returned by the class and its functions.

Ultimately, my program executes the tasks we initially implemented in a linear script with a few *while*, *for*, and *if* statements. The changes, while difficult to do from existing code, make adding or modifying features and fixing bugs in the code much more accessible due to the grouping and segregation of functionality.



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

What would you like to do: 1
Please enter the student's first name: bill
Student name should only contain alphabetic characters.
Please enter the student's first name: bill
Please enter the student's last name: 1GATES
Student name should only contain alphabetic characters.
Please enter the student's last name: GATES
Please enter the course name: PYthOn 100
You have added Bill Gates for course Python 100 to the registration list.

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

What would you like to do: 2
Bill Gates 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
-----------------------------------------

What would you like to do: 3
The following was saved to file:
Bill Gates 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
-----------------------------------------

What would you like to do: 4
Program closed successfully
```



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

What would you like to do: 1
Please enter the student's first name: vic
Please enter the student's last name: VU
Please enter the course name: PYTHON 200
You have added Vic Vu for course Python 200 to the registration list.

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

What would you like to do: 2
Bill Gates is enrolled in Python 100
Vic Vu is enrolled in Python 200

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

What would you like to do: 3
The following was saved to file:
Bill Gates is enrolled in Python 100
Vic Vu is enrolled in Python 200

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

What would you like to do: 4
Program closed successfully

E:\OneDrive\Documents\UW_IT-FDN-110\_Module07\Assignment\github>
```

```
[
  {
    "first_name": "Bill",
    "last_name": "Gates",
    "course_name": "Python 100"
  },
  {
    "first_name": "Vic",
    "last_name": "Vu",
    "course_name": "Python 200"
  }
]
```