Alex Novik

08/27/2025

Foundations of Programming: Python

Assignment 07

https://github.com/AlexBykov03/IntroToProg-Python-Mod07

Completing the 7th Assignment

Introduction

In this paper I will describe my process of completing Assignment 07, which focused on object-oriented programming (OOP) in Python. The main task was to create and use classes to manage data. Specifically, I needed to build a Person class and then extend it into a Student class, which would hold the student's first name, last name, and course name. At the same time, I had to change the starter file so that it no longer worked with plain dictionaries but instead worked with actual objects and their attributes. Along the way, I applied knowledge from the Module 07 notes.

Preparing

Before starting on the code, I reviewed the notes provided in the module. I thought I would review the labs and demos, but I was curious to try the assignment with just the knowledge from the notes. The example code in the notes turned out to be enough to grasp the subject and implement objects in this week's assignment. As I read the notes and the assignment document, I started playing with the code. I ended up completing the code without going through the videos and the labs.

Reading

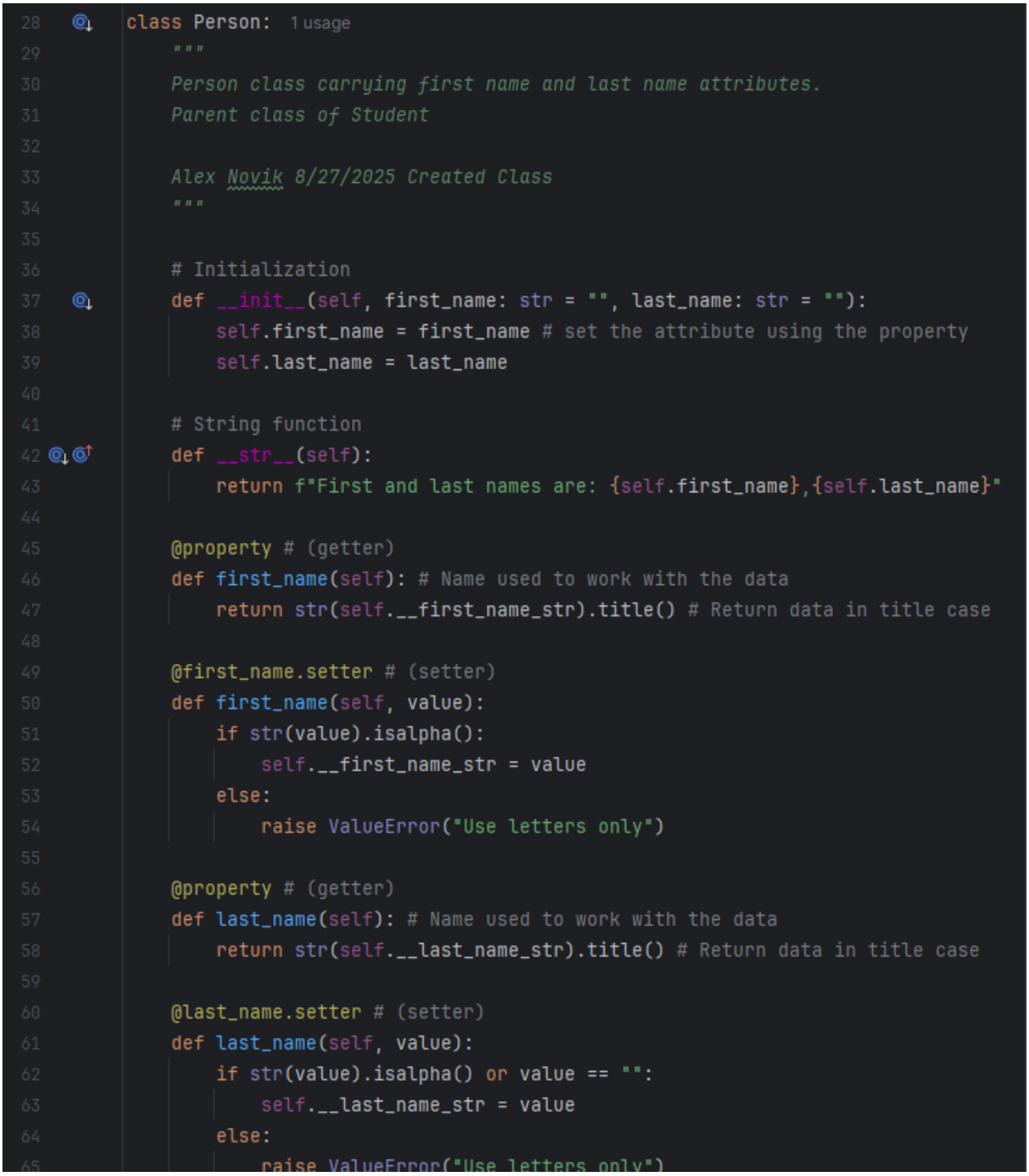
Module 07 Notes were detailed and introduced several new concepts. Here are the main points I pulled out that helped me succeed with this assignment:

- Classes vs. Functions: Unlike functions which perform tasks, classes allow us to combine both data and behavior. This makes them more powerful for managing larger programs.
- Types of Classes: The notes introduced data classes (hold information), processing classes (perform operations), and presentation classes (manage input and output). The assignment used all three.
- Constructors and Attributes: I learned that a constructor (__init__) is a special method used to initialize an object's attributes. For example, a Student object automatically starts with a first name, last name, and course name.
- Properties: These allow for controlled access to class attributes. I added simple validation in the setters to prevent empty names from being entered.
- Inheritance: One of the big ideas in this module was inheritance. The Student class inherited from the Person class, showing how code reuse works in OOP.

- Overridden Methods: Inherited methods can be replaced or extended in subclasses. This concept helped me understand how to customize class behavior.

Writing the Code

The next step was writing the program. I started with the Assignment07-Starter.py file and modified it piece by piece to match the acceptance criteria. First step was to create a Person class. This class handled first and last names. From there, I extended it into a Student class, which added a course name attribute. Last week I created a problem for myself by rushing to complete the code without checking individual functions.



```python
class Person:  1 usage
    """
    Person class carrying first name and last name attributes.
    Parent class of Student

    Alex Novik 8/27/2025 Created Class
    """

    # Initialization
    def __init__(self, first_name: str = "", last_name: str = ""):
        self.first_name = first_name # set the attribute using the property
        self.last_name = last_name

    # String function
    def __str__(self):
        return f"First and last names are: {self.first_name},{self.last_name}"

    @property # (getter)
    def first_name(self): # Name used to work with the data
        return str(self.__first_name_str).title() # Return data in title case

    @first_name.setter # (setter)
    def first_name(self, value):
        if str(value).isalpha():
            self.__first_name_str = value
        else:
            raise ValueError("Use letters only")

    @property # (getter)
    def last_name(self): # Name used to work with the data
        return str(self.__last_name_str).title() # Return data in title case

    @last_name.setter # (setter)
    def last_name(self, value):
        if str(value).isalpha() or value == "":
            self.__last_name_str = value
        else:
            raise ValueError("Use letters only")
```

Figure 1. Implementing Person Class

```python
class Student(Person):
    """
    Student class inheriting first name and last name attributes.
    Adds course name attribute
    Child of Person class

    Alex Novik 8/27/2025 Created Class
    """

    # Initialization
    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

    # String function
    def __str__(self):
        return (f"The name is {self.first_name} "
                f"{self.last_name}, and course is {self.course_name}")

    @property # (getter)  4 usages (2 dynamic)
    def course_name(self): # Name used to work with the data
        return str(self.__course_name_str).title() # Return data in title case

    @course_name.setter # (setter)  3 usages (2 dynamic)
    def course_name(self, value):
        self.__course_name_str = value
```

Figure 2. Implementing the Student Class

This week I was more careful and put more thought into how I can test each block. I tested the person and student classes by creating objects and printing their attributes. Then I started to change different functions. The first function was the “read_data_from_file” function and I made sure to test the JSON to object conversion.

```python
# TEST
test_data = FileProcessor.read_data_from_file(FILE_NAME)
for item in test_data:
    print(item.first_name, item.last_name, item.course_name)
```

Figure 3. Testing the object implementation of file data

The bigger challenge was taking the starter file, which had been written to process dictionaries, and reworking it to process Student objects instead. It felt confusing at first, but after I replaced a few lines of code and saw the program running, I started to understand why this approach was cleaner and easier to

maintain. I also had to make sure that when saving and loading from the JSON file, the data was properly converted between object attributes and dictionary rows.

Finally, I tested my code both in PyCharm and Command Prompt.

```
Enter your menu choice number: 2
-------------------------------------------------
Student Vic Vu is enrolled in Python 100
Student Sue Jones 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.
----------------------------------------


Enter your menu choice number: 1
Enter the student's first name: Peter
Enter the student's last name: Parker
Please enter the name of the course: Python

You have registered Peter Parker for Python.


---- 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 Vic Vu is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Peter Parker is enrolled in Python
-------------------------------------------------
```

Figure 4. Testing the Program in PyCharm

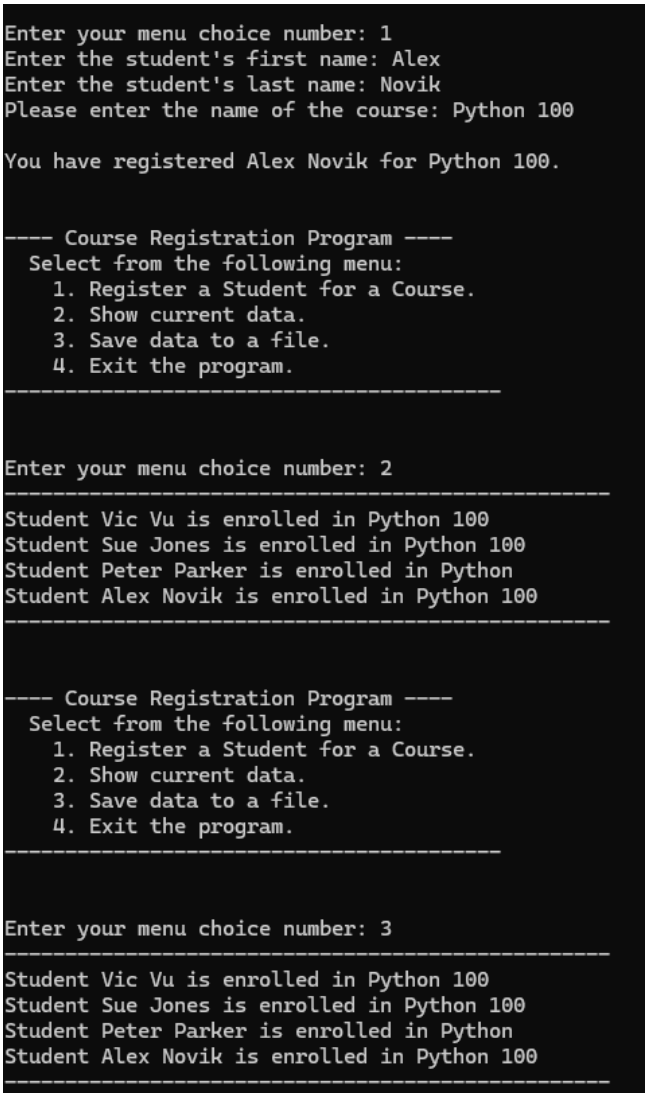


Figure 5. Testing the Program in CMD

Summary

This assignment showed the difference between functional programming and object-oriented programming. At first, it was challenging to stop thinking in terms of dictionaries and start thinking in terms of objects, but the Person and Student classes made that possible. I learned how constructors, attributes, and inheritance all fit together, and I saw how cleaner and more organized code becomes when it uses classes.