Katherine Sather

11/27/2024

Foundations of Programming: Python

Assignment No. 7

GitHub URL

Assignment No. 7

Introduction

In Module 7, I explored the foundations of object-oriented programming through Python, delving into concepts such as classes, objects, attributes, and methods. This was interesting to me as last year, I took a course to become certified in object-oriented UX, which has many similarities. Object-oriented UX (OOUX) is a method of designing user experiences rooted in the view that humans think about the world in terms of objects. Therefore, we need to approach design holistically, starting with defining objects, their relationships, and their attributes. In module 7, we worked in Python, and we did the very same thing.

This module emphasized using constructors, encapsulation, and inheritance to design robust programs. In the final assignment, I applied these lessons to a program that displays a message about a student's registration for a Python course.

Classes

In module 7 I read and thought a lot about how classes serve as blueprints, or templates, for creating objects. Within the class, you add properties and behaviors that future objects will use. It's like a recipe and cake or a factory and products.

We put functions in classes to organize our code and be more efficient. And here's another quirk: Once they are stored in classes, functions are known as "methods." The language details get a bit tough to track in Python.

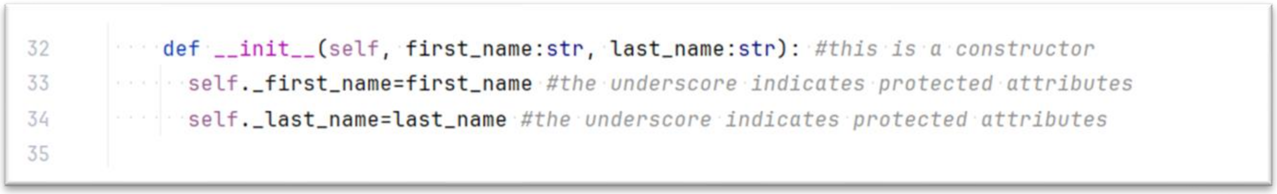
Classes can have some special components. For example, we create special methods called "constructors" to set (or initialize) an object's attribute values. Constructors are automatically called when you create a new object and are styled like this: __init__.

Attributes are just what you'd think based on the name: Class characteristics. And within the constructor method, we use the term "self" to designate content or data found only in a particular object instance.

This brings me to private attributes (see fig. 1) which is a concept that I struggled to wrap my head around. This seems like something that would be useful when dealing with security or sensitive data, such as social security numbers for example, but as I'll detail later – it created major issues with my script.

Private attributes require "getters" and "setters" to retrieve and set a private attribute value. In Python we use the @Property decorator to streamline this work.

All of these concepts together are described as abstraction and encapsulation (phew!). I need a nap. Hopefully I'm describing this correctly.



```
    def __init__(self, first_name:str, last_name:str): #this is a constructor
        self._first_name=first_name #the underscore indicates protected attributes
        self._last_name=last_name #the underscore indicates protected attributes

```

Figure 1: Using the underscore to indicate private attributes in my program. I noticed that it worked with both double and single underscores.

Magic Methods

Anything named "magic" is worthy of its own section and write-up. Python's magic methods, denoted by the double underscore, are auto-magically invoked by the Python interpreter in response to specific operations or functions. They include:

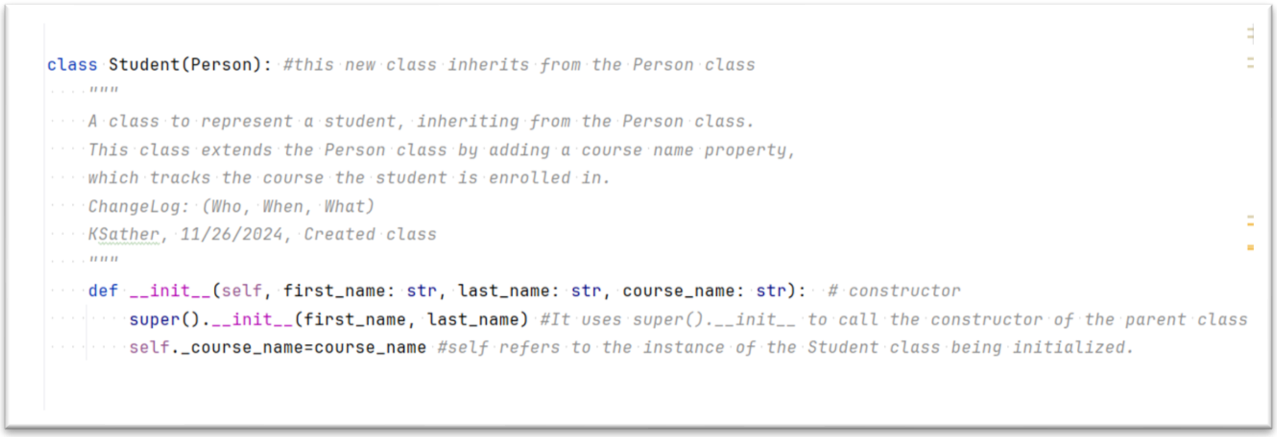
__init__(): Initializes an object's attributes when it is created.

__str__(): Returns a human-readable string representation.

.

Coding the assignment

Assignment 7 adds two new classes to the program I've been building: Person and Student. The person class represents a general person with basic attributes such as first_name and last_name. The student class, however, is a type of person that inherits attributes and methods from the Person class and adds a new course_name. This is shown in the code below (see fig. 2), where the super() function calls the constructor of the parent class. (At first I forgot to include the "super" function and it created errors.)



```
class Student(Person): #this new class inherits from the Person class
    """
    A class to represent a student, inheriting from the Person class.
    This class extends the Person class by adding a course name property,
    which tracks the course the student is enrolled in.
    ChangeLog: (Who, When, What)
    KSather, 11/26/2024, Created class
    """
    def __init__(self, first_name: str, last_name: str, course_name: str):  # constructor
        super().__init__(first_name, last_name) #It uses super().__init__ to call the constructor of the parent class
        self._course_name=course_name #self refers to the instance of the Student class being initialized.
```

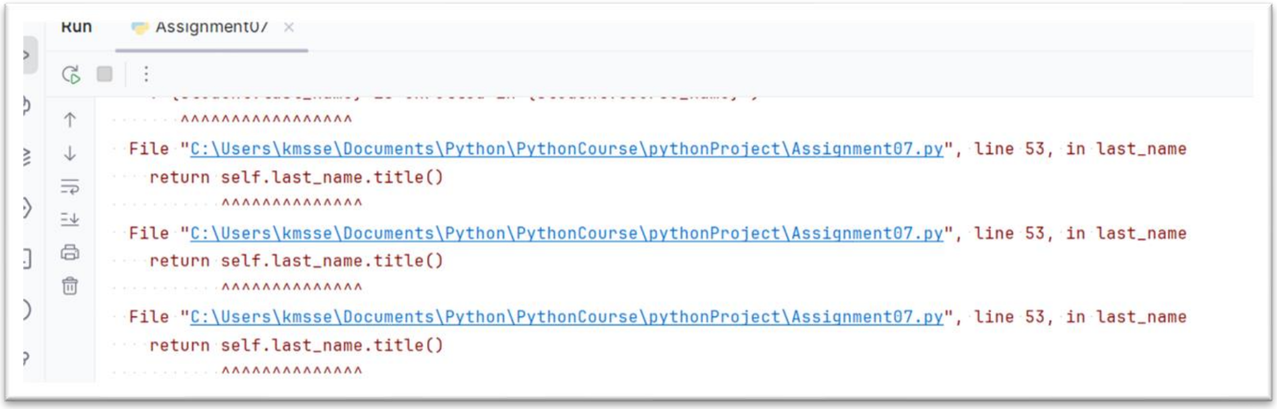
Figure 2: The Student class inherits attributes from the parent class (Person). The super() function calls the constructor of the parent class. This promotes code re-use.

Both of the new classes, Person and Student, include private attributes as a best practice, “getters” and “setters” for access control and data validation, as well as the __str__() method for formatting content.

In addition, in the Student class, the __str__ method overrides the one from Person. It uses super().__str__() to get the first_name and last_name string from the parent class and appends the course_name to it. This was another tricky bit.

After establishing the two new classes, the next step was to update the list of dictionaries the program uses to use the new student and person classes.

I followed along with the answer video to complete this code, and when I tried to run the program, I had many errors to debug (see fig. 3)



```
      ^^^^^^^^^^^^^^^^^
  File "C:\Users\kmsse\Documents\Python\PythonCourse\pythonProject\Assignment07.py", line 53, in last_name
    return self.last_name.title()
           ^^^^^^^^^^^^^^
  File "C:\Users\kmsse\Documents\Python\PythonCourse\pythonProject\Assignment07.py", line 53, in last_name
    return self.last_name.title()
           ^^^^^^^^^^^^^^
  File "C:\Users\kmsse\Documents\Python\PythonCourse\pythonProject\Assignment07.py", line 53, in last_name
    return self.last_name.title()
           ^^^^^^^^^^^^^^
```

Figure 3: I experienced many errors like this, all due to a lack of understanding around private attributes and how they’re implemented.

Ultimately, the issue was that I forgot the underscores when defining the course_name and last_name properties in their getters. Instead of referencing the private attributes of the class, the getters instead referenced their own functions, leading to errors. This is what led me to revisit the concept of private attributes and spend more time considering how they really work.

Summary

In Assignment 7, I expanded my understanding of object-oriented programming, focusing on classes, attributes, and methods. I created two new classes, Person and Student, with Student inheriting from Person and adding course_name. This demonstrated inheritance and the use of the super() function. Both classes include private attributes with "getters" and "setters" for access control, as well as the __str__() method – one of Python's "magic methods."

A key challenge was debugging errors caused by missing underscores in property definitions, which led to recursive calls.← *If I read that sentence a month ago it would have seemed like totally gibberish and nonsense in my brain, ha!* Overall, resolving this issue deepened my understanding of module 7.