Matthew Johnson

November 28, 2023

IT FDN 110 A

Assignment07

GitHub Link: https://github.com/jwehttam/IntroToProg-Python-Mod07

Python: Constructors, Property and Inheritance

Introduction

In this paper, we're focusing on three basic but key parts of Python: Constructors, Property, and Inheritance. Constructors are like the starting point, setting up our code's characters (objects) right when we make them. Property is like having a set of rules for safely changing or checking these characters. And Inheritance lets one part of the code use features from another part, kind of like how kids can inherit traits from their parents. We're going to explore how using these parts makes coding in Python easier and more organized.

Constructors

I think of constructors as the foundation of a building, needed for making an object ready to use. The '**__init__**' method acts as the constructor in Python classes. Let's look at how this method is used to create instances with specific attributes and values, laying out how the object operates within our program.

Basics of Constructors

Constructors in Python, defined through the **__init__** method, are the first step in the life cycle of an object. They are called automatically when a new instance of a class is created, initializing attributes. Constructors set the groundwork for how an object behaves throughout its life. Grasping how constructors work is key, because they kickstart objects with all the essential info they need.

Constructors in Our Code

In our course registration program, constructors are essential for creating the 'Person' and 'Student' categories. These constructors do more than just fill in basic details like names. They also set up necessary checks to ensure the information is correct. This shows how constructors help make our program strong and trustworthy by guaranteeing that each category starts off correctly. Here is an example from our program (Figure 1).

```python
class Person:
    def __init__(self, first_name: str = '', last_name: str = ''):
        self.first_name = first_name
        self.last_name = last_name
```

Figure 1: The Person Class Constructor

Property

Properties are like special tools that help you safely look at and change details, (‘attributes'), in our code. I think of them like a security system for our code’s information.

Understanding Properties

Properties use something called 'getters' and 'setters.' A getter is like asking nicely to see something, and a setter is like asking to change something, but with a rule-check first. Let’s look at how these getters and setters work and why they're important.

Understanding Properties in Our Code

In our course registration program, properties are used in the Person and Student classes. These properties ensure that the first name, last name, and course name are not only neatly packed away, but also validated before being set. For instance, the setter method for the first_name property in the Person class ensures that the name only contains letters. Using properties in our code, like in this example, ensures that the course names stay clean and correctly formatted (Figure 2).

```python
class Student(Person):
    @property
    def course_name(self):
        return self.__course_name.title()
    @course_name.setter
    def course_name(self, value: str):
        if all(char.isalnum() or char.isspace() for char in value):
            self.__course_name = value
        else:
            raise ValueError('The course name should only contain letters, numbers, 
and spaces')
```

Figure 2: Property method in the Student Class

Inheritance

Inheritance allows a new class to extend or modify the functionality of an existing class. Let's look at how inheritance promotes code reusability and hierarchy in our program. When we look at the Student class (where Student gets some features from Person), we can see how inheritance helps us make more detailed setups, but it also makes our code neat and easy to handle.

The Concept of Inheritance

Inheritance allows one class to get features from another class. I think of it like upgrading a smartphone. You get a new model with all the basic features of the old one, plus some cool new additions. This way, you don't have to build a phone from scratch; you're just improving on what's already there.. In Python, a new class can get stuff like attributes and methods from an existing class, which saves time and effort. A simple snip of code to show this would be (Figure 3).

```
class Student(Person):
    # Student class inheriting from Person
```

Figure 3: Student class inheriting from the Person class

Implementing Inheritance in Our Program

In our course registration program, the concept of inheritance is shown in how the Person and Student classes are connected. The Student class is an extension of the Person class, meaning it inherits Person's attributes and methods. This is shown in the ***__init__*** method of the Student class, where the ***super().__init__(first_name, last_name)*** line is crucial. It calls the constructor of the Person class using ***super()***, ensuring that Student inherits all the foundational attributes of a Person, like their first and last name. Additionally, Student introduces its own attribute, course_name, making it specific to the needs of a student. This shows how inheritance works in creating a structured and logical class hierarchy, and uses the ***super()*** function to integrate inherited features (Figure 4).

```
class Student(Person):
    def __init__(self, first_name: str = '', last_name: str = '', course_name: str
= ''):
        super().__init__(first_name, last_name)
        self.course_name = course_name
```

Figure 4: Implementation of Inheritance in our code with the Student class extending the Person class

Summary

So, wrapping up: We've seen how Constructors are like the starting block for our code's characters, setting them up with what they need. Then, Property methods are the rules that keep our code's info safe and in order. Inheritance is like handing down cool features from one part of the code to another, making things more organized. All these parts are really helpful in making actual programs work well. Now, our student registration program shows off how Python can handle big tasks smoothly and smartly.