Jocelyn Munoz

August 12, 2024

Foundations of Programming: Python

Assignment 07

https://github.com/Joce9322/IntroToProg-Python-Mod07

Class Components

Introduction

This document reviews working with data class components including attributes, constructors, and properties. The first section will cover the three main data class components discussed in Module 7: Classes and Objects. Each subsection reviews the modifications to our student registration program and how we incorporate parent and child classes to improve our code organization. The second section discusses the concept of inheritance and its benefits in object-oriented programming.

The program continues to build off the previous assignments, and the defined constants, classes, and functions, are similar. Please reference the previous assignments for a review of the program variables, constants, inputs/outputs, classes, and functions.

Class Components

In Module 07, we expanded from using static classes to making objects from a class through instantiation. We also delved into three main components of working with data classes: attributes, constructors, and properties. Attributes refer to the characteristics associated with an object and are also called fields. For example, in a Person class, a common set of attributes can be name, age, and gender. To instantiate the Person class, we would need to provide values for name, age, and gender for each new instance/object of the class.

A constructor is automatically called when a class is instantiated/called. Constructors are fundamental to working with classes because they create, initialize, and return a new object (RealPython, https://realpython.com/python-class-constructor/) (External Site). The .__init__() constructor method runs automatically whenever a new instance of the class is created. First, it takes the new object as the first argument, *self*. Then, it takes any arguments for the attributes provided with the __init__() constructor, such as name, age, and gender, following the example in the previous paragraph.

To account for error handling within a class, we can use private attributes in our __init__() constructor method. Private attributes cannot be changed outside of the class which minimizes potential errors occurring throughout a complicated program. Marking private attributes with an _ can also be helpful when sharing a program with multiple developers, that way it indicates the attribute is not intended to be changed. To enforce privacy in attributes, it's recommended to use property functions to get and set attribute data. Getter property functions are used to access data while setter property functions are used to add validation and error handling to the attribute.

In the following subsections, I review the code modifications and improvements we applied to our student registration program. First, we created two new classes: Person and Student. Each of these classes has new attributes passed in the __init__constructor. Then we used the get and set attribute properties to format and validate attribute data, respectively.

Attributes & Constructors

In our Person class, we have two attributes: first_name and last_name. In our Student class, we have one attribute for course_name. We added these properties to the __init__() constructor. (*Figure 1*).

```python
# TODO Create a Person Class
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:
        - JMunoz, 8.12.2024: Created the class.
    """

# 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
```

```python
class Student(Person):
    """
    A class representing student data.

    Properties:
        first_name (str): The student's first name.
        last_name (str): The student's last name.
        gpa (float): The gpa of the student.

    ChangeLog: (Who, When, What)
    JMunoz,8.12.2024,Created Class, added properties and private attributes, and moved
        first_name and last_name into a parent class
    """

# TODO call to the Person constructor and pass it the first_name and last_name data (Done)
# TODO add a assignment to the course_name property using the course_name parameter (Done)
    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
```

Figure 1. Code adding first_name and last_name attributes to the Person class and a course_name attribute in the Student class.

Properties

The getter properties for first_name and last_name were formatted with the title() method which returns a string with the first character in every word as upper case. Using the isalpha() method, first_name and last_name setter properties include error handling for the user to only include alphabet letters (*Figure 2*).

```python
# TODO Create a getter and setter for the first_name property (Done)
    @property
    def first_name(self):
        return self.__first_name.title()  # formatting code
        '''
        Returns the first_name as a title
        :return: The first name, properly formatted
        Change Log
        -JMunoz, 8.12.2024: Created getter for the first_name property
        '''
    @first_name.setter
    def first_name(self, value:str):
        '''
        Sets the first name while doing validations
        :param value: The value to set
        :return: None
        '''
        if value.isalpha() or value == "":  # is character or empty string
            self.__first_name = value
        else:
            raise ValueError("The first name should not contain numbers.")
```

```python
# TODO add the getter for course_name (Done)
    @property
    def course_name(self):
        '''
        Returns the course name
        :return: course name
        '''
        return self.__course_name

# TODO add the setter for course_name (Done)
    @course_name.setter
    def course_name(self, value:str):
        '''
        Sets the course name
        :param value: The value to set
        :return: None
        '''
        self.__course_name = value
```

Figure 2. "Getter" and "setter" code for first_name and course_name properties in the Person and Student class, respectively.

Inheritance

Inheritance allows us to create a class that shares a set of properties and methods from an existing parent class (GeeksforGeeks, https://www.geeksforgeeks.org/inheritance-in-python/) (External Site). Inheritance is beneficial for object-oriented programming because it allows us to model classes based on real-world relationships (i.e. Person & Student). Inheritance also provides efficient and reusable code and minimizes maintenance when errors occur due to the simplified organization and structure of parent and child classes.

The syntax to use for simple inheritance is show below:

Class ParentClassName:

{body}

Class DerivedClass (ParentClassName):

{body}

Our Student class inherits first_name and last_name properties from the Person class by using the super() method (*Figure 3*).

```python
# TODO Create a Student class the inherits from the Person class (Done)
class Student(Person):
    """
    A class representing student data.

    Properties:
        first_name (str): The student's first name.
        last_name (str): The student's last name.
        gpa (float): The gpa of the student.

    ChangeLog: (Who, When, What)
    JMunoz,8.12.2024,Created Class, added properties and private attributes, and moved
        first_name and last_name into a parent class
    """

# TODO call to the Person constructor and pass it the first_name and last_name data (Done)
# TODO add a assignment to the course_name property using the course_name parameter (Done)
    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
```

Figure 3. Code demonstrating the super() method to inherit first_name and last_name from the Person class.

Summary

In this assignment, we introduced two new classes to improve the readability of our code and adhere to object-oriented programming standards. Rather than working with static classes, we created instances of classes through objects. We assigned specific attributes to the constructors of each class and used getter and setter properties to format and validate each property. We also introduced the concept of inheritance to derive data from one class into another.

Citations

GeeksforGeeks. “Inheritance in Python.” GeeksforGeeks, 25 July 2024, www.geeksforgeeks.org/inheritance-in python.

Python, Real. Python Class Constructors: Control Your Object Instantiation. 25 Sept. 2023, realpython.com/python class-constructor.