Kelly Choi
Date: February 27, 2024
Title: Assignment 07
GitHub: https://github.com/kchoi78/IntroToProg-Python-Mod07

Classes and Objects

Introduction

In this lesson we expanded upon the three fundamental components of a Python program: statements, functions, and classes. Statements perform actions such as presenting data. Statements can be organized into functions where the function can be defined and called to perform a set of statements. Functions are grouped into classes, which can help in readability and repeatability of the code. Finally, Objects can be created from classes; objects can store data in their own memory location, allowing users to work from the same class. A helpful analogy is that a class is like a cookie cutter, whereas objects are individual cookies made from the cookie cutter. In this assignment I will be breaking down the changes I made to the previous program and explaining what each part entails.

Classes and Objects

```
# 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.

    Change Log:
    - Kelly, 2/26/24: created class
    """
        # * this creates a new class called Person that implicitly inherits code from Python's object class.
```

(Figure 1: creating Person class)
In Figure 1, I created a new class called `Person`.

```python
# TODO Add first_name and last_name properties to the constructor
    def __init__(self, first_name: str = '', last_name: str = ''):
        self.first_name = first_name
        self.last_name = last_name

        # * this adds a constructor to set the initial values for the first and last name data.
```

(Figure 2: creating constructors)

In Figure 2, a constructor is created to set the necessary initial data. Constructors are a special method that are called automatically when an object instance of a class is created. It is helpful to define the initial data so that the attributes, `first_name` and `last_name`, are not defaulted to undesirable values. The "`self`" keyword helps distinguish that this is data found in an object instance, and not directly from the class. The code in Figure 2 is loaded into memory just once, but it can have multiple object instances of the `Person` class, where each object instance represents a copy of the above code block.

```python
# TODO Create a getter and setter for the first_name property
    @property
    def first_name(self):
        return self.__first_name.title()

    @first_name.setter
    def first_name(self, value: str):
        if value.isalpha() or value == "":
            self.__first_name = value
        else:
            raise ValueError("The first name should not contain numbers.")

# TODO Create a getter and setter for the last_name property
    @property
    def last_name(self):
        return self.__last_name.title()

    @last_name.setter
    def last_name(self, value: str):
        if value.isalpha() or value == "":
            self.__last_name = value
        else:
            raise ValueError("The last name should not contain numbers.")

        # * Now we have the first and last name properties in the Person class.
```

(Figure 3: creating properties)

In Figure 3, properties are created for `first_name` and `last_name`. Properties are functions designed to manage attribute data. Getters, also known as Accessors, allow the user to access the data. Setters, also known as Mutators, allow the users to add data validation and error handling. The Getter-Setter pair only works when the setter decorator and function name match the name of the getter.

```python
# TODO Override the __str__() method to return Person data
    def __str__(self):
        return f'{self.first_name},{self.last_name}'
    # * now print() or str() will return the object's data, instead of the default which is the memory address
```

(Figure 4: Overriding built-in methods)

Because the `Person` class implicitly inherits certain built-in methods from the "object" class (the topmost class in Python), overriding methods can be used to return the object's data instead of the default which is the memory address.

```python
# TODO Create a Student class that inherits from the Person class
class Student(Person):  # * adding (Person) will inherit code from the Person class,
    # but additional code is needed to link the constructors
    """
    A collection data about students

    ChangeLog: (Who, When, What)
    RRoot,1.1.2030,Created Class
    RRoot,1.3.2030,Added properties and private attributes
    """

    # TODO call to the Person constructor and pass it the first_name and last_name data
    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
        # * above method connects the first and last name to the Person class, so we no longer need the
        # first_name and last_name attributes in the Student constructor.
```

(Figure 5: Creating a Student class that inherits from Person class)

Now that the `Person` class was created, another class called `Student` is created, inheriting code from the `Person` class. This is denoted by the `Person` encased in parenthesis after `Student`. The `Student` class will include the `course_name` parameter. Instead of defining the `first_name` and `last_name` constructors again in the `Student` class, a method called `super()` is used to connect the `first_name` and `last_name` to the constructors in the `Person` class.

Conclusion

The rest of the program remains unchanged from the previous assignment. I personally find these new concepts to still be quite confusing, but I hope that with practice these will become more familiar. I believe that the main purpose behind these new concepts is to successfully create object instances from classes and work with them.