Python Class Attributes

Summary: in this tutorial, you'll learn about the Python class attributes and when to use them appropriately.

Introduction to class attributes

Let's start with a simple `Circle` class (https://www.pythontutorial.net/python-oop/python-class/) :

```
class Circle:
    def __init__(self, radius):
        self.pi = 3.14159
        self.radius = radius

    def area(self):
        return self.pi * self.radius**2

    def circumference(self):
        return 2*self.pi * self.radius
```

The `Circle` class has two attributes `pi` and `radius`. It also has two methods that calculate the area and circumference of a circle.

Both `pi` and `radius` are called **instance attributes**. In other words, they belong to a specific instance of the `Circle` class. If you change the attributes of an instance, it won't affect other instances.

Besides instance attributes, Python also supports **class attributes**. The class attributes don't associate with any specific instance of the class. But they're shared by all instances of the class.

> If you've been programming in Java or C#, you'll see that class attributes are similar to the static members, but not the same.

To define a class attribute, you place it outside of the `__init__()` method. For example, the following defines `pi` as a class attribute:

```
class Circle:
    pi = 3.14159

    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return self.pi * self.radius**2

    def circumference(self):
        return 2 * self.pi * self.radius
```

After that, you can access the class attribute via instances of the class or via the class name:

```
object_name.class_attribute
class_name.class_attribute
```

In the `area()` and `circumference()` methods, we access the `pi` class attribute via the `self` variable.

Outside the `Circle` class, you can access the `pi` class attribute via an instance of the `Circle` class or directly via the `Circle` class. For example:

```
c = Circle(10)
print(c.pi)
print(Circle.pi)
```

Output:

```
3.14159
3.14159
```

How Python class attributes work

When you access an attribute via an instance of the class, Python searches for the attribute in the instance attribute list. If the instance attribute list doesn't have that attribute, Python continues looking up the attribute in the class attribute list. Python returns the value of the attribute as long as it finds the attribute in the instance attribute list or class attribute list.

However, if you access an attribute, Python directly searches for the attribute in the class attribute list.

The following example defines a `Test` class to demonstrate how Python handles instance and class attributes.

```
class Test:
    x = 10

    def __init__(self):
        self.x = 20
```

```
test = Test()
print(test.x)  # 20
print(Test.x)  # 10
```

How it works.

The `Test` class has two attributes with the same name (`x`) one is the instance attribute and the other is a class attribute.

When we access the `x` attribute via the instance of the `Test` class, it returns 20 which is the variable of the instance attribute.

However, when we access the `x` attribute via the `Test` class, it returns 10 which is the value of the `x` class attribute.

When to use Python class attributes

Class attributes are useful in some cases such as storing class constants, tracking data across all instances, and defining default values.

1) Storing class constants

Since a constant doesn't change from instance to instance of a class, it's handy to store it as a class attribute.

For example, the `Circle` class has the `pi` constant that is the same for all instances of the class. Therefore, it's a good candidate for the class attributes.

2) Tracking data across of all instances

The following adds the `circle_list` class attribute to the `Circle` class. When you create a new instance of the `Circle` class, the constructor adds the instance to the list:

```
class Circle:
    circle_list = []
    pi = 3.14159
    def __init__(self, radius):
        self.radius = radius
        # add the instance to the circle list
        self.circle_list.append(self)

    def area(self):
        return self.pi * self.radius**2

    def circumference(self):
        return 2 * self.pi * self.radius


c1 = Circle(10)
c2 = Circle(20)

print(len(Circle.circle_list))  # 2
```

3) Defining default values

Sometimes, you want to set a default value for all instances of a class. In this case, you can use a class attribute.

The following example defines a `Product` class. All the instances of the `Product` class will have a default discount specified by the `default_discount` class attribute:

```
class Product:
    default_discount = 0

    def __init__(self, price):
        self.price = price
        self.discount = Product.default_discount

    def set_discount(self, discount):
        self.discount = discount

    def net_price(self):
        return self.price * (1 - self.discount)


p1 = Product(100)
print(p1.net_price())
 # 100

p2 = Product(200)
p2.set_discount(0.05)
print(p2.net_price())
 # 190
```

Summary

- A class attribute is shared by all instances of the class. To define a class attribute, you place it outside of the `__init__()` method.
- Use `class_name.class_attribute` or `object_name.class_attribute` to access the value of the `class_attribute` .
- Use class attributes for storing class contants, track data across all instances, and setting default values for all instances of the class.