Python Readonly Property

Summary: in this tutorial, you'll learn how to define Python readonly property and how to use it to define computed properties.

Introduction to the Python readonly property

To define a readonly property (https://www.pythontutorial.net/python-oop/python-properties/) , you need to create a property with only the getter. However, it is not truly read-only because you can always access the underlying attribute and change it.

The read-only properties are useful in some cases such as for computed properties.

The following example defines a class called `Circle` that has a `radius` attribute and an `area()` method:

```
import math


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

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

And the following creates a new `Circle` object and returns its area:

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

Output:

```
314.1592653589793
```

This code works perfectly fine.

But it would be more natural that the area is a property of the `Circle` object, not a method. To make the `area()` method as a property of the `Circle` class, you can use the `@property` decorator (https://www.pythontutorial.net/python-oop/python-property-decorator/) as follows:

```
import math


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

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


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

The area is calculated from the `radius` attribute. Therefore, it's often called a calculated or computed property.

Cache calculated properties

Suppose you create a new circle object and access the area property many times. Each time, the area needs to be recalculated, which is not efficient.

To make it more performant, you need to recalculate the area of the circle only when the radius changes. If the radius doesn't change, you can reuse the previously calculated area.

To do it, you can use the caching technique:

- First, calculate the area and save it in a cache.
- Second, if the radius changes, reset the area. Otherwise, return the area directly from the cache without recalcuation.

The following defines the new `Circle` class with cached `area` property:

```
import math


class Circle:
    def __init__(self, radius):
        self._radius = radius
        self._area = None

    @property
    def radius(self):
        return self._radius

    @radius.setter
    def radius(self, value):
        if value < 0:
            raise ValueError('Radius must be positive')
```

```
        if value != self._radius:
            self._radius = value
            self._area = None

    @property
    def area(self):
        if self._area is None:
            self._area = math.pi * self.radius ** 2

        return self._area
```

How it works.

First, set the `_area` to `None` in the `__init__` method. The _area attribute is the cache that stores the calculated area.

Second, if the radius changes (in the setter), reset the `_area` to `None` .

Third, define the `area` computed property. The `area` property returns `_area` if it is not `None` . Otherwise, calculate the area, save it into the `_area` , and return it.

Summary

- Define only the getter to make a property readonly
- Do use computed property to make the property of a class more natural
- Use caching computed properties to improve the performance.