Properties

Juliana Pirolla - Revisão

Base da POO é manter a implementação interna encapsulada através do uso dos métodos. Algumas das vantagens que esse tipo de construção traz é

classe cujos objetos irão armazenar valores inteiros positivos

```
class Positive:
    def __init__(self, valor):
        if valor <= 0:
            raise ValueError('O valor precisa ser positivo')
        self.value = valor
```

Podemos ter usuários cuja utilização seja

```
bp = Positive(3)
bp.value
```

```
3
```

Note que podemos alterar o valor para algo que não é o esperado

```
bp.value = -2
```

Portanto é importante implementar o método mantendo a encapsulação fornecendo método para leitura e alteração do valor

```
class PositiveValue:
    def __init__(self, ini):
        self.set_value(ini)

    def get_value(self):
        return self._val
```

```
    def set_value(self, novo_valor):
        if novo_valor <= 0:
            raise ValueError('O valor precisa ser negativo')
        self._val = novo_valor
```

```
pv = PositiveValue(3)
```

Note que em vez de atribuir diretamente esse valor à variável "_val", o construtor chama o método "set_value" para fazer essa atribuição.

O método "set_value" verifica se o valor passado como argumento (x) é menor ou igual a zero. Se for o caso, uma exceção do tipo ValueError é levantada .

Essa abordagem de utilizar um método para atribuir valores, em vez de atribuí-los diretamente no construtor, permite a validação e o controle do valor a ser definido para o objeto, garantindo que sempre seja positivo.

Ponto negativo: a utilização é ruim

```
pv.set_value(pv.get_value() + 3) # pv += 3
pv.get_value()
```

```
6
```

Properties

Para melhorar a implementação, podemos implementar properties para acessar os objetos e

```
class Positive:
    def __init__(self, ini):
        self.set_value(ini)

    def get_value(self):
        return self._val

    def set_value(self, x):
        if x <= 0:
            raise ValueError('Value must be positive')
        self._val = x
```

```
    value = property(get_value, set_value, None, 'Value of the object (always positive)')
```

Sintaxe do properties

`property(get_value, set_value, del_value, docstring)`

- get_value: É uma função que define o comportamento da propriedade ao ser acessada. Se você não quiser fornecer um getter, pode passar None nessa posição.
- set_value: É uma função que define o comportamento da propriedade ao ser atribuída. Se você não quiser fornecer um setter, pode passar None nessa posição.
- del_value: É uma função que define o comportamento da propriedade ao ser excluída (delete). Se você não quiser fornecer um deleter, pode passar None nessa posição.
- docstring: É uma string que descreve a propriedade. Ela serve como documentação e pode ser acessada usando o atributo **doc** da propriedade.

```
class MyClass:
    def __init__(self, ini):
        self.set_value(ini)

    def get_value(self):
        return self._value

    def set_value(self, new_value):
        if new_value >= 0:
            self._value = new_value
        else:
            raise ValueError("Value must be positive.")

    value = property(get_value, set_value, None, 'Value of the object (always positive)')
```

```
p = MyClass(4)
p.value
```

```
4
```

Por que usar a função property?

Definir uma propriedade dentro de uma classe, usando a função property, permite controlar o acesso, a atribuição e até mesmo a exclusão de um atributo da classe de forma mais flexível e controlada.

Em resumo, definir uma propriedade dentro de uma classe oferece um controle mais refinado sobre a manipulação dos atributos, permite adicionar validações, transformações e documentação, e ajuda a manter uma interface consistente e coerente para os usuários da classe.

Diferença de usar a função property() e o decorador @property

- Função property: Na abordagem da função property, você *define explicitamente os métodos getter, setter e deleter dentro da classe*, e, em seguida, **usa a função property para criar a propriedade**, passando esses métodos como argumentos.

-Decorador @property: Com o decorador @property, você pode *definir diretamente um método como um getter* para a propriedade. Você simplesmente adiciona o decorador @property acima do método que você deseja expor como um getter. Para definir um setter e um deleter, você pode usar os decoradores @.setter e @.deleter, respectivamente.

Note também que ao utilizar a função property podemos já adicionar uma docstring que será usada quando fizermos o obj.value?.