Sobrecarga de operadores - Complementação

Juliana Pirolla - Revisão

Apesar de já termos implementado sobrecarga de operadores, se quisermos tratar de operadores com booleanos, precisamos antes sobrecarregar o **bool**.

```
class Teste:
    def __init__(self, nota):
        self._nota = nota
    def __bool__(self):
        return self._nota >= 5.0  # defino o que retorna true
```

```
t = [Teste(8.5), Teste(3.2)]
for n in t:
    if n:
        print('Passou')
    else:
        print('Reprovou')
```

```
Passou
Reprovou
```

Note que o método`__bool__` retorna True por default. O intuito de sobrecarregar esse operador é poder alterar tal retorno.

Se uma classe não implementar o método **bool**, o Python usará o resultado do método **len**. Se a classe não implementar ambos os métodos, os objetos serão True por padrão.

Valor absoluto

Para calcular o valor absoluto podemos sobrecarregar o método **abs** para a nossa classe.

```
class Vector2D:
    def __init__(self, x, y):
        self._vec = (x, y)

    def get_x(self):
        return self._vec[0]

    def get_y(self):
        return self._vec[1]

    def __abs__(self):
        from math import hypot
        return hypot(self._vec[0], self._vec[1])
```

```
v1 = Vector2D(3, 4)
v1.get_x(), v1.get_y(), abs(v1)
```

```
(3, 4, 5.0)
```

Note que nessa implementação, o abs fica responsável por calcular o valor da hipotenusa do triângulo formado pelos valores de x e y.

Objetos funcionais

O que são objetos funcionais??

São objetos de uma determinada classe nas quais existe um método mágio que o capacita a atuar tal como funções, ou seja, supondo que criamos o objeto `objeto1` de uma determinada classe implementada, podemos fazer `objeto1(arg1, arg2)`.

Vamos exemplifiar construindo um objeto que não recebe nenhum parametro;

```
class Incrementar:
    def __init__(self, init_val = 0):
        self._value = init_val

    def __call__(self):
        """ """
        self._value += 1
        return self._value
```

```
    def get_value(self):
        """apenas retorna o valor do objeto, não incrementa """
        return self._value
```

```
g = Incrementar()
g()
g()
```

2

```
g()
```

3

Podemos fazer um outro exemplo em que o método funcional que ao passar um valor, implementamos o seu quadrado

```
class NumeroQuadrado:
    def __init__(self):
        self._value = None

    def get_value(self):
        if self._value == None:
            print('O objeto ainda não foi inicializado com um valor numérico.')
        else:
            return self._value

    def __call__(self, value):
        self._value = value
        return self._value ** 2
```

```
meu_objeto = NumeroQuadrado()
```

```
meu_objeto = NumeroQuadrado()
meu_objeto(100)
```

10000

```
meu_objeto.get_value()
```

```
100
```