Orientação a Objetos com Python

Atributos de visibilidade e encapsulamento

WOMAKERS CODE

Encapsulamento em Python

- Em Python, todas os atributos e métodos declarados em uma classe são **públicos**, ou seja, podem ser acessados por códigos externos à classe.
- **Isso não quer dizer que eles devam ser usadas por quem instancia um objeto daquela classe.**
- Alguns atributos e métodos só existem na classe para seu funcionamento interno. Se forem alterados, podem gerar mal funcionamento e bugs no código.

```
class Quadrado:
    def __init__(self, medida):
        self.altura = medida
        self.largura = medida

    def area(self):
        return altura * largura

quadrado = Quadrado(2)
quadrado.altura = 3 # não é mais um quadrado
```

Encapsulamento em Python

- Para indicar ao usuário quais os atributos e métodos que ele não deve alterar na classe, nós utilizamos **convenções** em seus nomes.
- Existem duas convenções que são utilizadas em Python para se iniciar nomes de métodos e atributos.

Atributos e métodos que têm seus nomes iniciados com **_ (*underscore*)** são **protegidos** e não devem ser acessados pelo mundo externo a não ser que o usuário saiba exatamente o que está fazendo, ou seja, ainda pode existir algum caso de uso em que faça sentido ter acesso a esse método/atributo, mas não é o mais comum.

Atributos e métodos que têm seus nomes iniciados com **__ (*underscore* duplo)** são **privados** e não devem ser acessados pelo mundo externo de forma nenhuma.

Propriedades

Propriedades nos dão acesso a variáveis que se parecem com atributos, mas na verdade usam métodos por trás dos panos.

```python
class Quadrado:
  def __init__(self, medida):
    self.altura = medida
    self.largura = medida

  @property
  def altura(self):
    return self.__medida

  @altura.setter
  def altura(self, valor):
    # executa algum tipo de validação
    self.__medida = valor

  @property
  def largura(self):
    return self.__medida

  @largura.setter
  def largura(self, valor):
    # executa algum tipo de validação
    self.__medida = valor

  def area(self):
    return self.largura * self.altura

quadrado = Quadrado(2)
quadrado.altura = 3
quadrado.largura = 2
```

altura e **largura** são propriedades criadas com o decorator **@property**. Esses métodos são chamados **getter** porque retornam o valor da propriedade.

Propriedades

Propriedades nos dão acesso a variáveis que se parecem com atributos, mas na verdade usam métodos por trás dos panos.

```python
class Quadrado:
  def __init__(self, medida):
    self.altura = medida
    self.largura = medida

  @property
  def altura(self):
    return self.__medida

  @altura.setter
  def altura(self, valor):
    # executa algum tipo de validação
    self.__medida = valor

  @property
  def largura(self):
    return self.__medida

  @largura.setter
  def largura(self, valor):
    # executa algum tipo de validação
    self.__medida = valor

  def area(self):
    return self.largura * self.altura

quadrado = Quadrado(2)
quadrado.altura = 3
quadrado.largura = 2
```

O método **setter** altera o valor da propriedade.

Propriedades

```python
class Quadrado:
  def __init__(self, medida):
    self.altura = medida
    self.largura = medida

  @property
  def altura(self):
    return self.__medida

  @altura.setter
  def altura(self, valor):
    # executa algum tipo de validação
    self.__medida = valor

  @property
  def largura(self):
    return self.__medida

  @largura.setter
  def largura(self, valor):
    # executa algum tipo de validação
    self.__medida = valor

  def area(self):
    return self.largura * self.altura

quadrado = Quadrado(2)
quadrado.altura = 3
quadrado.largura = 2
```

As propriedades podem ser acessadas como fossem atributos comuns, mas na verdade os métodos **getter** e **setter** estão sendo chamados.