Paradigmas de Linguagem de Programação em Python

Paradigma Orientado a Objetos: Encapsulamento

Prof. Henrique Mota

Encapsulamento
Definição

- **Encapsulamento** é o que se faz quando se restringe o acesso aos dados (atributos) de uma classe (*information hiding*);

- A ideia é fazer da classe uma **cápsula**, onde seus atributos só poderão ser acessados por determinados métodos;

- Pode-se alcançar o encapsulamento de dados configurando os chamados **modificadores de visibilidade** para tornar os **atributos privados (encapsulados)** e os **métodos públicos**.

Encapsulamento
Benefícios

- **Proteção dos atributos da classe** de acessos indevidos ou acidentais.
- **Possibilidade de definir regras para alteração dos valores dos atributos** da classe

Encapsulamento
Modificadores de Visibilidade

- Especificam quais classes têm acesso aos elementos (classe, atributos, métodos e construtores) de uma determinada classe.
  - ***public***
    - Classe pode ser instanciada por qualquer outra classe.
    - Atributos e métodos são acessíveis (leitura e escrita) por objetos de qualquer classe

Encapsulamento
Modificadores de Visibilidade

- Especificam quais classes têm acesso aos elementos (classe, atributos, métodos e construtores) de uma determinada classe.
  - ***private***
    - Atributos só podem ser acessados pelos métodos dos objetos da mesma classe
    - Métodos só podem ser chamados por métodos da própria classe

Encapsulamento
Modificadores de Visibilidade

- Especificam quais classes têm acesso aos elementos (classe, atributos, métodos e construtores) de uma determinada classe.
  - ***protected***
    - Atributos e métodos são acessíveis dentro da própria classe, das subclasses e das classes que fazem parte do mesmo pacote

Encapsulamento
Modificadores de Visibilidade

- Especificam quais classes têm acesso aos elementos (classe, atributos, métodos e construtores) de uma determinada classe.
  - ***protected***
    - Atributos e métodos são acessíveis dentro da própria classe, das subclasses e das classes que fazem parte do mesmo pacote

Encapsulamento Em Python

- Diferentemente de outras linguagens como C++ e Java, **Python não tem as palavras reservadas *public, protected*** ou ***private*** para definir as regras de acesso.
- Sendo assim, podemos considerar que, por padrão, todos os atributos são públicos
- Apesar da ausência de palavras reservadas próprias para encapsulamento, Python permite utilizar esses conceitos através do uso do **"_"** (*underline*) na frente das variáveis e funções

Encapsulamento Em Python

```
class Encapsulamento:
 def __init__(self, a, b, c):
   self.public = a
   self._protected = b
   self.__private = c

 def metodo_public(self):
   print("publico")

 def _metodo_protected(self):
   print("protected")

 def __metodo_private(self):
   print("private")
```

```
encapsulamento = Encapsulamento(1, 2, 3)

print(encapsulamento.public)
print(encapsulamento._protected)
print(encapsulamento.__private)
```

```
1
2
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-2-0534a7944b40> in <module>()
     18 print(encapsulamento.public)
     19 print(encapsulamento._protected)
---> 20 print(encapsulamento.__private)

AttributeError: 'Encapsulamento' object has no attribute '__private'
```

Encapsulamento Em Python

```python
class Encapsulamento:
 def __init__(self, a, b, c):
   self.public = a
   self._protected = b
   self.__private = c

 def metodo_public(self):
   print("publico")

 def _metodo_protected(self):
   print("protected")

 def __metodo_private(self):
   print("private")
```

```python
encapsulamento = Encapsulamento(1, 2, 3)

encapsulamento.metodo_public()
encapsulamento._metodo_protected()
encapsulamento.__metodo_private()
```

```
publico
protected
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-3-f6d725ce1373> in <module>()
     22 encapsulamento.metodo_public()
     23 encapsulamento._metodo_protected()
---> 24 encapsulamento.__metodo_private()

AttributeError: 'Encapsulamento' object has no attribute '__metodo_private'
```

Encapsulamento Em Python

```python
class Encapsulamento:
 def __init__(self, a, b, c):
   self.public = a
   self._protected = b
   self.__private = c

 def metodo_public(self):
   print("publico")
   self.__metodo_private()

 def _metodo_protected(self):
   print("protected")

 def __metodo_private(self):
   print("private")
```

```python
encapsulamento = Encapsulamento(1, 2, 3)

encapsulamento.metodo_public()
encapsulamento._metodo_protected()
```

```
publico
private
protected
```

→ **Métodos e atributos privados só são acessíveis dentro da própria classe**

Encapsulamento Em Python

```python
class Encapsulamento:
 def __init__(self, a, b, c):
   self.public = a
   self._protected = b
   self.__private = c

 def get_private(self):
   return self.__private

 def metodo_public(self):
   print("publico")
   self.__metodo_private()

 def _metodo_protected(self):
   print("protected")

 def __metodo_private(self):
   print("private")
```

```python
encapsulamento = Encapsulamento(1, 2, 3)

print(encapsulamento.public)
print(encapsulamento._protected)
print(encapsulamento.get_private())

encapsulamento.metodo_public()
encapsulamento._metodo_protected()
```

```
1
2
3
publico
private
protected
```

→ Métodos e atributos privados só são acessíveis dentro da própria classe

Getters & Setters

- De maneira geral, utilizamos a nomenclatura e getters e setters para acessar e gerenciar acesso aos atributos encapsulados:
  - **get:** utilizado para acessar o valor mantido por um atributo.
  - **set:** utilizado para alterar o valor mantido por um atributo.

Encapsulamento
Exercício 1

1. Escreva a classe Departamento com as seguintes definições:
    a. Dois atributos privados;
        i. Código (inteiro) → não pode receber valores menores que zero
        ii. Nome (string) → não pode receber None ou string vazia
    b. Métodos de acesso aos atributos (get / set)
    c. Construtor que receba valores para todos os atributos
    d. Um método __str__ que retorna todos os dados do departamento em um formato String

Encapsulamento
Exercício 2

2. Escreva a classe Funcionário com as seguintes definições:
   a. Três atributos privados:
      i. matricula (inteiro) → não pode receber valores menores que zero
      ii. nome (string) → não pode receber None ou string vazia
      iii. Depto (utilize a classe Departamento implementada anteriormente) → privado não pode receber None.
   b. Métodos de acesso aos atributos (get / set).
   c. Construtor que receba valores para todos os atributos.
   d. Um método __str__ que retorna todos os dados do funcionário em um formato String.

Encapsulamento
Exercício 3

3. Implementar uma aplicação que cria um objeto do tipo **Funcionario**, a partir de dados fornecidos pelo usuário e, ao final, imprime os valores dos atributos do objeto Funcionario criado.

Obrigado!

Alguma dúvida?

Prof. Henrique Mota

profhenriquemota@gmail.com