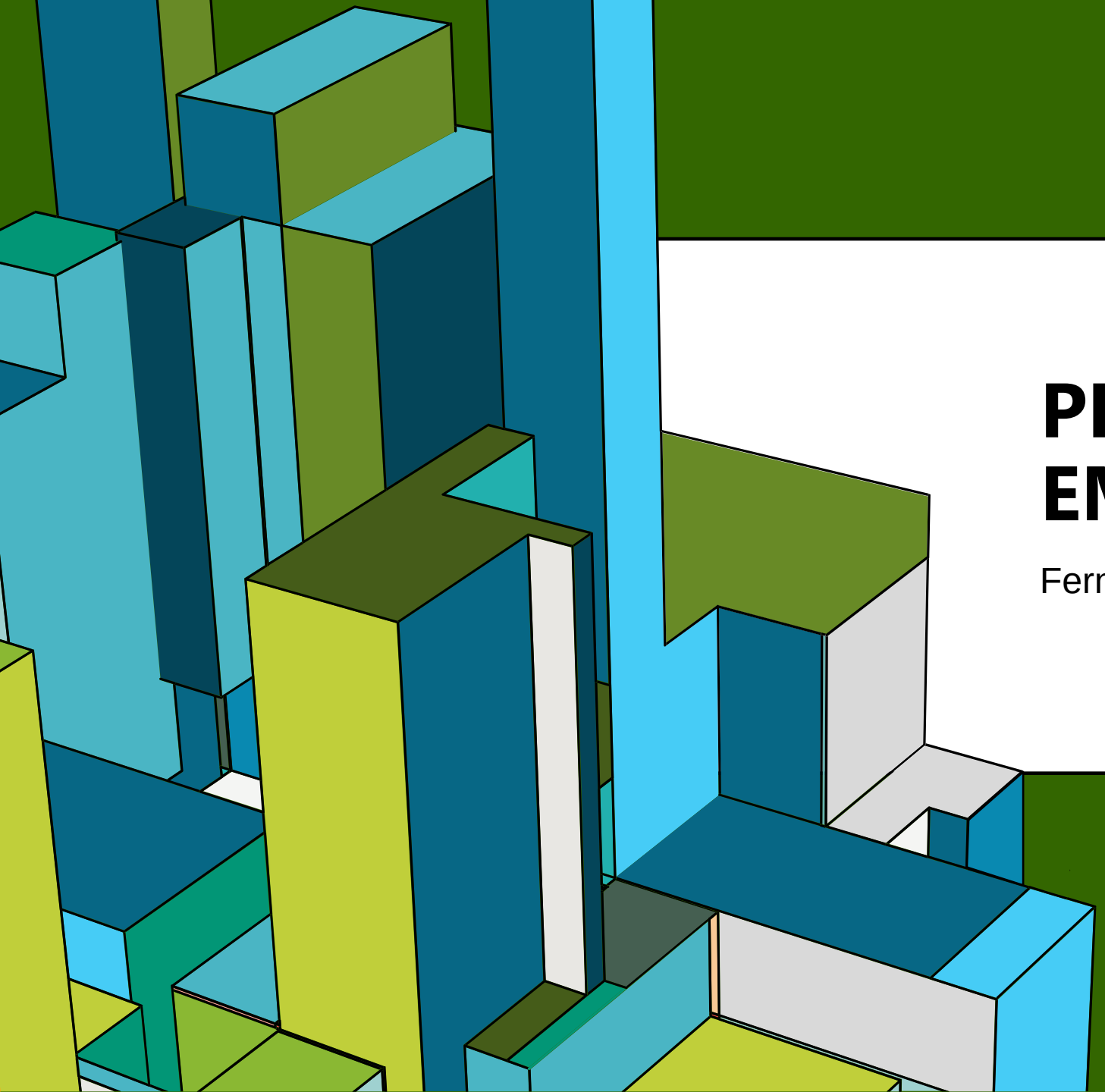


PROGRAMAÇÃO EM PYTHON

Fernando F. Santos

python

SENAI

AULA 17

27/03/2023

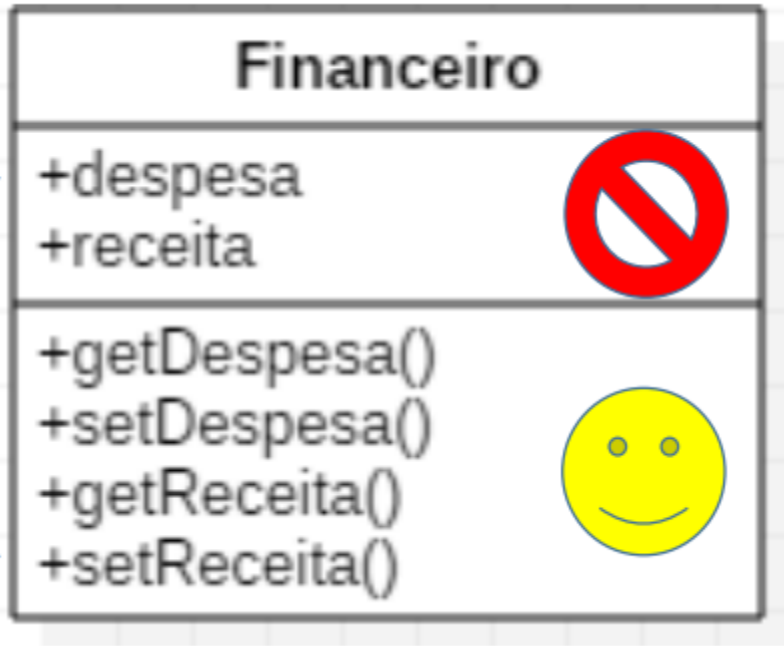
ENCAPSULAMENTO

- Encapsulamento é uma forma de fazer com que os objetos mantenham suas informações de estado privadas, tornando seu comportamento oculto para o mundo externo.
- Com encapsulamento não é possível alterar o estado interno dos objetos atuando diretamente neles.
- Para manipular as informações dos objetos, enviamos “mensagens” aos mesmos, utilizando funções como get e set
- Nas linguagens Java e C++, por exemplo, é possível definir propriedades como públicas ou privadas. As propriedades privadas não podem ser acessadas diretamente, necessitando da implementação dos métodos get e set.
- No Python o conceito de encapsulamento (ocultar dados e métodos) não é implícito, pois não existem palavras reservadas como no Java ou C++ (public, private e protected).

ENCAPSULAMENTO

No Java, quando definimos um atributo como privado (private), ele não é acessível fora da classe.

Sendo assim, são criados métodos get e set para manipular estes atributos.



ATRIBUTOS PÚBLICOS E PRIVADOS

- Variáveis de instância privadas, que são acessadas somente do interior do objeto não existem no Python.
- Porém, existe uma convenção seguida pelos programadores Python: Um nome prefixado com um underscore (por exemplo _carro) deve ser tratado como não público. Seja uma função, método ou membro de dados.

ATRIBUTOS PÚBLICOS E PRIVADOS

```
class Carro(object):

    def __init__(self, marca, modelo):
        self._marca = marca
        self._modelo = modelo

    def get_modelo(self):
        return self._modelo

    def set_modelo(self, modelo):
        self._modelo = modelo

carro = Carro("Ford", "Ranger")
print(carro._marca)
carro._marca = "Fiat"
print(carro._marca)
carro.set_modelo("Uno")
print(carro.get_modelo())
print(carro.get_modelo())
```

Apesar de ser possível acessar o "_marca" diretamente, seu nome indica que é um método "não público", sendo assim, é melhor não usá-lo diretamente.

ATRIBUTOS PÚBLICOS E PRIVADOS

- Para criar um atributo “privado” no Python, temos que nomear o atributo iniciando com dois underscores. Exemplo:

Quando definimos com dois underscores, o atributo não é Acessível externamente pelo Seu nome “__despesa”. O Python substitui seu nome por

_nomedaclasse__variável: _Financeiro__despesa.

- Na verdade, os atributos não se tornam realmente privados, porque mesmo assim, ainda é possível acessá-los externamente, mesmo que com outro nome.

EXEMPLO

```
class Carro(object):
    marca = "Ford"
    __modelo = "Focus"

    def get_modelo(self):
        return self.__modelo

    def set_modelo(self, modelo):
        self.__modelo = modelo
carro = Carro()
print(carro.marca)
```

```
carro.marca = "Fiat"
print(carro.marca)
carro.set_modelo("Uno")
print(carro.get_modelo())
carro._Carro__modelo = "Palio"
print(carro.get_modelo())
print(carro.__modelo)
AttributeError: 'Carro' object has no attribute '__modelo'
print(carro.modelo)
AttributeError: 'Carro' object has no attribute 'modelo'
```

EXEMPLO

```
# Não é correto acessar os atributos que iniciam com um
# underscore diretamente, o correto é usar os métodos get/set
carro._marca = "Fiat"
print(carro._marca)
carro.set_modelo("Uno")
print(carro.get_modelo())
print(carro.get_modelo())
```

CAMPOS “PRIVADOS” NÃO PODEM SER ACESSADOS POR UMA SUBCLASSE

```
class Mae(object):
    def __init__(self):
        self.__atributo_privado = 10

class Filha(Mae):
    def pegar_atributo_privado(self):
        return self.__atributo_privado

filha = Filha()
print(filha.__dict__)
x = filha.pegar_atributo_privado()
print(x)
```

__dic__: Retorna um dicionário com os atributos do objeto.

```
{'_Mae__atributo_privado': 10}
return self.__atributo_privado
AttributeError: 'Filha' object has no attribute '_Filha__atributo_privado'
```

```
class Mae(object):
    def __init__(self):
        self.__atributo_privado = 10

class Filha(Mae):
    def pegar_atributo_privado(self):
#         return self.__atributo_privado
        return self._Mae__atributo_privado

filha = Filha()
x = filha.pegar_atributo_privado()
print(x)
```

“Por que então a sintaxe para atributos privativos não assegura as restrições de visibilidade como deveria? A resposta mais simples é um dos lemas mais citados do Python: ‘Somos todos adultos aqui e consentimos em dar liberdade uns aos outros.’ Os programadores de Python acreditam que os benefícios da liberdade são maiores que as desvantagens de ter a cabeça fechada.”

SLATKIN, B. Python Eficaz. São Paulo: Novatec, 2016. 121 p