

Programação Orientada à Objetos: Encapsulamento



Programação Orientada à Objetos: Encapsulamento

- Consiste em separar os aspectos externos de um objeto dos detalhes internos de implementação;
- Evitar que dados específicos de uma aplicação possam ser acessados diretamente; e
- Proteger os atributos ou métodos de uma classe.

Programação Orientada à Objetos:

Encapsulamento

Tomamos como exemplo novamente a nossa classe gato:

```
class Gato:
    def __init__(self, peso, idade, nome="sem nome", raça="sem raça"):
        self.nome=nome
        self.raça=raça
        self.peso = peso
        self.idade = idade
    def mudar_nome(self, nome):
        self.nome=nome
    def engordar(self, peso):
        self.peso+=peso
    def envelhecer(self):
        self.idade +=1
```

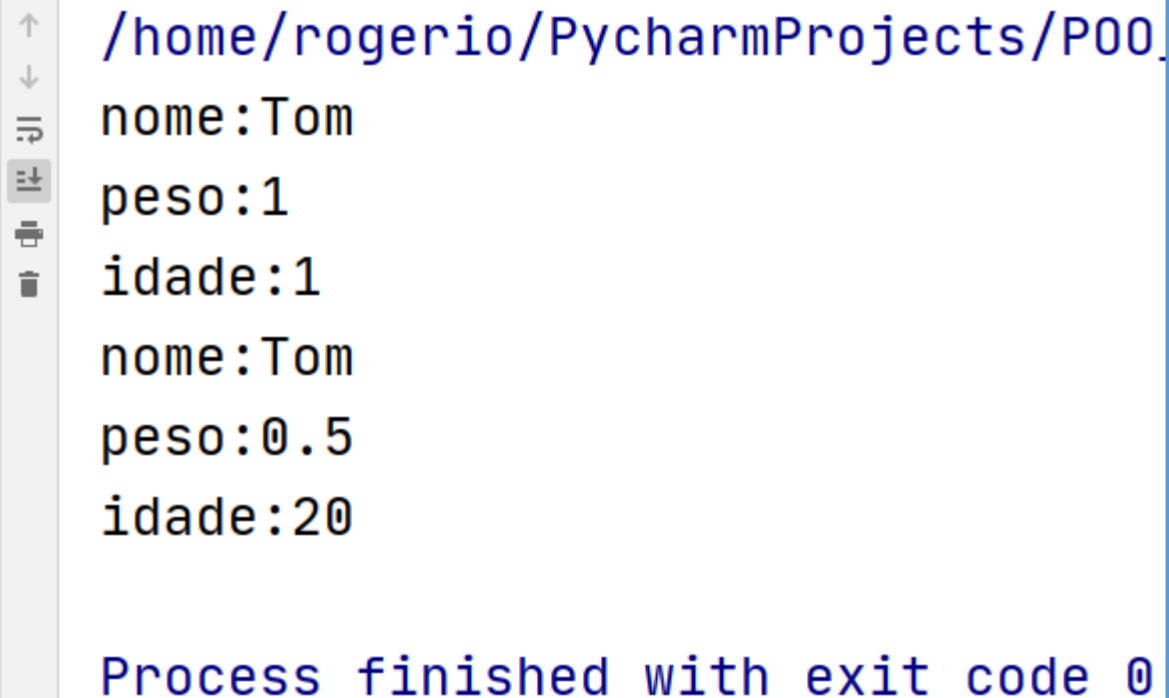
▲ 24 ✖ 17

- Podemos aumentar o peso do gato sem chamar o método engordar(...) ?
- Podemos aumentar a idade do gato sem chamar o método envelhecer(...) ?
- Um gato pode mudar de raça?

Programação Orientada à Objetos:

Encapsulamento

```
tom=Gato(1,1,nome="Tom")
print("nome:{}\npeso:{}\nidade:{}".format(tom.nome,tom.peso,tom.idade))
tom.peso=0.5
tom.idade=20
print("nome:{}\npeso:{}\nidade:{}".format(tom.nome,tom.peso,tom.idade))
```



```
/home/rogerio/PycharmProjects/P00.
nome:Tom
peso:1
idade:1
nome:Tom
peso:0.5
idade:20

Process finished with exit code 0
```

Programação Orientada à Objetos: Encapsulamento

O que vimos no slide anterior foi uma violação de encapsulamento!

Mas como evitar essa violação? Como o python implementa o encapsulamento?

Vamos por partes...

Programação Orientada à Objetos: Encapsulamento

Modificadores de acesso:

- Em Python, existem dois tipos de modificadores de acesso para atributos e métodos: Público ou Privado.
- Atributos ou métodos iniciados por dois sublinhados são “privados” e todas as outras formas são públicas.

Programação Orientada à Objetos: Encapsulamento

```
class Pessoa:
```

```
    def __init__(self, nome):
```

```
        self.__nome = nome
```

atributo privado

```
eu = Pessoa("Ivo")
```

```
print(eu.nome) ?
```

**Como obter o nome da pessoa
representada pelo objeto eu?**

```
File "/home/rogerio/PycharmProjects/P00_286/Pessoa.py",  
    print(eu.nome)  
AttributeError: 'Pessoa' object has no attribute 'nome'  
  
Process finished with exit code 1
```

Programação Orientada à Objetos: Encapsulamento

então seria assim?

```
7 print(eu.__nome)
```

Traceback (most recent call last):

File ["/home/rogerio/PycharmProjects/P00_286/Pessoa.py"](/home/rogerio/PycharmProjects/P00_286/Pessoa.py),

print(eu.__nome)

AttributeError: 'Pessoa' object has no attribute '__nome'

Process finished with exit code 1

Programação Orientada à Objetos: Encapsulamento

Agora temos um problema!

Nós encapsulamos o atributo nome porque achamos que uma pessoa não pode mudar de nome, mas como mostrar o nome dessa pessoa?

R- Para permitir o acesso atributos e métodos de forma controlada (já que eles são 'protegidos'), a prática mais comum é criar dois métodos: um que retorna o valor e um que muda o valor.

Programação Orientada à Objetos:

Encapsulamento

A convenção utilizada para esses métodos na maioria das linguagens orientadas a objeto é utilizar as palavras: get/set antes do nome do atributo encapsulado.

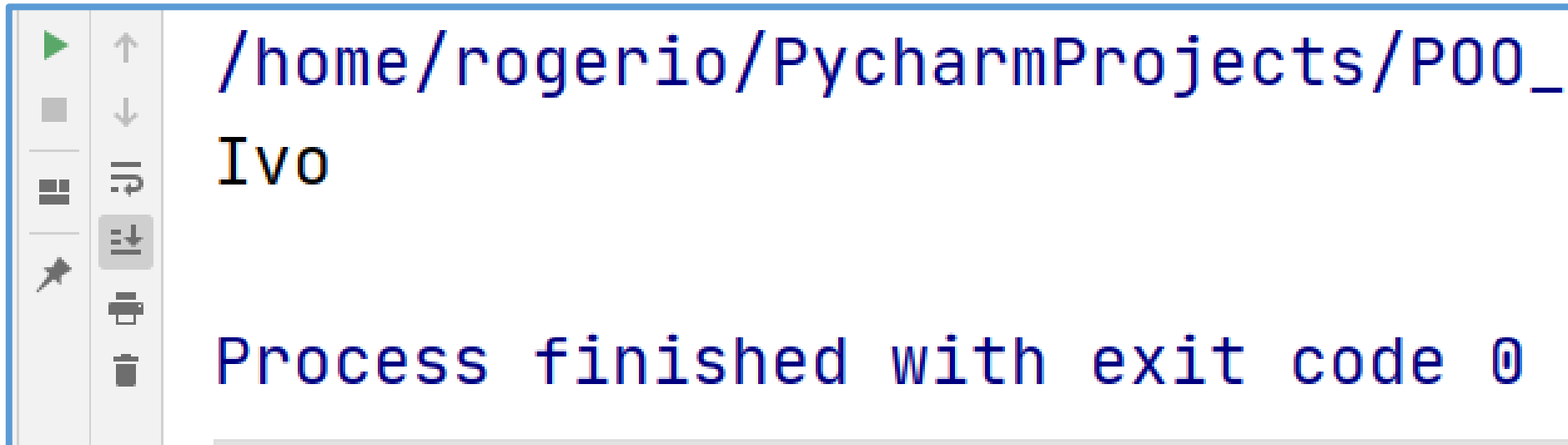
```
class Pessoa:
    def __init__(self, nome):
        self.__nome = nome

    def getnome(self):
        return self.__nome
```

Programação Orientada à Objetos: Encapsulamento

resultado da execução...

```
eu = Pessoa("Ivo")  
print(eu.getnome())  
.
```

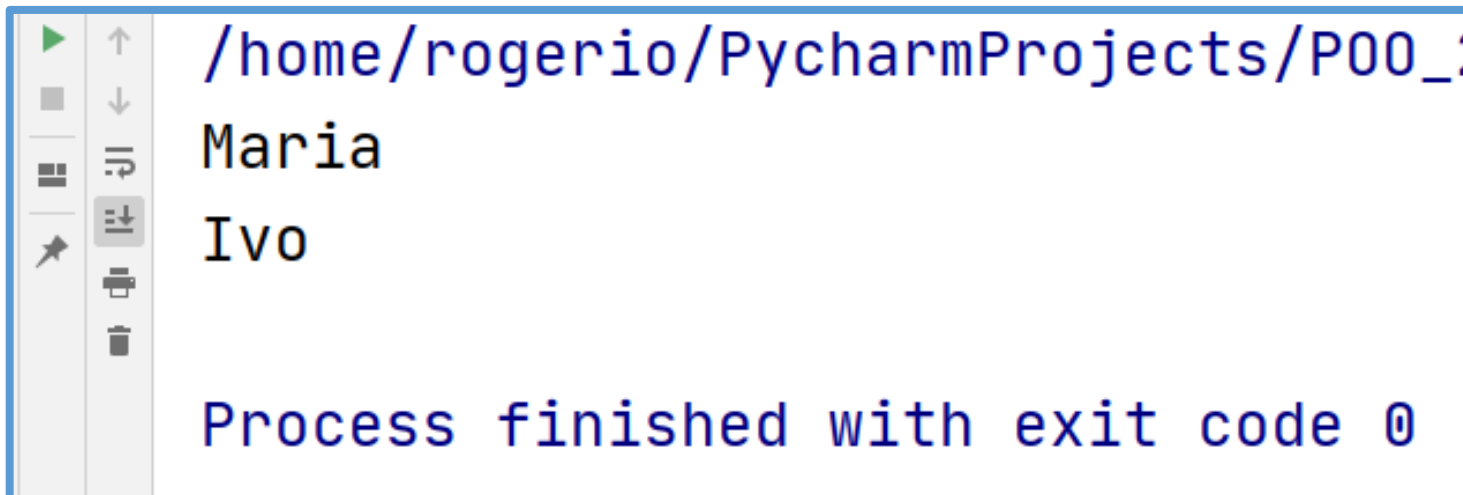


```
/home/rogerio/PycharmProjects/P00_  
Ivo  
  
Process finished with exit code 0
```

Programação Orientada à Objetos: Encapsulamento

furo no encapsulamento?

```
eu = Pessoa("Ivo")  
eu.nome="Maria" # ou eu.__nome="Maria"  
print(eu.nome) # ou print(eu.__nome)  
print(eu.getnome())
```



```
/home/rogerio/PycharmProjects/P00_1  
Maria  
Ivo  
  
Process finished with exit code 0
```

Programação Orientada à Objetos: Encapsulamento

Solução para este problema?

- Properties ou propriedades: uma forma mais elegante de encapsular nossos atributos.
- Um método que é usado para obter/alterar um valor (getter/setter) é decorado com `@property`, ou seja, colocamos esta linha exatamente acima da declaração do método cujo nome é o nome do próprio atributo.

Programação Orientada à Objetos: Encapsulamento

Sintaxe:

@property

```
def <nome_método>(self):  
    return self.__<nome_atributo>
```

método getter

nome_método = nome_atributo

```
@<nome_método>.setter  
def <nome_método>(self, valor):  
    self.__<nome_atributo>=valor
```

método setter

nome_método=nome_atributo

Programação Orientada à Objetos: Encapsulamento

Exemplo:

@property

```
def endereço(self):  
    return self.__endereço
```

```
@endereço.setter  
def endereço(self, valor):  
    self.__endereço=valor
```

Programação Orientada à Objetos:

Encapsulamento

```
class Pessoa:
    def __init__(self, nome):
        self.__nome=nome
    @property
    def nome(self):
        return self.__nome

    @nome.setter
    def nome(self, nome):
        a=self.__nome.split(" ")
        b=nome.split(" ")
        if a[0]==b[0]:
            self.__nome=nome
        else:
            print("primeiro nome não pode ser alterado")
```


Programação Orientada à Objetos:

Encapsulamento

Execução...

```
18 eu = Pessoa("Ivo")
19 print(eu.nome)
20 eu.nome="Ivo da Silva"
21 print(eu.nome)
22 eu.nome="Maria"
23 print(eu.nome)
```

```
↑ /home/rogerio/PycharmProjects/P00_2
↓
: Ivo
: Ivo da Silva
: primeiro nome não pode ser alterado
: Ivo da Silva

Process finished with exit code 0
```

Programação Orientada à Objetos: Encapsulamento

Exercícios