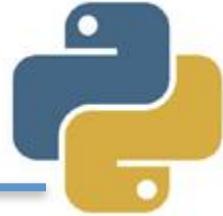


Capítulo 04

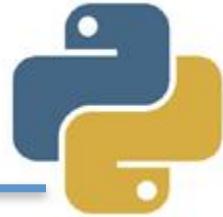


Encapsulamento, Atributos e Métodos de classe



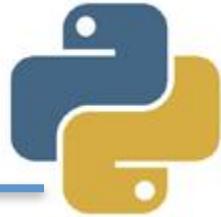
Márcio Palheta, M.Sc.
marcio.palheta@gmail.com

Apresentação



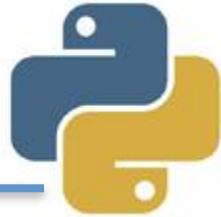
- Programador desde 2000
 - Professor de programação desde 2009
 - Mestre pelo ICOMP/UFAM – 2013
 - Fundador da Buritech – 2014
 - Doutorando pelo ICOMP/UFAM
 - Pesquisador das áreas: Banco de Dados, Recuperação da Informação, Big Data, Mineração de dados e Aprendizado de Máquina
-





Agenda

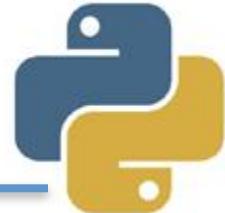
- Revisão da aula anterior
 - Controle de acesso a atributos
 - Encapsulamento pytônico
 - Entendendo a @property
 - Construtores com `__new__` e `__init__`
 - Atributos de classe e de instância
 - `@staticmethod` e `@classmethod`
-



Revisão

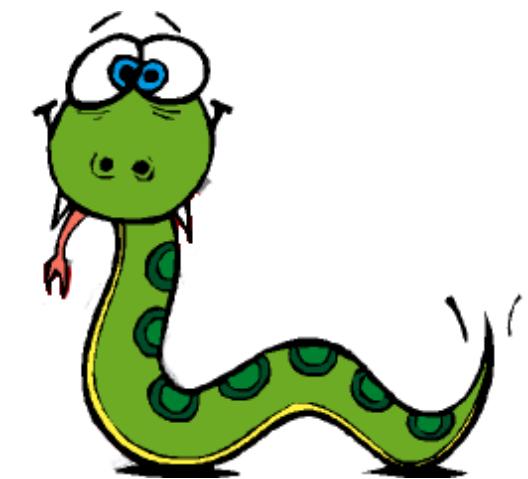
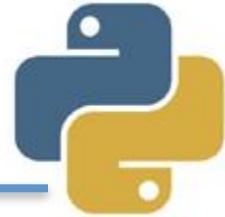
- Declaração de classes
 - Declaração de atributos e métodos
 - Criação de objetos
 - Manipulação de dados de objetos
 - Variáveis de referência
 - Associação entre classes
 - Tudo é objeto
-

O que temos por aqui?

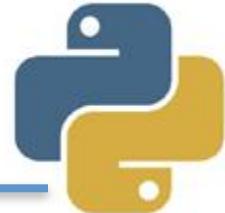


- Como restringir o acesso a atributos de instância?
 - Como trabalhar com construtores?
 - Qual a diferença entre variáveis de classe e instância?
 - O que são e como usar métodos estáticos?
-

Pensando em segurança

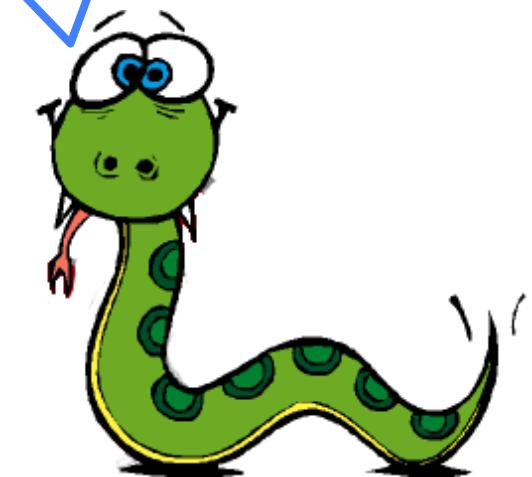
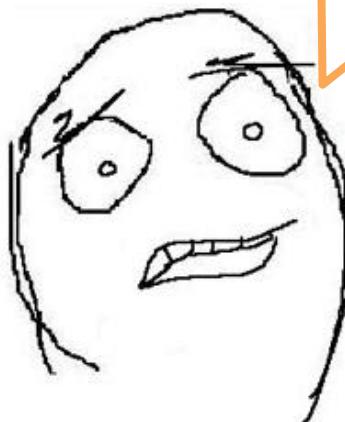


Pensando em segurança

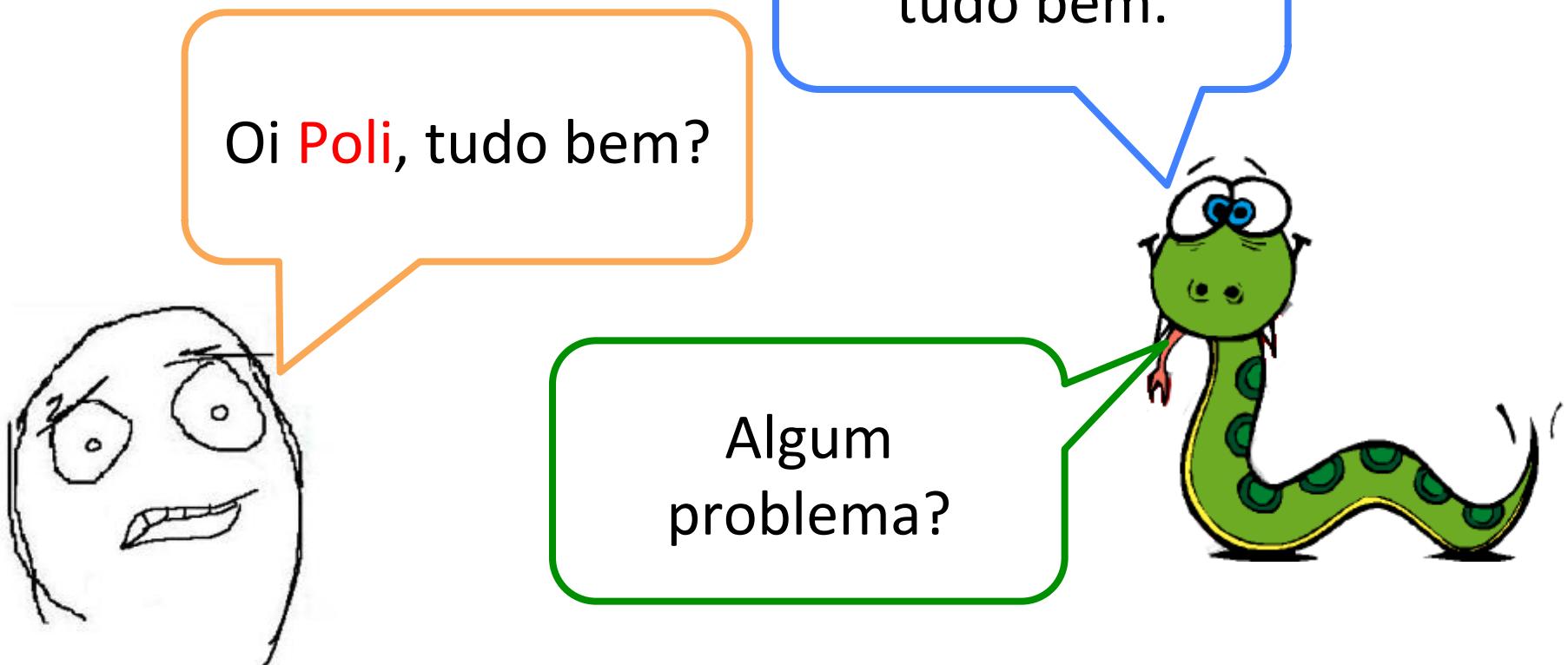
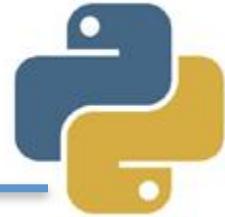


Oi **Poli**, tudo bem?

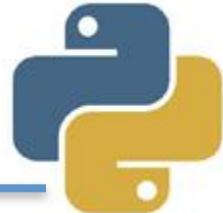
Oi **Morfismo**,
tudo bem.



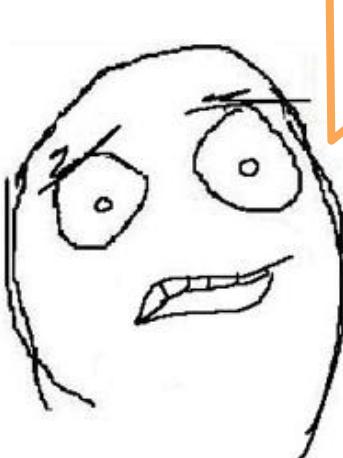
Pensando em segurança



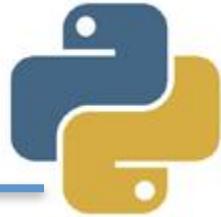
Pensando em segurança



Não. Só estou feliz
pelo nosso código do
capítulo anterior

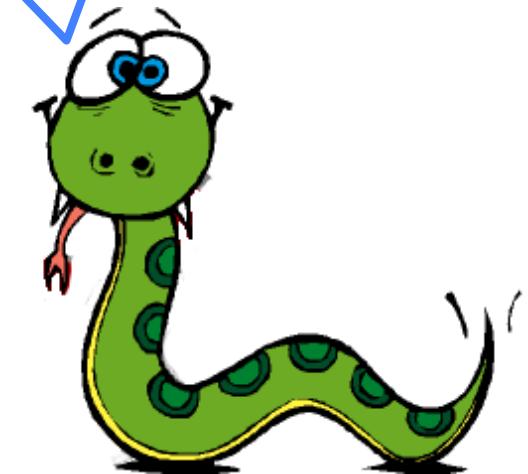
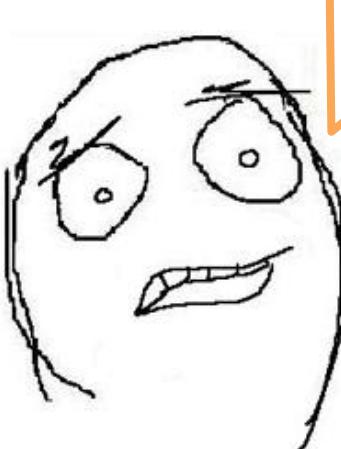


Pensando em segurança

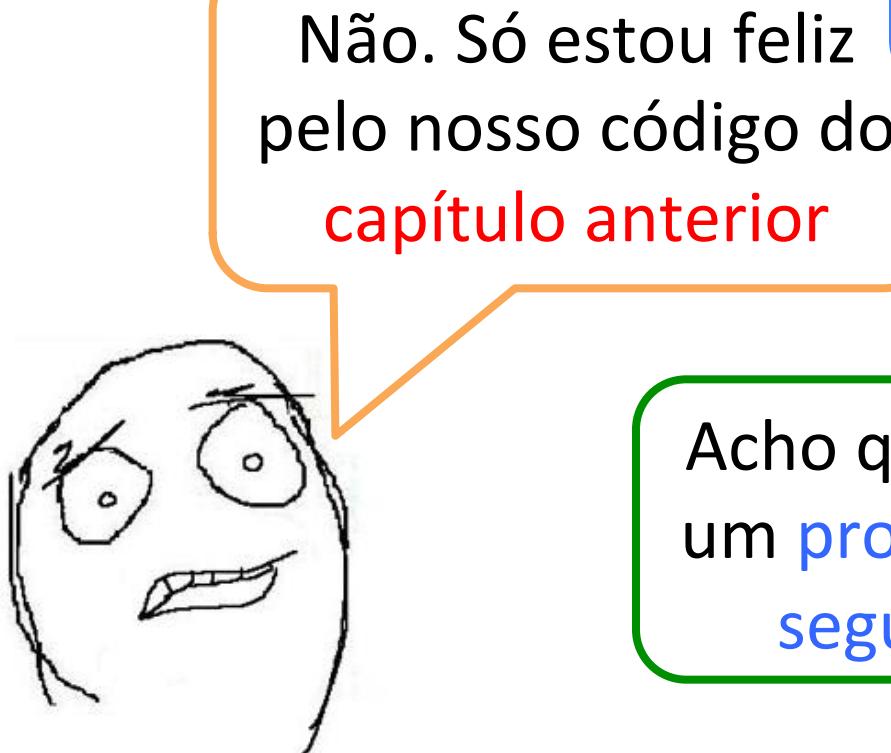
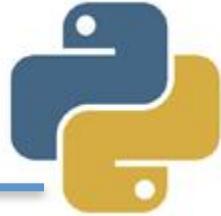


Não. Só estou feliz
pelo nosso código do
capítulo anterior

Ah, ainda bem
que você tocou
nesse assunto



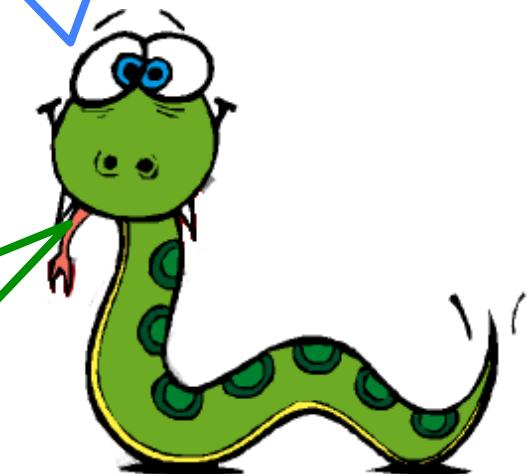
Pensando em segurança



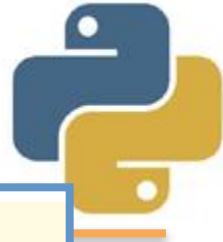
Não. Só estou feliz
pelo nosso código do
capítulo anterior

Ah, ainda bem
que você tocou
nesse assunto

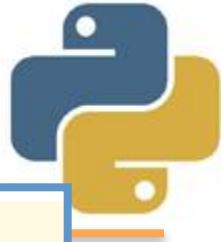
Acho que temos
um **problema de**
segurança



Revendo a ContaBancaria



```
3  class ContaBancaria(object):
4      def __init__(self):
5          self.agencia = None
6          self.numero = None
7          self.cliente = None
8          self.saldo = 0.0
9
10     def depositar(self, valor):
11         self.saldo += valor
12
13     def sacar(self, valor):
14         if valor <= self.saldo:
15             self.saldo -= valor
16             return True
17         return False
```

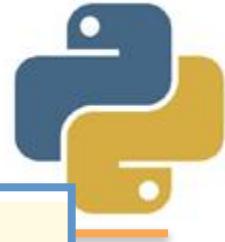


Revendo a ContaBancaria

```
3 class ContaBancaria(object):
4     def __init__(self):
5         self.agencia = None
6         self.numero = None
7         self.cliente = None
8         self.saldo = 0.0
9
10    def depositar(self, valor):
11        self.saldo += valor
12
13    def sacar(self, valor):
14        if valor <= self.saldo:
15            self.saldo -= valor
16            return True
17            return False
```

Criamos nossos
atributos de instância

Revendo a ContaBancaria

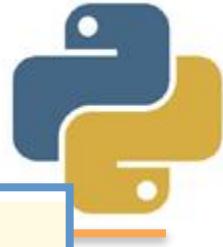


```
3 class ContaBancaria(object):
4     def __init__(self):
5         self.agencia = None
6         self.numero = None
7         self.cliente = None
8         self.saldo = 0.0
9
10    def depositar(self, valor):
11        self.saldo += valor
12        return True
13
14    def sacar(self, valor):
15        if self.saldo > valor:
16            self.saldo -= valor
17            return True
18        else:
19            return False
```

Criamos nossos
atributos de instância

Que são atributos que
pertencem a cada
objetos criado

Revisando a ContaBancaria



Também criamos métodos que atualizam o saldo da conta

```
7  
8     def __init__(self, nome, agencia, numero, cliente):  
9         self.nome = nome  
10        self.agencia = None  
11        self.numero = None  
12        self.cliente = None  
13        self.saldo = 0.0  
14  
15    def depositar(self, valor):  
16        self.saldo += valor  
17  
18    def sacar(self, valor):  
19        if valor <= self.saldo:  
20            self.saldo -= valor  
21            return True  
22        return False
```

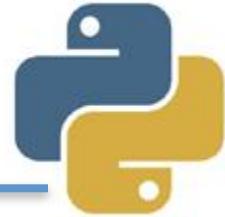
Revisando o Código

Também criamos métodos que atualizam o saldo da conta

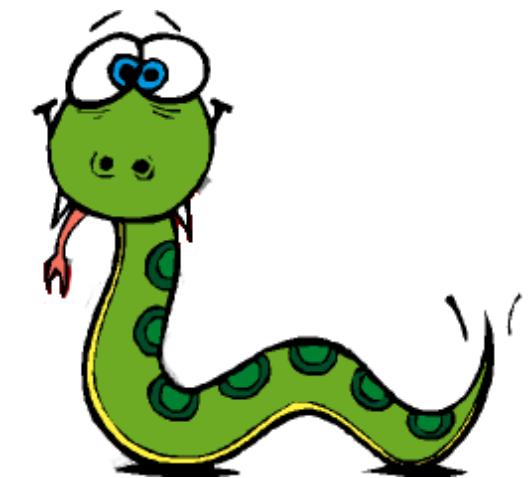
Teoricamente, só deveríamos alterar o saldo via um desses métodos

```
7
8     self.criar()
9
10    self.saldo
11
12
13    def depositar(self, valor):
14        self.saldo += valor
15
16    def sacar(self, valor):
17        if valor <= self.saldo:
18            self.saldo -= valor
19            return True
20        return False
```

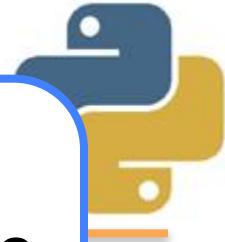
Pensando em segurança



E não é isso que
está acontecendo?

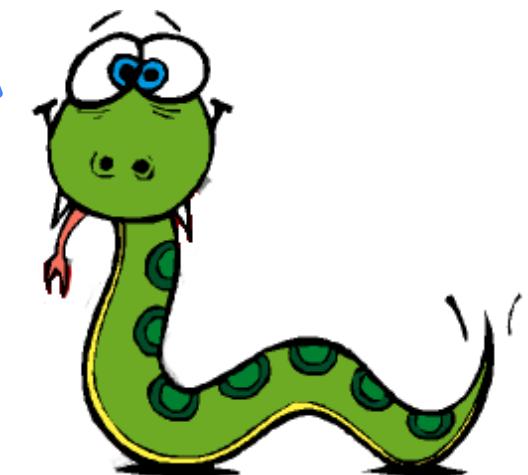
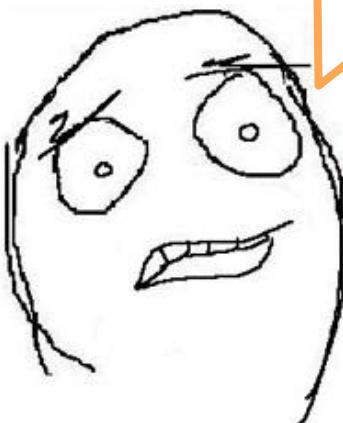


Pensando em

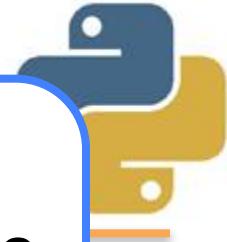


Não. Infelizmente,
conseguimos alterar o
saldo, **sem usar os**
métodos

E não é isso que
está acontecendo?



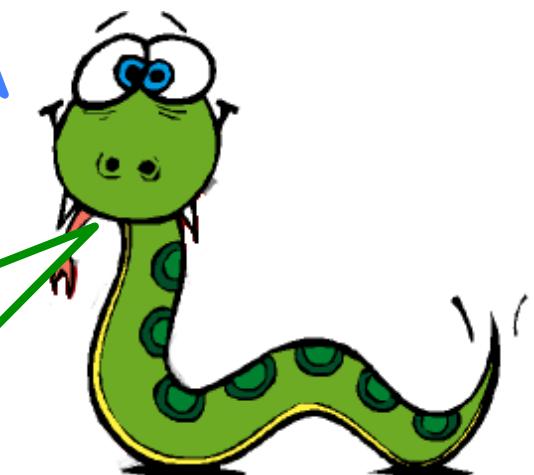
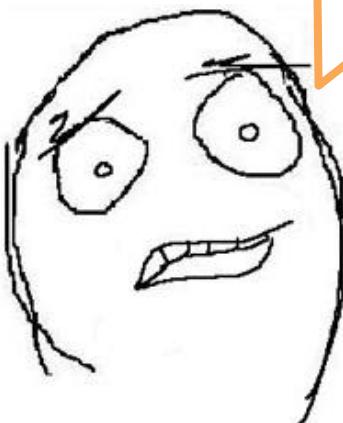
Pensando em



Não. Infelizmente,
conseguimos alterar o
saldo, **sem usar os**
métodos

E não é isso que
está acontecendo?

Dá uma **olhada** no
código do script
teste_conta.py





Revendo teste_conta.py

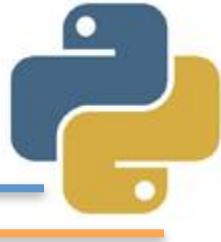
```
from model import ContaBancaria
conta = ContaBancaria()
conta.agencia = "0233"
conta.numero = "1234-5"
conta.nome_cliente = "Maria Jose"
conta.saldo = 1500.0

print "Cliente " + conta.nome_cliente
print "Agência", conta.agencia
print "Conta", conta.numero
print "Saldo "+str(conta.saldo)
```

ste_conta

```
/Library/Frameworks/Python.framework/V
Cliente Maria Jose
Agência 0233
Conta 1234-5
Saldo 1500.0
```





Revendo teste conta.py

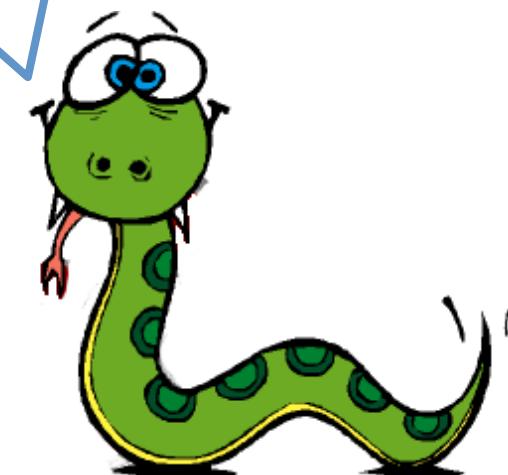
```
from model import ContaBanco
conta = ContaBanco()
conta.agencia = '0233'
conta.numero = "1234-5"
conta.nome_cliente = "MARIA JOSE"
conta.saldo = 1500.0
```

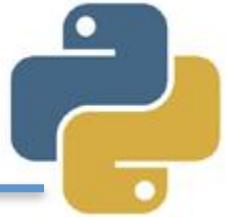
Conseguimos atribuir um **valor** qualquer para o **saldo** da conta 😞

```
print "Cliente " + conta.nome_cliente
print "Agência", conta.agencia
print "Conta", conta.numero
print "Saldo "+str(conta.saldo)
```

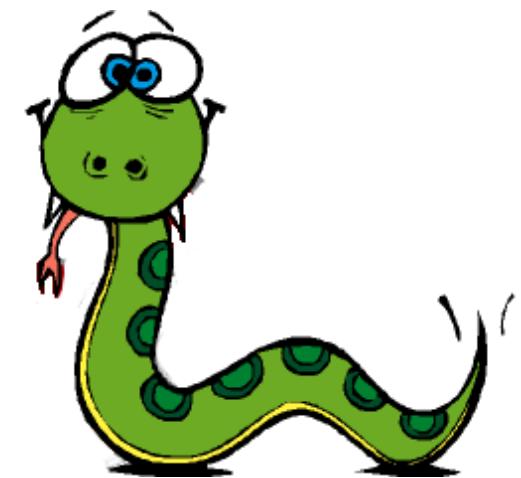
ste_conta

```
/Library/Frameworks/Python.framework/Versions/2.7/bin/python
Cliente Maria Jose
Agência 0233
Conta 1234-5
Saldo 1500.0
```





Hora da solução



Hora da solução



Em OO, podemos usar um conceito chamado **Encapsulamento**



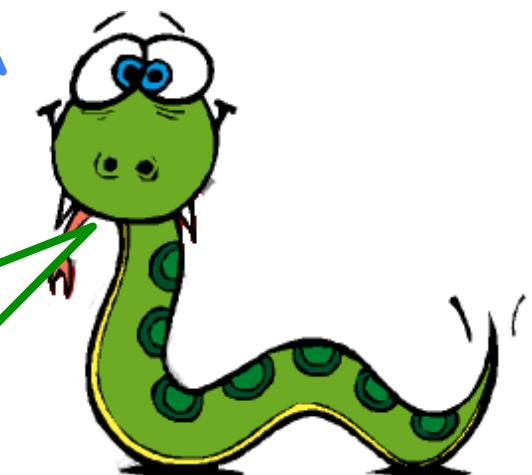
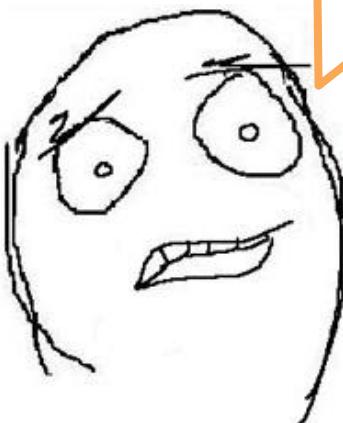
Hora da solução



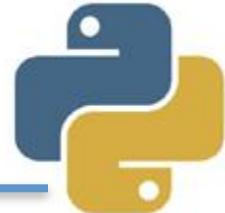
Em OO, podemos usar
um conceito chamado
Encapsulamento

Shiiiiii!
Como poderíamos
resolver isso

Onde criamos
métodos para
acesso a **atributos**



Encapsulamento

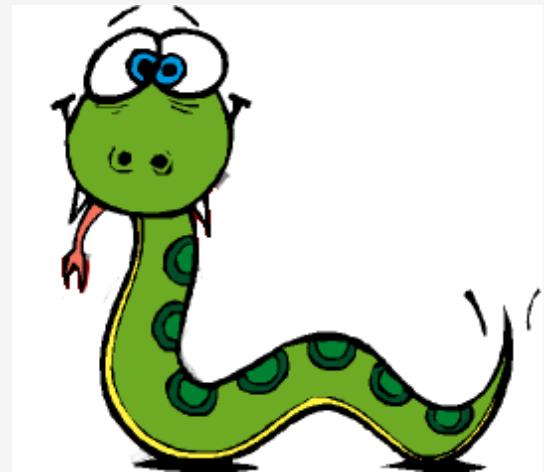


- Diferente de Java, em Python **não precisamos** usar encapsulamento em todos os atributos
 - Ou seja, encapsulamento **apenas para quem precisa** de encapsulamento
 - Atributos e métodos podem ser **públicos ou privados**
-

Imagine o código a seguir

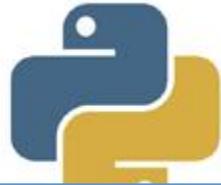


```
class Pessoa:  
    def __init__(self):  
        self.nome = ""  
  
p1 = Pessoa()  
  
p1.nome = "Maria"  
print p1.nome  
print "dir(Pe)", dir(Pessoa)  
print "dir(p1)", dir(p1)
```



Maria

```
dir(Pe) ['__doc__', '__init__', '__module__']  
dir(p1) ['__doc__', '__init__', '__module__', 'nome']
```



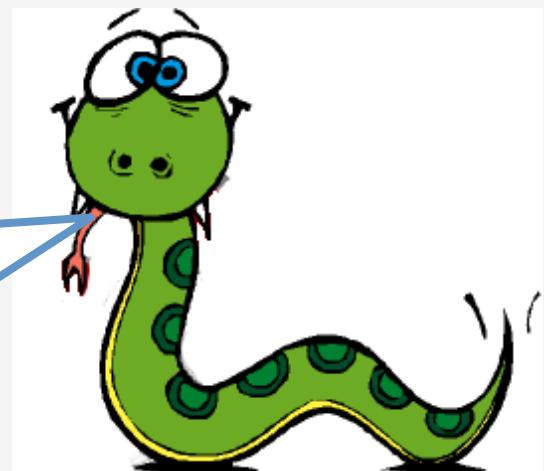
Imagine o código a seguir

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

```
p1 = Pessoa()
```

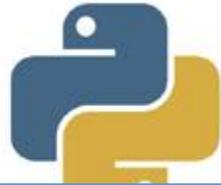
```
p1.nome = "Maria"  
print p1.nome  
print "dir(Pe  
print "dir(p1
```

Definimos a classe Pessoa, com atributo nome



```
Maria
```

```
dir(Pe) ['__doc__', '__init__', '__module__']  
dir(p1) ['__doc__', '__init__', '__module__', 'nome']
```



Imagine o código a seguir

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

```
p1 = Pessoa()
```

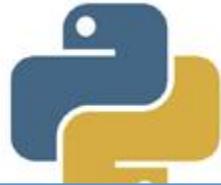
```
p1.nome = "Maria"  
print p1.nome  
print "dir(Pe)", dir(Pessoa)  
print "dir(p1)", dir(p1)
```

Criamos o a
variável p1



Maria

```
dir(Pe) ['__doc__', '__init__', '__module__']  
dir(p1) ['__doc__', '__init__', '__module__', 'nome']
```



Imagine o código a seguir

```
class Pessoa:  
    def __init__(self):  
        self.nome = None  
  
p1 = Pessoa()
```

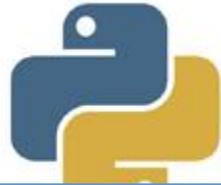
```
p1.nome = "Maria"  
print p1.nome  
print "dir(pe)", dir(Pessoa)  
print "dir(p1)", dir(p1)
```

Maria

```
dir(pe) ['__doc__', '__init__', '__module__']  
dir(p1) ['__doc__', '__init__', '__module__', 'nome']
```

Alteramos e
acessamos o valor do
atributo nome

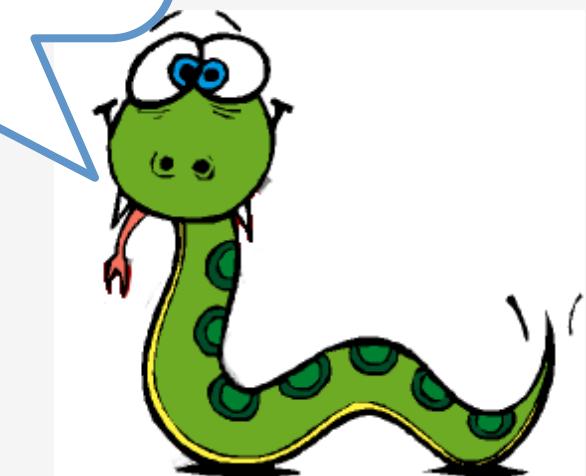




Imagine o código a seguir

```
class Pessoa:  
    def __init__(self):  
        self.nome = None  
  
p1 = Pessoa()  
  
p1.nome = "Maria"  
print p1.nome  
  
print "dir(Pe)", dir(Pessoa)  
print "dir(p1)", dir(p1)
```

Imprimimos as propriedades da classe e da entidade



Maria

```
dir(Pe) ['__doc__', '__init__', '__module__']  
dir(p1) ['__doc__', '__init__', '__module__', 'nome']
```

Tornando o atributo privado



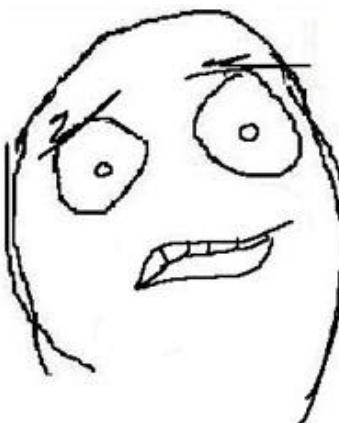
Em python, a definição
de **privado** vem no **nome**
do atributo



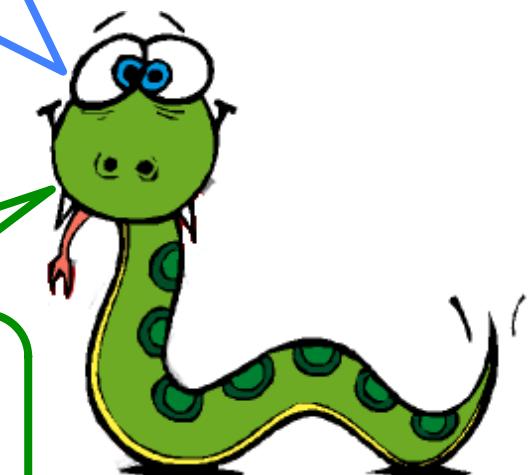
Tornando o atributo privado



Em python, a definição de **privado** vem no **nome do atributo**



Para isso, vamos incluir
dois underscores



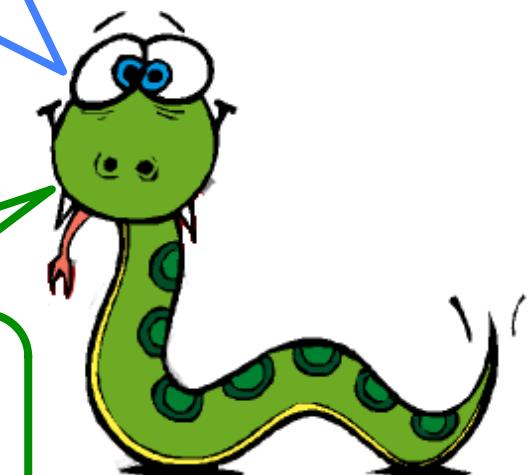
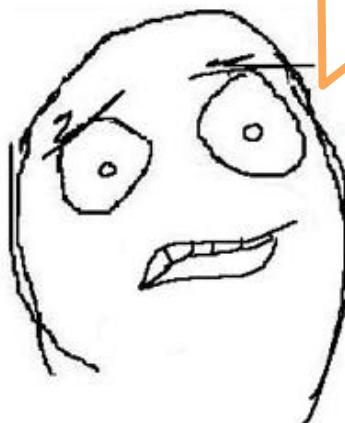
Tornando o atributo privado



Em python, a definição de **privado** vem no **nome do atributo**

Huuuuuum, parece fácil

Para isso, vamos incluir
dois underscores





Escondendo o atributo com __

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

p1 = Pessoa()

print p1.__nome

print "dir(Pe)", dir(Pessoa)
print "dir(p1)", dir(p1)
```



Escondendo o atributo com __

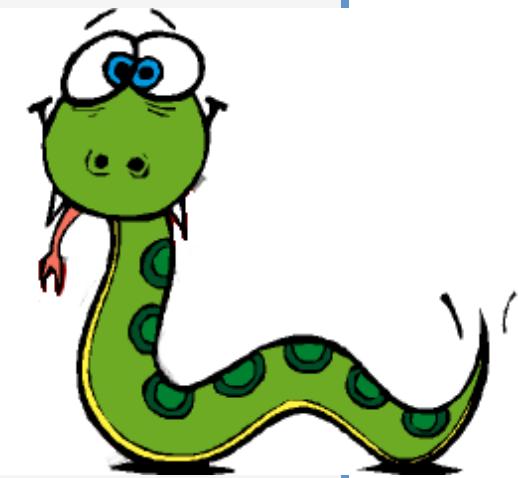


```
class Pessoa():
    def __init__(self):
        self.__nome = "Sem nome"
```

```
p1 = Pessoa()
```

```
print p1.nome
print "."
print ". nome para __nome"
```

Altere o atributo
nome para __nome





Escondendo o atributo com __

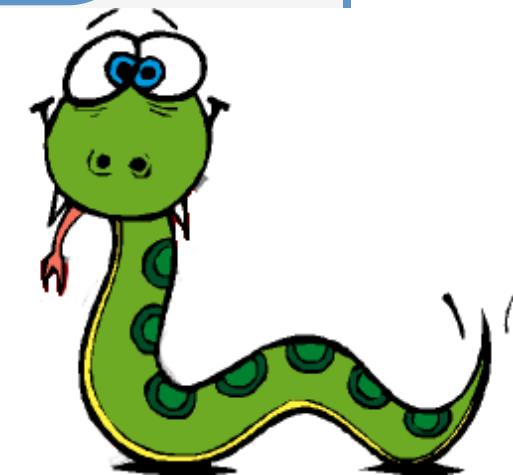
```
class Pessoa:  
    def __in  
    self
```

Tente acessar o
atributo p1.__nome
e execute novamente

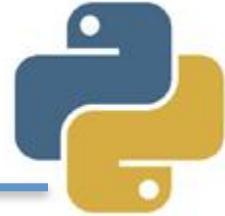
```
p1 = Pessoa()
```

```
print p1.__nome
```

```
print "dir(Pe)", dir(Pessoa)  
print "dir(p1)", dir(p1)
```



AttributeError: para __nome



```
-----  
AttributeError                                     Traceback (most r  
<ipython-input-12-25b231ce2e81> in <module>()  
      5 p1 = Pessoa()  
      6  
----> 7 print p1.__nome  
      8  
      9 print "dir(Pe)", dir(Pessoa)  
  
AttributeError: Pessoa instance has no attribute '__nome'
```



A

Ao tentar acessar
acessar __nome, é
lançado um
AttributeError

Attribut
<ipython-1>

```
5 p1 = Pessoa()  
6  
----> 7 print p1.__nome  
8  
9 print "dir(Pe)", dir(Pessoa)
```

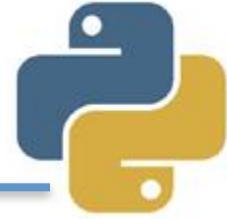
a __nome

<module>



AttributeError: Pessoa instance has no attribute '__nome'

Trabalhando com decorators



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

    @property
    def nome(self):
        return self.__nome

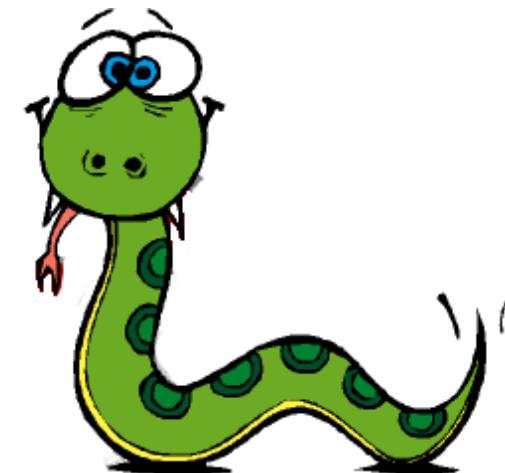
    @nome.setter
    def nome(self, nome):
        self.__nome = nome

p1 = Pessoa()

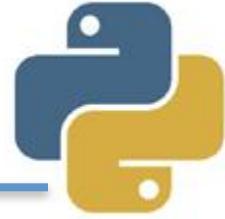
p1.nome = "Maria"

print p1.nome

print "\ndir(Pe)", dir(Pessoa)
print "dir(p1)", dir(p1)
```



Trabalhando com decorators



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

    @property
    def nome(self):
        return self.__nome

    @nome.setter
    def nome(self, nome):
        self.__nome = nome

p1 = Pessoa()

p1.nome = "Maria"

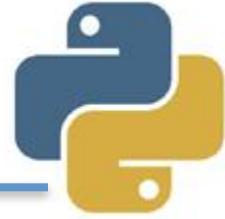
print p1.nome

print "\ndir(Pe)", dir(Pessoa)
print "dir(p1)", dir(p1)
```

Método que **devolve**
o valor de **__nome**



Trabalhando com decorators



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

    @property
    def nome(self):
        return self.__nome

    @nome.setter
    def nome(self, nome):
        self.__nome = nome

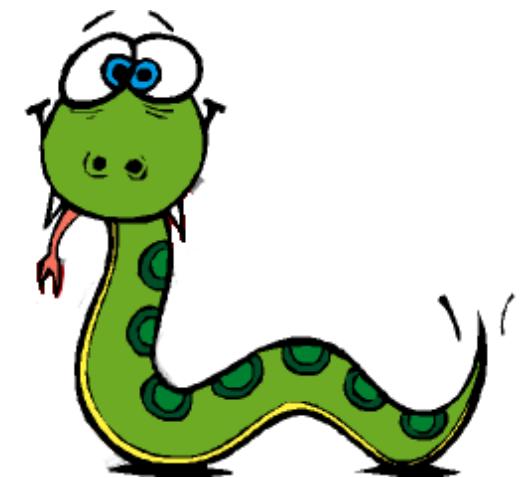
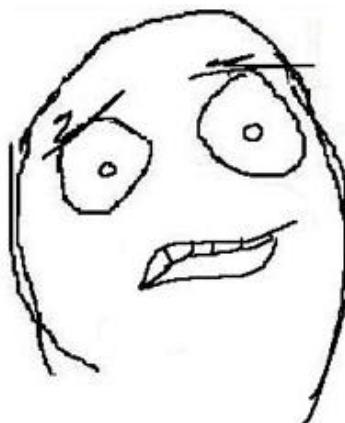
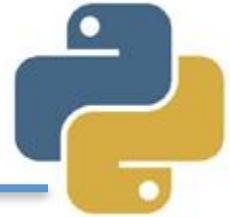
p1 = Pessoa()
p1.nome = "Maria"
print p1.nome

print "\ndir(Pe)", dir(Pessoa)
print "dir(p1)", dir(p1)
```

Método que **alterar** o valor de **__nome**

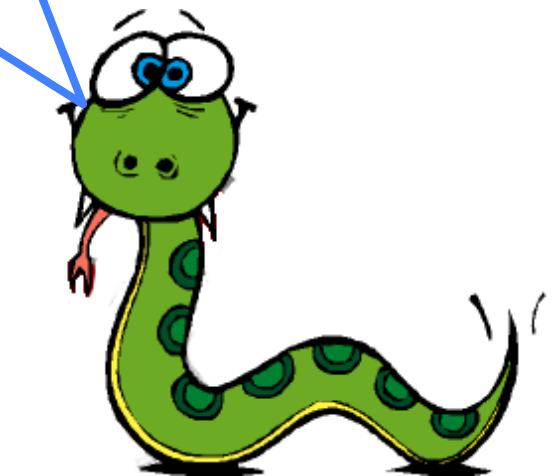


Encapsulamento do saldo

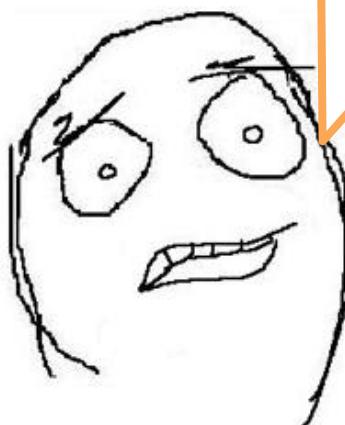


Encapsulame

E agora, depois dessa **explicação**, tens alguma ideia para resolver o **problema do saldo?**

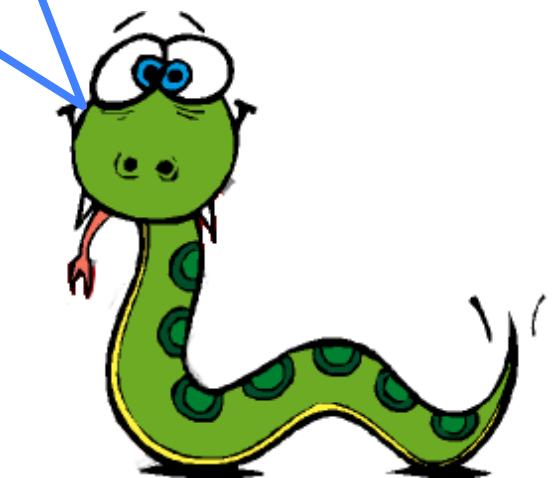


Encapsulame

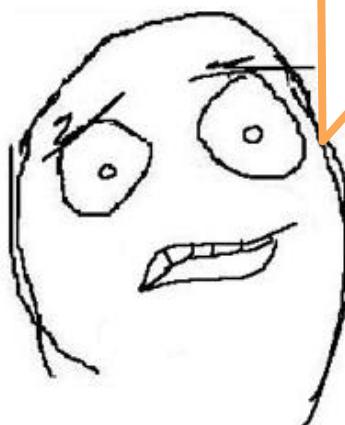


Podemos usar
decorator para
permitir **apenas**
leitura

E agora, depois dessa
explicação, tens alguma
ideia para resolver o
problema do saldo?



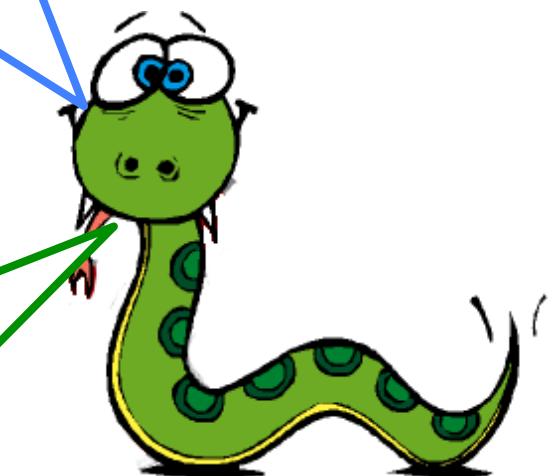
Encapsulame



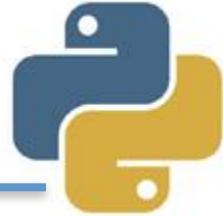
Podemos usar
decorator para
permitir **apenas**
leitura

E agora, depois dessa
explicação, tens alguma
ideia para resolver o
problema do saldo?

Isso mesmo.
Bom garoto.



Exercício



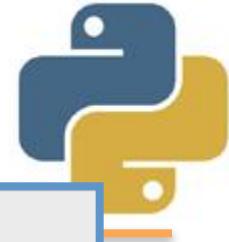
- Volte ao arquivo `model.py`
 - Altere a classe `ContaBancaria`
 - Altere a variável `saldo` para `__saldo`
 - Crie a `property` `saldo`, para oferecer
acesso de leitura ao atributo `__saldo`
 - Altere os métodos `sacar` e `depositar`,
indicando a variável correta
-

Exercício 1: encapsulamento

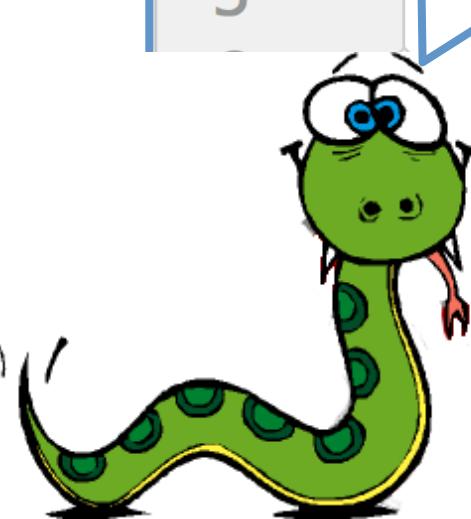


```
model.py  teste_cliente.py  teste_conta.py
          ContaBancaria saldo()

1  class ContaBancaria:
2      def __init__(self):
3          self.agencia = None
4          self.numero = None
5          self.cliente = None
6          self.__saldo = 0.0
7
8      @property
9      def saldo(self):
10         return self.__saldo
11
```



Exercício 1: encapsulamento



model.py teste_cliente.py teste_conta.py

ContaBancaria saldo()

```
1 class ContaBancaria:
2     def __init__(self):
3         self._saldo = None
4         self._cliente = None
5         self._numero = None
6
7     @property
8     def saldo(self):
9         return self._saldo
```

Altere o atributo **saldo** para **__saldo**

self.__saldo = 0.0

@property
def saldo(self):
 return self.__saldo

Exercício 1: encapsulamento



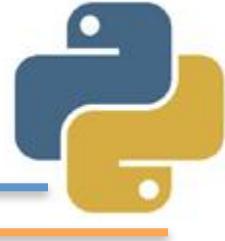
```
model.py  teste_cliente.py  teste_conta.py
          ContaBancaria saldo()
1 class ContaBancaria:
2     def __init__(self):
3         self._titular = None
4         self._numero = None
5         self._cliente = None
6         self.__saldo = 0.0
```

Crie a **property** para devolver o `_saldo`



```
@property
def saldo(self):
    return self.__saldo
```

Exercício 2: alterar métodos



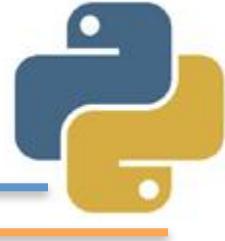
```
@property
```

```
def saldo(self):  
    return self.__saldo
```

```
def depositar(self, valor):  
    self.__saldo += valor
```

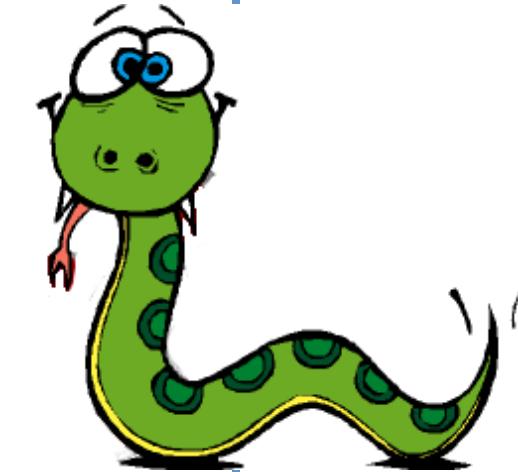
```
def sacar(self, valor):  
    if self.__saldo >= valor:  
        self.__saldo -= valor  
        return True  
    return False
```

Exercício 2: alterar métodos

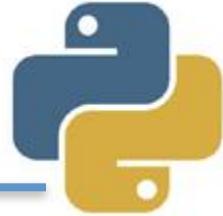


```
@property  
def saldo(self)  
    return self.saldo  
  
def depositar(self, valor)  
    self._saldo += valor  
  
def sacar(self, valor):  
    if self._saldo >= valor:  
        self._saldo -= valor  
        return True  
    return False
```

Altere self.saldo para self._saldo

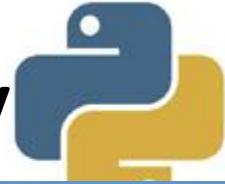


Exercício



- Crie o arquivo `teste_property.py`
 - No novo arquivo, importe a classe **ContaBancaria**
 - Crie uma instância de ContaBancaria
 - Deposite **1000**, Saque **200**
 - Imprima o saldo das operações
 - Tente alterar o valor do saldo
-

Exercício 3: teste da property



```
1  # -*- coding: UTF-8 -*-
2  from model import ContaBancaria
3
4  conta = ContaBancaria()
5  print 'Saldo inicial:', conta.saldo
6
7  conta.depositar(1000)
8  print 'Saldo após depósito:', conta.saldo
9
10 conta.sacar(200.0)
11 print 'Saldo após saque:', conta.saldo
12
13 conta.saldo = 500
14 print 'Saldo final:', conta.saldo
```

Exercício 3: teste da property

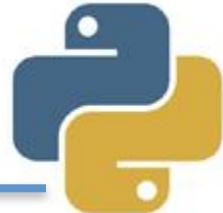


```
1  # -*- coding: UTF-8 -*-
2  from mode
3
4  conta = Conta('João')
5  print 'Saldo inicial:', conta.saldo
6
7  conta.depositar(1000)
8  print 'Saldo após depósito:', conta.saldo
9
10 conta.sacar(200.0)
11 print 'Saldo após saque:', conta.saldo
12
13 conta.saldo = 500
14 print 'Saldo final:', conta.saldo
```

Será que a **linha 13** vai
executar normalmente?

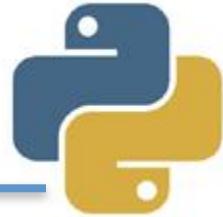


Resultado no console



```
Run teste_property
/Libra.../Python.frame
Saldo inicial: 0.0
Saldo após depósito: 1000.0
Saldo após saque: 800.0
Saldo final: 500
```

Resultado no console

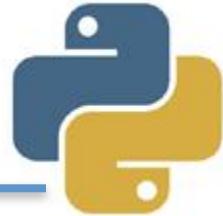


```
Run teste_property
/Librar
Saldo i
Saldo ap
Saldo após saque: 800.0
Saldo final: 500
```

Parece que, mesmo com
@property, foi possível
alterar o saldo 😢



Por que isso aconteceu ?



- O problema está na **forma de declaração** da classe, que segue o **old-style class**:

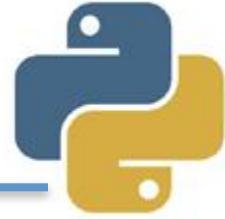
```
class ContaBancaria:
```

- Desde o **python 2.2**, passamos a usar o **new-style class**:

```
class ContaBancaria(object):
```

Ref.: <https://wiki.python.org/moin/NewClassVsClassicClass>

Exercício 4: new-style class



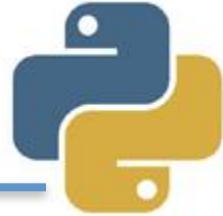
- Aplique o **new-style** em ContaBancaria

The screenshot shows a code editor with two tabs: "model.py" and "teste_property.py". The "model.py" tab is active and displays the following Python code:

```
1  class ContaBancaria(object):
2      def __init__(self):
3          self.agencia = None
4          self.numero = None
5          self.cliente = None
6          self.__saldo = 0.0
7
```

The code defines a new-style class named `ContaBancaria` that inherits from `object`. It includes an `__init__` method that initializes attributes `agencia`, `numero`, `cliente`, and `__saldo` to `None` and `0.0` respectively.

Exercício 4: new-style class



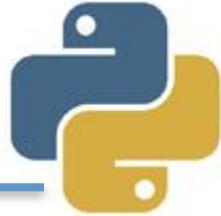
- Aplique o **new-style** em ContaBancaria

The screenshot shows a code editor with two tabs at the top: "model.py" and "teste_property.py". The "model.py" tab is active and contains the following Python code:

```
1  class ContaBancaria(object):
2      def __init__(self):
3          self.agencia = None
4          self.numero = None
5          self.cliente = None
6          self.__saldo = 0.0
7
```

The first line, "class ContaBancaria(object):", is highlighted with a red rectangular box. The code uses Python's new-style class syntax, where the class name is followed by a colon and the object type is specified in parentheses.

Exercício 5: rode novamente

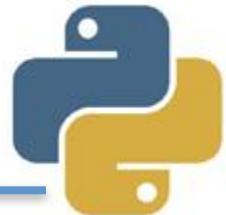


- Execute o script `teste_property.py` e veja que o resultado mudou:

A screenshot of a Python IDE showing the output of a script named `teste_property.py`. The output shows the initial balance, a deposit, and a withdrawal, followed by a traceback indicating an attempt to set the `saldo` attribute.

```
Run  Run teste_property
Run  Run teste_property
/ Library/Frameworks/Python.framework/Versions/3.7/bin/python3.7 "/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/runpy.py" "/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/IPython/qt/ipkernel_launcher.py"
Saldo inicial: 0.0
Traceback (most recent call last):
  File "/Users/marciopalheta/python/cap08/teste_property.py", line 11, in <module>
    conta.saldo = 500
AttributeError: can't set attribute
```

Exercício 5: rode novamente



- Execute o script `teste_property.py` e veja que o resultado mudou:



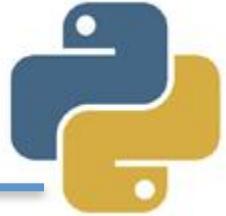
A screenshot of a Python code editor showing the output of a script named `teste_property.py`. The code defines a class `Conta` with a private attribute `_saldo` and a public property `saldo`. It demonstrates reading the saldo and attempting to write a new value of 500, which results in an `AttributeError`.

```
Run  teste_property
/ Library/Frameworks/Python.
Saldo: 1000.0
Saldo após saque: 800.0
conta.saldo = 500
AttributeError: can't set attribute
```

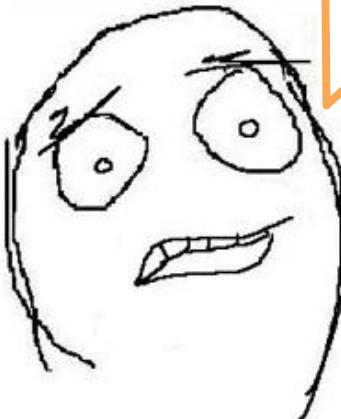
The output is annotated with two callouts:

- A blue callout points to the line `conta.saldo = 500` with the text: "Agora, não é possível alterar o valor do saldo".
- A red callout points to the error message `AttributeError: can't set attribute` with the text: "Agora, não é possível alterar o valor do saldo".

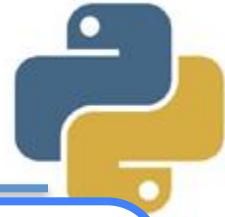
Encapsulamento do saldo



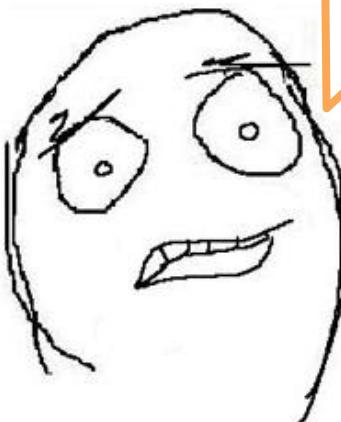
Poli, o que deveríamos fazer se, por algum motivo, precisássemos alterar o saldo?



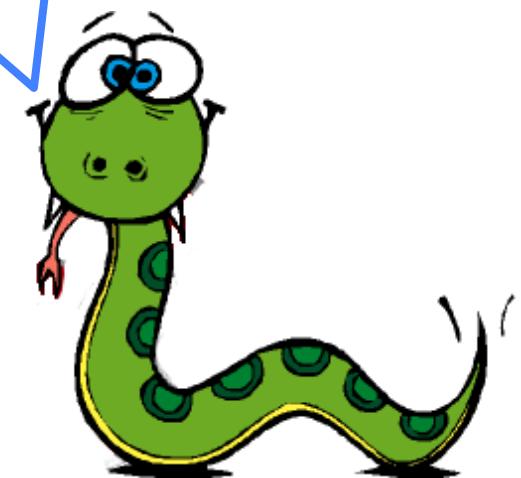
Encapsulamento do saldo



Poli, o que deveríamos fazer se, por algum motivo, precisássemos alterar o saldo?



Morfismo, isso não seria problema.



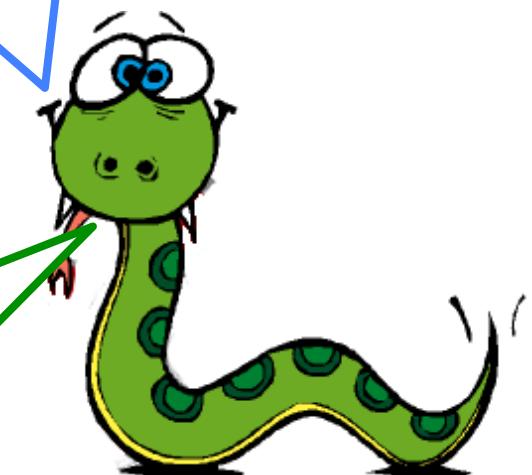
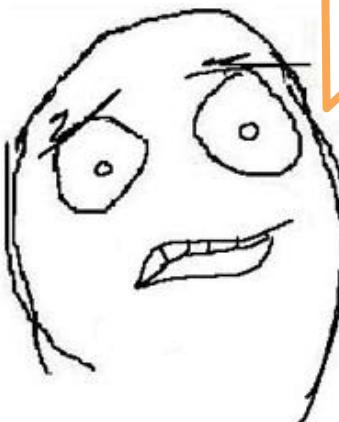
Encapsulamento do saldo



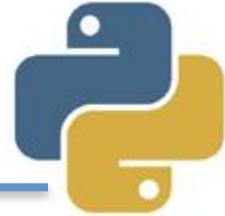
Poli, o que deveríamos fazer se, por algum motivo, precisássemos alterar o saldo?

Poderíamos criar um **setter** para **atualizar o saldo**. Olha só...

Morfismo, isso **não** seria problema.



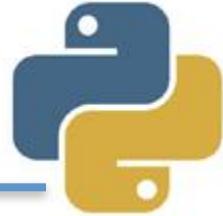
Definindo um setter



- Usando **decorator**, o método para alterar saldo seria marcado com **@saldo.setter**

```
@property  
def saldo(self):  
    return self.__saldo  
  
@saldo.setter  
def saldo(self, novo_saldo):  
    self.__saldo = novo_saldo
```

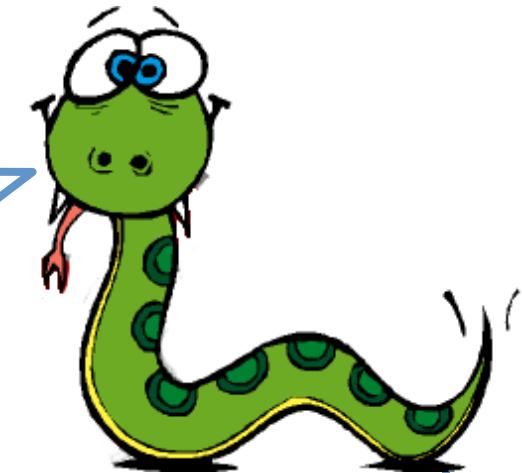
Definindo um setter



- Usando **decorator**, o método para alterar saldo seria marcado com **@**

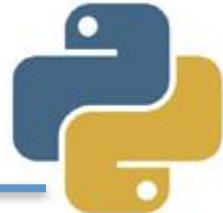
@p
de

O setter recebe o
valor do **novo saldo**



```
@saldo.setter  
def saldo(self, novo_saldo):  
    self.__saldo = novo_saldo
```

Usando getter e setter



```
1 # -*- coding: UTF-8 -*-
2 from model import ContaBancaria
3 conta = ContaBancaria()
4 print 'Saldo inicial:', conta.saldo
5 conta.saldo = 500
6 print 'Saldo final:', conta.saldo
```

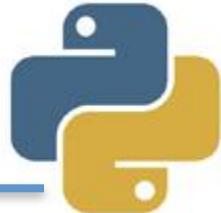
Run teste_property

/Library/Frameworks/Python.framework/Ve

Saldo inicial: 0.0

Saldo final: 500

Process finished with exit code 0



Usando getter e setter

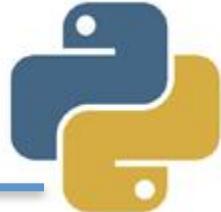
```
1 # -*- coding: UTF-8 -*-
2 from model import ContaBancaria
3 conta = ContaBancaria()
4 print(conta.saldo) # Chamando o getter do saldo
5 conta.depositar(500)
6 print(conta.saldo)
```

Run teste_property

```
try/Frameworks/Python.framework/Ve
initial: 0.0
final: 500
Process finished with exit code 0
```



Usando getter e setter



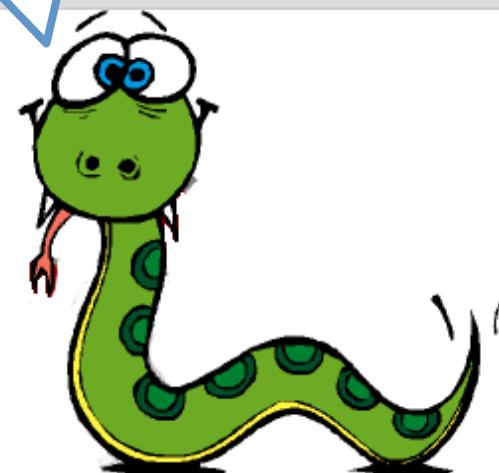
```
1 # -*- coding: UTF-8 -*-
2 from model import ContaBancaria
3 conta = ContaBancaria()
4 print 'Saldo inicial: '
5 conta.saldo = 500
6 print 'Saldo final: '
```

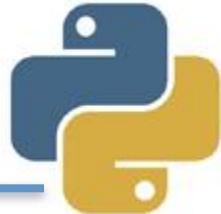
Chamando o
setter do saldo

Run teste_property

```
/Library/Frameworks/Pyth
Saldo inicial: 0.0
Saldo final: 500
```

Process finished with exit code 0





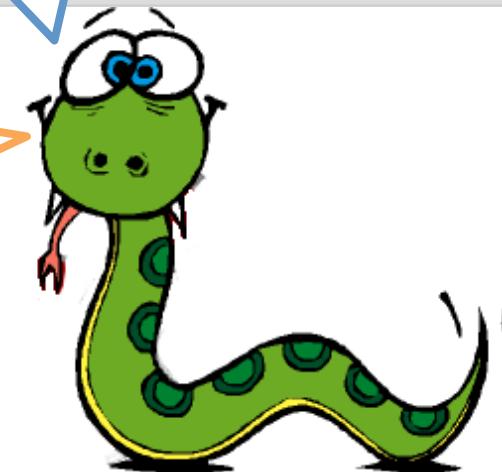
Usando getter e setter

```
1 # -*- coding: UTF-8 -*-
2 from model import ContaBancaria
3 conta = ContaBancaria()
4 print 'Saldo inicial: '
5 conta.saldo = 500
6 print 'Saldo final: '
```

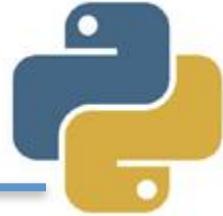
Chamando o
setter do saldo

O valor 500.0 é passado
como parâmetro para o
setter de saldo

ex

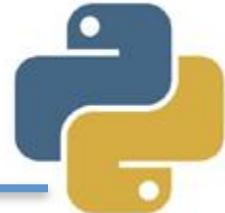


Convenções



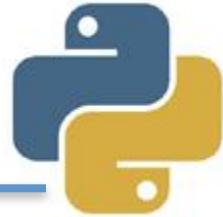
- Todas as classes devem herdar de `object`,
exceto as antigas (`old style`)
 - Não existem classes `antigas` no Python 3
 - Nomenclatura de classes(maiúsculas),
atributos e métodos(minúsculas):
 - Classes: `ContaBancaria`, `Cliente`
 - Atributos e métodos: `saldo`, `ver_saldo()`
-

Construtores



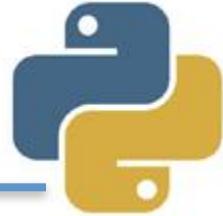
- Quando usamos o comando a seguir
 - `conta = ContaBancaria()`
 - Estamos chamando o construtor da classe **ContaBancaria**
 - Construtores são métodos usados para criação de objetos de uma classe
 - Nome do método construtor: **_new_**
-

__new__ versus __init__



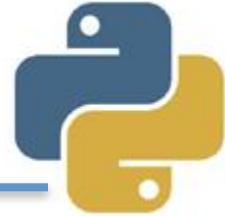
- Diferenças entre os métodos:
 - __init__ recebe uma **instância** já construída(**self**) e sua função é **inicializar os atributos** da instância
 - __new__ sobrescrito quando precisamos interferir no **processo de construção** da instância. Usado em **meta-programação**
-

Exercício



- Altere o método `__init__` da classe **ContaBancaria**, para imprimir uma mensagem, sempre que for executado
 - Crie o arquivo `teste_construtor.py`
 - No novo arquivo, crie um **objeto conta**, execute o script e veja se a mensagem do construtor foi impressa no console
-

Exercício 6: teste_construtor



```
model.py x teste_property.py x
ContaBancaria __init__()

1 class ContaBancaria(object):
2     def __init__(self):
3         self.agencia = None
4         self.numero = None
5         self.cliente = None
6         self.__saldo = 0.0
7         print 'ContaBancaria.__init__()'
```

Exercício 6:

Inclua uma mensagem de log em `__init__`



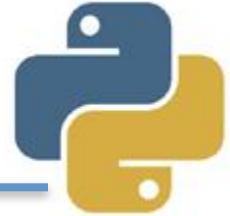
A cartoon illustration of a green snake with large eyes and a red tongue, positioned to the right of the code editor window.

model.py x teste_property.py x

ContaBancaria __init__()

```
1 class ContaBancaria(object):
2     def __init__(self):
3         self.agencia = None
4         self.numero = None
5         self.cliente = None
6         self.saldo = 0.0
7         print 'ContaBancaria.__init__()'
```

Exercício 7: teste_construtor

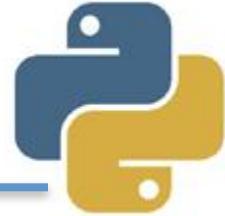


The screenshot shows a PyCharm IDE interface. On the left, the project tree displays a folder named 'financeiro' containing files like '.ipynb_checkpoints', 'model.py', 'teste_cliente.py', 'teste_construtor.py' (which is selected), 'teste_conta.py', 'teste_property.py', and 'Untitled.ipynb'. Below the project tree is an 'External Libraries' section. The main editor window shows the code for 'teste_construtor.py':

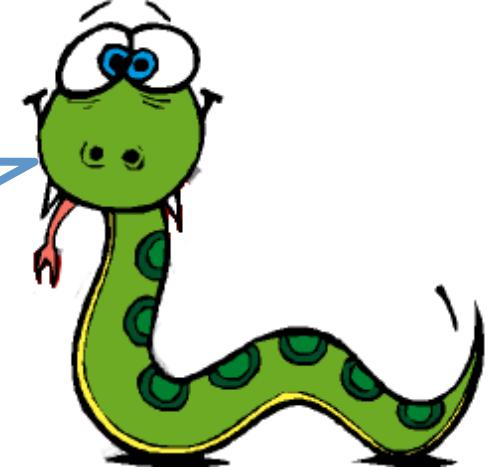
```
# -*- coding: UTF-8 -*-
from model import ContaBancaria
conta = ContaBancaria()
```

The code editor has syntax highlighting with numbers 1 through 5 on the left. The status bar at the bottom shows the path '/Library/Frameworks/Python.framework/Versions/2.7' and the function 'ContaBancaria.__init__()'.

Exercício 7: teste_construtor



Chamando o construtor de ContaBancaria

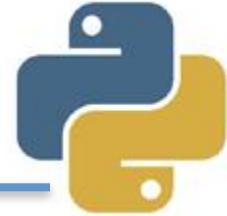


A cartoon illustration of a green snake with large eyes and a wide, open mouth, appearing to be speaking or reacting to the code.

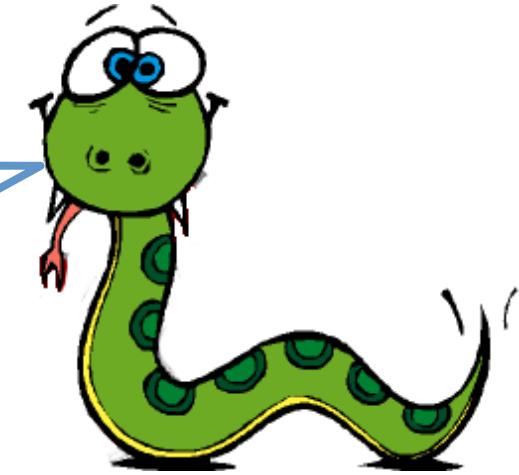
```
1
2
3
4
5    conta = ContaBancaria()
```

The code editor shows a snippet of Python code. Line 5, which contains the constructor call `conta = ContaBancaria()`, is highlighted with a red box. The output window below shows the path to the constructor: `/Library/Frameworks/Python.framework/Versions/2.7 ContaBancaria.__init__()`.

Exercício 7: teste_construtor



Resultado da execução do script

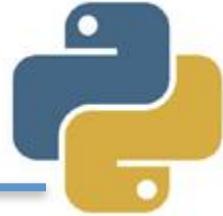


A cartoon illustration of a green snake with large eyes and a wide, open mouth, positioned next to the speech bubble.

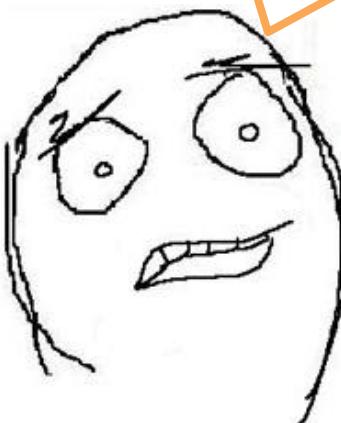
```
1
2
3
4
5 conta = ContaBancaria()
```

The code editor shows a Python script named `teste_construtor.py`. The fifth line of code is highlighted with a yellow background and contains the call to the constructor: `conta = ContaBancaria()`. A red box highlights the constructor call `ContaBancaria.__init__()` in the run output, which is displayed in a terminal window below the code editor.

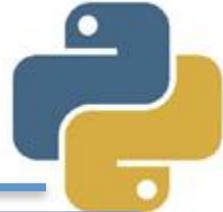
Inicialização de atributos



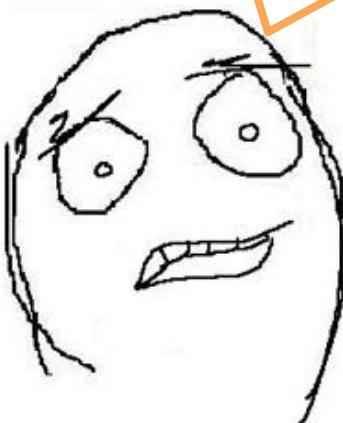
Entendi. Mas Poli, e se eu precisasse **iniciar** a conta com **saldo**?



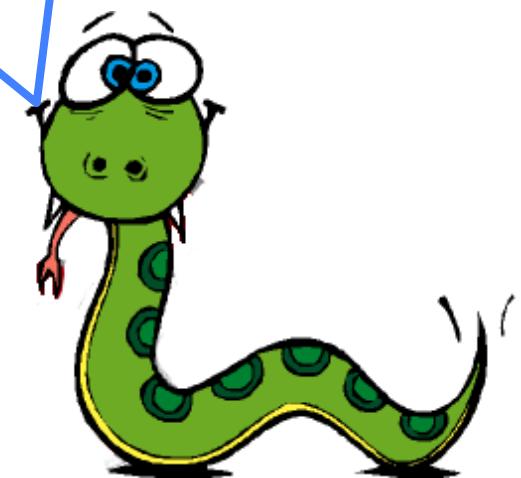
Inicialização de atributos



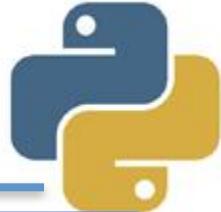
Entendi. Mas Poli, e se eu precisasse **iniciar** a conta com **saldo**?



Morfismo, isso **não** seria problema.



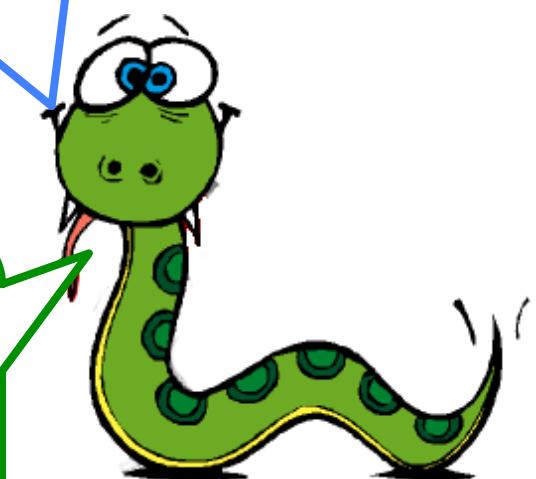
Inicialização de atributos



Entendi. Mas Poli, e se eu precisasse **iniciar** a conta com **saldo**?

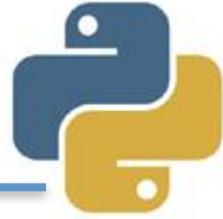


Morfismo, isso **não** seria problema.



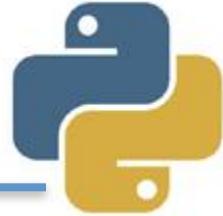
Poderíamos informar o **valor do saldo** para o método **`__init__`**

Exercício 8: saldo inicial



```
class ContaBancaria(object):
    def __init__(self, saldo):
        self.agencia = None
        self.numero = None
        self.cliente = None
        self.__saldo = saldo
    print 'ContaBancaria.__init__()'
    print 'Saldo inicial: ', saldo
```

Altere a assinatura do método, para receber o valor inicial do saldo

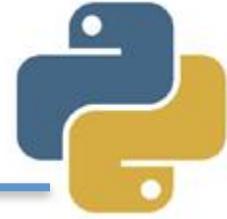


ial

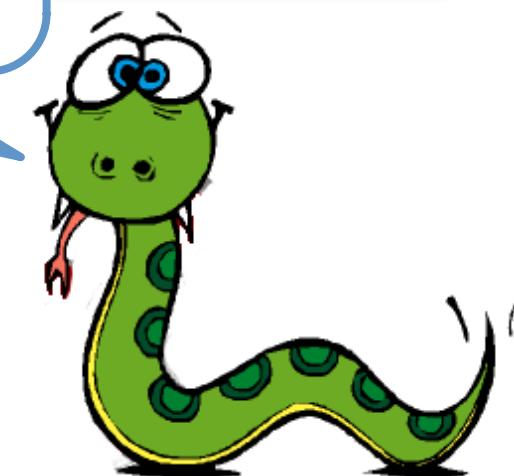
```
class ContaBancaria(object):
    def __init__(self, saldo):
        self.agencia = None
        self.numero = None
        self.cliente = None
        self.__saldo = saldo
    print 'ContaBancaria.__init__()'
    print 'Saldo inicial: ', saldo
```



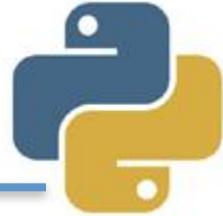
O atributo da instância deve receber o valor do parâmetro saldo



```
class ContaBancaria(object):
    def __init__(self, saldo):
        self.agencia = None
        self.numero = None
        self.cliente = None
        self.__saldo = saldo
    print 'ContaBancaria.__init__()'
    print 'Saldo inicial:', saldo
```



Crie uma mensagem para informar o **valor** do **saldo inicial** da conta



ial

```
class ContaBancaria(object):
    def __init__(self, saldo):
        self.agencia = None
        self.numero = None
        self.cliente = None
        self.__saldo = saldo
    print 'ContaBancaria.__init__()'
    print 'Saldo inicial: ', saldo
```



Exercício 9: teste saldo inicial

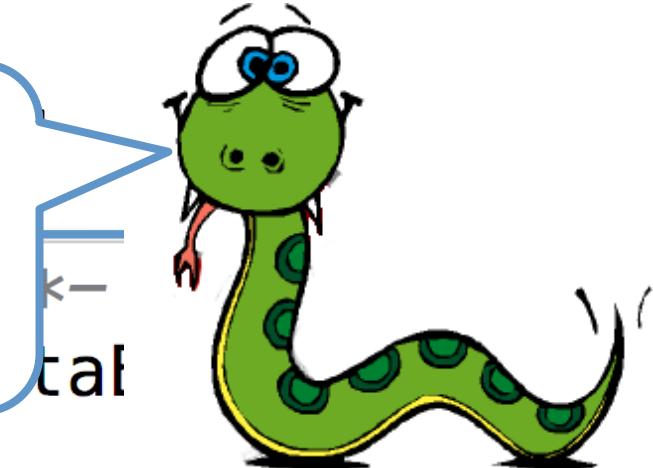


```
1 # -*- coding: UTF-8 -*-
2 from model import ContaBancaria
3
4 print 'Construtor SEM saldo'
5 conta = ContaBancaria()
6
```

Execute novamente o
script de testes

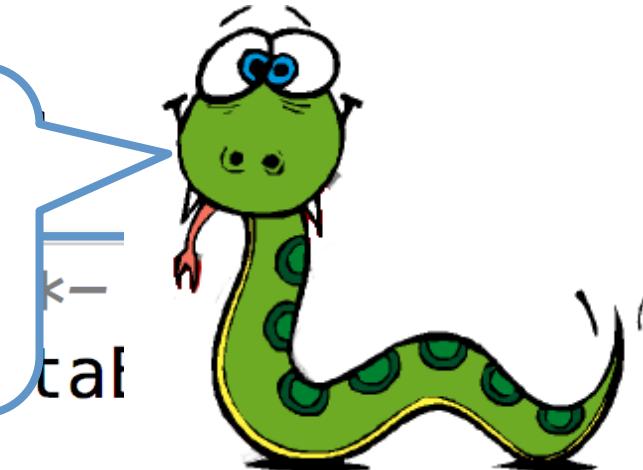
1
2
3
4
5
6

```
print 'Construtor SEM saldo'  
conta = ContaBancaria()
```



1
2
3
4
5
6

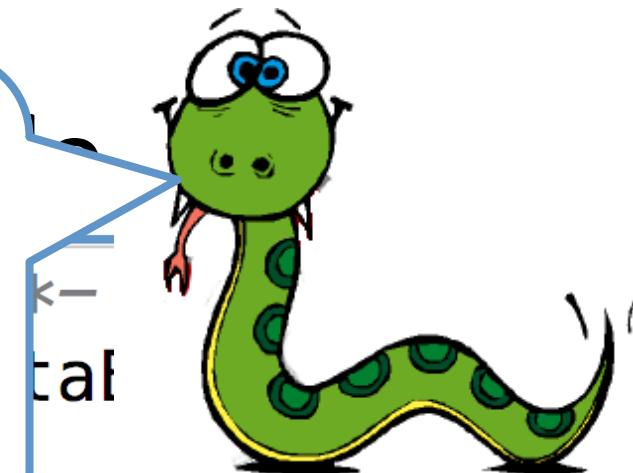
Execute novamente o
script de testes e veja que
agora ocorre um erro



```
print 'Construtor SEM saldo'  
conta = ContaBancaria()
```

```
Run teste_construtor  
/Library/Frameworks/Python.framework/Versions/2.7/bin/python  
Traceback (most recent call last):  
  File "/Users/marciopalheta/python/codigofonte/financeiro/  
    conta = ContaBancaria()  
TypeError: __init__() takes exactly 2 arguments (1 given)  
Construtor SEM saldo  
  
Process finished with exit code 1
```

O Interpretador do python te avisa que `__init__` requer 2 argumentos, mas recebeu apenas 1 (o `self`)

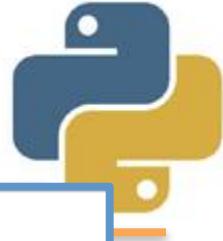


```
4     print 'CONSTRUTOR SEM saldo'
5 conta = ContaBancaria()
6
```

```
Run teste_construtor
/Library/Frameworks/Python.framework/Versions/2.7/bin/python
Traceback (most recent call last):
  File "/Users/marciopalheta/python/codigofonte/financeiro/",
    conta = ContaBancaria()
TypeError: __init__() takes exactly 2 arguments (1 given)
    Construtor SEM saldo

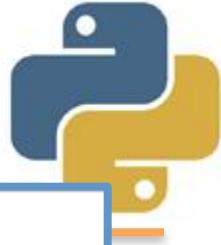
Process finished with exit code 1
```

Exercício 10: teste com saldo



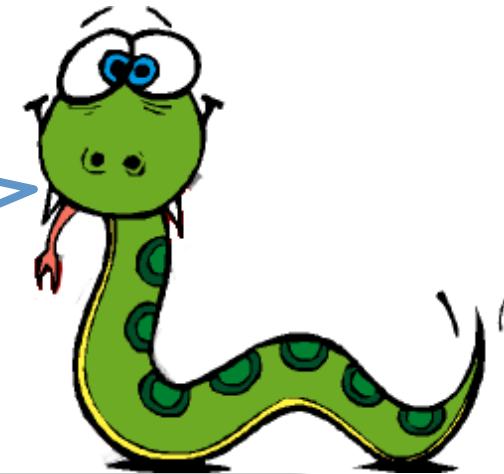
```
1 # -*- coding: UTF-8 -*-
2 # teste_construtor.py
3 from model import ContaBancaria
4
5 print 'Construtor COM saldo'
6 conta = ContaBancaria(1000.0)
7
```

Exercício 10: teste com saldo

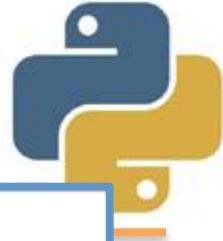


```
1 # -*- coding: UTF-8 -*-
2 # teste_construtor.py
3 from model import ContaBancaria
4
5 print 'Construtor COM saldo'
6 conta = ContaBancaria(1000.0)
7
```

Crie uma **ContaBancaria** informando que o **saldo inicial** é **1000.0** e **execute** o script



Exercício 10: teste com saldo

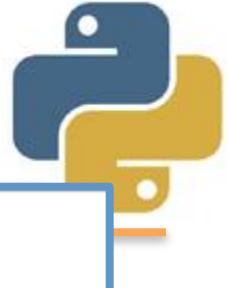


```
1 # -*- coding: UTF-8 -*-
2 # teste_construtor.py
3 from model import ContaBancaria
4
5 print 'Construtor COM saldo'
6 conta = ContaBancaria(1000.0)
7
```

Run teste_construtor

```
/Library/Frameworks/Python.framework/Versions/3.6/bin/pythonw /Users/luizmota/PycharmProjects/ExerciciosPython/ex10/teste_construtor.py
Construtor COM saldo
ContaBancaria.__init__()
Saldo inicial: 1000.0
Process finished with exit code 0
```

Exercício 10: teste com saldo



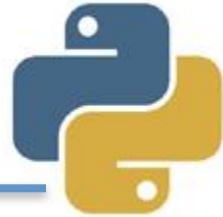
```
1 # -*- coding: UTF-8 -*-
2 # teste_construtor.py
3 from conta import ContaBancaria
4
5 print('Construtor COM saldo')
6 conta = ContaBancaria(1000)
7 print(conta.saldo)
```

Rpare que
tudo funciona
perfeitamente

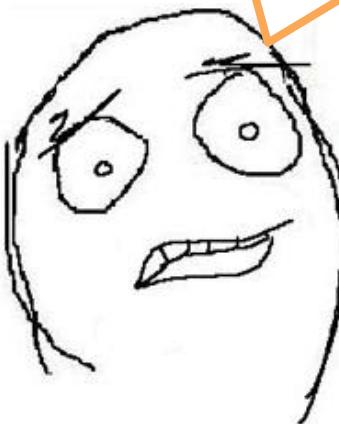


```
Run teste_construtor
/Library/Frameworks/Python.framework
Construtor COM saldo
ContaBancaria.__init__()
Saldo inicial: 1000.0
Process finished with exit code 0
```

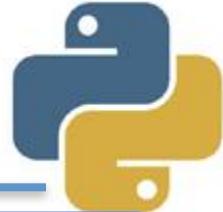
Inicialização de atributos



Entendi. Mas Poli, nem
sempre eu tenho o
saldo inicial. O que eu
poderia fazer?



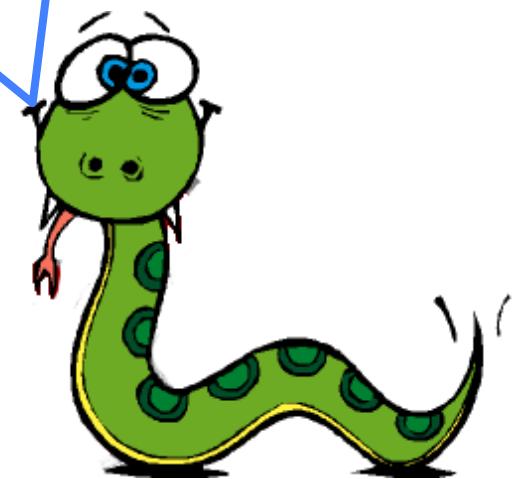
Inicialização de atributos



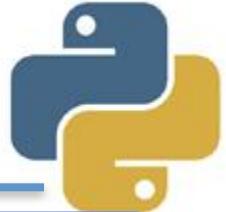
Entendi. Mas Poli, nem sempre eu tenho o saldo inicial. O que eu poderia fazer?



Morfismo, isso não é problema.



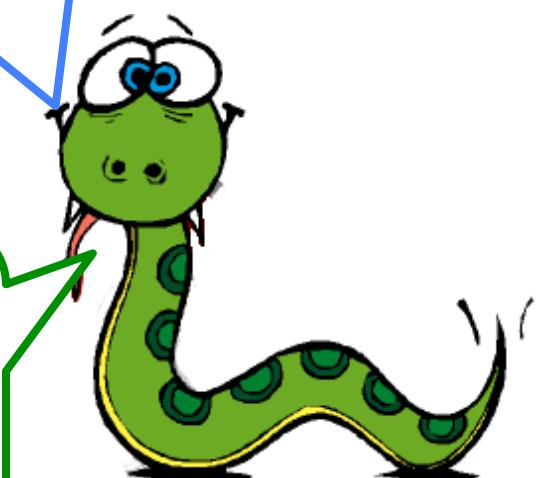
Inicialização de atributos



Entendi. Mas Poli, nem sempre eu tenho o saldo inicial. O que eu poderia fazer?

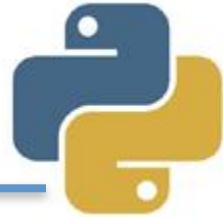


Morfismo, isso não é problema.



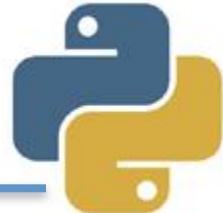
Podemos definir um valor padrão para o saldo inicial

Exercício 11: valor padrão



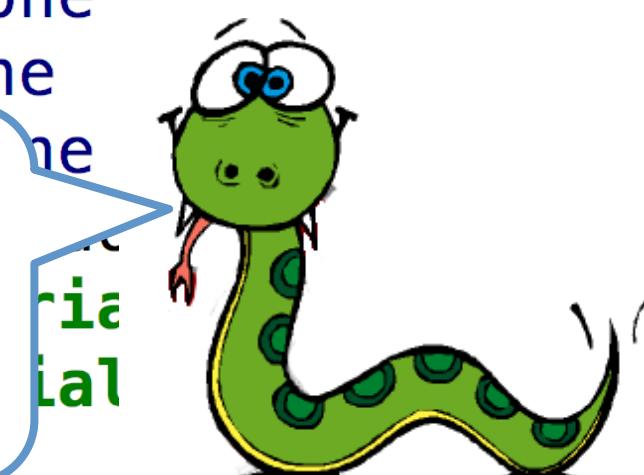
```
model.py x teste_construtor.py x
ContaN Bancaria __init__()
1 class ContaN Bancaria(object):
2     def __init__(self, saldo=0.0):
3         self.agencia = None
4         self.numero = None
5         self.cliente = None
6         self.__saldo = saldo
7         print 'ContaN Bancaria.__init__()'
8         print 'Saldo inicial:', saldo
9
```

Exercício 11: valor padrão

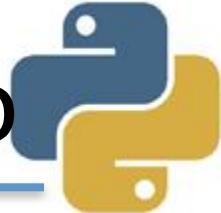


```
model.py  teste_construtor.py
1 class ContaBancaria(object):
2     def __init__(self, saldo=0.0):
3         self.agencia = None
4         self.numero = None
5
6     No método __init__,
7     defina o valor padrão
8     para o argumento saldo
9 
```

No método `__init__`,
defina o **valor padrão**
para o **argumento saldo**



Exercício 12: com e sem saldo



```
# -*- coding: UTF-8 -*-
# teste_construtor.py
from model import ContaBancaria

print 'Construtor COM saldo'
conta = ContaBancaria(1000.0)

print 'Construtor SEM saldo'
conta2 = ContaBancaria()
```

Altere o arquivo
teste_construtor.py e
teste contas com e sem
saldo inicial definido

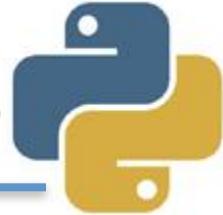
```
# - - - - -  
# t  
from model import ContaBancaria
```

 **print 'Construtor COM saldo'**
conta = ContaBancaria(**1000.0**)

print 'Construtor SEM saldo'
conta2 = ContaBancaria()



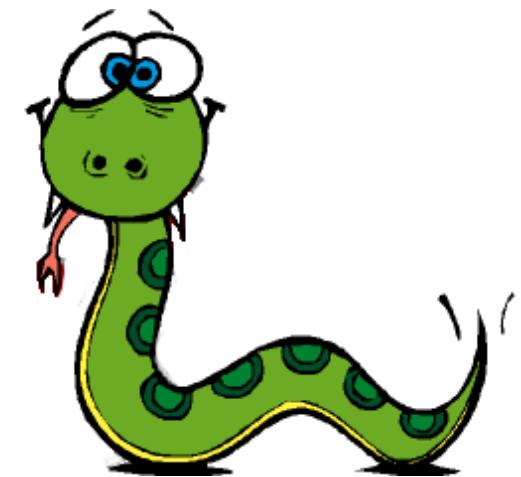
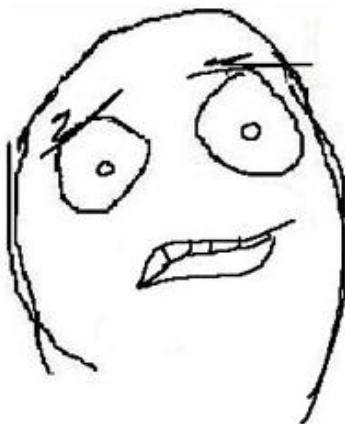
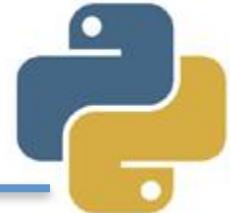
Resultado do processamento



```
Run teste_construtor
/Library/Frameworks/Python.framework
Construtor COM saldo
ContaBancaria.__init__()
Saldo inicial: 1000.0
Construtor SEM saldo
ContaBancaria.__init__()
Saldo inicial: 0.0

Process finished with exit code 0
```

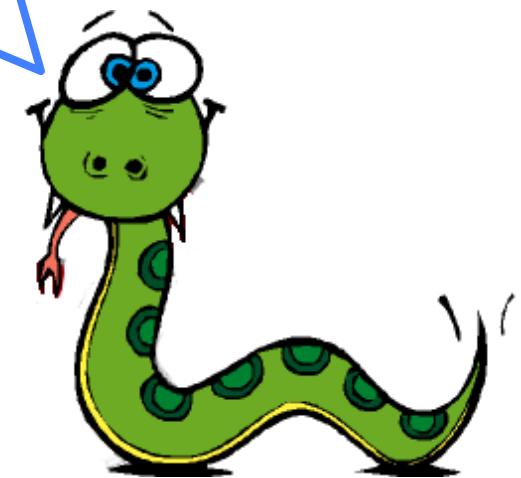
Agora, um probleminha





Agora, um probleminha

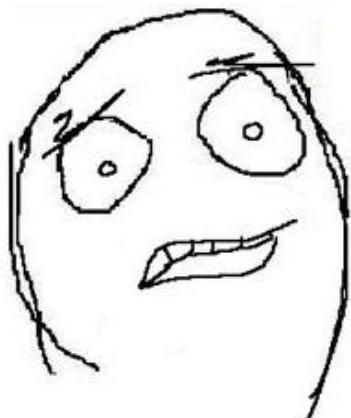
Morfismo, imagina que precisamos saber o total de contas criadas



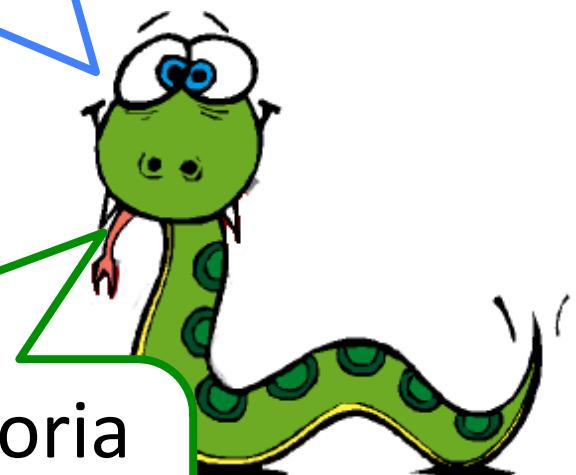


Agora, um probleminha

Morfismo, imagina que precisamos saber o total de contas criadas



Depois de toda essa teoria que discutimos, que solução você adotaria?





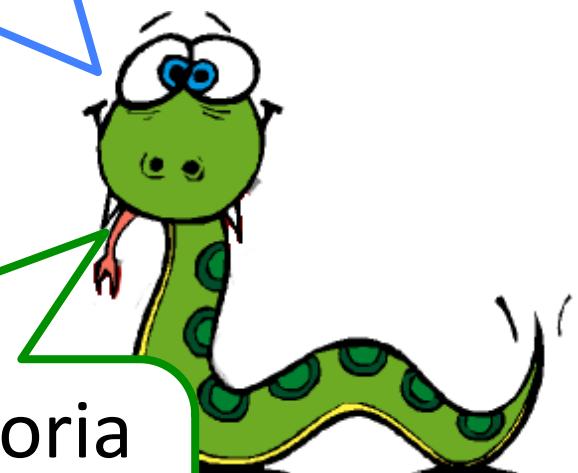
Agora, um probleminha

Poderíamos criar um **contador**, que é **atualizado** sempre que uma **conta** for criada

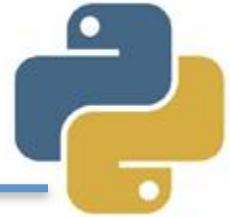
Morfismo, imagina que precisamos saber o **total de contas criadas**



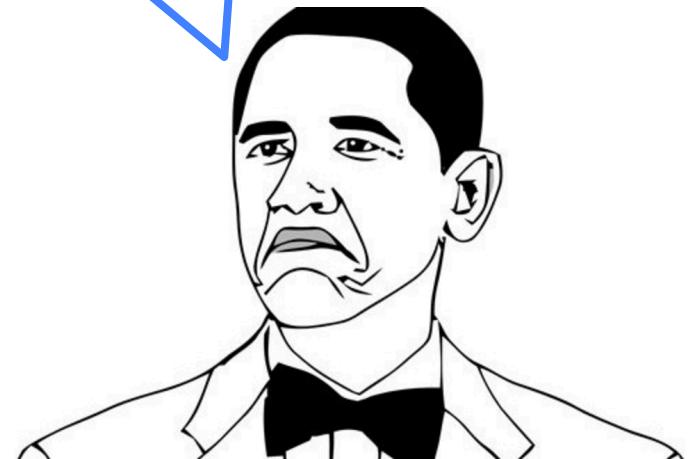
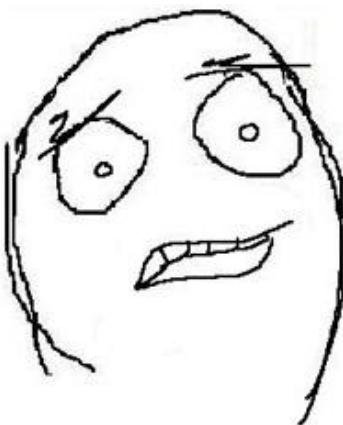
Depois de toda essa teoria que discutimos, que **solução** você **adotaria**?



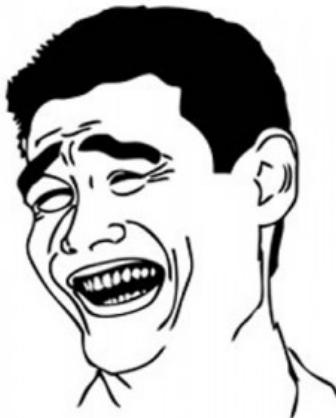
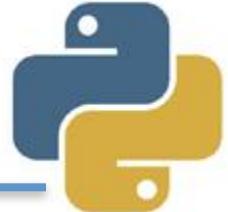
Agora, um probleminha



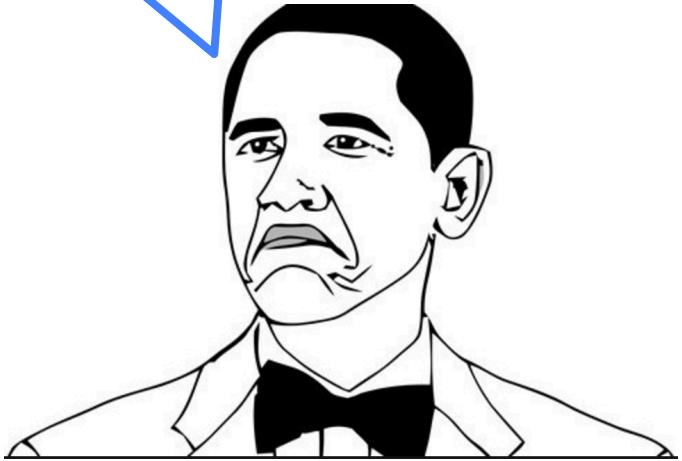
Nada mau



Agora, um probleminha

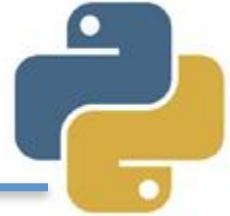


Eu sou demais!



Nada mau

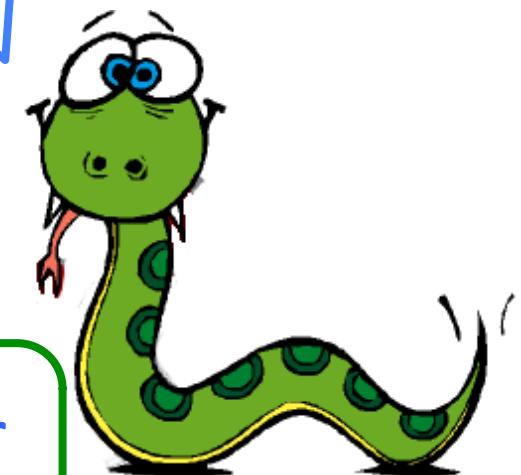
Agora, um probleminha



Eu sou demais!

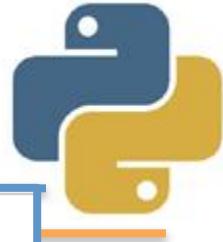


Nada mau



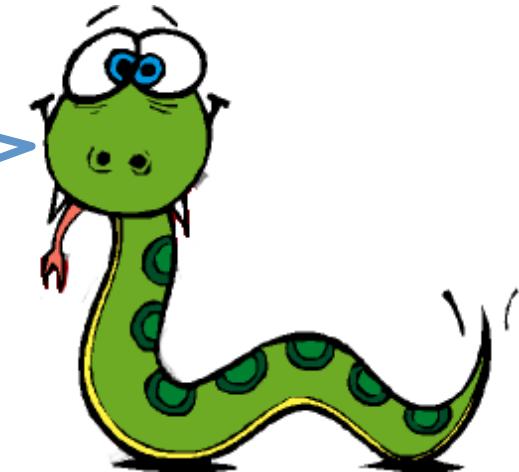
Vamos **implementar**
a sua **solução**

Exercício 13: contador



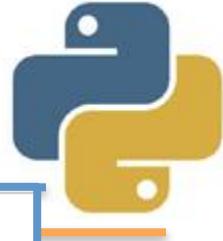
```
1 # -*- coding: UTF-8 -*-
2
3 class ContaBancaria(object):
4     def __init__(self, saldo=0.0):
5         self.agencia = None
6         self.numero = None
7         self.cliente = None
8         self.__saldo = saldo
9         #incrementa o contador
10        #sempre que uma conta eh
11        #instanciada
12        self.__contador += 1
13
14 @property
15 def contador(self):
16     return self.__contador
17
```

Cada vez que o `__init__` for chamado, precisamos incrementar o contador



```
4      def __init__(self, saldo=0.0):
5          self.agencia = None
6          self.numero = None
7          self.cliente = None
8          self.__saldo = saldo
9          #incrementa o contador
10         #sempre que uma conta eh
11         #instanciada
12         self.__contador += 1
13
14     @property
15     def contador(self):
16         return self.__contador
17
```

Exercício 13: contador

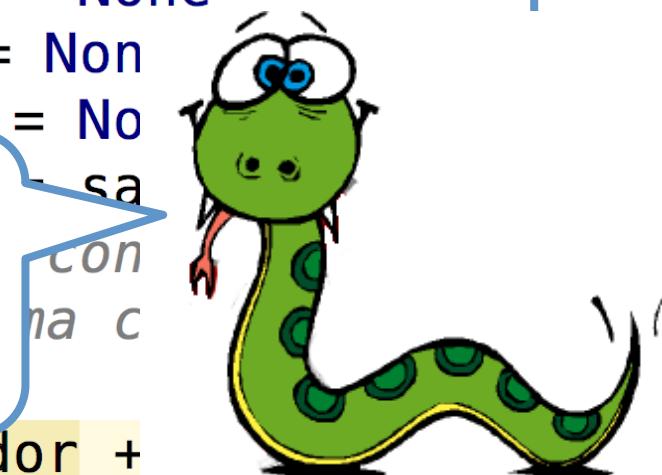


```
1 # -*- coding: UTF-8 -*-
2
3 class ContaBancaria(object):
4     def __init__(self, saldo=0.0):
5         self.agencia = None
6         self.numero = Non
7         self.cliente = No
```

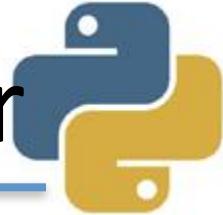
Vamos definir um **getter**
para o **contador**

```
12         self.__contador +
```

```
13
14 @property
15 def contador(self):
16     return self.__contador
17
```



Exercício 14: teste_construtor

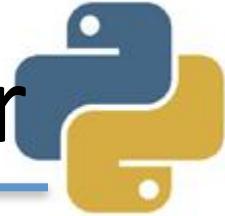


```
# -*- coding: UTF-8 -*-
# teste_construtor.py
from model import ContaBancaria

print 'Construtor COM saldo'
conta = ContaBancaria(1000.0)

print 'Construtor SEM saldo'
conta2 = ContaBancaria()
```

Exercício 14: teste_construtor



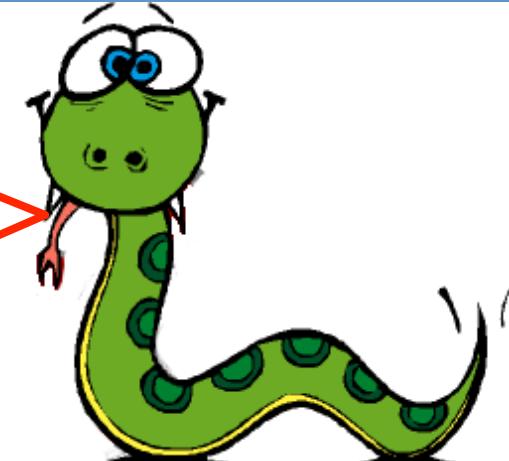
```
# -*- coding: UTF-8 -*-
```

```
# +  
# fr
```

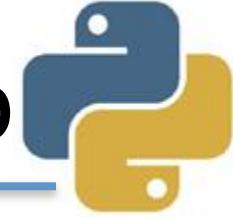
Execute novamente o
script `teste_construtor.py`

```
print 'Construtor COM saldo'  
conta = ContaBancaria(1000.0)
```

```
print 'Construtor SEM saldo'  
conta2 = ContaBancaria()
```



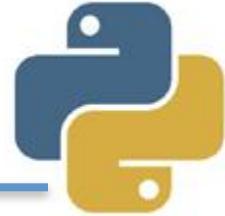
Resultado do processamento



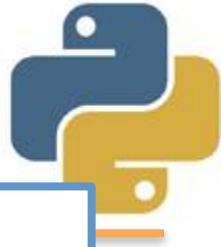
- Agora, como resultado, é lançado um **AttributeError**, informando que **contador** ainda **não é atributo** de **ContaBancaria**

```
/Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7 /Users/marciopalhe  
Construtor COM saldo  
Traceback (most recent call last):  
  File "/Users/marciopalheta/python/codigofonte/financeiro/teste_construtor.py", l:  
    conta = ContaBancaria(1000.0)  
  File "/Users/marciopalheta/python/codigofonte/financeiro/model.py", line 12, in  
    self.__contador += 1  
AttributeError: 'ContaBancaria' object has no attribute '_ContaBancaria__contador'  
Process finished with exit code 1
```

Tipos de atributos



- Em Orientação a objetos, os atributos podem ser de **dois tipos**:
 - Atributos de entidade: Pertencem aos objetos. Acesso via `self.nome_atributo`
 - Atributos de classe: Pertencem à classe. Acesso via `Classe.nome_atributo`. Chamados de atributos estáticos.
-



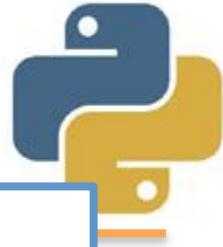
Atributos de classe

```
class Animal(object):
    # atributo da classe
    nome = 'Rex'

    def __init__(self):
        self.idade = 0

cao = Animal()
print 'Definido na classe:', cao.nome
cao.nome = 'Totó'
cao.idade = 10
print cao.nome, cao.idade, Animal.nome
Animal.nome = "Frodo"
print "Nome atualizado:", Animal.nome
```

Atributos de classe



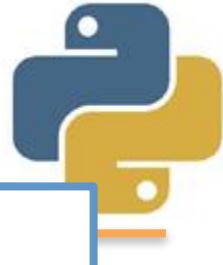
```
class Animal(object):
    # atributo da classe
    nome = 'Rex'
```

Atributos de estáticos
são **definidos fora** do
método `__init__`

```
cao.nome = 'Totó'
cao.idade = 10
print cao.nome, cao.idade, Animal.nome
Animal.nome = "Frodo"
print "Nome atualizado:", Animal.nome
```



Atributos de classe



```
class Animal(object):  
    # atributo da classe
```

Atributos de **estáticos**
podem ser **acessados** a
partir de **instâncias**

```
cao = Animal()  
print 'Definido na classe:', cao.nome  
cao.nome = 'Totó'  
cao.idade = 10  
print cao.nome, cao.idade, Animal.nome  
Animal.nome = "Frodo"  
print "Nome atualizado:", Animal.nome
```





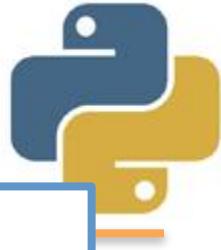
Atributos de classe

```
class Animal(object):  
    # atributo da classe  
    nome = 'Poxa'
```

Instâncias podem definir
atributos com nomes
iguais aos estáticos

```
print 'Definido na classe:', cao.nome  
cao.nome = 'Totó'  
cao.idade = 10  
print cao.nome, cao.idade, Animal.nome  
Animal.nome = "Frodo"  
print "Nome atualizado:", Animal.nome
```





Atributos de classe

```
class Animal(object):  
    # atributo da classe  
    nome = 'Rex'
```

```
def __init__(self):
```

Podemos acessar
os atributos **estáticos** a
partir da própria **classe**

```
cao.idade = 10
```

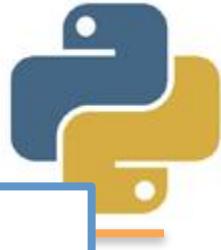
```
print cao.nome, cao.idade,
```

`Animal.nome`

```
Animal.nome = "Frodo"
```

```
print "Nome atualizado:", Animal.nome
```





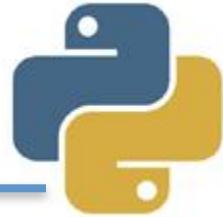
Atributos de classe

```
class Animal(object):
    # atributo da classe
    nome = 'Rex'

    def __init__(self):
        Para atualizarmos um
        # atributo estático,
        # precisamos da classe
        self.idade = 0
        print cao.nome, cao.idade, Animal.nome
        Animal.nome = "Frodo"
        print "Nome atualizado:", Animal.nome
```

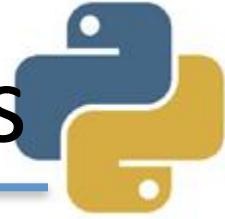


Resultado no console



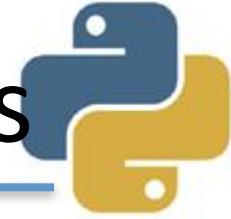
```
Run Python teste_atributo_classe
▶ /Library/Frameworks/Python.framework
Definido na classe: Rex
Totó 10 Rex
Nome atualizado: Frodo
Process finished with exit code 0
```

Atributos e métodos estáticos



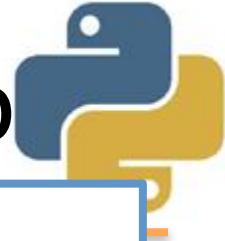
- Além de **atributos**, podemos definir **métodos estáticos**, ou seja, métodos que pertencem à **classe**
 - Podemos usar **@staticmethod** para definir um método estático
 - **Não são** muito usados, porque podem ser substituídos por **métodos de módulos**
-

Atributos e métodos estáticos



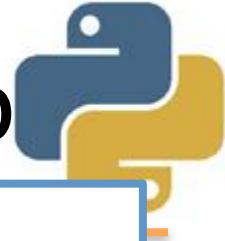
- Para **resolver o problema** do controle de contas bancárias, podemos definir o atributo **__contador** como **estático**
 - Além disso, podemos criar um **método** **estático** que nos devolva o **valor do contador**
-

Exercício 15: método estático



```
3  class ContaBancaria(object):
4      __contador = 0
5
6      def __init__(self, saldo=0.0):
7          self.agencia = None
8          self.numero = None
9          self.cliente = None
10         self.__saldo = saldo
11         #incrementa o contador
12         ContaBancaria.__contador += 1
13
14     @staticmethod
15     def contador():
16         return ContaBancaria.__contador
17
```

Exercício 15: método estático



```
3     class ContaBancaria(object):
4         __contador = 0
5
6         def __init__(self, saldo=0):
7             self.agencia = None
8
9             Defina __contador
10            como atributo estático
11
12             ContaBancaria.__co
13
14             @staticmethod
15             def contador():
16                 return ContaBancaria.__contador
17
```

A callout bubble with a blue border and a light blue arrow points from the text "Defina __contador como atributo estático" to the line of code " __contador = 0". The text inside the bubble is also blue.

A cartoon illustration of a green snake with yellow spots and large eyes, positioned to the right of the callout bubble.

Exercício 15: método estático

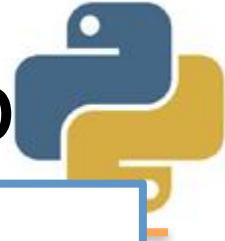


```
3 class ContaBancaria(object):
4     __contador = 0
5
6     def __init__(self, cliente, saldo):
7         self.cliente = cliente
8         self.__saldo = saldo
9
10    #incrementa o contador
11    ContaBancaria.__contador += 1
12
13
14    @staticmethod
15    def contador():
16        return ContaBancaria.__contador
17
```

Incrementa o contador
no método `__init__`

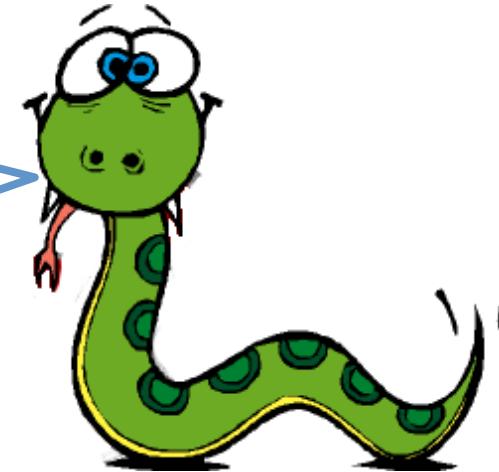


Exercício 15: método estático

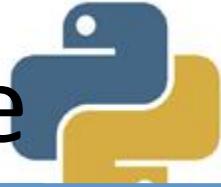


```
3  class ContaBancaria(object):
4      __contador = 0
5
6      def __init__(self, saldo=0.0):
7          self.agencia = None
8
9          self.saldo = saldo
10         self.agencia = agencia
11
12     @staticmethod
13     def contador():
14         return ContaBancaria.__contador
```

Método estático que devolve __contador



Exercício 16: teste novamente



```
1 # -*- coding: UTF-8 -*-
2 # teste_construtor.py
3 from model import ContaBancaria
4
5 print 'Construtor COM saldo'
6 conta = ContaBancaria(1000.0)
7 print conta.saldo, conta.contador(), ContaBancaria.contador()
8
9 print 'Construtor SEM saldo'
10 conta2 = ContaBancaria()
11 print conta2.saldo, conta2.contador(), ContaBancaria.contador()
```

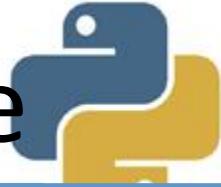
Ex

Em `teste_construtor.py`,
imprima o valor do
contador, após a **criação**
dos objetos e **execute**.



```
1 # -*-  
2 # teste_construtor.py  
3 from conta import ContaBancaria  
4  
5 print 'Construtor COM saldo'  
6 conta = ContaBancaria(1000.0)  
7 print conta.saldo, conta.contador(), ContaBancaria.contador()  
8  
9 print 'Construtor SEM saldo'  
10 conta2 = ContaBancaria()  
11 print conta2.saldo, conta2.contador(), ContaBancaria.contador()
```

Exercício 16: teste novamente



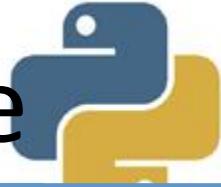
```
1 # -*- coding: UTF-8 -*-
2 # teste_construtor.py
3 from model import ContaBancaria
4
5 print 'Construtor COM saldo'
6 conta = ContaBancaria(1000.0)
7 print conta.saldo, conta.contador(), ContaBancaria.contador()
8
9 print 'Construtor SEM saldo'
10 conta2 = ContaBancaria()
11 print conta2.saldo, conta2.contador(), ContaBancaria.contador()
12
```

Run teste_construtor

```
/Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7 /U
Construtor COM saldo
1000.0 1 1
Construtor SEM saldo
0.0 2 2
```

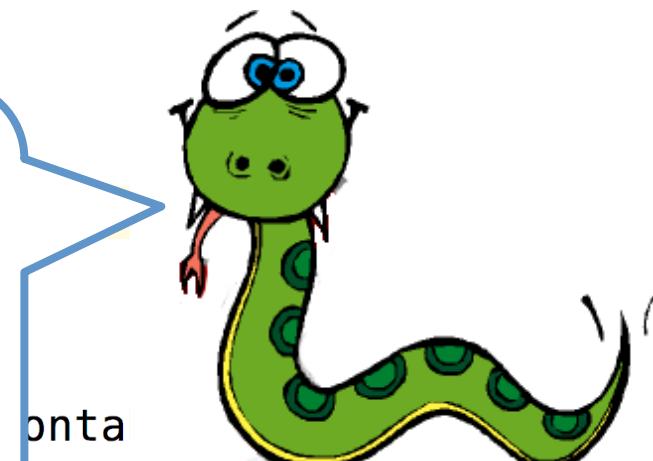
Process finished with exit code 0

Exercício 16: teste novamente



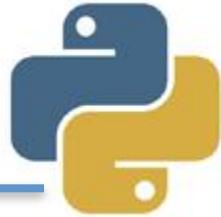
```
1 # -*- coding: UTF-8 -*-
2 # teste_construtor.py
3 from model import ContaBancaria
4
5 print
6 conta = ContaBancaria(1000.0)
7 print(conta.saldo)
8 print(conta.titular)
9 print(conta.numero)
10 print(conta.agencia)
11 print(conta)
12 print(conta.saldo)
```

Rpare que **classe** e
instâncias devolvem os
mesmos **valores** para o
contador



```
Run teste_construtor
/Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7 /U
Construtor COM saldo
1000.0 1 1
Construtor SEM saldo
0.0 2 2
```

Process finished with exit code 0



Exercício 17

- Crie a classe **Funcionário**, conforme a figura ao lado
 - A **matrícula** do funcionário deve ser **auto incremento** e **não** pode ser **alterada**
 - Na hora de criar um objeto, nome, e-mail e salário são **atributos opcionais**
 - Teste a nova classe
-

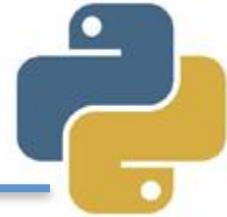
| Funcionario |
|--------------------|
| - matricula : int |
| - nome : String |
| - email : String |
| - salario : double |



FIM

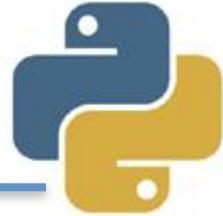


Contatos



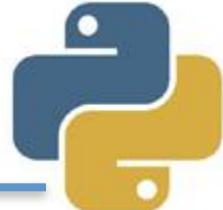
- Márcio Palheta
 - marcio.palheta@gmail.com
 - @marciopalheta
 - <https://sites.google.com/site/marciopalheta/>
-

Bibliografia



- LIVRO: Apress - Beginning Python From Novice to Professional
 - LIVRO: O'Reilly - Learning Python
 - <http://www.python.org>
 - <http://www.python.org.br>
 - Mais exercícios:
 - <http://wiki.python.org.br/ListaDeExercicios>
 - Documentação do python:
 - <https://docs.python.org/2/>
-

Capítulo 04



Encapsulamento, Atributos e Métodos de classe



Márcio Palheta, M.Sc.
marcio.palheta@gmail.com
