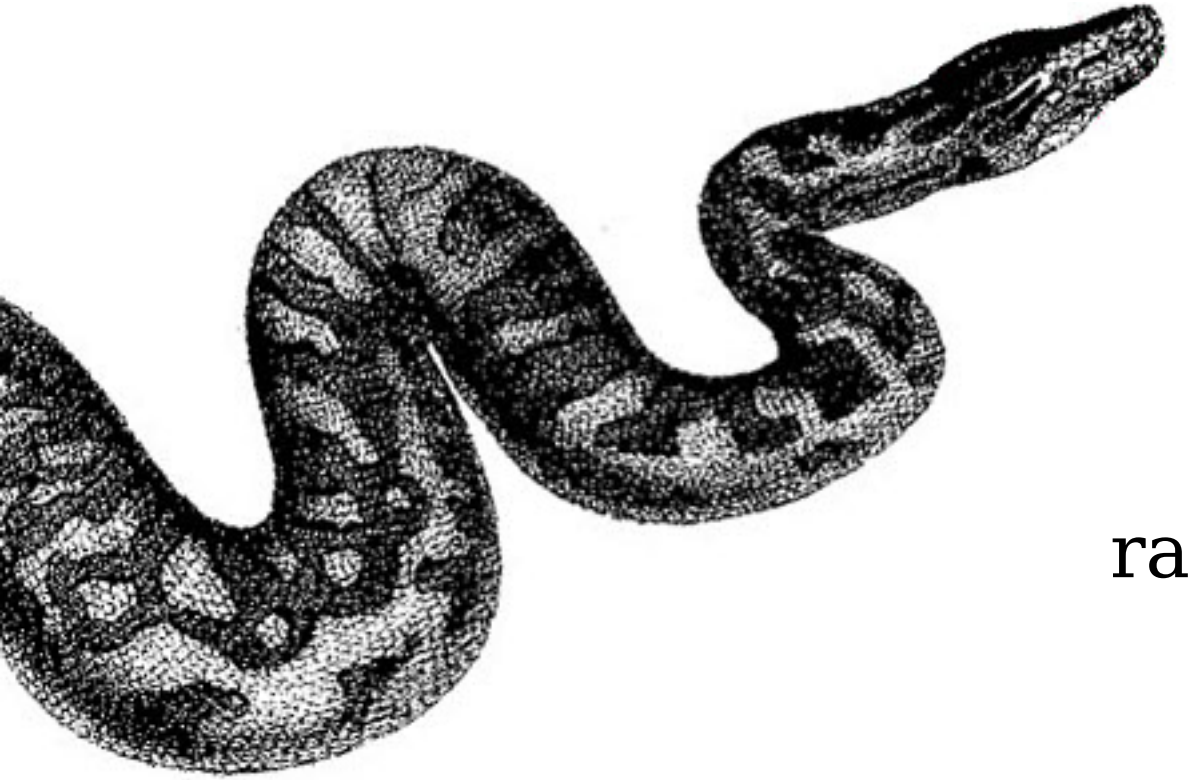
Python:
a primeira mordida



Luciano Ramalho
ramalho@python.pro.br

PythonPro

Classes

```
class Contador(object):

    def __init__(self):
        self.dic = {}

    def incluir(self, item):
        qtd = self.dic.get(item, 0) + 1
        self.dic[item] = qtd
        return qtd

    def contar(self, item):
        return self.dic[item]
```

declaração

inicializador
"construtor"

método

PythonPro

Classes “vazias”

Estilo antigo (old style)

```
class Coisa:
    pass
```

pass indica um bloco vazio

Estilo novo (new style)

```
class Coisa(object):
    pass
```

É possível definir atributos nas instâncias

```
>>> c = Coisa()
>>> c.altura = 2.5
>>> c.largura = 3
>>> c.altura, c.largura
(2.5, 3)
>>> dir(c)
['__doc__', '__module__', 'altura', 'largura']
```

PythonPro

Como extender uma classe

```
class ContadorTolerante(Contador):

    def contar(self, item):
        return self.dic.get(item, 0)
```

declaração

método sobrescrito

Como invocar métodos de super-classes:

```
class ContadorTotalizador(Contador):

    def __init__(self):
        super(ContadorTotalizador, self).__init__()
        self.total = 0

    def incluir(self, item):
        super(ContadorTotalizador, self).incluir(item)
        self.total += 1
```

Herança múltipla

```
class ContadorTolerante(Contador):
    def contar(self, item):
        return self.dic.get(item, 0)

class ContadorTotalizador(Contador):
    def __init__(self):
        super(ContadorTotalizador, self).__init__()
        self.total = 0

    def incluir(self, item):
        super(ContadorTotalizador, self).incluir(item)
        self.total += 1

class ContadorTT(ContadorTotalizador,ContadorTolerante):
    pass
```

pass indica um bloco vazio

Propriedades

Encapsulamento quando você precisa

```
>>> a = C()
>>> a.x = 10        #!!!
>>> print a.x
10
>>> a.x = -10
>>> print a.x       # ??????
0
```

PythonPro

Implementação de uma propriedade

Apenas para leitura

```
class C(object):
    def __init__(self, x):
        self.__x = x
    @property
    def x(self):
        return self.__x
```

```
class C(object):
    def __init__(self, x=0):
        self.__x = x
    def getx(self):
        return self.__x
    def setx(self, x):
        if x < 0: x = 0
        self.__x = x
    x = property(getx, setx)
```

Para leitura e escrita

Exemplo de propriedade

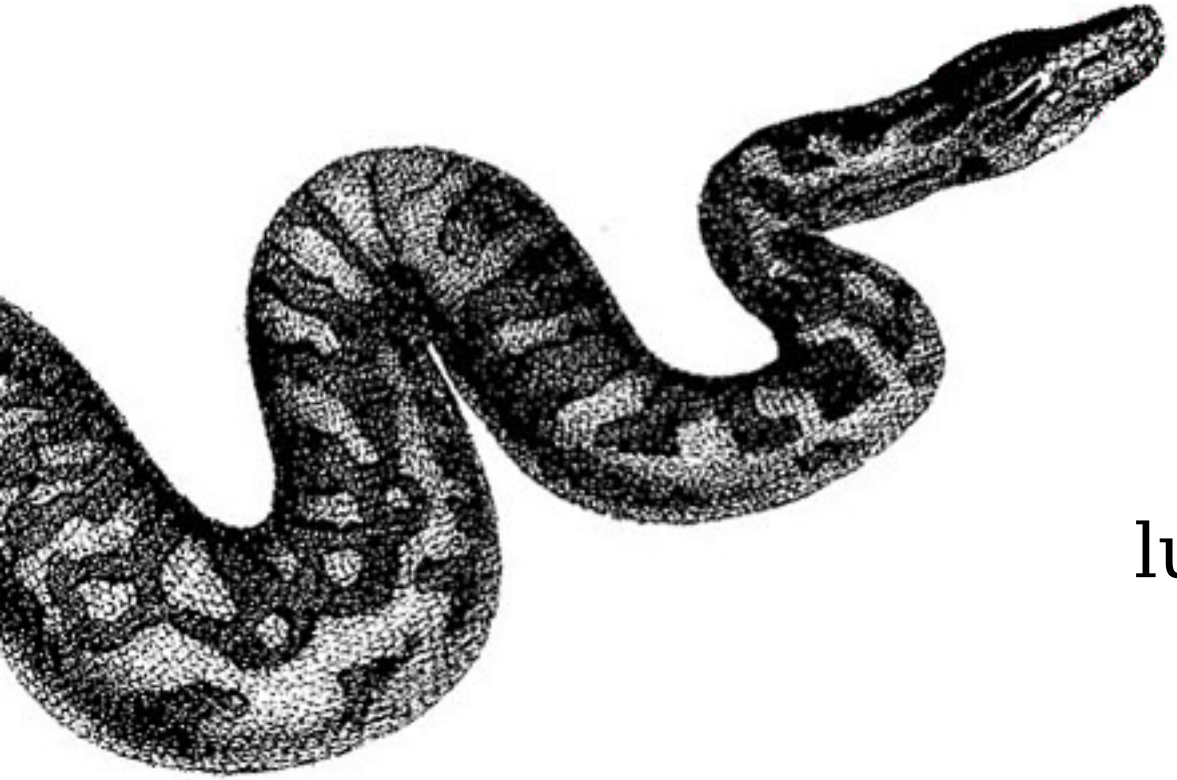
```
class ContadorTotalizador(Contador):
    def __init__(self):
        super(ContadorTotalizador, self).__init__()
        self.__total = 0

    def incluir(self, item):
        super(ContadorTotalizador, self).incluir(item)
        self.__total += 1

    @property
    def total(self):
      return self.__total
```

Funciona porque é uma classe estilo novo

extende de `Contador`, que extende `object`

PythonPro

python.mordida[0]



Luciano Ramalho
luciano@occam.com.br

PythonPro

occam