



Mini Talk de Python



Python nos dias atuais

- Desenvolvimento Web (Front End / Back End)
 - Brython, Flask, Django, Pyramid, Sanic, CherryPy, Bottle
- Aplicações Desktop e Mobile (Android / IOS)
 - Kivy, SL4A, QPython
- I.A.
 - Scikit-learn, Scipy, Numpy, TensorFlow, Anaconda
- Análise de dados e programação científica (Data science)
 - Numpy, Scikit-learn, Anaconda
- etc...



Python nos dias atuais

- Python hoje se encontra na versão **3.7.2** [24/12/2018]
- Estamos usando a versão 2.7.15 (03/07/2010) [Pequenas atualizações]



Diferenças entre as versões de Python



Print não é mais uma expressão

```
>>> ## Python 3
...
>>> print 'Olá, mundo!'
File "<stdin>", line 1
    print 'Olá, mundo!'
          ^
SyntaxError: Missing parentheses in call to 'print'. Did you mean print('Olá, mundo!')?
```

para quem ainda faz debug com print....



Iteradores por toda a parte! “Lazy Evaluation”

```
>>> ## Python 2
...
>>> numeros = range(5)
>>> numeros
[0, 1, 2, 3, 4]
>>> type(numeros)
<type 'list'>
```

```
>>> ## Python 3
...
>>> numeros = range(5)
>>> numeros
range(0, 5)
>>> type(numeros)
<class 'range'>
>>> list(numeros)
[0, 1, 2, 3, 4]
```

Python3 na maior parte do tempo vai devolver iteradores!

São mais eficientes em economia de memória e processamento!



Comparação de qualquer objeto...

```
>>> ## Python 2
...
>>> class MinhaClasse:
...     pass
...
>>> instancia = MinhaClasse()
>>>
>>> instancia < 'a'
True
>>> instancia < 1
True
>>> instancia < []
True
```

```
In [5]: '1' < 2
Out[5]: False
```

Python 2 permite comparação lógica de qualquer objeto!

Tipos agora importam!

```
>>> class A:
...     pass

>>> minha_instancia = A()

>>> minha_instancia < 'a'

-----
TypeError                                 Traceback (most recent call last)
<ipython-input-4-d345bc0dcdac> in <module>
----> 1 minha_instancia < 'a'

TypeError: '<' not supported between instances of 'A' and 'str'

>>> |
```

object agora é implicito! new style | old style

Tipos importam!

Classes em Python 3 são todas em New Style

Python 3 não é tipado mas tem inteligência sobre os tipos!



Tipos agora importam!

```
>>> '1' < 2
```

```
-----  
TypeError
```

```
Traceback (most recent call last)
```

```
<ipython-input-6-2ae56e567bff> in <module>
```

```
----> 1 '1' < 2
```

```
TypeError: '<' not supported between instances of 'str' and 'int'
```



Tipos numéricos

```
>>> ## Python 2
...
>>> import sys
>>>
>>> maior_int = sys.maxint
>>> maior_int
9223372036854775807
>>>
>>> type(maior_int)
<type 'int'>
>>>
>>> maior_int + 1
9223372036854775808L
>>>
>>> type(maior_int + 1)
<type 'long'>
```

```
>>> maior_int = 2147483647

>>> maior_int + 1
2147483648

>>> # Python 3|
```

Os tipos numéricos em Python 3 são int e float e complex



Divisão

```
>>> 4 / 2  
2
```

```
>>> 4 // 2  
2
```

```
>>> # Python 2
```

```
>>> 4 / 2  
2.0
```

```
>>> 4 // 2  
2
```

```
>>> # Python 3|
```

Python 3 sempre devolve um float nas divisões.
Para inteiros utiliza se o operador “//”

True e False agora são palavras reservadas!

Em Python 2.x, era possível fazermos coisas bizarras como:

```
1 >>> True = "Hello"
2 >>> False = "Hello"
3 >>> True == False
4 True
```

Ou então, tão estranho quanto:

```
1 >>> False = True
2 >>> True == False
3 True
```

Felizmente, em Python 3 isso não é mais possível. Veja uma tentativa:

```
1 >>> True = "Hello"
2 SyntaxError: assignment to keyword
```



Pesquisa em dicionários

Em Python 2.x, era comum verificar se um determinado elemento já era chave em um dicionário usando o método `has_key()`:

```
1 >>> d = {'nome': 'jose', 'idade': 18}
2 >>> d.has_key('nome')
3 True
4 >>> d.has_key('email')
5 False
```

Em Python 3.x, para fazer a mesma verificação, usamos o operador `in`:

```
1 >>> 'nome' in d
2 True
3 >>> 'email' in d
4 False
```

Sempre utilizar o operador “IN”



Agora vem a melhor parte



'Strings' em Python 2

```
>>> ## Python 2
...
>>> texto_normal = 'Texto normal'
>>> type(texto_normal)
<type 'str'>
>>>
>>> texto_unicode = u'Texto unicode'
>>> type(texto_unicode)
<type 'unicode'>
```

Não existe tipo string em Python 2 ou é byte ou é unicode
Mas byte é string... e aí começa a confusão!

- Não existe um tipo próprio para bytes.



Um velho amigo...

```
Traceback (most recent call last):
  File "c:\python27\lib\runpy.py", line 162, in _run_module_as_main
    "__main__", fname, loader, pkg_name)
  File "c:\python27\lib\runpy.py", line 72, in _run_code
    exec code in run_globals
  File "C:\Python27\Scripts\virtualenv.exe\__main__.py", line 9, in <module>
  File "c:\python27\lib\site-packages\virtualenv.py", line 708, in main
    symlink=options.symlink)
  File "c:\python27\lib\site-packages\virtualenv.py", line 917, in create_environment
    home_dir, lib_dir, inc_dir, bin_dir = path_locations(home_dir)
  File "c:\python27\lib\site-packages\virtualenv.py", line 971, in path_locations
    ret = GetShortPathName(u(home_dir), buf, size)
UnicodeDecodeError: 'ascii' codec can't decode byte 0xe9 in position 20: ordinal not in range(128)
```




Sem mais problemas de Unicode :)

```
>>> ## Python 3
...
>>> números = [1,2,3]
>>> números
[1, 2, 3]
>>>
>>> π = 3.14
>>> π
3.14
```

Em Python 3 todas as strings são Unicode por padrão

Reparem na variável PI



Sem mais problemas de Unicode :)

```
>>> ## Python 3
...
>>> texto_unicode = 'Texto em unicode'
>>> type(texto_unicode)
<class 'str'>
>>>
>>> texto_bytes = b'Texto em bytes'
>>> type(texto_bytes)
<class 'bytes'>
```

- Bytes tem seu próprio tipo.
- Strings são objetos do tipo string mesmo sem problemas de unicode



Re aprendendo Python com boas práticas
de codificação.



PEP 20 - Zen do Python

O Zen do Python, por Tim Peters

1. Bonito é melhor que feio.
2. Explícito é melhor que implícito.
3. Simples é melhor que complexo.
4. Complexo é melhor que complicado.
5. Linear é melhor do que aninhado.
6. Esparso é melhor que denso.
7. Legibilidade conta.
8. Casos especiais não são especiais o bastante para quebrar as regras.
9. Ainda que praticidade vença a pureza.
10. Erros nunca devem passar silenciosamente.
11. A menos que sejam explicitamente silenciados.
12. Diante da ambiguidade, recuse a tentação de adivinhar.
13. Deveria haver um — e preferencialmente só um — modo óbvio para fazer algo.
14. Embora esse modo possa não ser óbvio a princípio a menos que você seja holandês.
15. Agora é melhor que nunca.
16. Embora nunca frequentemente seja melhor que já.
17. Se a implementação é difícil de explicar, é uma má ideia.
18. Se a implementação é fácil de explicar, pode ser uma boa ideia.
19. *Namespaces* são uma grande ideia — vamos ter mais dessas!

(para sempre lembrar dessas dicas, escreva `import this` no interpretador!)



Gerenciando versões e pacotes de Python

- PyEnv
- virtualenv e virtualenvwrapper
- Pyenv VirtualenvWrapper
- PIP

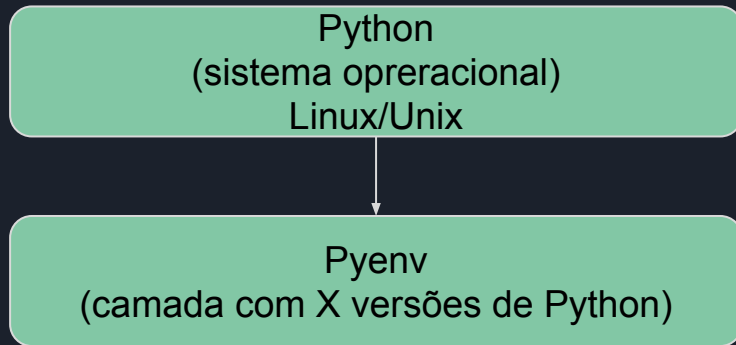


Entendo o modelo do ambiente

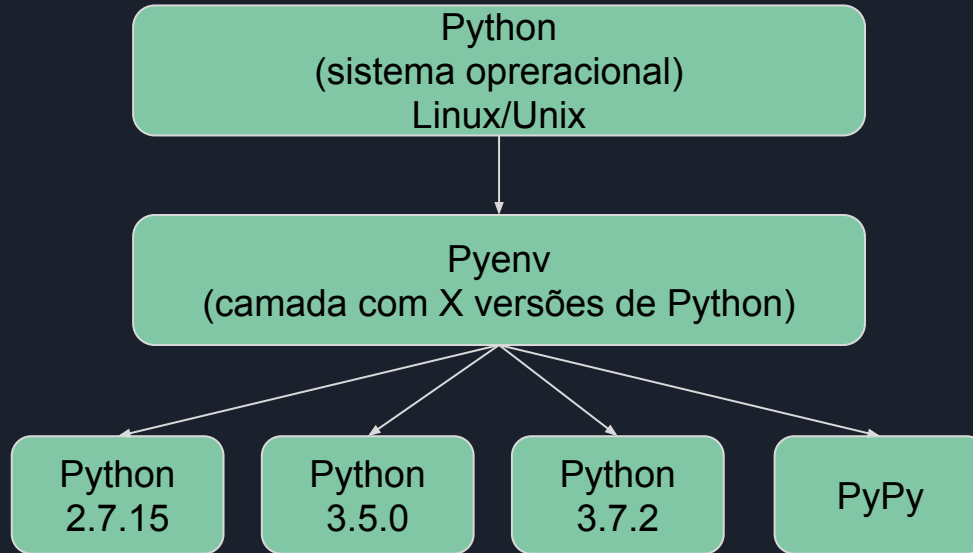
Python
(sistema operacional)
Linux/Unix



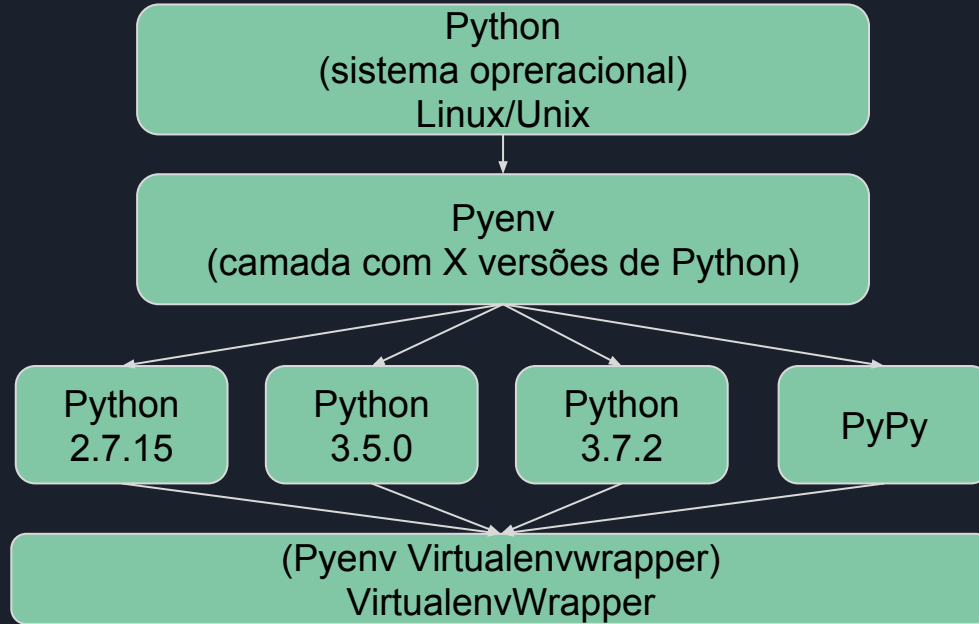
Entendo o modelo do ambiente



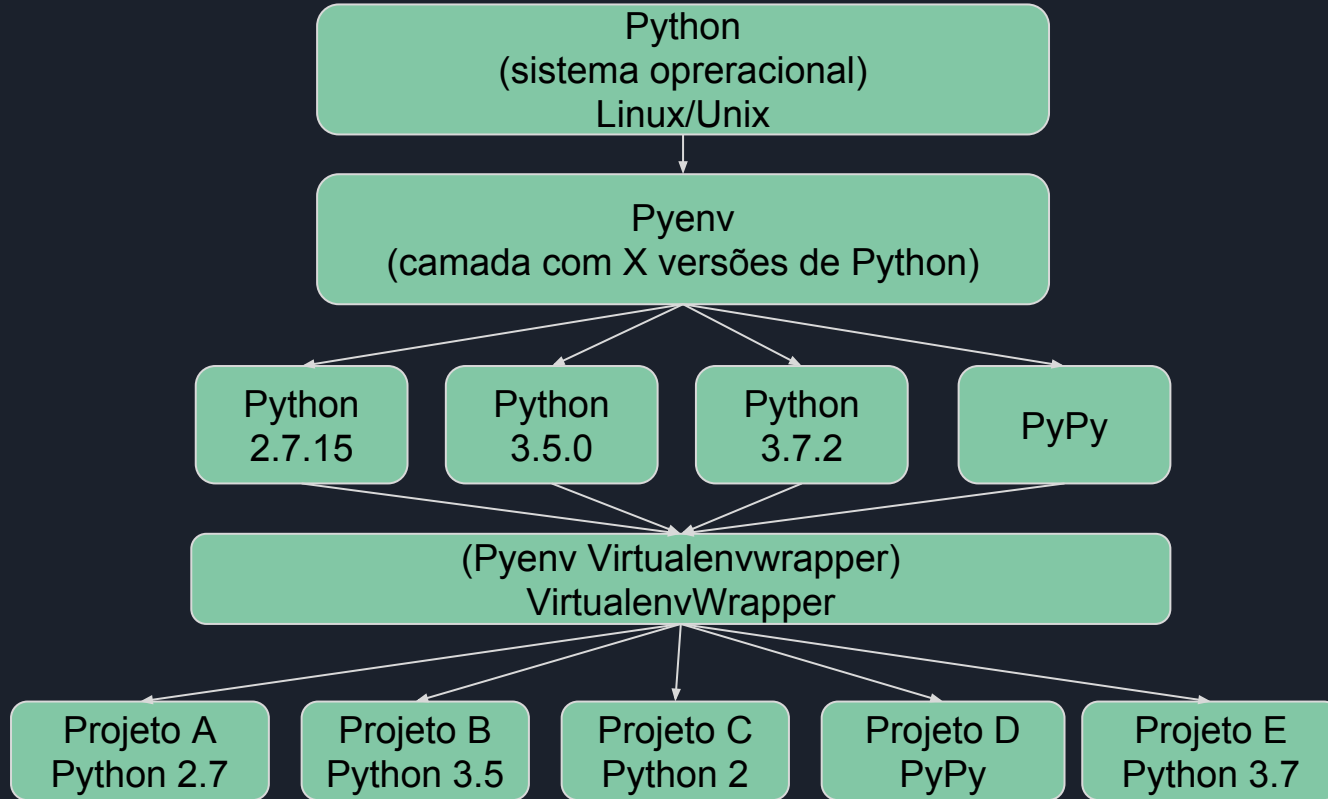
Entendo o modelo do ambiente



Entendo o modelo do ambiente

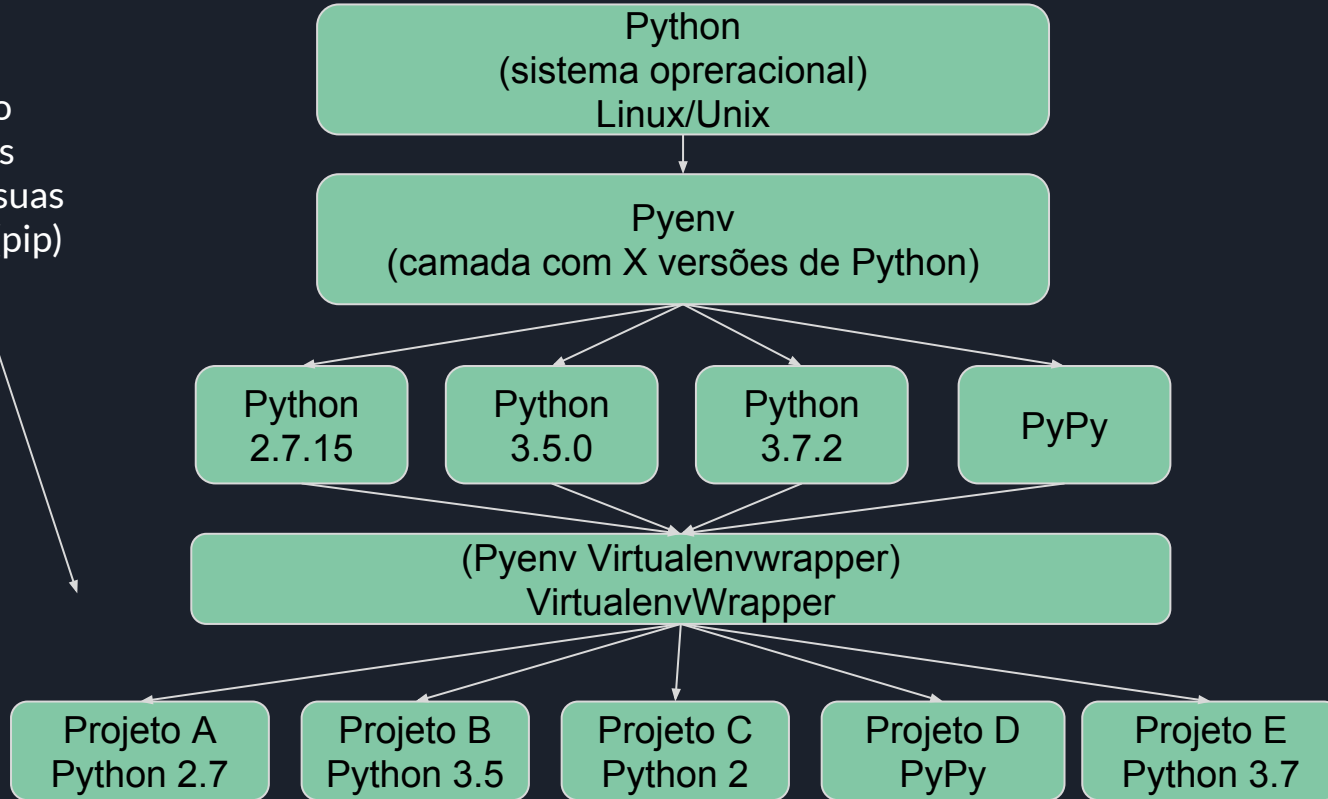


Entendo o modelo do ambiente



Entendo o modelo do ambiente

Cada projeto
mantém seus
pacotes em suas
virtualenvs (pip)






Parece muito bom mas como eu uso isso?



PEP 8



**KEEP
CALM
AND
FOLLOW
PEP 8**



O que isso? PEP o que?

- Um guia de estilos para os desenvolvedores poder codificar melhor
- Tem influência da PEP 20 - “Readability counts” *“Readability counts”*



Use 4 espaços para
indentação!

A maioria das IDEs,
editores de texto já
fazem isso!

Yes:

```
# Aligned with opening delimiter.
foo = long_function_name(var_one, var_two,
    .....var_three, var_four)

# More indentation included to distinguish this from the rest.
def long_function_name(
    .....var_one, var_two, var_three,
    .....var_four):
    ....print(var_one)

# Hanging indents should add a level.
foo = long_function_name(
    ....var_one, var_two,
    ....var_three, var_four)
```

No:

```
# Arguments on first line forbidden when not using vertical alignment.
foo = long_function_name(var_one, var_two,
    ....var_three, var_four)

# Further indentation required as indentation is not distinguishable.
def long_function_name(
    ....var_one, var_two, var_three,
    ....var_four):
    ....print(var_one)
```



Exemplos:

Python 3 não
permite misturar
tabs e espaços!

```
my_list = [  
    1, 2, 3,  
    4, 5, 6,  
]  
result = some_function_that_takes_arguments(  
    'a', 'b', 'c',  
    'd', 'e', 'f',  
)
```

```
my_list = [  
    1, 2, 3,  
    4, 5, 6,  
]  
result = some_function_that_takes_arguments(  
    'a', 'b', 'c',  
    'd', 'e', 'f',  
)
```




Respeite!

Tamanho máximo de código em uma linha deve ser 79 caracteres!

```
with open('/path/to/some/file/you/want/to/read') as file_1, \
.... open('/path/to/some/file/being/written', 'w') as file_2:
... file_2.write(file_1.read())
```

Caso precise use a barra invertida para continuar a linha




Operadores devem
ficar sempre à
esquerda

Para cenários de
cálculos compridos

```
# No: operators sit far away from their operands
income = (gross_wages +
..... taxable_interest +
..... (dividends - qualified_dividends) -
..... ira_deduction -
..... student_loan_interest)
```

```
# Yes: easy to match operators with operands
income = (gross_wages
..... + taxable_interest
..... + (dividends - qualified_dividends)
..... - ira_deduction
..... - student_loan_interest)
```



Imports

Imports should usually be on separate lines, e.g.:

Yes:

```
import os  
import sys
```

No:

```
import os, sys
```

It's okay to say this though:

```
from subprocess import Popen, PIPE
```



Importando do jeito certo

Os imports devem seguir uma ordem:

1. Bibliotecas do Python
2. Bibliotecas de terceiros instaladas via PIP
3. Bibliotecas locais criadas pelo desenvolvedor

Deve ter uma linha de espaço entre cada sequência de imports



Nomenclatura de variáveis

- Prefira nomes que sejam bem explicativos por exemplo uma variável de nome
- conta VS conta_corrente
- Evite usar nomes CamelCase para variáveis no Python utilize sempre snake_case
- **Somente ao criar Classes se utiliza CamelCase**
- Constantes devem ser sempre com todas as letras maiúscula

Criando e documentando do jeito certo

Observem a
documentação
:)

```
1 # coding: utf-8
2
3 # Exemplo de constante
4 PIZZARIA = "Domino's Pizza"
5
6
7 class Pizza:
8     """Representação de uma Pizza
9     """
10    def __init__(self, sabor, fatias=8):
11        """
12        Args:
13            sabor: (string) sabor da pizza
14            fatias: (int) fatias da pizza
15        """
16        self.sabor = sabor
17        self.fatias = fatias
18
19    # nomes explicito no que faz
20    @classmethod
21    def criar_pizza_quatro_queijos(cls, fatias=8):
22        """
23        Metodo que cria somente pizzas de quatro queijos
24        Args:
25            fatias: (int) fatias da pizza
26
27        Returns:
28            Retorna um novo objeto do tipo Pizza com sabor de quatro queijos
29        """
30        # cls é bom para criar objetos customizados!
31        return cls("quatro queijos", fatias)
32
33
34 # nome explicito do que faz não importa se ficar grande!
35 valor_total_da_compra = 50.0
36
37 # NUNCA FAÇA ISTO EM Python
38 valorTotalDaCompra = 50.0 # isto não é JavaScript!
```



Exemplo de tipos de métodos e seus usos

```
1  # coding: utf-8
2
3
4  class A:
5      # metodo que os programadores usam com cuidado
6      def _metodo_interno(self):
7          pass
8
9      # metodo interno da classe
10     def __metodo_privado(self):
11         pass
12
13     # exemplo de metodo statico
14     @staticmethod
15     def metodo_statico():
16         pass
17
18
19 # um uso comum de metodos staticos
20 class Util:
21     @staticmethod
22     def cortar_lenha():
23         pass
24
25     @staticmethod
26     def acender_a_fogueira():
27         pass
```

Exemplo de tipos de métodos e seus usos

Um método privado
não tão privado
assim!

```
In [6]: class A:
...:     # metodo que os programadores usam com cuidado
...:     def _metodo_interno(self):
...:         print("metodo interno")
...:
...:     # metodo interno da classe
...:     def __metodo_privado(self):
...:         print("metodo privado")
...:
...:     # exemplo de metodo statico
...:     @staticmethod
...:     def metodo_statico():
...:         print("metodo statico")
...:

In [7]: a = A()

In [8]: a._metodo_interno()
metodo interno

In [9]: a.__metodo_privado()
-----
AttributeError                                Traceback (most recent call last)
<ipython-input-9-370147355bc6> in <module>
----> 1 a.__metodo_privado()

AttributeError: 'A' object has no attribute '__metodo_privado'

In [10]: a._A__metodo_privado()
metodo privado

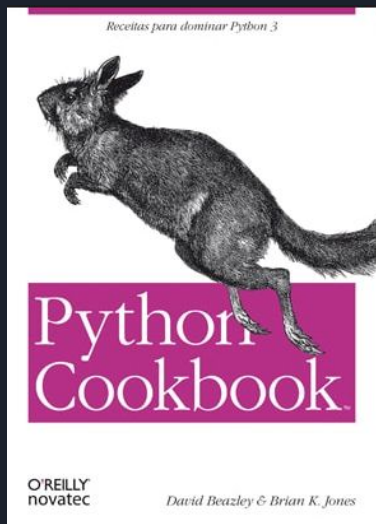
In [11]: a.metodo_statico()
metodo statico

In [12]: A.metodo_statico()
metodo statico

In [13]:
```


Para aprender mais...

3 Livros que todo Pythonista deve ler!





Para aprender mais...

Canais no Youtube

- Live de Python / Eduardo Mendes
- Python para Zumbis
- Ignorância Zero
- Pycon
- Python Brazil
- Luciano Ramalho
- PyCursos

Sites

- Python Help
- NewsLetters de Python (Chega no e-mail)
- PythonClub
- <https://wiki.python.org.br>
- docs.python.org



Obrigado!

:)



Referências Bibliográficas

<https://pep8.org/>

<https://ericstk.wordpress.com/2014/09/30/python-powered-coisas-que-python-pode-fazer-e-voce-nao-sabia/>

<https://www.profissionaisti.com.br/2009/01/10-motivos-para-voce-aprender-a-programar-em-python/>

<https://medium.freecodecamp.org/essential-libraries-for-machine-learning-in-python-82a9ada57aeb>

<https://pythonhelp.wordpress.com/2013/09/01/o-que-mudou-no-python-3/>

<https://we.riseup.net/python/o-zen-do-python>



Referências Bibliográficas

<http://blog.caelum.com.br/quais-as-diferencas-entre-python-2-e-python-3/>