Introdução ao Python

Roberto W. Nóbrega

Instituto Federal de Santa Catarina
Campus São José

Café com Linux
21 de maio de 2015

Introdução

Histórico

Guido van Rossum

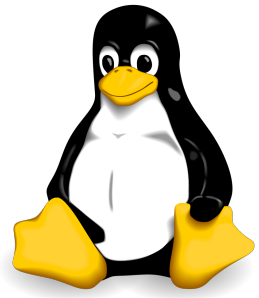
BDFL (*benevolent dictator for life*)

- 1989: primeira implementação
- 1994: versão 1.0
- 2000: versão 2.0
- 2008: versão 3.0
  (não compatível com a versão 2)
- 2014: versão 3.4
  (atual)
- 2015: versão 3.5
  (prevista para setembro)

Licença e implementação

- Python Software Foundation License (PSFL)
  (aprovada pela FSF e pela OSI; compatível com GPL)

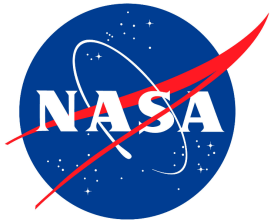
- Multi-plataforma



Características básicas

- Interpretada
- Multi-propósito
  - Scripting
  - Programação web
  - Interfaces gráficas (GUI)
  - Aplicações científicas
- Multi-paradigma
  - Orientada a objetos
  - Imperativa
  - Funcional

Quem usa Python?



https://wiki.python.org/moin/OrganizationsUsingPython
http://en.wikipedia.org/wiki/List_of_Python_software

The Zen of Python, por Tim Peters

Bonito é melhor que feio.

Explícito é melhor que implícito.

Simples é melhor que complexo.

Complexo é melhor que complicado.

Plano é melhor que aglomerado.

Esparso é melhor que denso.

Original: `https://www.python.org/dev/peps/pep-0020/`
Tradução: `http://wiki.python.org.br/TheZenOfPython`

The Zen of Python, por Tim Peters

Legibilidade faz diferença.
Casos especiais não são especiais o bastante para quebrar as regras.
Embora a praticidade vença a pureza.
Erros nunca devem passar silenciosamente.
A menos que sejam explicitamente silenciados.
Diante da ambiguidade, recuse a tentação de adivinhar.

Original: `https://www.python.org/dev/peps/pep-0020/`
Tradução: `http://wiki.python.org.br/TheZenOfPython`

The Zen of Python, por Tim Peters

Deve haver um -- e preferencialmente só um -- modo óbvio para fazer algo.
Embora esse modo possa não ser óbvio à primeira vista a menos que você seja holandês.
Agora é melhor que nunca.
*Embora nunca freqüentemente seja melhor que *exatamente* agora.*
Se a implementação é difícil de explicar, é uma má ideia.
Se a implementação é fácil de explicar, pode ser uma boa ideia.

Original: `https://www.python.org/dev/peps/pep-0020/`
Tradução: `http://wiki.python.org.br/TheZenOfPython`

Olá, mundo!

Java:

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```

Python:

```
print('Hello, world!')
```

Indentação

C:

```
if (condição_A) {
    comando_1;
    if (condição_B) {
        comando_2;
    }
}
else if (condição_C) {
    comando_3;
}
else {
    comando_4;
}
comando_5;
```

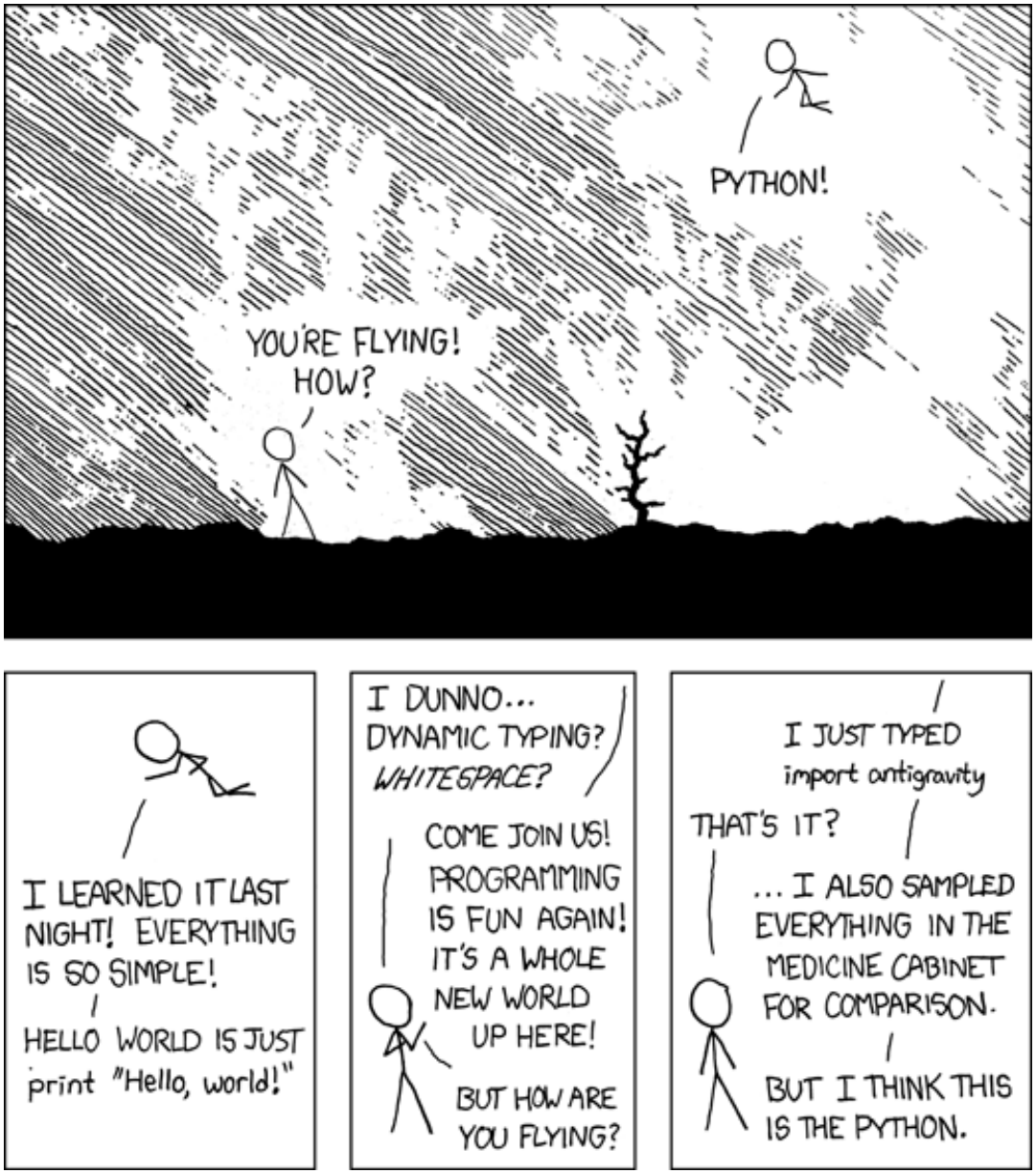
Python:

```
if condição_A:
    comando_1
    if condição_B:
        comando_2
elif condição_C:
    comando_3
else:
    comando_4
comando_5
```

Indentação é **obrigatória**!

Recomendação: sempre 4 espaços, nunca tab.

xkcd 353, por Randall Munroe



Python básico

Atribuição de variáveis

```
>>> a = 10
>>> a
10

>>> x, y = a - 1, a + 1
>>> x
9
>>> y
11

>>> x, y = y, x  # Swap
>>> x
11
>>> y
9
```

Tipos de dados padrão: Básicos

- Booleano
  - `bool : False`
- Numéricos
  - `int : -10`
  - `float : 6.28`
  - `complex : 1.0 - 4.2j`
- String
  - `str : 'Café com Linux'`

Booleanos

```
>>> not True
False

>>> True or False
True

>>> True and False
False

>>> 0 == 1
False
```

Inteiros

```
>>> a = 150
>>> a + 2
152

>>> a - 2
148

>>> 2 * a
300

>>> a / 2  # Resultado de '/' é sempre float!
75.0

>>> 2**a   # Exponenciação (** e não ^)
1427247692705959881058285969449495136382746624
```

Floats

```
>>> import math
>>> tau = 2 * math.pi
>>> tau
6.283185307179586

>>> math.sin(tau/4)
1.0

>>> 3 * 0.1        # Python não tenta te enganar
0.30000000000000004
```

Inteiros vs floats: Sutilezas

```
>>> 1 + 1       # Resultado é int
2
>>> 1.0 + 1     # Resultado é float
2.0
>>> 7 / 4
1.75
>>> 4 / 4       # Resultado de '/' é sempre float
1.0
>>> 7 // 4      # Divisão inteira
1
>>> 7 % 4       # Resto da divisão por 4
3
```

Inteiros vs floats: Conversões

```
>>> int(1.9)    # Conversão para int
1
>>> float(1)    # Conversão para float
1.0

>>> math.trunc(1.9), math.trunc(-1.9)  # Equivalente a int()
(1, -1)
>>> math.floor(1.9), math.floor(-1.9)
(1, -2)
>>> math.ceil(1.9), math.ceil(-1.9)
(2, -1)
>>> round(1.9), round(-1.9)
(2, -2)
```

Inteiros vs floats: Mais sobre round

```
>>> round(0.5), round(1.5), round(2.5), round(3.5)
... # Em caso de empate: arredonda para tornar par
(0, 2, 2, 4)

>>> round(1234.25)      # sem 2º argumento: retorna int
1234
>>> round(1234.25, 0)   # com 2º argumento: retorna float
1234.0

>>> round(1234.25, 1)
1234.2
>>> round(1234.25, -1)
1230.0
```

Operações de comparação

```
>>> 2**10 == 1024
True

>>> (1 + 2 + 3 + 4 + 5 + 6) != (6 * 7) // 2
False

>>> (tau/2 > 3, tau/2 <= 3.14)
(True, False)

>>> 3 * 0.1 == 0.3  # Extremo cuidado ao comparar floats!
False

>>> x = 0.75
>>> 0 <= x < 1   # Equivalente a (0 <= x) and (x < 1)
True
```

Numeros complexos

```
>>> z = 1/2 - 17j  # Alternativamente: complex(1/2, -17)
>>> z
(0.5-17j)

>>> z.real
0.5
>>> z.imag
-17.0

>>> from cmath import polar, rect
>>> polar(1 + 1j)
(1.4142135623730951, 0.7853981633974483)

>>> rect(1, tau/8)
(0.7071067811865476+0.7071067811865475j)
```

Strings

```
>>> 'Café com Linux'  # Unicode!
'Café com Linux'
>>> "Café com Linux"  # Aspas duplas também funcionam.
'Café com Linux'

>>> "Aspas \"duplas\" no string"
'Aspas "duplas" no string'
>>> 'Aspas "duplas" no string'
'Aspas "duplas" no string'

>>> "Aspas 'simples' no string"
"Aspas 'simples' no string"
>>> 'Aspas \'simples\' no string'
"Aspas 'simples' no string"
```

Combinação de atribuição e operação

```
>>> variavel_com_nome_longo = 10

>>> variavel_com_nome_longo = variavel_com_nome_longo + 2
>>> variavel_com_nome_longo
12

>>> variavel_com_nome_longo += 2
>>> variavel_com_nome_longo
14

>>> variavel_com_nome_longo /= 10
>>> variavel_com_nome_longo
1.4
```

Obs: A sintaxe `x++` não existe em Python

Tipos de dados padrão: Coleções

- Lista, tupla e range
  - `list : [0, 1, 4, 9, 16, 25]`
  - `tuple : ('Café com Linux', 2015)`
  - `range : range(10)`
- Dicionário
  - `dict : {'evento': 'Café com Linux', 'ano': 2015}`
- Conjunto
  - `set : {0, 1, True, 'Café'}`

Listas

```
>>> lista = [0, 1, 2, 3]

>>> lista.append('Linux')
>>> lista
[0, 1, 2, 3, 'Linux']

>>> lista.extend([True, 2])
>>> lista
[0, 1, 2, 3, 'Linux', True, 2]

>>> lista.reverse()
>>> lista
[2, True, 'Linux', 3, 2, 1, 0]

>>> 'Linux' in lista     # Checa se é membro
True
```

Indexação

```
>>> lista
[2, True, 'Linux', 3, 2, 1, 0]
>>> len(lista)  # Comprimento da lista
7

>>> lista[0]    # O primeiro elemento é indexado por 0
2
>>> lista[2]
'Linux'
>>> lista[6] = 'Café'
>>> lista
[2, True, 'Linux', 3, 2, 1, 'Café']

>>> lista[7]
# IndexError: list index out of range
```

Indexação negativa

```
>>> lista
[2, True, 'Linux', 3, 2, 1, 'Café']

>>> lista[-1]  # Equivalente a lista[6]
'Café'
>>> lista[-2]  # Equivalente a lista[5]
1
>>> lista[-7]  # Equivalente a lista[0]
2
>>> lista[-8]
# IndexError: list index out of range
```

Ranges

```
>>> r = range(4)
>>> r
range(0, 4)

>>> list(r)
[0, 1, 2, 3]

>>> list(range(1, 6))
[1, 2, 3, 4, 5]

>>> list(range(1, 18, 3))
[1, 4, 7, 10, 13, 16]
```

Compreensão de listas

```
>>> [n for n in range(1, 8)]  # Ou list(range(1, 8))
[1, 2, 3, 4, 5, 6, 7]

>>> [n**2 for n in range(1, 10)]
[1, 4, 9, 16, 25, 36, 49]

>>> [n**2 for n in range(1, 10) if n % 2 == 1]
[1, 9, 25, 49]

>>> [1 / n**2 for n in range(1, 10) if n % 2 == 1]
[1.0, 0.1111111111111111, 0.04, 0.02040816326530612]

>>> sum(1 / n**2 for n in range(1, 10000) if n % 2 == 1)
1.2336505501363413
>>> math.pi**2 / 8
1.2337005501361697
```

Tuplas

```
>>> t = ('Café com Linux', 2015)
>>> t[0]
'Café com Linux'
>>> t[0] = 'Café com Windows'
# TypeError: 'tuple' object does not support item assignment
```

Diferença entre listas e tuplas:

1. Listas são mutáveis; tuplas são imutáveis (mais rápido)
2. Semântica:

   Se você sair para uma caminhada, você pode anotar suas coordenadas a qualquer instante numa tupla (x, y). Se você quiser registrar sua jornada, você pode anexar (a cada minuto) sua posição em uma lista.

   — **dan-gph**, usuário do Stack Overflow

Dicionários

```
>>> d = {'evento': 'Café com Linux', 'ano': 2015}
>>> d['evento']
'Café com Linux'

>>> d['ano'] = 3000
>>> d
{'evento': 'Café com Linux', 'ano': 3000}

>>> d = dict([(2, 'two'), (1, 'one'), (3, 'three')])
>>> d
{1: 'one', 2: 'two', 3: 'three'}
```

Conjuntos

```
>>> s = {1, 2, 3, 1}  # Elementos repetidos são ignorados
>>> s
{1, 2, 3}

>>> s.add(0)          # Adiciona um novo elemento
>>> s
{0, 1, 2, 3}

>>> s.update({4, 5})  # Adiciona novos elementos
>>> s
{0, 1, 2, 3, 4, 5}

>>> set(['Café', 'Chá', 'Suco']) # set a partir de list
{'Café', 'Suco', 'Chá'}
```

Obs: A versão imutável de `set` é `frozenset`

Operações com conjuntos

```
>>> todos = {1, 2, 3, 4, 5, 6}
>>> impares = {1, 3, 5}
>>> primos = {2, 3, 5}

>>> impares | primos  # União
{1, 2, 3, 5}

>>> impares & primos  # Intersecção
{3, 5}

>>> impares ^ primos  # Diferença simétrica
{1, 2}

>>> todos - impares   # Diferença
{2, 4, 6}
```

Slicing

```
>>> import string
>>> alphabet = string.ascii_uppercase; alphabet
'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

>>> alphabet[3 : 5]
'DE'
>>> alphabet[1 : 6 : 2]
'BDF'

>>> alphabet[: 13]
'ABCDEFGHIJKLM'
>>> alphabet[13 :]
'NOPQRSTUVWXYZ'

>>> alphabet[5 : -2]
'FGHIJKLMNOPQRSTUVWX'
```

Controle de fluxo

for/in:

```
for element in iterable:
    corpo [utiliza element]
```

while:

```
while condição:
    corpo
```

if/elif/else:

```
if condição_1:
    corpo_A
elif condição_2:  # Pode ter mais de um elif
    corpo_B
else:
    corpo_C
```

Funções

```
def função(arg1, arg2, ...):
    corpo
    return valor  # return é opcional
```

Funções λ:

```
>>> square = lambda x: x**2
>>> square(4)
16
```

Obs: Funções são objetos que podem ser manipulados e passados a outras funções!

Funções com argumentos opcionais

```
def root(x, n=2):
    return x**(1/n)
```

```
>>> root(2)
1.4142135623730951
>>> root(2, 3)
1.2599210498948732
```

Classes

Tudo em Python é um objeto.

```
class Classe(cls1, cls2, ...):
    def método_1(self, arg1, arg2, ...):
        comandos

    def método_2(self, arg1, arg2, ...):
        comandos
```

Snippets

Fibonacci

```
def fib(n):
    a, b = 0, 1
    for _ in range(n):
        a, b = b, a + b
    return a
```

```
>>> [fib(n) for n in range(10)]
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

Quicksort

```
def qsort(list):
    if not list:
        return []
    else:
        head, tail = list[0], list[1:]
        less = [x for x in tail if x < head]
        more = [x for x in tail if x >= head]
        return qsort(less) + [head] + qsort(more)
```

```
>>> qsort([-1, 5, 2, 17, -4, 5])
[-4, -1, 2, 5, 5, 17]
```

Integração numérica: método dos trapézios

```
def trapz(f, a, b, n):
    h = (b - a) / n  # passo de integração
    s = f(a) / 2
    for i in range(1, n):
        s += f(a + i*h)
    s += f(b) / 2
    return s * h
```

```
>>> f = lambda x: math.sqrt(1 - x**2)  # semicírculo unitário
>>> trapz(f, -1, 1, 10000)
1.5707946637152863
>>> math.pi / 2
1.5707963267948966
```

Cifra de César

```
import string

def caesar(shift, str_):
    alphabet = string.ascii_uppercase
    N = len(alphabet)
    d = {' ': ' '}
    for (i, char) in enumerate(alphabet):
        d[char] = alphabet[(i + shift) % N]
    return ''.join(d[char] for char in str_)
```

```
>>> s_enc = caesar(5, 'CAFE COM LINUX'); s_enc
'HFKJ HTR QNSZC'
>>> s_dec = caesar(-5, s_enc); s_dec
'CAFE COM LINUX'
```

Crivo de Eratóstenes

```
import math

def sieve(n):
    A = [False, False] + [True]*(n - 2)
    for i in range(2, int(math.sqrt(n)) + 1):
        if A[i]:
            for j in range(i**2, n, i):
                A[j] = False
    return A
```

```
>>> is_prime = sieve(8); is_prime
[False, False, True, True, False, True, False, True]
>>> [n for n in range(8) if is_prime[n]]
[2, 3, 5, 7]
```

Daqui pro futuro

Documentação oficial: http://docs.python.org/

Python 3.4.3 documentation

Welcome! This is the documentation for Python 3.4.3, last updated May 16, 2015.

Parts of the documentation:

What's new in Python 3.4?
or all "What's new" documents since 2.0

Tutorial
start here

Library Reference
keep this under your pillow

Language Reference
describes syntax and language elements

Python Setup and Usage
how to use Python on different platforms

Python HOWTOs
in-depth documents on specific topics

Installing Python Modules
installing from the Python Package Index & other sources

Distributing Python Modules
publishing modules for installation by others

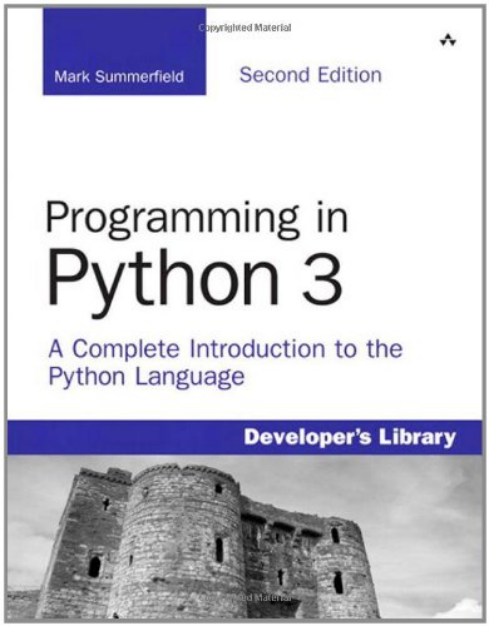
Extending and Embedding
tutorial for C/C++ programmers

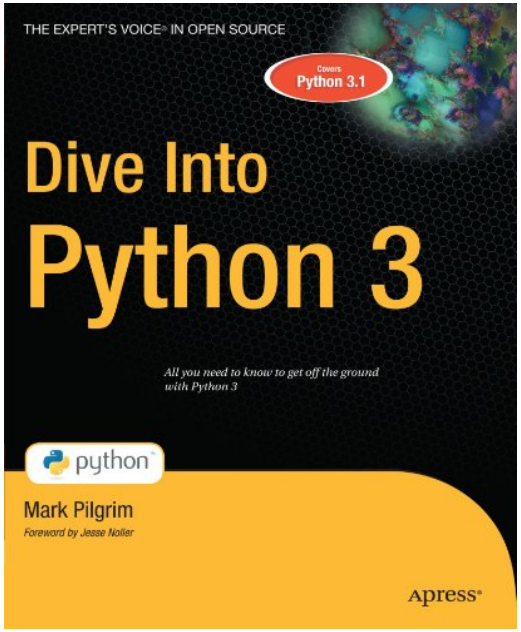
Python/C API
reference for C/C++ programmers

FAQs
frequently asked questions (with answers!)

Referências sugeridas

- **Summerfield, Mark.** *Programming in Python 3: A Complete Introduction to the Python Language*, 2nd Edition. Addison-Wesley Professional, 2009.
- **Pilgrim, Mark.** *Dive Into Python*. Apress, 2009. Disponível em `http://www.diveintopython3.net/`.





Aplicações científicas

- Bibliotecas (não padrão):
  - **NumPy**: Manipulação de arrays (vetores, matrizes)
  - **SciPy**: Suplementar ao NumPy, adicionando mais funções
  - **matplotlib**: Gráficos
  - **SymPy**: Matemática simbólica
- Outros:
  - **Jupyter**: Notebook web interativo
  - **SageMath**: Combina vários pacotes científicos open-source

Obrigado!

Roberto W. Nóbrega
http://rwnobrega.name/
rwnobrega@gmail.com