



Lorran Ferreira Maroco Sutter  
Universidade Federal de Juiz de Fora

# Conteúdo

## 1 Introdução à Linguagem Python

- Histórico
- Características
- Primeiros Passos
- Ambientes de programação
- Começando a programar em Python (finalmente)

## Sobre a Linguagem Python

- Criada por Guido van Rossum em 1991
- Nome originado do grupo humorístico Monty Python's Flying Circus

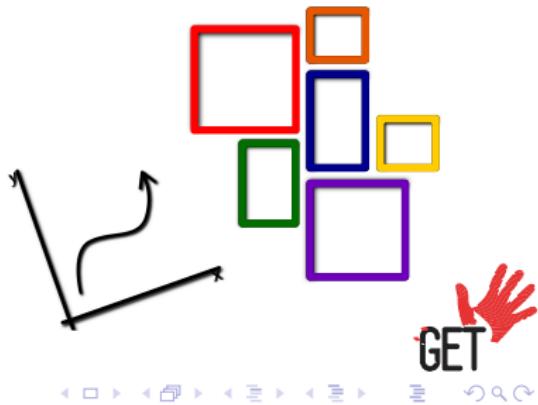


- Linguagem de alto nível

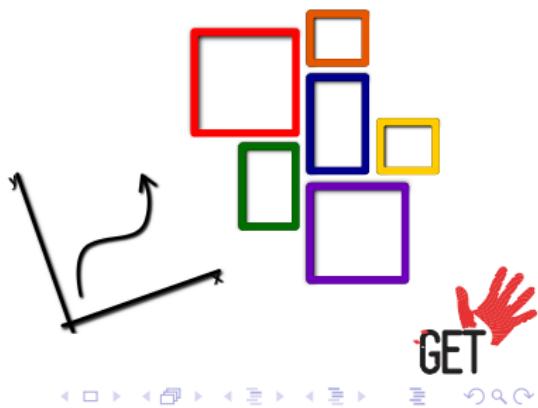
- Linguagem de alto nível
- Interpretada
  - Matlab
  - HTML, CSS, PHP, JavaScript
  - Perl, Lua, Ruby, Haskell



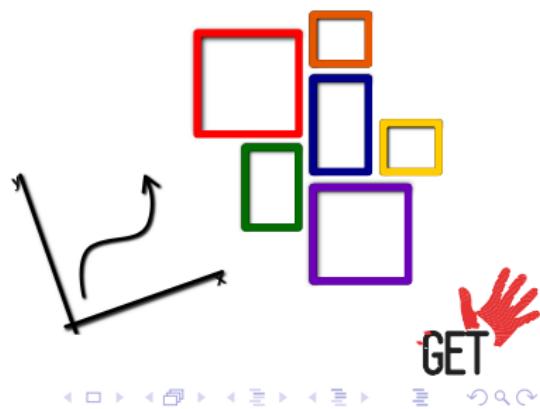
- Linguagem de alto nível
- Interpretada
  - Matlab
  - HTML, CSS, PHP, JavaScript
  - Perl, Lua, Ruby, Haskell
- Tipo de programação
  - Modular
  - Orientada a objetos
  - Funcional



- Linguagem de alto nível
- Interpretada
  - Matlab
  - HTML, CSS, PHP, JavaScript
  - Perl, Lua, Ruby, Haskell
- Tipo de programação
  - Modular
  - Orientada a objetos
  - Funcional
- Tipagem dinâmica e forte



- Linguagem de alto nível
- Interpretada
  - Matlab
  - HTML, CSS, PHP, JavaScript
  - Perl, Lua, Ruby, Haskell
- Tipo de programação
  - Modular
  - Orientada a objetos
  - Funcional
- Tipagem dinâmica e forte
- Código aberto
  - GPL - General Public License



## Quem usa Python?

 ABAQUS



YouTube

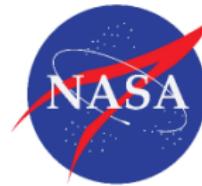


BATTLEFIELD 2

OpenOffice



Google



GET

# Porque usar Python?

- Fácil, simples, intuitiva
- Sintaxe limpa
- Diversas bibliotecas já inclusas
- Interface com outras linguagens como C/C++ e Fortran

## Instale o Python 3

- Mac OS X e Linux vem com o Python 2 pré-instalado, mas pode vir com a versão 3 também

## Instale o Python 3

- Mac OS X e Linux vem com o Python 2 pré-instalado, mas pode vir com a versão 3 também
- Verifique se o Python já está instalado no seu computador!
  - Mac OS X e Linux, abra o terminal e digite:  
→ *python3 -V*
  - Windows, abra o terminal e digite:  
→ *c:\Python3\python.exe -V*

# Instale o Python 3

- Mac OS X e Linux vem com o Python 2 pré-instalado, mas pode vir com a versão 3 também
- Verifique se o Python já está instalado no seu computador!
  - Mac OS X e Linux, abra o terminal e digite:  
→ *python3 -V*
  - Windows, abra o terminal e digite:  
→ *c:\Python3\python.exe -V*
- Para download da última versão do Python, acesse:  
→ [www.python.org](http://www.python.org)

The screenshot shows the official Python website at [www.python.org](http://www.python.org). The top navigation bar includes links for Python, PSF, Docs, PyPI, Jobs, and Community. The main header features the Python logo and the word "python". A search bar with a magnifying glass icon and a "GO" button is positioned above a blue navigation bar with links for About, Downloads, Documentation, Community, Success Stories, News, and Events. Below this is a dark sidebar containing Python code snippets. One snippet shows a for loop: 

```
# For loop on a list
>>> numbers = [2, 4, 6, 8]
>>> product = 1
```

 To the right of the code is a yellow button with a right-pointing arrow. Another snippet below it reads: **All the Flow You'd Expect** followed by the text: "Python knows the usual control flow statements that other". A red hand icon is visible on the far right.

# Ambiente para programar em Python

- Inúmeros ambientes para se programar em Python

# Ambiente para programar em Python

- Inúmeros ambientes para se programar em Python
- Instalando direto do site → *IDLE*  
→ abra o seu terminal e digite *python3*

# Ambiente para programar em Python

- Inúmeros ambientes para se programar em Python
- Instalando direto do site → *IDLE*  
→ abra o seu terminal e digite *python3*
- Ambiente muito utilizado:



*Spyder*

# Ambiente para programar em Python

- Inúmeros ambientes para se programar em Python
- Instalando direto do site → *IDLE*  
→ abra o seu terminal e digite *python3*
- Ambiente muito utilizado:



*Spyder*

- Para Linux:  
→ *sudo apt-get install spyder*
- Para Windows, acesse o site:  
→ [pypi.python.org/pypi/spyder](https://pypi.python.org/pypi/spyder)

# Ambiente para programar em Python

- Inúmeros ambientes para se programar em Python
- Instalando direto do site → *IDLE*  
→ abra o seu terminal e digite *python3*
- Ambiente muito utilizado:



*Spyder*

- Para Linux:  
→ *sudo apt-get install spyder*
- Para Windows, acesse o site:  
→ [pypi.python.org/pypi/spyder](http://pypi.python.org/pypi/spyder)
- Recomendável para Windows! Baixe o pacote *pythonxy*:



[code.google.com/p/pythonxy/](http://code.google.com/p/pythonxy/)

## Ambientes para programar em Python

- Para Linux, baixe o *ipython* e o *ipython-qtconsole*:  
→ *sudo apt-get install ipython*  
→ *sudo apt-get install ipython-qtconsole*
- Excelente ambiente para ajudar a debugar o código e visualizar cada variável individualmente

Vamos dar uma olhada na IDLE do Python inicialmente.  
Lembre-se de digitar *python3* no seu terminal.



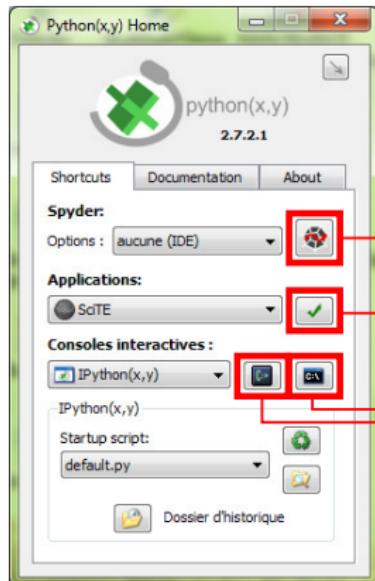
## Usando a IDLE do Python

Aqui é possível executar diversos comandos do Python, assim como escrever um código completo (apesar de não ser muito conveniente).

Vamos fazer alguns testes!

```
lorran@lorran-sutter:~  
lorran@lorran-sutter:~$ python3  
Python 3.4.0 (default, Apr 11 2014, 13:05:11)  
[GCC 4.8.2] on linux  
Type "help", "copyright", "credits" or "license" for more information.  
>>> █
```

# Usuários de Windows



Spyder

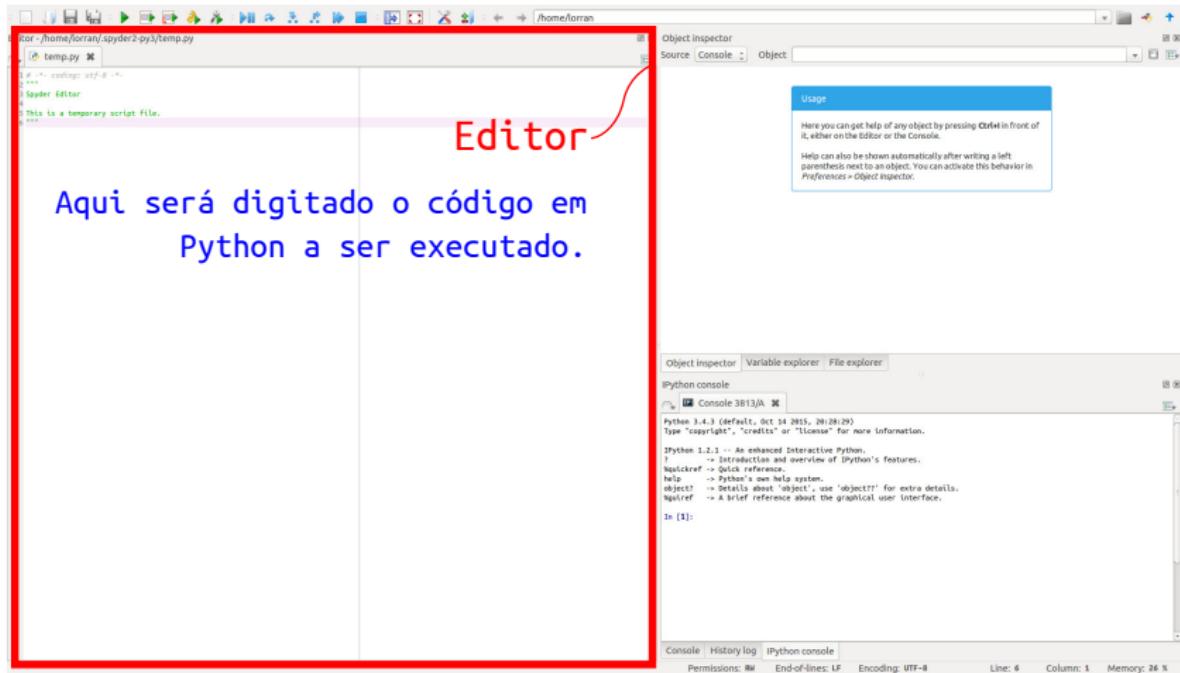
Aplicativos

Terminal do  
sistema operacional

Terminal próprio

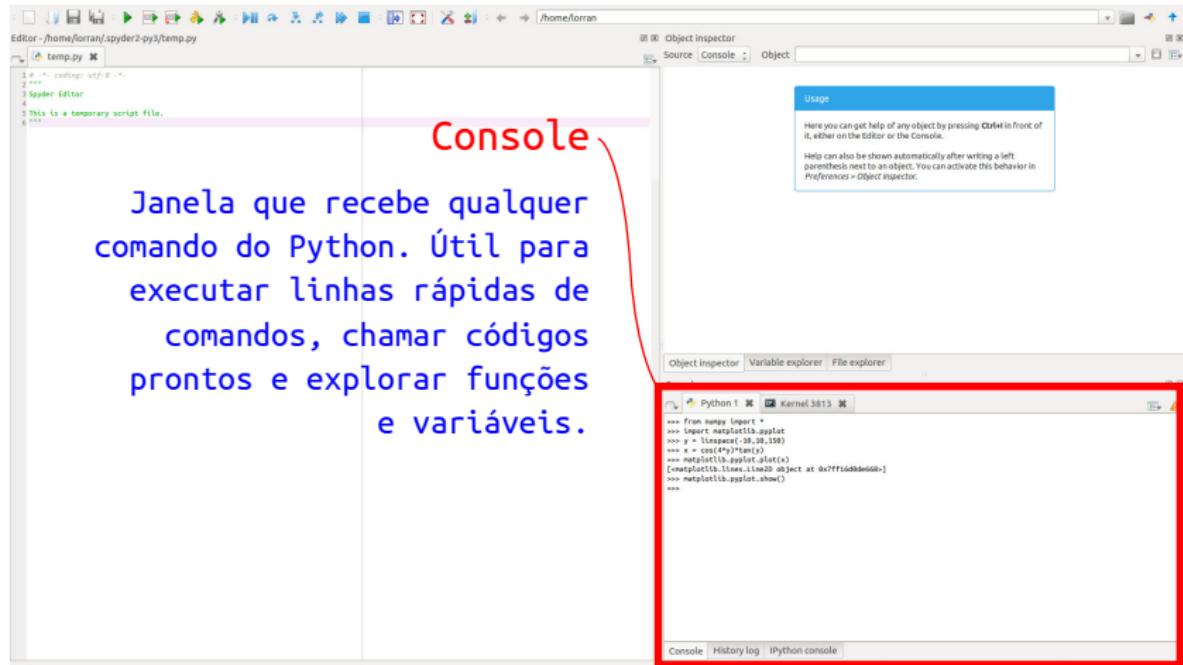


# spyder



Aqui será digitado o código em Python a ser executado.

GET



Janela que recebe qualquer comando do Python. Útil para executar linhas rápidas de comandos, chamar códigos prontos e explorar funções e variáveis.

The screenshot shows the Spyder IDE interface. On the left, there is a code editor window titled 'temp.py' containing the following Python code:

```
1 # -*- coding: utf-8 -*-
2 """
3 Spyder Editor
4
5 This is a temporary script file.
6 """
7
```

To the right of the code editor is the IPython console window, which has a red border around its main area. The console displays the following text:

```
Python 3.4.3 (default, Oct 14 2015, 20:29:19)
Type "copyright", "credits" or "license" for more information.

Python 3.2.3 -- An enhanced Interactive Python.
?          -- Introduction and overview of IPython's features.
%quickref -- Quick reference for IPython's commands.
%help     -- IPython's own help system.
object?   -- Details about 'object', use 'object??' for extra details.
helpref?  -- A brief reference about the graphical user interface.

In [1]:
```

At the top of the IPython window, there are tabs for 'Object inspector', 'Variable explorer', and 'File explorer'. Above the IPython window, there is a 'Usage' help panel with the following text:

Here you can get help of any object by pressing **Ctrl** in front of it, either on the editor or the console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in Preferences > Object Inspector.

## IPython Console

Janela que recebe qualquer comando do Python. Útil para executar linhas rápidas de comandos, chamar códigos prontos e explorar funções e variáveis.

*Bonito e colorido.*

Editor - /home/lorrann/spyder2/py3/temp.py

```
temp.py # coding: utf-8
# ...
# Spyder Editor
#
# This is a temporary script file.
# ...
```

Variable explorer

Name	Type	Size	Value
a	int	1	3
b	float	1	4.54
k	int	1	9
l	list	3	<list @ 0x7FFF368018FCB>
s	str	1	Hello world
t	tuple	5	<tuple @ 0x7FFF36783678>
X	float64	(200,)	array([-10.0, ..., 397.99, ..., -9.7999497, ..., -9.69840246, ...])
y	float64	(200,)	array([-1.7927559e-01, ..., -1.12144268e-01, ..., -6.05642337e-02, ..., -2.8747987 ...])

Object inspector | Variable explorer | File explorer

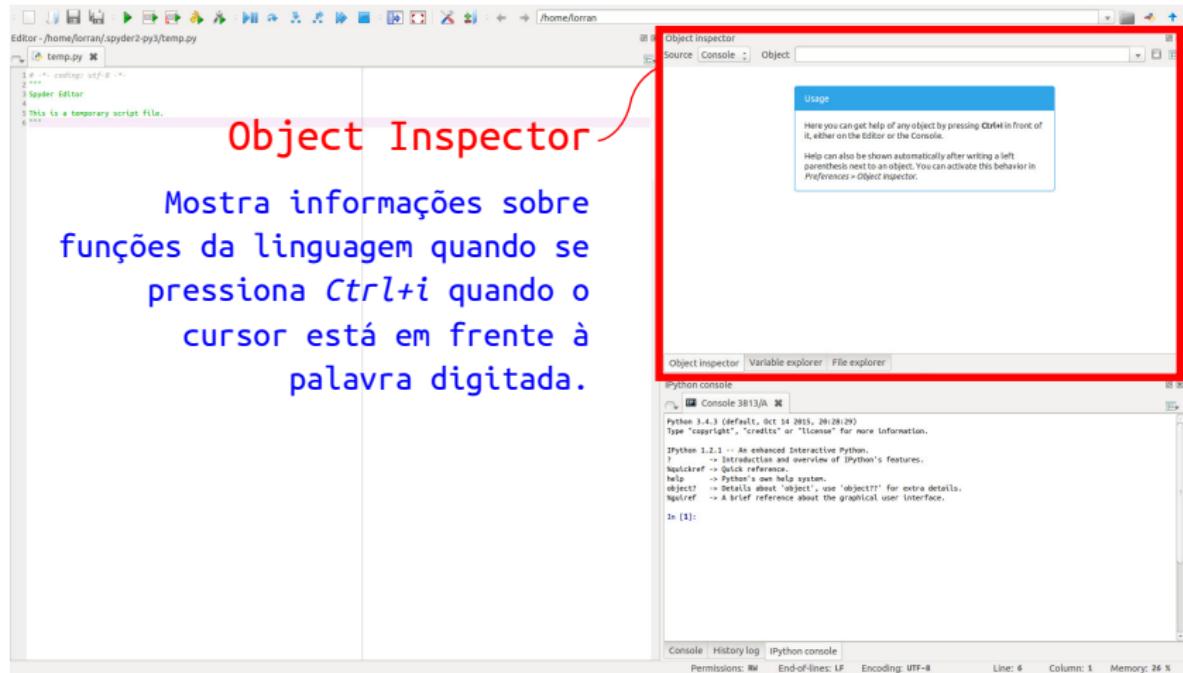
history.py

```
def hello():
    s = "Hello World"
    t = dict()
    t['x1', 'x2'] = (2, 2)
    t = {(1, 4, 2, 3, 2)}
    print(s)
    for k in range(50):
        print(k*0.32)

from numpy import *
import matplotlib.pyplot
x = linspace(-10,10,150)
y = cos(4*x)*tan(x)
x = linspace(-10,10,150)
y = cos(4*x)*tan(y)
matplotlib.pyplot.plot(x,y)
matplotlib.pyplot.show()
```

Console | History log | IPython console

GET



Variable Explorer

Lista todas as variáveis criadas com suas respectivas especificações.

The screenshot shows the Spyder IDE interface. On the left is the code editor with a temporary script file named 'temp.py'. The code is as follows:

```
1 # -*- coding: utf-8 -*-
2 ***
3
4
5 This is a temporary script file.
6 ***
```

In the center is the Variable explorer window, which lists variables 'a', 'b', 'k', 'l', 's', 't', 'x', and 'y' with their types and values. A red box highlights this window. Below it is the IPython console window, also highlighted with a red box. The console shows the execution of a list comprehension and prints the resulting list. At the bottom, there is a navigation bar with icons for back, forward, search, and other functions. A red hand icon is overlaid in the bottom right corner.

Name	Type	Size	Value
a	int	1	3
b	float	1	4.54
k	int	1	9
l	list	3	<list @ 0x7ff8368019FC>
s	str	1	Hello world
t	tuple	5	<tuple @ 0x7ff836783670>
x	float64	(200,)	array([-10.000000e+00, -3.597889e-01, -9.793994e-01, -9.698402e-01,
y	float64	(200,)	array([-1.7927559e+01, -1.12144268e+01, -6.05642337e+02,

Object Inspector Variable explorer File explorer

In [15]: t = [(1,2,3,2,1)]  
Out[15]: [(1, 2, 3, 2, 1)]

In [16]: SyntaxError: invalid syntax

In [17]: r = range(4)  
In [18]: for k in range(10):  
...: print(k\*\*0.5)  
...:  
0.0  
0.32  
0.44  
0.56  
1.28  
1.56  
1.92  
2.24  
2.56  
2.88

In [19]:

Permissions: RW End-of-lines: LF Encoding: UTF-8 Line: 6 Column: 1 Memory: 26 %

GET

**File Explorer**

Árvore contendo as pastas e arquivos do computador.

The screenshot shows a desktop interface with several windows open:

- File explorer:** A window titled "File explorer" showing a list of files and folders in the current directory. A red box highlights this window.
- IPython console:** A window titled "ipython console" containing Python code and its output. It includes a history log and an IPython console tab.
- Editor:** A window titled "temp.py" showing a Python script with a single line of code: "print('Hello, World!')".

The desktop background is blue, and there are various icons on the desktop.

GET

# Antes de começarmos

## Palavras Reservadas

and	else	import	return
as	except	in	True
assert	exec	is	try
break	False	lambda	while
class	finally	not	with
continue	for	or	yield
def	from	pass	
del	global	print	
elif	if	raise	

## Primeiro programa!

```
>>> print("Hello World")
```

## Primeiro programa!

```
>>> print("Hello World")
```

## Cuidado!

Até o Python2 era permitido:

```
>>> print "Hello World"
```

A partir do Python3, o *print* se tornou função!

## Funções úteis

### Funções

- `Ctrl + L` → Limpa a tela
- `help()` → Passe como parâmetro alguma função do Python para saber sua descrição
- `del` → Delete variáveis armazenadas
- `exit()` → Encerra o programa

### Somente no Ipython

- `funcao?` → Use uma interrogação na frente de alguma função do Python para ter uma descrição sucinta
- `whos` → Exibe as variáveis armazenadas
- `reset` → Apaga todas as variáveis



Variáveis! Sim, ai vêm elas!



## O Python não é tipado!

Como dito na introdução, o Python é uma linguagem de tipagem dinâmica e forte, ou seja, o interpretador do Python vai saber exatamente do que se trata a sua variável dependendo do que você armazenar nela.

## Algumas variáveis...

```
>>> str1 = "Dumbledore"  
>>> str2 = "Hagrid"  
>>> idade = 927
```

## Acesse suas variáveis!

Tente acessar suas variáveis digitando o nome delas novamente na linha de comando. Você pode alterá-las dando novos valores à elas!

## Outra forma de atribuição

```
>>> str1, str2, idade = "Dumbledore", "Hagrid", 927
```

## Onde está o ponto e vírgula?

O ponto e vírgula no final das sentenças do Python indica *fim de comando*, mas é facultativo. Quando há a quebra de linha, o interpretador já sabe que se encerrou aquele comando.

### Cuidado!

Não coloque dois comando diferentes na mesma linha! A não ser que estejam separados por ponto e vírgula.

### Funciona

```
>>> x = 2  
>>> y = 3
```

ou

```
>>> x = 2; y = 3
```



Agora pode usar *print* com as variáveis!

```
>>> print(str1)
```

Dumbledore

```
>>> print(str2)
```

Hagrid

```
>>> print(idade)
```

927

E com todas elas juntas também

```
>>> print(str1,str2,idade)
```

Dumbledore Hagrid 927

# Tipos de Variáveis



**TRUE**

**FALSE**



- **boolean**

- Valor lógico
  - *True ou False*
- Primeira letra **deve** ser maiúscula

P y t h o n

# Tipos de Variáveis

- **int**

- Numerais inteiros
  - -203, -94, 0, 1, 482, 120938
- O tipo *long*, presente antes do Python3 foi substituído somente pelo *int*. Assim, é possível trabalhar com números maiores de maneira mais facilitada.

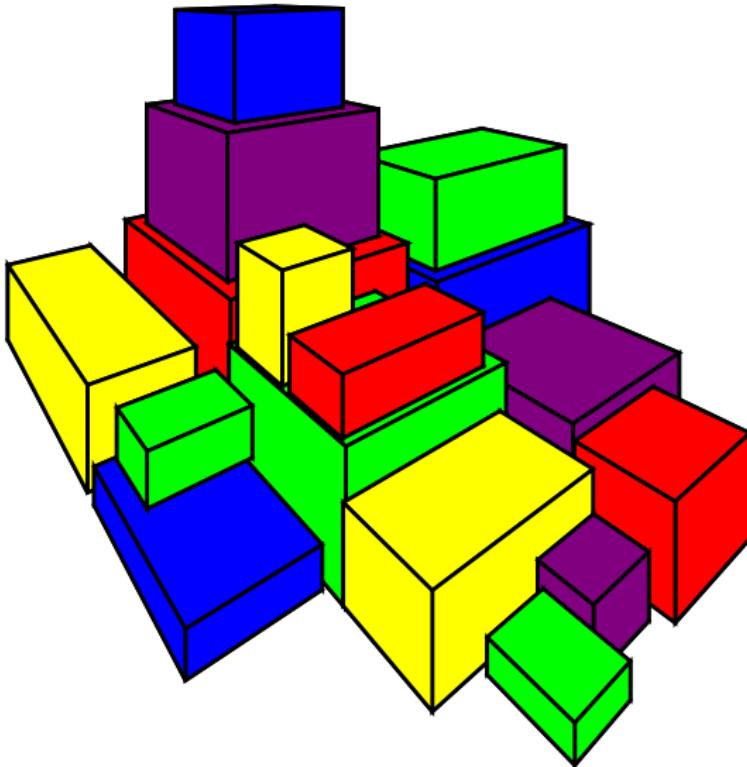
- **float**

- Numerais reais
  - -303.351, -9.35, 0.0, 3.12159, 1902.0001

- **complex**

- Numerais complexos
- Necessário utilizar a função *complex()*
  - `complex(3,5)`, `complex(3.53,-0.32)`

# Vamos brincar com as nossas variáveis



# Operações aritméticas

Operação	Sintaxe	Operação	Sintaxe
Adição	$a + b$	Valor absoluto	<code>abs(a)</code>
Subtração	$a - b$	Converte inteiro	<code>int(a)</code>
Produto	$a * b$	Converte float	<code>float(a)</code>
Divisão	$a / b$	Conjugado	<code>a.conjugate()</code>
Divisão inteira	$a // b$	Divisão e módulo	<code>divmod(a, b)</code>
Módulo	$a \% b$	Potenciação	<code>pow(a, b)</code>
Negação	$- a$	Potenciação	$a ** b$

## Exemplos bonitos

Sim, ainda podemos usar parênteses

```
>>> a, b, c, d = 2, 5, 6, 9  
>>> ((a + b) * d) // c  
10  
>>> divmod(((d % c) ** a),b)  
(145, 4)
```

Cuidado! Tipagem forte

```
>>> a + str1  
Traceback (most recent call last):  
    File "<stdin>", line 1, in <module>  
TypeError: cannot concatenate 'str' and 'int' objects
```



# Operações lógicas

Operação	Sintaxe
Ou	<code>or</code>
E	<code>and</code>
Negação	<code>not</code>
Objeto igual	<code>is</code>
Negação de objeto igual	<code>is not</code>
Estritamente menor	<code>&lt;</code>
Menor ou igual	<code>&lt;=</code>
Estritamente maior	<code>&gt;</code>
Maior ou igual	<code>&gt;=</code>
Igual	<code>==</code>
Diferente	<code>!=</code>

## Exemplos bonitos

Sim, sim, ainda é pode usar parênteses

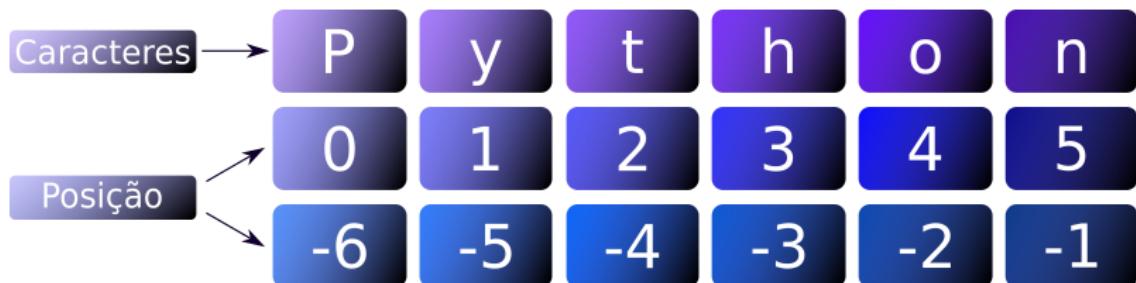
```
>>> (a < b) and (c >= d)  
False
```

## Exemplos

```
>>> str1 is not str2  
True  
>>> not a == b  
True  
>>> (a is a and str1 != str2) or d == c  
True
```

# Brincando com strings

- Cadeia de caracteres  
→ "Python" ou 'Python'



## Exemplos bonitos

### Acesso aos extremos

```
>>> str1[0]
```

```
'D'
```

```
>>> str1[-1]
```

```
'e'
```

string[início:fim]

### Acesso à substrings

```
>>> str1[0:2]
```

```
'Du'
```

```
>>> str1[3:-1]
```

```
'edor'
```



## Acesso à substrings

Note que quanto tentamos acessar algum intervalo da string, a sequência retornada pára antes do índice após os dois pontos. Em contrapartida, inicia-se exatamente no índice do número antes dos dois pontos.



string[1:4] → 'yth'  
começa exatamente aqui  
pára antes daqui

## Atenção!

Podemos acessar os extremos da string com os índices 0 e -1.

Mas isso não funciona para substrings quando tentamos acessar o fim das mesmas.

## Acesso aos extremos

```
>>> str1[0:3]  
'Dum'  
>>> str1[:3]  
'Dum'  
>>> str1[3:-1]  
'bledor'  
>>> str1[3:]  
'bledore'
```



# Métodos para strings

capitalize	isalnum	join	rsplit
casefold	isalpha	ljust	rstrip
center	isdecimal	lower	split
count	isdigit	lstrip	splitlines
encode	isidentifier	maketrans	startswith
endswith	islower	partition	strip
expandtabs	isnumeric	replace	swapcase
find	isprintable	rfind	title
format	isspace	rindex	translate
format_map	istitle	rjust	upper
index	isupper	rpartition	zfill

# Exemplos bonitos

## Exemplo

```
>>> s1 = "PYTHON IS POWERFUL"
>>> s1.lower()
'python is powerful'
>>> s1.lower().split()
['python', 'is', 'powerful']
>>> s2 = s1.lower().split()
>>> ". ".join(s2)
'python. is. powerful'
>>> ". ".join(s2) + "!!!!"
'python. is. powerful!!!!'
>>> (".". join(s2) + "!!!!").title()
'Python. Is. Powerful!!!!'
```



## Exercício legal

**Exercício:** Dada a string que representa uma URL de uma página da web, obter apenas o endereço da página principal.

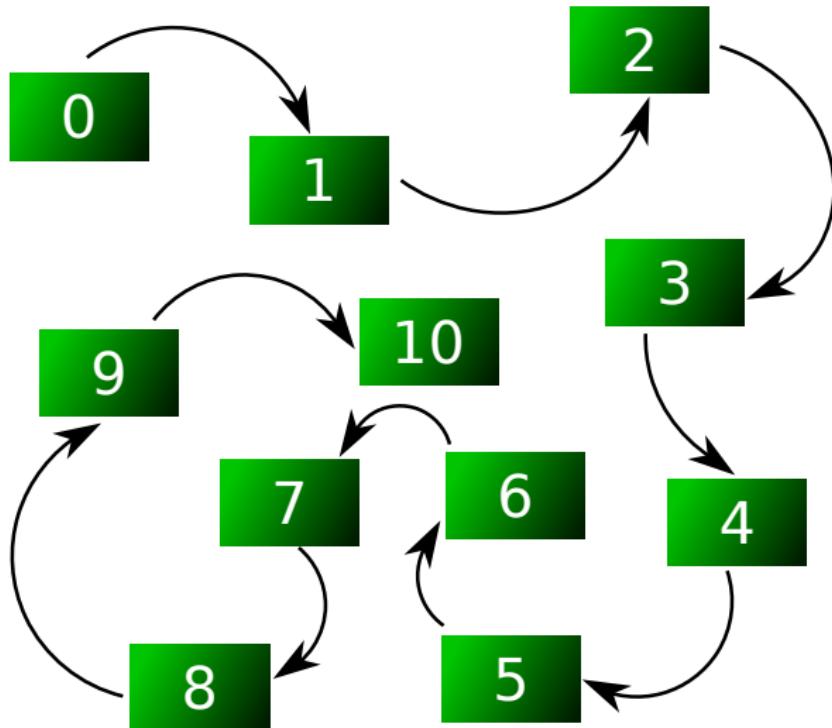
Entrada: '<http://www.facebook.com/bellatrix/spells>'

Saída: '[www.facebook.com](http://www.facebook.com)'

Não existem  
vetores  
em Python!

*Não da maneira convencional...*

# Listas e Tuplas



# Tipos de Variáveis

- **list**

- Coleção de itens de qualquer tipo homogêneos ou não
- É mutável após a criação
  - `list()`, lista vazia
  - `[3, 0.5, True, "string", complex(4,8)]`

- **tuple**

- Coleção de itens de qualquer tipo homogêneos ou não
- **Não** é mutável após a criação
  - `tuple()`, tupla vazia
  - `1, 2, 3, "string", False`
  - `(3, 0.5, True, "string", complex(4,8))`

## Exemplos bonitos

### Tupla

```
>>> t1 = (1,2,3)
>>> t2 = t1, ('a','b','c')
>>> t3 = ('um elemento',)
```

### Lista

```
>>> l1 = [1,2,3]
>>> l2 = [l1, ['a','b','c']]
>>> l3 = ['um elemento']
```

### Desempacotando

```
>>> x1,y1,z1 = t1
>>> x2,y2,z2 = l1
```



## Exemplos bonitos

### Acesso

O acesso aos elementos da tupla de da lista se dá da mesma forma que o acesso aos elementos da string.

### Acesso

```
>>> t1[0]  
1  
>>> t1[1:-1]  
(2,)  
>>> l2[1]  
['a', 'b', 'c']  
>>> l2[1][0]  
'a'
```



# Operações para sequências

Sintaxe	Operação
<code>x in L</code>	True se x está contido em L. False caso contrário
<code>x not in L</code>	Negação de <code>in</code>
<code>L1 + L2</code>	Concatenação
<code>n * L ou L * n</code>	n cópias de L concatenadas
<code>L[i:j:k]</code>	Fatia L de i até j com passo k

# Métodos para listas

Sintaxe	Operação
append(x)	Acrescenta item ao final da lista
copy()	Cópia superficial
extend(L)	Acrescenta a lista L ao final da lista
insert(x,y)	Insere y após o índice x
remove(x)	Remove o valor x
sort()	Ordena a lista
clear()	Limpa a lista
count(x)	Retornar a quantidade de ocorrências de x
index(x)	Retorna o primeiro índice correspondente ao valor x
pop(x)	Remove item de índice x (default é o último)
reverse()	Inverte a ordem da lista



GET

# Exemplos bonitos

## Listas

```
>>> l1 = ['a', 'z', 'u']
>>> l1.append('b')
>>> print(l1)
['a', 'z', 'u', 'b']
>>> l1.sort()
>>> print(l1)
['a', 'b', 'u', 'z']
```

## Exercício legal

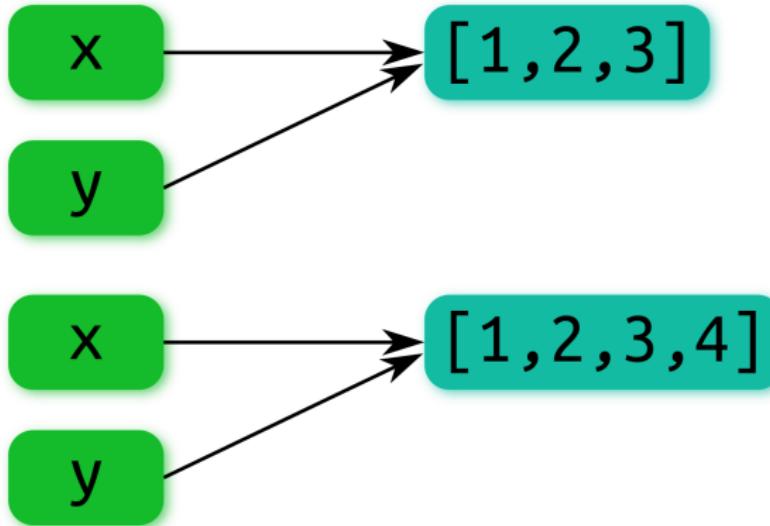
*Exercício:* Criar duas listas quaisquer de valores homogêneos. Concatenar as duas listas de forma que o resultado comece com o menor elemento da primeira lista e termine com o menor elemento da segunda lista.

## Atenção!

Em Python, uma variável é apenas um NOME que REFERENCIA um OBJETO.

## Referência

```
>>> x = [1,2,3]
>>> y = x
>>> x.append(4)
>>> print(x)
[1, 2, 3, 4]
>>> print(y)
[1, 2, 3, 4]
```



## Cópia

```
>>> x = [1,2,3]
>>> y = x[:]
>>> x is y
False
>>> y = x.copy()
>> x is y
False
```

# Tipos de Variáveis

- **set**

- Conjunto ordenado, elementos não duplicados, heterogêneo ou não
- É mutável após a criação
  - `set()`, set vazio
  - `{1, False, "String", 0.3}`
  - `set([0.4, -9, False, "Set"])`

- **frozenset**

- Conjunto ordenado, elementos não duplicados, heterogêneo ou não
- **Não** é mutável após a criação
- Necessário utilizar a função `frozenset()`
  - `frozenset()`, frozenset vazio
  - `frozenset({5, 7, False})`
  - `frozenset([-0.6, True, "Frozen"])`

Sintaxe	Operação
$x \text{ in } S$	True se $x$ está contido em $S$ . False caso contrário
$x \text{ not in } S$	Negação de <b>in</b>
$S1 - S2$	Diferença
$S1 \leq S2$	Testa se todos os elementos de $S1$ estão em $S2$
$S1 < S2$	Testa se $S1$ é subconjunto de $S2$
$S1   S2$	União
$S1 \& S2$	Interseção
$S1 ^ S2$	Ou exclusivo (ou ou xor)

# Exemplos bonitos

## Sets

```
>>> S1 = {'a', 'd', 'm', 'p'}  
>>> S2 = {'d', 'm', 'k', 'o'}  
>>> S1 & S2  
{'d', 'm'}  
>>> S1 & S2  
{'a', 'k', 'o', 'p'}
```

## Exercício legal

*Exercício:* Dada as listas abaixo:

L1 = [1,5,5,8,8,3]

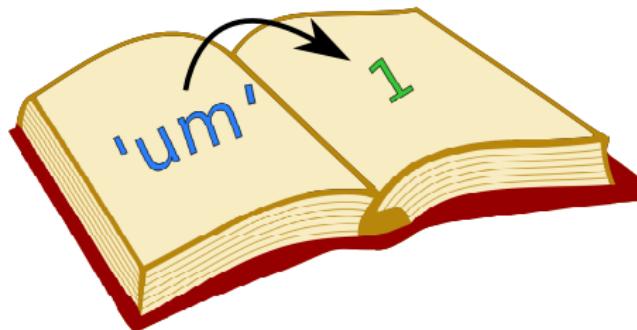
L2 = [9,0,9,4,7]

criar uma única lista ordenada sem valores repetidos.

# Tipos de Variáveis

- **dictionary**

- Conjunto associativo de itens homogêneos ou não
  - `dict()`, dicionário vazio
  - `{'um':1, 'dois':2, 'tres':3 }`
  - `dict(um=1, dois=2, tres=3)`
  - `dict(zip(['um', 'dois', 'tres'], [1, 2, 3]))`
  - `dict([('um', 2), ('dois', 1), ('tres', 3)])`



# Exemplos bonitos

## Dicionários

```
>>> D = {'um':1, 'dois':2, 'tres':3}
>>> print(D)
{'dois':2, 'tres':3, 'um':1}
>>> D.get('um')
1
>>> D.keys()
dict_keys(['dois', 'tres', 'um'])
>>> D.values()
dict_values([2,3,1])
>>> list(D.values())
[2,3,1]
```



## Funções Nativas

abs	dict	help	min	setattr
all	dir	hex	next	slice
any	divmod	id	object	sorted
ascii	enumerate	input	oct	staticmethod
bin	eval	int	open	str
bool	exec	isinstance	ord	sum
bytearray	filter	issubclass	pow	super
bytes	float	iter	print	tuple
callable	format	len	property	type
chr	frozenset	list	range	vars
classmethod	getattr	locals	repr	zip
compile	globals	map	reversed	__import__
complex	hasattr	max	round	
delattr	hash	memoryview	set	



GET

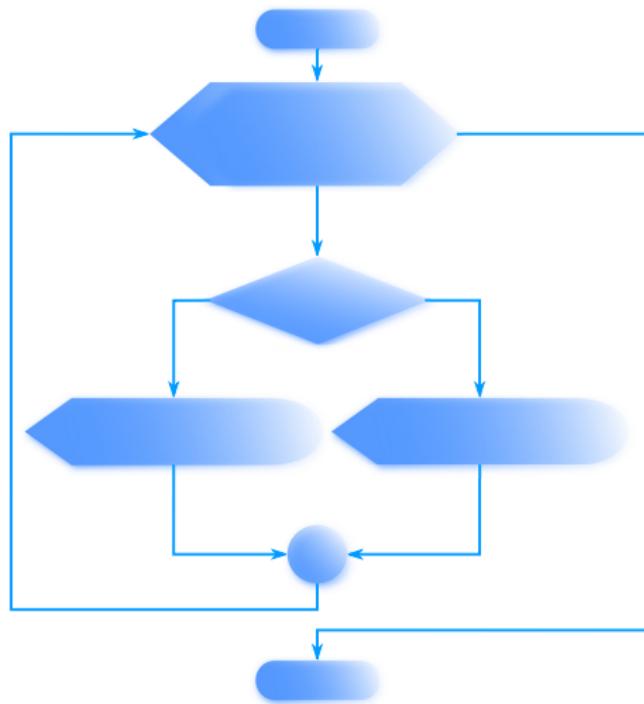
# Exemplos bonitos

## Funções Nativas

```
>>> L = [1,2,3]
>>> print(len(L))
3
>>> print(abs(-2))
2
>>> print(min(L))
1
>>> print(round(5.7684,2))
5.77
>>> sum(L)
6
>>> type(L)
builtins.list
```



# Estruturas de Controle



## Sintaxe

O `()` que delimitaria a expressão é facultativo.

Ao final da expressão é **obrigatório** haver um sinal de `:` para delimitar o fim da sentença.

## Atenção!

Os blocos de comandos das estruturas de controle do Python não é delimitada por `{}`. A **identação** dos comandos é obrigatória!

## if-else

```
# if-else
if expressao:
    comando1
    comando2
    #...
else:
    comando1
    comando2
    #...
```

## if-elif-else

### switch-case

Em Python não existe a estrutura *switch-case* como em outras linguagens. Esta pode ser substituída pela estrutura abaixo.

```
# if-elif-else
if expressao:
    comandos
    #...
elif expressao:
    comandos
    #...
else:
    comandos
    #...
```

## Exercício bonito

*Exercício:* Dada uma entrada de dados do usuário, verificar se é um número inteiro. Caso seja, solicitar outro número como entrada. Se o segundo número for maior que o primeiro, efetuar uma soma e exibir na tela. Caso contrário, efetuar uma divisão inteira e exibir na tela.

## Estruturas de Controle

```
for item in lista:  
    comandos  
  
for item in range():  
    comandos  
  
while expressao:  
    comandos  
  
while expressao1:  
    if expressao2:  
        comandos1  
        if expressao3:  
            break  
    comandos2
```

## Exercício legal

*Exercício:* Escreva um programa que, dada uma lista de palavras, retorne uma lista de inteiros representando o tamanho de cada palavra da primeira lista.

# Funções e procedimentos

```
# Procedimento
def nome(argumento1, argumento2, ...):
    comandos
    return

# Função
def nome1(argumento1, argumento2, ...):
    comandos
    return expressao1

# Função
def nome2(argumento1, argumento2, ...):
    comandos
    return expressao1, expressao2, ...

# Função
def nome3(argumento1, argumento2, argx = valor):
    comandos
    return expressao3
```

## Exercício legal

*Exercício:* Escreva uma função chamada `funcao_saborosa()` que recebe uma lista de palavras e retorna uma tupla contendo a maior e a menor palavra da lista passada como parâmetro. Retorne uma tupla vazia caso a lista seja vazia.