

# Tipos de dados em Python

Eduardo Mendes  
[github.com/z4r4tu5tr4](https://github.com/z4r4tu5tr4)

# Tipos de dados Python



**Python 101**

Eduardo Mendes  
[github.com/z4r4tu5tr4](https://github.com/z4r4tu5tr4)



**A introdução  
mais rápida  
da história**

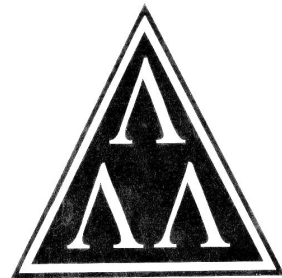


**Python 101**

**e dados  
em Python**

Eduardo Mendes  
[github.com/z4r4tu5tr4](https://github.com/z4r4tu5tr4)

# **z4r4tu5tr4@babbge: screenfetch**



**Nome:** Eduardo Mendes

**Instituição:** Fatec Americana

**Uptime:** 12097080s

**Email:** mendexeduardo@gmail.com

**git:** [github.com/z4r4tu5tr4](https://github.com/z4r4tu5tr4)

- Variáveis não são caixas
- Tudo é objeto
- Tipos numéricos
- Estruturas de decisão
- Strings
- Laço While
- Listas
- Laço for
- Funções

- Tuplas
- Conjuntos
- Dicionários

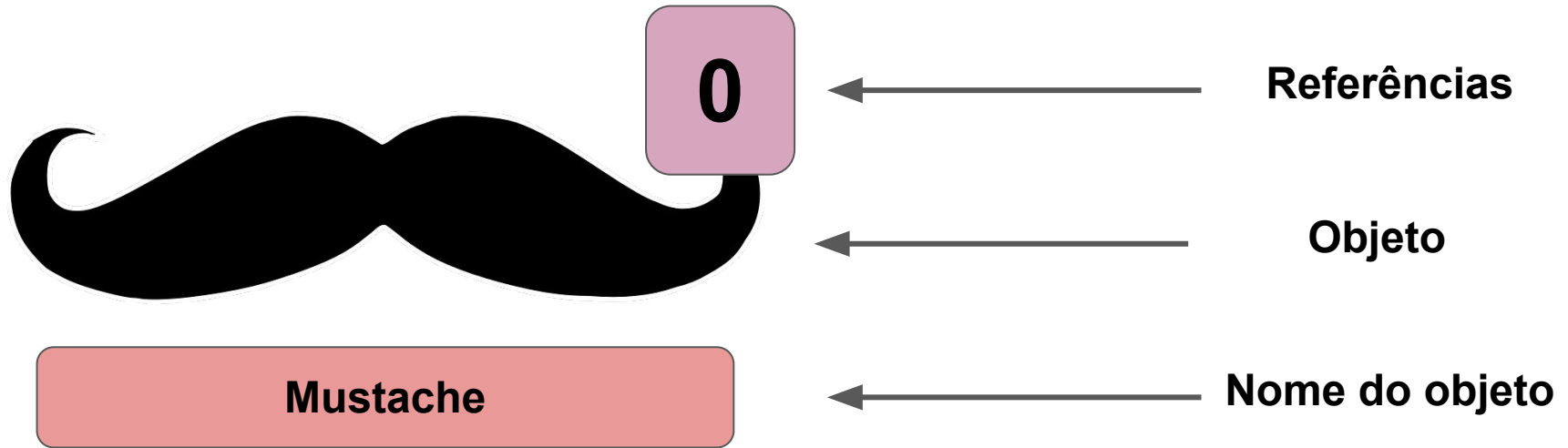
Se der tempo

## Estrutura

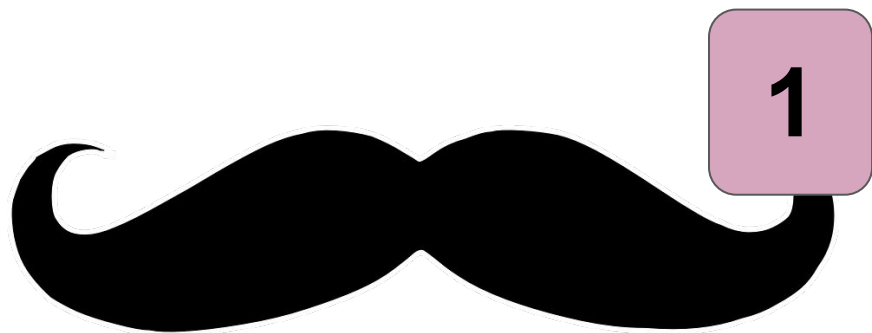
# Variáveis não são caixas

**Ou a triste história do bigode**

# Variaveis -> Alias



# Variaveis -> Alias



**Mustache**

**bigodinho**

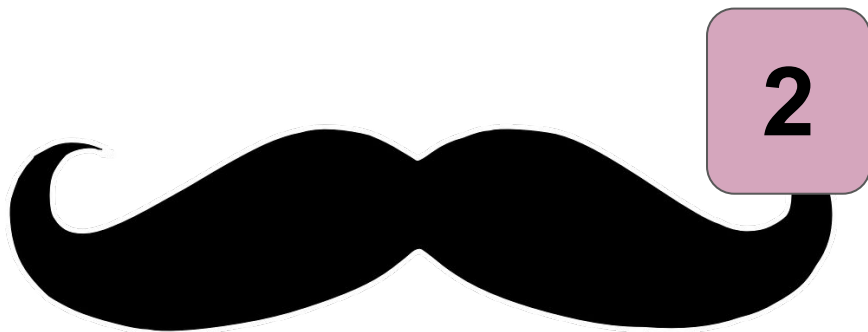


**Apelido carinhoso**

```
bigodinho = mustache()
```



# Variaveis -> Alias



**Mustache**

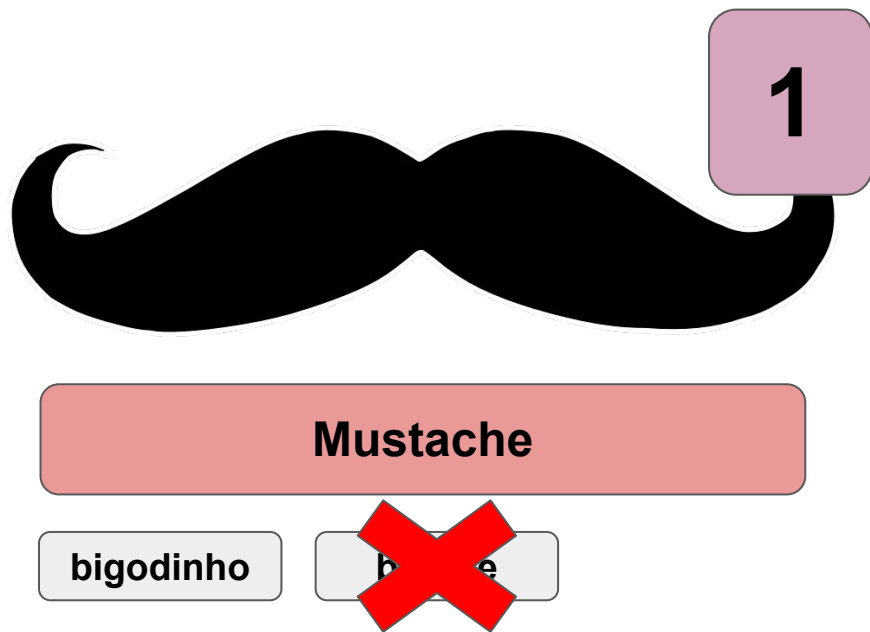
**bigodinho**

**bigode**

```
bigodinho = mustache()
```

```
bigode = bigodinho
```

# Variaveis -> Alias

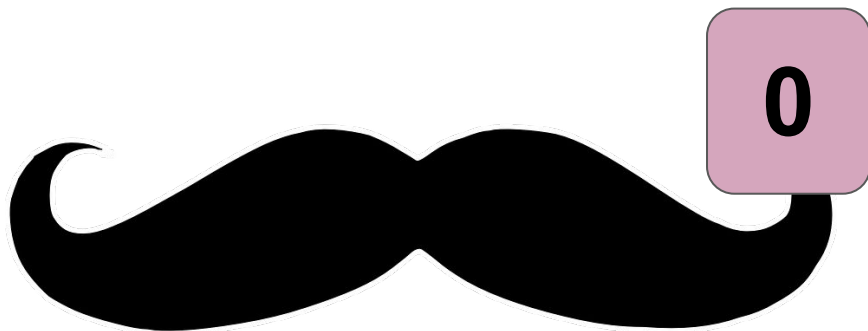


```
bigodinho = mustache()
```

```
bigode = bigodinho
```

```
del bigode
```

# Variaveis -> Alias



**Mustache**

~~bigodinho~~

~~bigode~~

```
bigodinho = mustache()
```

```
bigode = bigodinho
```

```
del bigode
```

```
del bigodinho
```

# Variaveis -> Alias



0

**Objeto  
destruído**

tache()

= bigodinho

**Mustache**

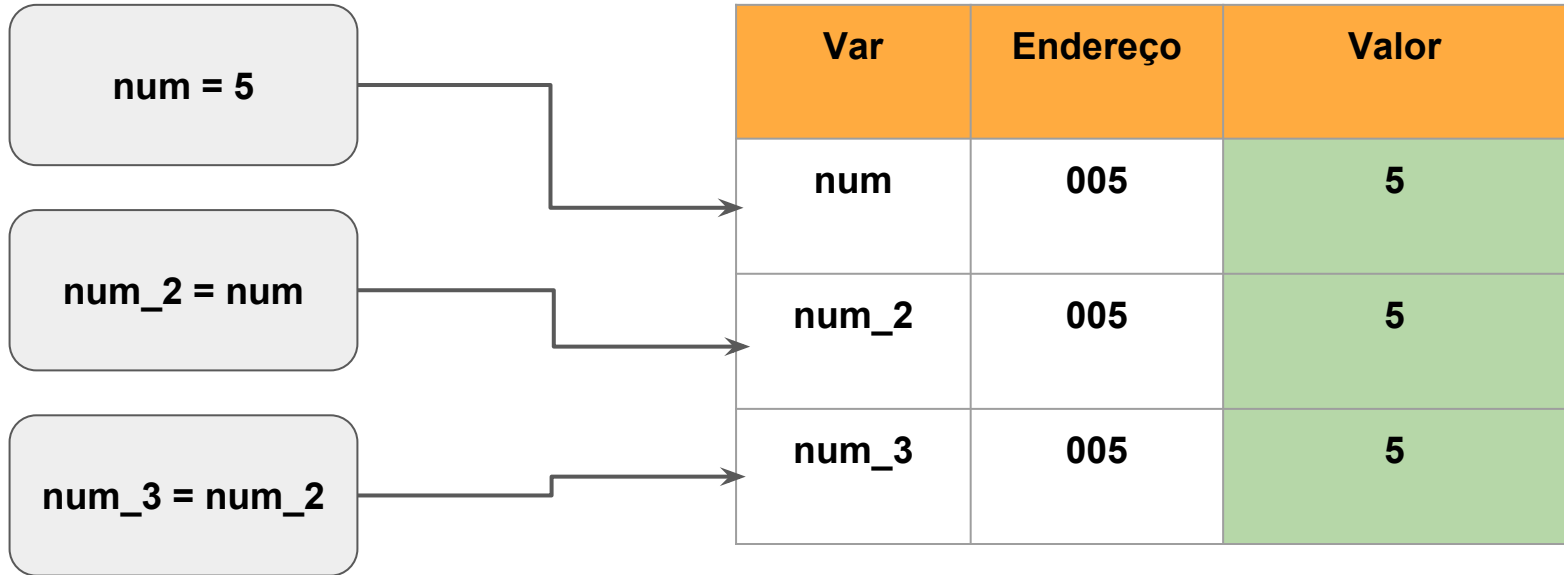
~~bigodinho~~

~~bigodinho~~

del bigode

del bigodinho

# Variáveis -> Alias



# Variáveis -> Alias

**num = 5**

**num\_2 = num**

**num\_3 = num\_2**

**num\_3 = 10**

Var	Endereço	Valor
num	005	10
num_2	005	10
num_3	005	10



# Tudo é objeto

**Um pouco sobre introspecção**

# Tudo é objeto

```
>>> num = 7  
>>> type(num)  
<class 'int'>
```

Atribuição de 7 a num

Retorna o tipo de dado  
contido na variável

Objeto inteiro



# Tudo é objeto

```
>>> num = 7  
>>> type(num)  
<class 'int'>
```

Atribuição de 7 a num

Retorna o tipo de dado  
contido na variável

Objeto inteiro

**As estruturas de  
dados tem atributos  
e métodos**

# Tudo é objeto

Retorna a lista de métodos do objeto

```
>>> type('a')  
      <class 'str'>
```

```
>>> dir('a')
```

```
>>> ['count', 'index', 'islower', 'lower', 'replace', 'upper']
```

# Tipos numéricos

**Int, Float, Complex**

# Números Inteiros

Tipo	Chamada	Resultado
Base 10	11	11
Base 2	0b11	3
Base 8	0o11	9
Base 16	0x11	17

```
>>> type(0x11)  
<class 'int'>
```

```
>>> print(0x11 + 0b11)  
20
```

# Números de ponto flutuante e complexos

```
>>> type(11.4)  
<class 'float'>
```

```
>>> type(11.0 + 1j)  
<class 'complex'>
```

```
>>> 11 + 1j + 11.04  
(22.04+1j)
```

# Operações com números [0]

**$+$ ,  $-$ ,  $*$ ,  $**$ ,  $/$ ,  $//$ ,  $\%$**

$$2 + 2 = 4$$

$$2 - 2 = 0$$

$$2 * 2 = 4$$

$$2 ** 2 = 4$$

$$2 / 2 = 1.0$$

$$2 \% 2 = 0$$

$$2 // 2 = 1$$


# Operações com números [1]

**$+$ ,  $-$ ,  $*$ ,  $**$ ,  $/$ ,  $//$ ,  $\%$**

**Toda operação de inteiros retorna inteiro\*;**

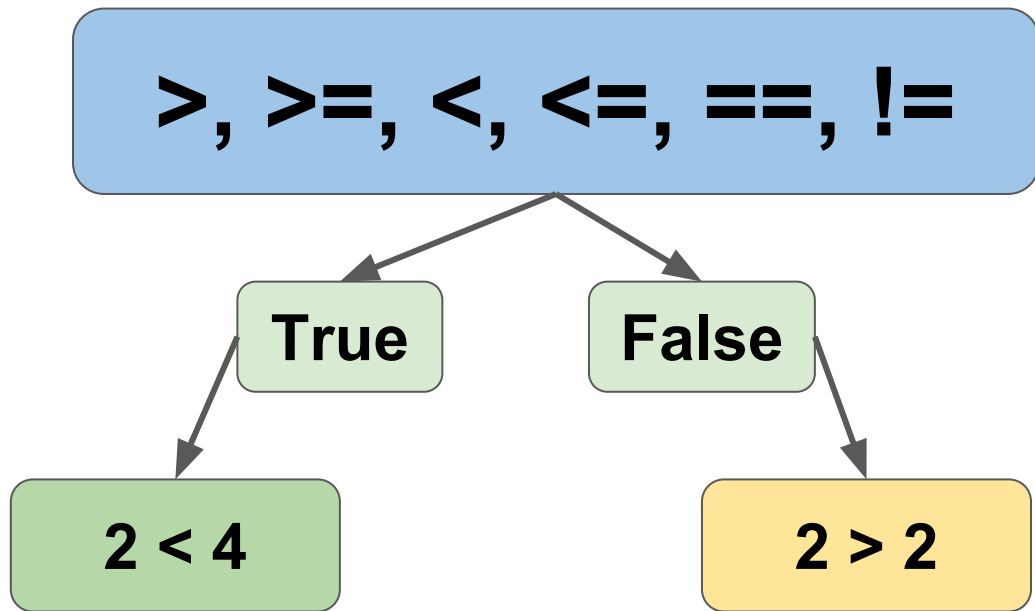
**Toda operação com um float retorna float;**

**Toda operação com complexos retorna complexos**


$$2 / 2 = 1.0$$

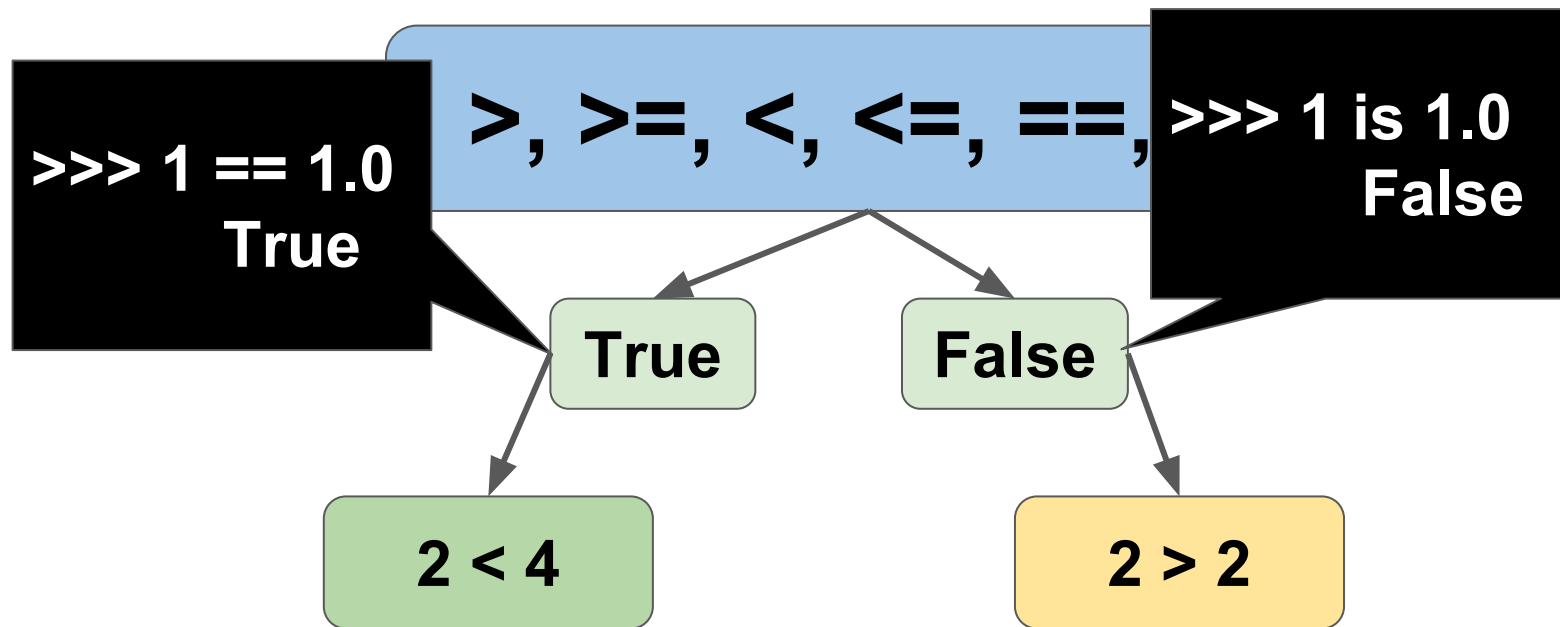
$$2 // 2 = 1$$

# Operações com números [2] - Bool

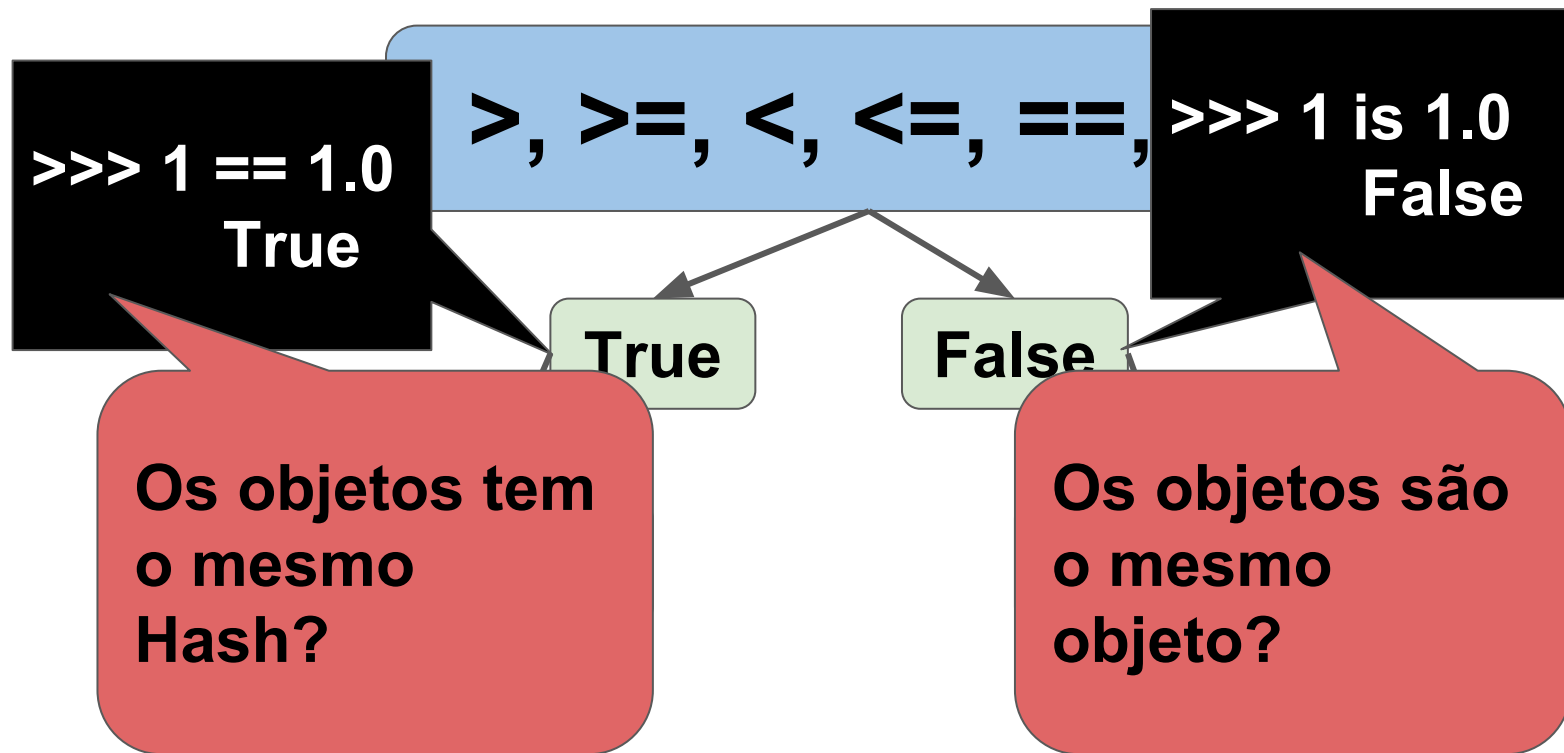




# Operações com números [3] - Bool



# Operações com números [3] - Bool



## Estruturas de decisão - if,elif,else

```
>>> x = 7
>>> y = 6
>>> if x == y:
    print("Mesmo valor")
elif x > y:
    print("%s" % "x é maior que y")
else:
    print(("Não sei resolver %s e %s" % (x,y)))
```

## Estruturas

Parenteses não são necessários,  
nem mesmo com conectivos, ex:  
if x>y or x<y

```
>>> x = 7
>>> y = 6
>>> if x == y:
    print("Mesmo valor")
elif x > y:
    print("%s" % "x é maior que y")
else:
    print(("Não sei resolver %s e %s" % (x,y)))
```

# Strings

**O básico necessário**

## Inicialização

```
brian = 'ROMANES EUNT DOMUS'
```

```
brian = "ROMANES EUNT DOMUS"
```

```
brian = """ROMANES  
EUNT  
DOMUS"""
```

## Concatenação

```
>>> brian + brian  
      ROMANES EUNT  
      DOMUSROMANES EUNT  
      DOMUS
```

```
>>> brian * 2  
      ROMANES EUNT  
      DOMUSROMANES EUNT  
      DOMUS
```

## Inicialização

```
brian = 'ROMANES EUNT DOMUS'
```

```
brian = "ROMANE
```

```
brian = """ROMANES  
EUNT  
DOMUS"""
```

Multilinha

## Concatenação

```
>>> brian + brian  
      ROMANES EUNT  
      DOMUSROMANES EUNT  
      DOMUS
```

```
>>> brian * 2  
      ROMANES EUNT  
      DOMUSROMANES EUNT  
      DOMUS
```

# UTF-8 - Enconde de strings

ASCII/8859-1 Text

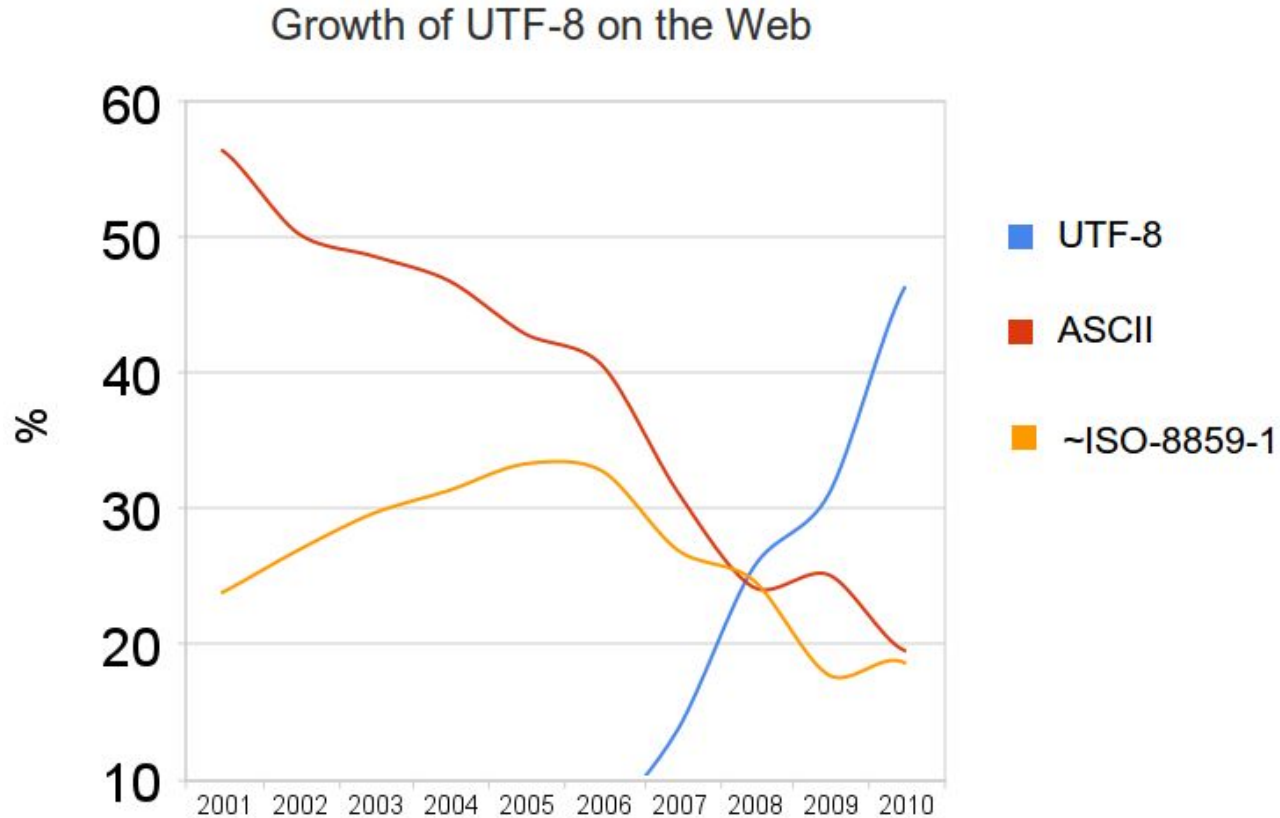
A	0100 0001
S	0101 0011
C	0100 0011
I	0100 1001
I	0100 1001
/	0010 1111
8	0011 1000
8	0011 1000
5	0011 0101
9	0011 1001
-	0010 1101
l	0011 0001
	0010 0000
t	0111 0100
e	0110 0101
x	0111 1000
t	0111 0100

Unicode Text

A	0000 0000 0100 0001
S	0000 0000 0101 0011
C	0000 0000 0100 0011
I	0000 0000 0100 1001
I	0000 0000 0100 1001
	0000 0000 0010 0000
天	0101 1001 0010 1001
地	0101 0111 0011 0000
	0000 0000 0010 0000
ج	0000 0110 0011 0011
ج	0000 0110 0100 0100
ا	0000 0110 0011 0111
م	0000 0110 0100 0101
	0000 0000 0010 0000
α	0000 0011 1011 0001
κ	0010 0010 0111 0000
γ	0000 0011 1011 0011



# UTF-8 - Enconde de strings



# Métodos - Strings

```
>>> a = 'Abracadabra'
```

```
>>> a.count('a') #4
```

```
>>> a.index('c') #4
```

```
>>> a.partition('c')  
# ('Abra', 'c', 'adabra')
```

```
>>> a.replace('a','c')  
# 'Abrcccdcbrc'
```

```
>>> a.lower()  
# 'abracadabra'
```

```
>>> a.split('a')  
# ['Abr', 'c', 'd', 'br', '']
```

# Métodos - Strings

```
>>> a = 'Abracadabra'
```

```
>>> 'a' in 'abracadabra'  
True
```

retorna uma  
Tupla

```
a.count('a') #4
```

```
>>> a.partition('c')  
# ('Abra', 'c', 'adabra')
```

```
>>> a.lower()  
# 'abracadabra'
```

```
>>> a.index('c') #4
```

```
>>> a.replace('c', 'd')  
# 'Abrccadabra'
```

```
>>> a.split('a')  
# ['Abr', 'c', 'd', 'br', '']
```

retorna uma  
Lista

# While

```
>>> while BOOL:  
    do  
else:  
    do
```

```
>>> while 3 < 4:  
    print("Três é menor")
```

```
>>> while True:  
    # ???
```

```
>>> x = 0  
>>> while x > 10:  
    x = input("um num")
```

# While

Nunca usei na vida

```
>>> while  
do  
else:  
do
```

```
>>> while 3 < 4:  
    print("Três é menor")
```

Entrada do teclado

```
>>> while True:  
    # ???
```

```
>>> x = 0  
>>> while x > 0:  
    x = input("um num")
```


# Listas

**Sim, elas aceitam tudo**

# Elas aceitam todos os tipos de objeto - Listas

```
>>> a = [1, 1. + ij, "eduardo", [1,2,3], (1,2)]
```

```
>>> a[0] # 1  
>>> a[1] # 1. +j  
>>> a[2] # "eduardo"  
>>> a[3] # [1,2,3]
```



```
>>> a[3][0] # 1  
>>> a[1][1] # 2  
>>> a[4][0] # 1  
>>> a[2][-1] # "o"
```

# Slice - Listas

```
>>> n = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
>>> n[0]      # 0  
>>> n[6:]     # [6, 7, 8, 9]  
>>> n[:-6]    # [0, 1, 2, 3]  
>>> n[::2]    # [0, 2, 4, 6, 8]
```



**[De onde : até onde: de quanto em quanto]**

**[2 : 10 : 3]**



```
>>> n[::2] # [0, 1, 2, 3]  
>>> n[::2] # [0, 2, 4, 6, 8]
```

## Slice - Listas

```
>>> matriz = [ [0, 1, 2], [3, 4, 5], [7, 8, 9] ]
```

```
>>> matriz[0]                # [0, 1, 2]
```

```
>>> matriz[0][1:]           # [1, 2]
```

```
>>> matriz3d = [ [ [0, 0, 0] ], [ [0, 0, 0] ] ]
```

```
>>> matriz3d[0][0][0]       # 0
```

# Métodos - Listas

```
>>> x = [1, 2, 3]
```

```
>>> x.append(4)  
# [1,2,3,4]
```

```
>>> x.remove(2)  
# [1, 3]
```

```
>>> x.insert(4)  
# [4,1,2,3]
```

```
>>> x.pop()  
# [1, 2]
```

```
>>> x.count(2)  
# 1
```

```
>>> x.reverse()  
# [3, 2, 1]
```

# Métodos - Listas

**Fila**

```
>>> x = [1, 2, 3]
```

```
>>> x.append(4)  
# [1,2,3,4]
```

```
>>> x.insert(4)  
# [4,1,2,3]
```

```
>>> x.count(2)  
# 1
```

**Lista**

```
>>> x.remove(2)  
# [1, 3]
```

```
>>> x.pop()  
# [1, 2]
```

```
>>> x.reverse()  
# [3, 2, 1]
```

**Pilha**

# For

```
>>> for e in <iter>:  
    do  
else:  
    do
```

```
>>> for e in "grupy":  
    print(e)
```

```
>>> for e in [1,2,3]:  
    print(e)
```

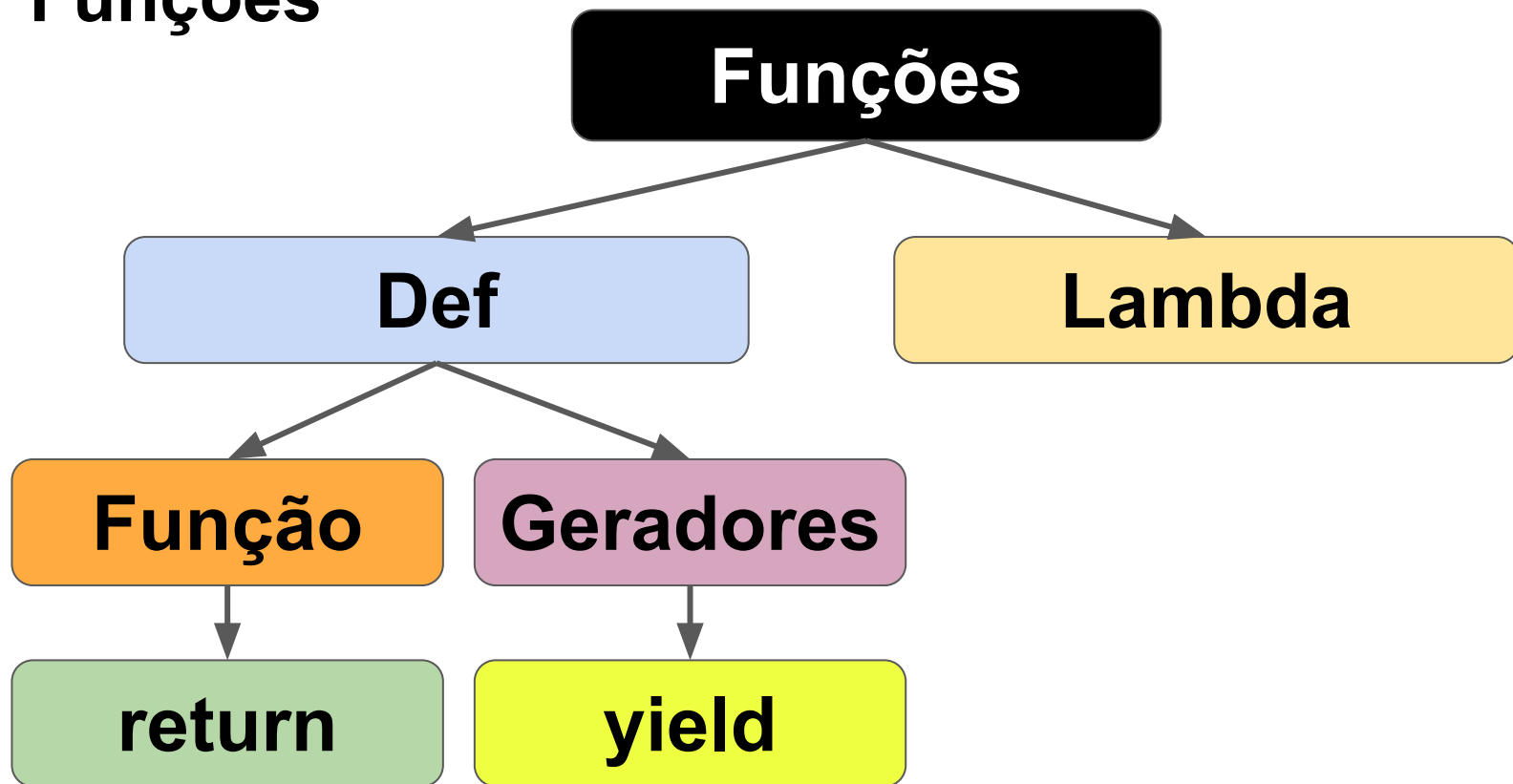
```
>>> for i,e in enumerate([1,2,3]):  
    print(i,e)
```

```
# 0 1  
# 1 2  
# 2 3
```

# Funções

**Elas também são objetos**

# Funções



## **Padrões[0] - Funções**

### **Função nomeada**

```
def nome (args):  
    return args
```

### **Função anônima**

```
lambda args: op(args)
```

### **Função geradora**

```
def nome (args):  
    yield args
```



## Padrões[1] - Funções

```
def func(a, b=1, *c, **d):  
    return a, b, c, d
```

```
func( * [1, 2, 3, 4, 5], x=7 )  
# 1 2 (3, 4, 5) {'x': 7}
```

## Padrões[2] - Funções

```
def func(a, b=1, *c, **d):  
    return a, b, c, d
```

```
func(*[1, 2, 3, 4, 5], x=7 )  
# 1 2 (3, 4, 5) {'x': 7}
```

**Empacotar**

**Dicionário**

**Desempacotar**

# Tuplas

**Elas não são só listas imutáveis**

# Métodos - Tuplas

```
>>> x = (1, 2, 3)
```

```
>>> x.count(2)  
# 1
```

```
>>> x.index(2)  
# 1
```

```
>>> x = 1  
>>> y = 2  
>>> x,y = y,x
```

```
>>> print(x)  
# 2  
>>> print(y)  
# 1
```

# “Empacotamento” - Tuplas

```
>>> x = ( 1, 2, 3, 4, 5 )
```

```
>>> *a, b, c = x  
# (1, 2, 3) 4 5
```

```
>>> a, *b, c = x  
# 1 (2, 3, 4) 5
```

```
>>> a, b, *c = x  
# 1, 2, (4, 5)
```

```
>>> *a, b, *c = x  
# (1, 2) 3 (4, 5)
```

# “Empacotamento” - Tuplas

```
>>> x = ( 1, 2, 3, 4, 5 )
```

```
>>> *a, b, c = x  
# (1, 2, 3) 4 5
```

```
>>> a, *b, c = x  
# 1 (2, 3, 4) 5
```

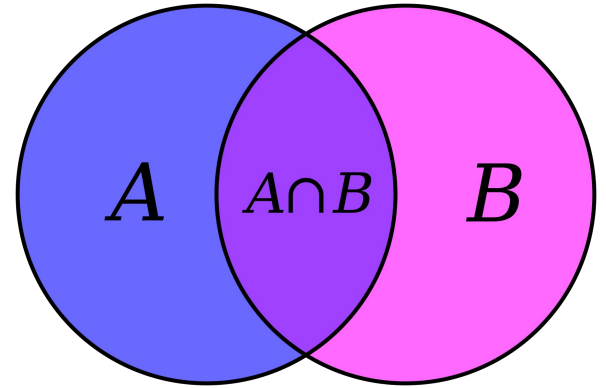
```
>>> a, b, *c = x  
# 1, 2, (4, 5)
```

```
>>> a, b, *c = x  
# 1, 2, (4, 5)
```

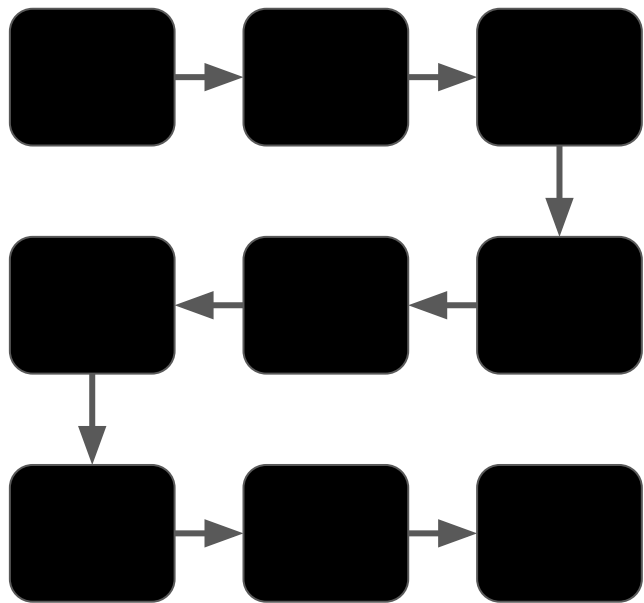


# Conjuntos

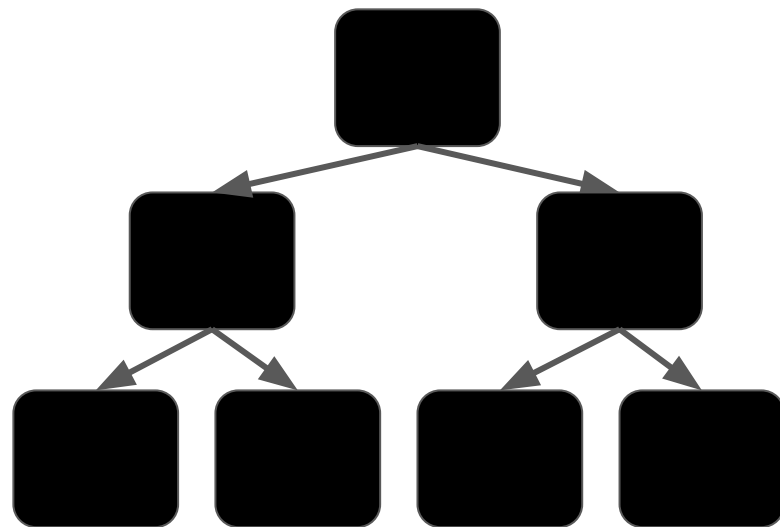
Valores “fixos”



# Que raios são hashable? - Conjuntos



**Litas e Tuplas**



**Conjuntos e dicionários**



# Métodos - Conjuntos

```
>>> x = {1, 2, 3}; y = {3, 4, 5}
```

```
>>> x.union(y)  
# {1, 2, 3, 4, 5}
```

```
>>> x.intersection(y)  
# {3}
```

```
>>> x.difference(y)  
# {1, 2}
```

```
>>> x.update(y)  
# {1, 2, 3, 4, 5}
```

```
>>> x.discard(1)  
# {2, 3}
```

```
>>> x.pop()  
# {2, 3}
```

# Métodos - Conjuntos

```
>>> x = {1, 2, 3}; y = {3, 4, 5}
```

Retorna um novo

```
>>> x.union(y)  
# {1, 2, 3, 4, 5}
```

```
>>> x.difference(y)  
# {1, 2}
```

Remove o 1º

```
>>> x.discard(1)  
# {2, 3}
```

```
>>> x.intersection(y)  
# {3}
```

Atualiza x

```
>>> x.update(y)  
# {1, 2, 3, 4, 5}
```

Remove o 1º

```
>>> x.pop()  
# {2, 3}
```

# Dicionários

**Programador: Humano responsável por transformar café em linhas de código**

# Como eles funcionam? - Dicionários

- A chave necessariamente deve ser hashable
- Os valores podem ser qualquer coisa
- Similar à um JSON

```
Pessoa = {  
    'nome': 'eduardo',  
    'cargo': 'programador',  
    'função': lambda f, *args: f(args),  
    'saldo': {'dia 5': 150, 'dia 10': [0, -100, -500]}  
}
```

# Como uso esse negócio? - Dicionários

```
>>> x = {'Maria': 50, 'Juana': 25}
```

```
>>> x['Maria']  
# 50
```

```
>>> x.keys()  
# ['Juana', 'maria']
```

```
>>> x['Maria'] = 10
```

```
>>> x.values()  
# [50, 25]
```

```
>>> x.popitem()  
# ('Juana', 25)
```

```
>>> x.setdefault('Carlos')  
# {'Juana': 25, 'maria': 100, 'Carlos':  
  None}
```