

85^a EDIÇÃO

SEQ UFRJ
20 a 24 de agosto



Introdução à programação para ciência e engenharia em *Python*

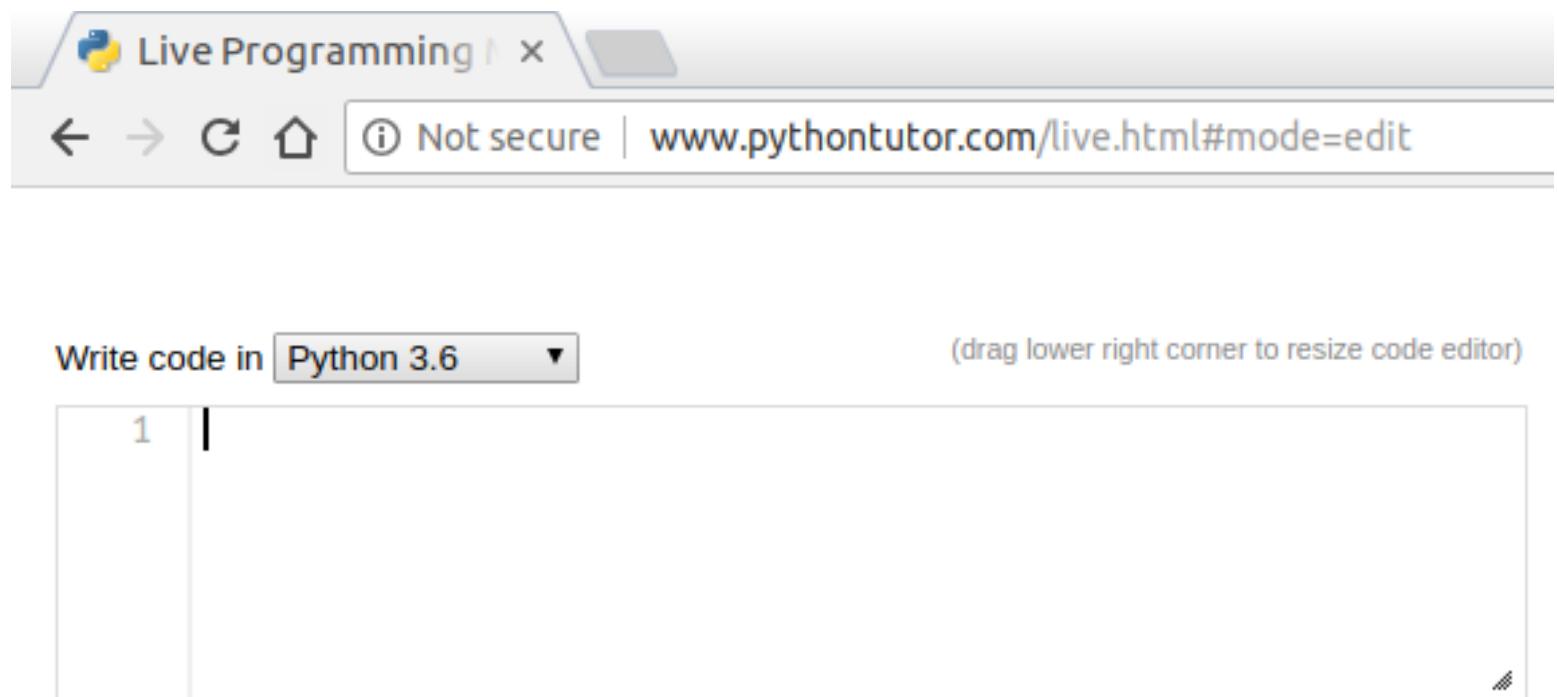
Iuri Soter Viana Segtovich

Parte 2: Lógica e Sintaxe

Tipo texto, números e operadores: (string, int, float)

python tutor

[www.pythontutor.com/
live.html#mode=edit](https://www.pythontutor.com/live.html#mode=edit)



→ line that has just executed

→ next line to execute

Print #imprimir

```
print ("Olá!")
```

1.

2.

3.

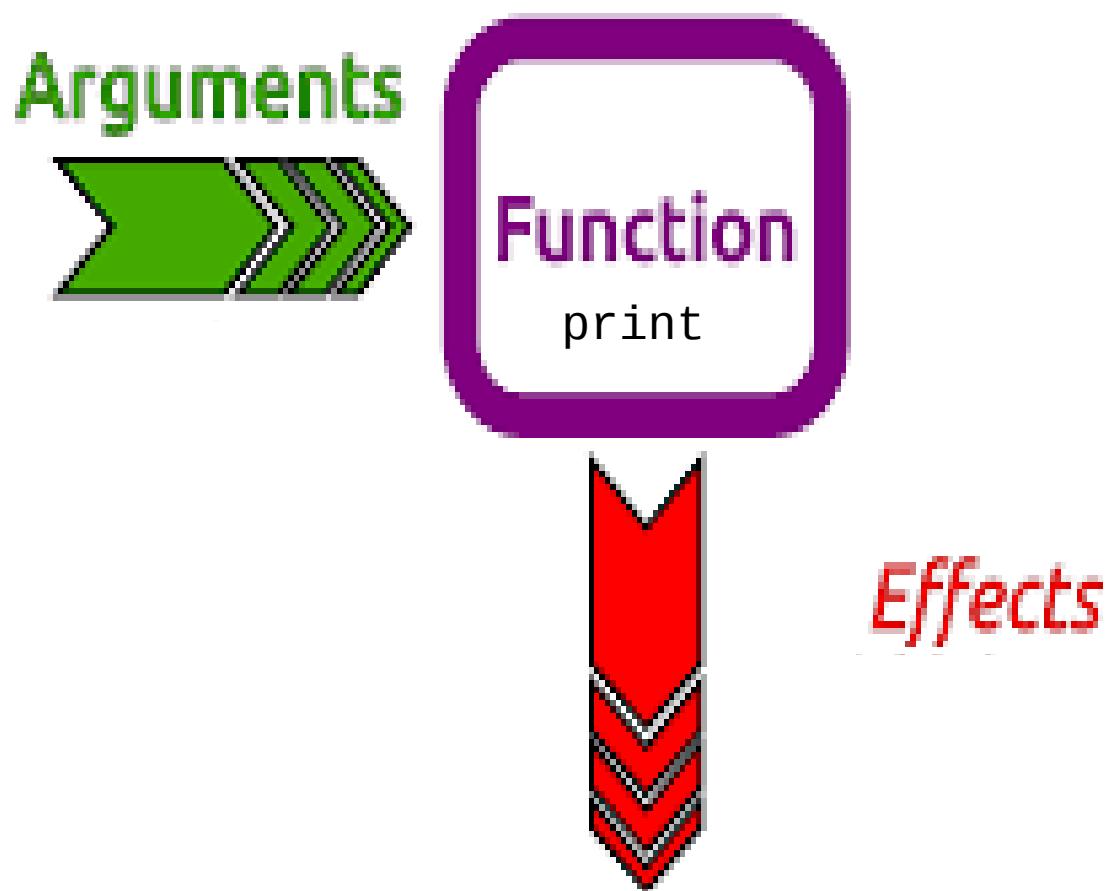
4.

```
Olá!
```

5.

- Código fonte
 - 1. O nome da função
 - 2. Abre parênteses
 - 3. O argumento
 - 4. Fecha Parênteses
- 5. Terminal de impressão

O que está acontecendo?



Argumentos para o print

```
print ("Olá!")
```

Olá!



- Um string
 - Uma sequência de caracteres



```
1 print("Olá!")
2 print('Tudo bem?')
3 print('''Tudo
4 certinho'''')
5 print("""Como é
6 que vai você?""")
```

- Aspas duplas
- Aspas simples
- Três aspas simples ou duplas (*multiline*)

Vários argumentos separados por vírgula

```
→ 1 print("oi!", 'tudo bem?', '''tudo certinho'''')
```

```
oi! tudo bem? tudo certinho?
```



Outros tipos de objeto que vêm a ser representados por strings.

- Tipo números

```
1 print(1)
2
→ 3 print(3.1415)
```

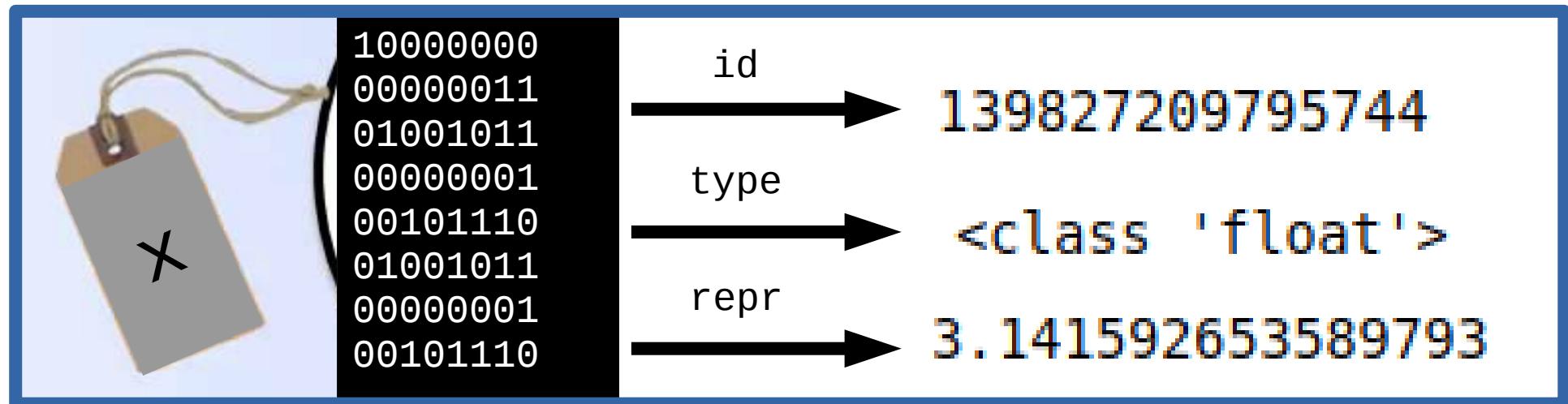
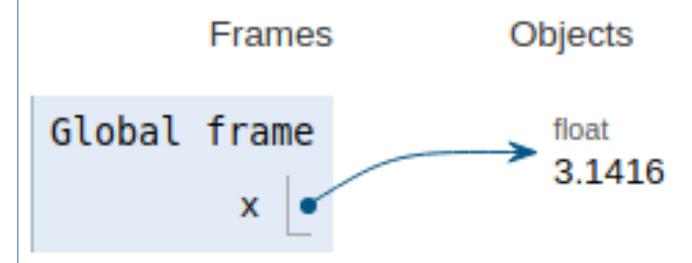
```
1
3.1415
```

O que são objetos e tipos? e nomes, identificações e representações?

```
1 x=3.1415926535897932384626433
2 print( "id: ", id(x) )
3 print( "type: ", type(x) )
→ 4 print( "repr: ", repr(x) )
```

```
id: 139827209795744
type: <class 'float'>
repr: 3.141592653589793
```

Gerar um objeto do tipo float a partir da expressão literal 3.1415926535897932384626433 e vinculá-lo ao nome x.

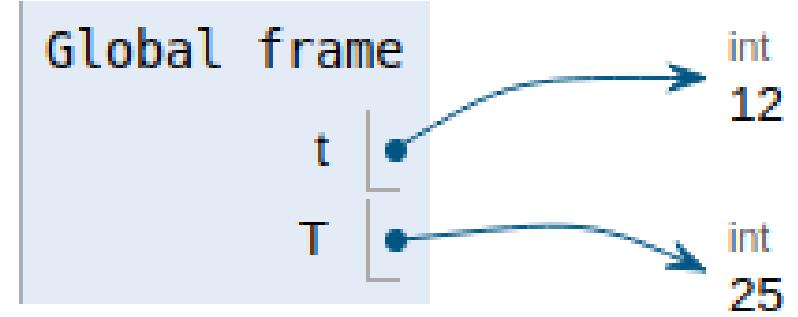


Nomes válidos

- Letras
- Underline
- Algarismos
 - Não pode começar com algarismos
- _nomes ou __nomes começando com um ou dois underline são utilizados para funções especiais
- Os nomes são CASE SENSITIVE
- Não pode espaço

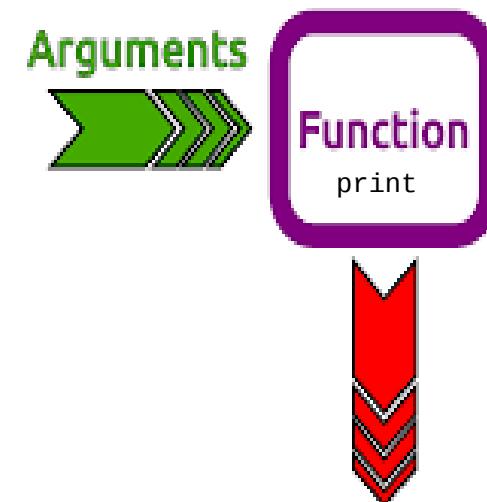
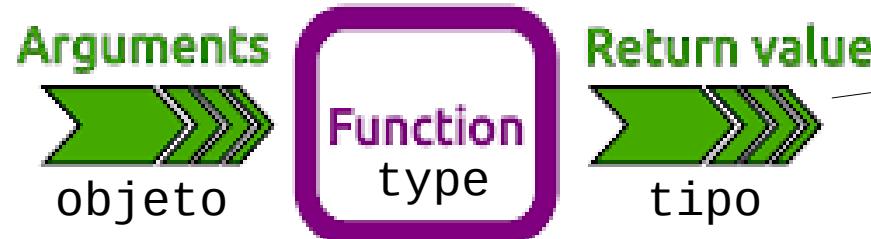
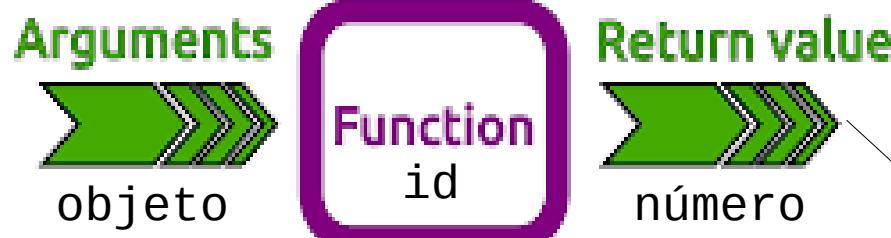


```
1  t= 12 #tempo
→ 2  T= 25 #TEMPERATURA|
```



Mais funções

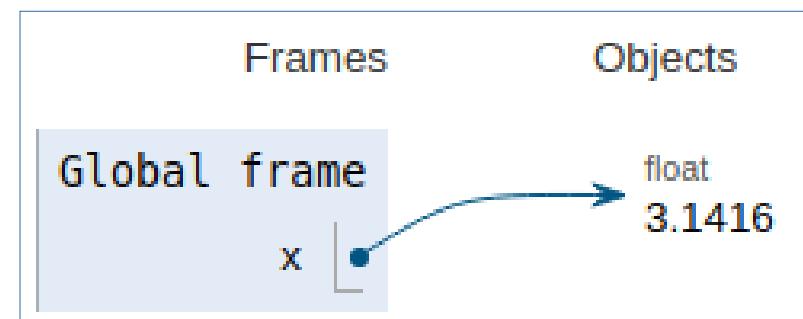
```
x = 3.1415926535897932384626433  
print( id( x ) )  
print( type( x ) )  
print( repr( x ) )
```



comentários

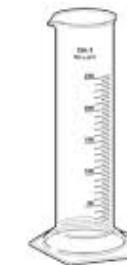
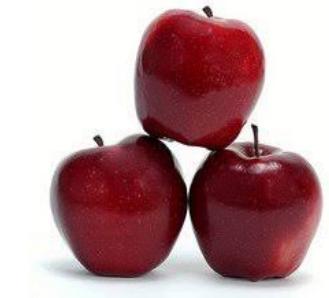
```
1 # vinculando um objeto float com o valor de pi ao nome x
2 x = 3.1415926535897932384626433
3 # imprimindo o número de identificação do objeto vinculado ao nome x
4 print ( id( x ) )
5 # imprimindo o número de identificação do objeto vinculado a x
6 print ( type( x ) )
7 # imprimindo a representação do objeto vinculado a x
8 print ( repr( x ) )
```

```
140572334176416
<class 'float'>
3.141592653589793
```



Outros tipos

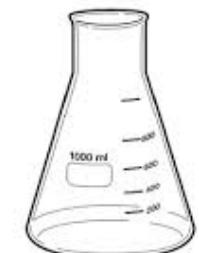
- Numbers
 - int
 - float
 - complex
- string
- bool



Graduated cylinder



Beaker



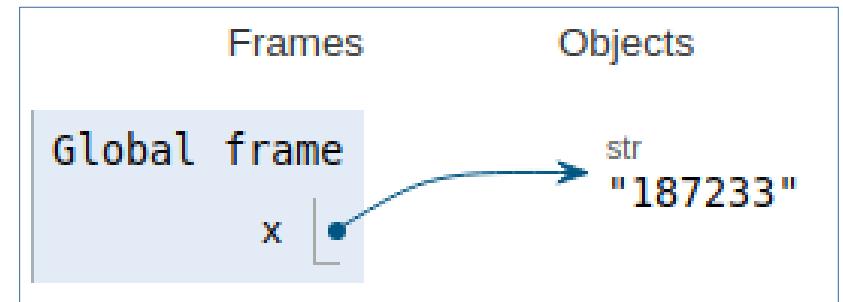
Erlenmeyer flask

$$i^2 = -1$$



input

```
1 | x=input()
```



Submit



Side Causes

Function
input

Return value

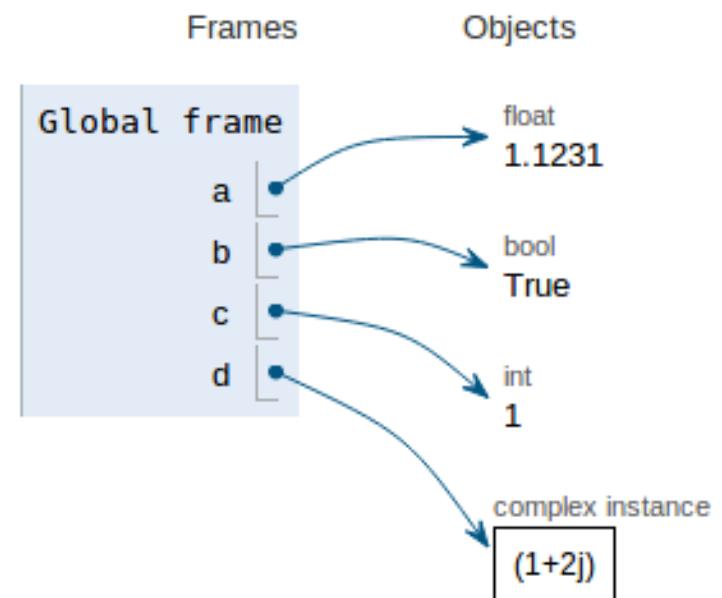


- Sempre interpreta a entrada como texto - retorna um string.
- Parênteses vazios indica chamada sem argumentos.

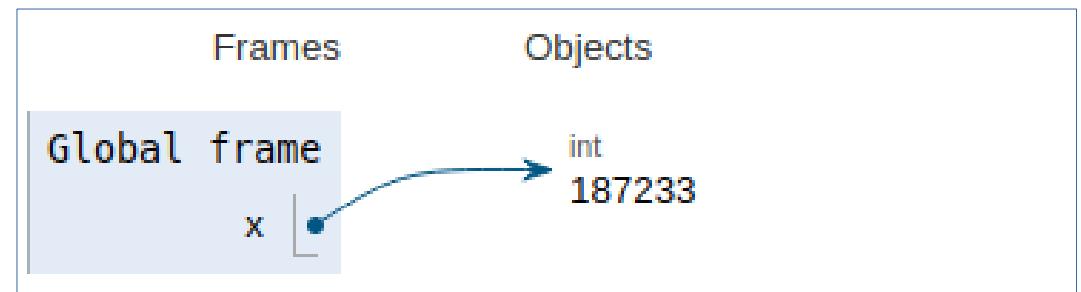
Conversões

- As funções de conversão tem os nomes dos tipos

```
1 a=float("1.123123")
2 b=bool("True")
3 c=int("1")
→ 4 d=complex("1+2j")
|
```



```
→ 1 x=int(input())
```

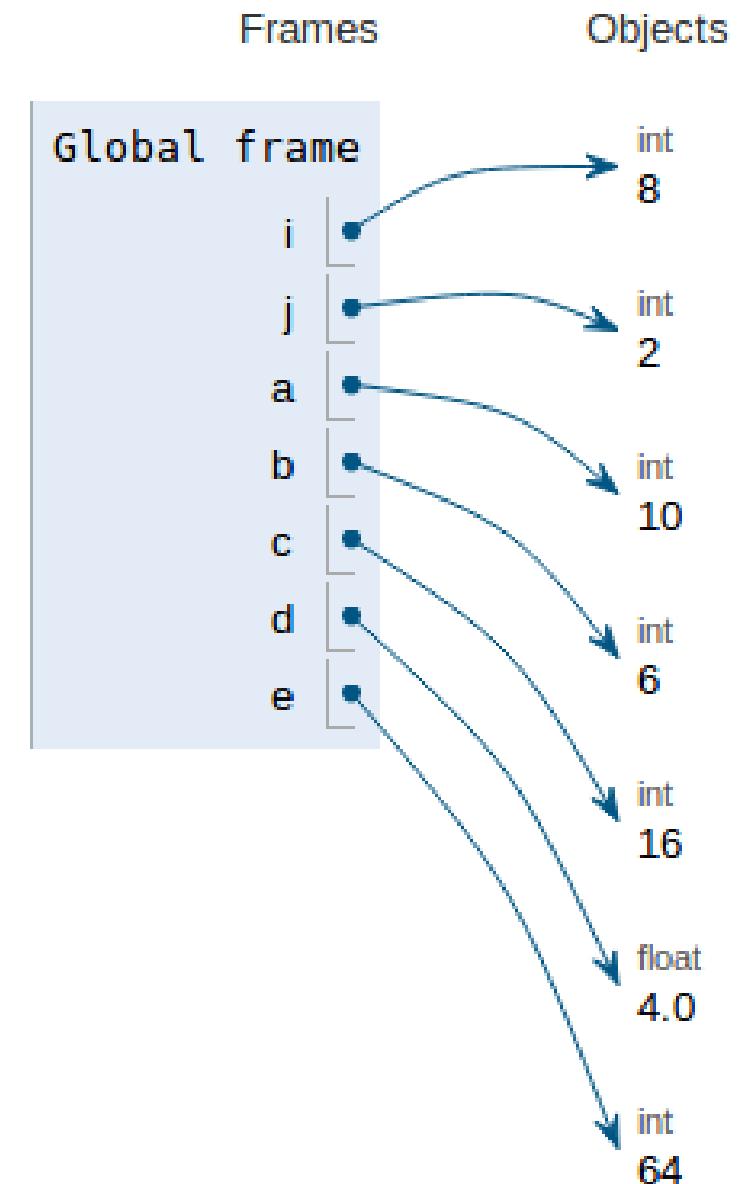


187233

Submit

Cada tipo tem suas operações

```
1 i=8  
2 j=2  
3 a=i+j  
4 b=i-j  
5 c=i*j  
6 d=i/j  
7 e|i**j
```



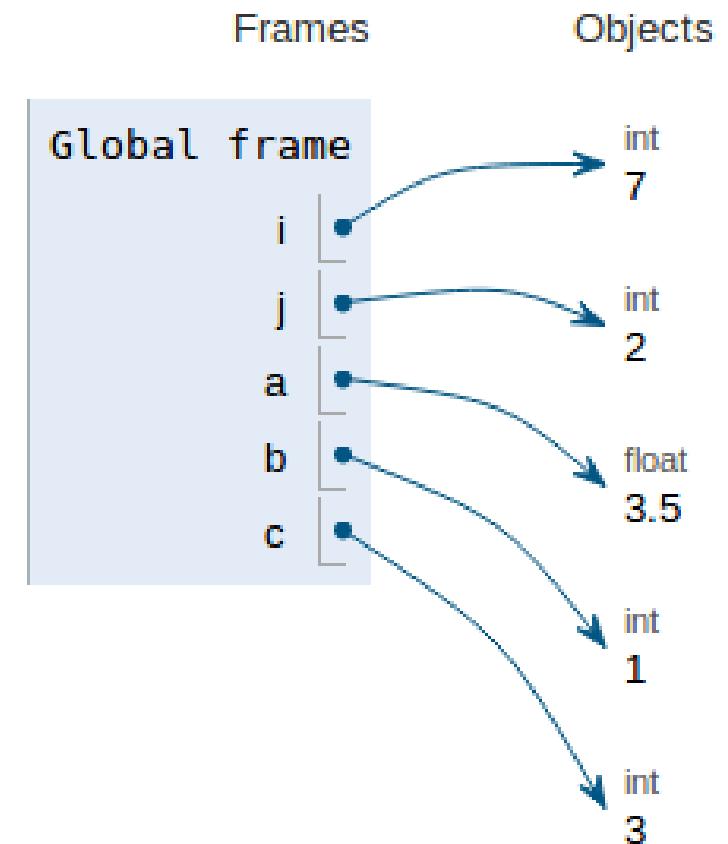
Operadores são funções por trás dos panos

```
object. __add__ (self, other)
object. __sub__ (self, other)
object. __mul__ (self, other)
object. __matmul__ (self, other)
object. __truediv__ (self, other)
object. __floordiv__ (self, other)
object. __mod__ (self, other)
object. __divmod__ (self, other)
object. __pow__ (self, other[, modulo])
object. __lshift__ (self, other)
object. __rshift__ (self, other)
object. __and__ (self, other)
object. __xor__ (self, other)
object. __or__ (self, other)
```

These methods are called to implement the binary arithmetic operations (+, -, *, @, /, //, %, `divmod()`, `pow()`, **, <<, >>, &, ^, |).

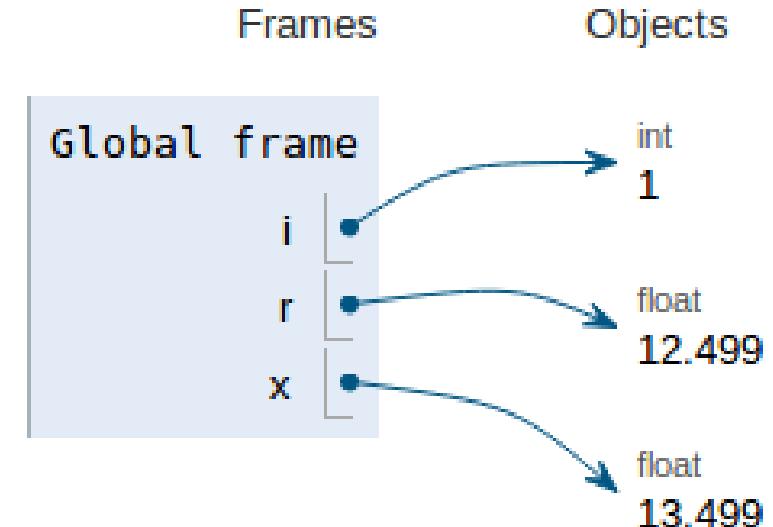
Divisão de inteiros

```
1 i=7  
2 j=2  
3  
4 a=i/j #retorna float|  
5  
6 b=i%j #retorna o resto da divisão  
7 c=i//j #retorna o quociente da divisão
```

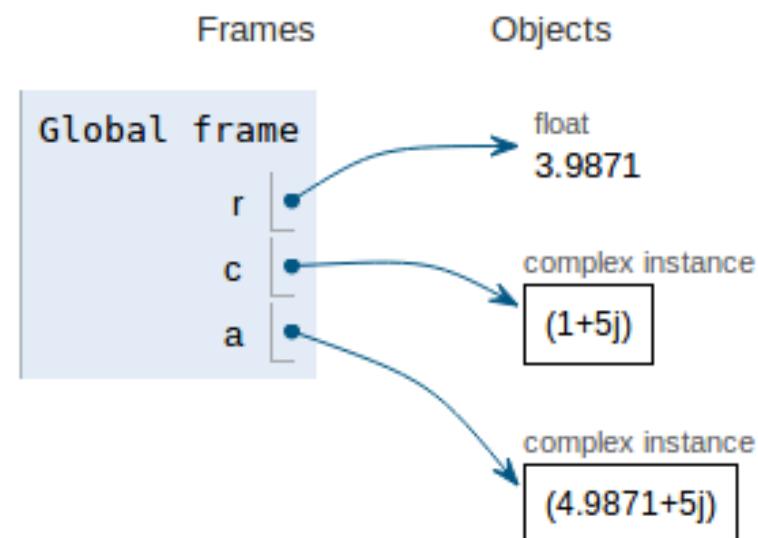


Conversões automáticas

```
1 i=1
2 r=12.49897
→ 3 x=i+r|
```



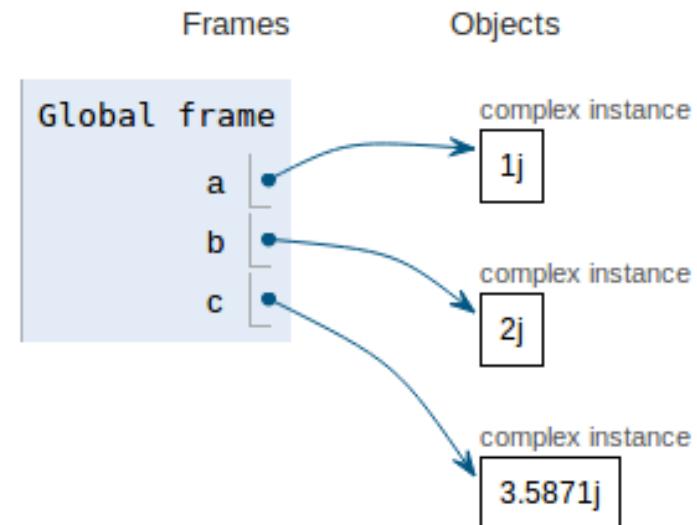
```
1 r=3.9871
2 c=1+5j
→ 3 a=r+c|
```



A expressão literal do complexo é 1j

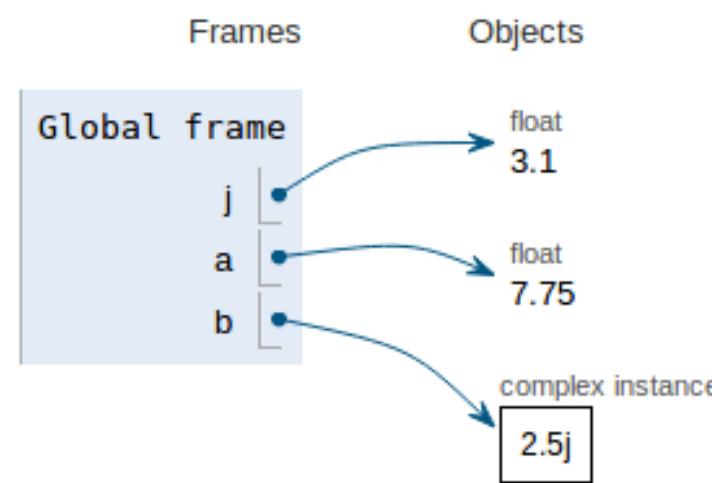
- Também valem seus múltiplos inteiros e decimais

```
1   a=1j  
2   b=2j  
→ 3   c=3.5871j
```



- Um j sozinho indica um nome

```
1   j=3.1  
2   a=2.5*j  
→ 3   b=2.5j
```



Precedência

1) funções

2) **

3) *, /, %, //

4) +, -

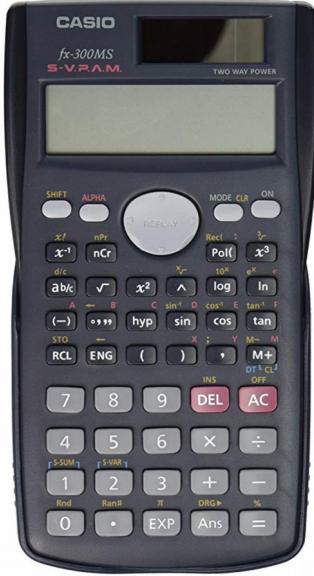
Frames

```
1 a = 2. ** 10. + 5. * 3. / 2. / 4.  
2 b = ((2. ** 10.) + (((5. * 3.) / 2.) / 4.))  
→ 3 c = 2. ** (10. + 5.) * (3. / (2. / 4.))|
```

Global frame	
a	1025.875
b	1025.875
c	196608.0

- Parênteses agrupam termos e estabelecem precedência

math



```
1 import math
2 print(math.pi)
3 print(math.log10(10))
4 print(math.e, math.exp(1))
5 print(math.log(math.e))
6 print(math.sin(0))
7 print(math.cos(0))
8 print(math.sin(math.pi/2.))
9 print(math.cos(math.pi/2.))
```

```
3.141592653589793
1.0
2.718281828459045 2.718281828459045
1.0
0.0
1.0
1.0
6.123233995736766e-17
```

Frames

Objects

Global frame
math

module instance

- Comando import
 - Dá acesso aos objetos de um pacote
 - Funções
 - Constantes
- Import umModulo as apelido

```
1 import math as m
2 print(m.pi)
```

float x real



```
4 pi_str="3.14159265358979323846264338327950288419716939937510"  
5 pi=float(pi_str)  
6 print(pi_str)  
7 print(pi)
```

```
| 3.14159265358979323846264338327950288419716939937510  
| 3.141592653589793
```

- Real: infinitas casas decimais
- Float: truncado após aproximadamente 16 algarismos significativos

Type decimal

```
from decimal import Decimal as dec
from decimal import localcontext

from decimal import getcontext

print(getcontext())
x=dec(1)/dec(7)
print(x)
y=x+x+x+x+x+x+x
print(y)
```

```
with localcontext() as ctx:
    print(ctx)
    ctx.prec=50
    x=dec(1)/dec(7)
    print(x)
    y=x+x+x+x+x+x+x
    print(y)
```

```
Context(prec=28, rounding=ROUND_HALF_EVEN, Emin=-999999, Emax=999999, capitals=1, clamp=0, flags=[], traps=[InvalidOperation])
0.1428571428571428571428571429
1.0000000000000000000000000000000000
Context(prec=28, rounding=ROUND_HALF_EVEN, Emin=-999999, Emax=999999, capitals=1, clamp=0, flags=[Inexact, Rounded])
0.14285714285714285714285714285714285714
0.999999999999999999999999999999999999999999999
```

O vínculo não é permanente

```
1 x=1
2 print(x)
3
4 x=3
5 print(x)|
```

```
1
3
```

Exemplo

```
1 dinheiro=50
2 custo = 5 #café
3 troco=dinheiro-custo
4 dinheiro=troco
5 print(dinheiro)
6
7 custo = 20 #almoço
8 troco=dinheiro-custo
9 dinheiro=troco
→ 10 print(dinheiro)|
```

45
25

Binding não é equação

```
1 x=3
2 print(x)
3
4 x=2*x+1|
5 # x <= 2*x+1
6 # 2*3+1
7 # x <= 7
8 print(x)
```

```
3
7
```

- Avaliar as operações do lado direito
- Vincular o resultado ao nome do lado esquerdo

Exemplo

```
1 dinheiro=50
2 print(dinheiro)
3
4 custo=5
5 dinheiro=dinheiro-custo
6 print(dinheiro)
7
8 custo=10
9 dinheiro=dinheiro-custo
→ 10 print(dinheiro)
```

```
50
45
35
```

SYNTATIC SUGAR

```
1 x=1
2 print(x)
3 x+=1
4 print(x)
5 x-=10
6 print(x)
7 x*=9
8 print(x)
9 x/=10
→ 10 print(x)
```

```
1
2
- 8
- 72
- 7.2
```

Mais funções por trás dos panos

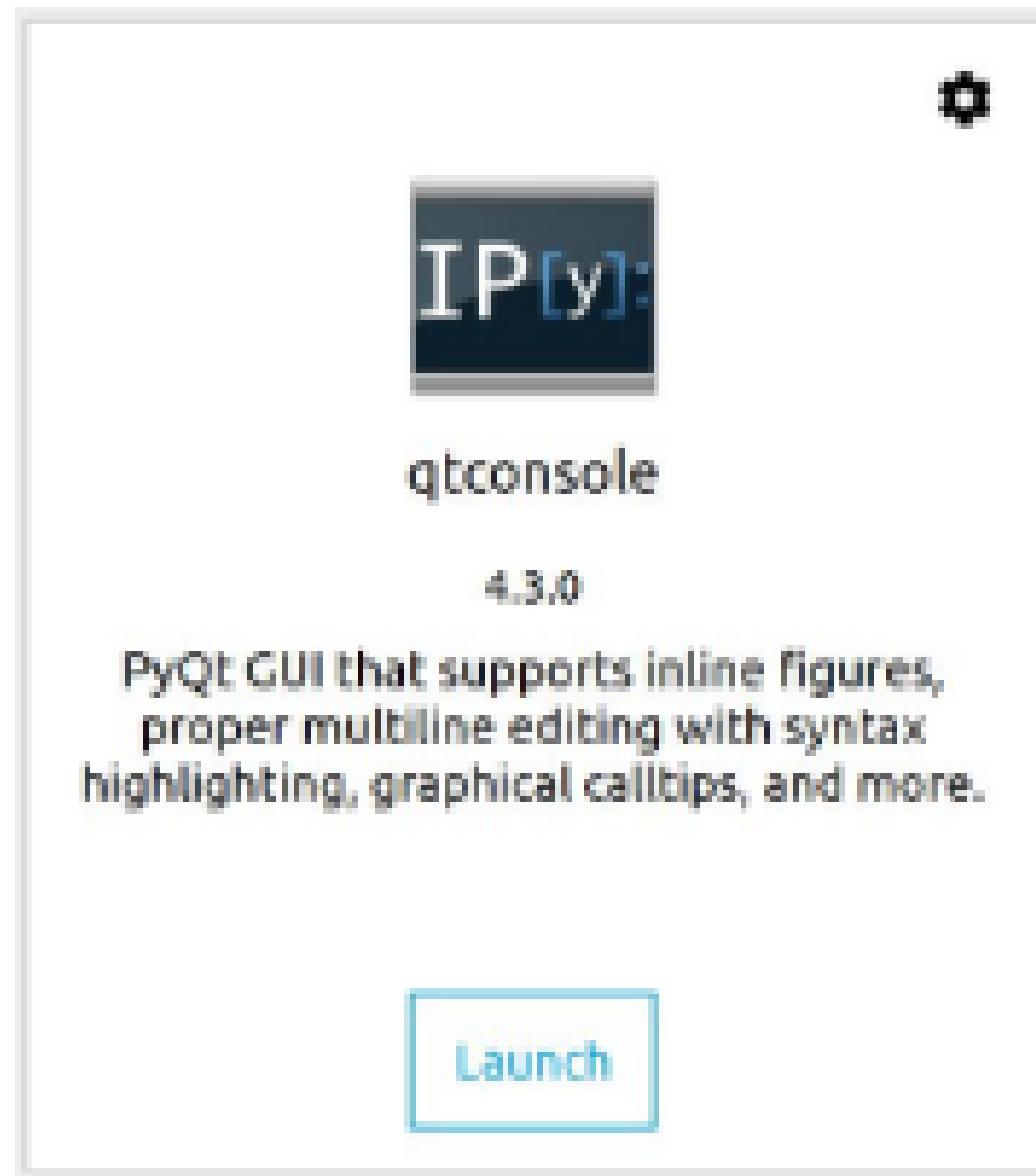
```
object.__iadd__(self, other)
object.__isub__(self, other)
object.__imul__(self, other)
object.__imatmul__(self, other)
object.__itruediv__(self, other)
object.__ifloordiv__(self, other)
object.__imod__(self, other)
object.__ipow__(self, other[, modulo])
object.__ilshift__(self, other)
object.__irshift__(self, other)
object.__iand__(self, other)
object.__ixor__(self, other)
object.__ior__(self, other)
```

These methods are called to implement the augmented arithmetic assignments (`+=`, `-=`, `*=`, `@=`, `/=`, `//=`, `%=`, `**=`, `<<=`, `>>=`, `&=`, `^=`, `|=`).

O python como uma calculadora

- Calcular quantos segundos tem em um ano.
- Dividir o custo do churrasco.
- Calcular o juros composto em um ano para 1% mensal.
- Converter unidades.
- Fazer as contas dos exercícios de introdução aos cálculos de processos.

Anaconda navigator / qtconsole



Ipython

```
In [4]: churrasco=1250.34
In [5]: pessoas=50
In [6]: caixa=300
In [7]: (1250.34-300)/50
Out[7]: 19.0068
In [8]: _*4
Out[8]: 76.0272
In [9]: %reset
Once deleted, variables cannot be recovered. Proceed (y/[n])? n
Nothing done.

In [10]: %reset -f
In [11]: caixa
-----
NameError                                 Traceback (most recent call last)
<ipython-input-11-f6d2f6859307> in <module>()
      1 caixa

NameError: name 'caixa' is not defined
In [12]: 
In [37]: %save "calc_churrasco.py" 0-10 15 19 26-37
```

Referências principais

[https://www.tutorialspoint.com/
python3/
python_basic_syntax.htm](https://www.tutorialspoint.com/python3/python_basic_syntax.htm)

[https://stackoverflow.com/
search](https://stackoverflow.com/search)

perguntas

