Introdução à programação para ciência e engenharia em *Python*

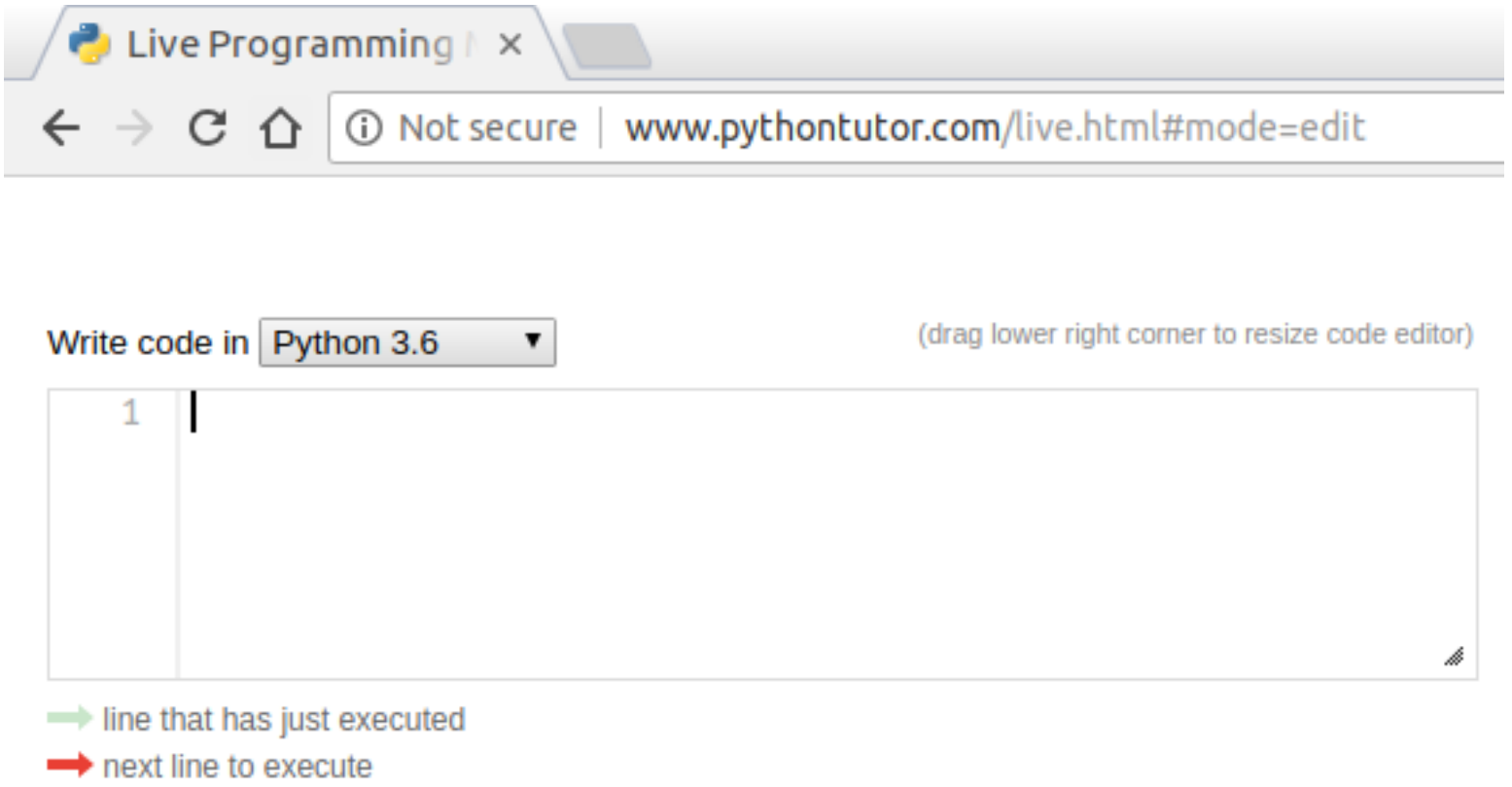
Iuri Soter Viana Segtovich

Parte 2: Lógica e Sintaxe

python tutor

www.pythontutor.com/

live.html#mode=edit



Print *#imprimir*

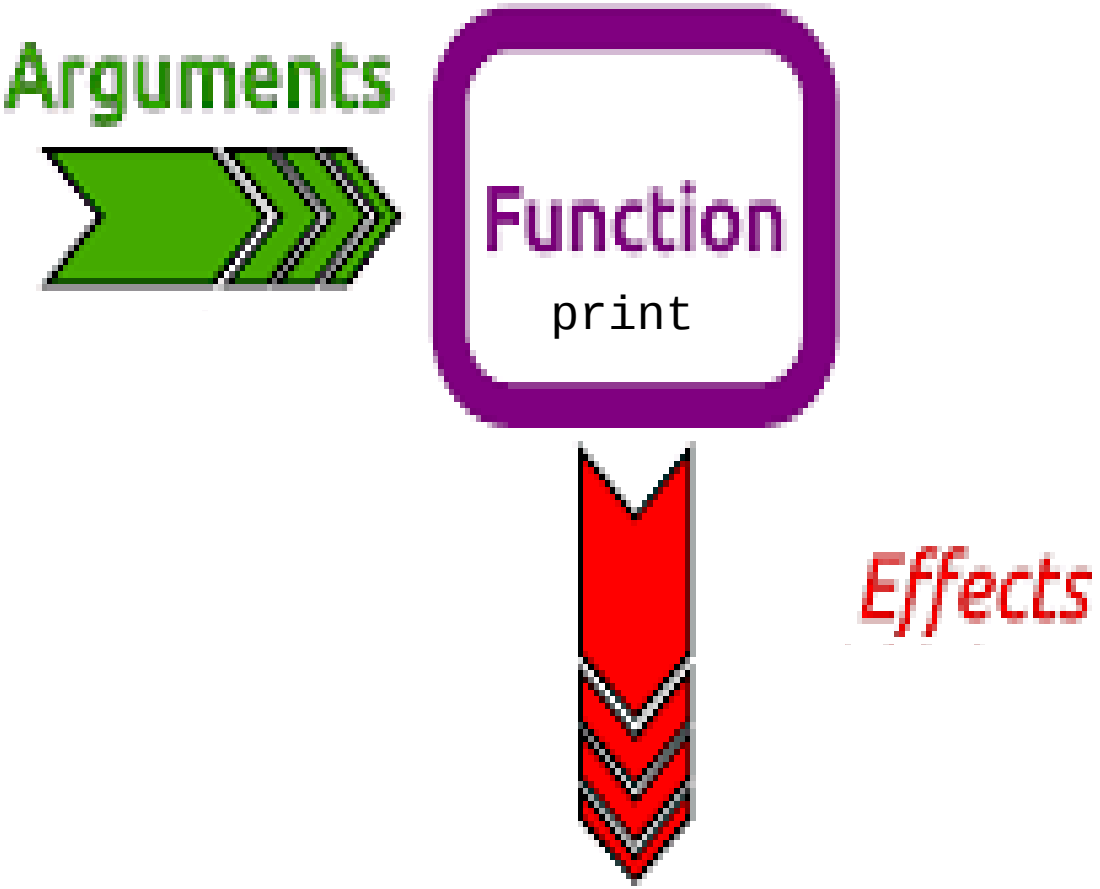
```
print  (  "Olá!"  )
```

```
Olá!
```

1. 2. 3. 4. 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("Oi!")
2
3  print('Tudo bem?')
4
5  print('''Tudo
6  certinho?''')
7
8  print("""Como é
9  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)
```

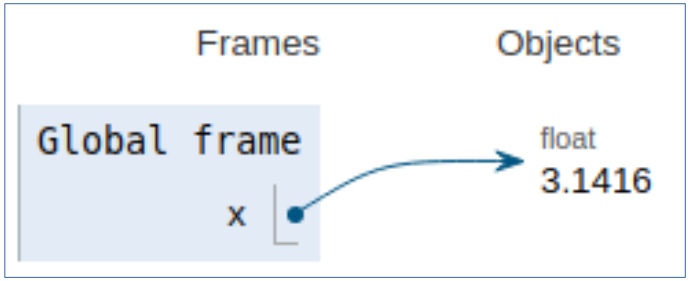
```
1
3.1415
```

O que são objetos e tipos?

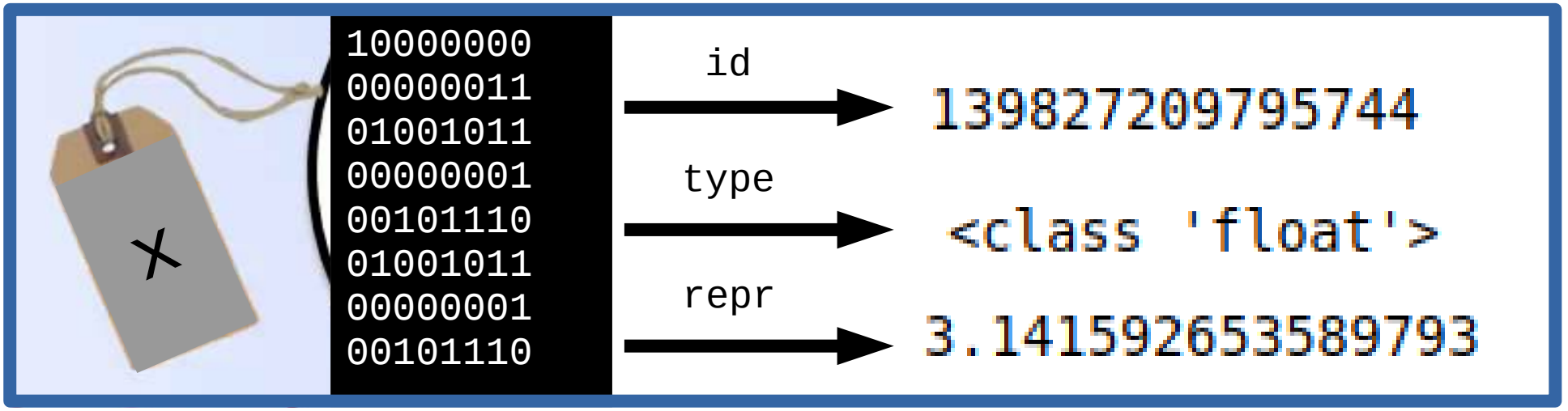
e nomes, identificações e representações?

```
1  x=3.14159265358979323846264433
2  print( "id: ", id(x) )
3  print( "type: ", type(x) )
4  print( "repr: ", repr(x) )
```

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



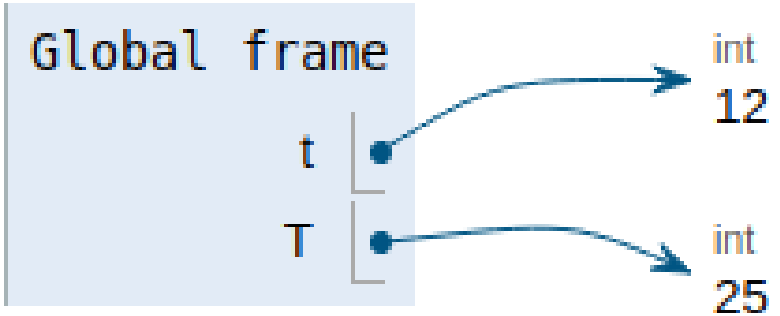
Gerar um objeto do tipo float a partir da expressão literal 3.14159265358979323846264433 e vinculá-lo ao nome x.



Nomes válidos

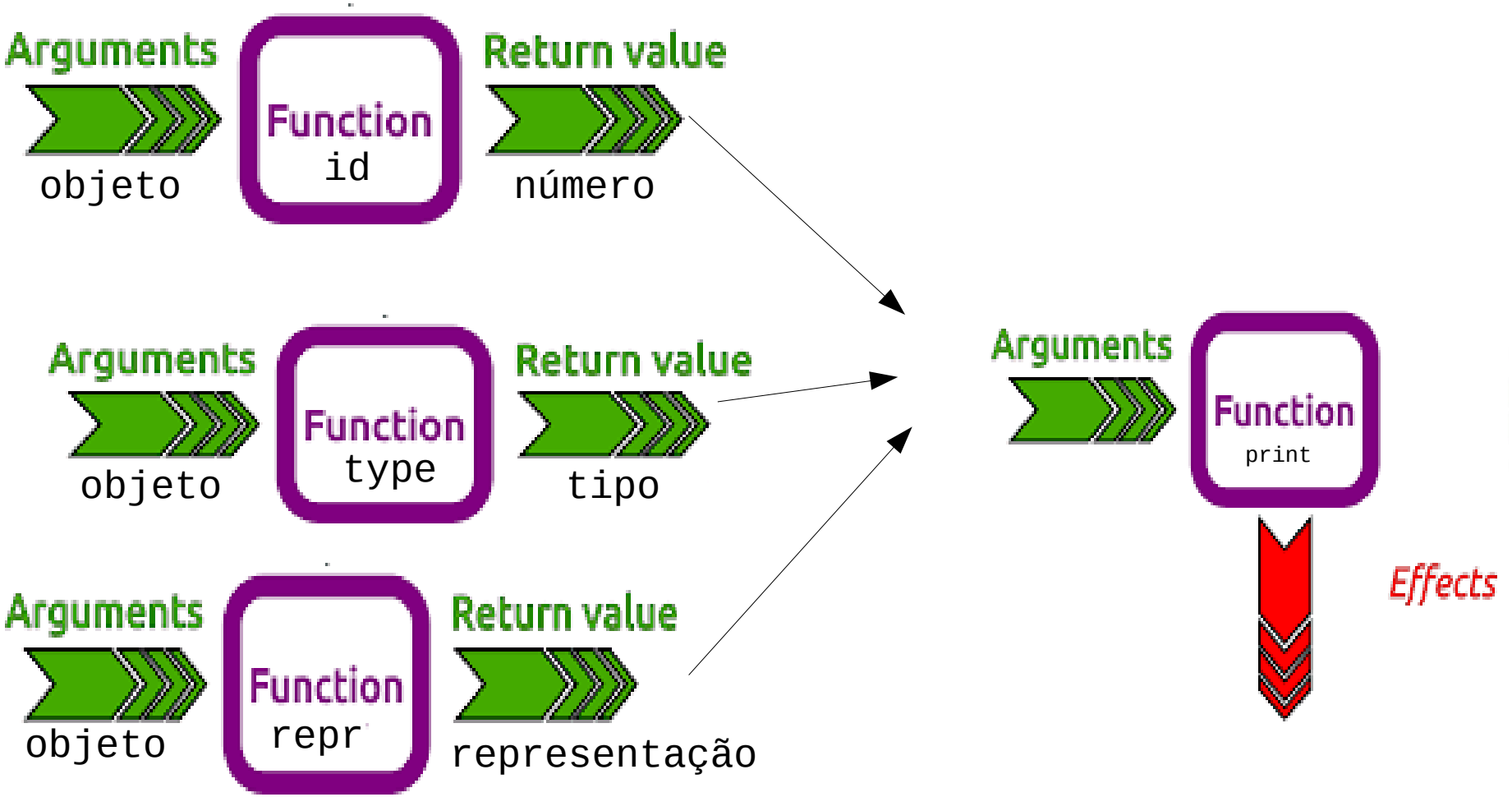
- Letras
- Underline
- Algarismos
  - Não pode começar com algarismos
- `_nomes` ou `__nomes` começando com um ou dois underline são utilizados para funções especiais
- Os nomes são CASE SENSITIVE

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



Mais funções

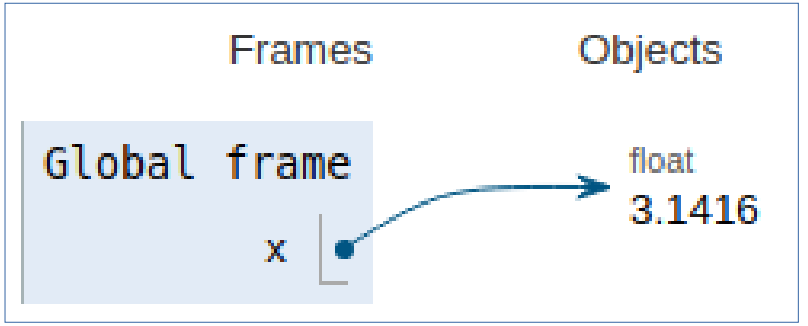
```
x = 3.14159265358979323846264433
print ( id( x ) )
print ( type( x ) )
print ( repr( x ) )
```



comentários

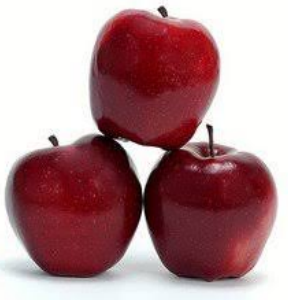
```
1  # vinculando um objeto float com o valor de pi ao nome x
2  x = 3.14159265358979323846264433
3  # imprimindo o número de identificação do objeto vinculado ao nome x
4  print ( id( x ) )
5  # imprimindo o número de identificação do objeto vinculado a x
6  print ( type( x ) )
7  # imprimindo a representação do objeto vinculado a x
8  print ( repr( x ) )
```

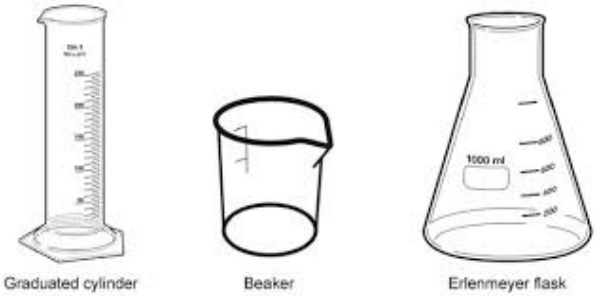
```
140572334176416
<class 'float'>
3.141592653589793
```



Outros tipos

- Numbers
  - int
  - float
  - complex





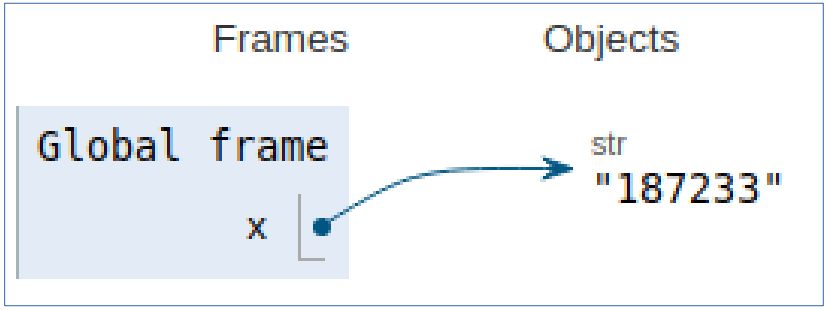
$$i^2 = -1$$

- string

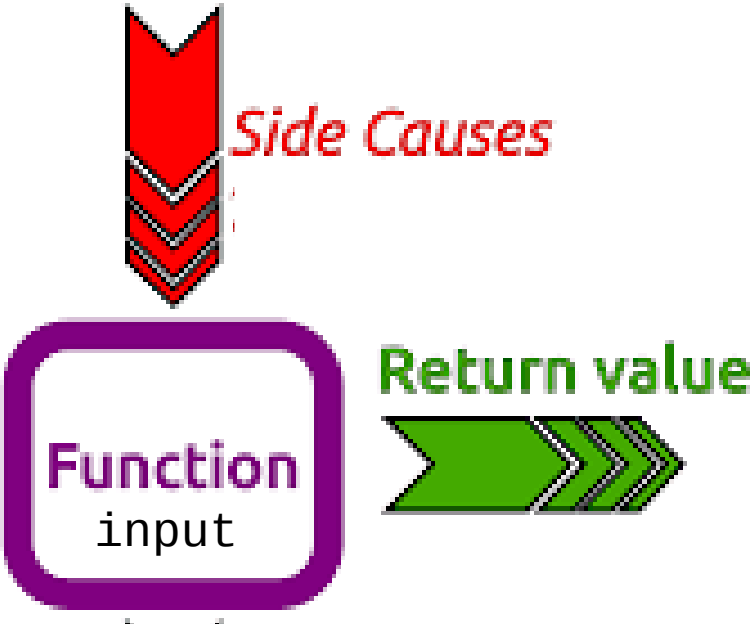
- bool

input

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



187233 Submit

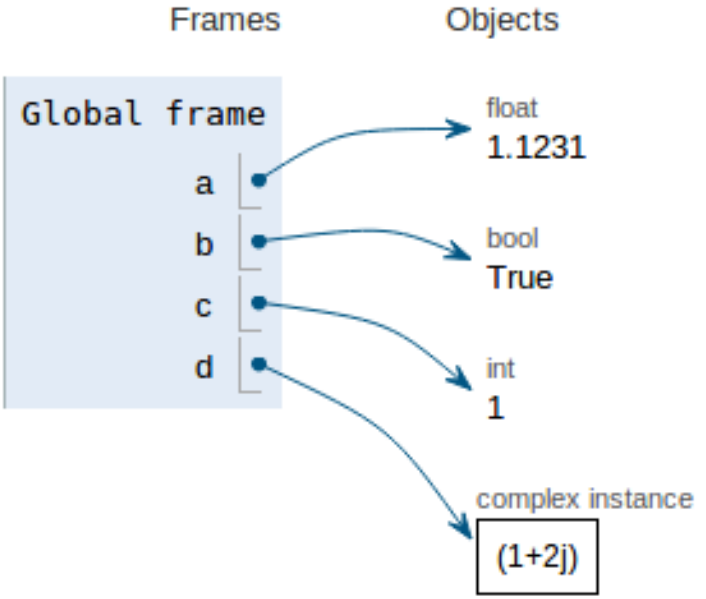


- Sempre interpreta a entrada como texto - retorna um string.
- Parênteses vazios indica chamada sem argumentos.

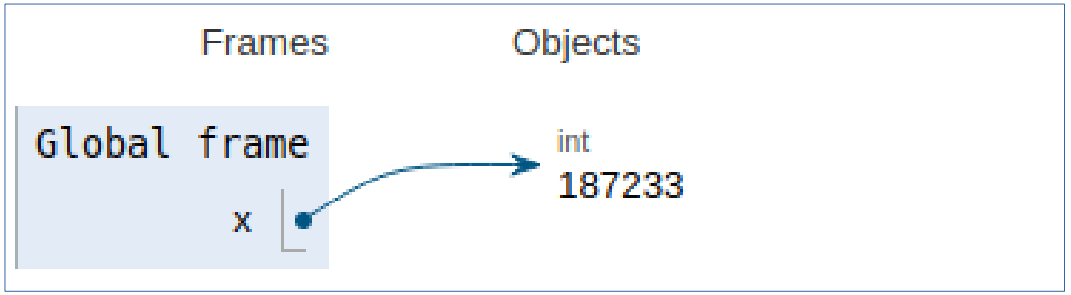
Conversões

- 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")
```

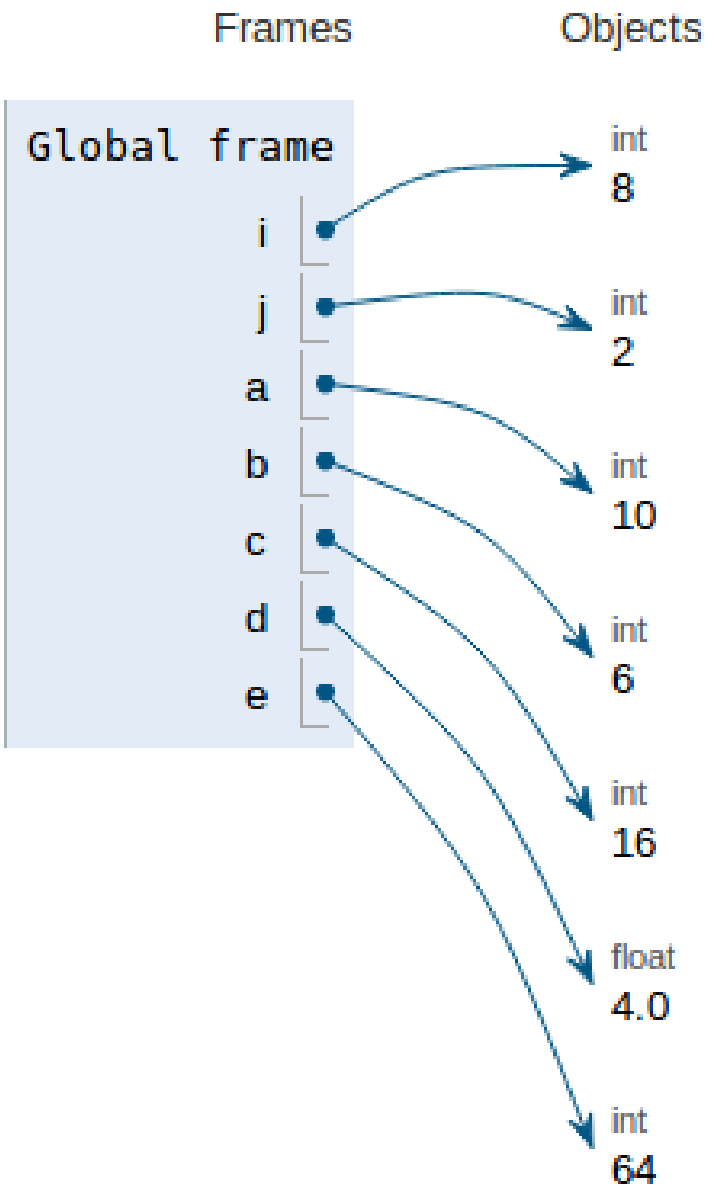



```
1  x=int(input())
```



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
```



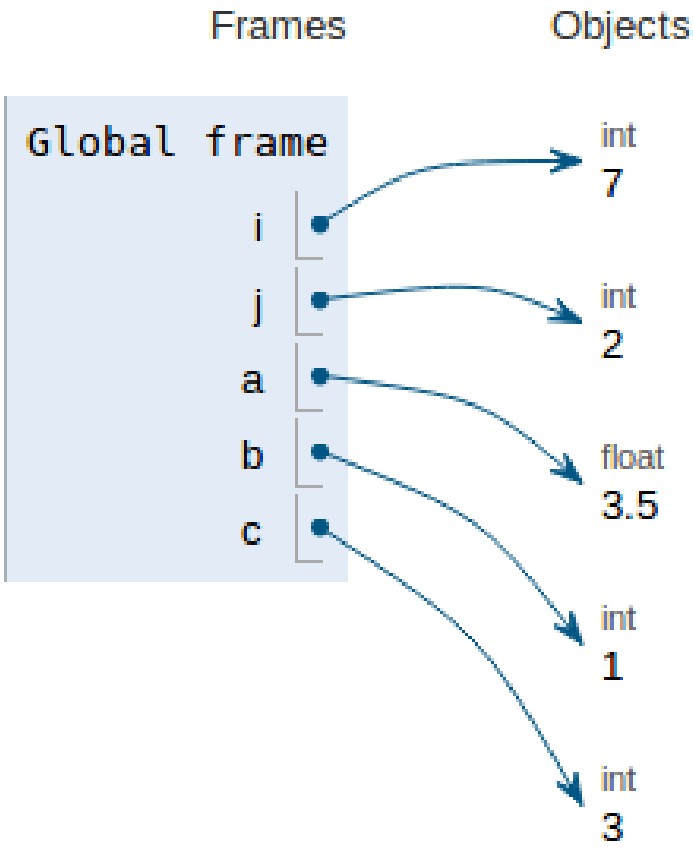
Operadores são funções por trás dos panos

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()`, `**`, `<<`, `>>`, `&`, `^`, `|`).

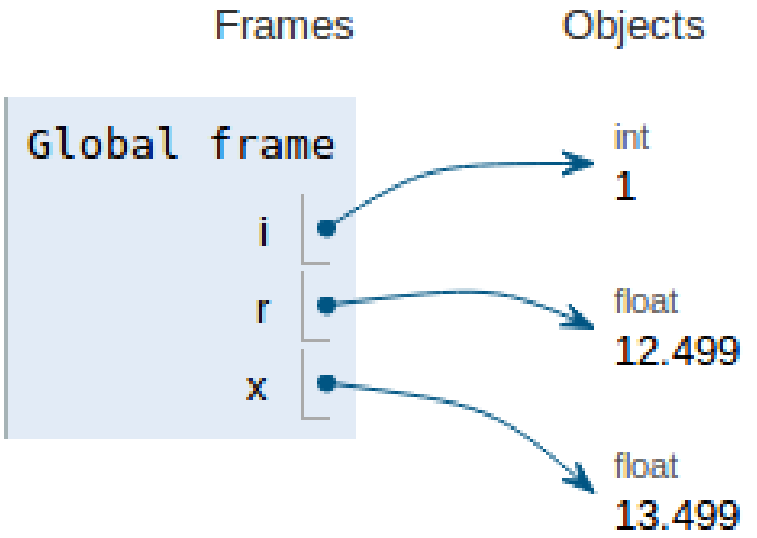
Divisão de inteiros

```
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
```

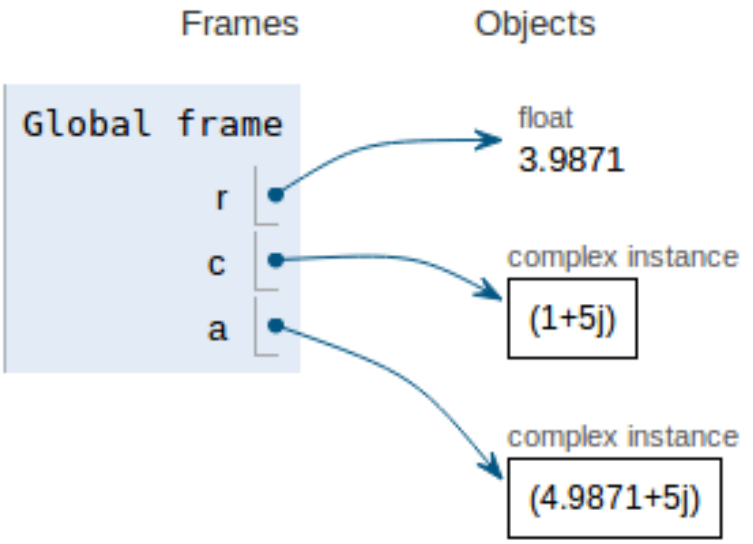


Conversões automáticas

```
1  i=1
2  r=12.49897
3  x=i+r
```



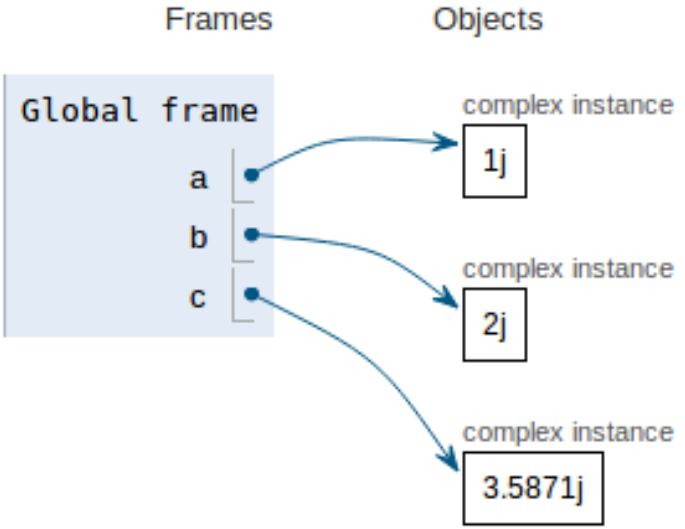

```
1  r=3.9871
2  c=1+5j
3  a=r+c
```



A expressão literal do complexo é 1j

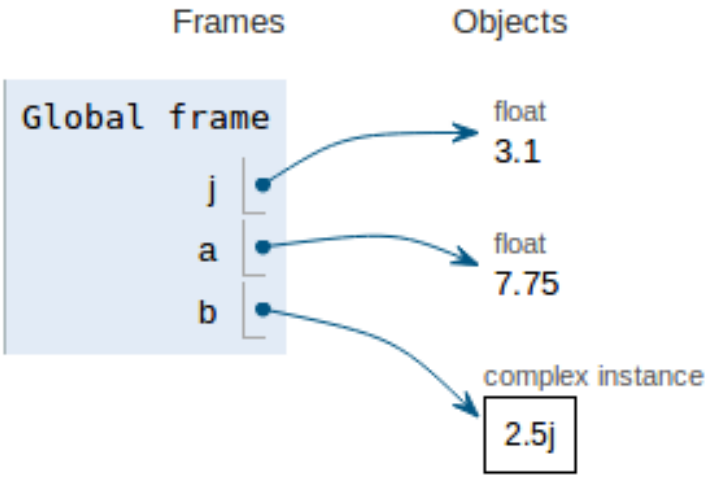
- Também valem seus múltiplos inteiros e decimais

```
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) +, -

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

Frames

Global frame

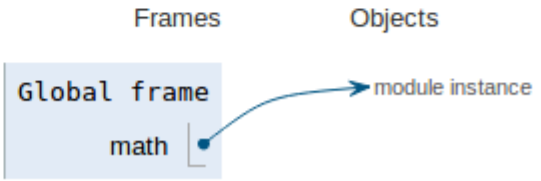
a	1025.875
b	1025.875
c	196608.0

- 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
```



- Comando import
  - Dá acesso aos objetos de um pacote
    - Funções
    - Constantes
- Import um*Modulo* 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
3.14159265358979
```

- 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=[Invalid(
0.14285714285714285714285714286
1.0000000000000000000000000
Context(prec=28, rounding=ROUND_HALF_EVEN, Emin=-999999, Emax=999999, capitals=1, clamp=0, flags=[Inexact, Rounded],
0.14285714285714285714285714285714285714285714285714
0.99999999999999999999999999999999999999999999999998
```

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
```

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

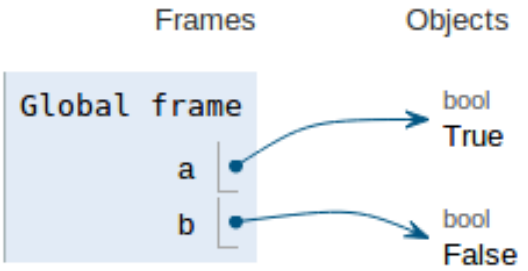
`object.`**`__iadd__`**(*self*, *other*)
`object.`**`__isub__`**(*self*, *other*)
`object.`**`__imul__`**(*self*, *other*)
`object.`**`__imatmul__`**(*self*, *other*)
`object.`**`__itruediv__`**(*self*, *other*)
`object.`**`__ifloordiv__`**(*self*, *other*)
`object.`**`__imod__`**(*self*, *other*)
`object.`**`__ipow__`**(*self*,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

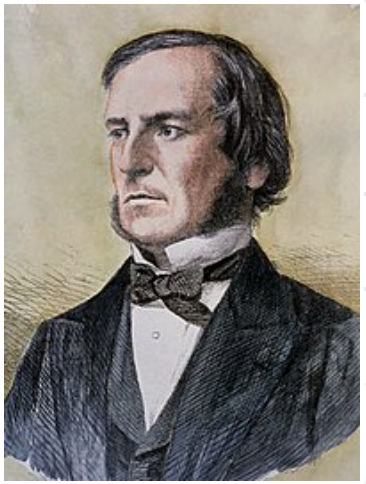
These methods are called to implement the augmented arithmetic assignments (`+=`, `-=`, `*=`, `@=`, `/=`, `//=`, `%=`, `**=`, `<<=`, `>>=`, `&=`, `^=`, `|=`).

Bool - lógicos

```
1  a=True
2  b=False
3  print("a and b:", a and b)
4  print("not a:",not a)
5  print("a or b:",a or b)
```

```
a and b: False
not a: False
a or b: True
```





A	B	A AND B	A OR B	NOT A
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False

Comparações

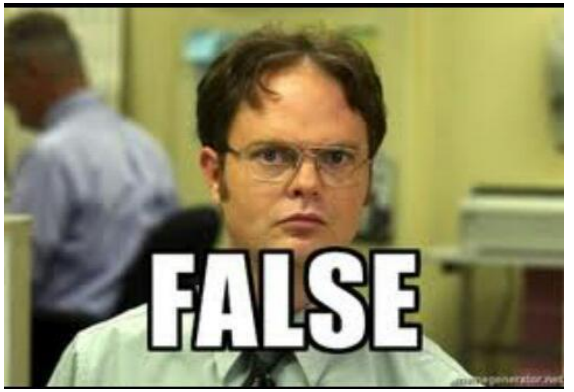
(number,number)→ bool

```
print(1<2)
```

```
True
```

Comparações

(number,number)→ bool




```
print(1>2)
```

```
False
```

Comparações

(number,number)→ bool

```
print(1==2)
print(1<2)
print(1>2)
print(1<=2)
print(1>=2)
print(1!=2)
```

```
False
True
False
True
False
True
```

Comparações

com floats

```
1  x=1./7.
2  print("  1 2 3 4 5 6 7")
3  print("y=x+x+x+x+x+x+x")
4  y=x+x+x+x+x+x+x
5  print(y)
6  print(y==1)
7
8  tol=1e-14
9  print(abs(y-1)<tol)
10
```

```
  1 2 3 4 5 6 7
y=x+x+x+x+x+x+x
0.9999999999999998
False
True
```

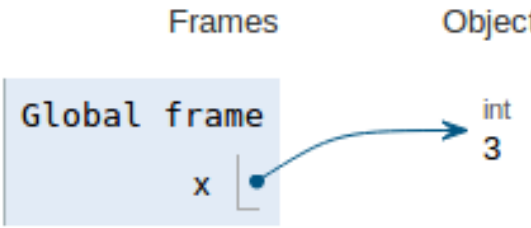
- abs() - função módulo

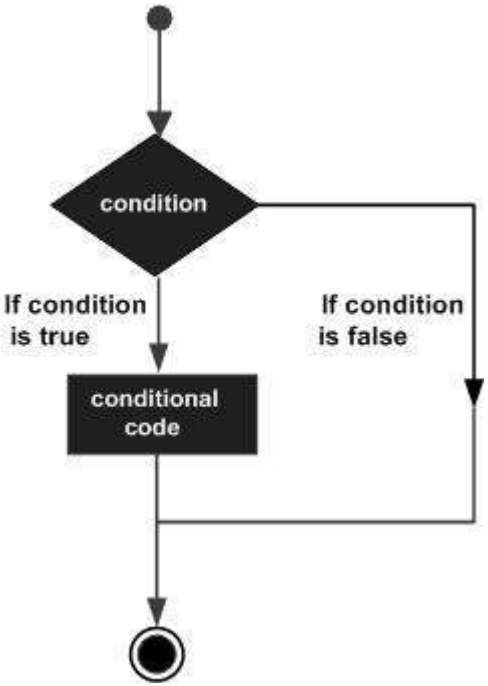
Condicionais

if

```
1  x=3
2  if x>0:
3      print('x positivo')
```

```
x positivo
```



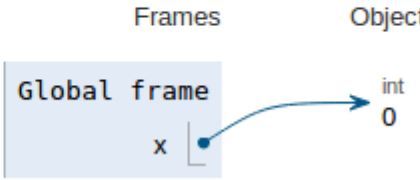


- Dois pontos
- Identação

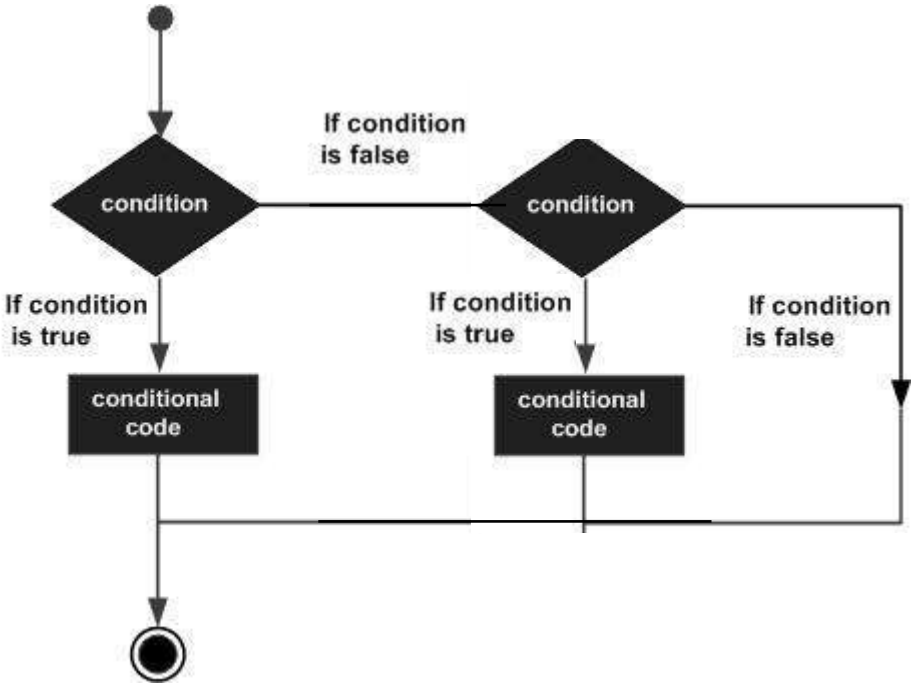
elif

```
1  x=0
2  if x>0:
3      print('x positivo')
4  elif x==0:
5      print('x nulo')
```

```
x nulo
```



- Dois pontos
- Identação



else

```
1  x=-1
2  if x>0:
3      print('positivo')
4  elif x==0:
5      print('nulo')
6  else:
7      print('''se nao é positivo nem zero
8      deve ser negativo''')
```

```
se nao é positivo nem zero
    deve ser negativo
```

Frames Objects

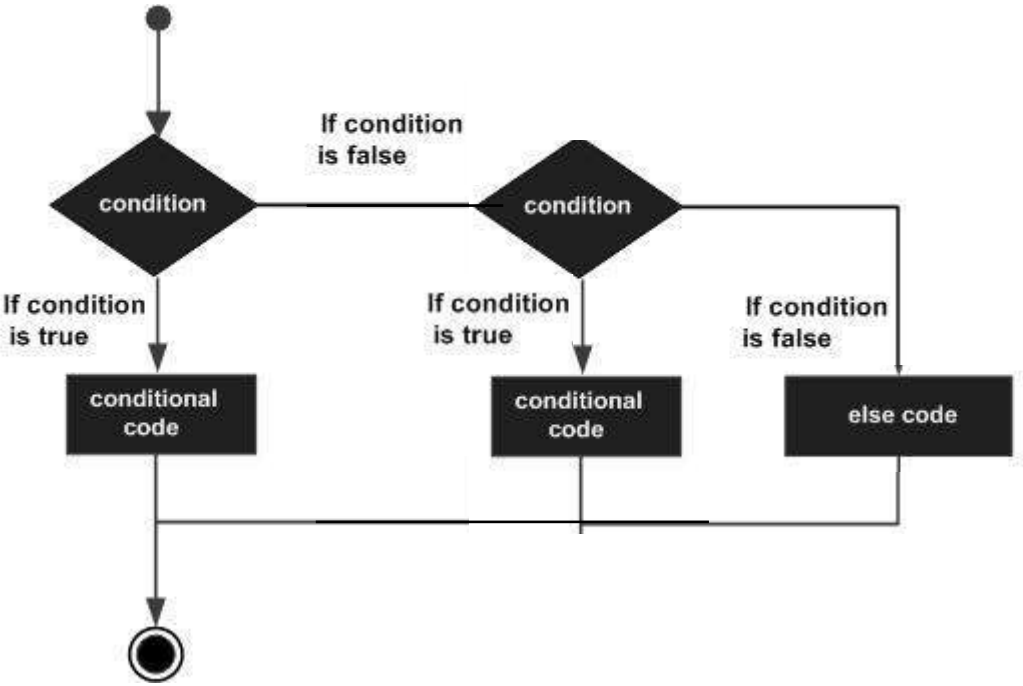
Global frame

x

int

-1

- Dois pontos
- Identação



Exemplo

- Tirou média maior que 7
- Tirou média final maior que 5

Nota P1 = 5
Nota P2 = 8
Verificar se média maior que 7
Verificar se nota menor que 3
Imprimir média e informação de aprovado ou em prova final ou reprovado direto

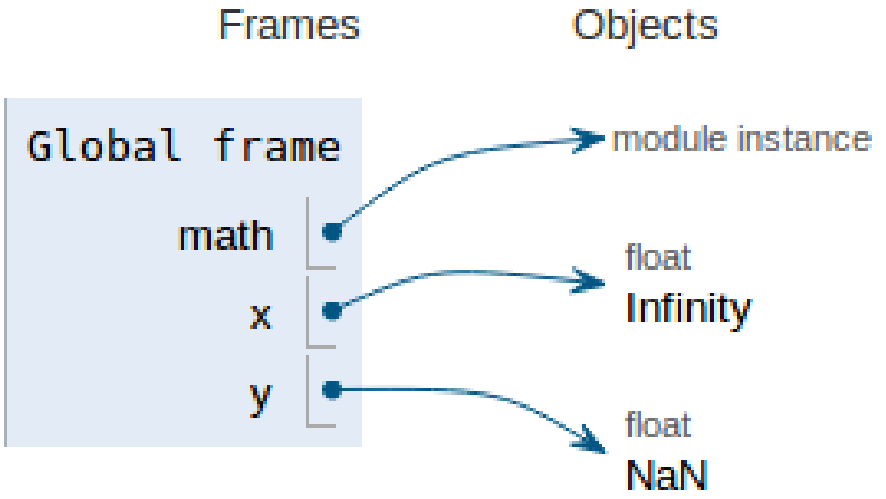
Apenas se foi pra PF:
Nota PF = 7
Verificar se nota final maior que 5
Imprimir média final e informação de aprovado ou reprovado

Comparação com NAN

- inf - infinity
- nan – not a number

```
1  import math
2  x=1.02e23
3  print(math.log10(x))
4  x=x*x
5  print(math.log10(x))
6  x=x*x
7  print(math.log10(x))
8  x=x*x
9  print(math.log10(x))
10 x=x*x
11 print(math.log10(x))
12 print(x)
13 y=x-x
14 print(x)
15
```

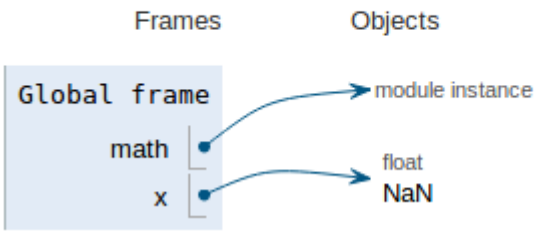
```
23.008600171761916
46.017200343523832
92.034400687047664
184.06880137409533
inf
inf
nan
```



Exemplo

```
1  import math
2  x=float("nan")
3  if (x>0):
4      print(x,": positivo")
5  elif x==0:
6      print(x,": nulo")
7  elif x<0:
8      print(x,": negativo")
9  elif math.isnan(x):
10     print(x,": erro tipo not a number")
11 else:
12     print(x,": exceção não prevista!")
13
```

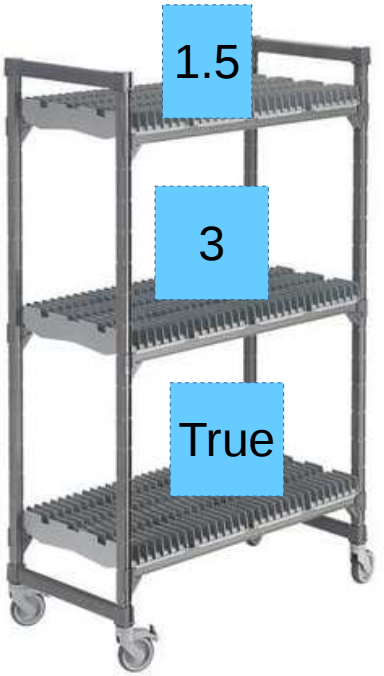
```
nan : erro tipo not a number
```

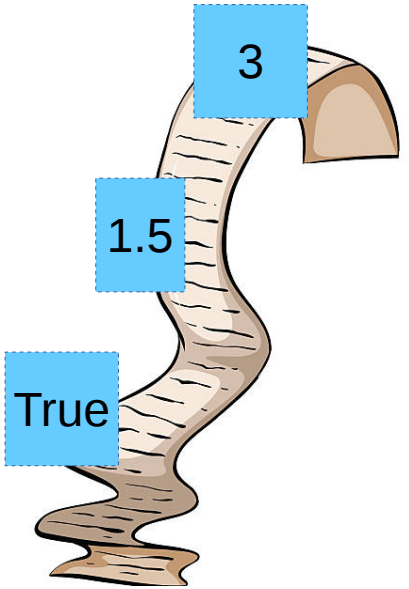


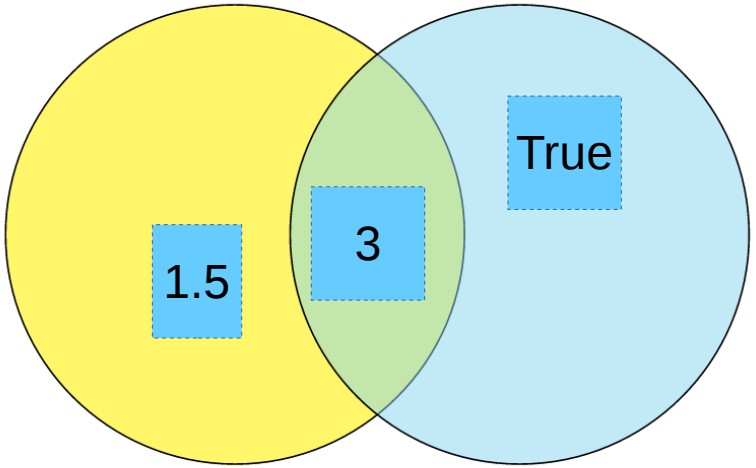
mais tipos

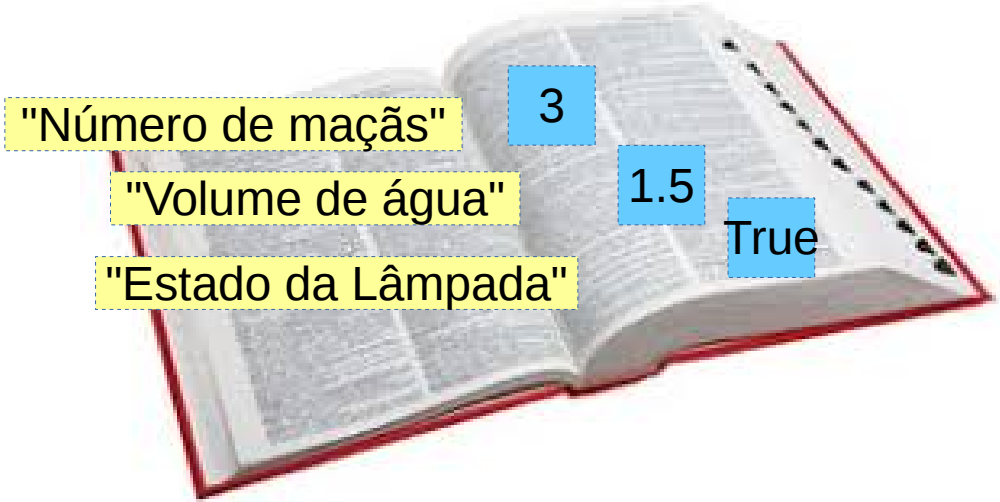
Containers / Collections

- tuple
- list
- set
- dictionary



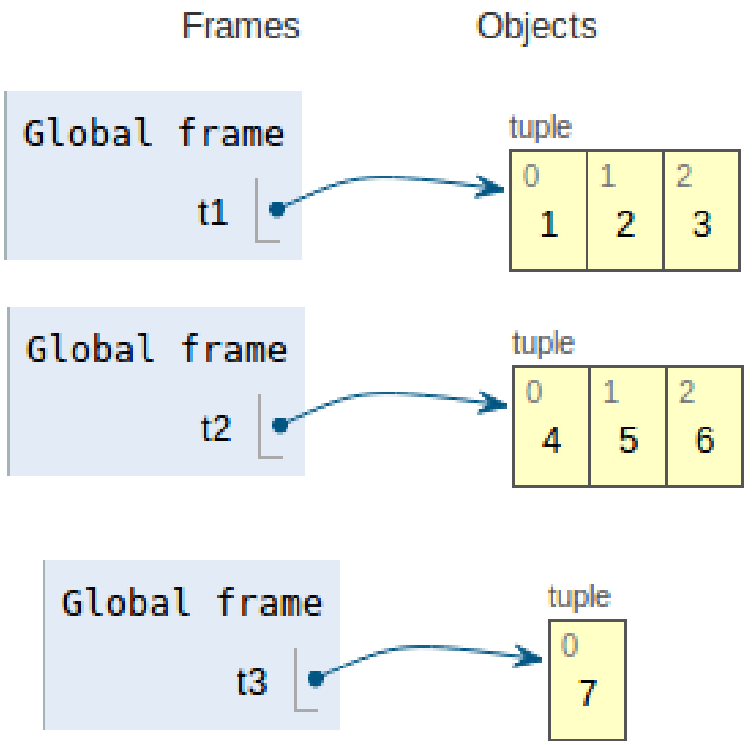






tuple

```
1  t1 = ( 1, 2, 3 )
2  t2 = 4, 5, 6
3  t3 = (7,)
```

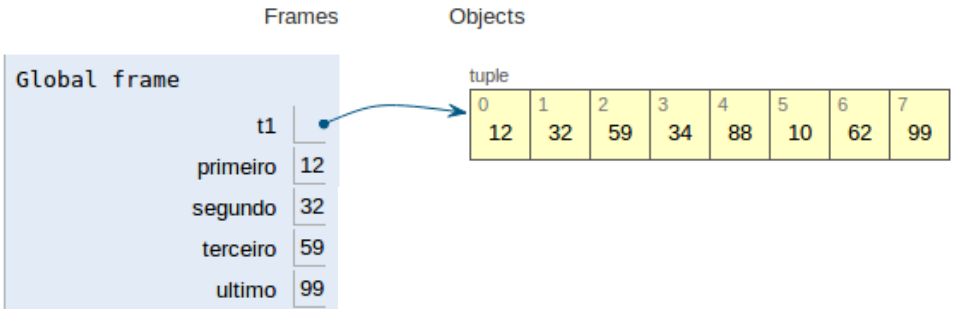


- Expressão literal
  - vírgulas e parênteses, opcionalmente

Indexing

acessar elementos por índice

```
1  t1 = (12, 32, 59, 34, 88, 10, 62, 99)
2  primeiro = t1[0]
3  segundo = t1[1]
4  terceiro = t1[2]
5
6  ultimo = t1[-1]
7
```



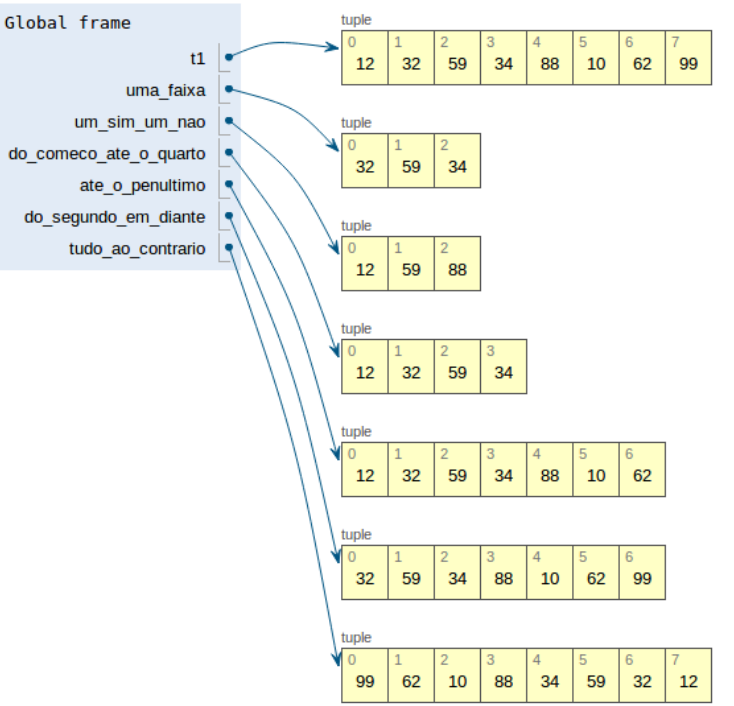
- Começa no zero
- -1 significa último

```
Index from rear:    -6  -5  -4  -3  -2  -1
Index from front:    0   1   2   3   4   5
                   +---+---+---+---+---+---+
                   | a | b | c | d | e | f |
                   +---+---+---+---+---+---+
```

Slicing

fatiar, cria cópias de seções

```
1  t1 = (12, 32, 59, 34, 88, 10, 62, 99)
2
3  uma_faixa = t1[1:4]
4  #começa no 1 e pára antes do 4
5
6  um_sim_um_nao = t1[0:5:2]
7  #começa no zero e pára antes do 5, com passo de 2
8
9  do_comeco_ate_o_quarto = t1[:4]
10 #primeiro argumento vazio, começa do começo
11
12 ate_o_penultimo = t1[:-1]
13 #pára antes do último
14
15 do_segundo_em_diante = t1[1:]
16 #segundo argumento vazio, vai até o final
17
18 tudo_ao_contrario = t1[::-1]
19 #do final ao começo com passo de -1
```

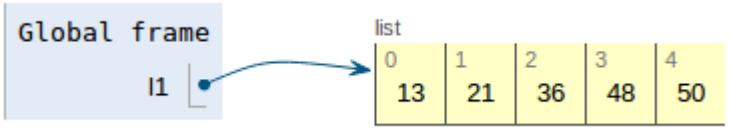


- Argumento vazio segue um comportamento padrão

```
Index from rear:    -6  -5  -4  -3  -2  -1
Index from front:    0   1   2   3   4   5
                   +---+---+---+---+---+---+
                   | a | b | c | d | e | f |
                   +---+---+---+---+---+---+
Slice from front:  :   1   2   3   4   5   :
Slice from rear:   :  -5  -4  -3  -2  -1   :
```

list

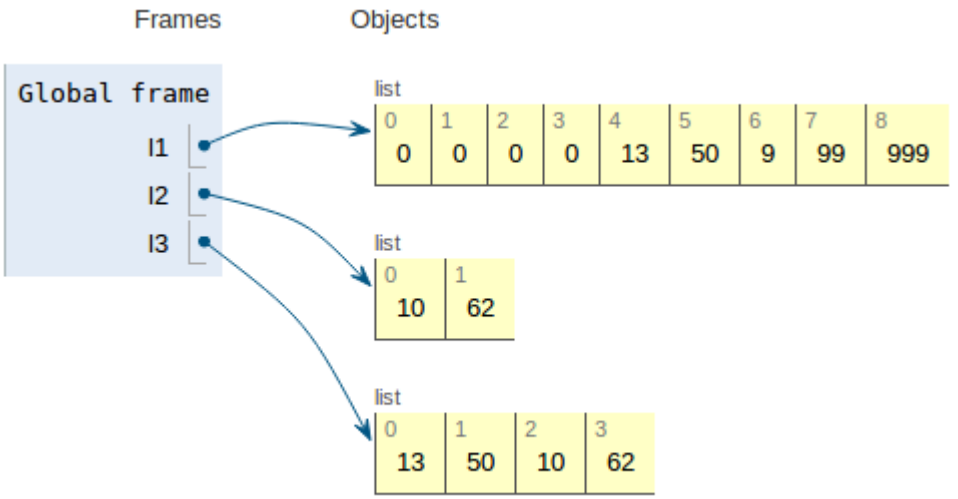
```
1  l1 = [13,21,36,48,50]
```



- Expressão literal – vírgulas e colchetes

list

```
1  l1 = [13,50]
2  l2 = [10,62]
3
4  l3 = l1+l2 #concatenation
5
6  l1+=[9] #extend
7  l1.extend([99]) #extend
8  l1*=2 #extend
9
10 l1.append(999) #append
11
12 l1[0]=0 #rebind item
13 l1[1:4]=3*[0] #rebind items
```



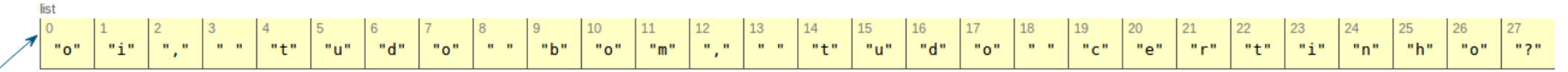
- Append (apensar)
- Extend (extender)
- Rebind item (re vincular elemento)

Slicing Strings

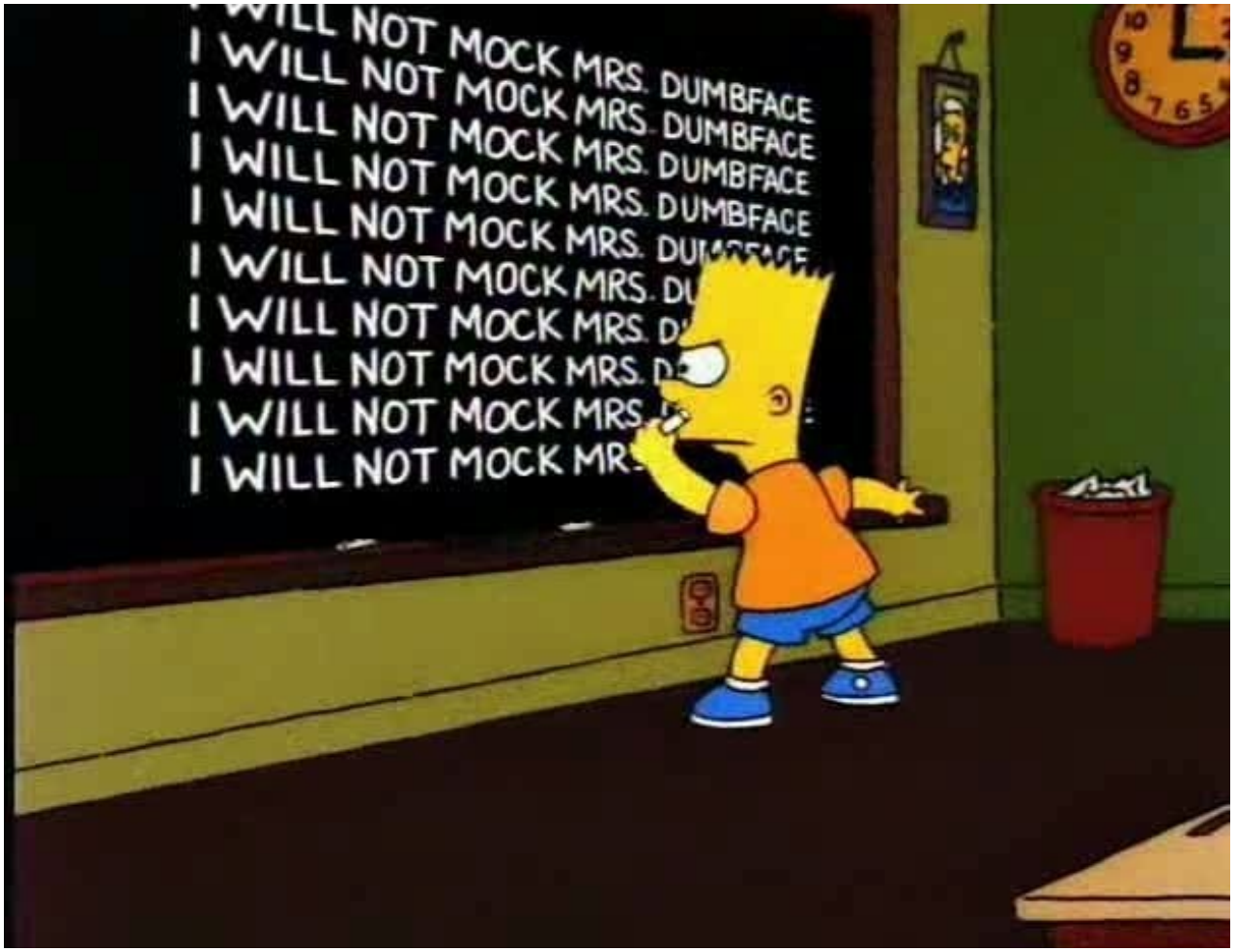
```
1  mensagem="oi, tudo bom, tudo certinho?"
2  comeco=mensagem[:4]
3  meio=mensagem[4:14]
4  fim=mensagem[14:]
5
6  caracteres=list(mensagem)
```

Global frame

mensagem	"oi, tudo bom, tudo certinho?"
comeco	"oi, "
meio	"tudo bom, "
fim	"tudo certinho?"
caracteres	



Iterações


I WILL NOT MOCK MRS. DUMBFACE
I WILL NOT MOCK MRS. DUMBFACE
I WILL NOT MOCK MRS. DUMBFACE
I WILL NOT MOCK MRS. DUMBFACE

for

```
1  n=8
2
3  for i in range(n):
4      print("I WILL NOT MOCK MRS. DUMBFACE")
```

```
I WILL NOT MOCK MRS. DUMBFACE
I WILL NOT MOCK MRS. DUMBFACE
I WILL NOT MOCK MRS. DUMBFACE
I WILL NOT MOCK MRS. DUMBFACE
I WILL NOT MOCK MRS. DUMBFACE
I WILL NOT MOCK MRS. DUMBFACE
I WILL NOT MOCK MRS. DUMBFACE
I WILL NOT MOCK MRS. DUMBFACE
```

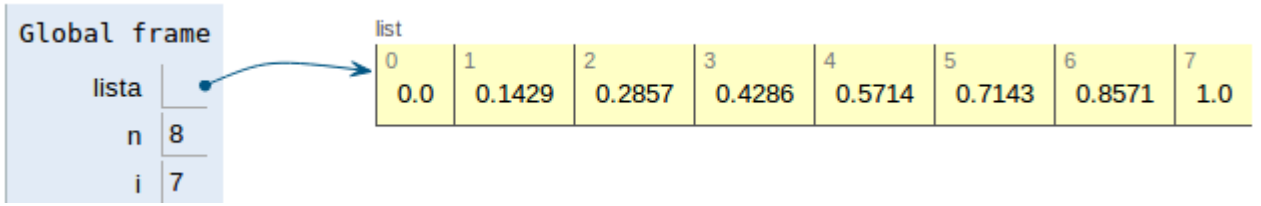
- Dois pontos
- Identação

- Função range

Operações com o contador

```
1  lista=[] #vazia
2  n=8
3
4  for i in range(n):
5      print("i: ", i)
6      lista.append(i/7)
```

```
i:  0
i:  1
i:  2
i:  3
i:  4
i:  5
i:  6
i:  7
```



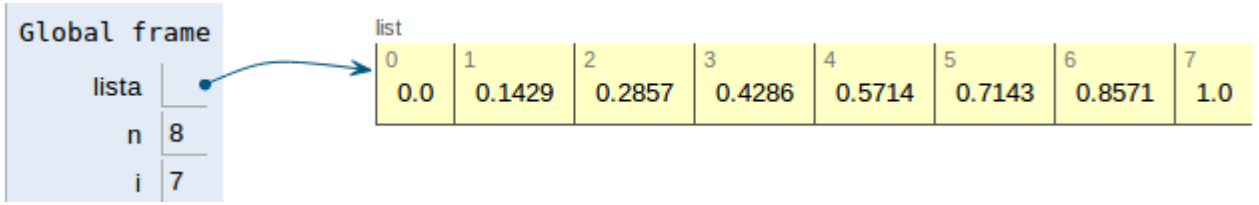
somatório

FAZER O SOMATÓRIO DA LISTA
[2,4,6,8,10,12]

Nested for

```
1  k=1
2  for i in range(3):
3      for j in range(4):
4          print(i,j,k)
5          k+=1
```

Iteraçao ao longo dos elementos de uma lista




```
8  for i in range(n):
9      print("item: ",lista[i])
```

```
item:  0.0
item:  0.14285714285714285
item:  0.2857142857142857
item:  0.42857142857142855
item:  0.5714285714285714
item:  0.7142857142857143
item:  0.8571428571428571
item:  1.0
```

```
11 for item in lista:
12     print("item: ",item)
13
```

```
item:  0.0
item:  0.14285714285714285
item:  0.2857142857142857
item:  0.42857142857142855
item:  0.5714285714285714
item:  0.7142857142857143
item:  0.8571428571428571
item:  1.0
```

- In - iteração sobre Cada elemento de uma lista

while

```
1  n=8
2  for i in range(n):
3      print("for:",i)
4
5  n=8
6  i=0
7  while i<n:
8      print("while:",i)
9      i+=1
```

```
for: 0
for: 1
for: 2
for: 3
for: 4
for: 5
for: 6
for: 7
while: 0
while: 1
while: 2
while: 3
while: 4
while: 5
while: 6
while: 7
```

- While (predicate)
- Loop infinito

Exemplo: achar maximo

Achar o maximo em [1,346432,68,1223,5,47,678]

Exemplo: Fatorial

$0 ! = 1$

$n ! = n * (n-1)!$ para $n > 0$

Exemplo: Ordenar (sort)

Colocar a lista [1,346432,68,1223,5,47,678] em ordem crescente

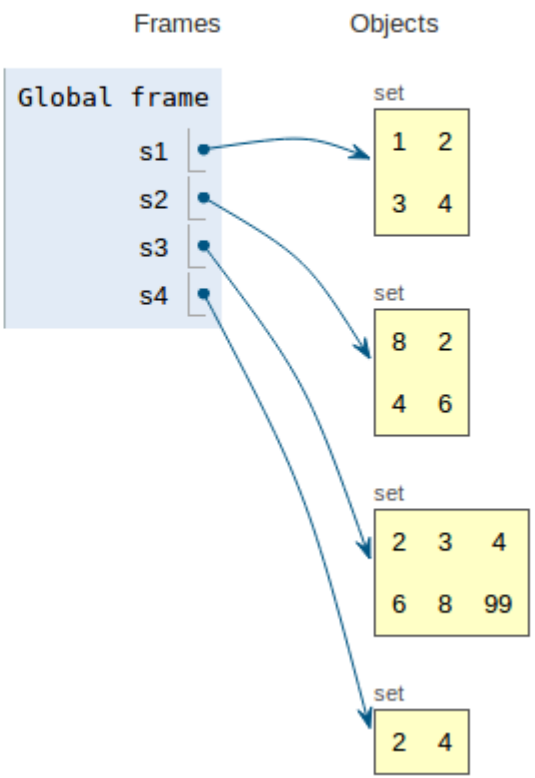
(1 < 5 < 47 < 68 < 678 < 1223 < 346432)

set

```
1  s1={1,2,3,4}
2  s2={2,4,6,8}
3
4  print(1 in s1)
5
6  s3=s1|s2 #union
7  s4=s1&s2 #intersection
8
9  s3|={99} #insertion
10 s3-={1} #removal
11
```

Print output (drag lower right corner to resize)

```
True
```



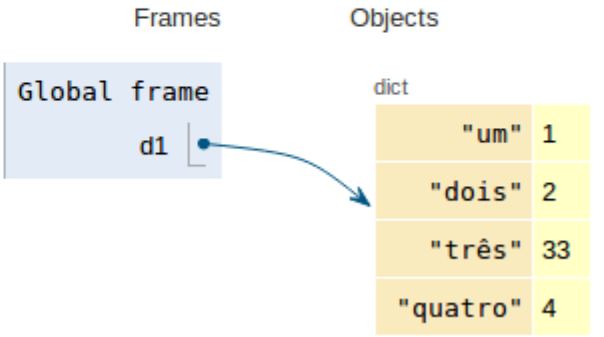
- União
- Interseção
- Adição / Remoção

dict

```
1  d1={"um":1,"dois":2,"três":3}
2  print(d1.keys())
3  print(d1.values())
4
5  d1["três"]=33 #modificação
6  d1["quatro"]=4 #nova entrada
7
```

Print output (drag lower right corner to resize)

```
dict_keys(['um', 'dois', 'três'])
dict_values([1, 2, 3])
```



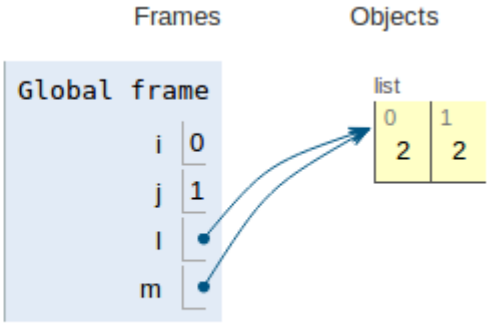
- Indexação por palavra-chave

Mutability

- IMMUTABLE
  - Numbers
  - String
  - Bool
  - Tuple
- MUTABLE
  - List
  - Set
  - Dictionary

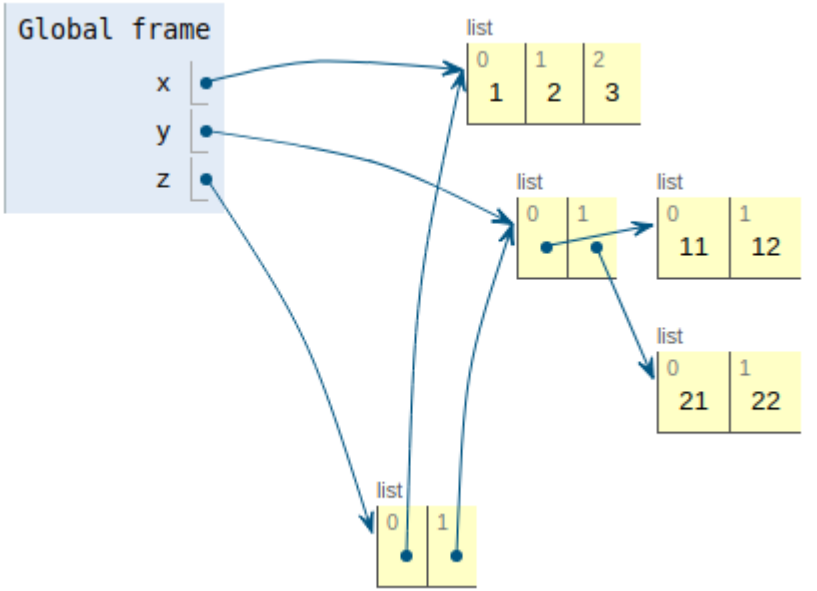
Mutability

```
1  i=0
2  j=i
3  j+=1 #j agora   dá nome a um novo objeto
4
5  l=[1,2]
6  m=l
7  m[0]+=1 #modificamos o objeto lista ao qual l e m dão nome
```



listas de listas

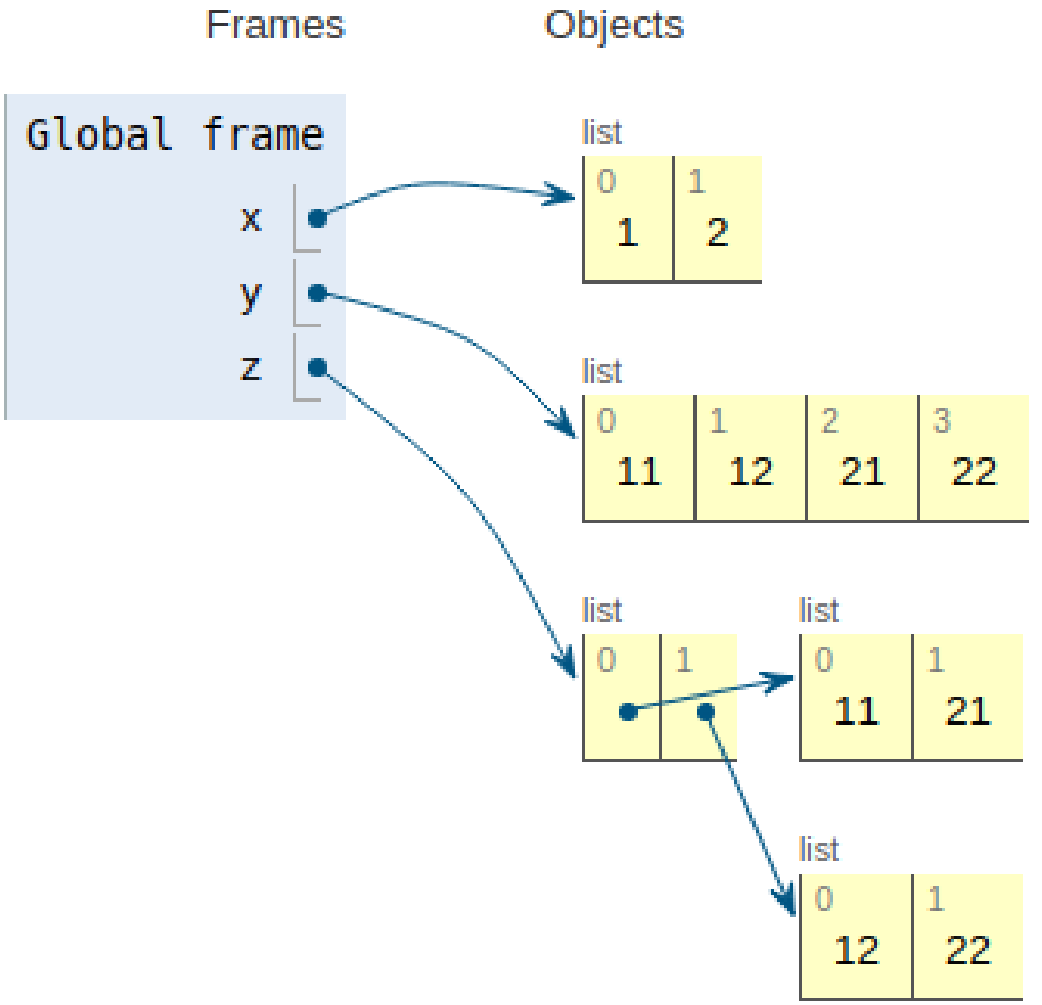
```
1  x=[1,2,3]
2
3  y=[[11,12],[21,22]]
4
5  z=[x,y]
```




```
7  print('x0',x[0])
8  print('y0',y[0])
9  print('y00',y[0][0])
10 print('z0',z[0])
11 print('z00',z[0][0])
12 print('z1',z[1])
13 print('z10',z[1][0])
14 print('z100',z[1][0][0])
```

```
x0 1
y0 [11, 12]
y00 11
z0 [1, 2, 3]
z00 1
z1 [[11, 12], [21, 22]]
z10 [11, 12]
z100 11
```

Comprehension



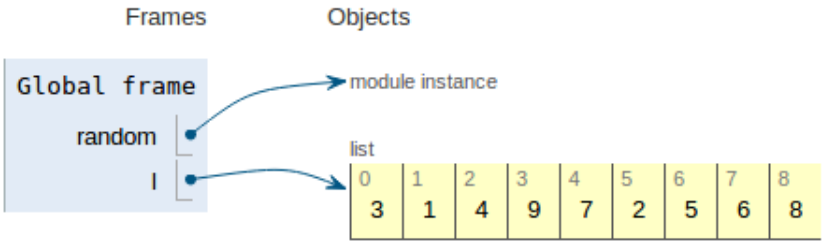

```
x=[i for i in range(1,3)]

y=[10*i+j for i in range(1,3) for j in range(1,3)]

z=[[10*i+j for i in range(1,3)] for j in range(1,3)]
```

module random

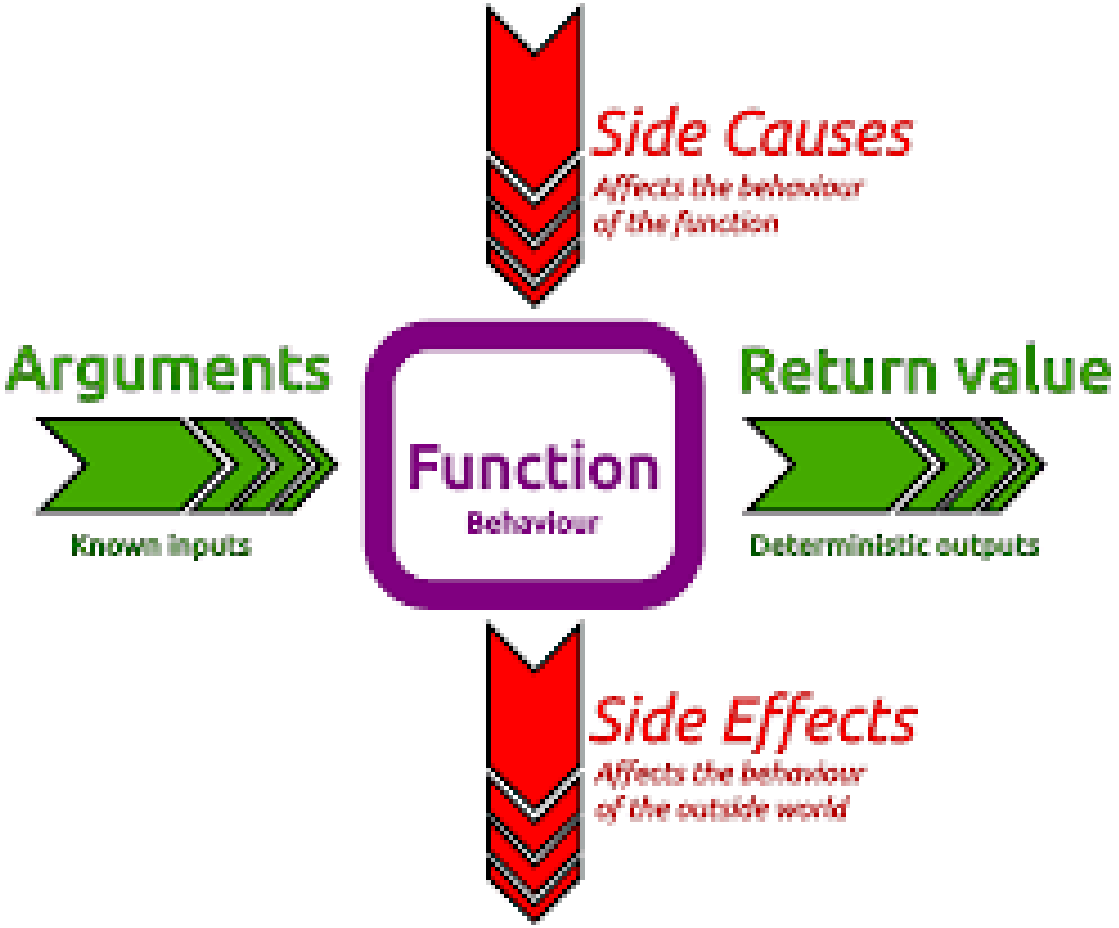
```
1  import random
2  l=[1,2,3,4,5,6,7,8,9]
3  print(random.choice(l))
4  print(random.choice(l))
5  print(random.choice(l))
6  print(random.choice(l))
7
8  print(random.random())
9  print(random.random())
10 print(random.random())
11 print(random.random())
12
13 random.shuffle(l)
```

Print output (drag lower right corner to resize)

```
5
0.9654648863619172
0.48592769656281265
0.9182343317851318
0.8298529036589914
```



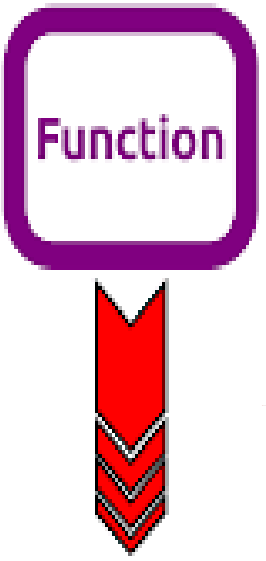
O caso geral das funções



Def funções

```
1  def diga_oi():
2      print("oi")
3
4  diga_oi()
```

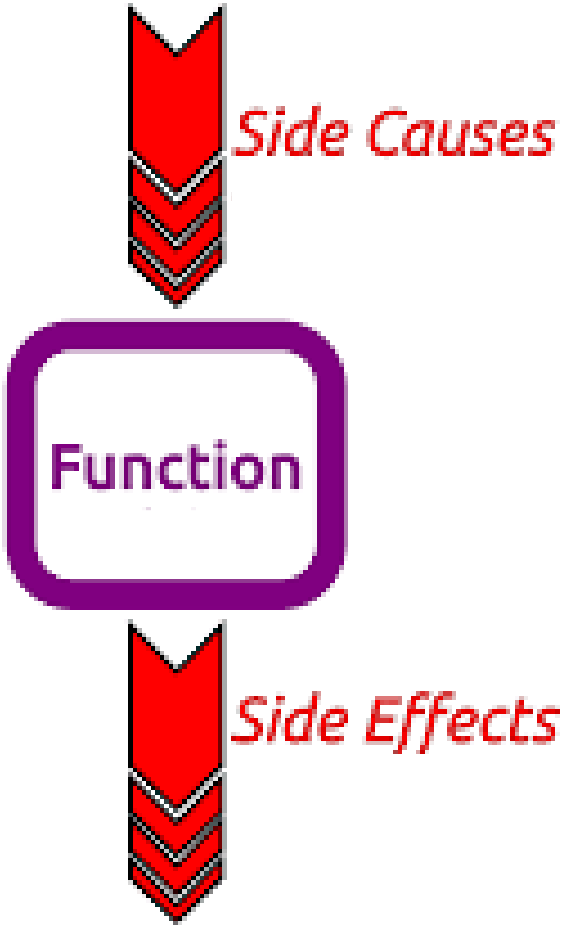
```
oi
```



- Def, Parênteses vazios - definição
- Dois pontos
- Identação
- Parênteses vazios - chamada

Side causes

```
1  x=1
2
3  def leia_x():
4      print("x é", x)
5
6  leia_x()
```

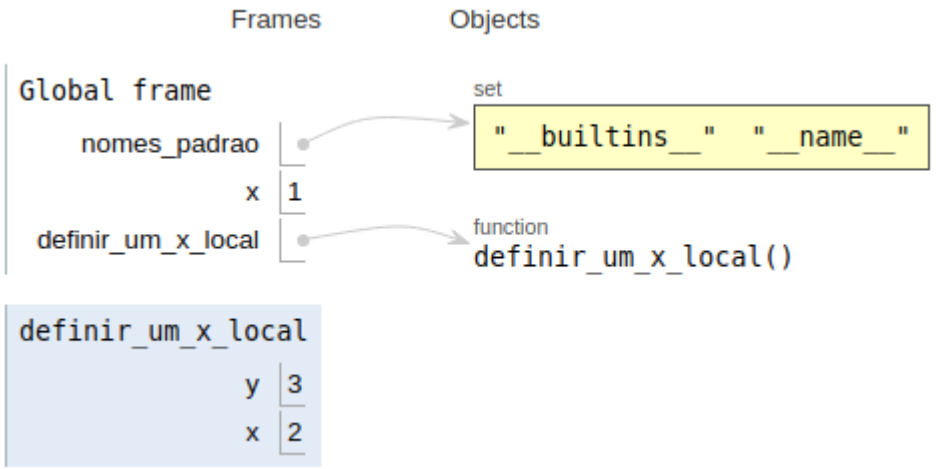


- Acesso de leitura às globais

Namespaces - função dir()

```
1  nomes_padrao=set(dir())
2  x=1
3  print("nomes no namepace externo", set(dir())-nomes_padrao)
4  def definir_um_x_local():
5      y=3 #definindo um nome y, localmente
6      x=2 #definindo um nome x, localmente
7      print("x do namespace interno é", x)
8      print("nomes no namepace interno",set(dir())-nomes_padrao)
9
10 print("x do namespace externo é", x)
11 definir_um_x_local()
12 print("x do namespace externo ainda é", x)
13 print("nomes no namepace externo",set(dir())-nomes_padrao)
```

```
nomes no namepace externo {'nomes_padrao', 'x'}
x do namespace externo é 1
x do namespace interno é 2
nomes no namepace interno {'x', 'y'}
x do namespace externo ainda é 1
nomes no namepace externo {'nomes_padrao', 'x', 'definir_um_x_local'}
```



global

```
x do namespace externo é 1
x do namespace externo é 1
x é 1
[]
x do namespace externo é 1
```

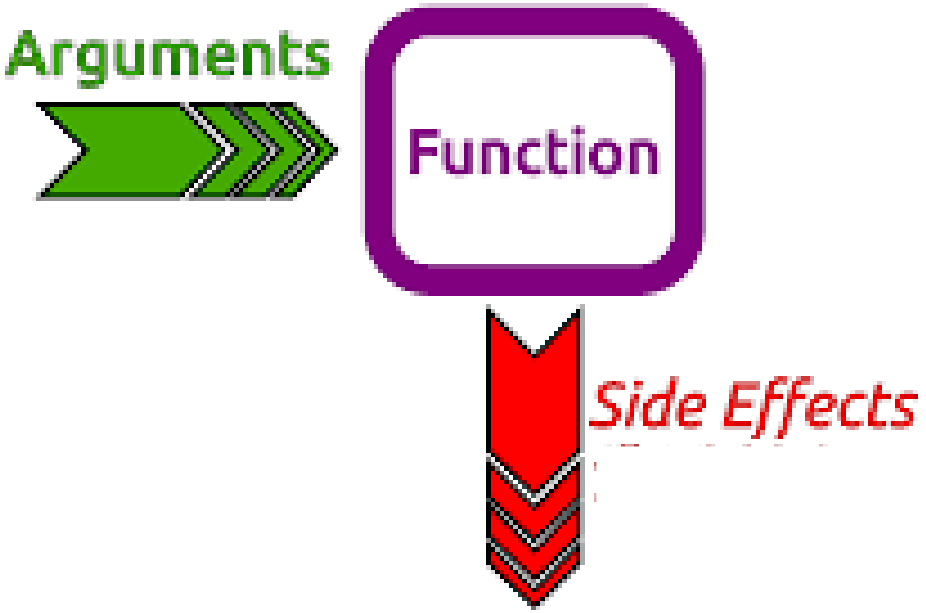
```
1  x=1
2
3  def mude_x():
4      global x
5      print("x do namespace externo é", x)
6      print("x é", x)
7      print(dir())
8
9  print("x do namespace externo é", x)
10 mude_x()
11 print("x do namespace externo é", x)
```

- Acesso de modificação
- (Rebinding) das globais

argumentos

```
1  def converte_m_para_cm(x_in):
2      x_out = 100.*x_in
3      print(x_out)
4
5
6  converte_m_para_cm(5.)
```

```
500.0
```



- argumentos posicionais

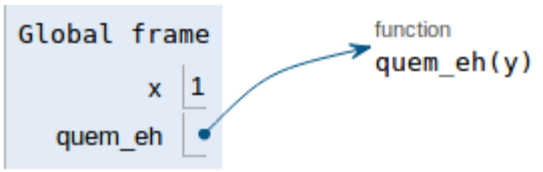
nome *dummy* x nome *actual*

```
1  x_actual=1
2
3  def f(x_dummy):
4      print(x_dummy)
5
6  f(x_actual)
7
8  #o nome x_dummy passa a referenciar
9  #aquele objeto ao qual o nome x_actual dava nome
10 #enquanto estivermos dentro da função
11
12 f(x_dummy=x_actual) #chamada com palavra_chave
```

```
1
1
```

Passa referência para um objeto único, não passa o nome

```
1  x=1
2  print(id(x))
3
4  def quem_eh(y):
5      print(id(y)) #outro nome, mesmo objeto
6
7  quem_eh(x)
8
```




```
1  x=1
2  print(id(x))
3
4  def quem_eh(y):
5      print(id(x))
6      print(id(y)) #outro nome, mesmo objeto
7      y+=1
8      print(id(x)) # x continua dando nome ao objeto 1
9      print(id(y)) # y dá nome a um novo objeto
10
11 quem_eh(x)
12
```

```
10538048
10538048
10538048
10538048
10538080
```

Se o argumento é objeto mutavel

```
1  x=[1]
2  print(id(x))
3
4  def quem_eh(y):
5      print(id(x))
6      print(id(y)) #outro nome, mesmo objeto
7      y[0]+=1
8      print(id(x)) # x continua dando nome ao objeto 1
9      print(id(y)) # y dá nome ao mesmo objeto, o qual foi modificado
10
11 quem_eh(x)
12
```

```
139666956102472
139666956102472
139666956102472
139666956102472
139666956102472
```

- É possivel mudar o conteúdo sem desvincular o nome

mais args menos global

```
 1  a=5
 2  b=99
 3  l=[1,2,3]
 4
 5  def f():
 6      print(gx**2)
 7
 8  gx=a
 9  f()
10
11  gx=a+b
12  f()
13
14  for i in l:
15      gx=i
16      f()
```

```
 1  a=5
 2  b=99
 3  l=[1,2,3]
 4
 5  def f(x):
 6      print(x**2)
 7
 8  f(a)
 9  f(a+b)
10  for i in l:
11      f(i)
```

return

```
1  def converte_m_para_cm(x_in):
2      x_out = 100.*x_in
3      return x_out
4
5  medida_m=5.
6  medida_cm = converte_m_para_cm(medida_m)
7  print(medida_cm)
```

```
500.0
```

- Return objetos ou objetos

Exemplo: função de conversão de unidades de temperatura de F para C e vice-versa.

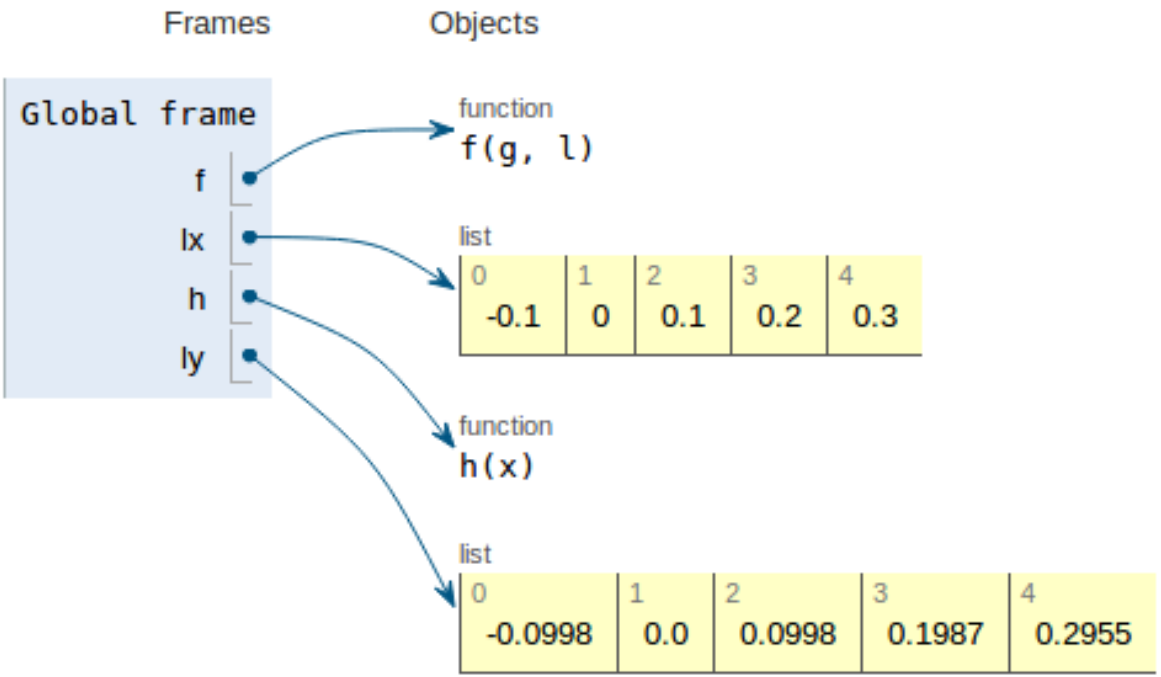
$$F = 1,8C + 32$$

Função f usando função g

```
1  def converte_m_para_cm(x_in):
2      x_out = 100.*x_in
3      return x_out
4
5  medida_m=5.
6  medida_cm = converte_m_para_cm(medida_m)
7  print(medida_cm)
8
9  def area_quadrado(l_m):
10     l_cm=converte_m_para_cm(l_m)
11     area_cm2=l_cm*l_cm
12     return area_cm2
13
14 lado_m=2.
15 area_cm2=area_quadrado(lado_m)
```

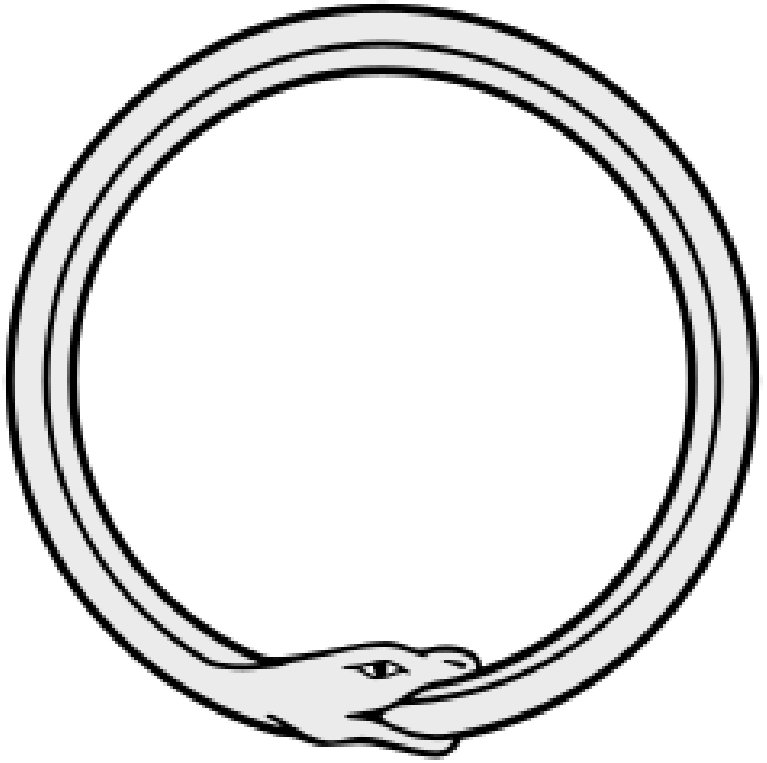
Função f recebendo uma função g por argumento

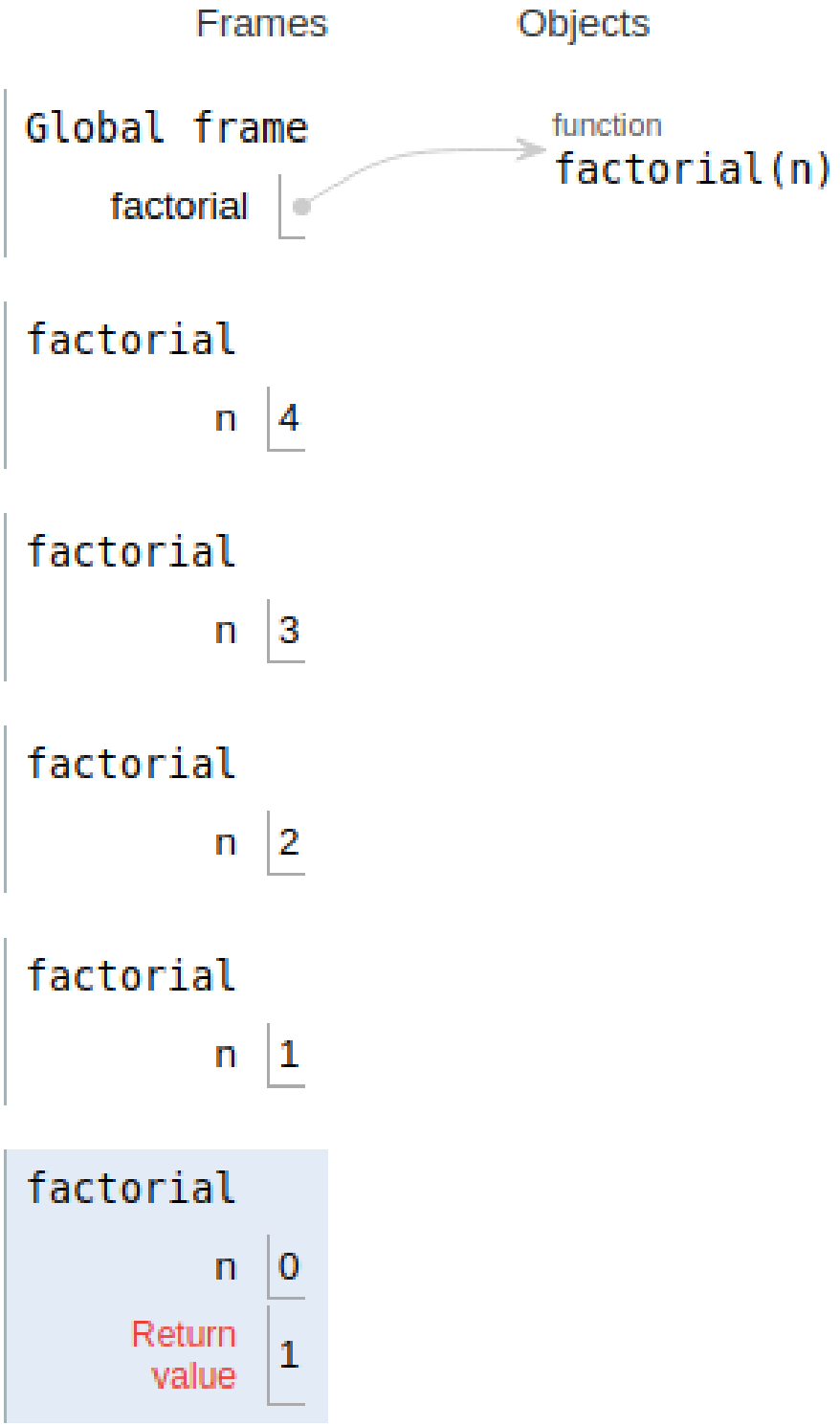
```
def f(g,l):
    fl=[]
    for li in l:
        fl.append(g(li))
    return fl

lx=[-.1,0,.1,.2,.3]

def h(x):
    from math import sin
    return sin(x)

ly=f(h,lx)
```



Função recursiva

```
def factorial( n ):
    if n <1:    # base case
        return 1
    else:                # recursive call
        return n * factorial( n - 1 )
```



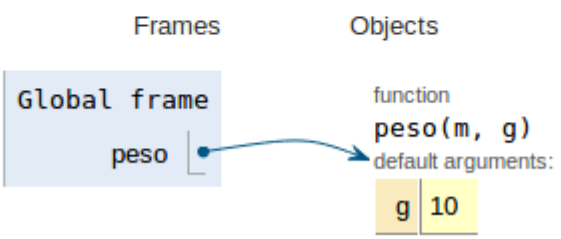


Opcionais e kargs

```
1  def peso(m,g=10):
2      p=m*g
3      return p
4
5  print(peso(100))
6  print(peso(100,g=9.8))
7  print(peso(100,g=3.711)) # em marte
8
```

Print output (drag lower right corner to resize)

```
1000
980.0000000000001
371.09999999999997
```



- kargs, argumentos palavra chave

closure

```
5  R=8.314
6
7  def v(n,T,P):
8      return n*R*T/P
9
10 Tamb=298.15
11 Patm=1e5
12 def v_amb_atm(n):
13     return v(n,T=Tamb,P=Patm)
14
15 print( v_amb_atm(3.) )
16
17
```

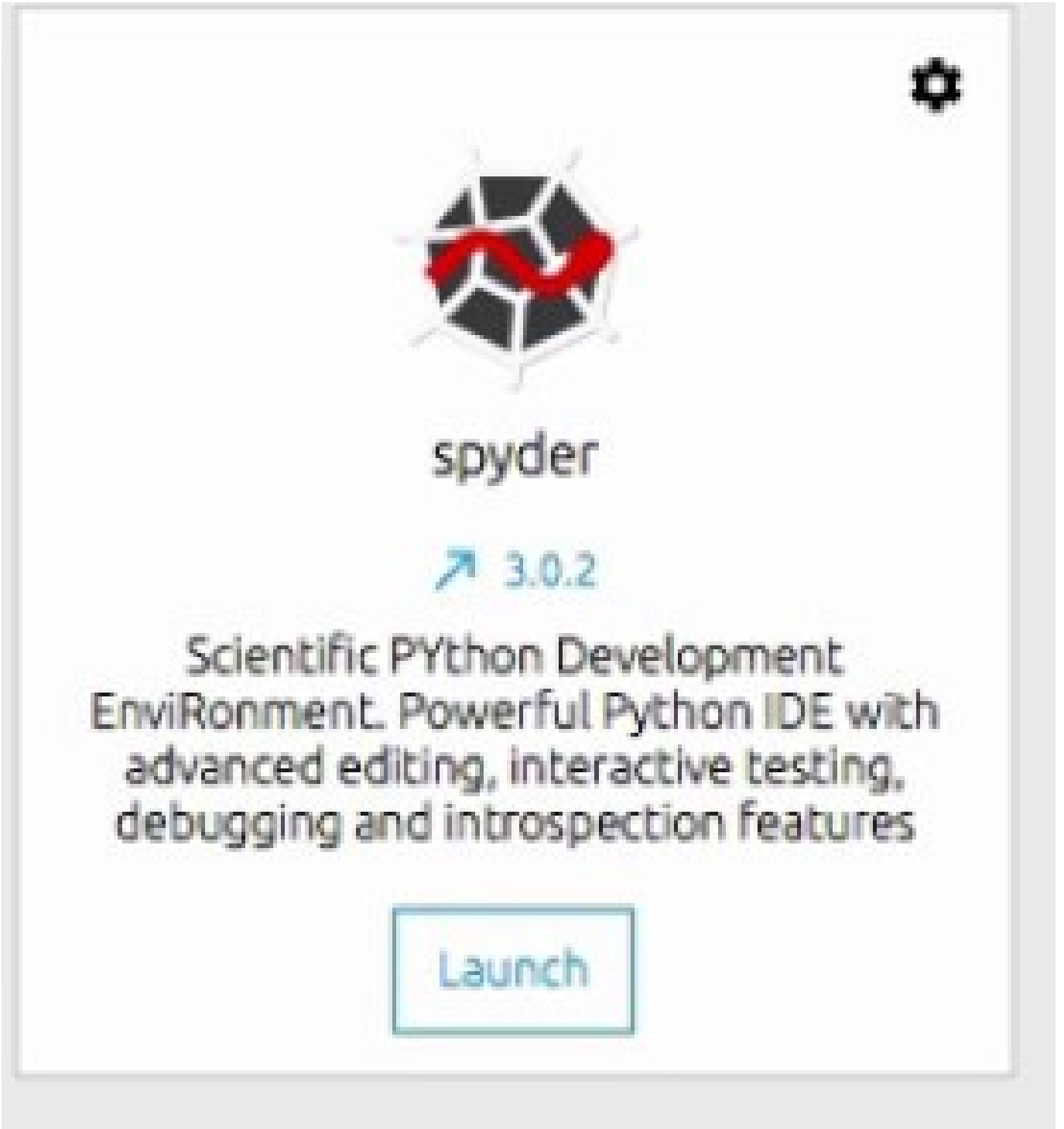
```
0.07436457299999999
```

- Pegar uma função de 3 argumentos e gerar uma função de 1 argumento, fixando os demais.

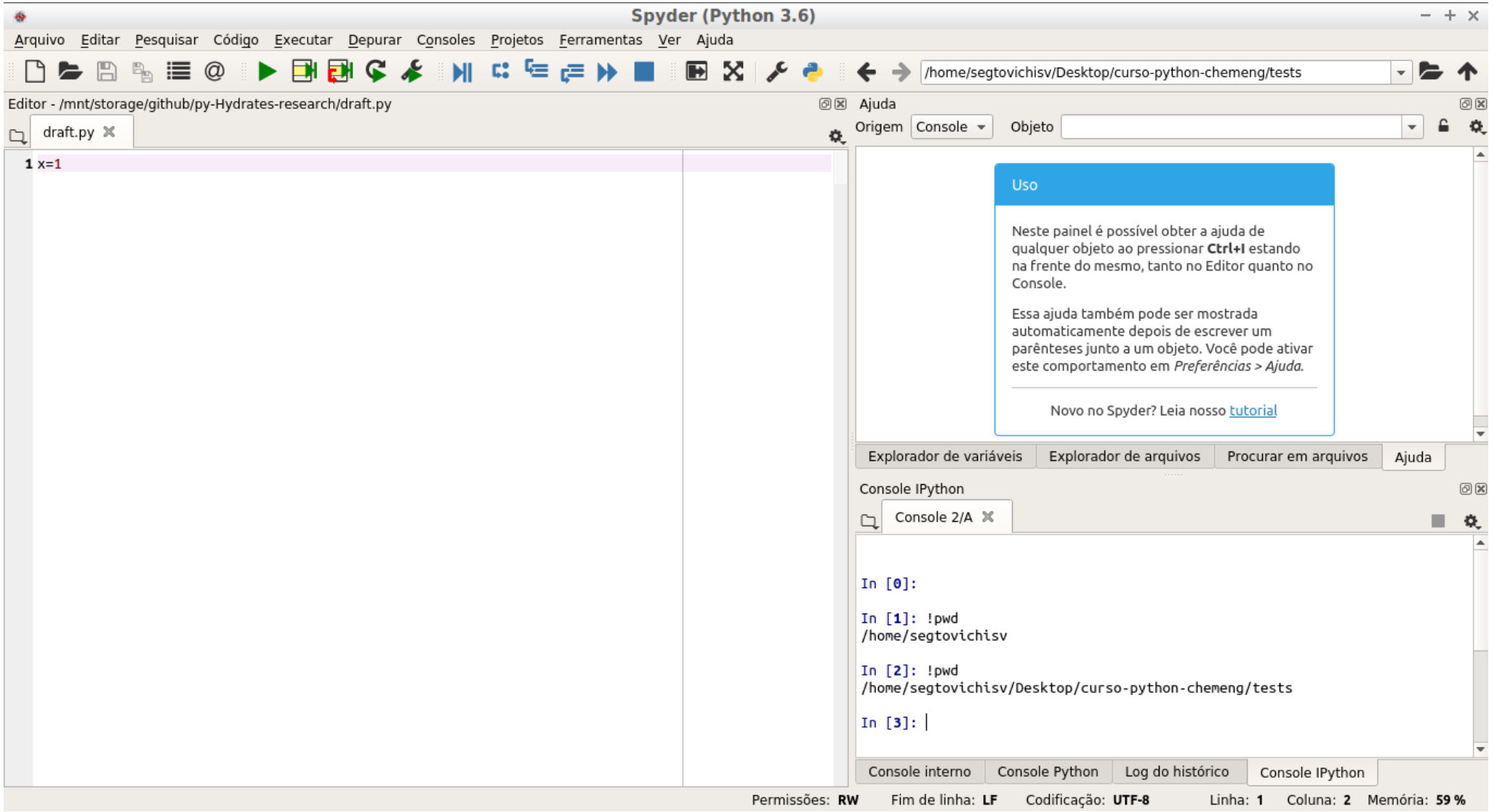
lambda

```
10  v_amb_atm = lambda n: v(n,Tamb,Patm)
11
12  print(v_amb_atm(3.))
13
```

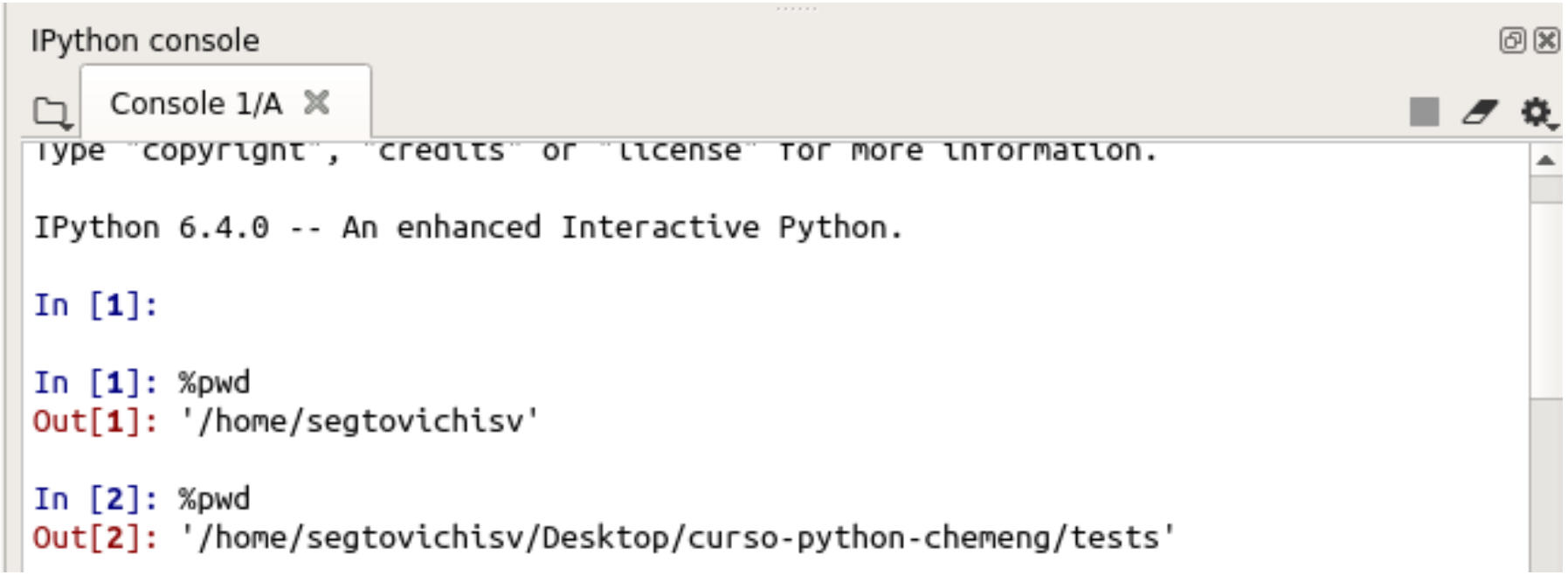
- lambda
  - nome_da_dummy
  - Dois pontos
  - Chamada de função usando
    - o nome da dummy
    - nomes definidos no namespace


spyder
3.0.2
Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features
Launch

Arquivos locais

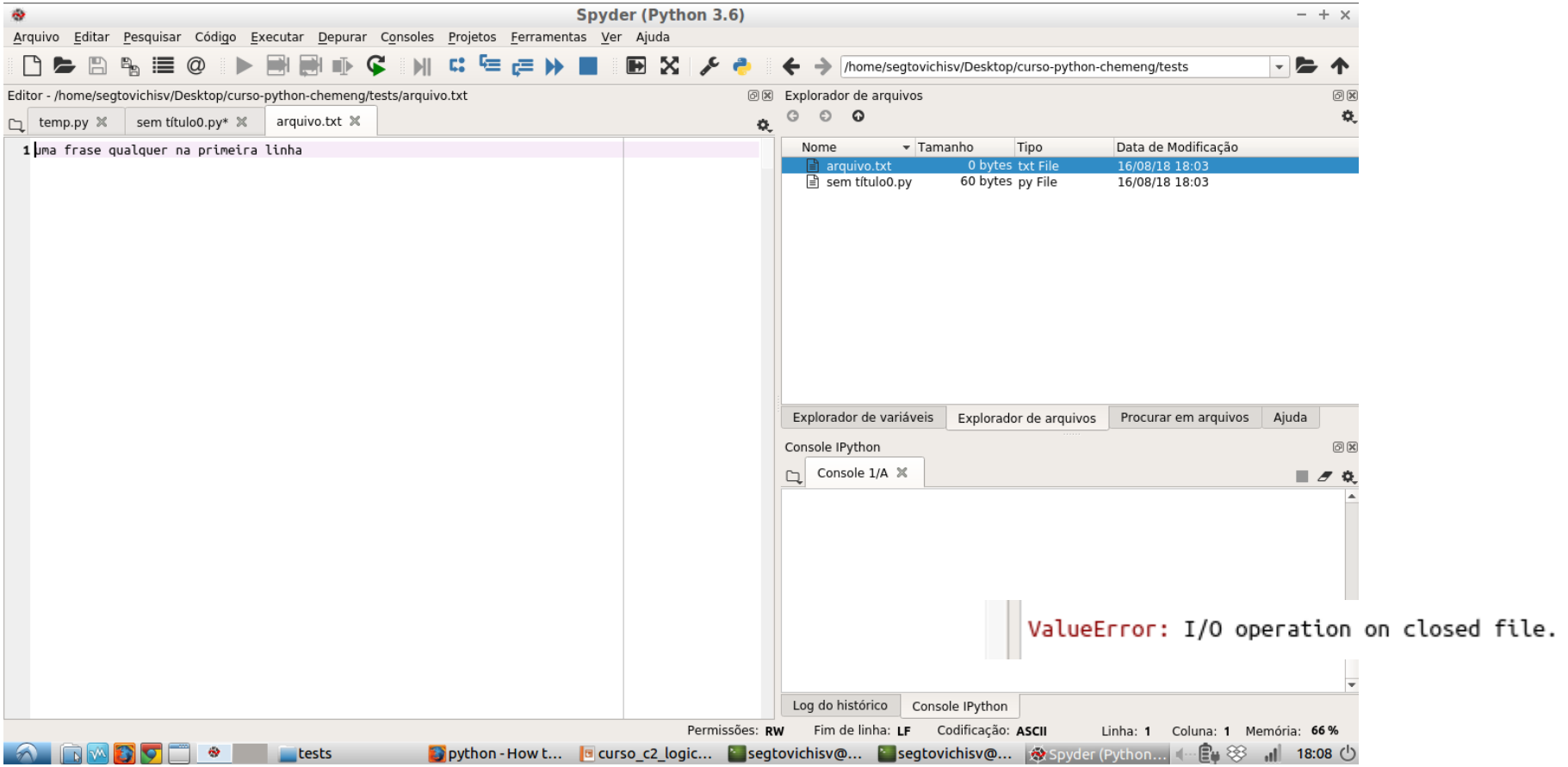


Working directory



Open, write

```
1 with open("arquivo.txt", 'w') as f:
2     print("abrindo...")
3     f.write("uma frase qualquer na primeira linha")
4 f.write("erro ") #arquivo fechado automaticamente após o "with"
```



format

```python
with open("arquivo.txt", 'w') as f:

    print("abrindo...")
    f.write("uma frase qualquer na primeira linha")
    f.write("\n")

    #formatação
    # '{ id : . precision type }'
    f.write('{:s}\n'.format("olá")) #imprimindo strings

    f.write('{:d}\n'.format(1))  #imprimindo inteiros

    f.write('{:.2e}\n'.format(1.963865200092e-3)) #imprimindo decimais
    f.write('{:.2f}\n'.format(1.963865200092e-3))
    f.write('{:.2e}\n'.format(1.963865200092e4))
    f.write('{:.2f}\n'.format(1.963865200092e4))

    #imprimindo várias coisas
    f.write("{:.2f} is greater than {:.2f}\n".format(2.,1.5))
    f.write("{1:.2f} isnt greater than {0:.2f}\n".format(2.,1.5))
```

```
uma frase qualquer na primeira linha
olá
1
1.96e-03
0.00
1.96e+04
19638.65
2.00 is greater than 1.50
1.50 isnt greater than 2.00

```

https://pyformat.info/

https://docs.python.org/3/library/string.html#string-formatting

readLINE

```
1 with open("input.txt", 'r') as f:
2
3     varname=f.readline()
4     print("alimentando", varname, "em x")
5     x=[]
6     linha=f.readline()
7     lista=linha.split(",")
8     for xi in lista:
9         print("convertendo", xi)
10        x.append(float(xi))
11
12 print("x: ",x)
13
14 n=len(x)
15 print("n: ",n)
16
17 with open("data.txt", 'r') as f:
18     import csv
19     data = csv.reader(f)
20     next(data) # skip header
21     T=[]
22     P=[]
23     for row in data:
24         T.append(float(row[0]))
25         P.append(float(row[1]))
26 print("T: ", T)
27 print("P: ", P)
```

```
alimentando x
 em x
convertendo .1
convertendo  .5
convertendo  .4

x:  [0.1, 0.5, 0.4]
n:  3
T:  [0.0, 1.1, 2.4, 3.2, 4.8, 5.2, 6.0]
P:  [1.0, 1.1, 1.0, 1.5, 1.6, 1.1, 2.1]
```

CSV.readER

```
1 with open("input.txt", 'r') as f:
2
3     varname=f.readline()
4     print("alimentando", varname, "em x")
5     x=[]
6     linha=f.readline()
7     lista=linha.split(",")
8     for xi in lista:
9         print("convertendo", xi)
10        x.append(float(xi))
11
12 print("x: ",x)
13
14 n=len(x)
15 print("n: ",n)
16
17 with open("data.txt", 'r') as f:
18     import csv
19     data = csv.reader(f)
20     next(data) # skip header
21     T=[]
22     P=[]
23     for row in data:
24         T.append(float(row[0]))
25         P.append(float(row[1]))
26 print("T: ", T)
27 print("P: ", P)
```

```
alimentando x
 em x
convertendo .1
convertendo  .5
convertendo  .4

x:  [0.1, 0.5, 0.4]
n:  3
T:  [0.0, 1.1, 2.4, 3.2, 4.8, 5.2, 6.0]
P:  [1.0, 1.1, 1.0, 1.5, 1.6, 1.1, 2.1]
```

ajuda

- Dir()

```
In [21]: dir(csv)
Out[21]:
['Dialect',
 'DictReader',
 'DictWriter',
 'Error',
 'OrderedDict',
 'QUOTE_ALL',
 'QUOTE_MINIMAL',
 'QUOTE_NONE',
 'QUOTE_NONNUMERIC',
 'Sniffer',
 'StringIO',
 '_Dialect',
 '__all__',
 '__builtins__',
 '__cached__',
 '__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 'excel',
 'excel_tab',
 'field_size_limit',
 'get_dialect',
 'list_dialects',
 're',
 'reader',
 'register_dialect',
 'unix_dialect',
 'unregister_dialect',
 'writer']
```

- Help()
- Objeto?
- Objeto??

```
In [22]: help(csv.reader)
Help on built-in function reader in module _csv:

reader(...)
    csv_reader = reader(iterable [, dialect='excel']
                            [optional keyword args])
        for row in csv_reader:
            process(row)

    The "iterable" argument can be any object that returns a line
    of input for each iteration, such as a file object or a list.  The
    optional "dialect" parameter is discussed below.  The function

    also accepts optional keyword arguments which override settings
    provided by the dialect.

    The returned object is an iterator.  Each iteration returns a row
    of the CSV file (which can span multiple input lines).
```

•Module

- "Leitura.py"
- New console / restart kernel
- Programa.py

```
1 import Leitura
2
3 print(Leitura.x, Leitura.T, Leitura.
```

```
alimentando x
 em x
convertendo .1
convertendo  .5
convertendo  .4

x:  [0.1, 0.5, 0.4]
n:  3
T:  [0.0, 1.1, 2.4, 3.2, 4.8, 5.2, 6.0]
P:  [1.0, 1.1, 1.0, 1.5, 1.6, 1.1, 2.1]
[0.1, 0.5, 0.4] [0.0, 1.1, 2.4, 3.2, 4.8, 5.2, 6.0] [1.0, 1.1, 1.0, 1.5, 1.6,
1.1, 2.1]
```

•Library-like module

Leitura.py | programa.py

```
'''
esse módulo faz a leitura'''

def fazer_leitura():
    with open("input.txt", 'r') as f:
        varname=f.readline()
#        print("alimentando", varname, "em x")
        x=[]
        linha=f.readline()
        lista=linha.split(",")
        for xi in lista:
#            print("convertendo", xi)
            x.append(float(xi))
#    print("x: ",x)
    n=len(x)
#    print("n: ",n)
    with open("data.txt", 'r') as f:
        import csv
        data = csv.reader(f)
        next(data) # skip header
        T=[]
        P=[]
        for row in data:
            T.append(float(row[0]))
            P.append(float(row[1]))
#    print("T: ", T)
#    print("P: ", P)
    return n, x, T, P
```

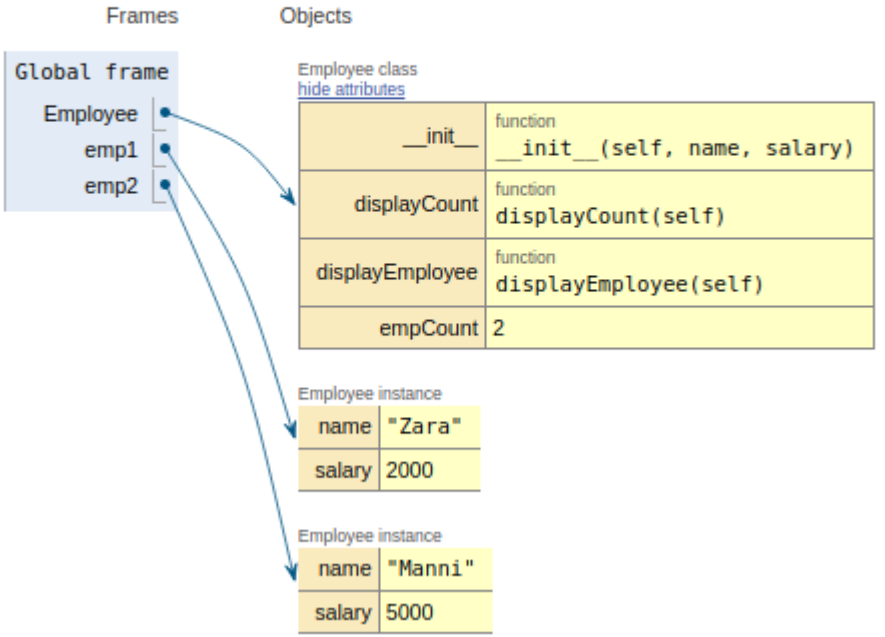
```
import Leitura
print(dir(Leitura))
print(help(Leitura))
n,x,T,P = Leitura.fazer_leitura()
print(n,x,T,P)
```

```
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__',
'__package__', '__spec__', 'fazer_leitura']
Help on module Leitura:

NAME
    Leitura - esse módulo faz a leitura

FUNCTIONS
    fazer_leitura()

FILE
    /home/segtovichisv/Desktop/curso-python-chemeng/tests/Leitura.py


None
3 [0.1, 0.5, 0.4] [0.0, 1.1, 2.4, 3.2, 4.8, 5.2, 6.0] [1.0, 1.1, 1.0, 1.5, 1.6,
1.1, 2.1]
```

OOP

```
1  class Employee:
2     'Common base class for all employees'
3     empCount = 0
4
5     def __init__(self, name, salary):
6        self.name = name
7        self.salary = salary
8        Employee.empCount += 1
9
10    def displayCount(self):
11      print("Total Employee %d" % Employee.empCount)
12
13    def displayEmployee(self):
14       print("Name : ", self.name,  ", Salary: ", self.salary)
15
16 "This would create first object of Employee class"
17 emp1 = Employee("Zara", 2000)
18 "This would create second object of Employee class"
19 emp2 = Employee("Manni", 5000)
20 emp1.displayEmployee()
21 emp2.displayEmployee()
22 print("Total Employee %d" % Employee.empCount)
```

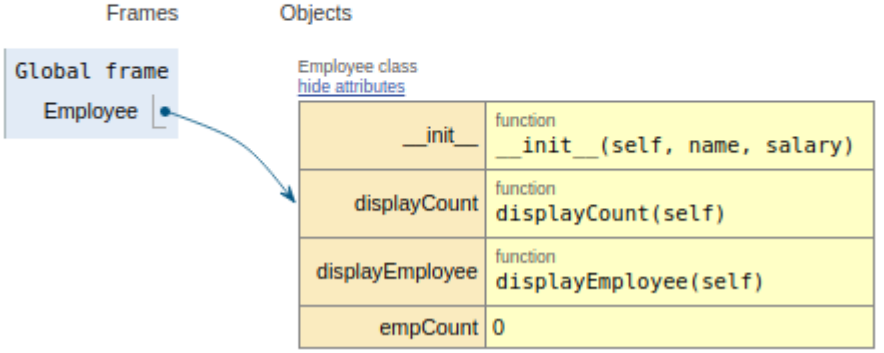
```
Name :  Zara , Salary:  2000
Name :  Manni , Salary:  5000
Total Employee 2
```



Built-In Class Attributes

```
1  class Employee:
2     'Common base class for all employees'
3     empCount = 0
4
5     def __init__(self, name, salary):
6        self.name = name
7        self.salary = salary
8        Employee.empCount += 1
9
10    def displayCount(self):
11      print("Total Employee %d" % Employee.empCount)
12
13    def displayEmployee(self):
14       print("Name : ", self.name,  ", Salary: ", self.salary)
15
16 print("Employee.__doc__:", Employee.__doc__)
17 print("Employee.__name__:", Employee.__name__)
18 print("Employee.__module__:", Employee.__module__)
19 print("Employee.__bases__:", Employee.__bases__)
20 print("Employee.__dict__:", Employee.__dict__)
```

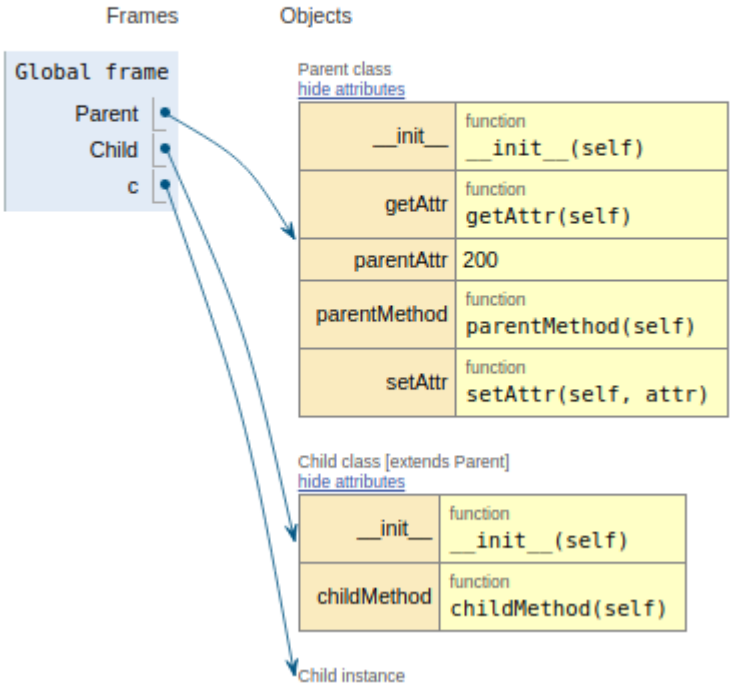
```
Employee.__doc__: Common base class for all emplo
Employee.__name__: Employee
Employee.__module__: __main__
Employee.__bases__: (<class 'object'>,)
Employee.__dict__: {'__module__': '__main__', '__
```



inheritance

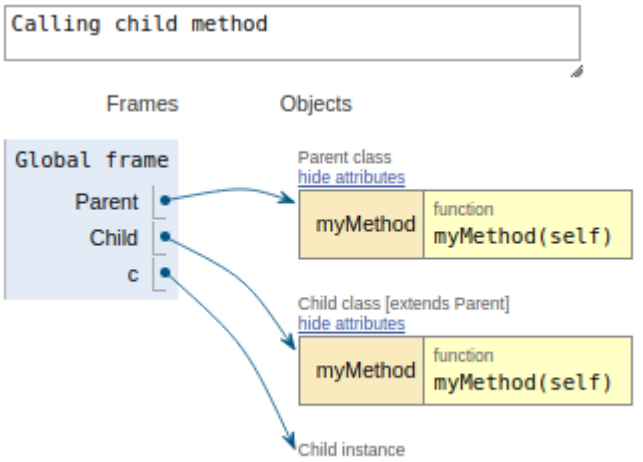
```python
#!/usr/bin/python

class Parent:        # define parent class
   parentAttr = 100
   def __init__(self):
      print("Calling parent constructor")

   def parentMethod(self):
      print('Calling parent method')

   def setAttr(self, attr):
      Parent.parentAttr = attr

   def getAttr(self):
      print("Parent attribute :", Parent.parentAttr)

class Child(Parent): # define child class
   def __init__(self):
      print("Calling child constructor")

   def childMethod(self):
      print('Calling child method')

c = Child()          # instance of child
c.childMethod()      # child calls its method
c.parentMethod()     # calls parent's method
c.setAttr(200)       # again call parent's method
c.getAttr()          # again call parent's method
```

```
Calling child constructor
Calling child method
Calling parent method
Parent attribute : 200
```



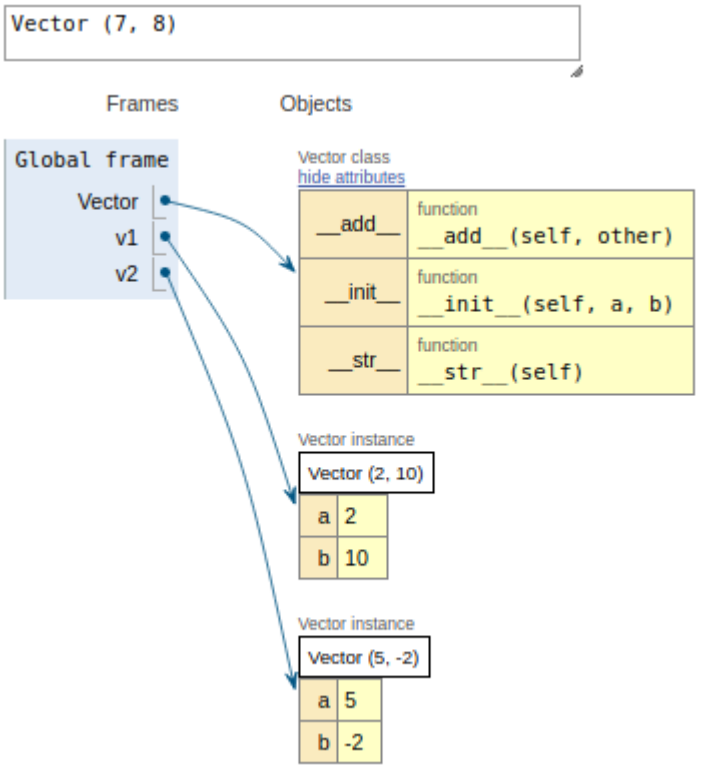
Method override

```
1  #!/usr/bin/python
2
3  class Parent:        # define parent class
4     def myMethod(self):
5        print('Calling parent method')
6
7  class Child(Parent): # define child class
8     def myMethod(self):
9        print('Calling child method')
10
11 c = Child()          # instance of child
12 c.myMethod()         # child calls overridden method
```



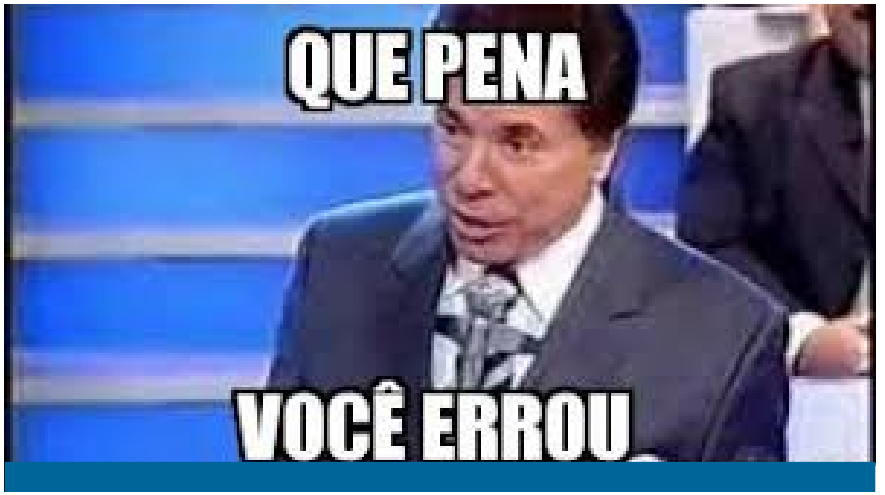
Operator overload

```
1  #!/usr/bin/python
2
3  class Vector:
4     def __init__(self, a, b):
5        self.a = a
6        self.b = b
7
8     def __str__(self):
9        return 'Vector (%d, %d)' % (self.a, self.b)
10
11    def __add__(self,other):
12       return Vector(self.a + other.a, self.b + other.b)
13
14 v1 = Vector(2,10)
15 v2 = Vector(5,-2)
16 print(v1 + v2)
```



Gerenciamento de exceções

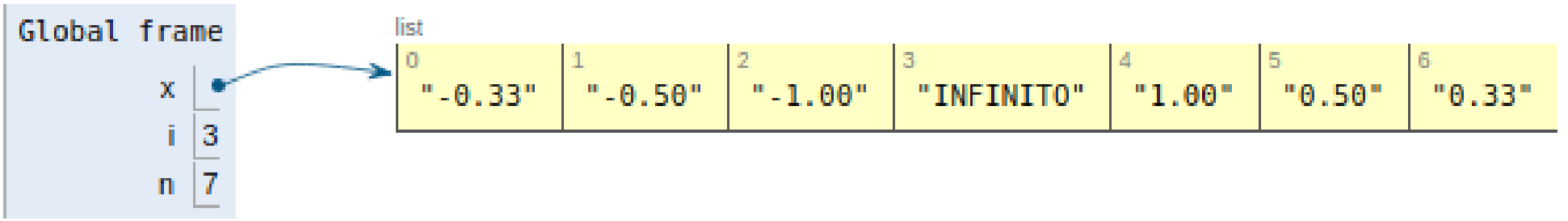
```
x=0
a=10
b=0
try:
    temp=x
    x+=a
    x/=b
except Exception as error:
    print("Que pena, você errou.")
    print(error)
finally:
    x=temp
    print("restaurando x a partir de temp")

print("a vida continua")
```




```
Que pena, você errou.
division by zero
restaurando x a partir de temp
a vida continua
```

Exemplo

```
x=[]
for i in range(-3,4):
    try:
        x.append('{:.2f}'.format(1/i))
    except Exception:
        x.append("INFINITO")
    finally:
        n=len(x)
```

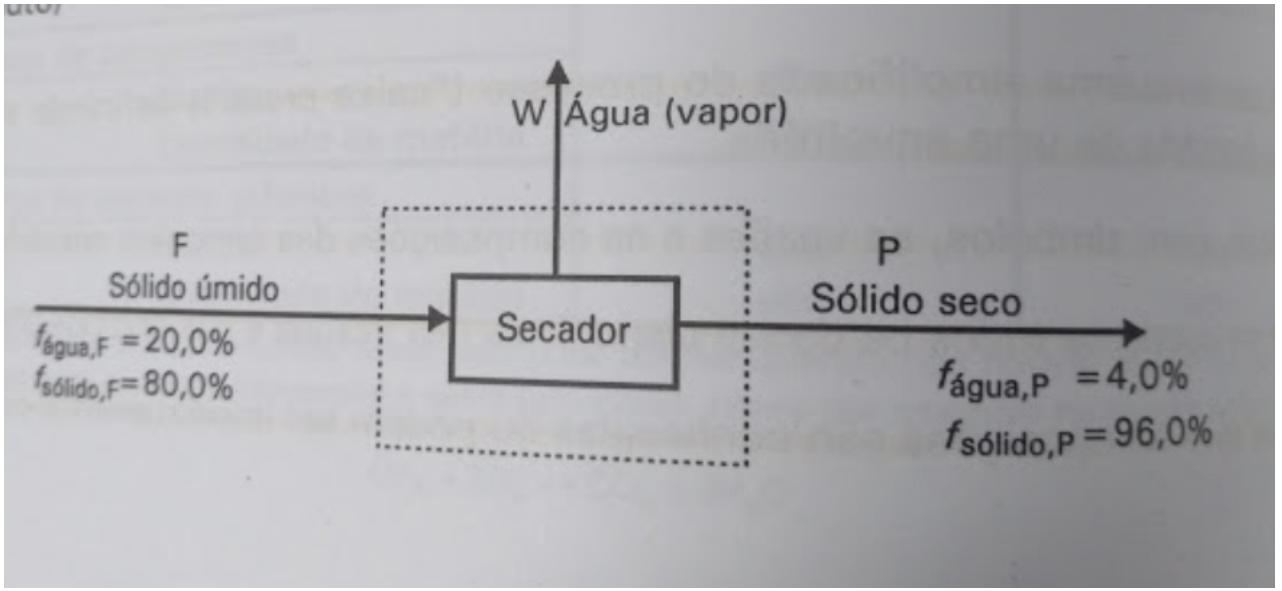


Prepare sua exceção

```
def funcao_bugada(x):
    if x<0:
        raise ValueError('Deu ruim aqui, x não pode ser negativo!
    else:
        return x**.5+x**1.5+x**2.5

y1=funcao_bugada(2.)
y2=funcao_bugada(-2.)
```

```
ValueError: Deu ruim aqui, x não pode ser negativo!
```

Exercício



$$P = F\frac{x_s^f}{x_s^p}$$

$$W = Fx_w^f - Px_w^p$$

$$R\% = 100\frac{W}{Fx_w^f}$$

```python
def CalcRecSecador(xfs,xfw,xps,xpw):
    #base de cálculo
    F=100
    #bm sólido
    P=F*xfs/xps
    W=F*xfw-P*xpw
    Rec=W/(F*xfw)*100
    return P,W,Rec

P, W, Rec = CalcRecSecador(.8,.2,.96,.04)

print("P, W, Rec %")
print(P, W, Rec)
```

```
P, W, Rec %
83.33333333333334 16.666666666666664 83.33333333333333
```

Referências principais

https://www.tutorialspoint.com/

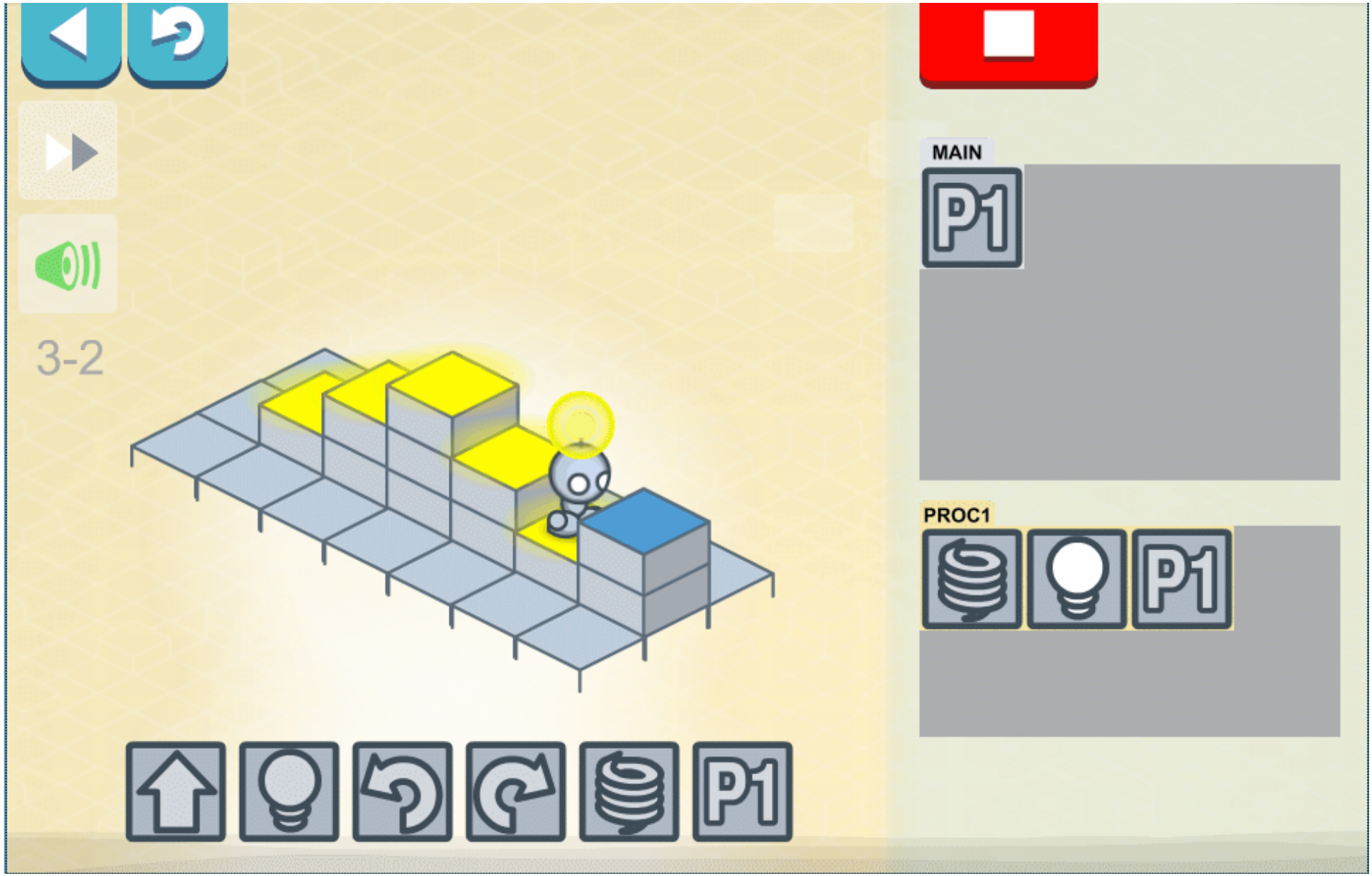
python3/

python_basic_syntax.htm

https://stackoverflow.com/

search

EXERCÍCIO DE lógica DE PROGRAMAÇÃO



perguntas