

FUNÇÕES

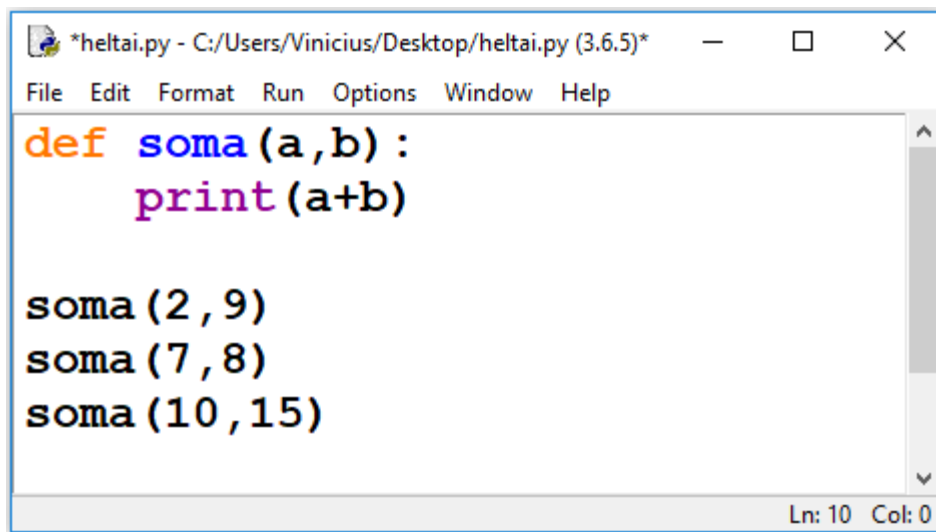


FUNÇÃO

FUNÇÃO

FUNÇÕES

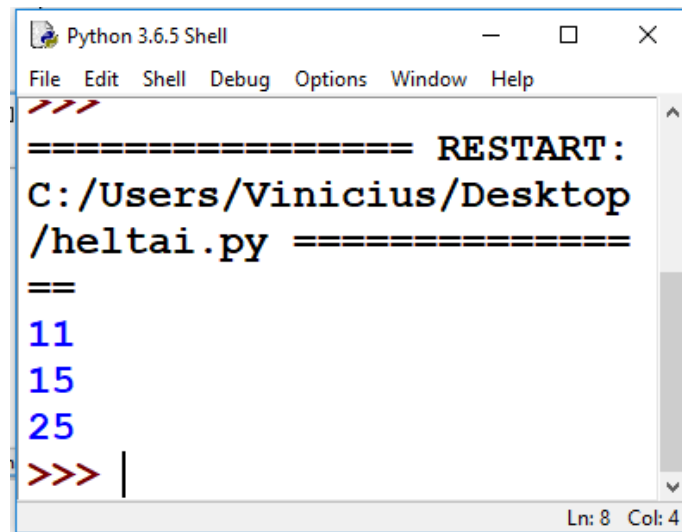
- Durante os estudos, foram utilizadas diversos tipos de funções. Ex.: **len**, **int**, **float**, **print**, **input**.
- Para definir uma nova função, utiliza-se a instrução **def**.



The screenshot shows a Python IDE window titled '*heltai.py - C:/Users/Vinicius/Desktop/heltai.py (3.6.5)*'. The menu bar includes File, Edit, Format, Run, Options, Window, and Help. The code in the editor is as follows:

```
def soma(a,b):  
    print(a+b)  
  
soma(2,9)  
soma(7,8)  
soma(10,15)
```

The status bar at the bottom indicates 'Ln: 10 Col: 0'.



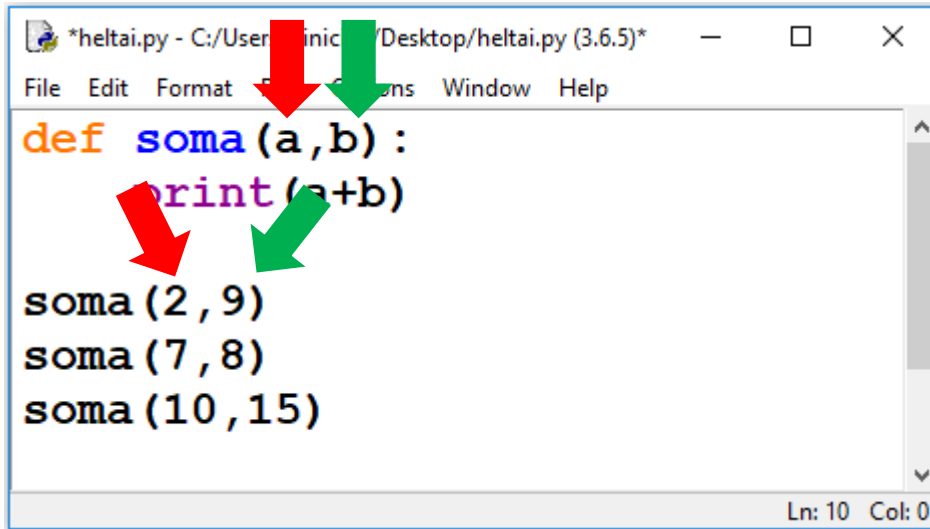
The screenshot shows a Python 3.6.5 Shell window titled 'Python 3.6.5 Shell'. The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The output of the program is displayed as follows:

```
==== RESTART: ====  
C:/Users/Vinicius/Desktop  
/heltai.py  
==  
11  
15  
25  
>>> |
```

The status bar at the bottom indicates 'Ln: 8 Col: 4'.

FUNÇÃO

- Funções são interessantes para isolar uma tarefa específica em um trecho do programa. Isso permite que a solução de um problema seja reutilizada em outras partes do programa, sem precisar repetir as mesmas linhas de programação.
- As informações dentro dos parênteses, são chamadas de **parâmetros**.
- Os parâmetros são substituídos na mesma ordem em que foram definidos. No exemplo passado, o número 2 vai ser atribuído ao parâmetro a e o valor 9 vai ser atribuído ao parâmetro b:

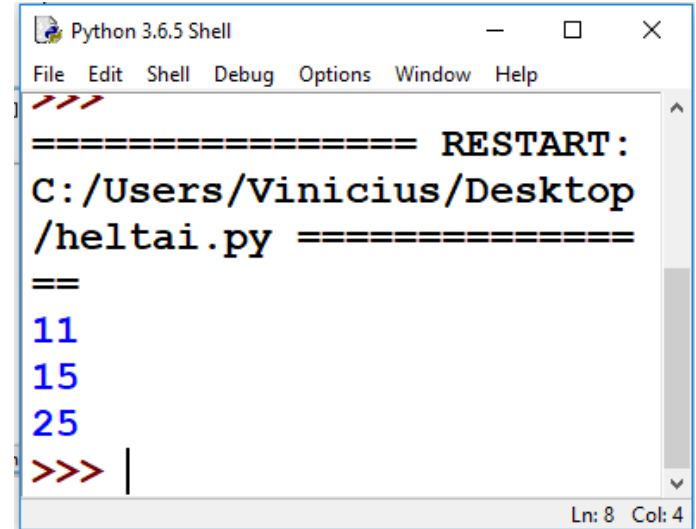


The screenshot shows a Python script editor window titled '*heltai.py - C:/Users/Vinicius/Desktop/heltai.py (3.6.5)*'. The code contains a function definition and three calls:

```
def soma(a,b):  
    print(a+b)  
  
soma(2,9)  
soma(7,8)  
soma(10,15)
```

Two arrows are overlaid on the code: a red arrow pointing from the '2' in the first call to the 'a' parameter in the function definition, and a green arrow pointing from the '9' in the first call to the 'b' parameter in the function definition.

Ln: 10 Col: 0



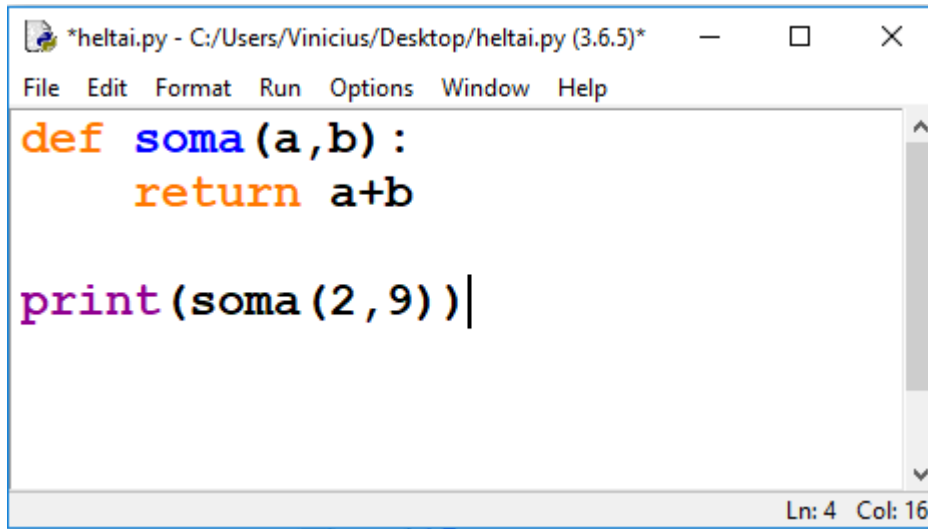
The screenshot shows a Python 3.6.5 Shell window titled 'Python 3.6.5 Shell'. The output of the script is displayed:

```
==== RESTART: C:/Users/Vinicius/Desktop/heltai.py =====  
==  
11  
15  
25  
>>> |
```

Ln: 8 Col: 4

FUNÇÃO

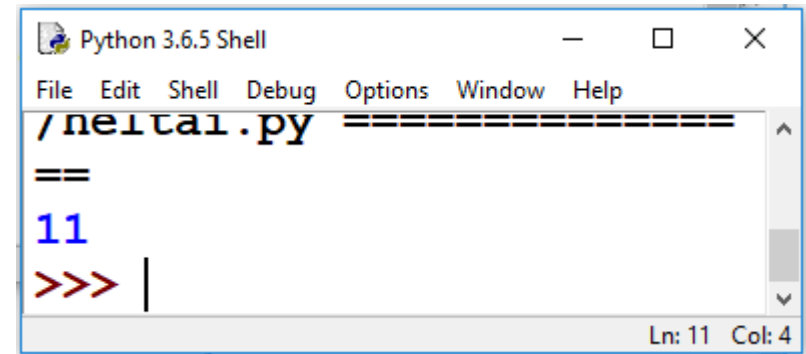
- No exemplo, a função não retorna valor para a função que chamou (invocou) essa função.
- Quando a função retorna um valor, usa-se o comando **return**.
- A função **return**, faz com que a função pare de executar e que o valor seja retornado imediatamente ao programa ou à função que chamou. Assim como ocorre com a função **break** dentro de laços.



```
*heltai.py - C:/Users/Vinicius/Desktop/heltai.py (3.6.5)*
File Edit Format Run Options Window Help
def soma(a,b):
    return a+b

print(soma(2,9))|
```

Ln: 4 Col: 16



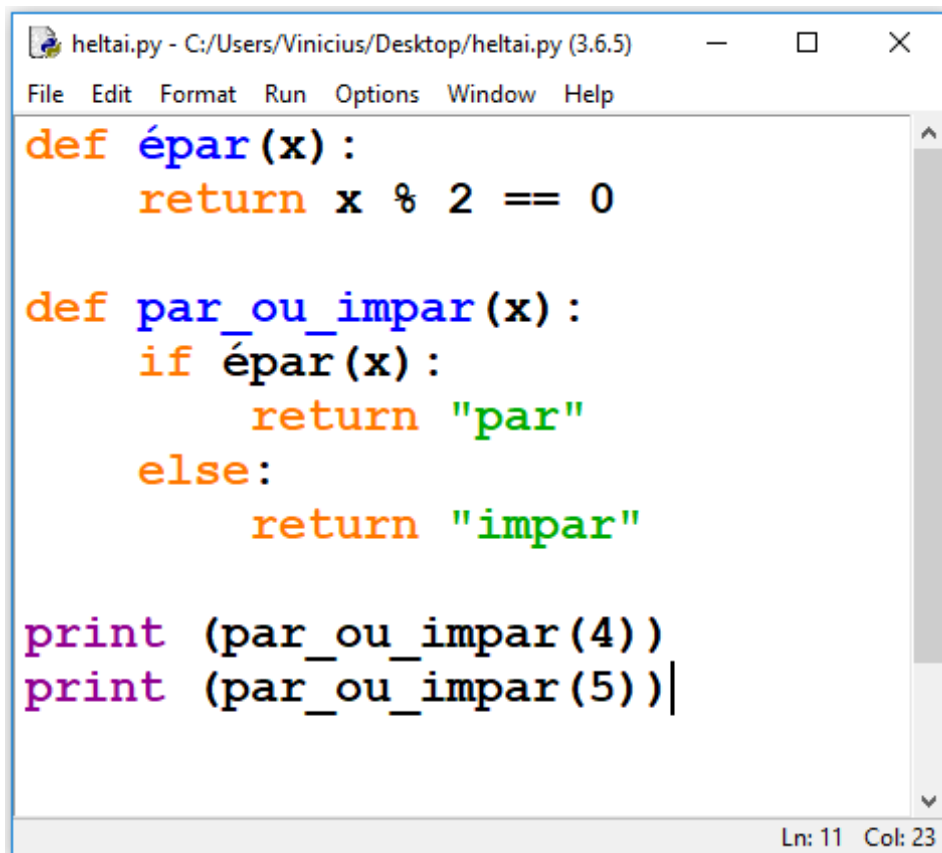
```
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
/heltai.py =====
==
11
>>> |
```

Ln: 11 Col: 4

- No exemplo acima, observe que a função soma realiza apenas a operação. O que será executado com o resultado, fica a encargo da rotina que chamou. No caso foi impresso o valor.

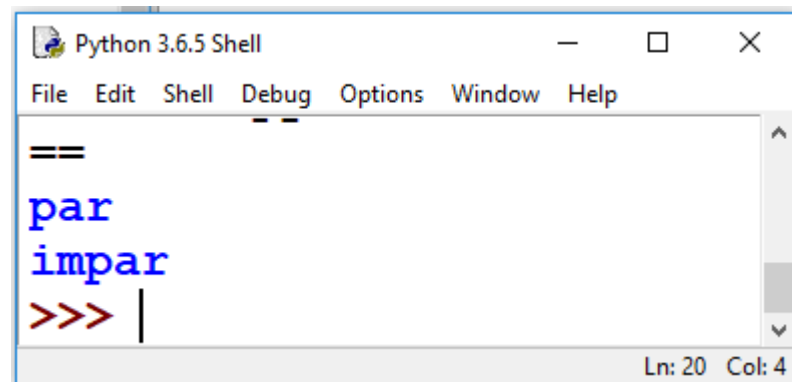
FUNÇÃO

- Outro exemplo: Na função abaixo, define-se uma função para retornar a palavra par ou impar.



```
def épar(x):  
    return x % 2 == 0  
  
def par_ou_impar(x):  
    if épar(x):  
        return "par"  
    else:  
        return "impar"  
  
print (par_ou_impar(4))  
print (par_ou_impar(5))
```

Ln: 11 Col: 23



```
==  
par  
impar  
>>> |
```

Ln: 20 Col: 4

FUNÇÃO

CALCULO DE SOMA E MEDIA :

```
heltai.py - C:/Users/Vinicius/Desktop/heltai.py (3.6.5)
File Edit Format Run Options Window Help
L = [10, 20, 30, 40]

def soma(L):
    total = 0
    for e in L:
        total += e
    return total

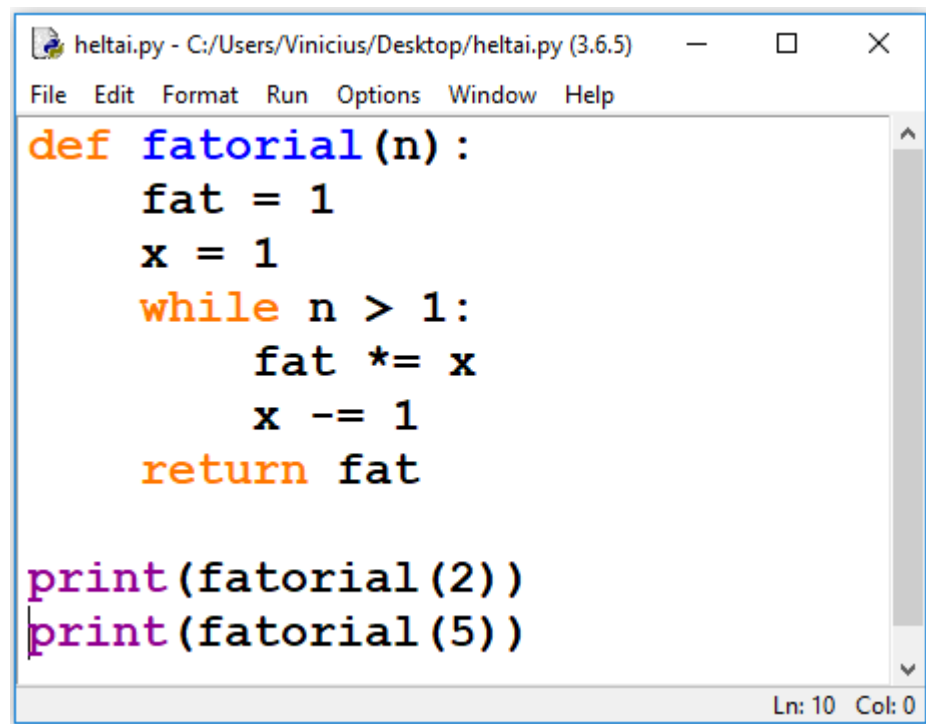
def media(L):
    return soma(L) / len(L)

print (media(L))
print (soma(L))
Ln: 6 Col: 0
```

```
Python 3.6.5 Shell
File Edit Shell Debug Options Window
Help
25.0
100
>>> |
Ln: 7 Col: 4
```

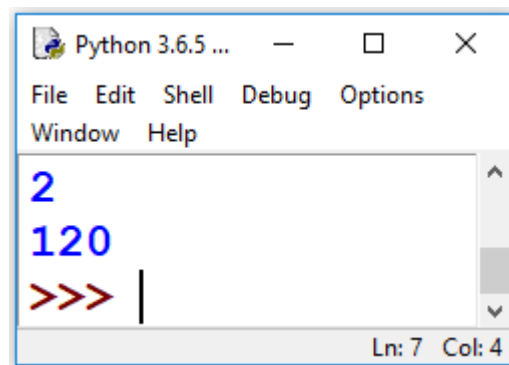
FUNÇÃO

CALCULO DE FATORIAL – TIPO 1:



```
heltai.py - C:/Users/Vinicius/Desktop/heltai.py (3.6.5)
File Edit Format Run Options Window Help
def fatorial(n):
    fat = 1
    x = 1
    while n > 1:
        fat *= x
        x -= 1
    return fat

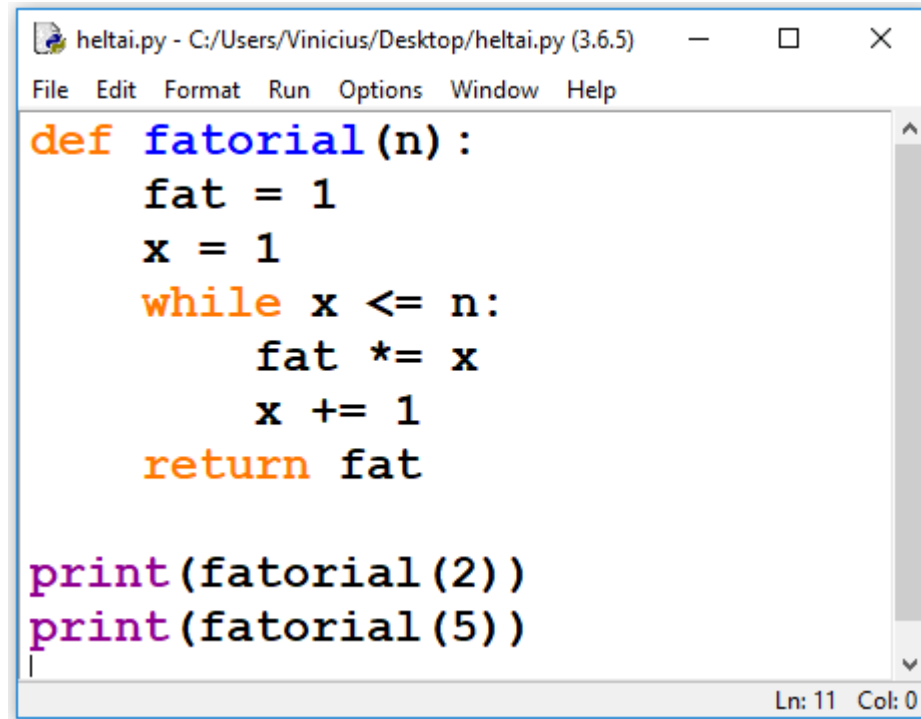
print(fatorial(2))
print(fatorial(5))
Ln: 10 Col: 0
```



```
Python 3.6.5 ...
File Edit Shell Debug Options
Window Help
2
120
>>> |
Ln: 7 Col: 4
```


FUNÇÃO

CALCULO DE FATORIAL – TIPO 2:

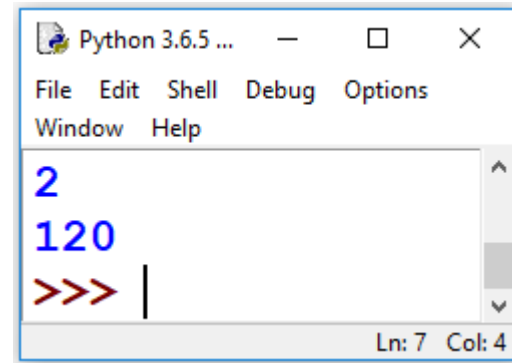


```
heltai.py - C:/Users/Vinicius/Desktop/heltai.py (3.6.5)
File Edit Format Run Options Window Help

def fatorial(n):
    fat = 1
    x = 1
    while x <= n:
        fat *= x
        x += 1
    return fat

print(fatorial(2))
print(fatorial(5))
|
```

Ln: 11 Col: 0



```
Python 3.6.5 ...
File Edit Shell Debug Options
Window Help

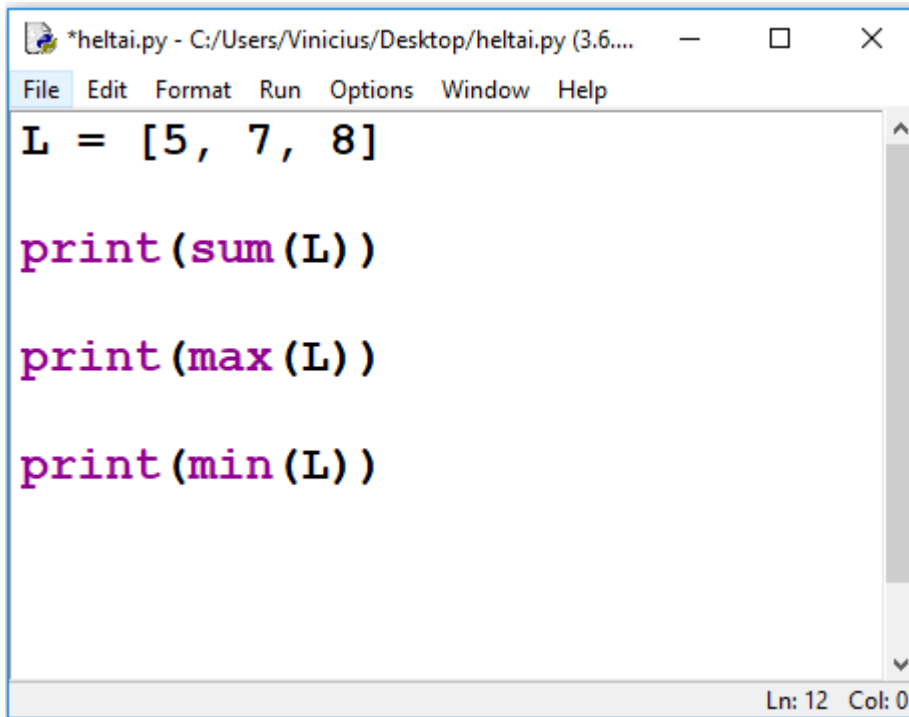
2
120
>>> |
```

Ln: 7 Col: 4

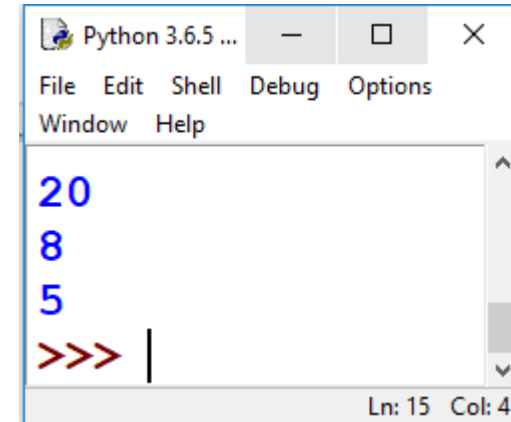
FUNÇÃO

FUNÇÃO PARA SOMA, MAXIMO E MINIMO:

- O python, já apresenta funções especificar para indicar soma, máximo e mínimo. Conforme exemplo abaixo:



```
*helta.py - C:/Users/Vinicius/Desktop/helta.py (3.6....  
File Edit Format Run Options Window Help  
L = [5, 7, 8]  
  
print(sum(L))  
  
print(max(L))  
  
print(min(L))  
  
Ln: 12 Col: 0
```



```
Python 3.6.5 ...  
File Edit Shell Debug Options  
Window Help  
20  
8  
5  
>>> |  
Ln: 15 Col: 4
```



VARIÁVEIS LOCAIS E GLOBAIS

VARIÁVEIS LOCAIS E GLOBAIS

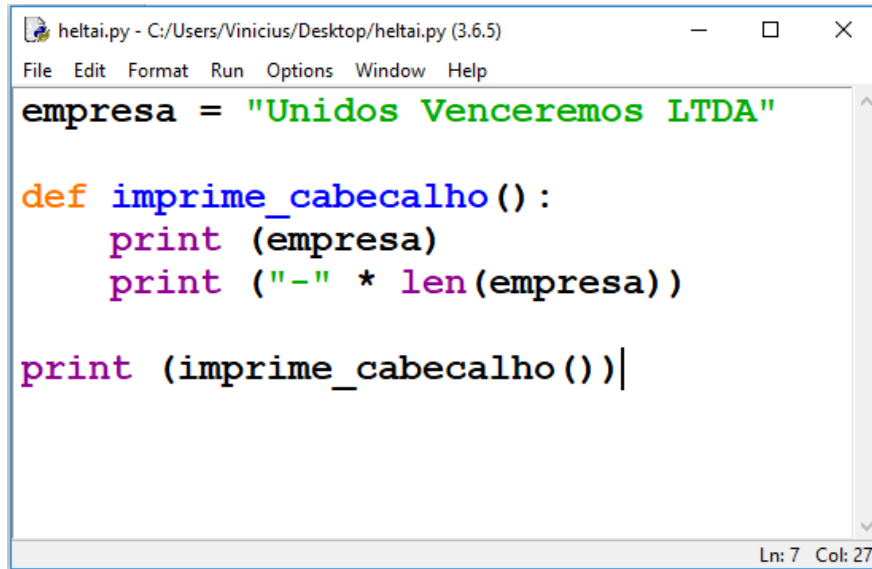
VARIAVEIS LOCAIS E GLOBAIS - DIFERENÇA

- Quando usamos funções, começamos a trabalhar com variáveis internas ou locais e com variáveis externas ou globais.
- A diferença entre elas é a visibilidade ou escopo
- **VARIAVEL LOCAL** – Alocado em uma função, existe apenas dentro dela, sendo normalmente inicializada a cada chamada. Desta forma, não podemos acessar o valor de uma variável local fora da função que a criou e, por isso, passamos parâmetros e retornamos valores nas funções, de forma a possibilitar a troca de dados no programa.
- **VARIAVEL GLOBAL** – É definida fora de uma função, podendo ser vista por todas as funções do módulo (programa) e por todos os módulos que importam o módulo que a definiu. As variáveis globais devem ser utilizadas o mínimo possível, pois dificultam a leitura e violam o encapsulamento da função.

VARIÁVEIS LOCAIS E GLOBAIS

EXEMPLO:

- No exemplo abaixo, observe que a função **imprime_cabecalho** não recebe parâmetros nem retorna valores. Ela simplesmente imprime o nome da empresa e traços abaixo dele. Observe que foi utilizado a variável **empresa** definida fora da função, desta forma, é uma variável global.

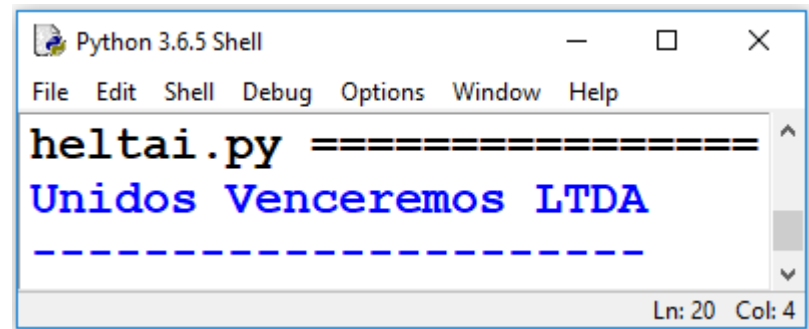


```
heltai.py - C:/Users/Vinicius/Desktop/heltai.py (3.6.5)
File Edit Format Run Options Window Help
empresa = "Unidos Venceremos LTDA"

def imprime_cabecalho():
    print (empresa)
    print ("-" * len(empresa))

print (imprime_cabecalho())
```

Ln: 7 Col: 27

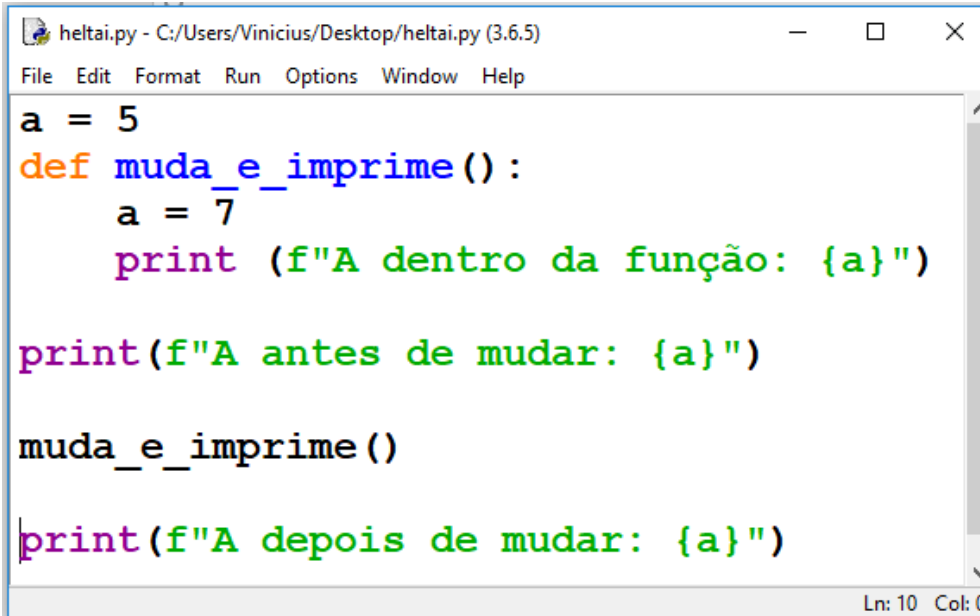


```
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
heltai.py =====
Unidos Venceremos LTDA
-----
Ln: 20 Col: 4
```

VARIÁVEIS LOCAIS E GLOBAIS

CUIDADOS:

- O Python tem a capacidade de declarar variáveis à medida que precisarmos, devemos tomar cuidado quando altera uma variável global dentro de uma função:

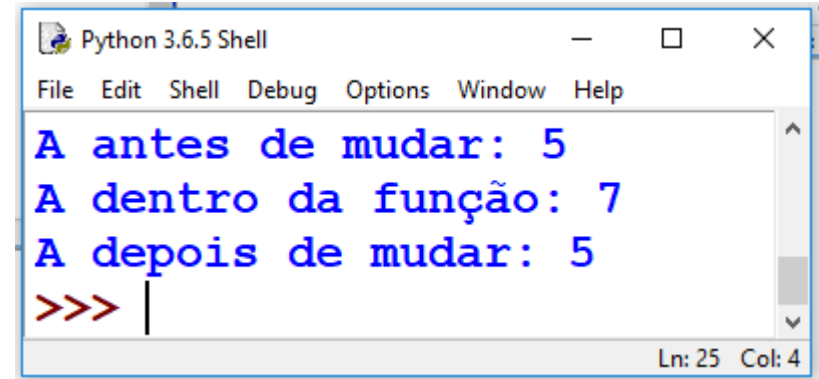


```
heltai.py - C:/Users/Vinicius/Desktop/heltai.py (3.6.5)
File Edit Format Run Options Window Help
a = 5
def muda_e_imprime():
    a = 7
    print (f"A dentro da função: {a}")

print(f"A antes de mudar: {a}")

muda_e_imprime()

print(f"A depois de mudar: {a}")
Ln: 10 Col: 0
```



```
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
A antes de mudar: 5
A dentro da função: 7
A depois de mudar: 5
>>> |
Ln: 25 Col: 4
```

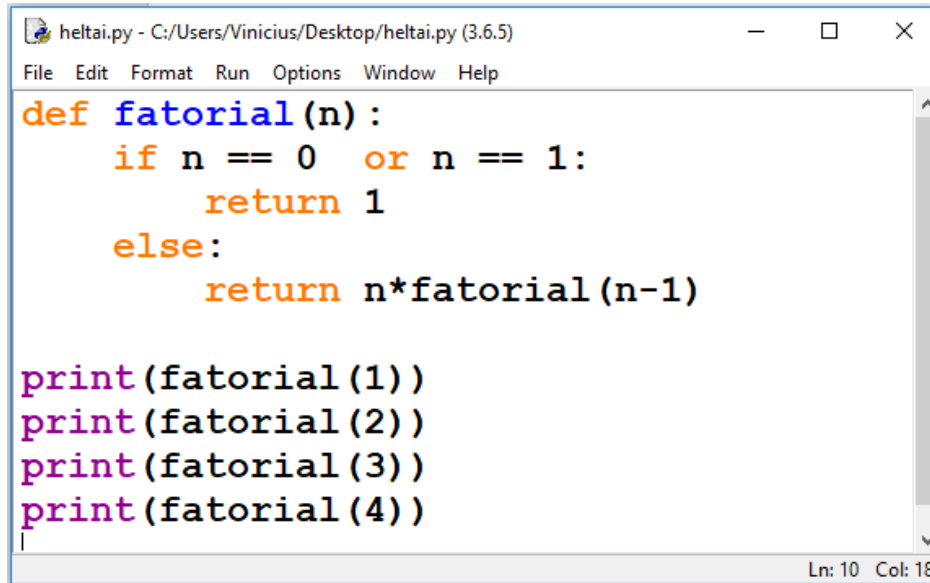


FUNÇÕES RECURSIVAS

FUNÇÕES RECURSIVAS

FUNÇÕES RECURSIVAS:

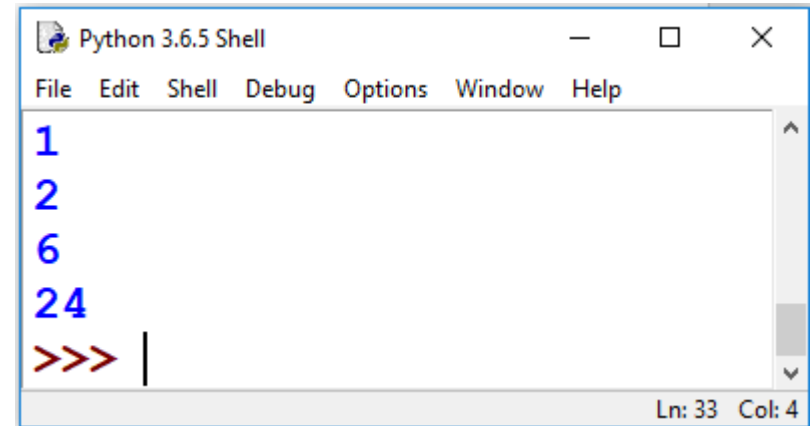
- Uma função recursiva é aquela que chama a si mesma.
- Exemplo: Função recursiva para solução do numero fatorial:



```
heltai.py - C:/Users/Vinicius/Desktop/heltai.py (3.6.5)
File Edit Format Run Options Window Help

def fatorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n*fatorial(n-1)

print(fatorial(1))
print(fatorial(2))
print(fatorial(3))
print(fatorial(4))
|
Ln: 10 Col: 18
```



```
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help

1
2
6
24
>>> |
Ln: 33 Col: 4
```


FUNÇÕES RECURSIVAS

NUMERO FATORIAL COM FUNÇÃO RECURSIVA:

- Exemplo: Calcular o numero fatorial, com rastreamento, alterando o programa anterior:

```
heltai.py - C:/Users/Vinicius/Desktop/heltai.py (3.6.5)
File Edit Format Run Options Window Help

def fatorial(n):
    print(f"Calculo do fatorial de {n}")
    if n == 0 or n == 1:
        print(f"Fatorial de {n} = 1")
        return 1
    else:
        return n*fatorial(n-1)
        print(f"Fatorial de {n} = {fat}")

print(fatorial(1))
print(fatorial(4))

Ln: 11 Col: 0
```

```
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help

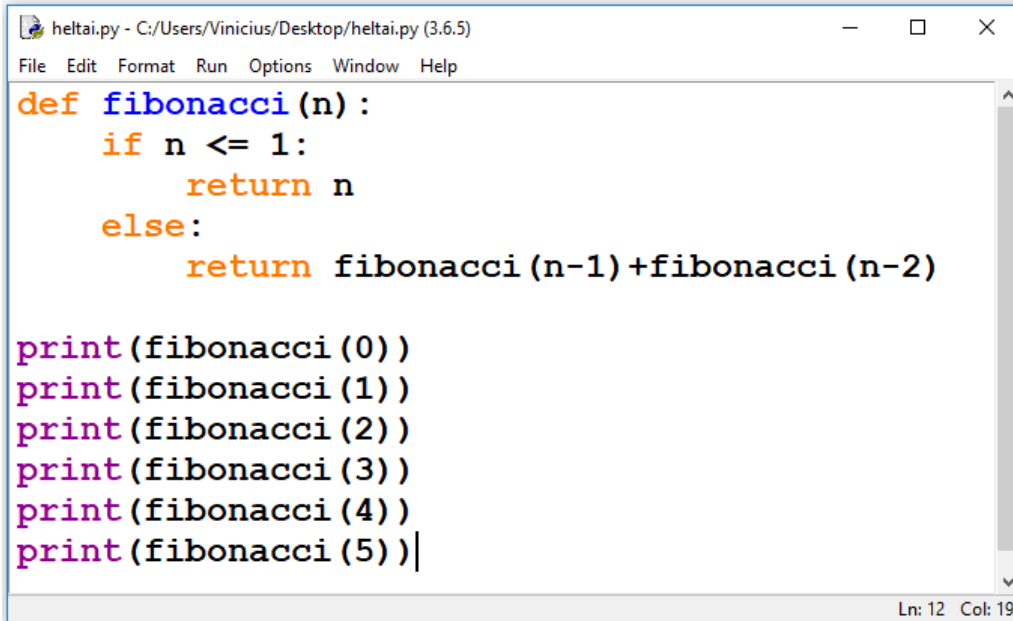
Calculo do fatorial de 1
Fatorial de 1 = 1
1
Calculo do fatorial de 4
Calculo do fatorial de 3
Calculo do fatorial de 2
Calculo do fatorial de 1
Fatorial de 1 = 1
24
>>> |

Ln: 44 Col: 4
```

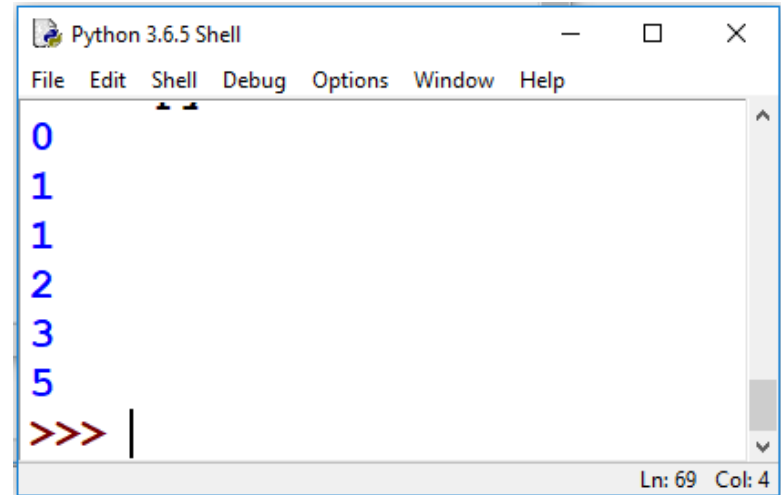
FUNÇÕES RECURSIVAS

NUMERO DE FIBONACCI:

- A sequencia de Fibonacci, esta presente nos principais problemas da humanidade. A sequencia é dada com dois numero 0 e 1. Os números seguintes são as soma dos dois anteriores: 0, 1, 1, 2, 3, 5, 8, 12, 21, ...

A screenshot of a Python IDE window titled 'heltai.py - C:/Users/Vinicius/Desktop/heltai.py (3.6.5)'. The window contains a Python script for a recursive Fibonacci function. The code is as follows:

```
def fibonacci(n):  
    if n <= 1:  
        return n  
    else:  
        return fibonacci(n-1)+fibonacci(n-2)  
  
print(fibonacci(0))  
print(fibonacci(1))  
print(fibonacci(2))  
print(fibonacci(3))  
print(fibonacci(4))  
print(fibonacci(5))
```

The status bar at the bottom right shows 'Ln: 12 Col: 19'.A screenshot of a Python 3.6.5 Shell window. The window displays the output of the Fibonacci function for values 0 through 5. The output is:

```
0  
1  
1  
2  
3  
5  
>>> |
```

The status bar at the bottom right shows 'Ln: 69 Col: 4'.

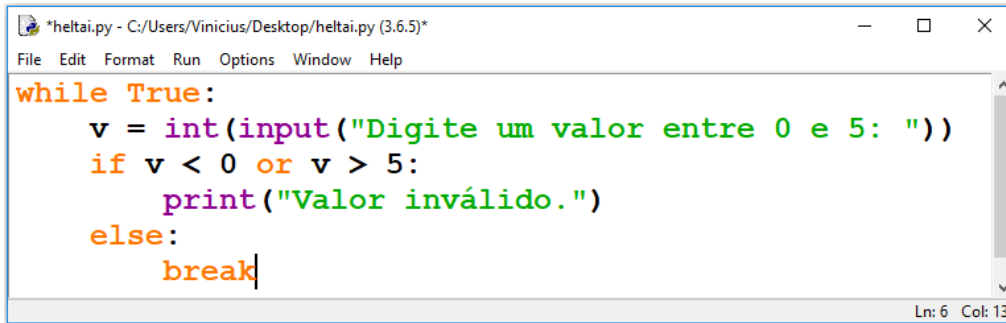
The background of the slide is a blue-tinted image of a complex electronic circuit board. The top half shows a close-up of various components like chips and capacitors. The bottom half shows a more detailed view of the circuit traces and components. In the center, the word "VALIDAÇÃO" is written in a large, white, sans-serif font.

VALIDAÇÃO

VALIDAÇÃO

VALIDAÇÃO:

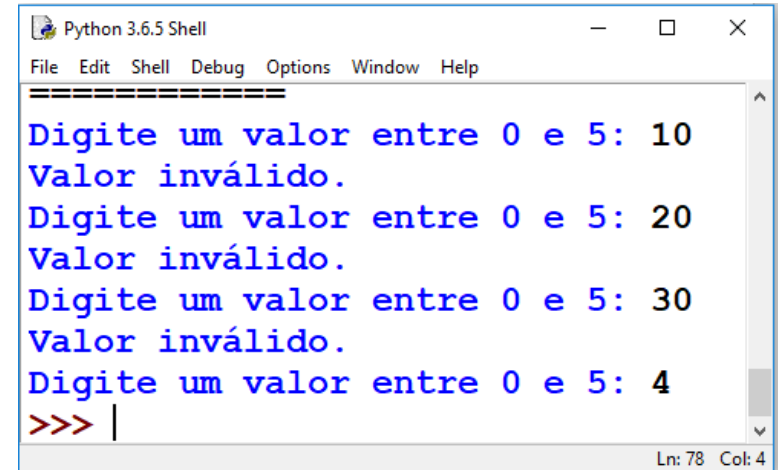
- Para validar dados de entrada, é muito útil a utilização de funções.
- No exemplo abaixo, lê um valor inteiro, limitado por um valor de mínimo e máximo. Esse trecho repete a entrada de dados até se obter um valor válido



```
*heltai.py - C:/Users/Vinicius/Desktop/heltai.py (3.6.5)*
File Edit Format Run Options Window Help

while True:
    v = int(input("Digite um valor entre 0 e 5: "))
    if v < 0 or v > 5:
        print("Valor inválido.")
    else:
        break

Ln: 6 Col: 13
```



```
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help

=====
Digite um valor entre 0 e 5: 10
Valor inválido.
Digite um valor entre 0 e 5: 20
Valor inválido.
Digite um valor entre 0 e 5: 30
Valor inválido.
Digite um valor entre 0 e 5: 4
>>> |

Ln: 78 Col: 4
```

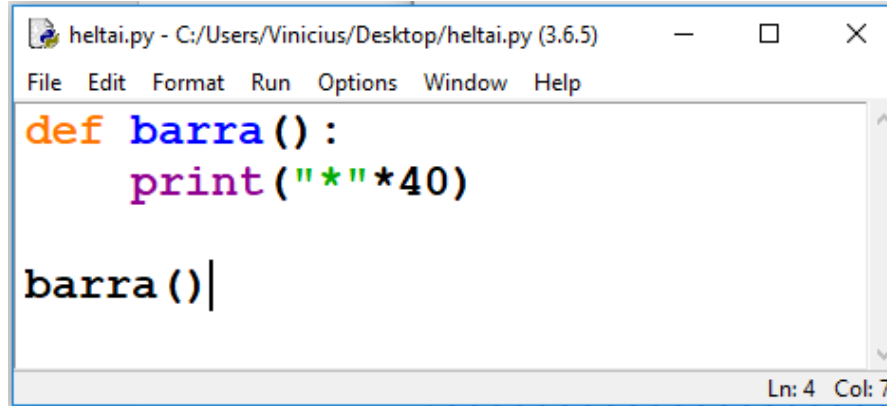


PARÂMETROS OPCIONAIS

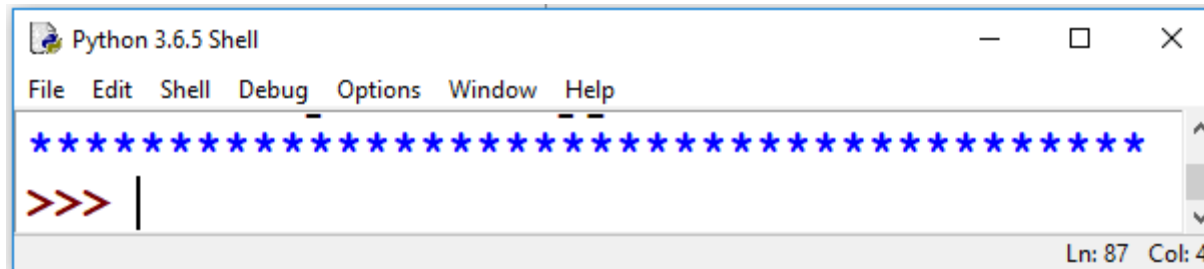
PARÂMETROS OPCIONAIS

PARÂMETROS OPCIONAIS

- Nem sempre precisamos passar todos os parâmetros para uma função, preferindo utilizar um valor previamente escolhido como padrão, mas deixando a possibilidade de alterá-lo, caso necessário.
- No exemplo abaixo, imprime uma barra na tela:



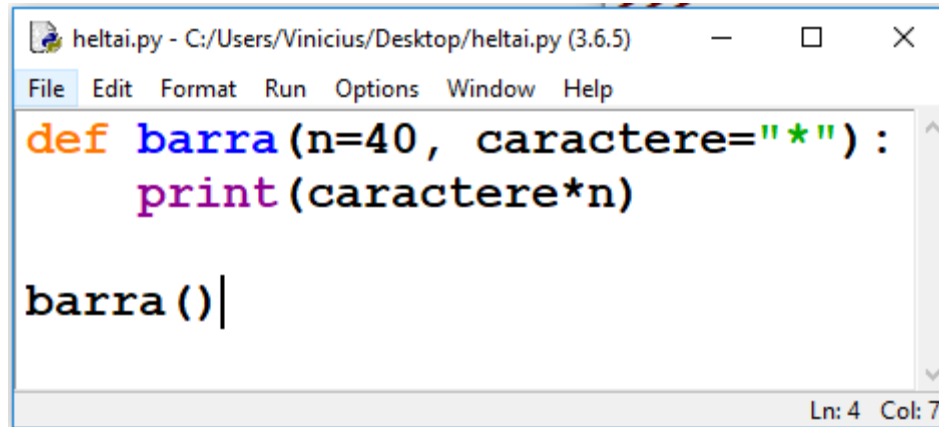
```
def barra():  
    print("★"*40)  
  
barra()
```



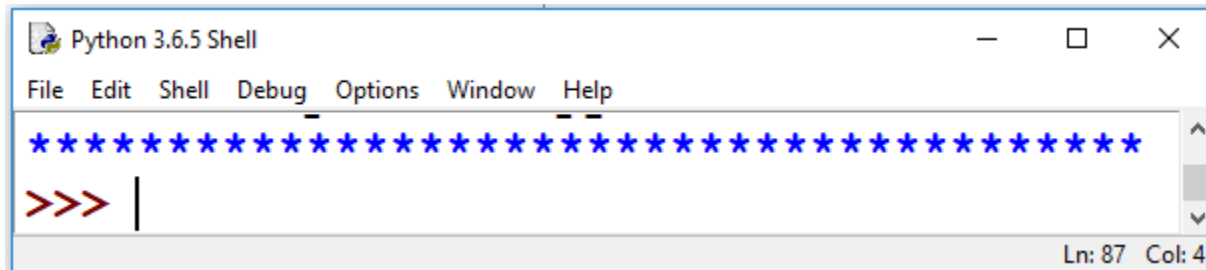
```
★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★  
>>> |
```

PARÂMETROS OPCIONAIS

- No exemplo anterior, a função barra não recebe algum parâmetro e pode ser chamada com barra(). Porém, tanto o asterisco (*) quanto a quantidade de caracteres a exibir podem precisar ser alterados. Para resolver esse problema, pode-se utilizar parâmetros opcionais:



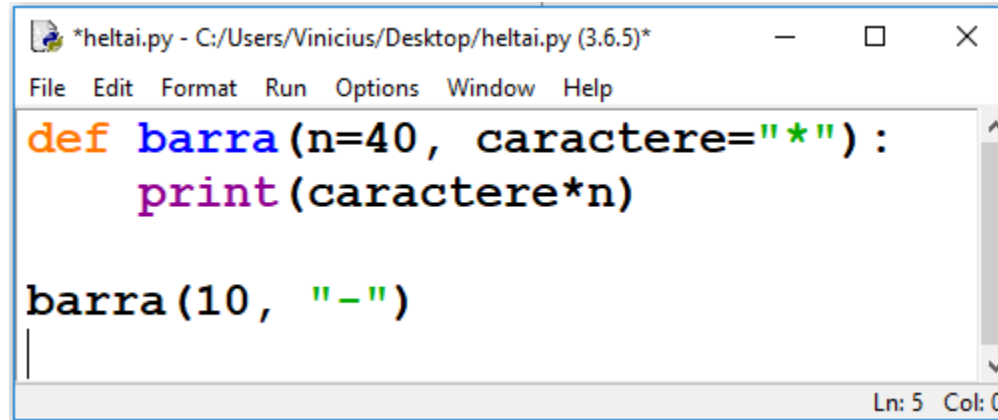
```
def barra(n=40, caractere="*"):  
    print(caractere*n)  
  
barra()
```



```
>>> barra()  
*****  
>>> 
```

PARÂMETROS OPCIONAIS

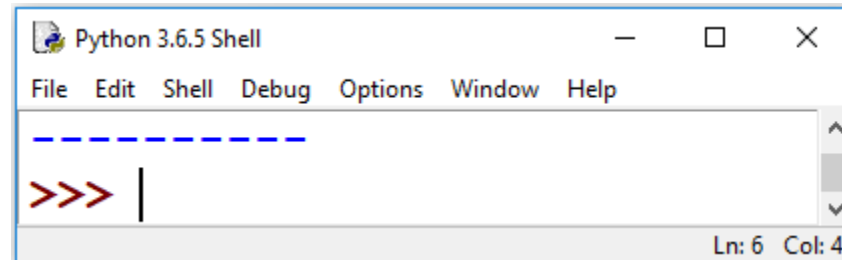
- No mesmo exemplo, seguindo com o parâmetro opcional, é possível alterar os parâmetros na “chamada” da função. Como 10 caracteres “-” ou invés de 40 caracteres “*”



The screenshot shows a window titled "*heltai.py - C:/Users/Vinicius/Desktop/heltai.py (3.6.5)*". The menu bar includes File, Edit, Format, Run, Options, Window, and Help. The code editor contains the following Python code:

```
def barra(n=40, caractere="*") :  
    print(caractere*n)  
  
barra(10, "-")
```

The status bar at the bottom right indicates "Ln: 5 Col: 0".

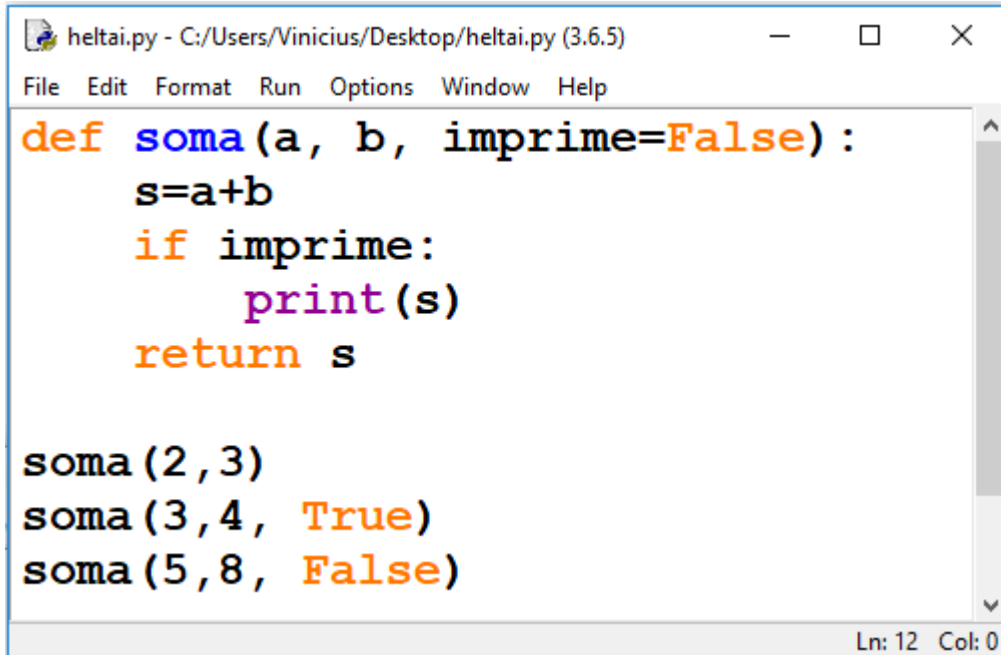


The screenshot shows a window titled "Python 3.6.5 Shell". The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The shell displays a dashed line followed by the prompt ">>>" and a vertical cursor, indicating that the code from the previous window has been executed.

The status bar at the bottom right indicates "Ln: 6 Col: 4".

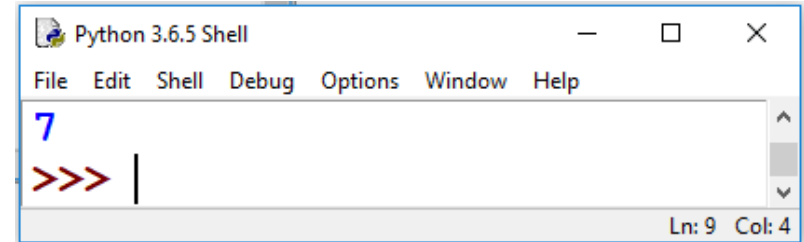
PARÂMETROS OPCIONAIS

- Outro exemplo, combinando com parâmetro opcional com parâmetros obrigatórios:



```
def soma(a, b, imprime=False):  
    s=a+b  
    if imprime:  
        print(s)  
    return s  
  
soma(2,3)  
soma(3,4, True)  
soma(5,8, False)
```

Ln: 12 Col: 0



```
Python 3.6.5 Shell  
File Edit Shell Debug Options Window Help  
7  
>>> |
```

Ln: 9 Col: 4

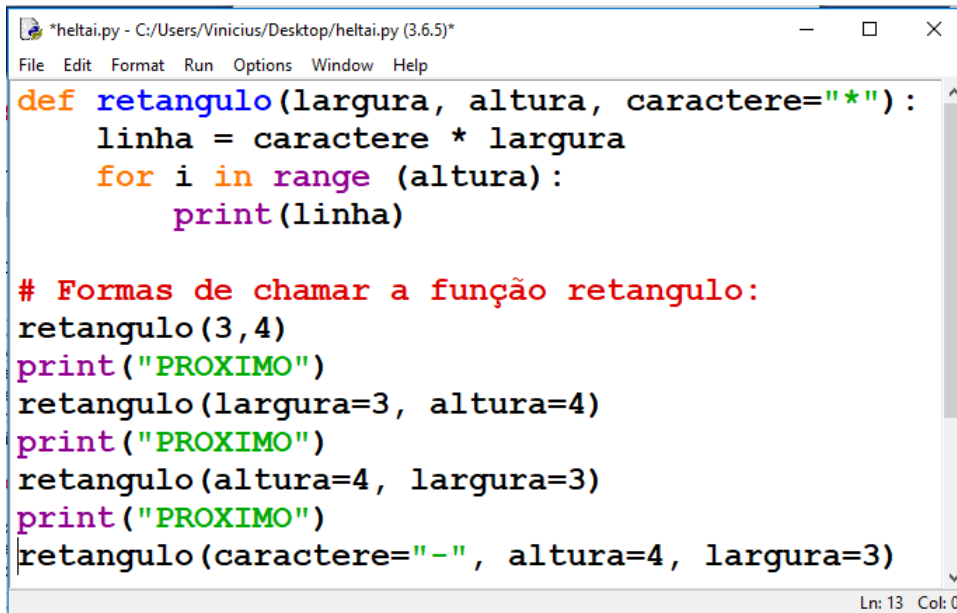


NOMEANDO PARÂMETROS

NOMEANDO PARÂMETROS

NOMEANDO PARÂMETROS:

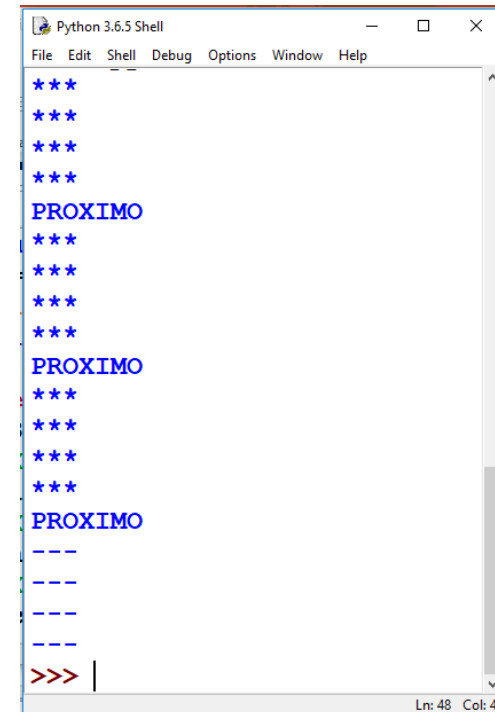
- Python suporta a chamada de funções com vários parâmetros seguindo a ordem no qual são passados (esquerda para direita, como visto até então).
- Quando especifica-se o nome dos parâmetros, pode-se passar o parâmetro em qualquer ordem:



```
*heltai.py - C:/Users/Vinicius/Desktop/heltai.py (3.6.5)*
File Edit Format Run Options Window Help

def retangulo(largura, altura, caractere="*"):
    linha = caractere * largura
    for i in range (altura):
        print(linha)

# Formas de chamar a função retangulo:
retangulo(3,4)
print("PROXIMO")
retangulo(largura=3, altura=4)
print("PROXIMO")
retangulo(altura=4, largura=3)
print("PROXIMO")
retangulo(caractere="-", altura=4, largura=3)
```



```
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help

***
***
***
***
PROXIMO
***
***
***
PROXIMO
***
***
***
PROXIMO
---
---
---
---
>>> |
```

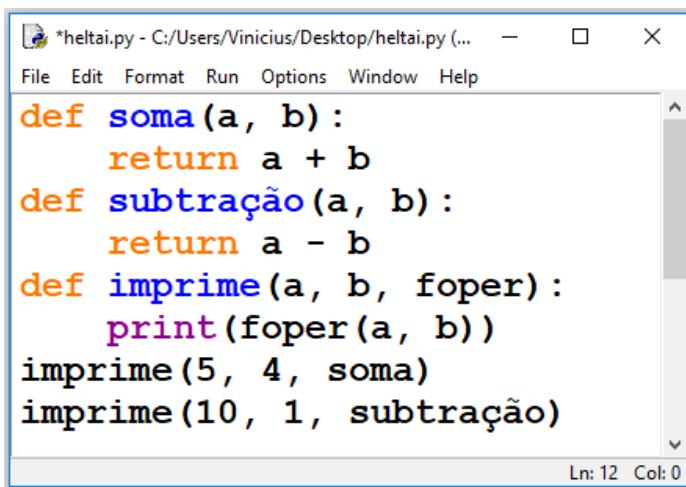


FUNÇÕES COMO PARÂMETROS

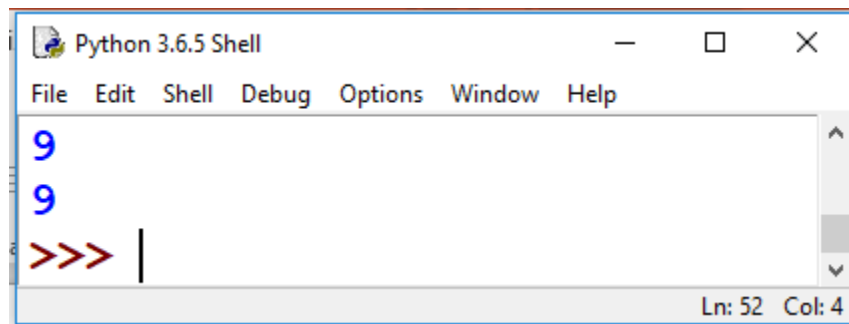
FUNÇÕES COMO PARÂMETROS

FUNÇÕES COMO PARÂMETROS:

- Python permite a passagem de funções como parâmetro. Isso permite combinar várias funções para realizar uma tarefa:



```
def soma(a, b):  
    return a + b  
def subtração(a, b):  
    return a - b  
def imprime(a, b, foper):  
    print(foper(a, b))  
imprime(5, 4, soma)  
imprime(10, 1, subtração)
```



```
9  
9  
>>> |
```

- As funções soma e subtração foram definidas normalmente, mas a função imprime recebe um parâmetro chamado foper que é uma função que está passando como função.
- Passar funções como parâmetro permite injetar funcionalidades dentro de outras funções, tornando-as configuráveis e mais genéricas.

FUNÇÕES COMO PARÂMETROS

- Exemplo: A função **imprime_lista** recebe uma lista e duas funções como parâmetro. A função **fimprime** é utilizada para realizar a impressão de cada elemento, mas só é chamada se o resultado da função passada como **fcondição** for verdadeira.

```
heltai.py - C:/Users/Vinicius/Desktop/heltai.py (3.6.5)
File Edit Format Run Options Window Help

def imprime_lista(L, fimpressao, fcondicao):
    for e in L:
        if fcondicao(e):
            fimpressao(e)

def imprime_elemento(e):
    print(f"Valor: {e}")

def épar(x):
    return x%2 == 0

def éimpar(x):
    return not épar(x)

L = [1, 7, 9, 2, 11, 0]

imprime_lista(L, imprime_elemento, épar)
imprime_lista(L, imprime_elemento, éimpar)
```

Ln: 28 Col: 0

```
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help

Valor: 2
Valor: 0
Valor: 1
Valor: 7
Valor: 9
Valor: 11
>>> |
```

Ln: 68 Col: 4

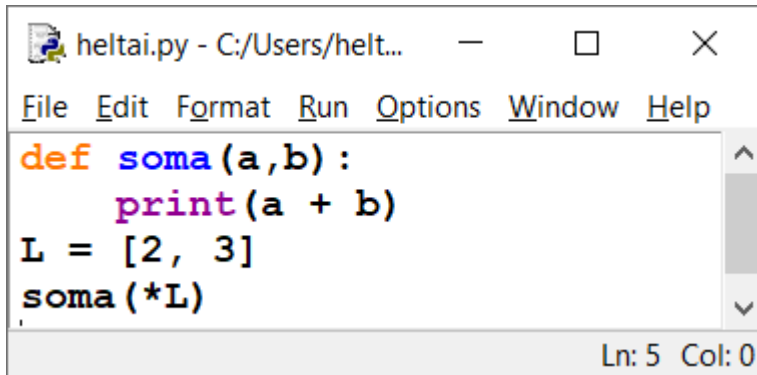


EMPACOTAMENTO E DESEMPACOTAMENTO DE PARÂMETROS

EMPACOTAMENTO E DESEMPACOTAMENTO

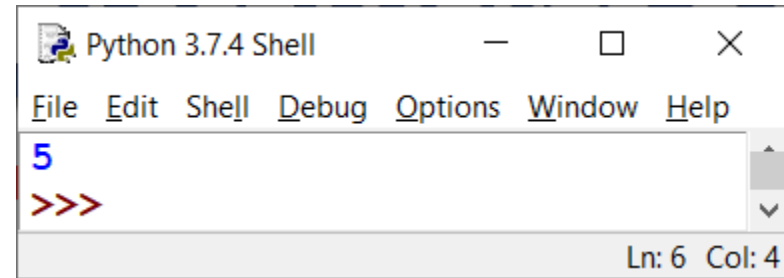
EMPACOTAMENTO E DESEMPACOTAMENTO DE PARÂMETROS:

- Uma das flexibilidades da linguagem Python é passar parâmetros empacotados em uma lista:



```
def soma(a,b):  
    print(a + b)  
L = [2, 3]  
soma(*L)
```

Ln: 5 Col: 0



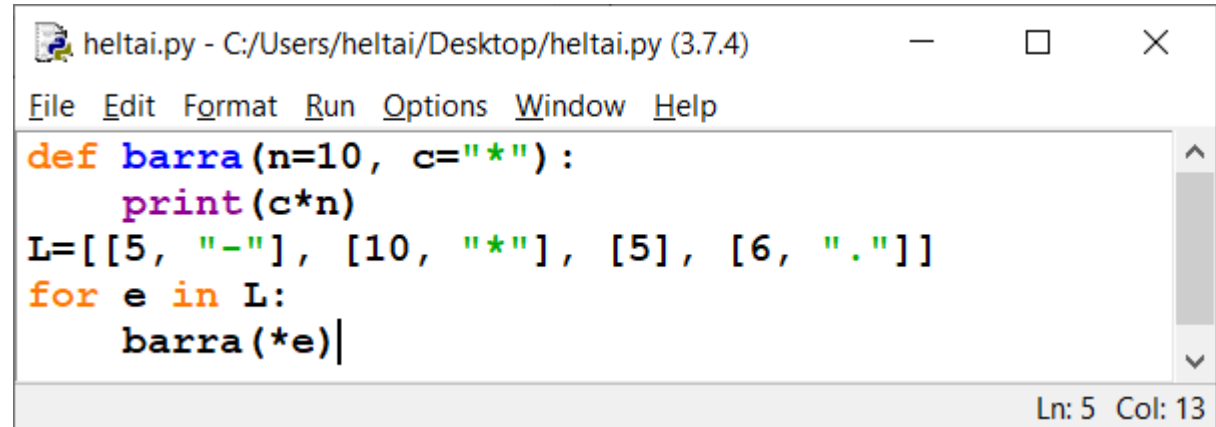
```
5  
>>>
```

Ln: 6 Col: 4

- Em soma(*L) é utilizado o * para indicar que queremos desempacotar a lista L utilizando seus valores como parâmetros para a função soma. Desta forma, L[0] é atribuído a variável (a) e L[1] a variável (b).
- Esse recurso permite armazenar os parâmetros em listas e evitar construção do tipo soma(L[0], L[1]).

EMPACOTAMENTO E DESEMPACOTAMENTO

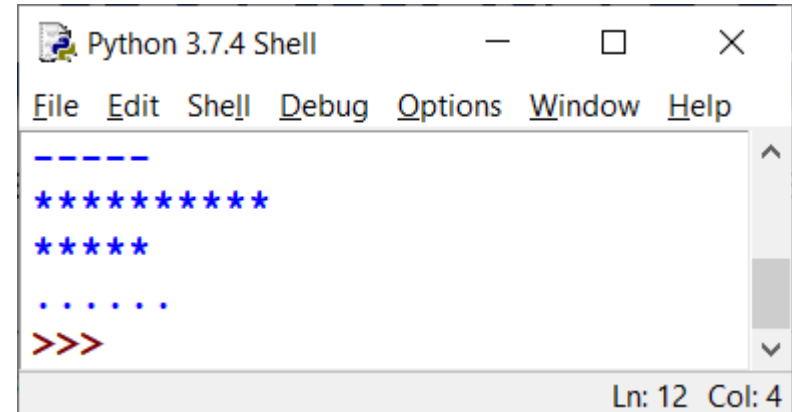
- No exemplo abaixo, em que utiliza uma lista de listas para realizar várias chamadas a uma função dentro de um **for**:



```
def barra(n=10, c="*") :  
    print(c*n)  
L=[[5, "-"], [10, "*"], [5], [6, "."]]  
for e in L:  
    barra(*e)|
```

Ln: 5 Col: 13

- Mesmo usando o empacotamento de parâmetros, recursos como parâmetros opcionais ainda são possíveis quando **e** contém apenas um elemento (L[1])



```
-----  
*****  
*****  
.....  
>>>
```

Ln: 12 Col: 4

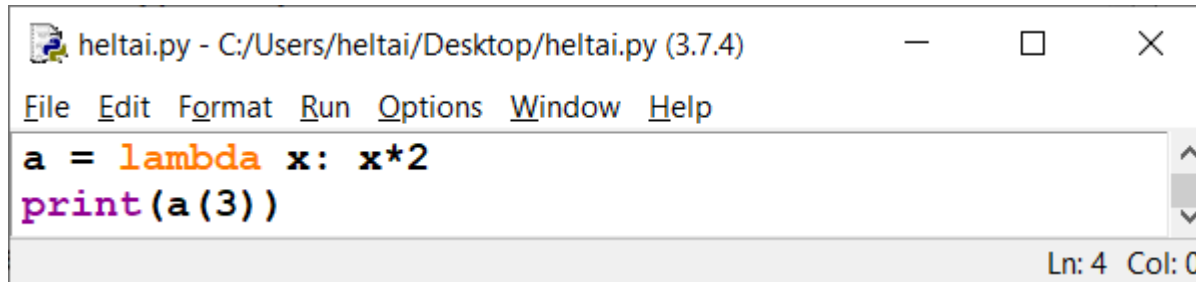


FUNÇÃO LAMBDA

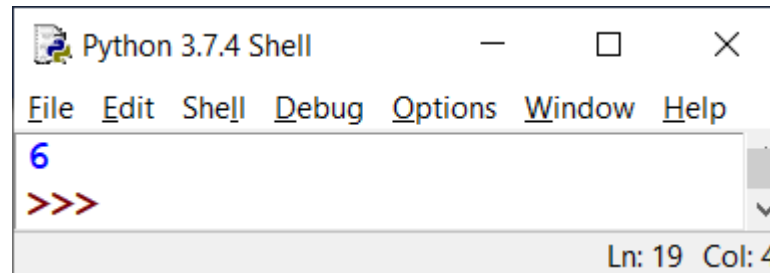
FUNÇÃO LAMBDA

FUNÇÃO LAMBDA:

- Função lambda é uma função simples, sem nome.



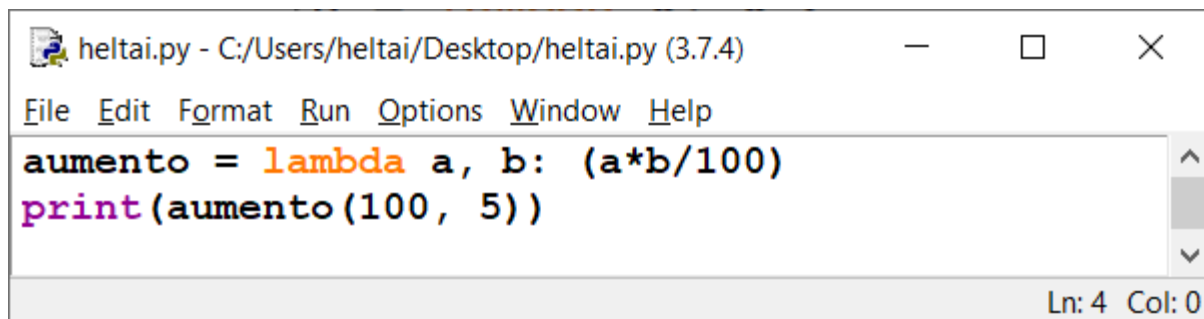
```
heltai.py - C:/Users/heltai/Desktop/heltai.py (3.7.4)  
File Edit Format Run Options Window Help  
a = lambda x: x*2  
print(a(3))  
Ln: 4 Col: 0
```



```
Python 3.7.4 Shell  
File Edit Shell Debug Options Window Help  
6  
>>>  
Ln: 19 Col: 4
```

FUNÇÃO LAMBDA

- Funções lambda também podem receber mais de um parâmetro:

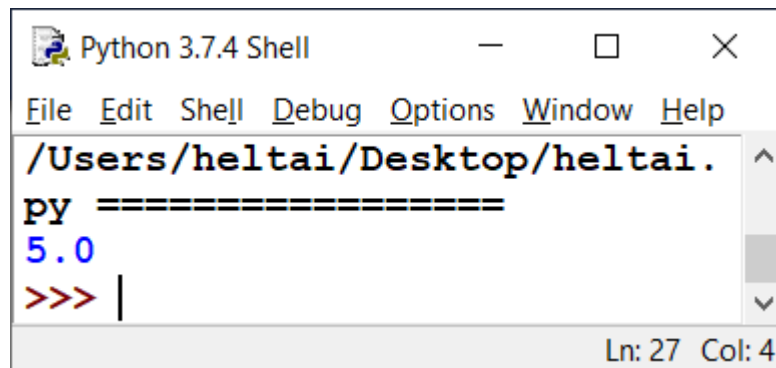


heltai.py - C:/Users/heltai/Desktop/heltai.py (3.7.4)

File Edit Format Run Options Window Help

```
aumento = lambda a, b: (a*b/100)
print(aumento(100, 5))
```

Ln: 4 Col: 0



Python 3.7.4 Shell

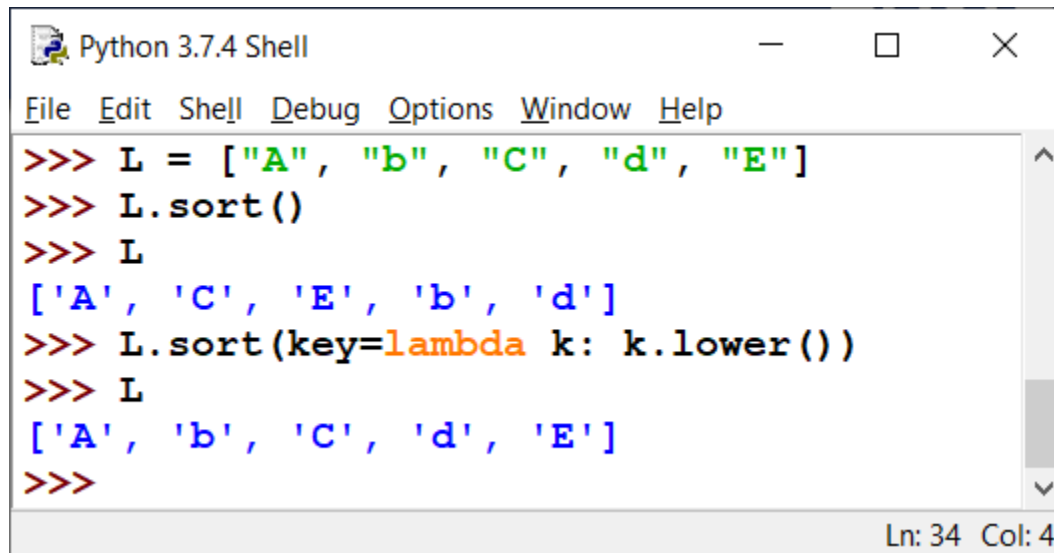
File Edit Shell Debug Options Window Help

```
/Users/heltai/Desktop/heltai.py =====
5.0
>>> |
```

Ln: 27 Col: 4

FUNÇÃO LAMBDA

- Funções lambda são utilizadas quando o código da função é muito simples ou utilizado poucas vezes. Algumas funções da biblioteca padrão do Python também permitem que funções sejam passadas como parâmetros, como o caso do método **sort** de lista, que recebe um parâmetro **key** que recebe o elemento e deve retornar a chave a utilizar para a ordenação:



```
Python 3.7.4 Shell
File Edit Shell Debug Options Window Help
>>> L = ["A", "b", "C", "d", "E"]
>>> L.sort()
>>> L
['A', 'C', 'E', 'b', 'd']
>>> L.sort(key=lambda k: k.lower())
>>> L
['A', 'b', 'C', 'd', 'E']
>>>
```

Ln: 34 Col: 4

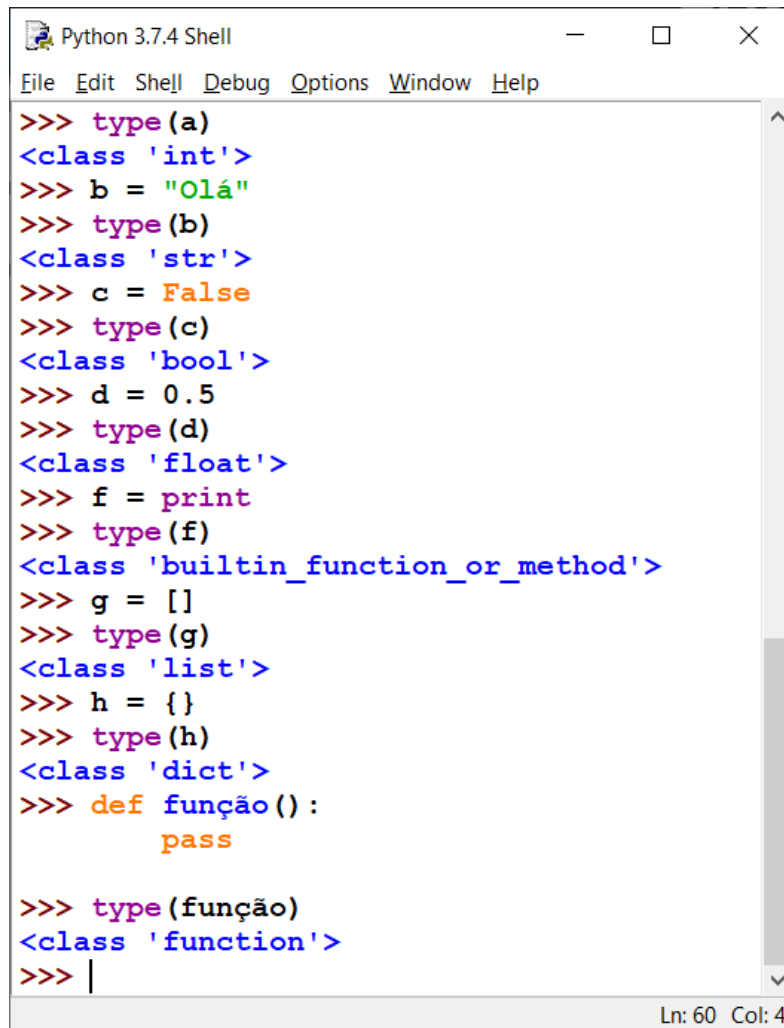


FUNÇÃO TYPE

FUNÇÃO TYPE

FUNÇÃO TYPE:

- A função type retorna o tipo de uma variável, função ou objeto em Python.



```
Python 3.7.4 Shell
File Edit Shell Debug Options Window Help

>>> type(a)
<class 'int'>
>>> b = "Olá"
>>> type(b)
<class 'str'>
>>> c = False
>>> type(c)
<class 'bool'>
>>> d = 0.5
>>> type(d)
<class 'float'>
>>> f = print
>>> type(f)
<class 'builtin_function_or_method'>
>>> g = []
>>> type(g)
<class 'list'>
>>> h = {}
>>> type(h)
<class 'dict'>
>>> def função():
    pass

>>> type(função)
<class 'function'>
>>> |
```

Ln: 60 Col: 4



PROF. VINICIUS HELTAI

vinicius.pacheco@docente.unip.br

www.heltai.com.br

(11) 98200-3932



Obrigado!

