

DICIONARIOS E TUPLAS



DICIONARIO

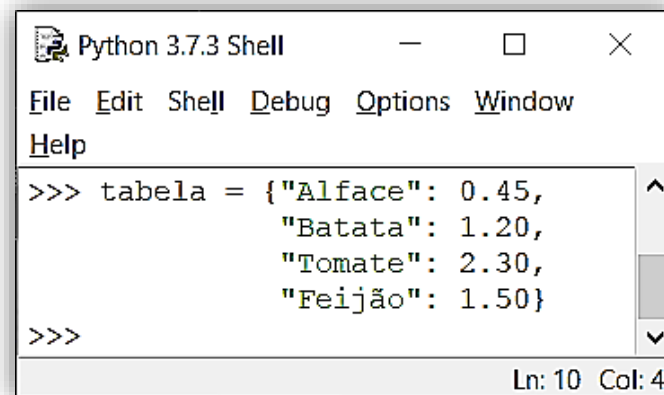
DICIONARIOS

O QUE SÃO OS DICIONARIOS EM PYTHON ?

- Dicionários consistem em uma estrutura de dados similares às listas, mas com propriedades de acesso diferentes.
- Um dicionário é composto de um conjunto de chaves e valores.
- O dicionário em si consiste em relacionar uma chave a um valor específico.
- Em Python, cria-se dicionário utilizando chaves `{}`. Cada elemento do dicionário é uma combinação de chaves e valores.

EXEMPLO: Preços de mercadorias de uma tabela.

PRODUTO	PREÇO
Alface	R\$ 0,45
Batata	R\$ 1,20
Tomate	R\$ 2,30
Feijão	R\$ 1,50



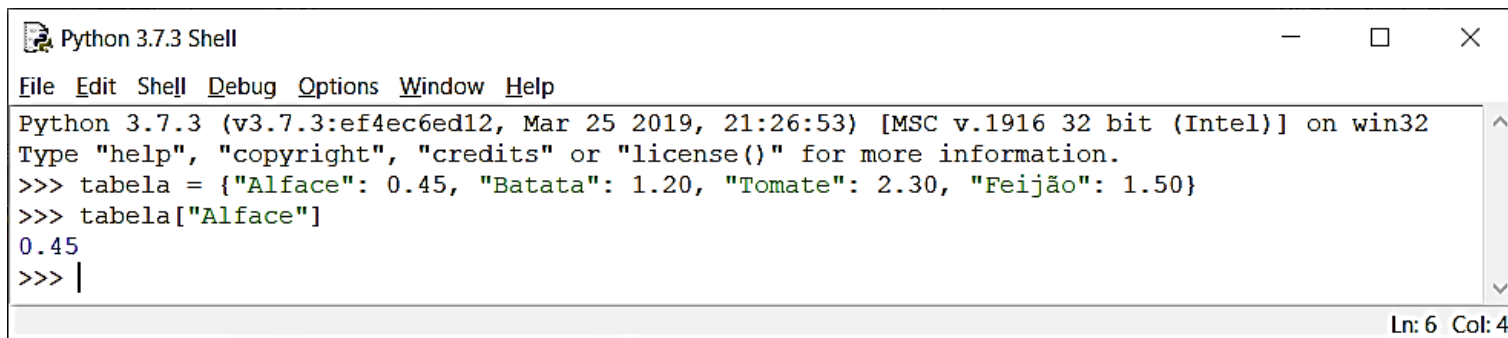
```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
>>> tabela = {"Alface": 0.45,
               "Batata": 1.20,
               "Tomate": 2.30,
               "Feijão": 1.50}
>>>
```

Ln: 10 Col: 4

DICIONARIOS

ACESSO AOS DADOS DE UM DICIONARIO:

- Um dicionário é acessado por meio das chaves.
- No exemplo anterior, para acessar um parâmetro (preço do item alface) pode ser feito da seguinte forma:

A screenshot of a Python 3.7.3 Shell window. The title bar says "Python 3.7.3 Shell". The menu bar includes "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The main text area shows the following code and output:

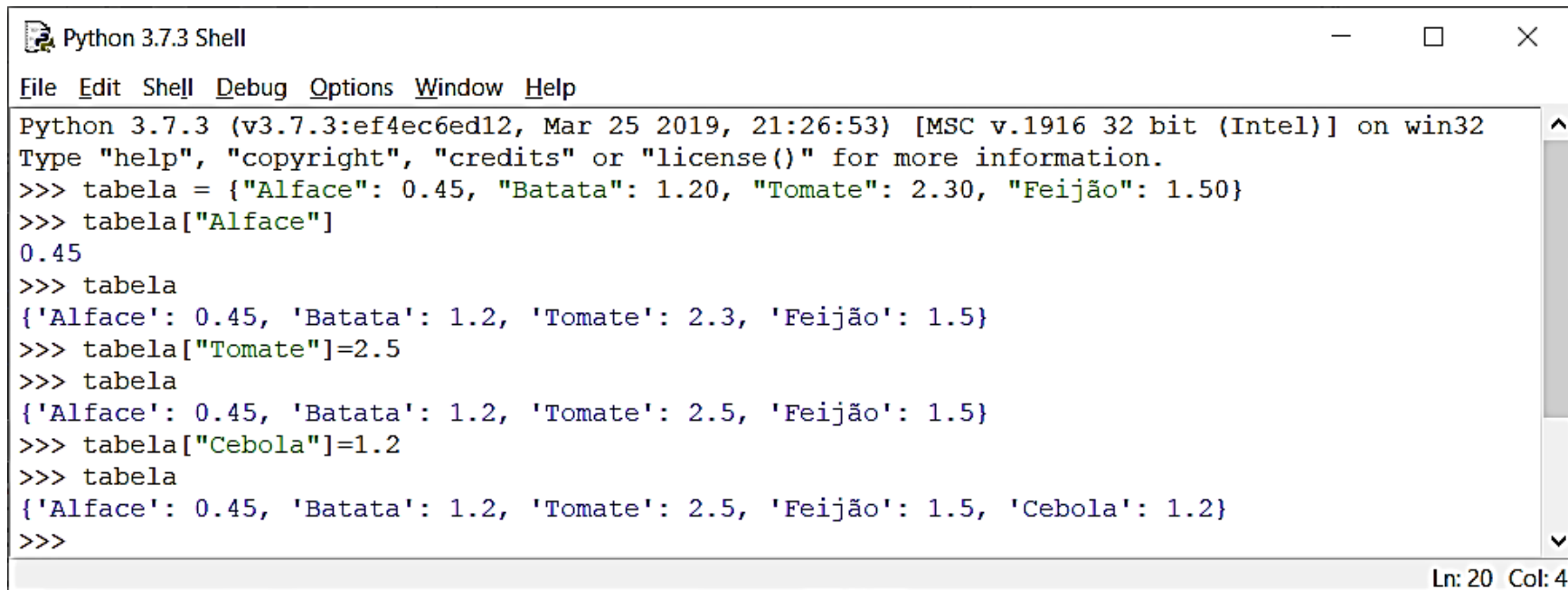
```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> tabela = {"Alface": 0.45, "Batata": 1.20, "Tomate": 2.30, "Feijão": 1.50}
>>> tabela["Alface"]
0.45
>>> |
```

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

- Diferente da listas, onde o índice é um numero, no caso do dicionário é utilizado a “chave” como índice. Esta forma, quando se atribui um valor a uma chave, duas possibilidades podem ocorrer:
 - Se a chave já existe – O valor associado é alterado para o novo valor.
 - Se a chave não existe – A nova chave será adicionada ao dicionário

DICIONARIOS

- Seguindo o mesmo exemplo:

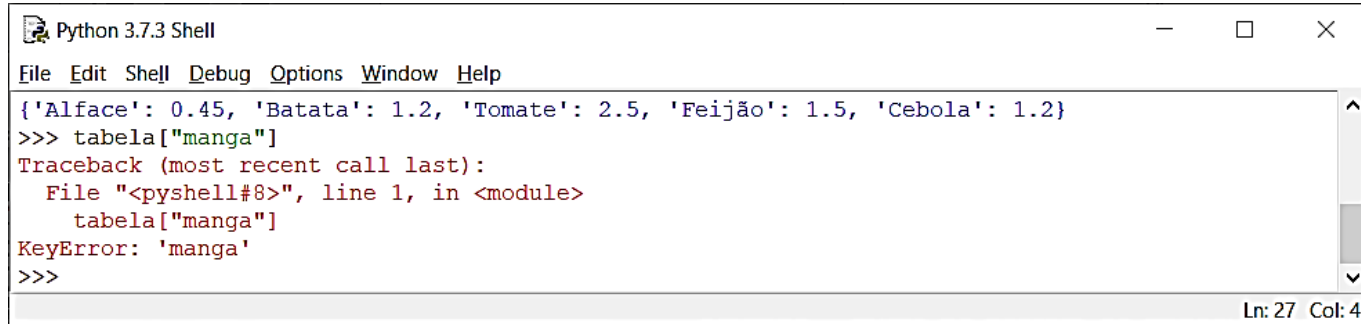
A screenshot of a Python 3.7.3 Shell window. The window has a title bar with a Python logo, the text "Python 3.7.3 Shell", and standard window controls (minimize, maximize, close). Below the title bar is a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The main area contains a Python prompt and several lines of code and output. The code creates a dictionary named 'tabela' with four items: 'Alface' (0.45), 'Batata' (1.20), 'Tomate' (2.30), and 'Feijão' (1.50). It then accesses 'tabela["Alface"]' to get 0.45, prints the entire dictionary, updates 'tabela["Tomate"]' to 2.5, prints the dictionary again, adds 'tabela["Cebola"]' with value 1.2, prints the dictionary a third time, and finally prints an empty prompt. A status bar at the bottom right shows "Ln: 20 Col: 4".

```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> tabela = {"Alface": 0.45, "Batata": 1.20, "Tomate": 2.30, "Feijão": 1.50}
>>> tabela["Alface"]
0.45
>>> tabela
{'Alface': 0.45, 'Batata': 1.2, 'Tomate': 2.3, 'Feijão': 1.5}
>>> tabela["Tomate"]=2.5
>>> tabela
{'Alface': 0.45, 'Batata': 1.2, 'Tomate': 2.5, 'Feijão': 1.5}
>>> tabela["Cebola"]=1.2
>>> tabela
{'Alface': 0.45, 'Batata': 1.2, 'Tomate': 2.5, 'Feijão': 1.5, 'Cebola': 1.2}
>>>
```

Ln: 20 Col: 4

DICIONARIOS

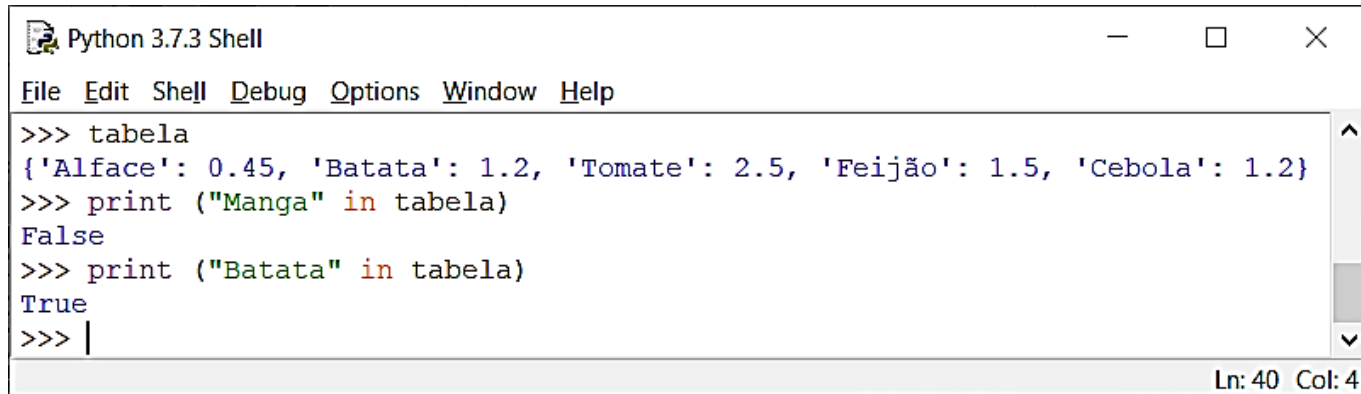
- **CUIDADO** – Antes de acessar os dados, há necessidade de verificar se uma chave já existe, antes de acessá-la:



```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
{'Alface': 0.45, 'Batata': 1.2, 'Tomate': 2.5, 'Feijão': 1.5, 'Cebola': 1.2}
>>> tabela["manga"]
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    tabela["manga"]
KeyError: 'manga'
>>>
```

Ln: 27 Col: 4

- Caso a chave não exista, uma exceção do tipo **KeyError** é ativada (conforme visto acima).
- Para verificar se uma chave pertence ao dicionário, pode ser usado o operador **in**:



```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
>>> tabela
{'Alface': 0.45, 'Batata': 1.2, 'Tomate': 2.5, 'Feijão': 1.5, 'Cebola': 1.2}
>>> print ("Manga" in tabela)
False
>>> print ("Batata" in tabela)
True
>>> |
```

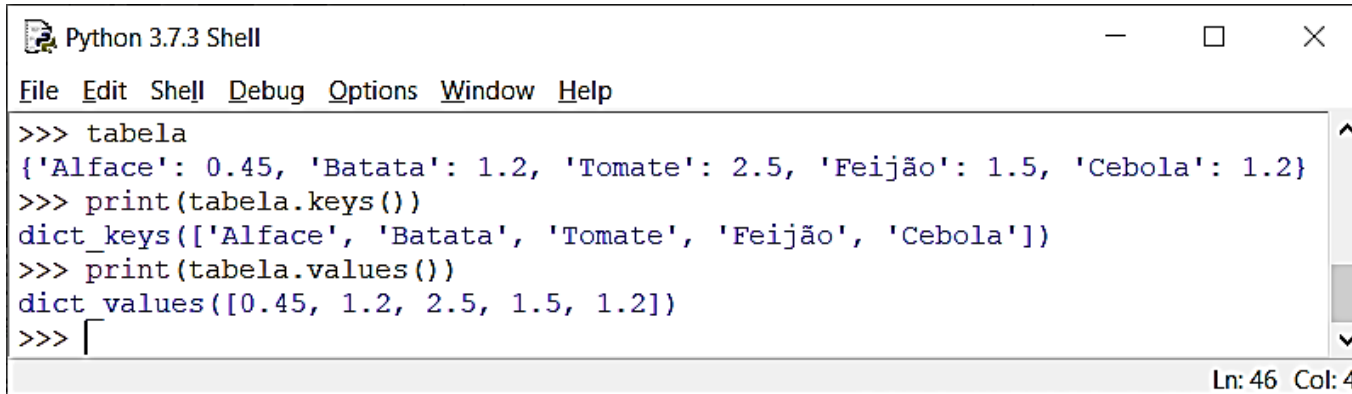
Ln: 40 Col: 4

DICIONARIOS

- Outro método é obter uma lista com as chaves do dicionários, ou mesmo uma lista dos valores associados:

<dicionário>.keys() – Retorna as chaves que pertencem aos dicionários

<dicionário>.values() – Retorna os valores que pertencem aos dicionários.

A screenshot of a Python 3.7.3 Shell window. The window has a title bar with the text 'Python 3.7.3 Shell' and standard window controls (minimize, maximize, close). Below the title bar is a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main area of the window contains a Python REPL session. The user has defined a dictionary named 'tabela' with five items: 'Alface': 0.45, 'Batata': 1.2, 'Tomate': 2.5, 'Feijão': 1.5, and 'Cebola': 1.2. They then use 'print(tabela.keys())' to display the keys as a list: ['Alface', 'Batata', 'Tomate', 'Feijão', 'Cebola']. Next, they use 'print(tabela.values())' to display the values as a list: [0.45, 1.2, 2.5, 1.5, 1.2]. The prompt '>>>' is visible at the end of the last line. A status bar at the bottom right shows 'Ln: 46 Col: 4'.

```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
>>> tabela
{'Alface': 0.45, 'Batata': 1.2, 'Tomate': 2.5, 'Feijão': 1.5, 'Cebola': 1.2}
>>> print(tabela.keys())
dict_keys(['Alface', 'Batata', 'Tomate', 'Feijão', 'Cebola'])
>>> print(tabela.values())
dict_values([0.45, 1.2, 2.5, 1.5, 1.2])
>>> |
```

Ln: 46 Col: 4

- Os métodos **keys()** e **values()** retornam geradores.
- É possível utilizar dentro de um comando de repetição **for** ou transformá-lo em lista usando a função **list**.

DICIONARIOS

- Exemplo – Programa que utiliza dicionários para exibir o preço de um produto:

```
heltai.py - C:/Users/heltai/Desktop/heltai.py (3.7.3)
File Edit Format Run Options Window Help

tabela = {"Alface": 0.45,
          "Batata": 1.20,
          "Tomate": 2.30,
          "Feijão": 1.50}

while True:
    produto = input("Digite o nome do produto, fim para terminar: ")
    if produto == "fim":
        break
    if produto in tabela:
        print(f"Preço {tabela[produto]:5.2f}")
    else:
        print("Produto não encontrado!")
```

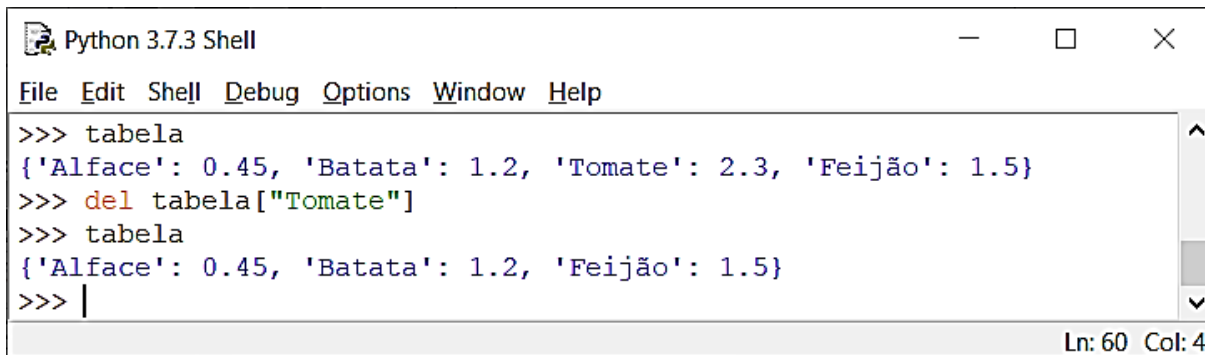
```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help

===== RESTART: C:/Users/heltai/Desktop/heltai.py =====
Digite o nome do produto, fim para terminar: Alface
Preço  0.45
Digite o nome do produto, fim para terminar: Batata
Preço  1.20
Digite o nome do produto, fim para terminar: cerveja
Produto não encontrado!
Digite o nome do produto, fim para terminar: fim
>>> |
```

Ln: 55 Col: 4

DICIONARIOS

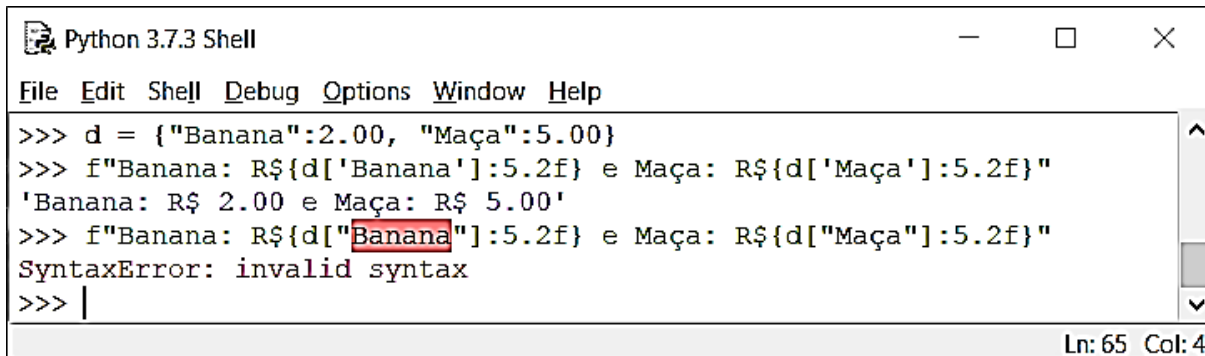
- Para apagar um item do dicionário, utiliza-se o comando **del**:

A screenshot of a Python 3.7.3 Shell window. The window has a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area shows the following code:

```
>>> tabela
{'Alface': 0.45, 'Batata': 1.2, 'Tomate': 2.3, 'Feijão': 1.5}
>>> del tabela["Tomate"]
>>> tabela
{'Alface': 0.45, 'Batata': 1.2, 'Feijão': 1.5}
>>> |
```

The status bar at the bottom right indicates 'Ln: 60 Col: 4'.

- Ao utilizar **f-strings** com dicionário. As aspas duplas (“), são substituídas por aspa simples (‘).

A screenshot of a Python 3.7.3 Shell window. The window has a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area shows the following code:

```
>>> d = {"Banana":2.00, "Maça":5.00}
>>> f"Banana: R${d['Banana']}:5.2f} e Maça: R${d['Maça']}:5.2f}"
'Banana: R$ 2.00 e Maça: R$ 5.00'
>>> f"Banana: R${d["Banana"]}:5.2f} e Maça: R${d["Maça"]}:5.2f}"
SyntaxError: invalid syntax
>>> |
```

The status bar at the bottom right indicates 'Ln: 65 Col: 4'.

DICIONARIOS

TRIVIA – CURIOSIDADE:

- “Dicionários usam uma técnica chamada espalhamento ou hash.
- Essa técnica utiliza um algoritmo que calcula um numero ou hash para a chave que esta sendo procurada.
- O valor é sempre igual se o valor da chave também for igual ao procurar o índice desejado.
- A técnica é mais complexa, pois precisa se lidar com colisões (chaves diferentes que têm o mesmo hash).
- Tais itens serão melhor abordados em estruturas de dados.

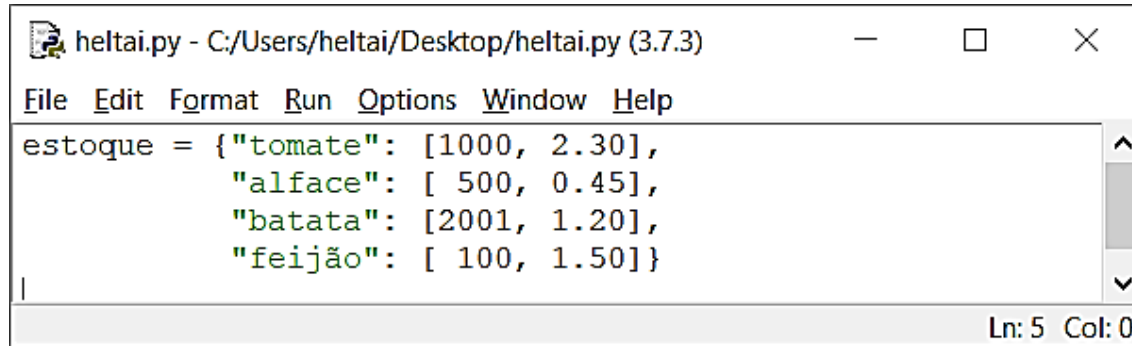


DICIONARIO COM LISTAS

DICIONARIOS COM LISTAS

DICIONARIOS COM LISTAS

- Em Python é possível ter um dicionário nos quais as chaves estão associadas a listas ou mesmo a outros dicionários.
- Isso pode ser exemplificado, através de uma relação de estoque de mercadorias na qual se tem além de preços a quantidade no estoque:



A screenshot of a Python IDE window titled "heltai.py - C:/Users/heltai/Desktop/heltai.py (3.7.3)". The window contains a Python dictionary named "estoque" with four items: "tomate" with values [1000, 2.30], "alface" with values [500, 0.45], "batata" with values [2001, 1.20], and "feijão" with values [100, 1.50]. The status bar at the bottom right indicates "Ln: 5 Col: 0".

```
estoque = {"tomate": [1000, 2.30],  
          "alface": [ 500, 0.45],  
          "batata": [2001, 1.20],  
          "feijão": [ 100, 1.50]}
```

- Neste caso, o nome do produto é a chave, e a lista consiste nos valores associados, uma lista por chave.
- O primeiro elemento da lista é a quantidade, o segundo o preço do produto.

DICIONARIOS COM LISTAS

- **EXEMPLO:** Programa que processa uma lista de operações e calcula o preço total da venda, atualizando também a quantidade em estoque:

```
heltai.py - C:/Users/heltai/Desktop/heltai.py (3.7.3)
File Edit Format Run Options Window Help

estoque = {"tomate": [1000, 2.30],
           "alface": [ 500, 0.45],
           "batata": [2001, 1.20],
           "feijão": [ 100, 1.50]}

venda = [{"tomate", 5}, {"batata", 10}, {"alface", 5}]
total = 0

print("Vendas: \n")

for operação in venda:
    produto, quantidade = operação
    preço = estoque[produto][1]
    custo = preço*quantidade
    print(f"{produto:12s}: {quantidade:3d} x {preço:6.2f} = {custo:6.2f}")
    estoque[produto][0] -= quantidade
    total += custo

print(f"Custo Total: {total:21.2f}\n")
print("Estoque:\n")

for chave, dados in estoque.items():
    print("Descrição: ", chave)
    print("Quantidade: ", dados[0])
    print(f"Preço: {dados[1]:6.2f}\n")

Ln: 24 Col: 38
```

```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help

===== RESTART: C:/Users/heltai
/Desktop/heltai.py =====
Vendas:

tomate      :   5 x   2.30 =  11.50
batata      :  10 x   1.20 =  12.00
alface      :   5 x   0.45 =   2.25
Custo Total:                25.75

Estoque:

Descrição:  tomate
Quantidade:  995
Preço:      2.30

Descrição:  alface
Quantidade:  495
Preço:      0.45

Descrição:  batata
Quantidade: 1991
Preço:      1.20

Descrição:  feijão
Quantidade:  100
Preço:      1.50

>>> |

Ln: 92 Col: 4
```

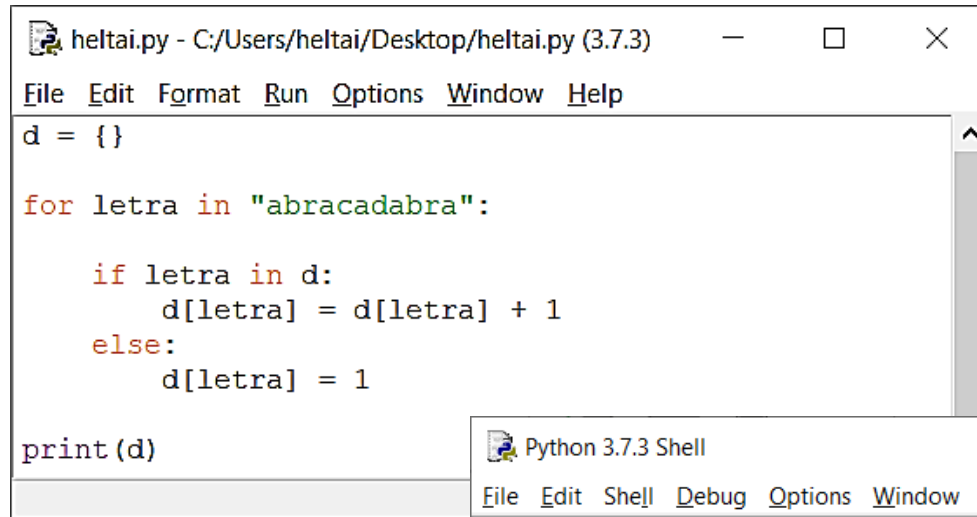


DICIONARIO COM VALOR PADRÃO

DICIONARIOS COM VALOR PADRÃO

DICIONARIOS COM VALOR PADRÃO:

- O uso padrão é uma operação a fim de recuperar o valor de uma chave e, caso não exista, utiliza um valor padrão.
- **EXEMPLO:** Um dicionário para contar o numero de ocorrências de uma letra em uma string:

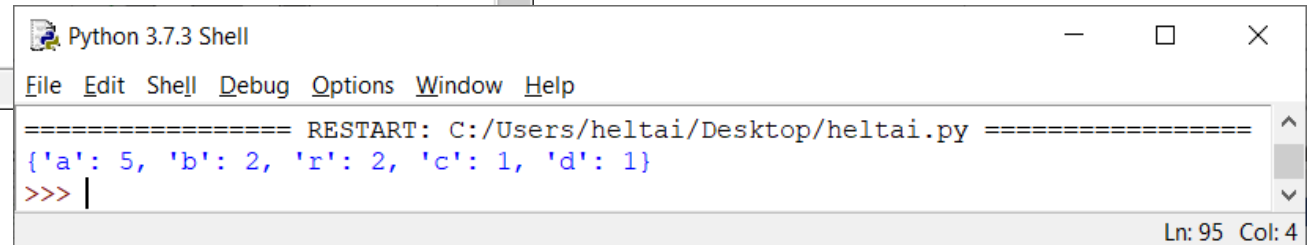


```
heltai.py - C:/Users/heltai/Desktop/heltai.py (3.7.3)
File Edit Format Run Options Window Help
d = {}

for letra in "abracadabra":

    if letra in d:
        d[letra] = d[letra] + 1
    else:
        d[letra] = 1

print(d)
```

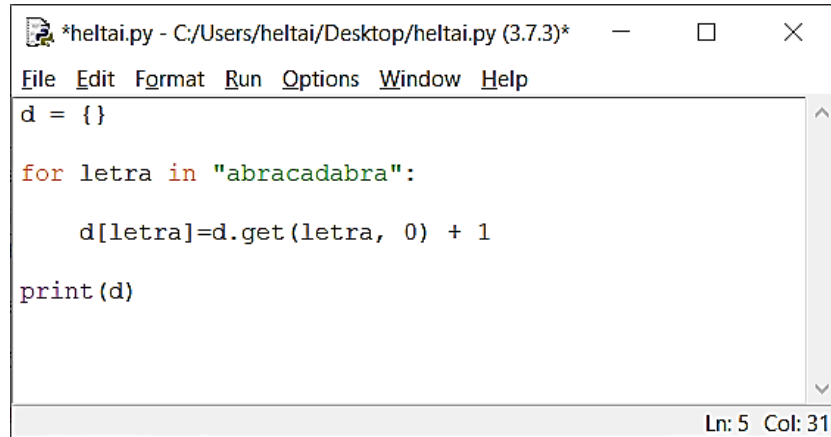


```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
===== RESTART: C:/Users/heltai/Desktop/heltai.py =====
{'a': 5, 'b': 2, 'r': 2, 'c': 1, 'd': 1}
>>> |
Ln: 95 Col: 4
```


DICIONARIOS COM VALOR PADRÃO

MÉTODO GET:

- Utilizando o método **get** do dicionário, pode simplificar o código anterior para:

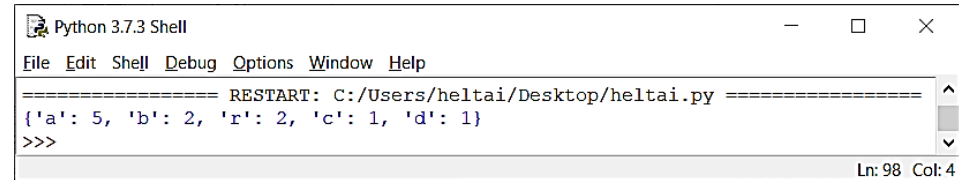


```
*heltai.py - C:/Users/heltai/Desktop/heltai.py (3.7.3)*
File Edit Format Run Options Window Help
d = {}

for letra in "abracadabra":
    d[letra]=d.get(letra, 0) + 1

print(d)

Ln: 5 Col: 31
```



```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
===== RESTART: C:/Users/heltai/Desktop/heltai.py =====
{'a': 5, 'b': 2, 'r': 2, 'c': 1, 'd': 1}
>>>

Ln: 98 Col: 4
```

- O método **get** tenta obter a chave procurada. Caso não a encontre, retorna o segundo parâmetro, no caso 0 (zero). Se o segundo parâmetro não for especificado, **get** retornará **None**. Quando a chave é encontrada no dicionário, **get** retorna o valor atualmente associado.
- None** é um tipo especial no Python que representa nada ou nenhum valor. Assim, uma sting vazia "" não é a mesma coisa que **None**. Todo objeto Python pode receber **None**.



TUPLAS

TUPLAS

TUPLAS

- Tuplas podem ser vistas como listas em Python, com a grande diferença de serem imutáveis.
- Tuplas são ideias para representar listas de valores constantes e para realizar operações de empacotamento e desempacotamento de valores.
- Para criar uma tupla, é seguido o mesmo processo se listas, porem utilizando parênteses em vez de colchetes:

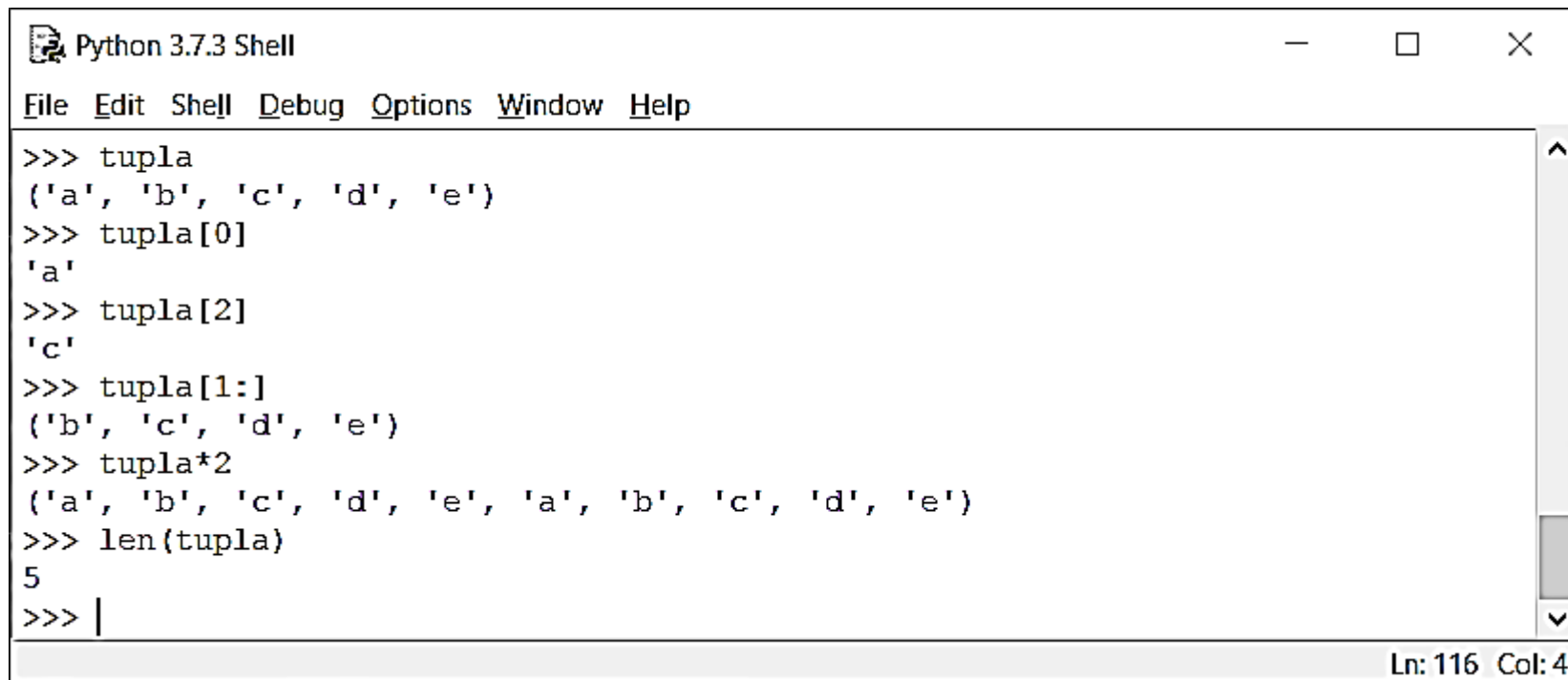
```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
>>> tupla = ("a", "b", "c")
>>> tupla
('a', 'b', 'c')
>>> |
```

```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
>>> tupla = "a", "b", "c", "d", "e"
>>> tupla
('a', 'b', 'c', 'd', 'e')
>>> |
```

- O uso de parênteses são opcionais, porem sua utilização aumenta a clareza da expressão em si, uma vez que visualizar a virgula pode não ser tão fácil.

TUPLAS

- Tuplas suportam a maior parte das operações de lista, como fatiamento e indexação:

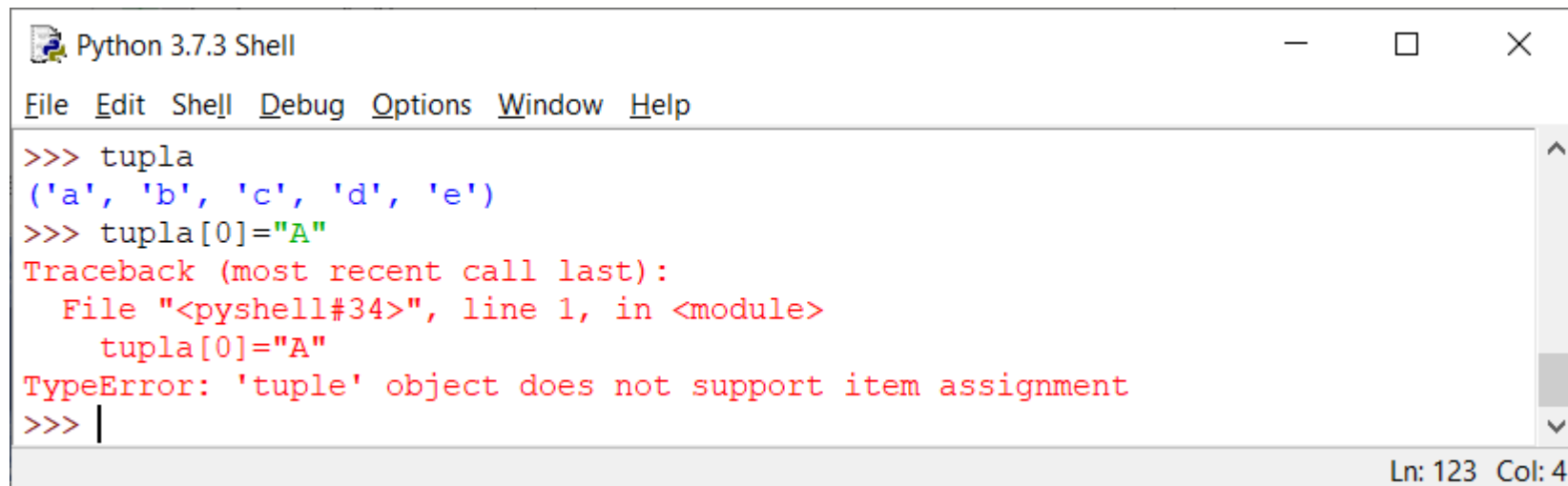


```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
>>> tupla
('a', 'b', 'c', 'd', 'e')
>>> tupla[0]
'a'
>>> tupla[2]
'c'
>>> tupla[1:]
('b', 'c', 'd', 'e')
>>> tupla*2
('a', 'b', 'c', 'd', 'e', 'a', 'b', 'c', 'd', 'e')
>>> len(tupla)
5
>>> |
```

Ln: 116 Col: 4

TUPLAS

- Tupla **não** podem ter seus elementos alterados:

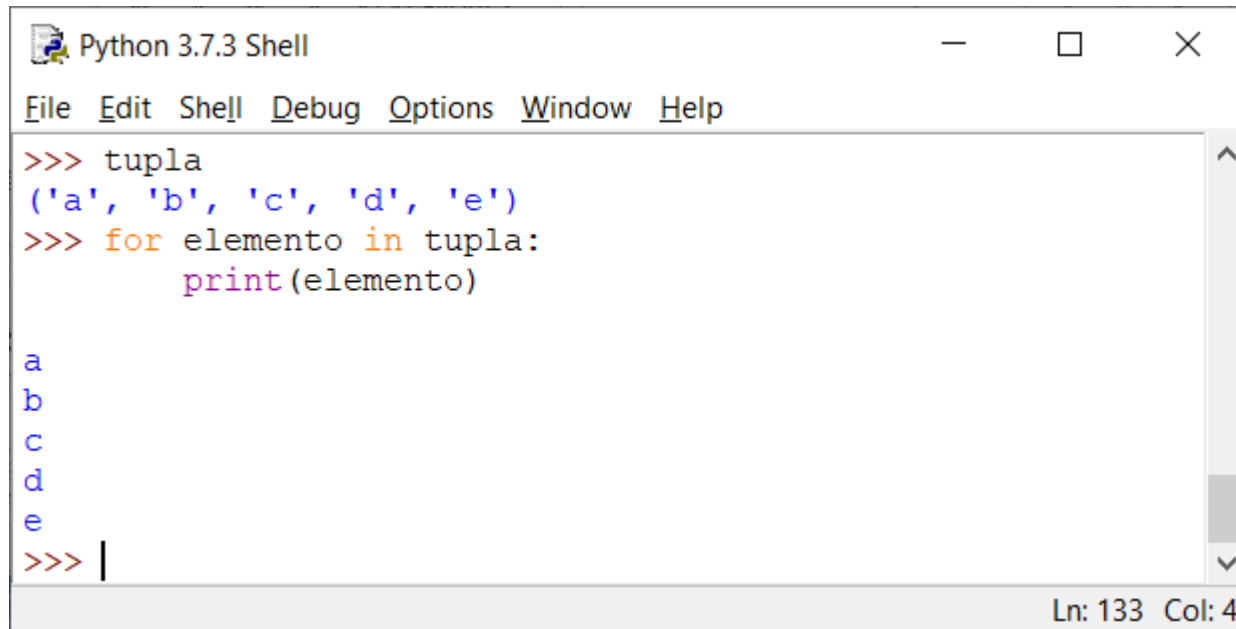


```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
>>> tupla
('a', 'b', 'c', 'd', 'e')
>>> tupla[0]="A"
Traceback (most recent call last):
  File "<pyshell#34>", line 1, in <module>
    tupla[0]="A"
TypeError: 'tuple' object does not support item assignment
>>> |
```

Ln: 123 Col: 4

TUPLAS

- Varias funções utilizam ou geram tuplas em Python.
- Tuplas podem ser utilizadas com **for**:



```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
>>> tupla
('a', 'b', 'c', 'd', 'e')
>>> for elemento in tupla:
    print(elemento)

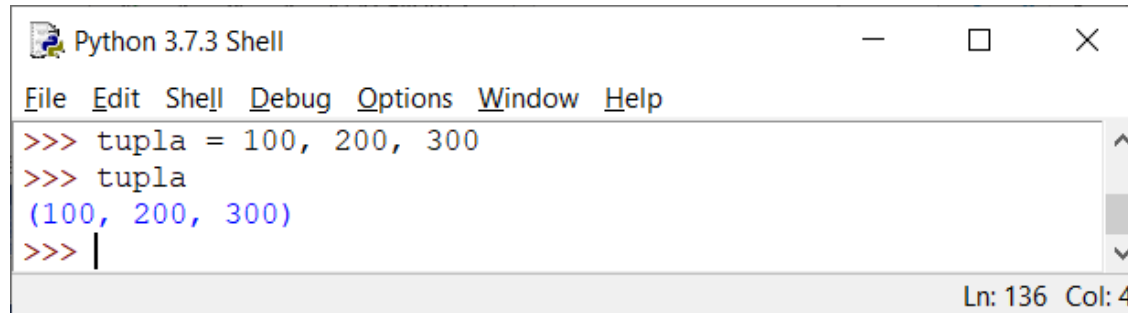
a
b
c
d
e
>>> |
```

Ln: 133 Col: 4

TUPLAS

ENPACOTAMENTO:

- **EXEMPLO:** Uso do tupla:



```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
>>> tupla = 100, 200, 300
>>> tupla
(100, 200, 300)
>>> |
```

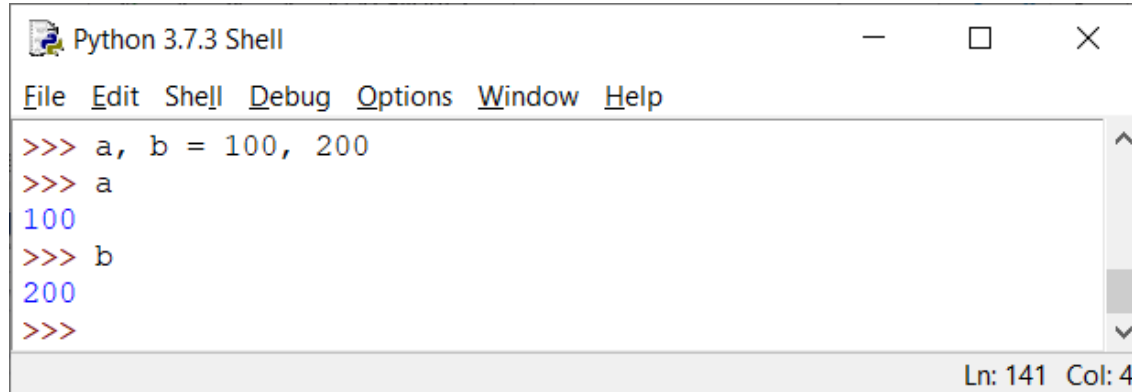
Ln: 136 Col: 4

- No exemplo acima, 100, 200 e 300 foram convertidos em uma tupla com três elementos. Esse tipo de operação é chamado de empacotamento.

TUPLAS

DESEMPACOTAMENTO:

- **EXEMPLO:** Uso do tupla:

A screenshot of a Python 3.7.3 Shell window. The window has a title bar with the text "Python 3.7.3 Shell" and standard window controls (minimize, maximize, close). Below the title bar is a menu bar with the following items: File, Edit, Shell, Debug, Options, Window, and Help. The main area of the window is a text editor showing the following code:

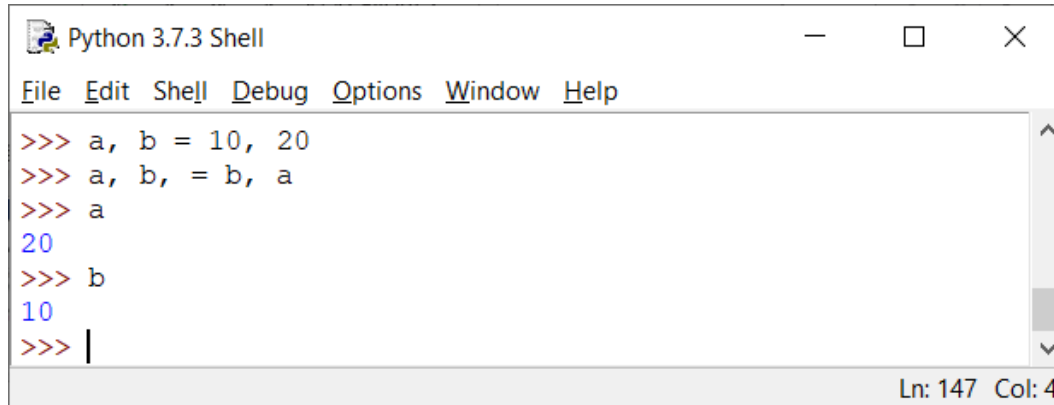
```
>>> a, b = 100, 200
>>> a
100
>>> b
200
>>>
```

The output of the code is shown in blue text. At the bottom right of the window, the status bar displays "Ln: 141 Col: 4".

- Em que o primeiro valor, **10**, foi atribuído à primeira variável **a** e **20**, à segunda, **b**.
- Esse é um exemplo de desempacotamento.
- Muito utilizado, esse tipo de construção é interessante para distribuir o valor de uma tupla em várias variáveis.

TUPLAS

- Há possibilidade de trocar os valores de variáveis com construções específicas:



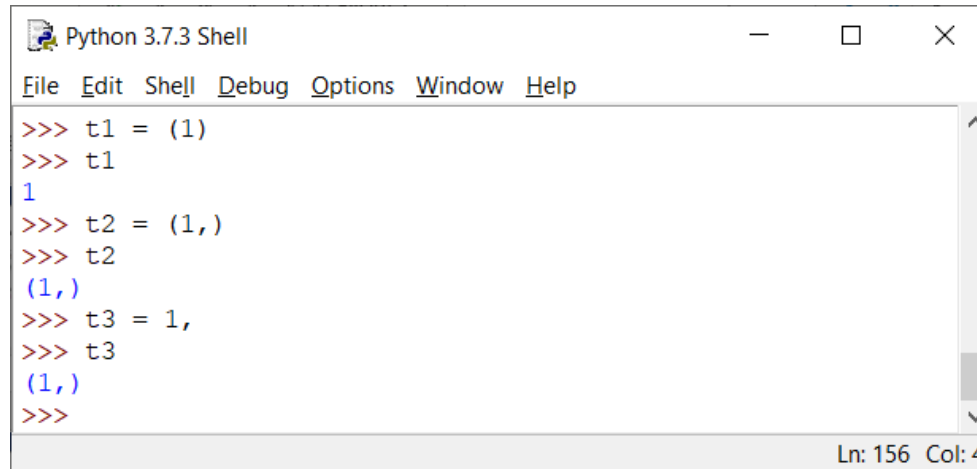
```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
>>> a, b = 10, 20
>>> a, b, = b, a
>>> a
20
>>> b
10
>>> |
```

Ln: 147 Col: 4

- Na tupla da esquerda é usada para atribuir os valores à direita.
- No caso do exemplo, as atribuições: **a = b** e **b = a** foram realizadas imediatamente, sem precisar utilizarmos uma variável intermediária para a troca.

TUPLAS

- A sintaxe do Python é um tanto especial quanto precisamos criar tuplas com apenas um elemento.
- Como os valores são escritos entre parênteses, quando apenas um valor estiver presente, deve-se acrescentar uma vírgula para indicar que o valor é uma tupla com apenas um elemento.
- Exemplos usando e não usando vírgula:



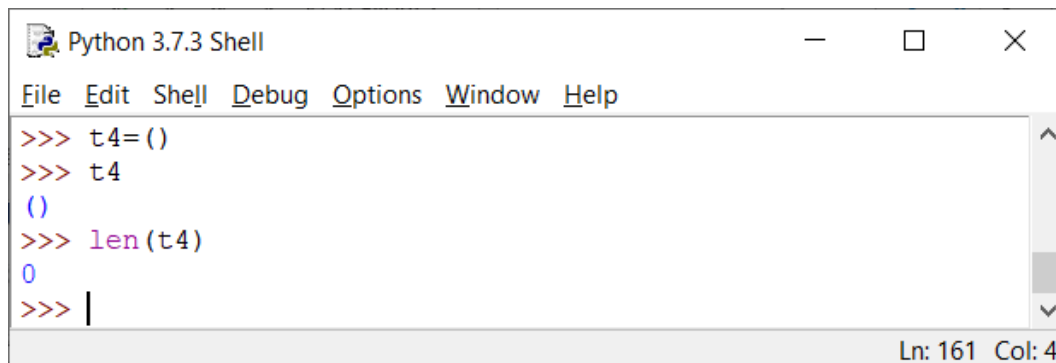
```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
>>> t1 = (1)
>>> t1
1
>>> t2 = (1,)
>>> t2
(1,)
>>> t3 = 1,
>>> t3
(1,)
>>>
```

Ln: 156 Col: 4

- Em t1 não utiliza-se vírgula, e o código foi interpretado como um número inteiro entre parênteses. Em t2, utiliza-se a vírgula e a tupla foi corretamente construída. Em t3, cria-se outra tupla, mas neste caso nem necessita de parênteses.

TUPLAS

- Para criar tupla vazias, escreve-se apenas os parênteses:

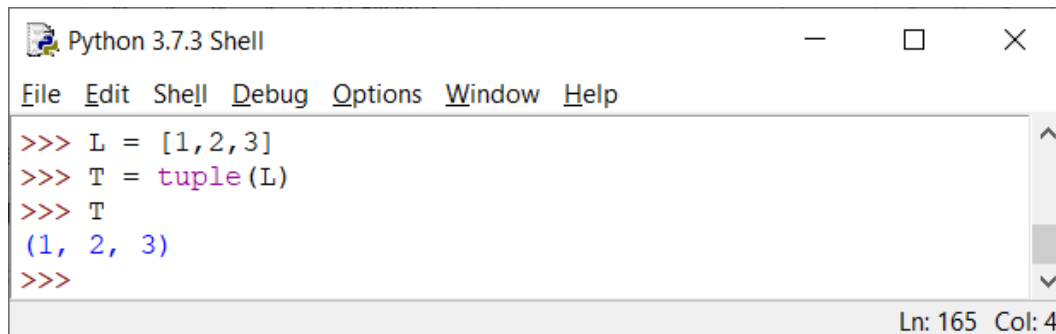


```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
>>> t4=()
>>> t4
()
>>> len(t4)
0
>>> |
```

Ln: 161 Col: 4

FUNÇÃO TUPLE:

- A criação da tuplas também podem ser criada a partir de listas, utilizando-se a função **tuple**:



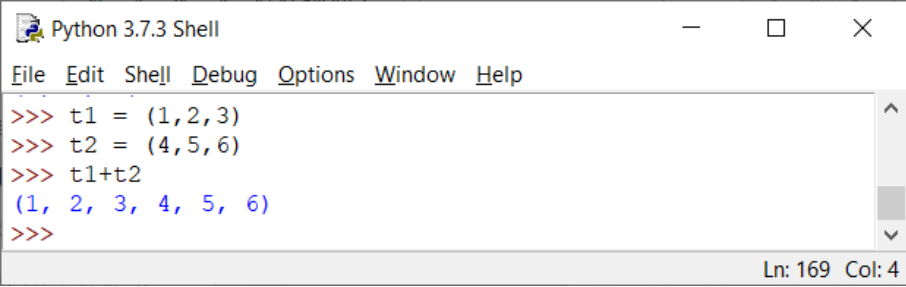
```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
>>> L = [1,2,3]
>>> T = tuple(L)
>>> T
(1, 2, 3)
>>>
```

Ln: 165 Col: 4

TUPLAS

CONCATENAÇÃO DE TUPLA:

- Não pode ser alterado uma tupla depois de sua criação, porem é possível concatenar, gerando novas tuplas:

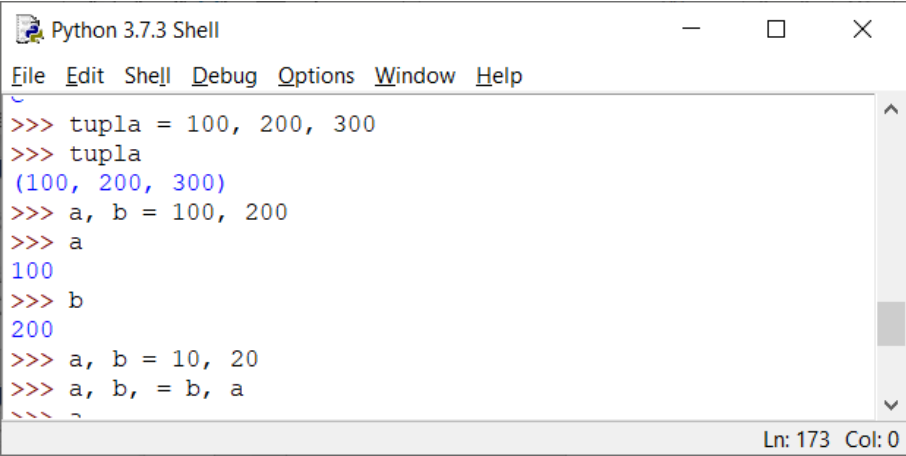
A screenshot of a Python 3.7.3 Shell window. The window has a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area shows the following code and output:

```
>>> t1 = (1,2,3)
>>> t2 = (4,5,6)
>>> t1+t2
(1, 2, 3, 4, 5, 6)
>>>
```

The status bar at the bottom right indicates 'Ln: 169 Col: 4'.

TUPLA COM LISTA:

- Uma tupla conter uma lista ou outro objeto que pode ser alterado, está continua a funcionar normalmente.

A screenshot of a Python 3.7.3 Shell window. The window has a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area shows the following code and output:

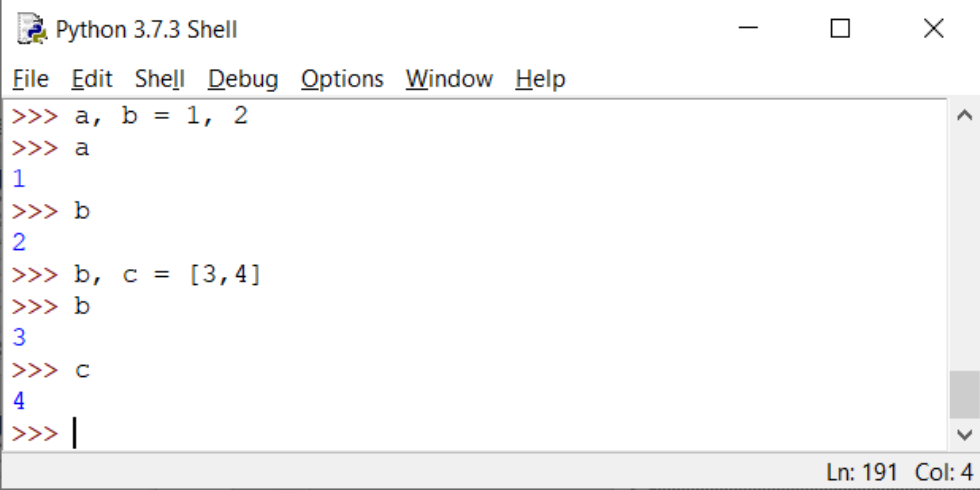
```
>>> tupla = 100, 200, 300
>>> tupla
(100, 200, 300)
>>> a, b = 100, 200
>>> a
100
>>> b
200
>>> a, b = 10, 20
>>> a, b, = b, a
>>>
```

The status bar at the bottom right indicates 'Ln: 173 Col: 0'.

TUPLAS

EMPACOTAMENTO E DESEMPACOTAMENTO COM LISTA:

- As operações de empacotamento e desempacotamento funcionam com listas

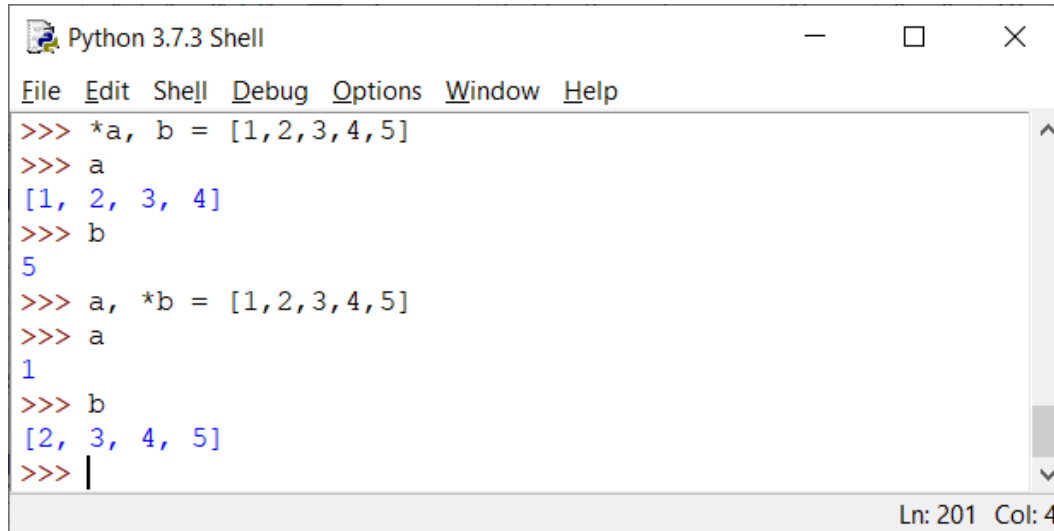


```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
>>> a, b = 1, 2
>>> a
1
>>> b
2
>>> b, c = [3, 4]
>>> b
3
>>> c
4
>>> |
```

Ln: 191 Col: 4

TUPLAS

- Pode-se usar o * para indicar vários valores a desempacotar:



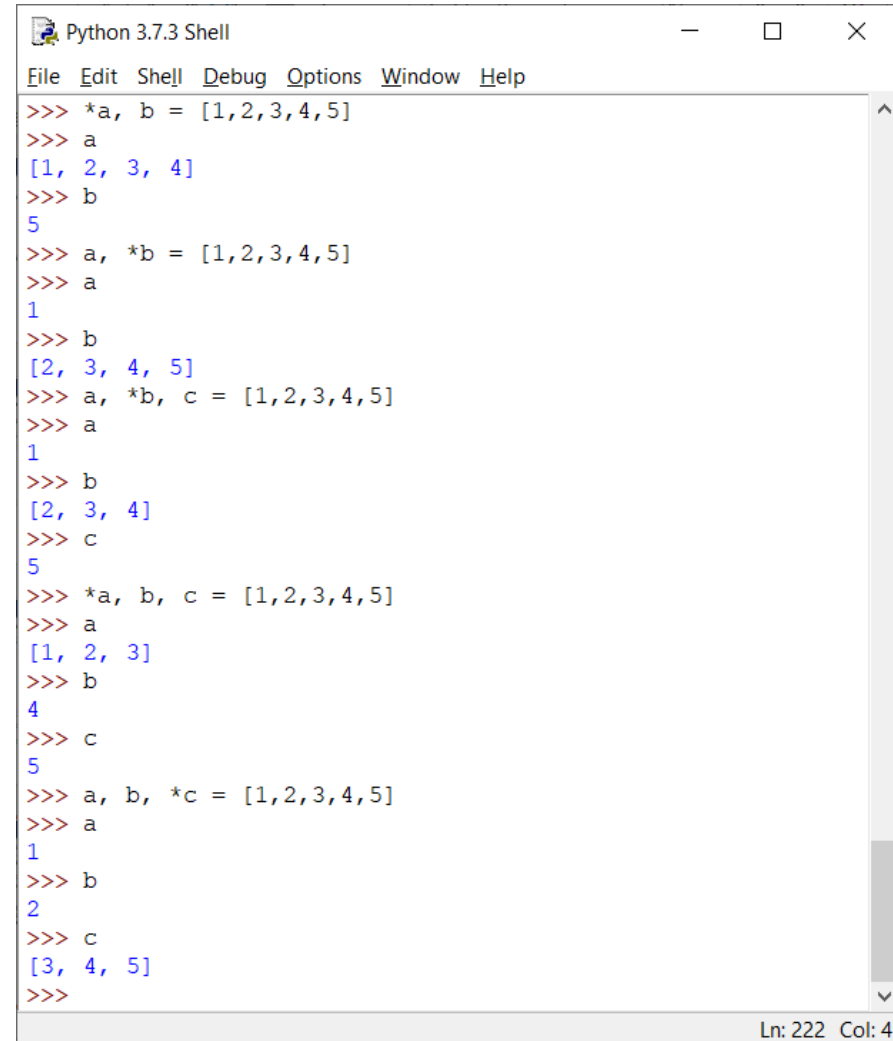
```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
>>> *a, b = [1,2,3,4,5]
>>> a
[1, 2, 3, 4]
>>> b
5
>>> a, *b = [1,2,3,4,5]
>>> a
1
>>> b
[2, 3, 4, 5]
>>> |
```

Ln: 201 Col: 4

- No caso de *a, b, se diz: coloque o ultimo valor em b e os restantes em a.
- No caso de a, *b, se diz: coloque o primeiro valor em a e os outros em b.

TUPLAS

- Entende-se que a sintaxe que a variável sem o * recebe apenas um valor, no caso, o valor na mesma posição da variável.
- A variável com * receberá os valores que sobraram após a distribuição dos valores às variáveis sem o *.
- Em *c, d, e, pode-se que dizer que c é a variável com o *, logo receberá todos os valores, salvo aqueles alocados a d e e (penúltimo e ultimo valor, respectivamente).



```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help

>>> *a, b = [1,2,3,4,5]
>>> a
[1, 2, 3, 4]
>>> b
5
>>> a, *b = [1,2,3,4,5]
>>> a
1
>>> b
[2, 3, 4, 5]
>>> a, *b, c = [1,2,3,4,5]
>>> a
1
>>> b
[2, 3, 4]
>>> c
5
>>> *a, b, c = [1,2,3,4,5]
>>> a
[1, 2, 3]
>>> b
4
>>> c
5
>>> a, b, *c = [1,2,3,4,5]
>>> a
1
>>> b
2
>>> c
[3, 4, 5]
>>>
```

Ln: 222 Col: 4

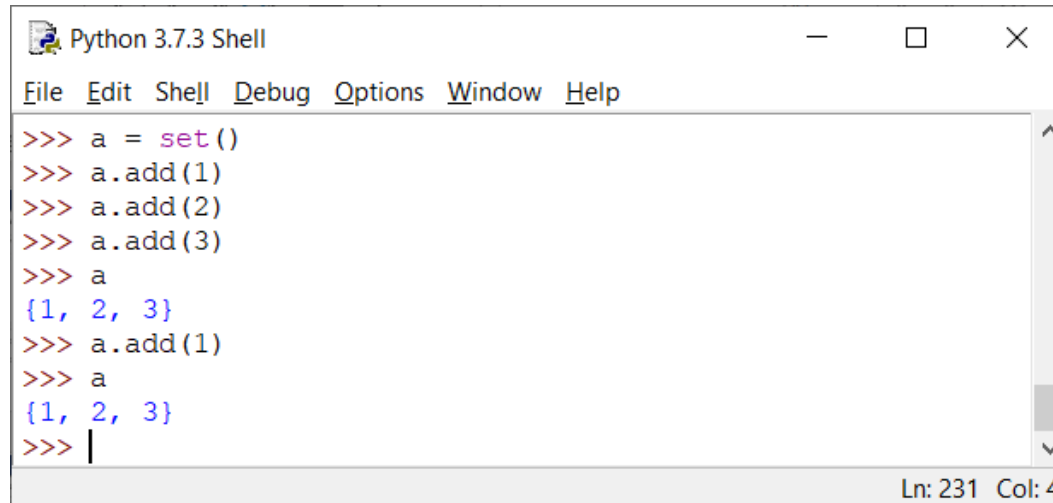
The background of the slide is a blue-tinted image of a complex electronic circuit board. It features numerous integrated circuits, resistors, and intricate white circuit traces. The top half of the image is slightly blurred, while the bottom half shows more detail of the components and wiring.

CONJUNTOS (SET)

CONJUNTOS (SET)

CONJUNTOS (SET):

- Conjuntos, **set** em Python, é uma estrutura de dados que implementam operações de união, intersecção e diferença, entre outras.
- A principal características dessa estrutura de dados é não admitir repetição de elementos, como os conjuntos da matemática.
- Além disso, conjuntos não matam a ordem de seus elementos.

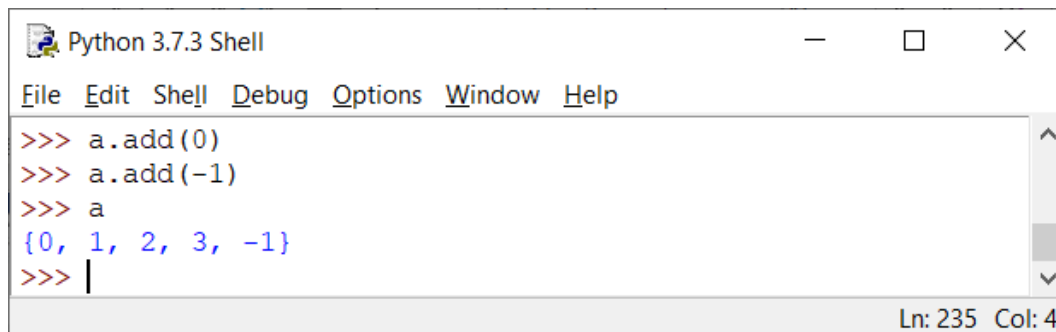


```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
>>> a = set()
>>> a.add(1)
>>> a.add(2)
>>> a.add(3)
>>> a
{1, 2, 3}
>>> a.add(1)
>>> a
{1, 2, 3}
>>> |
```

Ln: 231 Col: 4

CONJUNTOS (SET)

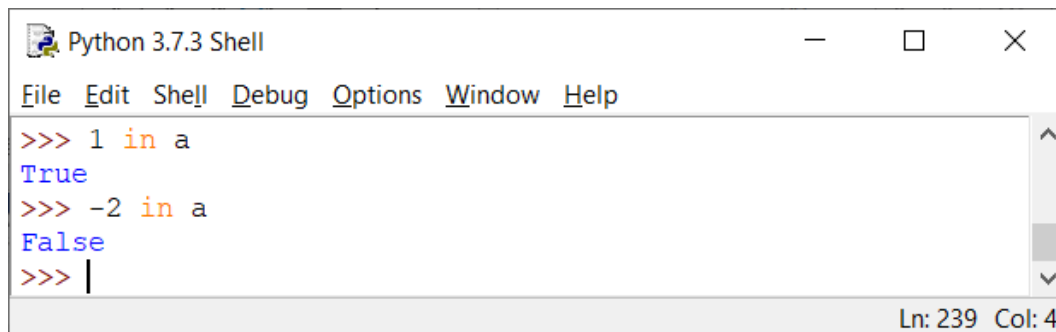
- No exemplo, seguindo com o processo de adicionar 0 e -1, A ordem relativa que se vê é mera coincidência:



```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
>>> a.add(0)
>>> a.add(-1)
>>> a
{0, 1, 2, 3, -1}
>>> |
```

Ln: 235 Col: 4

- É possível testar se o elemento faz parte de um conjunto usando o operador **in** do Python:

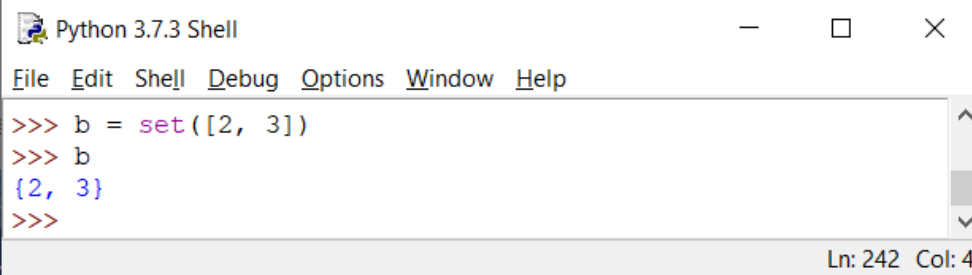


```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
>>> 1 in a
True
>>> -2 in a
False
>>> |
```

Ln: 239 Col: 4

CONJUNTOS (SET)

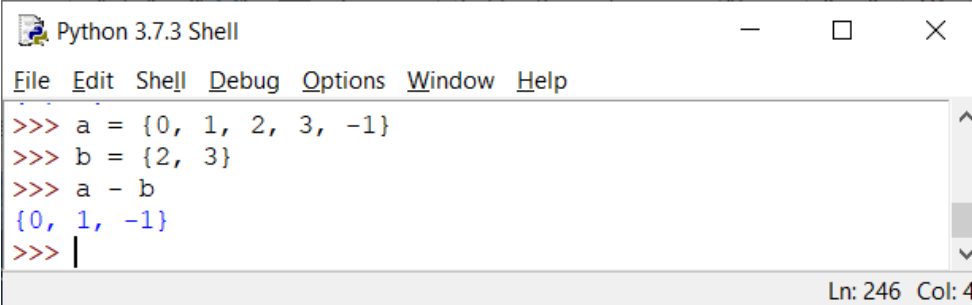
- Um **set** (conjunto) pode ser criado a partir de listas, tuplas e qualquer outra estrutura de dados que seja enumerável.



```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
>>> b = set([2, 3])
>>> b
{2, 3}
>>>
```

Ln: 242 Col: 4

- A operação disponível com conjuntos, tem-se a diferença entre conjuntos, que utiliza o operador - :



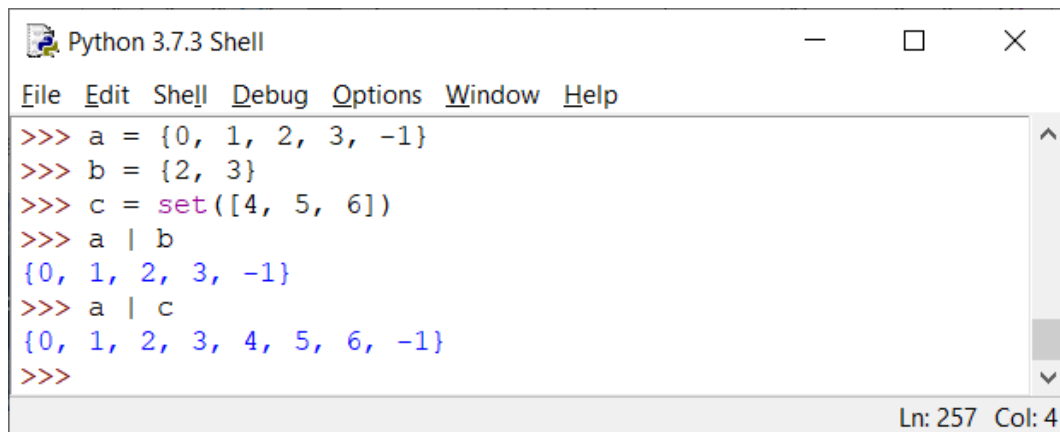
```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
>>> a = {0, 1, 2, 3, -1}
>>> b = {2, 3}
>>> a - b
{0, 1, -1}
>>> |
```

Ln: 246 Col: 4

- O resultado contém apenas os elementos de a que não pertence em b.

CONJUNTOS (SET)

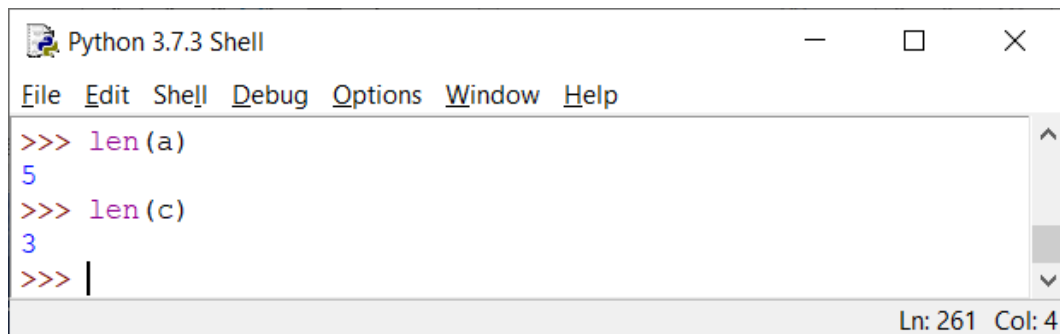
- A união é realizada pelo operador |



```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
>>> a = {0, 1, 2, 3, -1}
>>> b = {2, 3}
>>> c = set([4, 5, 6])
>>> a | b
{0, 1, 2, 3, -1}
>>> a | c
{0, 1, 2, 3, 4, 5, 6, -1}
>>>
```


Ln: 257 Col: 4

- Conjuntos set, possuem outras propriedades, como tamanho (numero de elementos), que podem ser obtidas com o uso da função **len**:



```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
>>> len(a)
5
>>> len(c)
3
>>> |
```

Ln: 261 Col: 4



QUAL ESTRUTURA DE DADOS UTILIZAR?

QUAL ESTRUTURA DE DADOS UTILIZAR?

- Como programador, precisa-se decidir constantemente qual estrutura de dados utilizar na programação.
- Abaixo um resumo das principais (vista até a presente):

	LISTAS	TUPLAS	DICIONÁRIOS	CONJUNTOS
Ordem dos elementos	Fixa	Fixa	Mantida a partir do Python 3.7	Indeterminada
Tamanho	Variável	Fixo	Variável	Variável
Elementos Repetidos	Sim	Sim	Pode repetir valores, mas as chaves devem ser únicas	Não
Pesquisa	Sequencial, índice, numérico	Sequencial, índice, numérico	Direta por chave	Direta por valores
Alterações	Sim	Não	Sim	Sim
Uso primário	Sequências	Sequências constantes	Dados indexados por chave	Verificação de unicidade, operações com conjuntos.



PROF. VINICIUS HELTAI

vinicius.pacheco@docente.unip.br

www.heltai.com.br

(11) 98200-3932



/vheltai



@vheltai



/vheltai



@Vinicius Heltai



@vheltai