

# Python

## Curso Intensivo

Prof. Cláudio Fleury  
Abr-22

Curso baseado no  
livro de Eric  
Mattes, **Python**  
**Crash Course**

Cap.6 –  
Dicionários

# Conteúdo

*Dicionários  
permitem a  
conexão de  
pedaços de  
informações  
relacionadas...*

1. Um Dicionário Simples
2. Definição
3. Acessando Valores do Dicionário
4. Existência de uma **chave**
5. Adicionando itens (pares **chave-valor**) ao Dicionário
6. Modificando Valores do Dicionário
7. Removendo Pares **chave-valor**
8. Dicionário de Valores Similares
9. Percorrendo um Dicionário
10. Mesclando Dicionários
11. Funções Incorporadas
12. Resumo



# Um Dicionário Simples

Considere um jogo com alienígenas que podem ter diferentes valores de cor e ponto. Este dicionário simples armazena informações sobre um alienígena em particular:

File Edit Selection Find View Goto Tools Project Preferences Help

FOLD

platypus\_doc.py

dimensoes2.py

carros3.py

alienigina.py

```
1 alien_0 = {'cor': 'verde', 'pontos': 5}
2 print(alien_0['cor'])
3 print(alien_0['pontos'])
```

verde

5

[Finished in 187ms]

# Definição

- ♦ Um dicionário em Python é uma coleção de pares **chave-valor**. Cada chave está conectada a um valor e você pode usar uma chave para acessar o valor associado a essa chave. O valor da chave pode ser um número, uma string, uma lista ou até mesmo outro dicionário
- ♦ Cada chave é conectada ao seu valor por **dois pontos** e os pares **chave-valor** individuais são separados por vírgulas. Você pode armazenar quantos pares **chave-valor** desejar em um dicionário

```
alien_0 = {'cor': 'verde', 'pontos': 5}
```



A partir do Python 3.7, os dicionários mantêm a ordem em que foram definidos. Ao percorrer seus elementos, você verá os elementos na mesma ordem em que foram adicionados

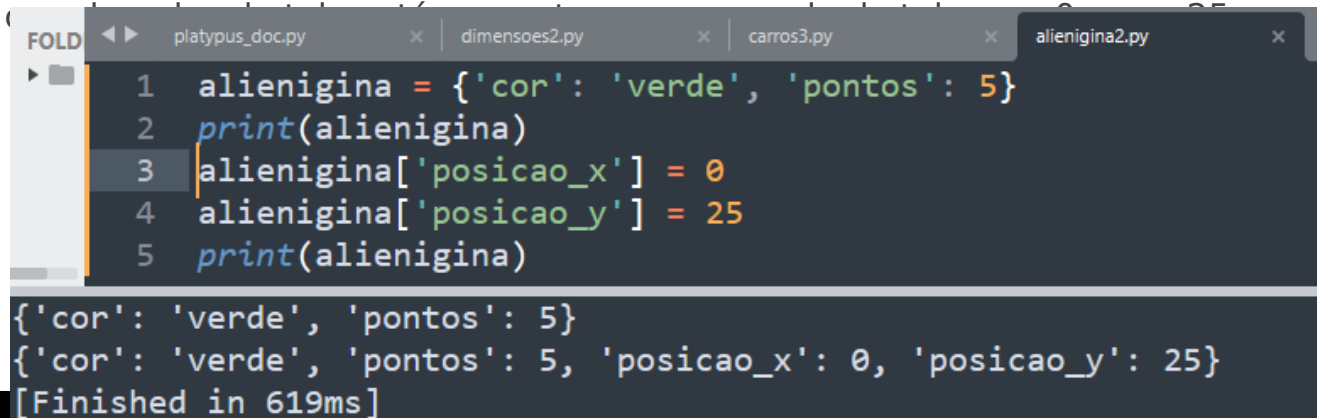
# Acessando Valores do Dicionário

- ◆ Forneça a chave como índice e o dicionário retorna o valor associado

```
alienigina = {'cor': 'verde', 'pontos': 5}
print(alienigina['cor'], alienigina['pontos'])
```

Verde 5

- ◆ Vamos colocar o alienígena na borda esquerda da tela, 25 pixels abaixo do topo. A origem das



```
FOLD ▶ platypus_doc.py x dimensoes2.py x carros3.py x alienigina2.py x
1 alienigina = {'cor': 'verde', 'pontos': 5}
2 print(alienigina)
3 alienigina['posicao_x'] = 0
4 alienigina['posicao_y'] = 25
5 print(alienigina)

{'cor': 'verde', 'pontos': 5}
{'cor': 'verde', 'pontos': 5, 'posicao_x': 0, 'posicao_y': 25}
[Finished in 619ms]
```


Dicionário vazio:  
`alien_0 = { }`

# Acessando Valores do Dicionário

- ◆ Usando **chave** entre colchetes para recuperar o **valor** associado de um dicionário pode causar um problema se a **chave** especificada não existir no dicionário, ou seja, você receberá um erro:

```
alienigina = {'cor': 'verde', 'veloc': 'lento'}  
print(alienigina['pontos'])
```

```
Traceback (most recent call last):  
File "alienigina.py", line 2, in <module>  
print(alienigina['pontos'])  
KeyError: 'pontos'
```




# Existência de uma **chave**

- ◆ Uma maneira segura de acessar um elemento de um dicionário é verificando primeiro se a **chave** existe

```
cores = {'amarelo': 10, 'vermelho': 2, 'cinza': 55}

# Verificando se uma dada chave existe
if 'amarelo' in cores:
    print(f"Estoque da cor desejada: {cores['amarelo']}")

# Verificando se um dado valor existe
if 10 in cores.values():
    print('O valor desejado existe!')
```



# Adicionando itens (pares **chave-valor**) ao Dicionário

- ◆ A maneira mais simples de adicionar um item ao dicionário é criar uma nova **chave** e atribuir um **valor**

```
alunos = {'joão': 10, 'maria': 2}  
alunos['ana'] = 8.5  
print(alunos) → {'joão': 10, 'maria': 2, 'ana': 8.5}
```

- ◆ Pode-se usar o método **update()** para criar ou atualizar dados:

```
alunos.update({'maria': 6})  
alunos.update({'carlos': 5.2})  
print(alunos) → {'joão': 10, 'maria': 6, 'ana': 8.5, 'carlos': 5.2}
```



# Modificando Valores do Dicionário

- ◆ Para modificar um valor em um dicionário, use o nome do dicionário com a chave entre colchetes e, depois do operador de atribuição, o novo valor que você deseja associar aquela chave

```
alienigina = {'cor': 'verde', 'veloc': 'lento'}  
print(f"O alienígena é {alienigina['cor']}".)
```

```
alienigina['cor'] = 'amarelo'  
print(f"O alienígena é {alienigina['cor']}".)
```

```
In [2]: alienigina = {'cor': 'verde', 'veloc': 'lento'}  
...: print(f"O alienígena é {alienigina['cor']}".)  
...:  
...: alienigina['cor'] = 'amarelo'  
...: print(f"O alienígena é {alienigina['cor']}".)  
O alienígena é verde.  
O alienígena é amarelo.
```

## Removendo Pares **chave-valor**

- ♦ Para remover uma informação armazenada num dicionário, usamos a instrução **del**
- ♦ É preciso o nome do dicionário e a chave do par que se deseja remover
- ♦ Removendo o par da chave 'pontos'

```
FOLD  platypus_doc.py  x  dimensoes2.py  x  carros3.py  x  alienigina3.py  x
1  alienigina = {'cor': 'verde', 'pontos': 5, 'posicao_x': 0,
2  print(alienigina)
3  del alienigina['pontos']
4  print(alienigina)
```

```
{'cor': 'verde', 'pontos': 5, 'posicao_x': 0, 'posicao_y': 25}
{'cor': 'verde', 'posicao_x': 0, 'posicao_y': 25}
[Finished in 174ms]
```

# Removendo item com métodos

- ♦ **pop(chave)** - remove e retorna o valor associado à chave informada

```
compras = {'maça': 2, 'ovos': 6, 'farinha': 2}
qtd = compras.pop('ovos')
print("Não comprar ", qtd, "ovos")
print(compras)
```

Não comprar 6 ovos  
{ 'maça': 2, 'farinha': 2 }

- ♦ **popitem()** - remove o último item do dicionário, e retorna o par chave-valor removido

```
compras = {'maça': 2, 'ovos': 6, 'farinha': 2}
fora = compras.popitem()
print("Não comprar ", fora)
print(compras)
```

Não comprar ('farinha', 2)  
{ 'maça': 2, 'ovos': 6 }

# Copiando um Dicionário

- ♦ **Cópia rasa:** `dict2 = dict1` para copiar apenas a referência do dict1 ao dict2.
- ♦ **Cópia funda:** para copiar todo o dicionário usamos o método **`copy()`**

```
tarefas = {"operacao": "web scraping", "dados": 250}
copia = tarefas.copy()
print(copia)
print(dicio)
```

```
{'operacao': 'web scraping', 'dados': 250}
{'operacao': 'web scraping', 'dados': 250}
```



# Dicionário de Valores Similares

- ♦ O exemplo anterior armazenou diferentes tipos de informações sobre um objeto, um alienígena de um jogo
- ♦ Podemos usar um dicionário para armazenar informações sobre muitos objetos similares

```
FOLD  linguagens.py
1  linguagem_favorita = {
2      'Márcia': 'java',
3      'João': 'python',
4      'Rubens': 'c',
5      'Ana': 'fortran'
6  }
7  print(linguagem_favorita)
8  linguagem = linguagem_favorita['João'].title()
9  print(f"A linguagem favorita do João é {linguagem}.")

{'Márcia': 'java', 'João': 'python', 'Rubens': 'c', 'Ana': 'fortran'}
A linguagem favorita do João é Python.
[Finished in 179ms]
```

# Acessando Dados do Dicionário

## ◆ Usando o método **get(chave, msg)**

O método **get()** requer uma chave como primeiro argumento, e como segundo argumento opcional, o valor a ser retornado se a chave não existir no dicionário.

```
FOLD  linguagens.py x
1  linguagem_favorita = {
2      'Márcia': 'java',
3      'João': 'python',
4      'Rubens': 'c',
5      'Ana': 'fortran'
6  }
7  print(linguagem_favorita)
8  linguagem = linguagem_favorita['João'].title()
9  print(f"A linguagem favorita do João é {linguagem}.")
10 linguagem = linguagem_favorita.get('Maria', '**Usuário não cadastrado**')
11 print(f"A linguagem favorita do Maria é {linguagem}.")
```

```
{'Márcia': 'java', 'João': 'python', 'Rubens': 'c', 'Ana': 'fortran'}
A linguagem favorita do João é Python.
A linguagem favorita do Maria é **Usuário não cadastrado**.
[Finished in 188ms]
```

# Exercícios

- 1. Pessoa:** Use um dicionário para armazenar informações sobre uma pessoa que você conhece. Armazene seu primeiro nome, sobrenome, idade e a cidade em que moram. Você deve ter chaves como `first_name`, `last_name`, `age` e `city`. Imprimir cada informação armazenada em seu dicionário.
- 2. Números favoritos:** use um dicionário para armazenar os números favoritos das pessoas. Pense em cinco nomes e use-os como chaves em seu dicionário. Pense em um número favorito para cada pessoa e armazene cada um como um valor em seu dicionário. Imprimir o nome de cada pessoa e seu número favorito. Para se divertir ainda mais, pesquise alguns amigos e obtenha alguns dados reais para o seu programa.
- 3. Glossário:** Um dicionário Python pode ser usado para modelar um dicionário real. No entanto, para evitar confusão, vamos chamá-lo de glossário. Pense em cinco palavras de programação que você aprendeu nos capítulos anteriores. Use essas palavras como chaves em seu glossário armazene seus significados como valores. Imprima cada palavra e seu significado como saída bem formatada. Você pode imprimir a palavra seguida de dois pontos e depois o seu significado, ou imprimir a palavra em uma linha e, em seguida, imprimir seu significado recuado em uma segunda linha. Use o caractere de nova linha (`\n`) para inserir uma linha em branco entre cada par palavra-significado em sua saída.

# Percorrendo um Dicionário

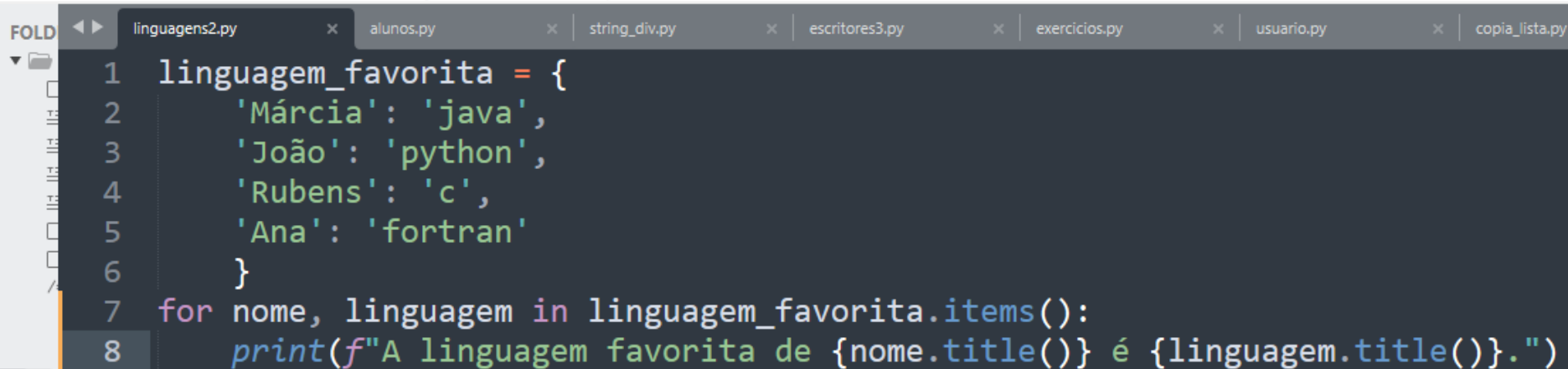
- ◆ Um dicionário pode conter apenas alguns pares de **chave-valor** ou milhões de pares
- ◆ Como um dicionário pode conter grandes quantidades de dados, o Python permite que se percorra um dicionário de várias formas
  - ◆ Pares chave-valor
  - ◆ Chaves
  - ◆ Chaves em uma Ordem em Particular
  - ◆ Valores





# Percorrendo um Dicionário

- ◆ Pares chave-valor: método **items()**
- ◆ Experimente: `print(dicionario.items())`



```
FOLD ◀ ▶ linguagens2.py x alunos.py x string_div.py x escritores3.py x exercicios.py x usuario.py x copia_lista.py
1 linguagem_favorita = {
2     'Márcia': 'java',
3     'João': 'python',
4     'Rubens': 'c',
5     'Ana': 'fortran'
6 }
7 for nome, linguagem in linguagem_favorita.items():
8     print(f"A linguagem favorita de {nome.title()} é {linguagem.title()}")
```

```
A linguagem favorita de Márcia é Java.
A linguagem favorita de João é Python.
A linguagem favorita de Rubens é C.
A linguagem favorita de Ana é Fortran.
```

# Percorrendo um Dicionário

- ◆ Chaves: método **keys()**; forma padrão de se percorrer um dicionário

```
FOLD <> linguagens3.py
1 ling_favorita = {
2     'Márcia': 'java',
3     'João': 'python',
4     'Rubens': 'c',
5     'Ana': 'fortran'
6 }
7 for nome in ling_favorita: # ou ling_favorita.keys()
8     print(nome.title())
```

```
Márcia
João
Rubens
Ana
```

# Percorrendo um Dicionário

- ◆ Chaves - Vamos percorrer os nomes no dicionário, e quando o nome corresponder ao de um dos nossos amigos, exibiremos uma mensagem sobre a linguagem favorita dele:

```
FOLD  linguagens4.py x
1  ling_favorita = {
2      'márcia': 'java',
3      'joão': 'python',
4      'rubens': 'c',
5      'ana': 'fortran'
6  }
7  amigos = ['rubens', 'márcia']
8  for nome in ling_favorita.keys():
9      print(nome.title())
10     if nome in amigos:
11         linguagem = ling_favorita[nome].title()
12         print(f"\t{nome.title()}, eu vi que você gosta de {linguagem}!")
```

Márcia

Márcia, eu vi que você gosta de Java!

João

Rubens

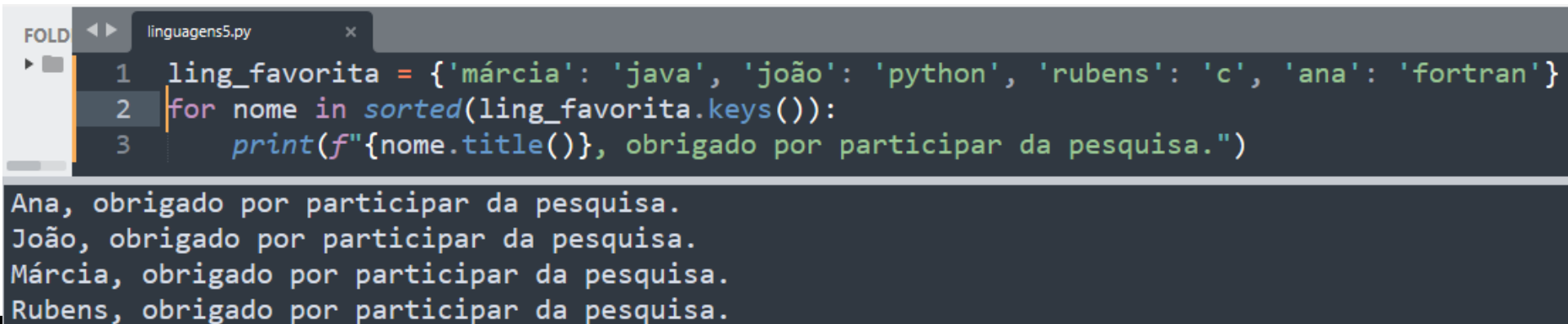
Rubens, eu vi que você gosta de C!

Ana

# Percorrendo um Dicionário

## ◆ Chaves em uma **Ordem em Particular**

- ◆ A partir do Python 3.7, ao percorrer um dicionário os itens são acessados na mesma ordem em que foram inseridos
- ◆ Às vezes, queremos percorrer o dicionário numa ordem diferente da entrada das chaves, classificando-as numa ordem em particular...

A screenshot of a code editor window titled 'linguagens5.py'. The code defines a dictionary 'ling\_favorita' with four entries: 'márcia': 'java', 'joão': 'python', 'rubens': 'c', and 'ana': 'fortran'. A 'for' loop iterates over the sorted keys of this dictionary. For each key, it prints the name in title case followed by a thank-you message. The output in the terminal below the code shows the names sorted alphabetically: Ana, João, Márcia, and Rubens.

```
FOLD ◀ ▶ linguagens5.py ×
1 ling_favorita = {'márcia': 'java', 'joão': 'python', 'rubens': 'c', 'ana': 'fortran'}
2 for nome in sorted(ling_favorita.keys()):
3     print(f"{nome.title()}, obrigado por participar da pesquisa.")
```

```
Ana, obrigado por participar da pesquisa.
João, obrigado por participar da pesquisa.
Márcia, obrigado por participar da pesquisa.
Rubens, obrigado por participar da pesquisa.
```

# Percorrendo um Dicionário

- ◆ Valores: método **values()**
- ◆ Se você estiver interessado principalmente nos **valores** de um dicionário, use o método **values()** para retornar uma lista de valores sem nenhuma **chave**

```
FOLD  linguagens6.py x
1  ling_favorita = {'márcia': 'java', 'joão': 'python', 'rubens': 'c', 'ana': 'fortran'}
2  print("Linguagens mais votadas na pesquisa:")
3  for ling in ling_favorita.values():
4      print(ling.title(), end=', ')
5  print('...\n')
```

Linguagens mais votadas na pesquisa:  
Java, Python, C, Fortran, ...

# Percorrendo um Dicionário

- ◆ Essa abordagem extrai todos os valores do dicionário sem verificar repetições
- ◆ Isso funciona bem com um pequeno número de valores, mas numa enquete com um grande número de respondentes, teríamos uma lista bem repetitiva
- ◆ Para ver cada linguagem votada, e sem repetição, podemos usar um conjunto (**set**)

```
FOLD  linguagens7.py x
1 ling_favorita = {'márcia': 'java', 'joão': 'python', 'rubens': 'c', 'ana': 'python'}
2 print("Linguagens mais votadas na pesquisa:")
3 for ling in ling_favorita.values():
4     print(ling.title(), end=', ')
5 print('...\n')
```

```
Linguagens mais votadas na pesquisa:
Java, Python, C, Python, ...
```

# Percorrendo um Dicionário

```
>>> linguagens = {'python', 'ruby', 'python', 'c'}  
>>> linguagens  
{'ruby', 'python', 'c'}
```

- ◆ Um **set** é uma coleção de dados (assim como lista ou tupla), na qual cada item é único (não ocorrem repetições)
- ◆ Podemos construir um conjunto diretamente usando chaves e separando os elementos com vírgulas

```
FOLD  linguagens8.py x  
1 ling_favorita = {'márcia': 'java', 'joão': 'python', 'rubens': 'c', 'ana': 'python'}  
2 print("Linguagens mais votadas na pesquisa:")  
3 for ling in set(ling_favorita.values()):  
4     print(ling.title(), end=', ')  
5 print('...\n')  
  
Linguagens mais votadas na pesquisa:  
Java, Python, C, ...
```

**Dicionários e Conjuntos** usam chaves, { } como delimitadores, mas chaves sem par(es) de **chave-valor**, então teremos um **conjunto**. Diferentemente das **listas** e **dicionários**, os **conjuntos** não retêm itens em nenhuma ordem específica.

# Mesclando Dicionários

- ◆ Dados dois dicionários a serem combinados, usamos o método **update()**
  - Para **dic1.update(dic2)**, os pares **chave-valor** de **dic2** serão escritos no dicionário **dic1**
  - Para chaves idênticas em **dic1** e **dic2**, o valor em **dic1** será substituído pelo valor correspondente em **dic2**

```
FOI  linguagens9.py x
1  ling_favorita = {'márcia':'java', 'joão':'python', 'rubens':'c'}
2  novos = {'luiz':'c', 'ana':'python'}
3  ling_favorita.update(novos)
4  print(ling_favorita)
5  print(novos)

{'márcia': 'java', 'joão': 'python', 'rubens': 'c', 'luiz': 'c', 'ana': 'python'}
{'luiz': 'c', 'ana': 'python'}
```



# Exercícios

1. **Rios:** Faça um dicionário contendo três grandes rios e o país cada rio passa. Um par de chave-valor pode ser 'nilo': 'egito'.
2. • Use um loop para imprimir uma frase sobre cada rio, como: O Nilo atravessa todo o Egito.
3. • Use um loop para imprimir o nome de cada rio do dicionário.
4. • Use um loop para imprimir o nome de cada país do dicionário.
5. **Pesquisa:** Use o código de linguagens favoritas.
6. • Faça uma lista de pessoas que deveriam participar da pesquisa de linguagens favoritas. Incluir alguns nomes que já estão no dicionário e outros que não estão.
7. • Percorra a lista de pessoas que devem participar da enquete. Se elas já fizeram enquete, imprima uma mensagem agradecendo a resposta. Se elas ainda não participaram da enquete, imprima uma mensagem convidando-as a participar da enquete.

# Funções Incorporadas (*Built-in*)

## A

abs()  
aiter()  
all()  
any()  
anext()  
ascii()

## B

bin()  
bool()  
breakpoint()  
bytearray()  
bytes()

## C

callable()  
chr()  
classmethod()  
compile()  
complex()

## D

delattr()

dict()

dir()

divmod()

## E

enumerate()  
eval()  
exec()

## F

filter()  
float()  
format()  
frozenset()

## G

getattr()  
globals()

## H

hasattr()

hash()

help()

hex()

## I

id()  
input()  
int()  
isinstance()  
issubclass()

iter()

## L

len()  
list()  
locals()

## M

map()  
max()  
memoryview()  
min()

## N

next()

## O

object()  
oct()  
open()  
ord()

## P

pow()  
print()  
property()

## R

range()  
repr()  
reversed()  
round()

## S

set()  
setattr()  
slice()  
sorted()  
staticmethod()  
str()  
sum()  
super()

## T

tuple()  
type()

## V

vars()

## Z

zip()

\_\_import\_\_()

# Funções Incorporadas (*Built-in*)

- Funções **globals()** e **locals()** retornam tabela de símbolos globais e locais, respectivamente. O interpretador Python mantém uma estrutura de dados contendo informações sobre cada identificador que aparece no código fonte
- Função **bytes()** retorna um objeto **bytes**, podendo converter objetos em objetos **bytes** ou criar objetos **bytes** vazios de tamanho especificado. A diferença entre **bytes()** e **bytearray()** é que a primeira retorna um objeto que não pode ser modificado e a segunda retorna um objeto que pode ser modificado
- A função **compile()** retorna o código fonte como código objeto, pronto para ser executado:  
`compile(source, filename, mode, flag, dont_inherit, optimize)`
- Ex.: `x = compile('print(55)', 'test', 'eval'); exec(x)`
- A função **eval()** avalia a expressão especificada, se ela for uma instrução Python legal, então ela será executada
- A função **divmod(a,b)** retorna o quociente e o resto da divisão inteira de **a** por **b**
- A função **frozenset()** retorna um conjunto imutável (tal como um objeto **set**, apenas imutável).

# Funções Incorporadas (*Built-in*)

- Função **hash()** retorna o valor **hash** de um objeto especificado. Os valores **hash** são apenas inteiros usados para comparar chaves de dicionário durante uma busca rápida
- Função **isinstance()** retorna **True** se o objeto for do tipo especificado, caso contrário, **False**. Se o parâmetro de **tipo** for uma tupla, então retornará **True** se o objeto for um dos tipos da tupla
- Função **iter()** retorna um objeto iterador. Um iterador é um objeto que contém um número contável de valores. Um iterador é um objeto que pode ser iterado, o que significa que você pode percorrer por todos os valores
- Função **memoryview()** retorna um objeto de visualização de memória de um dado objeto:

```
<memory at 0x14d3677cda00>
73 70 71
```
- `x = memoryview(b"IFG"); print(x)`
- `# retorna o Unicode dos caracteres`
- `print(x[0], x[1], x[2])`
- Função **ord('h')** retorna o inteiro que representa o Unicode do caractere 'h': **104**
- Função **slice(inic,fim,passo)** retorna um objeto fatia, o qual é usado para especificar como partir uma sequência

```
('b', 'c', 'd', 'e')
```
- `a = ("a", "b", "c", "d", "e", "f", "g", "h")`
- `x = slice(1,5); print(a[x])`

# Funções Incorporadas (*Built-in*)

- Função **zip()** retorna um objeto **zip**, que é um iterador de tuplas, onde o primeiro item em cada iterador passado é emparelhado e, em seguida, o segundo item em cada iterador passado é emparelhado, e assim por diante.

```
a = ("John", "Charles", "Mike")
• b = ("Jenny", "Christy", "Monica")
• x = zip(a, b)
• # usando a função tuple() para mostrar um versão legível do resultado:
• print(tuple(x))
```

```
(('John', 'Jenny'), ('Charles', 'Christy'), ('Mike', 'Monica'))
```

# Resumo

- Dicionário Simples
- Definição de Dicionário
- Operações em Dicionários
  - Acessando Valores do Dicionário
  - Adicionando Pares chave-valor ao Dicionário
  - Modificando Valores do Dicionário
  - Removendo Pares chave-valor
- Dicionário de Valores Similares
- Percorrendo os dados de um Dicionário
- Mesclando Dicionários
- Funções Incorporadas

```
FOI  << linguagens9.py x vendas.py x
1 vendas = {1: "volkswagen", 2: "chevrolet", 3: "fiat", 4: "hiunday"}
2 print(vendas.keys())
3 print(vendas.values())
4 print(vendas.items())

dict_keys([1, 2, 3, 4])
dict_values(['volkswagen', 'chevrolet', 'fiat', 'hiunday'])
dict_items([(1, 'volkswagen'), (2, 'chevrolet'), (3, 'fiat'), (4, 'hiunday')])
```