

No módulo “variáveis”, descobrimos que os textos no Python são em geral variáveis do tipo STRING. No entanto, ainda temos alguns outros conhecimentos muito importantes sobre esse tipo de variável que são fundamentais. Veja abaixo:

STRINGs no Python são listas:

Ainda não vimos em detalhes o que são listas no Python, mas por enquanto, guarde essa informação e entenda que para o Python cada caractere é um item de uma lista.

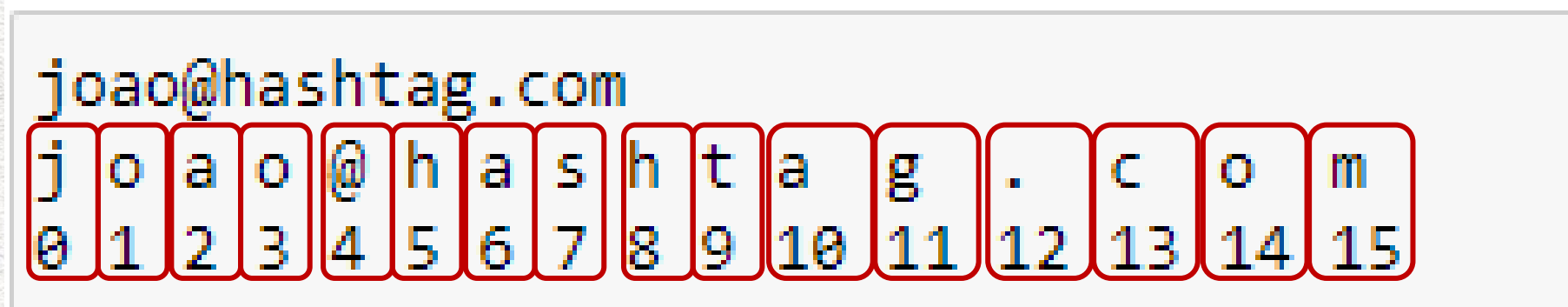


Diagrama ilustrando a string "joao@hashtag.com" tratada como uma lista de caracteres no Python. Cada caractere é armazenado em uma caixa individual, e abaixo de cada caixa há um índice numérico. Os índices variam de 0 a 15, correspondendo a cada um dos 16 caracteres da string.

j	o	a	o	@	h	a	s	h	t	a	g	.	c	o	m
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Se pegarmos um e-mail genérico 'joao@hashtag.com', podemos dividir todos os seus caracteres em uma lista.

Perceba que o primeiro caractere 'j' é o [0] e o 'm' o da posição [15].

Algo que pode gerar confusão no início, é que o número de caracteres é sempre um número a mais que a posição.

Como vemos na figura acima, como o Python inicia na posição 0, o número de caracteres sempre será 1 a mais que o número de posições.

Módulo 7 – Strings – Textos e importância no Python – Índice e tamanho da String (1/2)

Usando o mesmo exemplo, vamos agora ver como utilizar as posições no Python.

```
joao@hashtag.com
j o a o @ h a s h t a g . c o m
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

email='joao@hashtag.com'
print(email[0]) # PEGANDO A POSIÇÃO [0] do valor atribuído a variável email

j

print(email[0:5]) # PEGANDO A POSIÇÃO [0] até a posição [5]* do valor atribuído a variável email

joao@
```

Perceba, que podemos acessar qualquer posição ou range de posições através da estrutura VARIÁVEL[POSIÇÃO]!

Outro ponto de atenção é o caso de range. Ao colocarmos [0:5] não estamos pegando a posição 5. Para o Python, este intervalo se inicia na posição [0], mas finaliza na posição [4].

Módulo 7 – Strings – Textos e importância no Python – Índice e tamanho da String (2/2)

Podemos associar métodos à nossa string que facilitarão muito o tratamento desses dados. Um dos métodos mais usados é o LEN()

```
joao@hashtag.com  
j o a o @ h a s h t a g . c o m  
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

```
In [1]: email='joao@hashtag.com'  
len(email) # Fornece o tamanho da variável LEN. ATENÇÃO! CONSIDERA TODOS OS CARACTERES  
Out[1]: 16
```



ATENÇÃO!

O LEN() sempre contará TODOS os caracteres da sua STRING. Ou seja, ESPAÇO(' '), VÍRGULAS (','), PONTOS('.'), etc serão considerados!

Módulo 7 – Strings – Textos e importância no Python – Índice Negativo e Pedaco de String (1/4)

Como vimos anteriormente, o índice da posição dos caracteres segue o modelo abaixo.

joao@hashtag.com															
j	o	a	o	@	h	a	s	h	t	a	g	.	c	o	m
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Outra forma possível é a posição com índice negativo. Veja o exemplo abaixo:

joao@hashtag.com															
j	o	a	o	@	h	a	s	h	t	a	g	.	c	o	m
-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

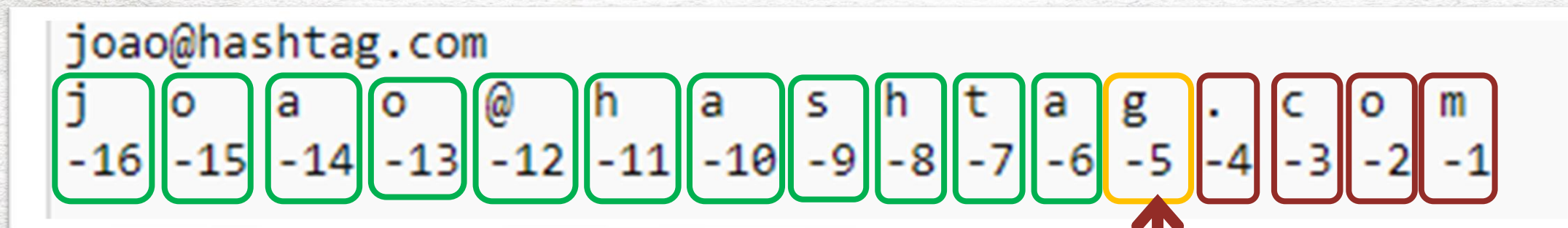
Perceba que as duas formas são válidas e coexistem no Python.
Se quisermos o caractere '@', podemos usar tanto [4] como [-12].



DICA

Em geral, vamos usar o índice negativo em casos que é sabido que o que buscamos está mais próximo do fim da string.
MAS nada impede que você use só o positivo ou só o negativo.

Módulo 7 – Strings – Textos e importância no Python – Índice Negativo e Pedaco de String (2/4)



Não incluso

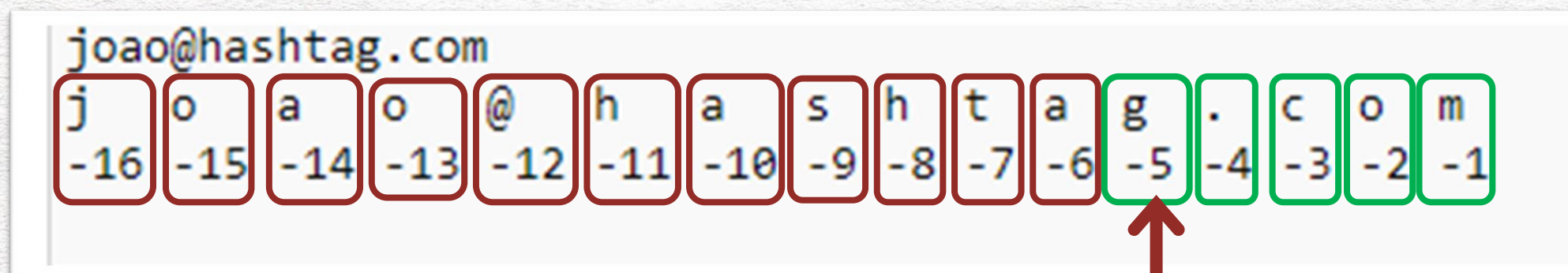
Assim como vimos nos índices positivos, podemos utilizar a mesma lógica para pegarmos pedaços da string.

Todos os dados até o caractere [-5](não incluso)

```
In [8]: email[: -5]
Out[8]: 'joao@hashta'
```

'.' Indica que se trata de um intervalo

Módulo 7 – Strings – Textos e importância no Python – Índice Negativo e Pedaco de String (3/4)



Incluso

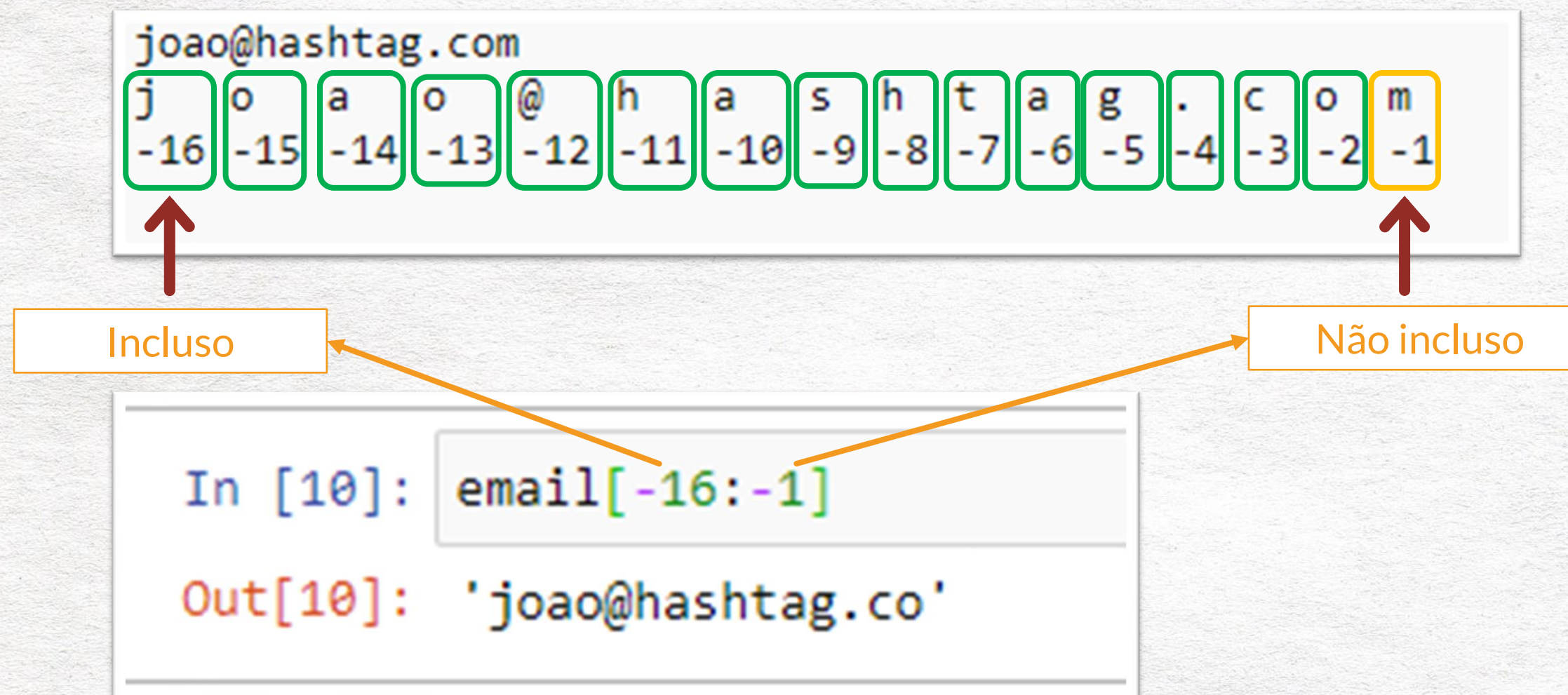
```
In [9]: email[-5:]
```

```
Out[9]: 'g.com'
```

Assim como vimos nos índices positivos, podemos utilizar a mesma lógica para pegarmos pedaços da string.

Todos os dados a partir do caractere [-5] (nesse caso incluso)

Módulo 7 – Strings – Textos e importância no Python – Índice Negativo e Pedaco de String (4/4)



Assim como vimos nos índices positivos, podemos utilizar a mesma lógica para pegarmos pedaços da string.

Todos os caracteres do índice [-16] até o [-1] (não incluso)

Módulo 7 – Strings – Textos e importância no Python – Operações com String

Vamos agora trabalhar com algumas das principais operações em string, aqui embaixo está um resumo dessas operações. E dentre as operações de concatenar ou substituir valores, a principal é o **format**, que é muito usado no **Python 3**

Resumo das Operações com String

- **str** -> transforma número em string;
- **in** -> verifica se um texto está contido dentro do outro
- **operador +** -> concatenar string
- **format e {}** -> substitui valores
- **%s** -> substitui textos
- **%d** -> substitui números decimais

Para mostrar as diferenças entre essas operações, o exemplo abaixo será usado como base.

```
faturamento = 2000
custo = 500
lucro = faturamento - custo
```



Módulo 7 – Strings – Textos e importância no Python – Operações com String

- Uso do `str()` e do concatenar com `+`

```
print ('O faturamento da loja foi de: ' + faturamento)
```

```
-----  
TypeError                                Traceback (most recent call last)  
~\AppData\Local\Temp\ipykernel_13788\2500951794.py in <module>  
----> 1 print ('O faturamento da loja foi de: ' + faturamento)  
  
TypeError: can only concatenate str (not "int") to str
```

Esse erro acontece porque só é possível concatenar string com string, e nesse caso a variável `faturamento` é do tipo `int`. Para concatenarmos essa variável, devemos transformar seu valor em string com o uso do `str()`.

```
print ('O faturamento da loja foi de: ' + str(faturamento))
```

```
O faturamento da loja foi de: 2000
```


Módulo 7 – Strings – Textos e importância no Python – Operações com String

As desvantagens desse método são:

- Ter sempre que lembrar de transformar as variáveis com o uso do `str()`
- Para textos grandes esse método se torna muito mais trabalhoso do que os outros métodos

```
print('O faturamento da loja foi de: ' + str(faturamento) + ' O custo foi de ' + str(custo) + ' O lucro foi de ' + str(lucro))
```

```
O faturamento da loja foi de: 2000 O custo foi de 500 O lucro foi de 1500
```

- Uso do **Format**

```
print('O faturamento foi de: {}'.format(faturamento))
```

```
O faturamento foi de: 2000
```

Como vimos no módulo 5, para usarmos o **format** precisamos inserir um par de chaves `{}` na posição que gostaríamos de ter as nossas variáveis ou valores substituídos. E no final do texto usamos o método **format** com os valores que queremos substituir entre parênteses, como neste exemplo de cima.

Módulo 7 – Strings – Textos e importância no Python – Operações com String

```
print('O faturamento foi de {}. O custo foi de {} e o Lucro foi de {}'.format(faturamento))
```

```
-----  
IndexError                                Traceback (most recent call last)  
~\AppData\Local\Temp\ipykernel_13468\2492006071.py in <module>  
----> 1 print('O faturamento foi de {}. O custo foi de {} e o Lucro foi de {}'.format(faturamento))  
  
IndexError: Replacement index 1 out of range for positional args tuple
```


Se tentarmos substituir apenas 1 variável em 3 chaves como no exemplo acima, será gerado um erro. Porque como temos 3 chaves, é esperado que tenha 3 informações dentro do format. Na imagem abaixo podemos ver como resolver este erro.'

```
print('O faturamento foi de {}. O custo foi de {} e o Lucro foi de {}'.format(faturamento, custo, lucro))
```

```
O faturamento foi de: 2000. O custo foi de 500 e o Lucro foi de 1500
```


Módulo 7 – Strings – Textos e importância no Python – Operações com String

Outra forma de usar o **format** é com os índices das variáveis dentro das chaves. No exemplo que estamos trabalhando, a variável **faturamento** tem **índice 0 (zero)**, a variável **custo** tem **índice 1** e a variável **lucro** tem **índice 2**.



```
print('O faturamento foi de: {0}. O custo foi de {1} e o Lucro foi de {2}'.format(faturamento, custo, lucro))
```

O faturamento foi de: 2000. O custo foi de 500 e o Lucro foi de 1500

Com isso, é possível alterar a ordem em que as variáveis aparecem.

```
print('O faturamento foi de: {2}. O custo foi de {0} e o Lucro foi de {1}'.format(faturamento, custo, lucro))
```

O faturamento foi de: 1500. O custo foi de 2000 e o Lucro foi de 500



ATENÇÃO!

Quando usar os índices das variáveis dentro das **chaves {}**, nenhuma das **chaves** pode ficar **vazia**

Módulo 7 – Strings – Textos e importância no Python – Operações com String

É possível também repetir valores dentro das **chaves {}**.

```
print('O faturamento foi de: {0}. O custo foi de {0} e o Lucro foi de {0}'.format(faturamento, custo, lucro))
```

```
O faturamento foi de: 2000. O custo foi de 2000 e o Lucro foi de 2000
```

Índice 0
(zero)

```
print('O faturamento foi de: {0}. O custo foi de {0} e o Lucro foi de {0}'.format(faturamento))
```

```
O faturamento foi de: 2000. O custo foi de 2000 e o Lucro foi de 2000
```

```
print('O faturamento foi de: {0}. O custo foi de {1} e o Lucro foi de {2}. Lembrando, o faturamento foi de {0}'.format(faturamento, custo, lucro))
```

```
O faturamento foi de: 2000. O custo foi de 500 e o Lucro foi de 1500. Lembrando, o faturamento foi de 2000
```

Índice 0
(zero)

Índice 1

Índice 2

Apesar de nos 2 últimos exemplos termos mais **chaves {}** do que variáveis dentro do **format**, isso não é um problema porque as variáveis a serem substituídas estão sinalizadas dentro das **chaves {}** através dos seus índices.

Módulo 7 – Strings – Textos e importância no Python – Operações com String

- Uso do %s e %d

Funciona de forma parecida com o format

```
print('O faturamento foi de: %s' % 'faturamento')  
O faturamento foi de: faturamento
```

Quando queremos fazer substituição por uma **string**, usamos **%s**

```
print('O faturamento foi de: %d' % faturamento)  
O faturamento foi de: 2000
```

Quando queremos fazer substituição por um **número**, usamos **%d**

```
print ('O faturamento foi de: %d. O custo foi de %d e o Lucro foi de %d' % (faturamento, custo, lucro))  
O faturamento foi de: 2000. O custo foi de 500 e o Lucro foi de 1500
```

Mesmo exemplo usado no **format**, mas agora usando **%**

O recomendado é que se use o **format**, porque o **format** é mais flexível, é a forma mais utilizada e que ficou padronizada para a versão do **Python 3**



Módulo 7 – Strings – Textos e importância no Python – Operações com String

- Uso do `in`

Verifica se um valor específico está contido em uma determinada sequência (strings, listas, etc.). Retorna **True** se o valor estiver contido na sequência, e **False** se o valor não estiver contido na sequência.

```
print('@' in 'lira@gmail.com')
```

```
True
```

Verifica se o @ está contido em **lira@gmail.com**
Retornou **True**, pois está contido.

```
print('@' in 'lira.gmail.com')
```

```
False
```

Verifica se o @ está contido em **lira.gmail.com**
Retornou **False**, pois não está contido.