O que é e como funciona list comprehension no Python

Saiba como usar iterações como for loop para criar e processar listas em Python

Gustavo Santos Follow
Feb 4 · 5 min read ★

Python oferece diferentes formas de iteração. Você pode criar um `for loop` ou `while`, por exemplo. Um dos conceitos mais comuns é o chamado *List Comprehension*, que permite você criar e processar listas de forma limpa e fácil. Nesse post você vai aprender o que é *List Comprehension* e como aplicar em Python.

Photo by Cathryn Lavery on Unsplash

O que é List Comprehension

List Comprehension nada mais é do que uma forma rápida e eficiente de criar uma lista a partir de outra lista. Por exemplo, pense em uma lista no Python: `[1,2,3,4,5]`. Agora imagine que você precisa multiplicar todos os números dessa lista por 5 e colocar todos os resultados em uma nova lista. É exatamente isso que a List Comprehension vai fazer para você.

> Para cada valor de uma lista, você aplica uma operação e coloca todos os resultados em uma nova lista. Isso é List Comprehension.

Como funciona List Comprehension

Existem algumas maneiras de resolver o problema da seção anterior, mas acredito que, se você é iniciante em Python, provavelmente pensou em usar um *loop for* ou algo parecido — como a solução abaixo:

```
# Lista
lst = [1,2,3,4,5]

# Cria nova lista para receber os valores multiplicados
mult5 = []

# Loop For para multipicar tudo por 5
for numero in lst:
    n = numero * 5
    mult5.append(n)

# Print
mult5

[Out] [5, 10, 15, 20, 25]
```

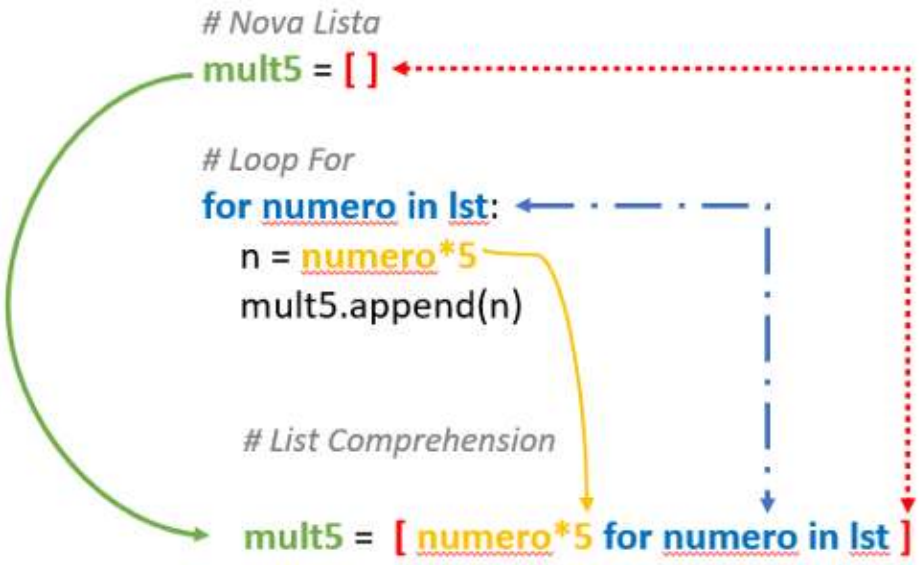
Justo. É uma boa forma de resolver o problema. Entretanto, estamos aqui para aprender uma coisa nova, certo? Queremos ver a List Comprehension em ação.

Antes, no entanto, deixe-me explicar a lógica por trás do conceito, uma vez que, se você compreender isso, vai tirar de letra o uso dela.

List Comprehension (LC) funciona basicamente como um Loop For, porém usando apenas uma linha de código para realizar a operação e deixando o seu código mais limpo e legível. A nova lista vazia criada para abrigar os valores transformados pelo Loop For agora é usada diretamente na linha de código da List Comprehension, indicando que todos os valores resultantes estão dentro da lista. A expressão do valor transformado vai na primeira parte da List Comprehension, seguido da instrução do loop for.

```
[expressão for cada elemento in lista de valores]
```

> Armazene em uma lista os valores transformados por uma expressão e faça isso para cada valor dentro de uma lista que eu indiquei.



A lógica da List Comprehension: os elementos do Loop For são rearranjados em apenas uma linha de código.

Resolvendo o nosso problema com List Comprehension:

```
# Considere a mesma lista anterior
# Multiplica por 5 com List Comprehension
mult5_lc = [numero*5 for numero in lst]

# Print
mult5_lc
```

```
[Out]:
[5, 10, 15, 20, 25]
```

Se quiser, dá pra fazer até uma função de tabuada:

```
# Tabuada com LC
def tabuada(n):
    lst = range(1,11)
    resultado = [numero*n for numero in lst]
    return (f'Resultados da Tabuada do {n}: {resultado}')

# Tabuada do 6
tabuada(6)

[Out]:
Resultados da Tabuada do 6: [6, 12, 18, 24, 30, 36, 42, 48, 54, 60]
```

List Comprehension + Condicional

Também podemos usar List Comprehension com cláusula condicional IF. Isto é: *realize a operação nos valores contidos na lista fornecida SE o valor atender a uma condição.*

Combinar o poder de ambas abre um leque ainda maior, te oferecendo flexibilidade para mudar apenas os valores que você quer.

Sintaxes possíveis

```
[expressão for cada elemento in lista if condição]

[expressão if condição else expressão for elemento in lista]
```

Exemplo:

```
# Lista
lst = [100,45,294,23904,395,3864,87495,233,2245,67,788,998,776,900]

# LC sintaxe
[x for x in lst if x < 300]

[Out]:
[100, 45, 294, 233, 67]

# If - Else
[x if x<300 else "Maior" for x in lst]
```

```
[Out]:
[100,  45,  294,  'Maior',  'Maior',  'Maior',  'Maior',  233,
'Maior',  67,  'Maior',  'Maior',  'Maior',  'Maior']
```

Quando usar List Comprehension

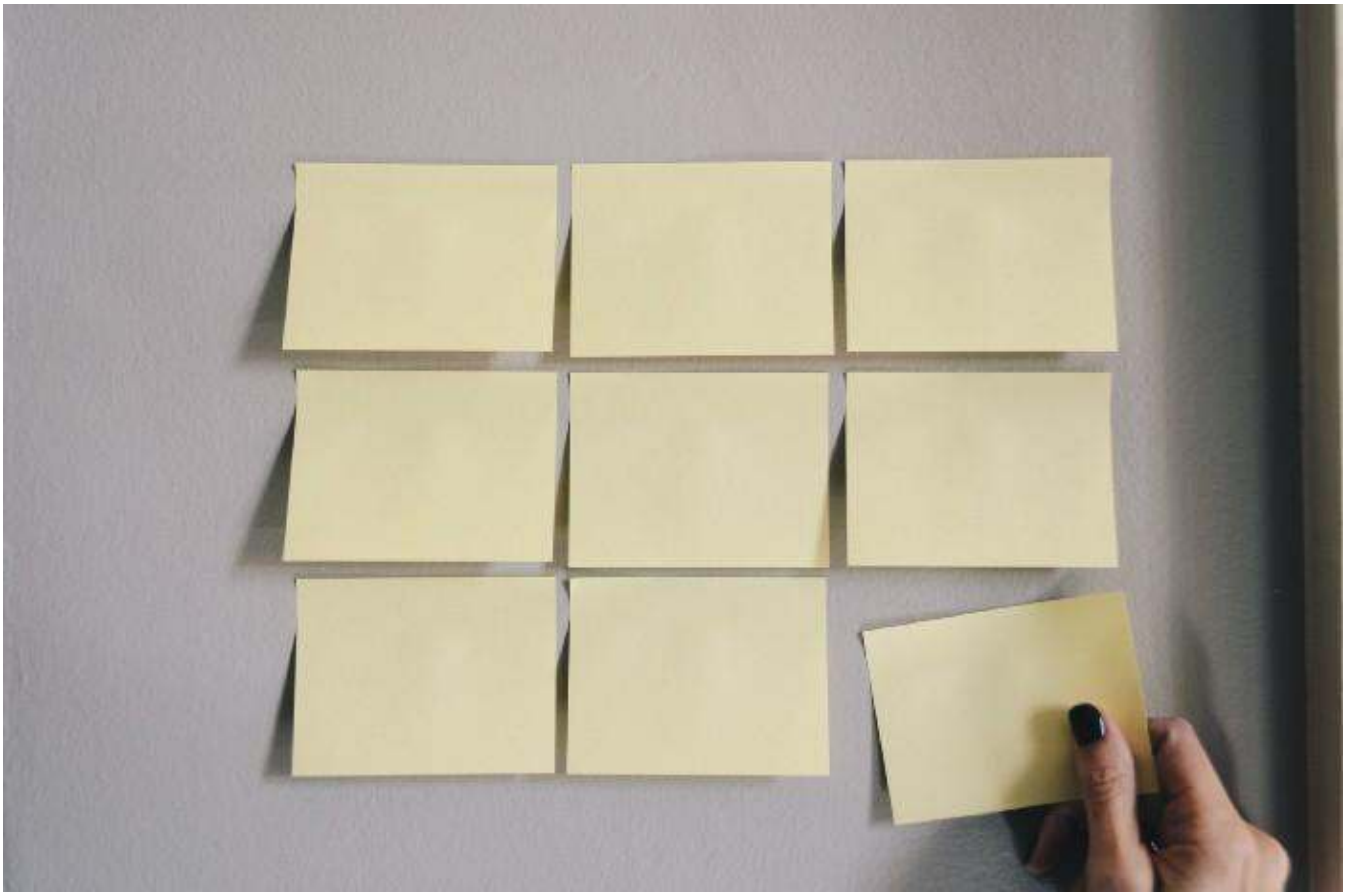


Photo by Kelly Sikkema on Unsplash

As opções de uso das List Comprehension são extensas. Imagino que você mesmo, lendo este artigo, está pensando em várias maneiras de aplicar a ferramenta nos seus projetos.

Além de ser uma ótima solução para substituir loops simples e deixar seu código mais elegante, usa-se muito as LCs em projetos de Processamento de Linguagem Natural (NLP), quando precisamos percorrer o texto palavra por palavra para remover pontuações ou para remover stopwords.

```
[palavra for palavra in tokens if palavra not in stopwords]
```

Outra situação de utilização muito interessante é para a transformação de dados em tabelas, como a criação de novas colunas de um dataframe.

```
df['high_low_tip'] = ['High' if tip > df.tip.mean() else "Low" for tip in df.tip]
```

Bônus: Dictionary Comprehension

Além de trazer as List Comprehension, o Python ainda te fornece a mesma funcionalidade com dicionários. São os *Dictionary Comprehensions*.

A ideia é a mesma das listas. Usar um loop de linha única para criar dicionários as partir de uma expressão de transformação e uma lista de números.

Sintaxe

```
{key: value for elemento in lista}
```

```
{key: value for (key, value) in dictionary.items()}
```

```
# Lista
frutas = ['banana', 'maçã', 'pera', 'tomate', 'uva']

# Dictionary Comprehension
{i:frutas[i] for i in range(len(frutas))}

[Out]:
{0: 'banana', 1: 'maçã', 2: 'pera', 3: 'tomate', 4: 'uva'}
_____

# Dictionary
d = {'a': 1, 'b':2, 'c':3}

# Dobra valor do dictionary
{k:v*2 for (k,v) in d.items()}

[Out]:
{'a': 2, 'b': 4, 'c': 6}
```

Conclusão

List Comprehensions (LC) e Dictionary Comprehension (DC) são ferramentas *buit-in* do Python e servem para substituir Loop For simples, deixando o seu código mais legível e elegante.

Em resumo:

- LC ou DC funcionam como Loop For.
- Rearranjando a sintaxe do Loop For, teremos uma LC

- Menos linhas de código
- Mais legibilidade

Código

Links Úteis

Post do Datacamp sobre List Comprehension
Mais detalhes sobre estruturas de dados em Python (item 5.1.3)
Post do Datacamp sobre Dictionary Comprehension