PYTHON ACADEMY

LIST COMPREHENSIONS (COMPREENSÃO DE LISTAS) NO PYTHON

Aprenda a dominar o poder de List Comprehensions no Python. Nesse ebook você vai ver como utilizar, declarar e manipular listas com List Comprehensions.

PYTHONACADEMY.COM.BR

Olá Pythonista!

No *post* de hoje, vamos aprender sobre uma ferramenta **muito** útil no dia a dia do Pythonista: *List Comprehensions*!

Com esse conceito, podemos otimizar a utilização de listas, sua criação e seu manuseio (e de quebra, diminuir algumas linhas de código).

Vamos ver as mais diversas formas de se utilizar *list comprehensions* e praticar com exemplos!

Ah, você sabia que a mesmo conceito pode ser aplicado aos dicionários (`dict`) do Python?

Já abre o post sobre *Dict Comprehensions* em outra aba e corre pra lá quando terminar aqui! 😉

Listas em Python

Lista é uma estrutura de dados provida pela própria linguagem e que utilizamos muito na programação Python.

Saber como manuseá-las corretamente, otimizando seu código e tirando maior proveito daquilo que o Python nos proporciona, é sempre uma **boa ideia**.

Os seguintes métodos estão disponíveis em uma lista:

- `list.append(x)`: Adiciona um item ao fim da lista.
- `list.extend(iterable)`: Adiciona todos os itens do iterável *iterable* ao fim da lista.
- `list.insert(i, x)`: Insere um item em uma dada posição **i**.

- `list.remove(x)`: Remove o primeiro elemento, cujo valor seja **x**.
- `list.pop(i)`: Remove o item de posição **i** da lista e o retorna. Caso **i** não seja especificado, retorna o último elemento da lista.
- `list.clear()`: Remove todos os elementos da lista.
- `list.index(x[, start[, end]])`: Retorna o índice do primeiro elemento cujo valor seja **x**.
- `list.count(x)`: Retorna o número de vezes que o valor **x** aparece na lista.
- `list.sort(key=None, reverse=False)`: Ordena os items da lista (os argumentos podem ser usados para customizar a ordenação).
- `list.reverse()`: Reverte os elementos da lista.
- `list.copy()`: Retorna uma lista com a cópia dos elementos da lista de origem.

Em Python, utilizamos colchetes para criação de listas. Exemplo:

```
# Apenas números
lista_numerica = [1, 2, 3, 4, 5, 6, 7, 8, 9]

# Letras e números
lista_alfanumerica = ['a', 'b', 'c', 1, 2, 3]
```

List Comprehensions (Compreensão de Listas)

List Comprehension foi concebida na PEP 202 e é uma forma concisa de criar e manipular listas.

Sua sintaxe básica é:

```
[expr for item in lista]
```

Em outras palavras: aplique a expressão `expr` em cada `item` da `lista`.

Exemplo: dado o seguinte código:

```
for item in range(10):
  lista.append(item**2)
```

Podemos reescrevê-lo, utilizando *list comprehensions*, da seguinte forma:

```
lista = [item**2 for item in range(10)]
```

Ou seja: aplique a potência de 2 em todos os itens da lista.

Outro Exemplo: dado o seguinte código, que transforma os itens da lista em maiúsculos:

```
for item in lista:
  resultado.append(str(item).upper())
```

Podemos reescrevê-lo da seguinte forma:

```
resultado = [str(item).upper() for item in lista]
```

List Comprehensions com if

List comprehensions podem utilizar expressões condicionais para criar listas ou modificar listas existentes.

Sua sintaxe básica é:

```
[expr for item in lista if cond]
```

Ou seja:

> *Aplique a expressão `expr` em cada `item` da `lista` caso a condição `cond` seja satisfeita.*

Vamos criar algumas listas utilizando condições.

Por exemplo, podemos retirar os números ímpares de um conjunto de número da seguinte forma:

```
resultado = [numero for numero in range(20) if numero % 2 == 0]
```

O que resulta em:

```
resultado = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

Vamos ver como fica com vários *if's*.

List Comprehensions com vários *if's*

Podemos verificar condições em duas listas diferentes dentro da mesma *list comprehension*.

Por exemplo: gostaríamos de saber os **Múltiplos Comuns** de 5 e 6.

Utilizando múltiplos `if's` e *list comprehensions*, podemos criar o seguinte código:

```
resultado = [numero for numero in range(100) if numero % 5 == 0 if nu-
        mero % 6 == 0]
```

Ou seja, o número só será passado para lista `resultado` caso sua divisão por 5 **E** por 6 seja igual à zero.

O resultado do código acima será:

```
resultado = [0, 30, 60, 90]
```

List Comprehensions com if + else

Outra forma de se utilizar expressões condicionais e *list comprehension* é usar o conjunto `if` + `else`.

A sintaxe básica para essa construção é:

```
[resultado_if if expr else resultado_else for item in lista]
```

Em outras palavras: para cada item da lista, aplique o resultado `resultado_if` se a expressão `expr` for verdadeira, caso contrário, aplique `resultado_else`.

Por exemplo, queremos criar uma lista que contenha “1” quando determinado número for múltiplo de 5 e “0” caso contrário.

Podemos codificá-lo da seguinte forma:

```
resultado = ['1' if numero % 5 == 0 else '0' for numero in range(16)]
```

Dessa forma, teremos o seguinte resultado:

```
resultado = ['1', '0', '0', '0', '0', '1', '0', '0', '0', '0', '1',
'0', '0', '0', '0', '1']
```

> 💡 *Estou construindo o **DevBook**, uma plataforma que usa IA para criar ebooks técnicos — com código formatado e exportação em PDF. Depois de ler, dá uma passada lá!*

Múltiplas *List Comprehensions* (aninhadas)

É aqui que a brincadeira fica **séria**!

Vamos supor que queiramos transpor uma matriz.

Pra quem não lembra o que é a **Transposição de uma Matriz**, vamos relembrar:

> *Transpor uma matriz, significa transformar as linhas em colunas e vice-versa.*

Ou seja, data a seguinte matriz:

```
matrix = [
  [1, 2, 3, 4],
  [5, 6, 7, 8],
  [9, 10, 11, 12]
]
```

Queremos o seguinte resultado:

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix} = \begin{bmatrix} 1 & 5 & 9 \\ 2 & 6 & 10 \\ 3 & 7 & 11 \\ 4 & 8 & 12 \end{bmatrix}^{\mathrm{T}}$$

Em Python, podemos fazer isso da seguinte forma:

```
transposta = []
matriz = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]

for i in range(len(matriz[0])):
    linha_transposta = []

    for linha in matriz:
        linha_transposta.append(linha[i])
    transposta.append(linha_transposta)
```

A matriz `transposta` conteria:

```
transposta = [[1, 4, 9], [2, 5, 10], [3, 6, 11], [4, 8, 12]]
```

Podemos reescrever o código acima, de transposição de matrizes, da seguinte forma, utilizando *list comprehension*:

```
matriz = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
transposta = [[linha[i] for linha in matriz] for i in range(4)]
```

No código acima:

- No **primeiro** ***loop***, `i` assume o valor de **0**, portanto `[linha[0] for linha in matriz]` vai retornar o primeiro elemento de cada linha: `[1, 4, 9]`
- No **segundo** ***loop***, `i` assume o valor de **1**, portanto `[linha[1] for linha in matriz]` vai retornar o segundo elemento de cada linha: `[2, 5, 10]`

- No **terceiro *loop***, `i` assume o valor de **2**, portanto `[linha[2] for linha in matriz]` vai retornar o terceiro elemento de cada linha: `[3, 6, 11]`
- No **quarto *loop***, `i` assume o valor de **3**, portanto `[linha[3] for linha in matriz]` vai retornar o quarto elemento de cada linha: `[4, 8, 12]`

Obtendo, assim, o mesmo resultado.

Conclusão

Nesse *post* vimos como podemos usar *list comprehensions* para criar e manipular listas de maneira concisa e eficiente.

Vimos quão poderosa essa ferramenta é e as diversas formas de utilizá-la.

Agora que você está craque em *List Comprehensions*, ***que tal começar a utilizá-lo?***

Então… **Mão na massa**! 💪 💪

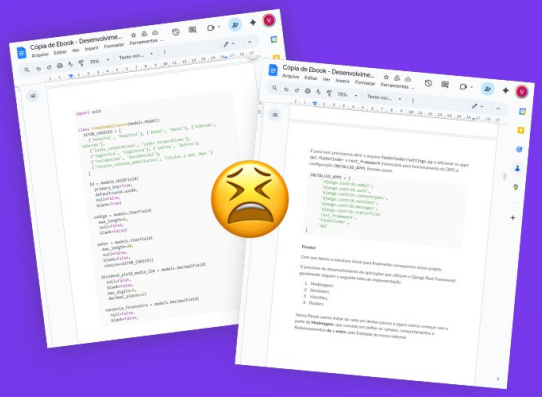
Até o próximo *post*!

Não se esqueça de conferir!

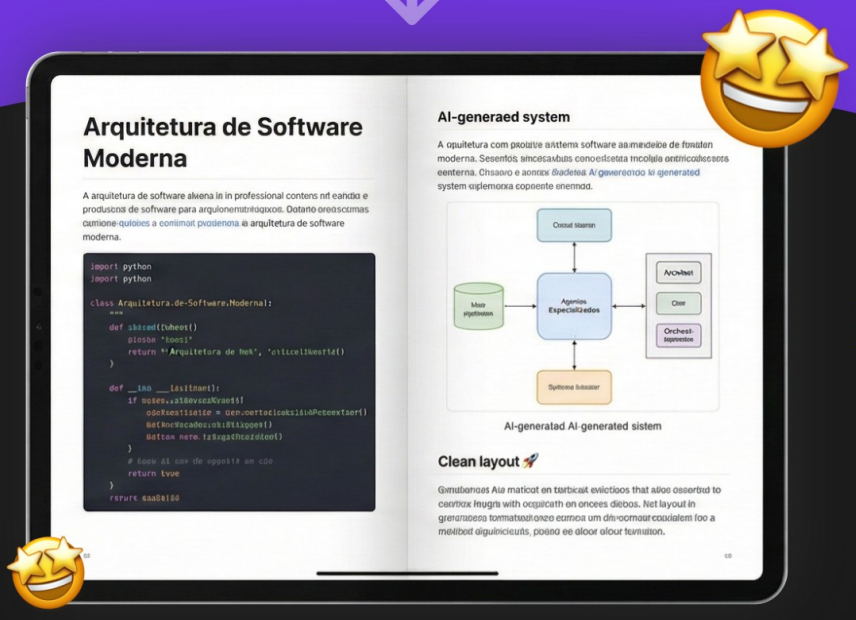
DevBook

Crie Ebooks técnicos em minutos com IA

Conheça a 1ª IA Especializada na criação de Ebooks **com código**!



Chega de formatar código no Google Docs



Syntax Highlight

Infográficos feitos por IA

Adicione Banners Promocionais

Deixe que nossa IA faça o trabalho pesado

Edite em Markdown em Tempo Real

TESTE AGORA

PRIMEIRO CAPÍTULO 100% GRÁTIS