**Key takeaways**

1. **A compreensão de lista** permite-lhe criar novas listas a partir de listas existentes de uma forma concisa e elegante. A sintaxe de uma compreensão de lista é a seguinte:

[expression for element in list if conditional]

que é na verdade um equivalente ao seguinte código:

for element in list:

if conditional:

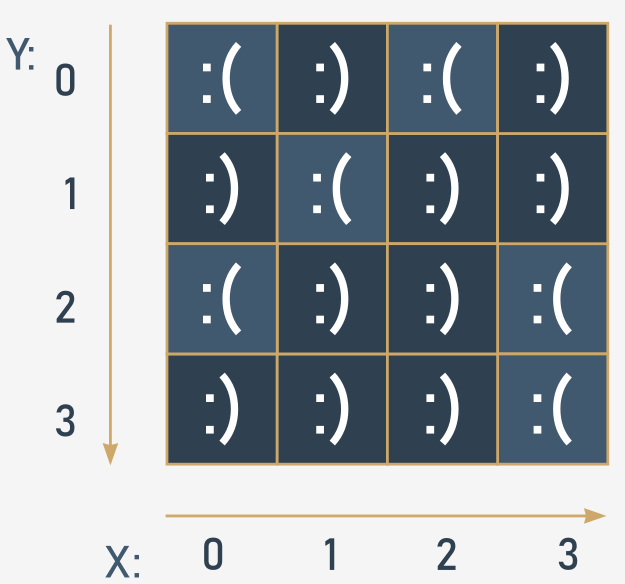
expression

Eis um exemplo de compreensão de uma lista - o código cria uma lista de cinco elementos preenchida com os primeiros cinco números naturais elevados à potência de 3:

cubed = [num \*\* 3 for num in range(5)]

print(cubed) # outputs: [0, 1, 8, 27, 64]

2. Pode usar **listas nested** em Python para criar **matrizes** (ou seja, listas bidimensionais). Por exemplo:



# A four-column/four-row table - a two dimensional array (4x4)

table = [[":(", ":)", ":(", ":)"],

[":)", ":(", ":)", ":)"],

[":(", ":)", ":)", ":("],

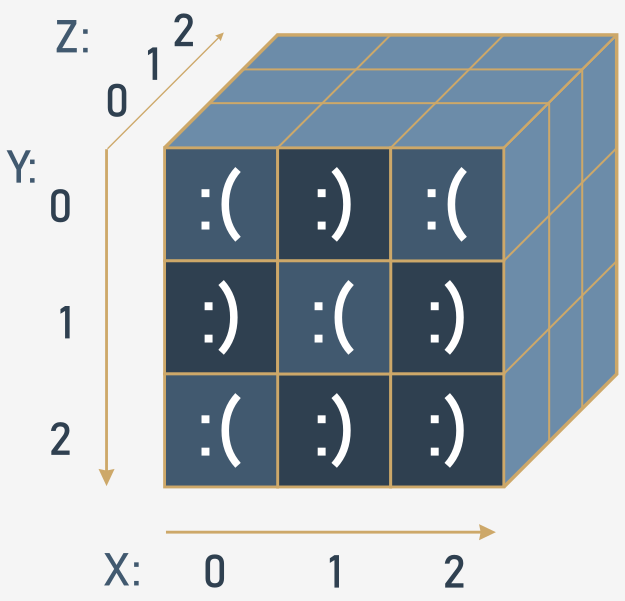
[":)", ":)", ":)", ":("]]

print(table)

print(table[0][0]) # outputs: ':('

print(table[0][3]) # outputs: ':)'

3. Pode fazer nest de quantas lists-in-lists quiser, e portanto criar listas n-dimensionais, por exemplo, três, quatro ou mesmo sessenta e quatro arrays dimensionais. Por exemplo:



# Cube - a three-dimensional array (3x3x3)

cube = [[[':(', 'x', 'x'],

[':)', 'x', 'x'],

[':(', 'x', 'x']],

[[':)', 'x', 'x'],

[':(', 'x', 'x'],

[':)', 'x', 'x']],

[[':(', 'x', 'x'],

[':)', 'x', 'x'],

[':)', 'x', 'x']]]

print(cube)

print(cube[0][0][0]) # outputs: ':('

print(cube[2][2][0]) # outputs: ':)'