Puntos Clave

1. **La comprensión de listas** te permite crear nuevas listas a partir de las existentes de una manera concisa y elegante. La sintaxis de una comprensión de lista es la siguiente:

```
[expression for element in list if conditional]
```

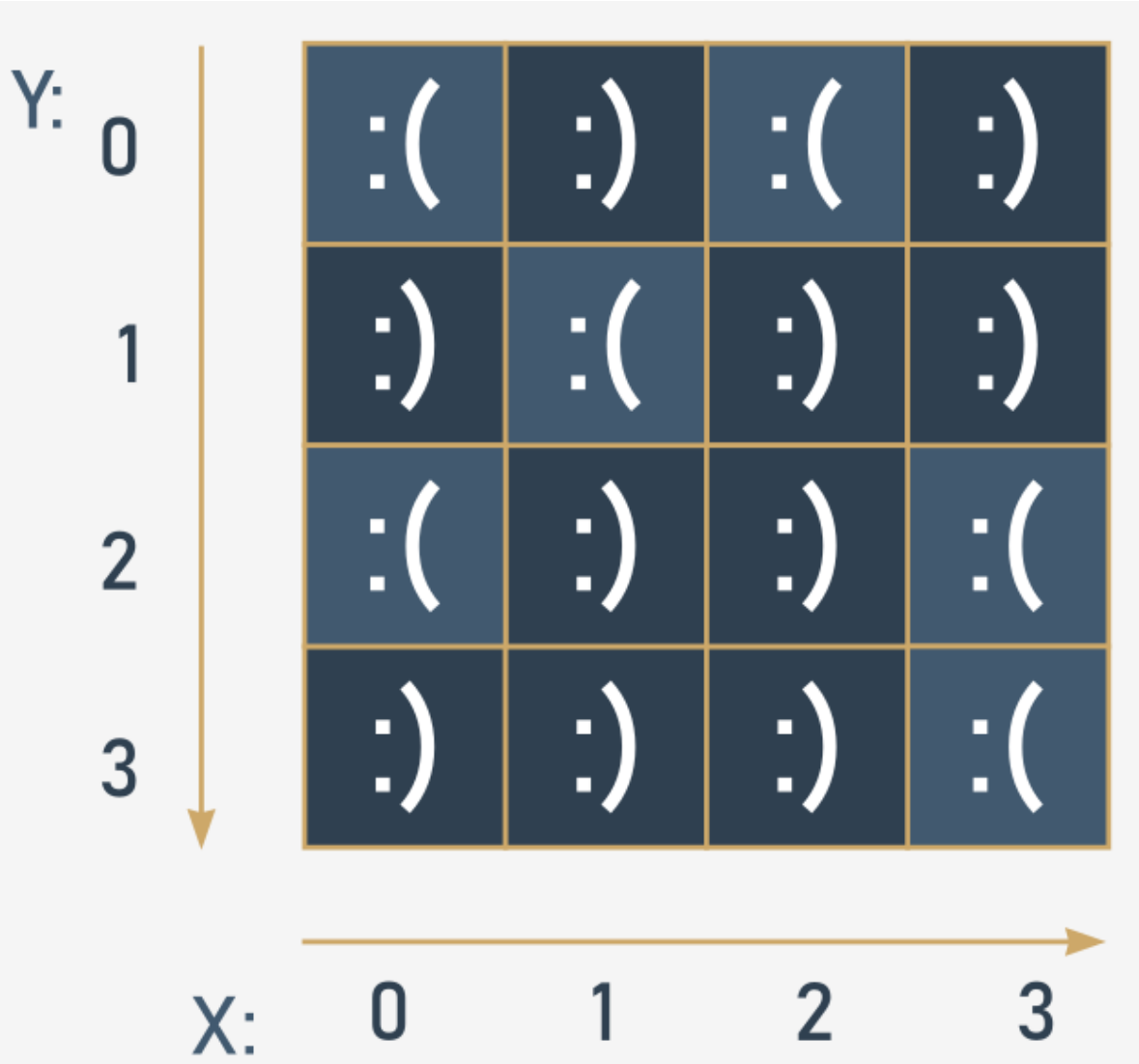
El cual es un equivalente del siguiente código:

```
for element in list:
    if conditional:
        expression
```

Este es un ejemplo de una comprensión de lista: el código siguiente crea una lista de cinco elementos con los primeros cinco números naturales elevados a la potencia de 3:

```
cubed = [num ** 3 for num in range(5)]
print(cubed)  # outputs: [0, 1, 8, 27, 64]
```

2. Puedes usar **listas anidadas** en Python para crear **matrices** (es decir, listas bidimensionales). Por ejemplo:

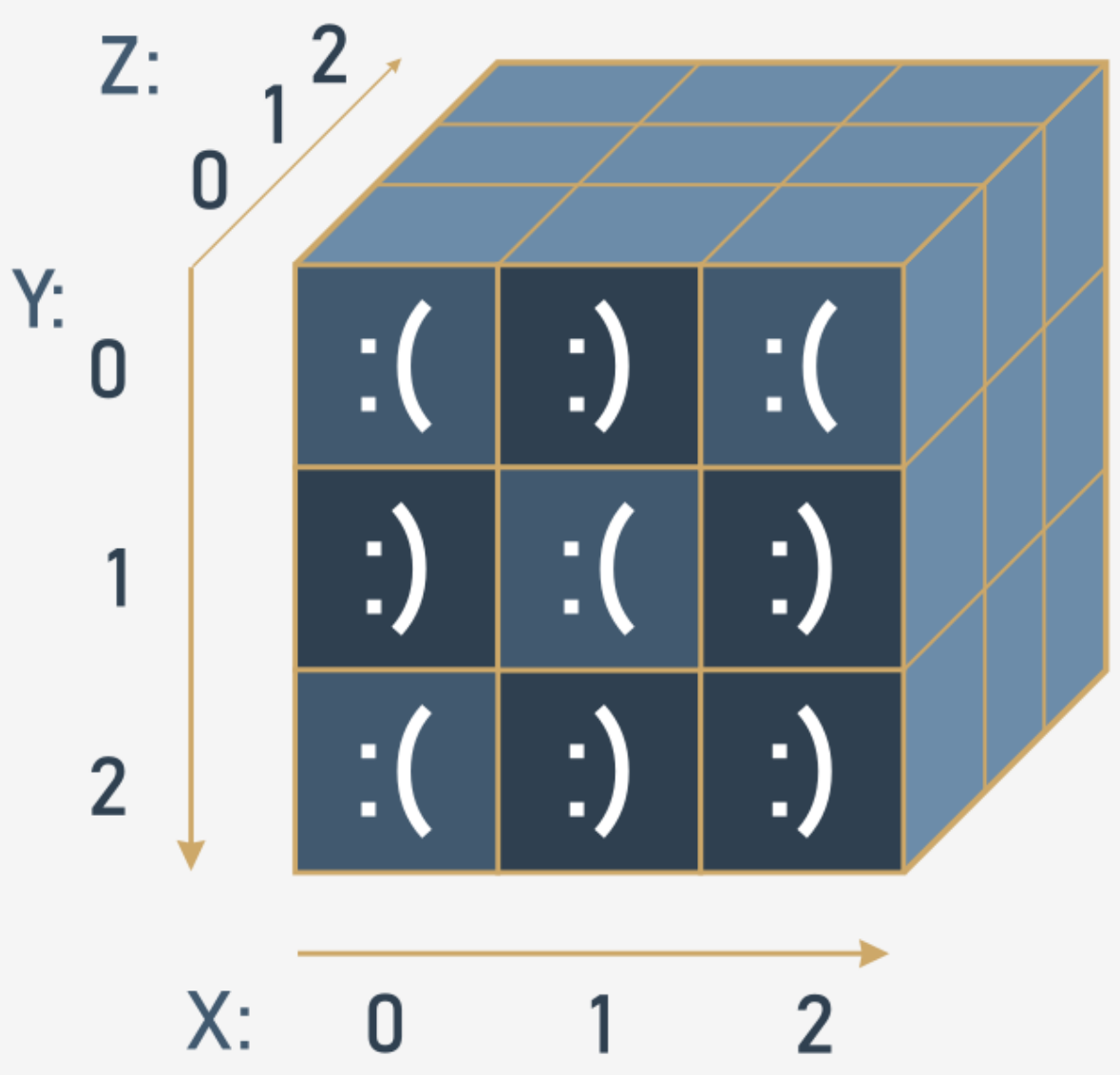



```
#  Una tabla de cuatro columnas y cuatro filas: un arreglo bidimensional (4x4)
```

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

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

3. Puedes anidar tantas listas en las listas como desee y, por lo tanto, crear listas n-dimensionales, por ejemplo, arreglos de tres, cuatro o incluso sesenta y cuatro dimensiones. Por ejemplo:




```
# Cubo - un arreglo tridimensional (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: ':)'
```