



Clase 5: Colección y Procesamiento de Datos en Python: List Comprehensions



Introducción

Las *list comprehensions* son una característica muy útil en Python, que permiten crear listas de manera concisa y eficiente. Este enfoque elimina la necesidad de escribir ciclos for largos y estructuras complejas, lo que hace que el código sea más fácil de leer y mantener. 📖

Una *list comprehension* básicamente es una forma compacta de generar listas en Python, permitiendo aplicar expresiones y filtros sobre iterables en una sola línea de código. ✨

1. ¿Qué es una List Comprehension? 🤔

Una *list comprehension* permite crear listas nuevas aplicando una expresión a cada uno de los elementos de un iterable (por ejemplo, una lista, tupla o rango), todo dentro de corchetes.

La sintaxis básica de una *list comprehension* es:

```
[ expresión for elemento in iterable ]
```

Donde:

- `expresión` es la operación o transformación que quieres aplicar a cada `elemento`.
- `iterable` es la secuencia de datos que estás recorriendo.

2. Ejemplos Prácticos de List Comprehensions 🖥️

Ejemplo 1: Cuadrados de números del 1 al 10

Vamos a encontrar los cuadrados de los números del 1 al 10 usando *list comprehensions*.

Código Tradicional:

```
cuadrados = []
for x in range(1, 11):
    cuadrados.append(x ** 2)
print(cuadrados)
```

Usando List Comprehension:

```
cuadrados = [x ** 2 for x in range(1, 11)]
print(cuadrados)
```

Salida:

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

🔍 **Análisis:** La comprensión de listas toma el número `x` de un rango de 1 a 10 y calcula su cuadrado. Este enfoque hace que el código sea más legible y corto. ✨

Ejemplo 2: Convertir grados Celsius a Fahrenheit 🌡️

Supongamos que tienes una lista de temperaturas en grados Celsius y deseas convertirlas a Fahrenheit usando la fórmula:

$$F = 9/5 \cdot C + 32$$

Código Tradicional:


```
celsius = [0, 10, 20, 30, 40]
fahrenheit = []
for temp in celsius:
    fahrenheit.append((temp * 9/5) + 32)
print(fahrenheit)
```

Usando List Comprehension:

```
celsius = [0, 10, 20, 30, 40]
fahrenheit = [(temp * 9/5) + 32 for temp in celsius]
print(fahrenheit)
```

Salida:

```
[32.0, 50.0, 68.0, 86.0, 104.0]
```

 **Tip:** Las list comprehensions también permiten usar operaciones matemáticas complejas dentro de una sola línea de código.

3. Filtrado en List Comprehensions

Las *list comprehensions* también permiten agregar una condición para filtrar los elementos que se incluirán en la nueva lista. La sintaxis para esto es:

```
[ expresión for elemento in iterable if condición ]
```

Ejemplo 3: Números pares del 1 al 20

Vamos a encontrar los números pares entre el 1 y el 20.

Código Tradicional:

```
pares = []
for x in range(1, 21):
    if x % 2 == 0:
        pares.append(x)
print(pares)
```

Usando List Comprehension:

```
pares = [x for x in range(1, 21) if x % 2 == 0]
print(pares)
```

Salida:

```
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

🧠 **Nota:** Usar `if` en una list comprehension permite filtrar elementos basados en una condición.

4. Trabajando con Matrices y List Comprehensions 📁

Es posible usar *list comprehensions* para trabajar con matrices, como encontrar su transpuesta. La transposición de una matriz consiste en convertir las filas en columnas.

Ejemplo 4: Transponer una matriz 🔄

Dada una matriz 3×3:

$$Matriz = (1, 2, 3), (4, 5, 6), (7, 8, 9)$$

Queremos transponerla usando *list comprehensions*.

Código Tradicional:


```
matriz = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
transpuesta = []
for i in range(3):
    fila = []
    for fila_original in matriz:
        fila.append(fila_original[i])
    transpuesta.append(fila)
print(transpuesta)
```

Usando List Comprehension:

```
matriz = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
transpuesta = [[fila[i] for fila in matriz] for i in range(3)]
print(transpuesta)
```

Salida:

```
[[1, 4, 7], [2, 5, 8], [3, 6, 9]]
```

 **Explicación:** La *list comprehension* anidada recorre cada fila de la matriz original, y para cada índice `i`, construye una nueva fila con los elementos correspondientes de las columnas de la matriz original.

Ejercicios Prácticos

Ejercicio 1: Números impares del 1 al 50

Problema: Crea una lista de todos los números impares entre 1 y 50 usando una *list comprehension*.

Solución:

```
impares = [x for x in range(1, 51) if x % 2 != 0]
print(impares)
```

Ejercicio 2: Calcular el precio con descuento 💰

Problema: Dada una lista de precios originales de productos, aplica un descuento del 10% y obtén una nueva lista con los precios finales.

```
precios_originales = [100, 200, 300, 400]
precios_descuento = [precio * 0.9 for precio in precios_originales]
print(precios_descuento)
```

Explicación: Aplicamos un descuento del 10% multiplicando el precio original por `0.9`.

Conclusión 🎓

Las *list comprehensions* son una herramienta poderosa en Python para crear listas de manera eficiente y elegante. No solo hacen el código más legible, sino que también mejoran el rendimiento en algunos casos. Estas técnicas son útiles tanto para principiantes como para expertos, ya que permiten realizar tareas comunes de procesamiento de datos de manera concisa.

Recuerda practicar para dominar estas estructuras, ya que son ampliamente utilizadas en el desarrollo de software y análisis de datos.

🚀 Consejos adicionales:

- Utiliza *list comprehensions* cuando quieras crear listas de forma concisa.
- No abuses de ellas cuando la expresión se vuelva demasiado compleja o difícil de leer.
- Son una excelente herramienta para manipulación de datos, filtrado y mapeo.