Lectura

Comprensiones y estructuras de construcción dinámica en Python

Python Intermedio

Introducción

Las comprensiones en Python representan una de las formas más elegantes y eficientes de construir colecciones, donde unas se forman a partir de otras, con base en expresiones claras, iteraciones y condiciones. Este recurso se enfoca en dos variantes fundamentales: las comprensiones de listas y de diccionarios, que permiten al programador transformar y filtrar datos con gran precisión y legibilidad.

List Comprehensions

Las list comprehensions son expresiones que permiten construir listas nuevas a partir de iterables existentes, en una sola línea de código. Combinan bucles y condicionales dentro de una estructura declarativa. Esta técnica no solo mejora la legibilidad, sino también la eficiencia del código.

Su sintaxis general es:

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

Un ejemplo de esto, es **ordenar los cuadrados de los números, del 1 al 10:**

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

O también **filtrar los múltiplos de 3 en una lista:**

```
[x for x in range(20) if x % 3 == 0]
```

Las **ventajas** de las list comprehensions son:

- ➔ Código más compacto.
- ➔ Mejora en la comprensión de transformaciones simples.
- ➔ Posible ganancias de rendimiento frente a bucles explícitos.

Algunas **buenas prácticas** recomendadas son:

- ➔ Usar list comprehensions solo cuando la lógica es sencilla.
- ➔ Para casos complejos o unidades, es mejor usar estructuras tradicionales (*for*, *if*, etc) para preservar la claridad.

Dict Comprehensions

Las dict comprehensions extienden el mismo principio de las listas, pero para crear diccionarios. Son útiles cuando se desea construir estructuras *clave: valor* a partir de iterables, especialmente cuando hay una relación lógica entre ambos componentes.

Su sintaxis general es:

```
{clave: valor for elemento in iterable if condición}
```

Un ejemplo de lo anterior sería **crear un diccionario con números y sus cuadrados:**

```
{n: n**2 for n in range(1, 6)}
```

O también es posible **invertir claves y valores de un diccionario**:

```
original = {'a': 1, 'b': 2, 'c': 3}
invertido = {v: k for k, v in original.items()}
```

A continuación, se presentan **ejemplos comparativos** de los 2 conceptos anteriores:

Tarea	Bucle tradicional	Comprehension

Lista de pares	pares = [ ]
	for i in range (10):
	if i % 2 == 0:
	pares.append(i)
		[i for i in range (10)
		if i % 2 == 0]

Diccionario con potencias	potencias = {} for i in range(5):potencias[i] = 2**i	{ i: 2** i for i in range (5)}

Conclusión

El uso de comprensiones en Python favorece la escritura de código más expresivo, limpio y eficiente. Dominar estas estructuras permite optimizar tareas comunes como transformación, filtrado o mapeo de datos, lo cual es fundamental en cualquier proyecto real. No obstante, deben aplicarse con criterio, evitando su uso en contextos de lógica compleja o anidamientos profundos.

Elaboró: Enrique Quezada Próspero

Contenido: Enrique Quezada Próspero

DI: Génesis Equihua

Referencias:

1. Beazley, D. M. (2022). *Python Distilled*. Addison-Wesley Professional.

2. Campesato, O. (2023). *Intermediate Python*. Mercury Learning and Information.

3. Lutz, M. (2013). *Learning Python* (5th ed.). O'Reilly Media.

4. Python Software Foundation. (2025). *Data Structures — Python 3.11.4 documentation*. Recuperado el 11 de agosto del 2025, de: https://docs.python.org/3/tutorial/datastructures.html