Compresión de listas en Python

En programación, muchas veces necesitamos **crear una nueva lista** a partir de otra existente (o de un rango de números, de los elementos de un archivo, etc.).
Lo más “tradicional” es usar un bucle `for`, donde vamos recorriendo los elementos y agregando lo que nos interesa a una lista vacía.

Por ejemplo:

```
cuadrados = []
for x in range(10):
    cuadrados.append(x**2)
```

Esto funciona bien, pero resulta **verboso**: hay que abrir la lista vacía, recorrer, usar `append()`, etc.

La solución: comprensiones de listas

Python ofrece una sintaxis especial llamada **list comprehension** (compresión de lista).
Es una forma **más compacta, clara y expresiva** de escribir ese mismo proceso en una sola línea:

```
cuadrados = [x**2 for x in range(10)]
```

Esta línea hace exactamente lo mismo que el bucle anterior:

- Recorre `range(10)` (los números del 0 al 9),
- Calcula `x**2` para cada `x`,
- Y arma una lista nueva con esos resultados.

Ventajas

1. **Claridad**: la intención del código se entiende de un vistazo: *“quiero una lista de esto, a partir de esto otro”*.
2. **Menos código repetitivo**: no hay que inicializar listas ni usar `append()`.
3. **Flexibilidad**: se pueden combinar transformaciones y condiciones en una sola expresión.
4. **Eficiencia**: suelen ser un poco más rápidas que el bucle explícito.

Estructura general

La forma básica es:

```
[EXPRESION for elemento in iterable]
```

- **EXPRESIÓN** → lo que quiero que vaya en la nueva lista (puede ser el elemento tal cual, una operación, una función aplicada, un condicional, etc.).
- **for elemento in iterable** → el bucle que recorre la estructura de datos de origen (lista, rango, string, archivo, etc.).

Opcionalmente, se pueden agregar:

- **Condiciones (`if`)** al final, para filtrar.
- **Condiciones (`if-else`)** dentro de la expresión, para elegir qué guardar.
- **Múltiples `for`** anidados, para recorrer estructuras más complejas (matrices, combinaciones de listas, etc.).

Comparación rápida

- **Bucle tradicional**:

```
resultado = []
for x in range(10):
    if x % 2 == 0:
        resultado.append(x)
```

- **Compresión de lista**:

```
resultado = [x for x in range(10) if x % 2 == 0]
```

Ambos devuelven `[0, 2, 4, 6, 8]` , pero la segunda forma es **más compacta y legible**.

Nota importante:

Las comprensiones de listas son muy útiles, pero hay que usarlas con criterio:

- Si la lógica es **muy compleja**, puede volverse difícil de leer.
- En esos casos conviene mantener el `for` tradicional con variables intermedias, aunque ocupe más líneas.
- La regla práctica: **si se lee de un tirón y se entiende, usá comprensión; si no, usá un bucle común**.

Ejemplos

Patrón general:

```
nueva_lista = [EXPRESION for elemento in iterable]
```

Ejemplo 1 — cuadrados (tu ejemplo, más explícito)

```
cuadrados = [x**2 for x in range(10)]
# [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Ejemplo 2 — transformar strings

```
palabras = ["Python", "FastAPI", "datos"]
en_minusculas = [p.lower() for p in palabras]
# ["python", "fastapi", "datos"]
```

Ejemplo 3 — aplicar una función

```
def longitud(s):
    return len(s)

longitudes = [longitud(p) for p in palabras]
# [6, 7, 5]
```

Filtro con `if` (al final)

Patrón:

```
nueva_lista = [EXPRESION for e in iterable if CONDICION]
```

Ejemplo 4 — múltiplos de 3 al cuadrado

```
cuadrados_mult3 = [x**2 for x in range(20) if x % 3 == 0]
# [0, 9, 36, 81, 144, 225, 324]
```

Ejemplo 5 — limpiar y filtrar texto

```
texto = "Hola, mundo! Esto es Python; muy, muy copado."
palabras_limpias = [
    p.strip('.,;:!?"').lower()        # transformación
    for p in texto.split()            # división en palabras
    if len(p.strip('.,;:!?"')) > 3    # filtro: más de 3 letras
]
# ["hola", "mundo", "esto", "python", "muy", "muy", "copado"]
```

Ejemplo 6 — alumnos que promocionan

```
alumnos = [
    {"nombre": "Ana", "promedio": 8.5},
    {"nombre": "Luis", "promedio": 6.9},
    {"nombre": "Sofía", "promedio": 7.0},
]
promocionan = [a["nombre"] for a in alumnos if a["promedio"] >= 7]
# ["Ana", "Sofía"]
```

Condición en la expresión (if-else “ternario”)

Patrón:

```
nueva_lista = [EXPRESION1 if CONDICION else EXPRESION2 for e in iterable]
```

> Ojo: acá el `if` va **antes** del `for`. Si además querés filtrar, podés sumar un `if` al final.

Ejemplo 7 — par/impar como etiqueta

```
etiquetas = ["par" if x % 2 == 0 else "impar" for x in range(6)]
# ["par", "impar", "par", "impar", "par", "impar"]
```

Ejemplo 8 — combinar con filtro

```
etiquetas_positivos = [
    ("par" if n % 2 == 0 else "impar")
    for n in range(-3, 4)
    if n > 0
]
# ["impar", "par", "impar"]
```

Ejemplo 9 — enriquecer diccionarios dentro de una lista

```
alumnos_con_condicion = [
    a | {"condicion": "promociona" if a["promedio"] >= 7 else "regular"}
    for a in alumnos
]
# [
#   {'nombre': 'Ana', 'promedio': 8.5, 'condicion': 'promociona'},
#   {'nombre': 'Luis', 'promedio': 6.9, 'condicion': 'regular'},
#   {'nombre': 'Sofía', 'promedio': 7.0, 'condicion': 'promociona'}
# ]
```

Varios `for` (bucles anidados)

Patrón:

```
[EXPRESION for a in A for b in B if CONDICION]
# Equivale a:
# for a in A:
#     for b in B:
#         if CONDICION: ...
```

Ejemplo 10 — producto cartesiano simple

```
letras = ["A", "B", "C"]
numeros = [1, 2, 3]
pares = [(letra, num) for letra in letras for num in numeros]
# [('A', 1), ('A', 2), ('A', 3), ('B', 1), ...]
```

Ejemplo 11 — combinaciones “válidas”

```
combinaciones_validas = [
    (l, n)
    for l in letras
    for n in numeros
    if (l != "B" or n != 2)   # filtro arbitrario
]
```

“Aplanar” listas

Ejemplo 12 — aplanar una matriz en una lista de elementos

```
matriz = [
    [1, 2, 3],
    [4, 5, 6],
]
aplanada = [elem for fila in matriz for elem in fila]
# [1, 2, 3, 4, 5, 6]
```

Ejemplo 13 — aplanar y transformar

```
aplanada_al_cuadrado = [elem**2 for fila in matriz for elem in fila]
# [1, 4, 9, 16, 25, 36]
```

Trabajar con índices (`enumerate`) y pares (`zip`)

Ejemplo 14 — índices de números negativos

```
numeros = [10, -3, 5, -1, 0]
indices_negativos = [i for i, valor in enumerate(numeros) if valor < 0]
# [1, 3]
```

Ejemplo 15 — sumar listas en paralelo con `zip`

```
a = [1, 2, 3]
b = [10, 20, 30]
sumas = [x + y for x, y in zip(a, b)]
# [11, 22, 33]
```

Matrices con compresiones

Ejemplo 16 — crear una matriz de 4x3 con ceros

> Usar comprensión evita el clásico bug de referencias compartidas (`[[0]*3]*4`).

```
filas, columnas = 4, 3
matriz_0 = [[0 for _ in range(columnas)] for _ in range(filas)]
# [
#   [0, 0, 0],
#   [0, 0, 0],
#   [0, 0, 0],
#   [0, 0, 0]
# ]
```

Ejemplo 17 — transponer una matriz

```
matriz = [
    [1, 2, 3],
    [4, 5, 6],
]
transpuesta = [[fila[i] for fila in matriz] for i in range(len(matriz[0]))]
# [
#   [1, 4],
#   [2, 5],
#   [3, 6]
# ]
```

Ejemplos prácticos “de aula” y de archivos

Ejemplo 18 — elegir apellidos de alumnos que empiezan con P

```
apellidos = ["Pérez", "Gómez", "Palazzesi", "Rojas"]
apellidos_p = [a for a in apellidos if a.startswith("P")]
# ["Pérez", "Palazzesi"]
```

Ejemplo 19 — normalizar campos (trim y minúsculas)

```
nombres_crudos = ["  Ana ", " Luis", "soFÍA  "]
nombres_ok = [n.strip().capitalize() for n in nombres_crudos]
# ["Ana", "Luis", "Sofía"]
```

Ejemplo 20 — leer líneas “no vacías” de un archivo (patrón)

> Patrón común cuando procesás archivos de texto.

```
# with open("datos.txt", encoding="utf-8") as f:
#     lineas_utiles = [ln.strip() for ln in f if ln.strip()]
# # líneas sin espacios a los costados y no vacías
```

Combinado con función: primos

```
def es_primo(n):
    if n < 2:
        return False
    # prueba simple de divisores hasta la raíz
    limite = int(n**0.5) + 1
    for d in range(2, limite):
        if n % d == 0:
            return False
    return True

primos = [n for n in range(2, 50) if es_primo(n)]
# [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
```

Comparación con `map` / `filter` (rápido)

- `map(func, iterable)` transforma; `filter(func, iterable)` filtra.
- La comprensión **lee mejor** cuando hay transformación + filtro juntos:

```
# map+filter (menos legible si hay lógica)
resultado = list(map(lambda x: x**2, filter(lambda x: x % 3 == 0, range(20))))

# comprensión (más directo)
resultado = [x**2 for x in range(20) if x % 3 == 0]
```

Buenas prácticas

- **Usá comprensiones** cuando la lógica es **simple y clara**. Si la expresión se vuelve larga/complicada, un `for` tradicional con variables intermedias suele ser más legible.
- Podés **anidar** `for` y usar **condiciones**; evitá anidar demasiado (si duele leerlo, pasalo a bucles).
- En **Python 3**, las variables del `for` en la comprensión no “se filtran” al scope exterior (no “ensucian” el entorno).

- Para **matrices**, preferí `[[0 for _ in range(m)] for _ in range(n)]` antes que `[[0]*m]*n` (este último comparte la misma sublista repetida y genera bugs al modificar).

Mini-retos

1) Sumar dígitos de cada cadena numérica

Entrada: `["123", "907", "56"]` → `[6, 16, 11]`

```
cadenas = ["123", "907", "56"]

# Convertimos cada carácter a int y sumamos por cadena
sumas = [sum(int(d) for d in s) for s in cadenas]
# [6, 16, 11]
```

Idea: comprensión externa por cadena; generador interno que convierte y suma dígitos.

2) Oraciones.

De una lista de oraciones, quedate solo con las que tienen más de 5 palabras, en minúsculas y sin signos.

```python
oraciones = [
    "Hola, esto es una prueba.",
    "Python es un lenguaje muy potente y versátil!",
    "Bienvenidos."
]
signos = '.,;:!?"¿¡()[]{}'

# Normalizamos: quitamos signos, pasamos a minúscula; contamos palabras
limpias = [
    " ".join(palabra for palabra in oracion.lower().split())
    for oracion in (
        "".join(ch for ch in o if ch not in signos)  # quitar signos
        for o in oraciones
    )
]

resultado = [o for o in limpias if len(o.split()) > 5]
# ["python es un lenguaje muy potente y versátil"]  (sin el signo original)
```

Idea: primero limpiamos, luego filtramos por cantidad de palabras.

3) “aprobado”/“desaprobado”)

Dado un `list` de dicts `{"nombre":..., "nota":...}`, devolvé una lista con `("nombre", "aprobado"/"desaprobado")` usando un `if-else` en la **expresión**.

```python
alumnos = [
    {"nombre": "Ana", "nota": 8},
    {"nombre": "Luis", "nota": 5},
    {"nombre": "Sofía", "nota": 7},
]

# if-else en la EXPRESIÓN
estado = [
    (a["nombre"], "aprobado" if a["nota"] >= 6 else "desaprobado")
    for a in alumnos
]
# [("Ana", "aprobado"), ("Luis", "desaprobado"), ("Sofía", "aprobado")]
```

Idea: if-else “ternario” dentro de la parte izquierda de la comprensión.

4) Lista de listas de strings.

Aplanar una lista de listas de strings y quedarte con los que tengan **longitud par**.

```
grupos = [
    ["sol", "luna", "mar"],
    ["cielo", "rio"],
    ["pez", "tigre"]
]

aplanados_par = [
    palabra
    for grupo in grupos
    for palabra in grupo
    if len(palabra) % 2 == 0
]
# ["luna", "mar", "cielo"]
```

Idea: doble `for` para aplanar + filtro por longitud.

5) Matriz identidad

Crear una **matriz identidad** `n×n` usando una comprensión anidada.

```
n = 5

identidad = [
    [1 if i == j else 0 for j in range(n)]
    for i in range(n)
]
# [
#  [1,0,0,0,0],
#  [0,1,0,0,0],
#  [0,0,1,0,0],
#  [0,0,0,1,0],
#  [0,0,0,0,1]
# ]
```

Idea: anidada por filas/columnas con condición diagonal.

Retos extra (con solución)

A) Normalizar una lista de nombres “crudos”

- Quitar espacios a los lados y capitalizar (Primera en mayúscula, resto minúscula).

```
nombres_crudos = ["  marIA ", "JOSe ", "  soFIA"]
nombres_ok = [n.strip().capitalize() for n in nombres_crudos]
# ["Maria", "Jose", "Sofia"]
```

B) Números perfectos en rango dado

- Un número perfecto = suma de sus divisores propios (sin incluirse) es el mismo número.

```
limite = 10000
perfectos = [
    n for n in range(2, limite + 1)
    if sum(d for d in range(1, n // 2 + 1) if n % d == 0) == n
]
# [6, 28, 496, 8128]
```

Nota: es costoso para límites grandes; didáctico para comprensión + generadores.

C) Transponer una matriz con comprensión

```
matriz = [
    [1, 2, 3],
    [4, 5, 6]
]
transpuesta = [[fila[j] for fila in matriz] for j in range(len(matriz[0]))]
# [[1, 4], [2, 5], [3, 6]]
```

D) Etiquetar números: “negativo”, “cero”, “positivo”

```
numeros = [-2, 0, 5, -1, 3]
etiquetas = [
    ("negativo" if n < 0 else ("cero" if n == 0 else "positivo"))
    for n in numeros
]
# ["negativo", "cero", "positivo", "negativo", "positivo"]
```

E) Producto cartesiano filtrado

- Pares (letra, número) donde letra ≠ “B” y número impar.

```
letras = ["A", "B", "C"]
numeros = [1, 2, 3, 4, 5]

pares = [
    (l, n)
    for l in letras
    for n in numeros
    if l != "B" and n % 2 == 1
]
# [('A', 1), ('A', 3), ('A', 5), ('C', 1), ('C', 3), ('C', 5)]
```

Ejercicios extra

1. **Sólo vocales**
   Dada una lista de palabras, crear una nueva lista con las palabras que empiezan y terminan con vocal (a, e, i, o, u). Todo en minúsculas y sin signos.
2. **Reemplazo condicional**
   Dada una lista de enteros, crear otra donde los múltiplos de 3 aparezcan como `"Fizz"`, los múltiplos de 5 como `"Buzz"`, y los múltiplos de 3 y 5 como `"FizzBuzz"`. El resto, dejar el número.
3. **Notas normalizadas**
   Dada una lista de notas en escala 0–100, crear una lista reescalada a 0–10 (con dos

decimales), usando una sola comprensión.

4. **Filtrar columnas**
   Tenés una “tabla” como lista de tuplas `(nombre, edad, ciudad)`. Crear una lista con `nombre` para quienes vivan en “Córdoba” y tengan edad ≥ 18.
5. **Aplanado con índice**
   Dada una matriz (lista de listas) de enteros, producir una lista de tuplas `(valor, fila, columna)` sólo para los valores pares.
6. **Palabras únicas ordenadas**
   A partir de un texto, obtener una lista (no repetida) de palabras en minúsculas, sin signos, **ordenadas alfabéticamente**. (Pista: puede ser útil `set` en combinación con comprensión de listas, aunque el orden lo vas a definir luego con `sorted`).
7. **Triángulo de 1s y 0s**
   Construir una matriz `n×n` que tenga 1s por encima de la diagonal (exclusiva) y 0s en el resto.
8. **Criba simple**
   Crear una lista de números primos hasta `N` usando una comprensión basada en una función `es_primo(n)` (como hicimos antes).
9. **Distancias**
   Dados dos vectores `a` y `b` de igual longitud, obtener la lista de diferencias absolutas elemento a elemento, y luego su suma total (en una línea por cada cosa).
10. **CSV casero**
    Dada una lista de filas (listas de strings) representar cada fila como una línea CSV uniendo con comas y escapando comas internas con comillas (mínimo viable). Hacer una lista con todas las líneas.