21 – Generator Expressions

Le **generator expressions** sono simili alle list comprehension, ma **non creano una lista completa in memoria**, bensì un **oggetto generatore** che produce gli elementi **su richiesta** (*lazy evaluation*). Sono utili per risparmiare memoria quando si lavora con grandi sequenze o calcoli complessi.

Sintassi

```
(espressione for elemento in iterabile if condizione)
```

- Si usano **parentesi tonde** invece delle quadre delle list comprehension.
- Restituiscono un **generator**, che può essere iterato con un ciclo `for` o convertito in lista, tupla, set ecc.

Esempio base

Creare un generatore dei quadrati dei numeri da 1 a 5:

```
gen = (x**2 for x in range(1, 6))
print(gen)  # <generator object ...>

for val in gen:
    print(val)
# Output: 1 4 9 16 25
```

Conversione in lista

Se vuoi ottenere tutti gli elementi in una volta:

```
gen = (x**2 for x in range(1, 6))
quadrati = list(gen)
print(quadrati)  # [1, 4, 9, 16, 25]
```

Con condizione

Si possono filtrare gli elementi come nelle list comprehension:

```
numeri = range(10)
pari_quadrati = (x**2 for x in numeri if x % 2 == 0)

for val in pari_quadrati:
    print(val)
# Output: 0 4 16 36 64
```

Vantaggi

- **Risparmio di memoria**, perché gli elementi vengono generati uno alla volta.
- Ideale per **streaming di dati** o sequenze molto grandi.
- Può essere combinato con funzioni come `sum()`, `max()`, `min()` senza creare una lista intermedia.

Nota

Dopo aver iterato un generatore, gli elementi **non sono più disponibili** a meno di ricrearlo.
Per accesso ripetuto, conviene trasformarlo in lista o altra struttura.

Vuoi procedere con il **paragrafo 22 – Unpacking Operator**?