18. List comprehension va generatorlar



Ro'yxatni sikl va append bilan yasash — eng ko'p uchraydigan ish. Python uni bitta qatorda yozish imkonini beradi: **list comprehension**. Bu Python dasturchilari eng sevgan imkoniyatlardan biri.

1. Oddiy comprehension

Shakli: `[ifoda for element in ketma-ketlik]`. Quyidagi ikki kod bir xil natija beradi:

```
kvadratlar = []
for x in range(1, 6):
    kvadratlar.append(x * x)
print(kvadratlar)

kvadratlar = [x * x for x in range(1, 6)]
print(kvadratlar)
```

Natija:

```
[1, 4, 9, 16, 25]
[1, 4, 9, 16, 25]

sozlar = ["olma", "nok", "anor"]
print([s.upper() for s in sozlar])
print([len(s) for s in sozlar])
print([int(c) for c in "2026"])
```

Natija:

```
['OLMA', 'NOK', 'ANOR']
[4, 3, 4]
[2, 0, 2, 6]
```

2. Shart bilan: filtrlash

`[ifoda for x in … if shart]` — faqat shart bajarilgan elementlar kiradi:

```
sonlar = [3, -1, 8, 0, -5, 12, 7]
print([x for x in sonlar if x > 0])
print([x * x for x in sonlar if x % 2 == 0])
print([s for s in ["Ali", "Malika", "Bo", "Sardor"] if len(s) > 3])
```

Natija:

```
[3, 8, 12, 7]
[64, 0, 144]
['Malika', 'Sardor']
```

Har bir elementni ikki xil o'zgartirish kerak bo'lsa — shart ifoda qismida, `if … else` ko'rinishida:

```
sonlar = [3, -1, 8, -5]
print([x if x > 0 else 0 for x in sonlar])
print(["juft" if x % 2 == 0 else "toq" for x in range(1, 6)])
```

Natija:

```
[3, 0, 8, 0]
['toq', 'juft', 'toq', 'juft', 'toq']
```

3. Ichma-ich comprehension

```
jadval = [[i * j for j in range(1, 4)] for i in range(1, 4)]
print(jadval)
matritsa = [[1, 2], [3, 4], [5, 6]]
print([x for qator in matritsa for x in qator])
print([(a, b) for a in "ab" for b in [1, 2]])
```

Natija:

```
[[1, 2, 3], [2, 4, 6], [3, 6, 9]]
[1, 2, 3, 4, 5, 6]
[('a', 1), ('a', 2), ('b', 1), ('b', 2)]
```

4. Lug'at va to'plam comprehension

```
ismlar = ["Ali", "Malika", "Sardor"]
print({ism: len(ism) for ism in ismlar})
ballar = {"Ali": 78, "Malika": 95, "Sardor": 52}
print({k: v for k, v in ballar.items() if v >= 60})
```

```
print({x % 3 for x in range(10)})
```

Natija:

```
{'Ali': 3, 'Malika': 6, 'Sardor': 6}
{'Ali': 78, 'Malika': 95}
{0, 1, 2}
```

5. Generator ifoda — xotirani tejash

Kvadrat qavs o'rniga oddiy qavs — **generator ifoda**. U butun ro'yxatni xotirada yaratmaydi, elementlarni birma-bir, kerak bo'lganda hisoblaydi. sum, max, any kabi funksiyalarga to'g'ridan-to'g'ri berish mumkin:

```
print(sum(x * x for x in range(1, 1001)))
print(max(len(s) for s in ["olma", "shaftoli", "nok"]))
gen = (x * 2 for x in range(3))
print(next(gen), next(gen), next(gen))
```

Natija:

```
333833500
8
0 2 4
```

```
import sys
royxat = [x for x in range(100000)]
gen = (x for x in range(100000))
print("ro'yxat:", sys.getsizeof(royxat) > 100000, "| generator:", sys.getsizeo
```

Natija:

```
ro'yxat: True | generator: True
```

6. yield — generator funksiya

`return` o'rniga `yield` ishlatilgan funksiya — **generator**. Har `yield` da u qiymat beradi va «pauza» qiladi; keyingi so'rovda o'sha joydan davom etadi:

```
def sanoq(n):
    i = 1
    while i <= n:
        yield i
        i += 1

for x in sanoq(4):
    print(x, end=" ")
print()
print(list(sanoq(6)))
```

Natija:

```
1 2 3 4
[1, 2, 3, 4, 5, 6]
```

```
def fibonachchi():
    a, b = 0, 1
    while True:          # cheksiz — lekin faqat so'ralganicha hisoblanadi
        yield a
        a, b = b, a + b

f = fibonachchi()
print([next(f) for _ in range(12)])
```

Natija:

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

> **Eslatma**
>
> Cheksiz generatordan `list()` qilmang — u hech qachon tugamaydi. Kerakli miqdorni `next()` yoki sikl ichida `break` bilan oling.

7. Qachon nimani ishlatish

Vaziyat	Tanlov
Natija ro'yxat bo'lishi kerak	`[... for ...]`
Faqat bir marta yurib chiqiladi (sum, max)	`(... for ...)` generator ifoda
Juda katta yoki cheksiz ketma-ketlik	`yield` bilan generator funksiya
Murakkab mantiq (ko'p shart, xato ushlash)	Oddiy `for` sikl — o'qish osonroq

8. Xulosa

- `[ifoda for x in a]`, filtr — `if shart` oxirida, almashtirish — `a if s else b` boshida.
- Lug'at `{k: v for …}`, to'plam `{x for …}`.
- Generator ifoda `(… for …)` va `yield` — xotirani tejaydi, qiymatlar kerak bo'lganda hisoblanadi.