Python — Liste e ciclo for

Marini Mattia

November 08, 2026

1 Liste

Cos'è una lista

Una **lista** è una sequenza **ordinata** e **modificabile** di valori.

```
voti = [6, 7, 8]
vuota = []
mista = [1, "due", 3.0, True]
```

- l'**ordine** conta: `[6, 7]` è diversa da `[7, 6]`;
- i **duplicati** sono ammessi;
- si accede per **posizione** (indice);
- si può **modificare**: aggiungere, togliere, cambiare elementi.

Accedere agli elementi

```
voti = [6, 7, 8, 9]
voti[0]     # 6    primo elemento
voti[1]     # 7    secondo
voti[3]     # 9    quarto
voti[-1]    # 9    ultimo
voti[-2]    # 8    penultimo
```

> **N.B.** Gli indici partono da `0`. In una lista di `n` elementi gli indici validi vanno da `0` a `n - 1` (oppure da `-1` a `-n`).

Modificare e misurare

```
voti[1] = 10      # cambia il secondo elemento
len(voti)         # 4 → quanti elementi
```

Aggiungere elementi

```
voti = [6, 7, 8]
voti.append(9)        # [6, 7, 8, 9]        in fondo
voti.insert(1, 10)    # [6, 10, 7, 8, 9]    in posizione 1
```

- `append(x)` aggiunge **un** elemento in fondo;
- `insert(i, x)` inserisce `x` **prima** della posizione `i`;
- entrambe modificano la lista e **non restituiscono niente** (`None`).

Togliere elementi

```
voti = [6, 7, 8, 7]
voti.remove(7)    # [6, 8, 7]   toglie il PRIMO 7 (per valore)
voti.pop()        # [6, 8]      toglie e restituisce l'ultimo
voti.pop(0)       # [8]         toglie e restituisce quello in posizione 0
del voti[0]       # []          toglie per posizione, senza restituire
```

- `remove(x)` cerca **per valore** → `ValueError` se non c'è;
- `pop(i)` lavora **per posizione**;
- `del` non restituisce il valore rimosso.

Cercare

```
7 in voti          # True / False (sicuro)
voti.index(7)      # posizione della prima occorrenza
voti.count(7)      # quante volte compare
```

- `in` è il modo **sicuro** per chiedersi "c'è?";
- `index(x)` dà un `ValueError` se `x` non è presente.

Funzioni utili

```
numeri = [3, 8, 5, 10]
sum(numeri)       # 26
max(numeri)       # 10
min(numeri)       # 3
sorted(numeri)    # [3, 5, 8, 10]  NUOVA lista
numeri.sort()     # ordina sul posto, restituisce None
numeri.reverse()  # inverte sul posto
```

> **N.B.** `sorted()` crea una **copia** ordinata; `sort()` modifica la lista originale e restituisce `None`.

2 Ciclo for

Sintassi

```
for variabile in sequenza:
    # corpo: ripetuto una volta per ogni elemento
```

- `for` + **nome della variabile** + `in` + **sequenza** + `due punti :`;
- il corpo è **indentato** (1 tab);
- a ogni giro `variabile` assume il valore dell'elemento successivo;
- il numero di giri è **già deciso** dalla lunghezza della sequenza.

for su una lista

```
voti = [6, 7, 8]
for v in voti:
    print(v)        # 6 7 8
```

`v` riceve il **valore** dell'elemento: riassegnare `v` **non** cambia la lista.

```
for v in voti:
    v = v + 1       # ✗ la lista resta [6, 7, 8]
```

for su una stringa

```
for c in "ciao":
    print(c)        # c i a o
```

range() — 1 parametro

```
range(n)        # 0, 1, 2, ..., n-1
for i in range(4):
    print(i)    # 0 1 2 3
```

Parte da `0`, **esclude** `n`: produce esattamente `n` valori.

range() — 2 parametri

```
range(a, b)     # a, a+1, ..., b-1
for i in range(2, 6):
    print(i)    # 2 3 4 5
```

`a` è **incluso**, `b` è **escluso**.

range() — 3 parametri

```
range(a, b, passo)    # da a a b-1, saltando di "passo"
for i in range(0, 10, 2):
    print(i)    # 0 2 4 6 8

for i in range(5, 0, -1):
    print(i)    # 5 4 3 2 1
```

- con passo **negativo** si conta all'indietro;
- l'estremo `b` resta sempre **escluso**;
- se il passo va nella direzione sbagliata, il ciclo **non fa nessun giro** (`range(0, 5, -1)` è vuoto).

for con indice

```
voti = [6, 7, 8]
for i in range(len(voti)):
    print(i, voti[i])    # 0 6 | 1 7 | 2 8
```

`len(voti) = 3` → `range(3)` = `0 1 2`, cioè **tutti gli indici validi**. Serve quando bisogna **modificare** gli elementi.

```
for i in range(len(voti)):
    voti[i] = voti[i] + 1    # ✓ ora la lista cambia
```

enumerate()

```
for i, v in enumerate(voti):
    print(i, v)    # 0 6 | 1 7 | 2 8
```

Scorciatoia per avere **indice e valore** insieme, senza `range(len(...))`.

3 Esempi ultra comuni

Scambiare due variabili (swap)

```
a = 1
b = 2
a, b = b, a
print(a, b)      # 2 1
```

In Python non serve una variabile temporanea: la tupla a destra viene valutata **prima** delle assegnazioni.

```
temp = a         # versione "classica", equivalente
a = b
b = temp
```

Controllare se un numero è pari

`%` è il **resto** della divisione: `n % 2` vale `0` se `n` è pari.

```
n = 8
if n % 2 == 0:
    print("pari")
else:
    print("dispari")
```

Scorrere tutti i numeri da 1 a n

```
n = 5
for i in range(1, n + 1):
    print(i)      # 1 2 3 4 5
```

> **N.B.** `n + 1` perché l'estremo destro di `range()` è **escluso**. Con `range(1, n)` ti fermeresti a `n - 1`.

Scorrere una lista con due indici

Un indice `i` parte da **sinistra**, uno `j` da **destra**: si avvicinano fino a incontrarsi.

```
v = [1, 2, 3, 4, 5]
i = 0
j = len(v) - 1
while i < j:
    v[i], v[j] = v[j], v[i]    # scambia gli estremi
    i = i + 1
    j = j - 1
print(v)      # [5, 4, 3, 2, 1]
```

Con un `for` si fanno **metà** dei giri e `j` si calcola da `i`:

```
v = [1, 2, 3, 4, 5]
for i in range(len(v) // 2):
    j = len(v) - 1 - i
    v[i], v[j] = v[j], v[i]
```

> **N.B.** `len(v) // 2` e non `len(v) / 2`: `range()` vuole un **intero**, e `/` produce un `float`.

Pattern: accumulatore

Una variabile inizializzata **prima** del ciclo e aggiornata **dentro**.

```
numeri = [3, 8, 5, 10]

somma = 0
for n in numeri:
    somma = somma + n
print(somma)          # 26

massimo = numeri[0]
for n in numeri:
    if n > massimo:
        massimo = n
print(massimo)        # 10
```

Pattern: costruire una lista filtrata

```
numeri = [3, 8, 5, 10]
pari = []
for n in numeri:
    if n % 2 == 0:
        pari.append(n)
print(pari)           # [8, 10]
```

4 Errori ultra comuni

Indice fuori range → IndexError

```
voti = [6, 7, 8]
voti[3]     # ✗ IndexError: list index out of range
voti[2]     # ✓ ultimo elemento
voti[-1]    # ✓ ultimo elemento
```

Con `n` elementi l'ultimo indice è `n - 1`, **non** `n`.

Modificare la lista mentre la scorri

```
for v in voti:
    if v < 6:
        voti.remove(v)    # ✗ salta elementi
```

Togliere elementi **sposta** gli indici successivi e il `for` va avanti lo stesso: alcuni elementi non vengono controllati. Usa una **nuova lista**:

```
ok = []
for v in voti:
    if v >= 6:
        ok.append(v)      # ✓
```

append() vs extend()

```
v = [1, 2]
v.append([3, 4])    # [1, 2, [3, 4]]   una lista dentro la lista

v = [1, 2]
v.extend([3, 4])    # [1, 2, 3, 4]     gli elementi uno per uno
```

`append()` aggiunge **un solo** elemento: se è una lista, resta una lista annidata.

`sort()` restituisce None

```
v = [3, 1, 2]
v = v.sort()        # ✗ v diventa None
v.sort()            # ✓ ordina sul posto
ordinata = sorted(v)  # ✓ nuova lista ordinata
```

Vale anche per `append()`, `insert()`, `remove()`, `reverse()`: modificano la lista e **non restituiscono** la lista.

remove() per valore, pop() per posizione

```
voti = [6, 7, 8]
voti.remove(1)    # ✗ ValueError: 1 non è un valore della lista
voti.pop(1)       # ✓ toglie l'elemento in posizione 1 → 7
```

range(): estremo escluso e passo

```
range(5)          # 0 1 2 3 4      (non 5!)
range(1, 5)       # 1 2 3 4        (non 5!)
range(0, 5, -1)   # vuoto: passo negativo ma estremi crescenti
range(5, 0, 1)    # vuoto: passo positivo ma estremi decrescenti
```

Chiediti sempre: l'estremo destro è **incluso**? E il passo va nella **direzione giusta**?

range() vuole un intero

```
range(len(v) / 2)     # ✗ TypeError: / produce un float
range(len(v) // 2)    # ✓ // produce un int
```

Dimenticare i due punti `:` o l'indentazione

```
for i in range(3)     # ✗ SyntaxError: expected ':'
    print(i)

for i in range(3):
print(i)              # ✗ IndentationError
```

Vale per `for`, `if`, `while`, `def`.

Altri classici

- **Modificare `v` nel `for`**: `for v in lista: v = v + 1` non tocca la lista.
- **Usare `i` fuori dal ciclo**: dopo il `for` la variabile esiste ancora e vale l'**ultimo** valore.
- **Nome sbagliato**: `lista.append(5)` → `AttributeError` (attenzione a `append`).
- **`=` invece di `==`** dentro un `if`: `if n % 2 = 0:` → `SyntaxError`.
- **Lista vuota**: `max([])` e `lista[0]` danno errore; controlla prima con `len(lista) > 0`.
- **Confondere lista nuova e stessa lista**: `b = a` non copia, `b = a.copy()` (o `a[:]`) sì.