Liste e tuple

Cos'è una lista?

Una lista è tipo lo **zaino della scuola**: ci infili dentro roba di tutti i tipi e puoi tirare fuori quello che ti serve quando vuoi. È una **collezione ordinata di elementi**, dove ogni elemento ha la sua posizione (indice).

```
# Lista di numeri (tipo i voti del registro)
voti = [8, 7, 9, 6, 8, 10]
print(voti)

# Lista di stringhe (i personaggi di Mario)
nomi = ["Mario", "Luigi", "Peach"]
print(nomi)

# Lista mista (si può, ma di solito è una cattiva idea)
mista = [42, "ciao", True, 3.14]
print(mista)

# Lista vuota (pronta per essere riempita!)
vuota = []
print(vuota)
```

Accesso agli elementi

Come per le stringhe, gli indici partono da **0**. Lo so, ci sei già passato, ormai dovresti esserci abituato!

```
frutta = ["mela", "banana", "arancia", "kiwi", "pera"]

print(frutta[0])    # Primo: mela
print(frutta[2])    # Terzo: arancia
print(frutta[-1])   # Ultimo: pera
print(frutta[-2])   # Penultimo: kiwi
```

Slicing (affettare la lista come una pizza)

Funziona esattamente come con le stringhe. Se hai capito quello, sei già a cavallo:

```
numeri = [10, 20, 30, 40, 50, 60, 70]

print(numeri[1:4])    # [20, 30, 40]
print(numeri[:3])     # [10, 20, 30]
print(numeri[4:])     # [50, 60, 70]
print(numeri[::2])    # [10, 30, 50, 70] (uno sì, uno no)
print(numeri[::-1])   # [70, 60, 50, 40, 30, 20, 10] (al contrario!)
```

Modificare una lista

Ecco la differenza GROSSA rispetto alle stringhe: le liste **sono mutabili**! Puoi cambiare, aggiungere, togliere elementi come ti pare. Sono tipo la plastilina del coding:

```
colori = ["rosso", "verde", "blu"]
print("Prima:", colori)

# Modificare un elemento (swap diretto!)
colori[1] = "giallo"
print("Dopo modifica:", colori)

# Aggiungere in fondo con append()
colori.append("viola")
print("Dopo append:", colori)

# Inserire in una posizione specifica
colori.insert(1, "arancione")
print("Dopo insert:", colori)
```

Rimuovere elementi

Tanti modi per buttare fuori roba dalla lista. Scegli il tuo preferito:

```
animali = ["gatto", "cane", "pesce", "coniglio", "cane"]

# remove() — toglie per valore (solo la PRIMA occorrenza!)
animali.remove("cane")
print("Dopo remove:", animali)

# pop() — toglie per posizione e ti dice cosa ha tolto
rimosso = animali.pop(1)
print(f"Rimosso: {rimosso}")
print("Dopo pop:", animali)

# pop() senza argomenti — toglie l'ultimo
ultimo = animali.pop()
print(f"Ultimo rimosso: {ultimo}")
print("Lista finale:", animali)
```

Operazioni sulle liste

Le liste hanno un sacco di **superpoteri integrati**. Tipo un coltellino svizzero:

```
numeri = [3, 1, 4, 1, 5, 9, 2, 6]

print("Lunghezza:", len(numeri))
print("Somma:", sum(numeri))
print("Minimo:", min(numeri))
print("Massimo:", max(numeri))

# Ordinamento (modifica la lista originale!)
numeri.sort()
print("Ordinata:", numeri)

numeri.sort(reverse=True)
print("Ordine inverso:", numeri)

# Contare quante volte appare un elemento
print("Quanti 1:", numeri.count(1))

# Verificare se un elemento c'è
print("5 è presente?", 5 in numeri)
print("7 è presente?", 7 in numeri)
```

Iterare sulle liste

Con `for` (il modo classico)

```
frutta = ["mela", "banana", "arancia"]

for f in frutta:
    print(f"Mi piace la {f}")
```

Con `enumerate()` — quando ti serve anche l'indice

`enumerate()` è tipo il biglietto con il numero al deli: ti dà sia la posizione che il valore!

```
frutta = ["mela", "banana", "arancia"]

for i, f in enumerate(frutta):
    print(f"{i}: {f}")
```

List comprehension

Questo è il **trucco ninja** di Python: creare liste in una sola riga. La prima volta sembra magia nera, poi non ne puoi più fare a meno:

```
# Modo classico (noioso ma chiaro)
quadrati = []
for i in range(1, 6):
    quadrati.append(i ** 2)
print("Classico:", quadrati)

# List comprehension (stessa cosa, una riga! 🤯)
quadrati = [i ** 2 for i in range(1, 6)]
print("Comprehension:", quadrati)

# Con condizione: solo numeri pari
pari = [i for i in range(1, 21) if i % 2 == 0]
print("Pari:", pari)
```

!!! tip "La formula magica"

```
La struttura è: `[espressione for variabile in sequenza if condizione]`

Leggila così: "dammi `espressione` per ogni `variabile` in `sequenza`, m
```

Le tuple

Una tupla è la **sorella rigida** della lista: si crea con le parentesi tonde `()` ed è **immutabile** — una volta creata, non la puoi più toccare. È tipo un contratto firmato!

```
# Tupla
coordinate = (10, 20)
print(coordinate)
print(coordinate[0])
print(coordinate[1])

# Provare a modificare? NOPE!
# coordinate[0] = 30  # TypeError! Mani giù!

# Unpacking: estrarre i valori in variabili separate
x, y = coordinate
print(f"x = {x}, y = {y}")
```

Quando usare tuple vs liste?

Caratteristica	Lista `[]`	Tupla `()`
Mutabile	Sì	No
Uso tipico	Collezione che cambia	Dati fissi, coordinate
Performance	Leggermente più lenta	Leggermente più veloce

Regola pratica: se i dati **non devono cambiare**, usa una tupla. Altrimenti, lista tutta la vita!

```
# La tupla è perfetta per dati che non devono cambiare
giorni_settimana = ("lun", "mar", "mer", "gio", "ven", "sab", "dom")
print(giorni_settimana)

# Funzione che restituisce più valori (usa una tupla dietro le quinte)
def min_max(lista):
    return min(lista), max(lista)

numeri = [4, 2, 7, 1, 9, 3]
minimo, massimo = min_max(numeri)
print(f"Min: {minimo}, Max: {massimo}")
```

Esercizi

Esercizio 1: Media voti

Chiedi all'utente 5 voti, salvali in una lista e calcola la media. Il registro elettronico fatto in casa!

```
voti = []

# Chiedi 5 voti e aggiungili alla lista

# Calcola e stampa la media
```

Esercizio 2: Lista senza duplicati

Data una lista, crea una nuova lista senza elementi duplicati. Tipo fare pulizia nel cassetto dei calzini!

```
numeri = [1, 3, 5, 3, 7, 1, 9, 5, 3]

# Crea una lista senza duplicati
```

Esercizio 3: Inversione lista

Inverti una lista senza usare il metodo `reverse()` o lo slicing `[::-1]`. Sì, devi farlo "a mano"! Usa un ciclo.

```
originale = [10, 20, 30, 40, 50]

# Inverti la lista manualmente con un ciclo
```