Array e matrici

Liste come array

Python non ha un tipo "array" nativo come Java o C, ma le **liste** fanno lo stesso lavoro — anzi, meglio! Una lista di numeri È un array a tutti gli effetti. Zero sbattimenti aggiuntivi.

```
# Array di temperature (la settimana del meteo)
temperature = [22.5, 24.0, 19.8, 21.3, 25.1, 23.7, 20.5]

print("Temperature della settimana:", temperature)
print("Lunedì:", temperature[0])
print("Domenica:", temperature[6])
print("Media:", sum(temperature) / len(temperature))
```

Operazioni tipiche sugli array

Queste sono le operazioni classiche che trovi in TUTTI i libri di informatica. Roba che dovete saper fare anche di notte!

Ricerca del massimo e minimo (a mano!)

Sì, Python ha `max()` e `min()`, ma il prof vuole che li facciate con il ciclo. Ed è giusto, perché dovete capire COME funzionano:

```
numeri = [34, 12, 67, 45, 89, 23, 56]

# Trova il massimo: parti dal primo e confronta con tutti
massimo = numeri[0]
for n in numeri:
    if n > massimo:
        massimo = n

print(f"Massimo: {massimo}")

# Trova il minimo: stessa logica, ma al contrario
minimo = numeri[0]
for n in numeri:
    if n < minimo:
        minimo = n

print(f"Minimo: {minimo}")
```

Ricerca lineare

Cercare un elemento in un array: scorri uno per uno finché lo trovi (o finché arrivi alla fine). Tipo cercare le chiavi in casa!

```
def cerca(lista, valore):
    for i in range(len(lista)):
        if lista[i] == valore:
            return i  # Trovato! Restituisci la posizione
    return -1  # Non trovato (il -1 è la convenzione universale)

numeri = [10, 25, 30, 45, 50, 65, 70]

pos = cerca(numeri, 45)
if pos >= 0:
    print(f"45 trovato alla posizione {pos}")
else:
    print("45 non trovato")
```

Ordinamento (Bubble Sort)

Il **Bubble Sort** è l'algoritmo di ordinamento più semplice da capire (ma anche il più lento per grandi quantità di dati). Il concetto: confronta coppie di elementi adiacenti e scambiali se sono nell'ordine sbagliato. Tipo le bollicine che salgono in un bicchiere!

```
def bubble_sort(lista):
    n = len(lista)
    for i in range(n):
        for j in range(0, n - i - 1):
            if lista[j] > lista[j + 1]:
                # Scambio! (Python lo fa in modo elegante)
                lista[j], lista[j + 1] = lista[j + 1], lista[j]

numeri = [64, 34, 25, 12, 22, 11, 90]
print("Prima:", numeri)

bubble_sort(numeri)
print("Dopo:", numeri)
```

Matrici (liste di liste)

Una **matrice** è una tabella di numeri organizzata in righe e colonne. Tipo un foglio Excel, ma nel codice. In Python si rappresenta come una **lista di liste** — sì, tipo le matrioske!

```
# Matrice 3x3
matrice = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

# Accesso: matrice[riga][colonna]
print("Elemento [0][0]:", matrice[0][0])  # 1 (prima riga, prima colonna
print("Elemento [1][2]:", matrice[1][2])  # 6 (seconda riga, terza colon
print("Elemento [2][1]:", matrice[2][1])  # 8 (terza riga, seconda colon
```

Visualizzare una matrice

Stampare una matrice in modo carino richiede un doppio ciclo. Un `for` per le righe, un `for` per le colonne:

```
matrice = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

# Stampa formattata (allineata e tutto!)
for riga in matrice:
    for elemento in riga:
        print(f"{elemento:4}", end="")
    print()
```

Creare matrici

Matrice di zeri (il modo classico)

```
righe = 3
colonne = 4

# Matrice 3x4 piena di zeri
matrice = []
for i in range(righe):
    riga = []
    for j in range(colonne):
        riga.append(0)
    matrice.append(riga)

# Stampa
for riga in matrice:
    print(riga)
```

Con list comprehension (il modo ninja)

```
righe = 3
colonne = 4

# Matrice di zeri (una riga di codice!)
matrice = [[0 for j in range(colonne)] for i in range(righe)]

for riga in matrice:
    print(riga)

print()

# Matrice identità 4x4 (gli 1 sulla diagonale, 0 altrove)
identita = [[1 if i == j else 0 for j in range(4)] for i in range(4)]

for riga in identita:
    print(riga)
```

Operazioni sulle matrici

Somma di due matrici

Per sommare due matrici, sommi gli elementi che stanno nella stessa posizione. Tipo sommare due tabelle Excel cella per cella:

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

B = [
    [7, 8, 9],
    [10, 11, 12]
]

righe = len(A)
colonne = len(A[0])

# Matrice risultato (parte da zeri)
C = [[0 for j in range(colonne)] for i in range(righe)]

for i in range(righe):
    for j in range(colonne):
        C[i][j] = A[i][j] + B[i][j]

print("A + B =")
for riga in C:
    print(riga)
```

Trasposta di una matrice

La trasposta **scambia righe e colonne**: la riga 1 diventa la colonna 1, la riga 2 diventa la colonna 2, ecc. Tipo girare il foglio di 90 gradi!

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

righe = len(matrice)
colonne = len(matrice[0])

# La trasposta ha dimensioni invertite!
trasposta = [[0 for j in range(righe)] for i in range(colonne)]

for i in range(righe):
    for j in range(colonne):
        trasposta[j][i] = matrice[i][j]

print("Originale:")
for riga in matrice:
    print(riga)

print("
Trasposta:")
for riga in trasposta:
    print(riga)
```

Somma per righe e colonne

Un classicone: calcolare le medie per riga (ogni studente) e per colonna (ogni materia):

```
voti = [
    [8, 7, 9],  # Studente 1
    [6, 8, 7],  # Studente 2
    [9, 9, 10], # Studente 3
]
materie = ["Mate", "Ita", "Ing"]

# Media per studente (scorri le righe)
print("Medie per studente:")
for i, riga in enumerate(voti):
    media = sum(riga) / len(riga)
    print(f"  Studente {i+1}: {media:.2f}")

# Media per materia (scorri le colonne — un po' più tosto!)
print("
Medie per materia:")
for j in range(len(materie)):
    somma = 0
    for i in range(len(voti)):
        somma += voti[i][j]
    media = somma / len(voti)
    print(f"  {materie[j]}: {media:.2f}")
```

Esempio pratico: griglia di gioco

Le matrici sono perfette per i giochi su griglia! Ecco un mini Tris:

```
# Inizializza griglia 3x3 (tris)
griglia = [
    ["X", "O", "X"],
    ["O", "X", "O"],
    ["O", "X", "X"]
]

# Stampa la griglia (con coordinate!)
print("  0 1 2")
for i, riga in enumerate(griglia):
    print(f"{i} {' '.join(riga)}")

# Controlla se X ha vinto (diagonale principale)
if griglia[0][0] == griglia[1][1] == griglia[2][2]:
    print(f"
{griglia[0][0]} ha vinto sulla diagonale!")
```

Esercizi

Esercizio 1: Somma diagonale

Calcola la somma degli elementi sulla **diagonale principale** di una matrice quadrata. La diagonale principale sono gli elementi dove riga == colonna.

```
matrice = [
    [5, 2, 3],
    [1, 8, 6],
    [4, 7, 9]
]

# Calcola la somma della diagonale principale (5 + 8 + 9 = 22)
```

??? success "Soluzione"

```pyodide
matrice = [
    [5, 2, 3],
    [1, 8, 6],
    [4, 7, 9]
]
somma = 0
for i in range(len(matrice)):
    somma += matrice[i][i]
print(f"Somma diagonale: {somma}")
```

Esercizio 2: Matrice moltiplicata per scalare

Moltiplica tutti gli elementi di una matrice per un numero. Tipo lo zoom di una foto, ma con i numeri!

```
matrice = [
    [1, 2, 3],
    [4, 5, 6]
]
scalare = 3

# Moltiplica ogni elemento per lo scalare e stampa il risultato
```

??? success "Soluzione"

```pyodide
matrice = [
    [1, 2, 3],
    [4, 5, 6]
]
scalare = 3
risultato = []
for riga in matrice:
    nuova_riga = []
    for elemento in riga:
        nuova_riga.append(elemento * scalare)
    risultato.append(nuova_riga)
print("Risultato:")
for riga in risultato:
    print(riga)
```

Esercizio 3: Cerca nella matrice

Cerca un valore in una matrice e stampa la sua posizione (riga, colonna). Se non c'è, dillo!

```
matrice = [
    [10, 20, 30],
    [40, 50, 60],
    [70, 80, 90]
]

valore = int(input("Che numero cerchi? "))

# Cerca il valore e stampa la posizione
```

??? success "Soluzione"

```pyodide
matrice = [
    [10, 20, 30],
    [40, 50, 60],
    [70, 80, 90]
]
valore = int(input("Che numero cerchi? "))
trovato = False
for i in range(len(matrice)):
    for j in range(len(matrice[i])):
        if matrice[i][j] == valore:
            print(f"Trovato {valore} alla posizione ({i}, {j})")
            trovato = True
if not trovato:
    print(f"{valore} non trovato nella matrice")
```