# Esercizi propedeutici sulle matrici in Python

Anna Ficotto

# 1 Crea una matrice

**Consegna** Crea una matrice 3x4 formata solo da zeri e stampala.
**Soluzione**

```
numero_righe, numero_colonne = 3, 4
m = [] # lista vuota
for i in range(numero_righe):
    riga = [] # lista vuota per la riga corrente
    for j in range(numero_colonne):
        riga.append(0) # aggiungo uno zero alla riga corrente
    m.append(riga) # aggiungo la riga alla matrice
```

**Spiegazione** Una lista di liste con 3 righe e 4 colonne, ogni valore è 0.

# 2 Stampa ogni elemento riga per riga

**Consegna** Stampa ogni numero di una matrice riga per riga.
**Soluzione**

```
m = [
        [1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]
    ]
for riga in m:
    for valore in riga:
        print(valore)
```

**Spiegazione** Doppio ciclo per stampare ogni elemento.

# 3 Somma di tutti gli elementi

**Consegna** Calcola la somma di tutti gli elementi della matrice.
**Soluzione**

```
somma = 0
for riga in m:
    for valore in riga:
        somma += valore
print(somma)
```

**Spiegazione** In ogni riga, aggiungo di volta in volta il valore corrente alla variabile `somma`.

# 4 Verifica se la matrice è quadrata

**Consegna** Scrivi una funzione che ritorna True se la matrice è quadrata.
**Soluzione**

```python
def e_quadrata(m):
    numero_righe = len(m) # nota 1
    for riga in m:
        if len(riga) != numero_righe:
            return False
    return True

m1 = [[1,2,3], [4,5,6], [7,8,9]]
m2 = [[1,2], [3,4,5]]

print(e_quadrata(m1))  # True
print(e_quadrata(m2))  # False
```

**Spiegazione** Conta quante "righe" (sotto-liste) ci sono nella matrice, e per ogni "riga" dentro la matrice `m` controlla se la lunghezza di questa "riga" (cioè quante "colonne" ha) è DIVERSA dal `numero righe` calcolato prima.

# 5 Trasposta di una matrice

**Consegna** Crea la trasposta di una matrice (le righe diventano colonne e viceversa).
**Soluzione**

```python
m_trasposta = []
for i in range(len(m[0])):          # ciclo sulle colonne
    nuova_riga = []
    for j in range(len(m)):         # ciclo sulle righe
        nuova_riga.append(m[j][i])
    m_trasposta.append(nuova_riga)
```

**Spiegazione** Inizia con una lista vuota, `m_trasposta`, che sarà la nostra matrice finale. Poi, il codice "cammina" attraverso la matrice originale. Ma invece di leggere le righe da sinistra a destra come faresti normalmente, comincia a leggere le colonne dall'alto verso il basso.
Ogni volta che finisce di leggere una colonna della matrice originale, questa colonna diventa una nuova riga nella matrice trasposta.
L'elemento chiave è `m[j][i]`: normalmente, per prendere un elemento dalla matrice `m`, useresti `m[riga][colonna]`. Qui, stiamo scambiando gli indici, usando `m[j][i]`, dove `j` rappresenta l'indice di riga e `i` quello di colonna. Questo scambio di indici è proprio il trucco che fa sì che le righe originali diventino colonne e viceversa.
Alla fine del processo, `m_trasposta` conterrà la matrice originale, ma con righe e colonne invertite.

# 6 Somma della diagonale principale

**Consegna** Somma i valori della diagonale principale di una matrice. La diagonale principale è la linea di valori che va dall'angolo in alto a sinistra della matrice fino all'angolo in basso a destra.
**Soluzione**

```
somma = 0
for i in range(len(m)):
    somma += m[i][i]
print(somma)
```

*Spiegazione*: Inizializza una variabile somma a zero. Sarà il nostro "contenitore" per il totale.
Poi, usa un ciclo **for** per "camminare" lungo questa diagonale.
Per ogni passo, prende l'elemento della matrice in posizione `m[i][i]`. Gli indici i uguali significano proprio che stiamo prendendo elementi che si trovano sulla diagonale principale (ad esempio, il primo elemento `m[0][0]`, il secondo `m[1][1]`, il terzo `m[2][2]` e così via).
Ogni elemento che trova sulla diagonale lo aggiunge alla somma.
Alla fine, `somma` conterrà il totale di tutti i numeri che si trovano su quella diagonale.

# 7 Somma della diagonale secondaria

**Consegna** Somma i valori della diagonale secondaria di una matrice. La diagonale secondaria è la linea di valori che va dall'angolo in alto a destra della matrice fino all'angolo in basso a sinistra.
**Soluzione**

```
n = len(m)
somma = 0
for i in range(n):
    somma += m[i][n - 1 - i]
print(somma)
```

# 8 Somma delle diagonali

**Consegna** Somma i valori che si trovano nelle due diagonali di una matrice quadrata (principale e secondaria).
**Soluzione**

```
def sommaDiagonali(m):
    n = len(m)
    somma_principale = 0
    somma_secondaria = 0
    for i in range(n):
        somma_principale += m[i][i]
    for i in range(n):
        somma_secondaria += m[i][n - 1 - i]
        return somma_principale + somma_secondaria
```

**Spiegazione** Gli elementi della diagonale secondaria di una matrice quadrata `m` si trovano nelle posizioni `m[i][n - 1 - i]`, dove `i` è l'indice della riga e `n` è la dimensione della matrice.

# 9 Conta i numeri pari

**Consegna** Conta il numero di valori pari nella matrice. **Soluzione**

```
conta = 0
for riga in m:
    for valore in riga:
        if valore % 2 == 0:
            conta += 1
print(conta)
```

**Spiegazione** Il programma scorre ogni riga, e ad ogni iterazione scorre anche tutti i valori di quella riga. Per ogni valore il codice controlla se quest'ultimo è pari con l'operatore `%`. Se il risultato dell'operazione è `True`, viene incrementata la variabile `conta`.

# 10 Verifica presenza di almeno uno zero

**Consegna** Verifica che in una matrice sia presente almeno un valore pari a 0. **Soluzione 1 (base)**

```
def contiene_zero(m):
    for riga in m:
                for val in riga:
                if val == 0:
                    return True
    return False
```

**Spiegazione** Il programma scorre ogni riga, e per ogni valore trovato nella riga controlla se esso è pari a 0.

**Soluzione 2 (avanzata)**

```
def contiene_zero(m):
    for riga in m:
        if 0 in riga:
            return True
    return False
```

**Spiegazione** Il programma controlla ogni riga, ma invece di iterare anche tutti i singoli valori, usa la funzione `in`, che si può tradurre con *"se nella riga ci sono degli 0..."*.

# 11 Raddoppia ogni elemento

**Consegna** Raddoppia ogni elemento della matrice. **Soluzione**

```
def raddoppia(m):
    for riga in m:
                for val in riga:
                        val = val*2
```

**Spiegazione** Il programma "cicla" tutti i valori della matrice e li raddoppia uno ad uno.

# 12 Valore massimo e minimo

**Consegna** Trova il valore massimo e il valore minimo della matrice.
**Soluzione 1 (senza funzioni max() e min())**

```
massimo = m[0][0]
minimo = m[0][0]

for riga in m:
    for elemento in riga:
        if elemento > massimo:
            massimo = elemento
        if elemento < minimo:
            minimo = elemento

print("Max:", massimo, "Min:", minimo)
```

**Spiegazione** Inizializzo massimo e minimo col primo elemento della matrice (`m[0][0]`). Faccio due cicli annidati: il primo scorre ogni riga (`for riga in m`), il secondo scorre ogni elemento della riga (`for elemento in riga`). A ogni passo, confronto `elemento` con `massimo` e `minimo`: se è più grande, aggiorno `massimo`, se è più piccolo, aggiorno `minimo`.

**Soluzione 2**

```
massimo = m[0][0]
minimo = m[0][0]

for riga in m:
    max_riga = max(riga)  # massimo nella riga corrente
    min_riga = min(riga)  # minimo nella riga corrente

    if max_riga > massimo:
        massimo = max_riga

    if min_riga < minimo:
        minimo = min_riga

print("Max:", massimo, "Min:", minimo)
```

**Spiegazione** Inizializzo massimo e minimo con il primo elemento della matrice. Per ogni riga uso `max(riga)` per trovare il massimo nella riga e `min(riga)` per trovare il minimo. Confronto questi valori con quelli trovati finora. Alla fine, stampo i risultati.

# 13 Tutte le righe hanno la stessa somma?

**Consegna** Controllare se i valori di ogni riga, sommati, danno tutti la stessa somma.
**Soluzione**

```python
def righeConSommaUguale(m):
    # Calcolo la somma della prima riga
    somma_riferimento = 0
    for num in m[0]:
        somma_riferimento += num

    # Controllo le altre righe
    for i in range(1, len(m)):
        somma_riga = 0
        for num in m[i]:
            somma_riga += num

        if somma_riga != somma_riferimento:
            return False  # Appena ne trovo una diversa, ritorno False

    return True  # Tutte le somme sono uguali
```

**Spiegazione** Calcolo la somma della prima riga: mi servirà come riferimento. Poi controllo tutte le altre righe, una per una: calcolo la loro somma e la confronto con quella della prima riga. Se tutte sono uguali → True, altrimenti → False.

# 14 Specchia orizzontalmente

**Consegna** Scrivi una funzione che ritorni la matrice specchiata orizzontalmente:

$$\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{matrix} \quad => \quad \begin{matrix} 3 & 2 & 1 \\ 6 & 5 & 4 \\ 9 & 8 & 7 \end{matrix}$$

**Soluzione 1 (base)**

```python
def specchia_orizzontalmente(m):
    specchiata = []
    for riga in m:
        nuova_riga = []
        i = len(riga) - 1
        while i >= 0:
            nuova_riga.append(riga[i])
            i -= 1
        specchiata.append(nuova_riga)
    return specchiata
```

**Spiegazione** Prendo ogni singola riga. La leggo a partire dalla fine (cioè dall'ultimo elemento verso il primo). Man mano che leggo gli elementi al contrario, li aggiungo in una nuova riga. Poi aggiungo questa nuova riga a una nuova matrice.

**Soluzione 2 (con `reverse()`)**

```python
def specchia_orizzontalmente(m):
    specchiata = []
    for riga in m:
        nuova_riga = []
        i = len(riga) - 1
        while i >= 0:
            nuova_riga.append(riga[i])
            i -= 1
        specchiata.append(nuova_riga)
    return specchiata
```

**Spiegazione** Prendiamo una riga alla volta dalla matrice. Ne facciamo una copia, così non roviniamo la riga originale. La copia si fa in modo veloce scrivendo riga[:], che significa: "copio tutta la lista". A questo punto abbiamo una riga indipendente, e possiamo invertirla usando un comando già pronto in Python: `reverse()`. Questo comando cambia l'ordine degli elementi della lista, da ultimo a primo. Poi prendiamo questa riga invertita e la ricostruiamo passo passo in una nuova lista. Quando la nuova riga è pronta, la aggiungiamo alla nuova matrice che conterrà tutte le righe invertite. Ripetiamo questo processo per tutte le righe della matrice. Alla fine otteniamo una nuova matrice in cui ogni riga è specchiata, cioè letta da destra a sinistra.

# 15 Ruota la matrice di 90° (senso orario)

**Consegna** Ruotare la matrice di 90° in senso orario come in figura sottostante.

$$\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{matrix} \quad => \quad \begin{matrix} 7 & 4 & 1 \\ 8 & 5 & 2 \\ 9 & 6 & 3 \end{matrix}$$

**Soluzione**

```python
def ruota90(m):
    n = len(m)
    # Creo una matrice vuota n x n
    ruotata = []
    for i in range(n):
        ruotata.append([0] * n)

    # Ruoto la matrice: ogni elemento m[i][j] va in ruotata[j][n-1 - i]
    for i in range(n):
        for j in range(n):
            ruotata[j][n - 1 - i] = m[i][j]

    return ruotata
```

**Spiegazione** Creiamo una matrice vuota con le stesse dimensioni della matrice originale. Per ogni elemento `m[i][j]` nella matrice originale, lo spostiamo in `ruotata[j][n - 1 - i]`. Questo significa che la colonna diventa riga, e la riga viene invertita per fare la rotazione.

# Approfondimento: List comprehensions

Python ci offre un modo più breve per scrivere alcune istruzioni riguardanti una lista. In questo esempio troviamo dei cicli annidati (esercizio 5).
È un modo **avanzato** di scrivere codice in Python.

```python
def trasposta(m):
    return [[m[j][i] for j in range(len(m))] for i in range(len(m[0]))]
```

Per l'esercizio 12:

```python
massimo = max(max(riga) for riga in m)
minimo = min(min(riga) for riga in m)
```