Usando listas aninhadas para representar matrizes

Matrizes (as mesmas da matemática) são necessárias em diversas ocasiões. Por exemplo

- Imagens são armazenadas como matrizes de pixels
- Computação gráfica usa funções sobre vetores e matrizes para definir transformações
- Tabelas, como as de uma planilha de cálculo, são matrizes
- Muitos jogos usam matrizes, como por exemplo as Damas, o Xadrez, o Sudoku, etc.

Uma forma simples de representar uma matriz é através de uma lista cujos elementos são listas. Por exemplo, a matriz

$$\begin{bmatrix} 23 & -2 & 10 \\ 0 & 1 & 1 \end{bmatrix}$$

pode ser representado em python pela lista de listas

```
[[23, -2, 10], [0, 1, 1]]
```

onde, o primeiro elemento se corresponde com a primeira fila da matriz enquanto que o segundo elemento se corresponde com a segunda fila da matriz.

Acessando os elementos de uma matriz

In [1]:

```
matriz = [[23, -2, 10], [0, 1, 1]]
print(matriz[0])    # toda a primeira fila
print(matriz[1])    # toda a segunda fila
print(matriz[0][1]) # o segundo elemento da primeira fila
print(matriz[1][2]) # o terceiro elemento da segunda fila
```

```
[23, -2, 10]
[0, 1, 1]
-2
1
```

Na literatura, listas de listas também são conhecidas pelo nome de **listas bidimensionais**, **listas aninhadas** ou **arrays aninhados**.

Processando listas aninhadas elemento por elemento

Listas normalmente são processadas elemento por elemento usando laços `for` . Para processar listas aninhadas normalmente usam-se laços `for` aninhados. Por exemplo, para imprimir uma lista aninhada podemos definir a seguinte função

In [5]:

```
def imprimirMatriz(m):
    for fila in m:
        for elem in fila:
            print("{:5}".format(elem), end=' ')
        print()


imprimirMatriz(matriz)
```

```
   23    -2    10
    0     1     1
```

Como outro exemplo, temos esta função que cálcula o menor elemento da matriz

In [6]:

```
def menor(m):
    menor = m[0][0]
    for fila in m:
        for elem in fila:
            if elem < menor:
                menor = elem
    return menor

menor(matriz)
```

Out[6]:

```
-2
```

Em diversas ocasiões é conveniente varrer a lista usando a combinação `for` com índices. Por exemplo, se queremos definir uma função que retorne as coordenadas do menor elemento da matriz.

In [8]:

```
def posicaoDoMenor(m):
    menx, meny = 0, 0
    for i in range(len(m)):                    # i: índices das filas
        for j in range(len(m[i])):             # j: índices das colunas
            if m[i][j] < m[menx][meny]:
                menx, meny = i, j
    return menx, meny

posicaoDoMenor(matriz)
```

Out[8]:

```
(0, 1)
```

Criando listas aninhadas

Suponha que queremos criar uma matriz $m \times n$ inicializada toda com zeros. Precisamos criar m filas (listas), cada uma destas sendo uma lista com n zeros.

Para exemplificar, fixemos $m = 3$ e $n = 4$. Isto significa que teremos 3 filas, cada uma com 4 elementos.

Podemos criar uma fila usando o operador de repetição de listas `*` assim:

In [10]:

```
[0] * 4
```

Out[10]:

```
[0, 0, 0, 0]
```

Ficamos tentados a criar a matriz inteira assim:

In [11]:

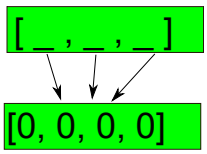
```
matriz1 = [[0] * 4] * 3

matriz1
```

Out[11]:

```
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
```

No entanto, esta solução não funciona bem, pois lembrem que listas são tratadas por referência. Assim, a operação de repetição de 3 filas cria uma lista aninhada onde cada fila é a mesma.



In [12]:

```
matriz1[0][0] = 2

matriz1
```

Out[12]:

```
[[2, 0, 0, 0], [2, 0, 0, 0], [2, 0, 0, 0]]
```

A solução é criar explicitamente 3 filas novas usando um laço `for`

In [13]:

```
matriz1 = []
for _ in range(3):
    matriz1.append([0] * 4)

print(matriz1)
matriz1[0][0] = 2
print(matriz1)
```

```
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
[[2, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
```

Uma alternativa mais compacta usa compreensões

In [14]:

```
matriz1 = [[0]*4 for _ in range(3)]

print(matriz1)
matriz1[0][0] = 2
print(matriz1)
```

```
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
[[2, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
```

Lendo uma matriz

Dado o número n de filas da matriz e depois cada uma das filas, uma por linha, um programa que faz a leitura carregando em uma nova matriz é

In [15]:

```
n = int(input())
matriz2 = []
for _ in range(n):
    fila = input().split()
    matriz2.append( [int(elem) for elem in fila] )

matriz2
```

```
3
2 7 8 9
7 6 5 1
0 0 1 2
```

Out[15]:

```
[[2, 7, 8, 9], [7, 6, 5, 1], [0, 0, 1, 2]]
```

Ou usando compreensões

In [16]:

```
n = int(input())
matriz2 = [ [int(elem) for elem in input().split()] for _ in range(n) ]

matriz2
```

```
3
2 7 8 9
7 6 5 1
0 0 1 2
```

Out[16]:

```
[[2, 7, 8, 9], [7, 6, 5, 1], [0, 0, 1, 2]]
```

Processando partes da matriz

Suponha que queremos calcular a soma de todos os elementos da diagonal principal da matriz. Para identificar os elementos da diagonal, processamos pelos índices

In [17]:

```
matriz3 = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]]
imprimirMatriz(matriz3)

soma = 0
for i in range(len(matriz3)):
    for j in range(len(matriz3[i])):
        if i == j:
            soma += matriz3[i][i]

soma
```

```
 1  2  3  4
 5  6  7  8
 9 10 11 12
13 14 15 16
```

Out[17]:

```
34
```

```
Podemos escrever uma solução mais eficiente da que está acima iterando com `for` sem
aninhamento, assim
```

In [18]:

```
soma = 0
for i in range(len(matriz3)):
    soma += matriz3[i][i]

soma
```

Out[18]:

```
34
```

Suponha agora que queremos colocar zeros em toda a parte superior da diagonal de uma matriz quadrada. Também precisamos processar pelos índices.

In [19]:

```
for i in range(len(matriz3)):
    for j in range(len(matriz3[i])):
        if j > i:
            matriz3[i][j] = 0

imprimirMatriz(matriz3)
```

```
 1  0  0  0
 5  6  0  0
 9 10 11  0
13 14 15 16
```

Também neste caso temos uma forma mais eficiente, porém ainda usando `for` aninhados. A diferença é que

pordemos restringir a varrida somente à parte superior da matriz, assim

In [20]:

```
matriz3 = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]]

for i in range(len(matriz3)):
    for j in range(i+1, len(matriz3[i])):
        matriz3[i][j] = 0

imprimirMatriz(matriz3)
```

```
 1  0  0  0
 5  6  0  0
 9 10 11  0
13 14 15 16
```