



Introdução à Programação

Aula 11: Listas 2d para a representação de matrizes

Prof. Eduardo Corrêa



Introdução (1/3)

 A lista 2d é uma lista de listas, ou seja, uma lista onde cada elemento também é uma lista.

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

- A lista 2d também é chamada de lista bidimensional.
- Veja outro exemplo... Uma lista contendo 2 listas de 3 elementos.
- Cada lista tem o nome de um livro, depois o nome do 1º autor, depois o nome do 2º autor.

```
livros = [ ["Estatística Básica", "Bussab", "Morettin"], ["Data Mining", "Zaki", "Meira Jr."] ]
```



Introdução (2/3)

No Python, a lista 2d é especialmente útil para a representação de matrizes.

m = [[100, -1, 90, 0, -5], [-13, 27, 88, -1, 0], [0, 14, 30, -20, 25], [11, 0, 13, 32, -99]]



	[0]	[1]	[2]	[3]	[4]
[0]	100	-1	90	0	-5
[1]	-13	27	88	-1	0
[2]	0	14	30	-20	25
[3]	11	0	13	32	-99

m



Introdução (3/3)

- Importante!!! para o Python, não é uma matriz... é uma lista de listas.
- Mas por conveniência, "fingimos" que é matriz, já que isso é muito útil para a resolução de diversos problemas práticos

```
m = [[100, -1, 90, 0, -5], [-13, 27, 88, -1, 0], [0, 14, 30, -20, 25], [11, 0, 13, 32, -99]]
```



m é uma lista2d com 4 listas de 5 inteiros. Então vou tratar como matriz de inteiros 4x5 (4 linhas, 5 colunas)

	[0]	[1]	[2]	[3]	[4]
[0]	100	-1	90	0	-5
[1]	-13	27	88	-1	0
[2]	0	14	30	-20	25
[3]	11	0	13	32	-99

m



Criação de Listas 2d (1/2)

- Para criar uma lista 2d, devemos especificar uma relação de listas, separadas por vírgula, entre colchetes.
- Para "fingir" que é uma matriz, basta tratar cada lista como se fosse uma linha da matriz.

```
• 2 linhas e 2 colunas

matriz_pesos = [

[0.58, 0.19],

[0.33, 0.65]

]
```

• 4 linhas e 6 colunas matriz_binaria = [[0, 1, 0, 0, 1, 0], [1, 0, 0, 1, 1, 1], [0, 0, 0, 0, 0, 0, 1], [0, 1, 1, 0, 1, 0] • 5 linhas, onde cada uma delas possui um número diferente de colunas (*mais uma prova da flexibilidade das listas*).

```
nao_retangular = [
    [8, 0, 5, 3, 0, 9],
    [2, 6, 0],
    [],
    [3, 8, 3],
    [4, 7]
]
```



Criação de Listas 2d (2/2)

A lista de listas abaixo pode ser tratada como uma matriz de inteiros, com 4 linhas e 6 colunas $(4 \times 6 = 24 \text{ células})$.

```
matriz_binaria = [
[0, 1, 0, 0, 1, 0],
[1, 0, 0, 1, 1, 1],
[0, 0, 0, 0, 0, 0, 1],
[0, 1, 1, 0, 1, 0]
]
```



	[0]	[1]	[2]	[3]	[4]	[5]
[0]	0	1	0	0	1	0
[1]	1	0	0	1	1	1
[2]	0	0	0	0	0	1
[3]	0	1	1	0	1	0

matriz_binaria



Indexação (1/4)

- Para acessar (indexar) um elemento (valor) da lista 2d devemos especificar:
 - O nome da lista.
 - Seguido do **índice da lista interna (sublista)** que desejamos acessar, entre colchetes.
 - Seguido do índice do elemento dessa lista interna, também entre colchetes.

```
m = [[100, -1, 90, 0, -5], [-13, 27, 88, -1, 0], [0, 14, 30, -20, 25], [11, 0, 13, 32, -99]]

print(m[0][0]) # imprime 100 (lista 0, elemento 0)
print(m[2][1]) # imprime 14 (lista 2, elemento 1)
print(m[3][4]) # imprime -99 (lista 3, elemento 4)
```



Indexação (2/4)

- Fica mais intuitivo imaginar que a lista 2d é uma matriz... Veja os exemplos:
- m[0][0]: referencia o elemento da linha 0, coluna 0 (ou célula [0][0]), cujo valor é 100
- m[2][1]: referencia o elemento da linha 2, coluna 1, cujo valor é 14
- m[3][4]: referencia o elemento da linha 4, coluna 3, cujo valor é -99

m = [[100, -1, 90, 0, -5], [-13, 27, 88, -1, 0], [0, 14, 30, -20, 25], [11, 0, 13, 32, -99]]



	[0]	[1]	[2]	[3]	[4]
[0]	100	-1	90	0	-5
[1]	-13	27	88	-1	0
[2]	0	14	30	-20	25
[3]	11	0	13	32	-99



Indexação (3/4)

IMPORTANTE

■ Não esqueça que a indexação começa do 0 e não de 1 !!!

	[0]	[1]	[2]	[3]	[4]
[0]	100	-1	90	0	-5
[1]	-13	27	88	-1	0
[2]	0	14	30	-20	25
[3]	11	0	13	32	-99



Indexação (4/4)

 Uma lista 2d é uma lista de listas. Logo o seu 1º elemento é uma lista, o 2º é outra lista etc.

- Assim, ao tratar lista 2d como matriz, veja que é possível <u>recuperar uma linha inteira</u> de uma vez !!!
- Mas, infelizmente não há uma forma direta de processar uma coluna inteira.
- Isso porque, os elementos de uma coluna estão armazenados em listas diferentes (há uma solução com a função zip() ... veremos na aula sobre tuplas).



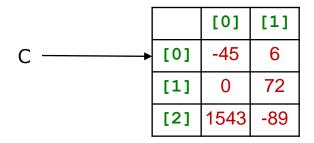
```
# criação da matriz
C = [[-45, 6],
  [0, 72],
  [1543, -89]]
# impressão das células [0][0] e [2][0]
print('C[0][0] =', C[0][0])
print('C[2][0] =', C[2][0])
# novas atribuições
C[1][0] = C[1][0] + 1 # agora C[1][0] vale 1!
C[2][1] = C[1][1] # agora C[2][1] vale 72!
# impressão das células [1][0] e [2][1]
print('C[1][0] =', C[1][0])
print('C[2][1] =', C[2][1])
```

- O programa Python ao lado, usa lista 2d para:
 - Criar uma matriz de inteiros de 3 linhas e 2 colunas (6 células) chamada C.
 - Atribuir valores para elementos de C.
 - Imprimir o conteúdo de algumas células da matriz na tela.
 - Modificar alguns dos valores inicialmente atribuídos.
 - Imprimir o conteúdo das posições modificadas.
- Vamos examinar a sua execução!



```
# criação da matriz
C = [[-45, 6], \iff
 [0, 72],
  [1543, -89]]
# impressão das células [0][0] e [2][0]
print('C[0][0] =', C[0][0])
print('C[2][0] =', C[2][0])
# novas atribuições
C[1][0] = C[1][0] + 1 # agora C[1][0] vale 1!
C[2][1] = C[1][1] # agora C[2][1] vale 72!
# impressão das células [1][0] e [2][1]
print('C[1][0] =', C[1][0])
print('C[2][1] =', C[2][1])
```

- Este comando cria a lista2d em memória.
- A lista chama-se C e contém 3 sublistas, cada uma com 2 elementos.
- Trata-se da reperesentação de uma matriz 3x2.
- Obs.: não é preciso especificar as sublistas uma embaixo da outra. Pode ser ao lado.

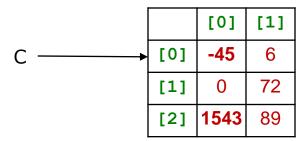




```
# criação da matriz
C = [[-45, 6],
 [0, 72],
  [1543, -89]]
# impressão das células [0][0] e [2][0]
print('C[0][0] =', C[0][0]) <=
print('C[2][0] =', C[2][0])   
# novas atribuições
C[1][0] = C[1][0] + 1 # agora C[1][0] vale 1!
C[2][1] = C[1][1] # agora C[2][1] vale 72!
# impressão das células [1][0] e [2][1]
print('C[1][0] =', C[1][0])
print('C[2][1] =', C[2][1])
```

 Os dois comandos destacados imprimem na tela:

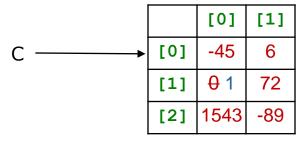
```
C[0][0] = -45
C[2][0] = 1543
```





```
# criação da matriz
C = [[-45, 6],
  [0, 72],
  [1543, -89]]
# impressão das células [0][0] e [2][0]
print('C[0][0] =', C[0][0])
print('C[2][0] =', C[2][0])
# novas atribuições
C[1][0] = C[1][0] + 1 agora C[1][0] vale 1!
C[2][1] = C[1][1] # agora C[2][1] vale 72!
# impressão das células [1][0] e [2][1]
print('C[1][0] =', C[1][0])
print('C[2][1] =', C[2][1])
```

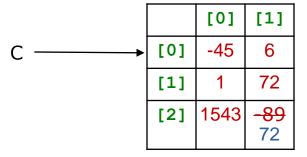
- O valor do elemento da célula [1][0] foi modificado.
- Antes valia 0 e agora vale 1
- Note que os valores das outras células são preservados.





```
# criação da matriz
C = [[-45, 6],
  [0, 72],
  [1543, -89]]
# impressão das células [0][0] e [2][0]
print('C[0][0] =', C[0][0])
print('C[2][0] =', C[2][0])
# novas atribuições
C[1][0] = C[1][0] + 1 # agora C[1][0] vale 1!
C[2][1] = C[1][1] \iff \# \text{ agora } C[2][1] \text{ vale } 72!
# impressão das células [1][0] e [2][1]
print('C[1][0] =', C[1][0])
print('C[2][1] =', C[2][1])
```

- Este comando faz com que o valor da célula [2][1] de C, passe a ser igual ao valor de sua célula [1][1].
- Antes C[2][1] valia -89
 e agora vai passar a
 valer 72.



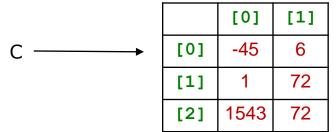


```
# criação da matriz
C = [[-45, 6],
 [0, 72],
  [1543, -89]]
# impressão das células [0][0] e [2][0]
print('C[0][0] =', C[0][0])
print('C[2][0] =', C[2][0])
# novas atribuições
C[1][0] = C[1][0] + 1 # agora C[1][0] vale 1!
C[2][1] = C[1][1] # agora C[2][1] vale 72!
# impressão das células [1][0] e [2][1]
print('C[1][0] =', C[1][0]) <
print('C[2][1] =', C[2][1])
```

 Os dois comandos destacados imprimem na tela:

$$C[1][0] = 1$$

 $C[2][1] = 72$





```
1 # criação da matriz
  2 C = [[-45, 6],
 3 [0, 72],
     [1543, -89]]
 6 # impressão das células [0][0] e [2][0]
 7 print('C[0][0] =', C[0][0])
 8 print('C[2][0] =', C[2][0])
10 # novas atribuições
11 C[1][0] = C[1][0] + 1 # agora C[1][0] vale 1!
12 C[2][1] = C[1][1] # agora C[2][1] vale 72!
13
14 # impressão das células [1][0] e [2][1]
15 print('C[1][0] =', C[1][0])
16 print('C[2][1] =', C[2][1])
Shell ×
>>> %Run exemplo_matriz_C
 C[0][0] = -45
 C[2][0] = 1543
 C[1][0] = 1
 C[2][1] = 72
```



Operações: Certo x Errado (1/3)

- Os programas dos dois slides a seguir apresentam, respectivamente, exemplos operações que não podem ser realizadas com listas 2d e operações que podem ser realizadas com listas 2d
 - Na verdade, com listas comuns também.
- Não tente copiar o primeiro programa para o Python, pois ele não vai funcionar!
- O primeiro programa foi criado apenas com o propósito de apresentar as formas erradas de manipular uma lista.



Operações: Certo x Errado (2/3)

```
# PROGRAMA 1: COISAS QUE NÃO DÃO CERTO
m1 = [[1000, 2],
   [-300, 0],
   [999, -272]]
m2 = [[1, 2],
   [3, 4],
   [5, 6]]
m3 = m1
             # ERRADO: não é possível clonar uma lista (simples ou 2d) assim
             # Essa sintaxe serve apenas para fazer m3 virar um apelido de m1, ou
             # seja, os 2 nomes apontarão para o mesmo objeto em memória.
m1 = m1 + 1 # ERRADO: não é possível realizar operações aritméticas sobre todos
              # os elementos da lista (simples ou 2d) de uma vez. Só é possível
              # usando um laço (ou list comprehension) que percorra e processe
              # cada célula, uma por uma.
m4 = m1 - m2 # ERRADO: Como no caso anterior, é preciso criar um laço e
               # laço e manipular os elementos da lista individualmente.
```



Operações: Certo x Errado (3/3)

```
# PROGRAMA 1: COISAS QUE FUNCIONAM
m1 = [[1000, 2],
   [-300, 0],
   [999, -272]]
m2 = [[1, 2],
   [3, 4],
   [5, 6]]
if (m1 == m2):
  print ('m1 é igual a m2') # CERTO: é possível comparar se duas listas
                       # possuem o mesmo conteúdo dessa maneira
print(m1) # CERTO: é possível imprimir todos os elementos
             de uma lista sem usar um laço. Porém, no caso de
       #
             listas 2d, a desvantagem é que as linhas serão
              impressas lado a lado e não uma embaixo da outra
             Ainda nessa aula veremos como resolver isso...
       #
```



Lista 2d - Percorrendo os elementos (1/8)

- A técnica básica para percorrer as células de uma matriz (todos os elementos de uma lista 2d) é utilizar dois laços for aninhados.
 - No laço externo, a variável "i" é usada para controlar a posição da linha.
 - E no laço interno, "j" é usada para controlar a posição da coluna.

```
# RECEITA para visitar todas as células de "mat"
mat = qualquer lista 2d retangular
# considere que existe uma matriz "mat" qualquer em memória
num lins = len (mat) # número de linhas = len da matriz
num cols = len (mat[0]) # número de colunas = len de qq linha
# percorre todas as células matriz
for i in range(num lins):
    for j in range(num cols):
        comando, p/ processar célula i,j
        comando p/ processar célula i,j
    comando(s) após finalizar linha i (pode ser necessário)
```



Lista 2d - Percorrendo os elementos (2/8)

Exemplo 1: percorre e imprime todos os elementos da matriz, um embaixo do outro.

```
w = [[10, 20],
        [30, 40],
        [50, 60]]

num_lins = len(w); num_cols = len(w[0])

# percorre e imprime os elementos da matriz
for i in range(num_lins):
    for j in range(num_cols):
        print(w[i][j])
```

Saída:

```
10
20
30
40
50
```



Lista 2d - Percorrendo os elementos (3/8)

Exemplo 2: imprime a matriz de uma maneira "bonita"

Saída:



Lista 2d - Percorrendo os elementos (4/8)

Exemplo 2: imprime a matriz de uma maneira "bonita"

- p/ imprimir os dados de cada coluna (dentro do laço for j), um ao lado do outro, utilizamos o print() com o parâmetro end = " ", para evitar o salto de linha após a impressão do valor
- Usamos ainda a máscara de formatação ":>6" para pegar o valor armazenado em mat[i][j] e o alinhar à direita, reservando 6 espaços.

24



Lista 2d - Percorrendo os elementos (5/8)

Exemplo 2: imprime a matriz de uma maneira "bonita"

- Assim que todas as colunas de uma linha são impressas, o laço for j finaliza.
- Então, colocamos um print() vazio para forçar um salto de linha. Com isso, na próxima iteração do laço for i, a próxima linha da matriz será impressa abaixo da anterior.



Lista 2d - Percorrendo os elementos (6/8)

Exemplo 3: armazena os elementos da diagonal principal da matriz em uma lista.

```
m = [[1, 2, 3],
    [4, 5, 6],
     [7, 8, 9]] # define uma matriz 3 x 3
ordem = len(m) # nesse caso, como matriz é quadrada, basta
               # pegar len(m) - ordem da matriz quadrada
# pega os elementos da diagonal principal
diagonal = []
for i in range(ordem):
    for j in range(ordem):
        if i == j:
            diagonal.append(m[i][j])
print(diagonal) # [1, 5, 9]
```



Lista 2d - Percorrendo os elementos (7/8)

Exemplo 3: armazena os elementos da diagonal principal da matriz em uma lista.

```
m = [[1, 2, 3],
    [4, 5, 6],
     [7, 8, 9]] # define uma matriz 3 x 3
ordem = len(m) # nesse caso, como matriz é quadrada, basta
               # pegar len(m) - ordem da matriz quadrada
# pega os elementos da diagonal principal
diagonal = []
for i in range(ordem):
    for j in range(ordem):
        if i == i:
            diagonal.append(m[i][j])
print(diagonal) # [1, 5, 9]
```

Sabemos que estamos em uma célula que corresponde à diagonal principal quando o valor de i (índice da linha) é igual ao de j (índice da coluna).



Lista 2d - Percorrendo os elementos (8/8)

Exemplo 4: multiplica todos os elementos de uma matriz por 10.

 Precisamos visitar cada célula i,j e alterá-la com uma atribuição, multiplicando o valor armazenado por 10



OUTROS EXEMPLOS

- * * * * * **ATENÇÃO** * * * *
- Consulte o arquivo PDF "exercicios_resolvidos03_lista2d.pdf" (no repositório de aulas) para ver diversos outros exemplos.
 - É muito importante você consultar, pois são exemplos de **programas maiores** do que os apresentados nessa aula.
- **Obs.**: o Python permite criar listas 3d (lista de lista de lista), listas 4d (lista de lista de lista de lista) ou de qualquer outra dimensão.
- Mas no nosso curso, cobriremos apenas as listas 2d.



Carregando Lista 2d via teclado (1/4)

- É um pouco mais complicado do que carregar uma lista simples.
- Há várias formas de fazer. Apresentaremos duas:
 - Forma 1 (mais simples, menos eficiente):
 - Criando todas as células em memória com valor 0 e depois trocar pelo valor digitado pelo usuário.
 - Forma 2 (menos simples, mais eficiente):
 - Inserindo célula por célula a medida que o usuário digita.



Carregando Lista 2d via teclado (2/4)

- **Forma 1** (mais simples, menos eficiente):
 - (1) criar a matriz, preenchendo todas as células com o valor 0
 - (2) depois, trocar o valor de cada célula pelo digitado pelo usuário.

```
# FORMA 1 para preencher matriz via teclado
# cria a matriz m x n, pré-preenchendo células com 0
m = 4; n = 2 # define o número de linhas e colunas
mat = []
for i in range(m): mat.append([0] * n) # insere cada linha com
                                        # n colunas contendo 0
# a matriz está criada, podemos ler os valores do teclado,
# armazenando na célula correspondente
# (nesse exemplo, é matriz de inteiros)
for i in range(m):
    for j in range(n):
           print(f'Elemento[{i},{j}] = ', end="")
           mat[i][j] = int(input())
print(mat)
```



Carregando Lista 2d via teclado (3/4)

Forma 2 (menos simples, mais eficiente):

print(mat)

- (1)Insere os elementos em uma linha um por um, a medida que usuário digita.
- (2) Após ele digitar elemento da última coluna, insere a linha.

```
# FORMA 2 para preencher matriz via teclado
# cria a matriz m x n, pré-preenchendo células com 0
m = 4; n = 2 # define o número de linhas e colunas
# a matriz é criada vazia, vamos ler os valores de
# cada linha teclado e armazenaremos cada linha na lista
mat = []
for i in range(m):
    linha = [] # linha i é criada vazia
    for j in range(n):
           print(f'Elemento[{i},{j}] = ', end="")
           linha.append(int(input())) # insere valor j da linha i
   mat.append(linha) # linha i terminou, será inserida em mat
```



Carregando Lista 2d via teclado (4/4)

- Tendo em vista que:
 - Em muitos exercícios será assumido que a matriz já existe em memória.
 - Em muitas linguagens de programação apenas a Forma 1 está disponível.
 - Irei aceitar ambas as formas na prova e exercícios.



Exercícios

(1) Faça um programa que gere automaticamente (sem ler do usuário e sem atribuições individuais para cada célula) uma matriz 3x4 onde todos os elementos têm o valor 1:

(2) Modifique o programa anterior, para que agora a matriz seja gerada automaticamente com os seguintes valores:

```
| 1 2 3 4 |
| 5 6 7 8 |
| 9 10 11 12 |
```

- (3) Dadas duas matrizes 4x3 A e B em memória, ambas preenchidas, faça um programa que calcule e mostre:
 - A soma das duas matrizes, resultando em uma nova matriz C.
 - A diferença das duas matrizes, resultando em uma nova matriz D.
- (4) Faça um programa que imprima os elementos da **diagonal secundária** de uma matriz quadrada de ordem m.