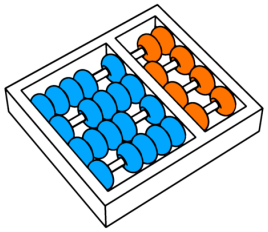

# MC102 – Aula 09
# Objetos Multidimensionais

## Algoritmos e Programação de Computadores

Zanoni Dias

2019

Instituto de Computação

# Roteiro

Matrizes e Objetos Multidimensionais

Exercícios

Extra: NumPy

# Matrizes e Objetos Multidimensionais

# Matrizes e Objetos Multidimensionais

- Matrizes e objetos multidimensionais são generalizações de objetos simples vistos anteriormente (listas e tuplas).
- Esses tipos de dados nos permitem armazenar informações mais complexas em uma única variável.
- Exemplo de informações que podem ser armazenadas/manipuladas utilizando matrizes e objetos multidimensionais:
  - Matemática: operações com matrizes.
  - Processamento de imagem: cor de cada pixel presente na imagem.
  - Mapas e geolocalização: informação sobre o relevo em cada ponto do mapa.
  - Jogos de tabuleiro: Xadrez, Damas, Go, Batalha Naval, etc.

# Matrizes e Objetos Multidimensionais

- Uma lista pode conter elementos de tipos diferentes.
- Uma lista pode conter inclusive outras listas.
- Exemplo de declaração de um objeto multimensional:

```
obj = [
  7, 42, True, "MC102", 3.14,
  [0.1, 0.2, 0.3]
]
```

- Exemplo de declaração de um objeto multimensional:

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

# Declaração de Matrizes

- Uma matriz é um objeto bidimensional, formada por listas, todas do mesmo tamanho.
- Sua representação é dada na forma de uma lista de listas (a mesma ideia pode ser aplicada para tuplas).
- Exemplo de declaração de uma matriz $2 \times 2$:

```
matriz = [
  [1, 2], # linha 1
  [3, 4]  # linha 2
]
```

- Exemplo de declaração de uma matriz $3 \times 4$:

```
matriz = [
  [11, 12, 13, 14], # linha 1
  [21, 22, 23, 24], # linha 2
  [31, 32, 33, 34]  # linha 3
]
```

# Declaração de Matrizes

- Podemos criar uma matriz com as informações fornecidas pelo usuário.
- Exemplo de como receber uma matriz de dimensões $l \times c$ como entrada:

```
l = int(input())
c = int(input())
matriz = []

for i in range(l):
  linha = []
  for j in range(c):
    linha.append(int(input()))
  matriz.append(linha)
```

# Declaração de Matrizes

- Podemos ainda inicializar uma matriz com valores pré-definidos.
- Inicializando uma matriz de dimensões $l \times c$ e atribuindo valor zero para todos os elementos:

```
l = 3
c = 2
matriz = []
for i in range(l):
  linha = []
  for j in range(c):
    linha.append(0)
  matriz.append(linha)
print(matriz)
# [[0, 0], [0, 0], [0, 0]]

# Forma alternativa/compacta de inicializar uma matriz
matriz = [[0 for i in range(c)] for j in range(l)]
```

# Declaração de Matrizes

- Inicializando uma matriz de dimensões $l \times c$ e atribuindo valores de 1 até $l \times c$:

```
l = 2
c = 4
matriz = []

for i in range(l):
  linha = []
  for j in range(c):
    linha.append(i * c + j + 1)
  matriz.append(linha)

print(matriz)
# [[1, 2, 3, 4], [5, 6, 7, 8]]
```

# Acessando Elementos de uma Matriz

- Note que uma matriz nada mais é que uma lista de listas.
- Podemos acessar um elemento em uma determinada linha e coluna da seguinte forma:

```
matriz[linha - 1][coluna - 1]
# Lembrete: primeiro elemento começa na posição zero.
```

- Exemplo:

```
matriz = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
print(matriz[0][2])
# 3
print(matriz[2][1])
# 8
```

# Acessando Elementos de uma Matriz

- Similar ao que vimos em listas e tuplas, caso ocorra uma tentativa de acessar uma posição inexistente da matriz um erro será gerado.
- Exemplo:

```
matriz = [[1, 2], [3, 4]]
print(matriz[0][0])
# 1
print(matriz[1][1])
# 4
print(matriz[2][2])
# IndexError: list index out of range
```

# Modificando Elementos de uma Matriz

- Podemos alterar um elemento em uma determinada linha e coluna da seguinte forma:

```
matriz[linha - 1][coluna - 1] = valor
```

- Exemplo:

```
matriz = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
matriz[0][0] = 0
matriz[2][2] = 10
print(matriz)
# [[0, 2, 3], [4, 5, 6], [7, 8, 10]]
```

# Objetos Multidimensionais

- Matrizes são objetos bidimensionais, mas podemos criar objetos com mais dimensões.
- Podemos criar objetos com $d$ dimensões utilizando a mesma ideia de listas de listas.
- Exemplo de um objeto com dimensões $2 \times 2 \times 2$:

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

# Objetos Multidimensionais

- Acessamos um elemento em um objeto com dimensões $d_1 \times d_2 \times \cdots \times d_n$ da seguinte forma:

```
objeto[index_1][index_2]...[index_n]
```

- Exemplo:

```
obj = [[[1, 2], [3, 4]], [[5, 6], [7, 8]]] # 2 x 2 x 2
print(obj[0][0][0])
# 1
print(obj[1][0][0])
# 5
print(obj[1][1][0])
# 7
print(obj[1][1][1])
# 8
```

# Objetos Multidimensionais

- Modificamos um elemento em um objeto com dimensões $d_1 \times d_2 \times \cdots \times d_n$ da seguinte forma:

```
objeto[index_1][index_2]...[index_n] = valor
```

- Exemplo:

```
obj = [[[0, 0], [0, 0]], [[0, 0], [0, 0]]] # 2 x 2 x 2
obj[1][0][1] = 5
obj[0][1][0] = 3
print(obj)
# [[[0, 0], [3, 0]], [[0, 5], [0, 0]]]
```

# Exercícios

# Exercícios

1. Escreva uma função que leia e retorne uma matriz de inteiros fornecida pelo usuário. Sua matriz deve ler os números linha a linha. Os números devem estar separados por espaços em branco. Sua função deve interromper a leitura ao receber uma linha em branco.
2. Escreva uma função que, dada uma lista bidimensional (lista de listas), verifique se ela é uma matriz. Em caso positivo, sua função deve retornar um tupla com o número de linhas e de colunas da matriz. Em caso negativo, deve retornar uma tupla vazia.
3. Escreva uma função que imprime, linha a linha, os valores de uma matriz bidimensional dada como argumento.

# Exercício 1 - Lendo uma Matriz

```
1  def lê_matriz():
2    M = []
3    while True:
4      temp = input().split()
5      if temp == []:
6        return M
7      linha = []
8      for i in temp:
9        linha.append(int(i))
10     M.append(linha)
```

# Exercício 2 - Dimensões de uma Matriz

```python
def dimensões(M):
  linhas = len(M)
  colunas = len(M[0])
  for i in range(1, linhas):
    if len(M[i]) != colunas:
      return ()
  return (linhas, colunas)
```

# Exercício 3 - Imprimindo uma Matriz

```
1 def imprime_matriz(M):
2   (linhas, colunas) = dimensões(M)
3   for i in range(linhas):
4     for j in range(colunas):
5       print(M[i][j], end = " ")
6     print()
```

# Exercício 3 - Imprimindo uma Matriz

```
1 def imprime_matriz(M):
2   (linhas, colunas) = dimensões(M)
3   for i in range(linhas):
4     for j in range(colunas):
5       if j != colunas - 1:
6         print(M[i][j], end = " ")
7       else:
8         print(M[i][j])
```

# Exercício 3 - Imprimindo uma Matriz

```
1 def imprime_matriz(M):
2   for linha in M:
3     print(" ".join(linha))
```

# Exercícios

4. Escreva uma função que dada uma matriz ($M$), calcule a sua transposta ($M^t$). Exemplo:

$$\underset{M}{\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{bmatrix}} \qquad \underset{M^t}{\begin{bmatrix} 1 & 5 \\ 2 & 6 \\ 3 & 7 \\ 4 & 8 \end{bmatrix}}$$

5. Escreva uma função que recebe duas matrizes ($A$ e $B$). Se as duas matrizes tiverem dimensões compatíveis, sua função deve retornar a soma das duas ($C = A + B$). Caso contrário, sua função deve retornar uma lista vazia. Exemplo:

$$\underset{A}{\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}} + \underset{B}{\begin{bmatrix} 5 & 6 \\ 1 & 3 \\ 4 & 2 \end{bmatrix}} = \underset{C}{\begin{bmatrix} 6 & 8 \\ 4 & 7 \\ 9 & 8 \end{bmatrix}}$$

# Exercício 4 - Matriz Transposta

```
1 def transposta(M):
2   T = []
3   (linhas, colunas) = dimensões(M)
4   for j in range(colunas):
5     linha = []
6     for i in range(linhas):
7       linha.append(M[i][j])
8     T.append(linha)
9   return T
```

# Exercício 4 - Matriz Transposta

```
1 def transposta(M):
2   T = []
3   (linhas, colunas) = dimensões(M)
4   for j in range(colunas):
5     T.append([])
6     for i in range(linhas):
7       T[j].append(M[i][j])
8   return T
```

# Exercício 5 - Soma de Matrizes

```
1  def soma(A, B):
2    C = []
3    dim_a = dimensões(A)
4    dim_b = dimensões(B)
5    if dim_a == dim_b:
6      (linhas, colunas) = dim_a
7      for i in range(linhas):
8        linha = []
9        for j in range(colunas):
10         linha.append(A[i][j] + B[i][j])
11       C.append(linha)
12   return C
```

# Exercícios

6. Escreva uma função que recebe duas matrizes ($A$ e $B$). Se as duas matrizes tiverem dimensões compatíveis, sua função deve retornar o produto das duas ($C = A \times B$). Caso contrário, sua função deve retornar uma lista vazia. Exemplo:

$$\overset{A}{\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}} \times \overset{B}{\begin{bmatrix} 5 \\ 6 \end{bmatrix}} = \overset{C}{\begin{bmatrix} 17 \\ 39 \end{bmatrix}}$$

7. Escreva uma função que dada uma matriz quadrada, verifique se ela é uma matriz diagonal. Exemplo:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 \end{bmatrix}$$

# Exercício 6 - Multiplicação de Matrizes

```
1  def multiplicação(A, B):
2    C = []
3    dim_a = dimensões(A)
4    dim_b = dimensões(B)
5    if dim_a[1] == dim_b[0]:
6      (l, m) = dim_a
7      c = dim_b[1]
8      for i in range(l):
9        linha = []
10       for j in range(c):
11         v = 0
12         for k in range(m):
13           v = v + A[i][k] * B[k][j]
14         linha.append(v)
15       C.append(linha)
16   return C
```

# Exercícios

8. Escreva uma função que dada uma matriz quadrada, verifique se ela é uma matriz triangular inferior. Exemplo:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 & 0 \\ 0 & 4 & 4 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 9 & 0 & 2 & 3 \end{bmatrix}$$

9. Escreva uma função que dada uma matriz quadrada, verifique se ela é uma matriz triangular superior. Exemplo:

$$\begin{bmatrix} 1 & 0 & 8 & 9 & 8 \\ 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 7 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 \end{bmatrix}$$

# Exercícios

10. Uma matriz quadrada de números inteiros é um *quadrado mágico* se o valor da soma dos elementos de cada linha, de cada coluna e da diagonal principal e da diagonal secundária é o mesmo. Além disso, a matriz deve conter todos os números inteiros do intervalo $[1..n \times n]$. Exemplo:

$$\begin{bmatrix} 15 & 8 & 1 & 24 & 17 \\ 16 & 14 & 7 & 5 & 23 \\ 22 & 20 & 13 & 6 & 4 \\ 3 & 21 & 19 & 12 & 10 \\ 9 & 2 & 25 & 18 & 11 \end{bmatrix}$$

A matriz acima é um quadrado mágico, cujas somas valem 65. Escreva um programa que, dada uma matriz quadrada, verifique se ela é um *quadrado mágico*.

# Exercícios

11. Uma matriz de permutações é uma matriz quadrada cujos elementos são zeros ou uns, tal que em cada linha e em cada coluna exista exatamente um elemento igual a 1. Exemplo:

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Escreva um programa que, dada uma matriz quadrada, verifique se ela é uma matriz de permutações.

# Extra: NumPy

# NumPy

- `NumPy` é uma biblioteca para Python que contém tipos para representar vetores e matrizes juntamente com suporte para diversas operações, dentre elas operações comuns de álgebra linear e transformadas de Fourier.
- `NumPy` foi desenvolvida para obter uma maior eficiência do código em Python para aplicações científicas.

# NumPy

- Primeiramente deve-se instalar o NumPy baixando o pacote de `http://www.numpy.org`.
- Para usar esta biblioteca deve-se importá-la com o comando `import numpy`.

# Array

- O objeto mais simples da biblioteca é o `array` que serve para criar objetos homogêneos multidimensionais.
- Um `array` pode ser criado a partir de uma lista:

```
1  import numpy
2  obj = numpy.array([1, 2, 3])
3  print(type(obj))
4  # <class 'numpy.ndarray'>
5  print(obj)
6  # [1 2 3]
7  print(obj.ndim)
8  # 1
9  print(obj.size)
10 # 3
```

- Neste exemplo, usamos a biblioteca NumPy para criar um `array` de dimensão 1 com tamanho 3.

# Array

- Um `array` pode ser criado a partir de um objeto multidimensional:

```
import numpy
obj = numpy.array([[1, 2, 3], [4, 5, 6]])
print(obj)
# [[1 2 3]
#  [4 5 6]]
print(obj.ndim)
# 2
print(obj.size)
# 6
```

- Neste exemplo, usamos a biblioteca `NumPy` para criar um `array` de dimensão 2 com tamanho 6.

# Método arange

- Um **array** também pode ser criado utilizando o método **arange**.
- O método **arange** cria um **array** com apenas uma dimensão (lista), similar a função **range**.
- Exemplo:

```
1  import numpy
2  obj = numpy.arange(6)
3  print(obj)
4  # [0 1 2 3 4 5]
5  obj = numpy.arange(1, 6)
6  print(obj)
7  # [1 2 3 4 5]
8  obj = numpy.arange(1, 6, 2)
9  print(obj)
10 # [1 3 5]
```

# Método `reshape`

- Objetos do tipo `array` podem ter suas dimensões modificadas utilizando o método `reshape`.
- Esse método recebe como parâmetros os tamanhos das dimensões desejadas.
- Exemplo:

```
import numpy
obj = numpy.arange(6)
print(obj)
# [0 1 2 3 4 5]
print(obj.reshape(2, 3))
# [[0 1 2]
#  [3 4 5]]
print(obj.reshape(3, 2))
# [[0 1]
#  [2 3]
#  [4 5]]
```

# Método `zeros`

- `NumPy` possui o método `zeros` que cria um `array` contendo apenas zeros. O parâmetro desse método é uma tupla com os tamanhos de cada dimensão.

```
1 import numpy
2 print(numpy.zeros((3)))
3 # [0. 0. 0.]
4 print(numpy.zeros((3, 4)))
5 # [[0. 0. 0. 0.]
6 #  [0. 0. 0. 0.]
7 #  [0. 0. 0. 0.]]
```

# Método ones

- NumPy possui também um método ones que cria um array inicializado todos os elementos com valor 1.

```
1 import numpy
2 print(numpy.ones((2, 5)))
3 # [[1. 1. 1. 1. 1.]
4 #  [1. 1. 1. 1. 1.]]
```

# Método `transpose`

- `Arrays` possuem o método `transpose`, que retorna a matriz transposta.
- Exemplo:

```
1  import numpy
2  v = numpy.arange(1, 7)
3  m = v.reshape(2, 3)
4  print(m)
5  # [[1 2 3]
6  #  [4 5 6]]
7  t = m.transpose()
8  print(t)
9  # [[1 4]
10 #  [2 5]
11 #  [3 6]]
```

# Operadores para Arrays

- Os operadores `+`, `-`, `*`, `/`, `**`, `//` e `%`, quando utilizados sobre `arrays`, são aplicados em cada posição dos mesmos.

```
1  import numpy
2  M1 = numpy.array([[1, 2, 3], [4, 5, 6]])
3  M2 = numpy.array([[6, 5, 4], [3, 2, 1]])
4  print(M1 + 2)
5  # [[3 4 5]
6  #  [6 7 8]]
7  print(M2 % 2)
8  # [[0 1 0]
9  #  [1 0 1]]
10 print(M * 2)
11 # [[12 10  8]
12 #  [ 6  4  2]]
```

# Operadores para Arrays

- Os operadores `+`, `-`, `*`, `/`, `**`, `//` e `%`, quando utilizados sobre `arrays`, são aplicados em cada posição dos mesmos.

```
1  import numpy
2  M1 = numpy.array([[1, 2, 3], [4, 5, 6]])
3  M2 = numpy.array([[6, 5, 4], [3, 2, 1]])
4  print(M1 + M2)
5  # [[7 7 7]
6  #  [7 7 7]]
7  print(M1 // M2)
8  # [[0 0 0]
9  #  [1 2 6]]
10 print(M1 * M2)
11 # [[ 6 10 12]
12 #  [12 10  6]]
```

# Método dot

- `Arrays` possuem o método `dot`, que calcula a multiplicação de matrizes.
- Como parâmetro, o método recebe outra matriz.
- Exemplo:

```
1  import numpy
2  A = numpy.arange(2, 6).reshape(2, 2)
3  B = numpy.arange(1, 5).reshape(2, 2)
4  print(A)
5  # [[2 3]
6  #  [4 5]]
7  print(B)
8  # [[1 2]
9  #  [3 4]]
10 print(A.dot(B))
11 # [[11 16]
12 #  [19 28]]
```

# NumPy

- Na biblioteca `NumPy` existe uma variedade de outras funções e métodos, por exemplo, para calcular autovalores e autovetores, para resolução de sistemas de equações lineares, etc.
- A biblioteca fornece uma documentação completa: `https://numpy.org/devdocs/reference`.