

**BCC701 – Programação de Computadores I**

Universidade Federal de Ouro Preto

Departamento de Ciência da Computação

[www.decom.ufop.br/gustavo](http://www.decom.ufop.br/gustavo)



## Aula Teórica 10

### Matrizes

Material Didático Proposto

( 1 )

BCC701

# Agenda

- Introdução;
- Declaração de Matrizes;
- Algumas operações com matrizes;
- Algumas funções aplicadas a matrizes;
- Exercícios.

# Introdução

- Utilizaremos a mesma estrutura dos vetores: *lista*
- Consideramos que ela representa "conceitualmente" uma matriz quando armazenar apenas valores de um mesmo tipo e que tem uma estrutura definida para  $A_{m \times n}$
- Isso deverá ser garantido em nossos programas

# Conjunto de variáveis

- Ao estudar **vetores** observamos que, em determinadas situações, é necessário utilizar muitas variáveis com um propósito comum. Relembrando exemplos:
  - Para armazenar três notas de um mesmo aluno:
    - `Nota(1) = float(input('Digite a nota 1: '))`
    - `Nota(2) = float(input('Digite a nota 2: '))`
    - `Nota(3) = float(input('Digite a nota 3: '))`
  - Ler um vetor com cinco números , VALOR POR VALOR:
    - `Vetor = []`
    - `for i in range( 5):`
      - `n = float(input('Digite um número: '))`
      - `Vetor.append(n)`

# Conjunto de variáveis

Lendo um vetor com cinco números reais, elemento a elemento:

```
1 vetor = []
2 for i in range(5):
3     num = float(input(f"v[{i}] = "))
4     vetor.append(num)
5     print(f"Conteúdo do vetor: {vetor}")
6 print("Vetor completo\n==== Fim do programa ====")
7
```

Shell ×

```
>>> %Run lendo_vetor.py

v[0] = 34
Conteúdo do vetor: [34.0]
v[1] = 56
Conteúdo do vetor: [34.0, 56.0]
v[2] = 32.2
Conteúdo do vetor: [34.0, 56.0, 32.2]
v[3] = 4.5
Conteúdo do vetor: [34.0, 56.0, 32.2, 4.5]
v[4] = 56
Conteúdo do vetor: [34.0, 56.0, 32.2, 4.5, 56.0]
Vetor completo
==== Fim do programa ====
```

# O tipo de dados **Matriz**

- Agora imagine a seguinte situação:
  - Desejo armazenar **3 notas** para **5 alunos**;
  - Para isto eu preciso de **3 vetores** ou de **5 vetores**?

# O tipo de dados Matriz

- Agora imagine a seguinte situação:
  - Desejo armazenar **3 notas** para **5 alunos**;
  - Para isto eu preciso de **3 vetores** ou de **5 vetores**?
  - **Nenhum dos dois:** podemos utilizar uma matriz na qual **cada linha representa um aluno** e **cada coluna representa uma nota**:

	Nota 1	Nota 2	Nota 3
Aluno 1	8.1	9.2	6.0
⋮	5.2	6.8	9.5
⋮	6.0	6.1	6.2
⋮	3.5	5.2	8.3
Aluno 5	2.4	1.5	5.3

# O tipo de dados **Matriz**

- Matrizes são variáveis que contêm uma quantidade grande de valores do mesmo tipo;
- Assim como nos vetores, elementos da matriz são acessados através de índices, no caso de **dois índices, linha e coluna**;
- Uma matriz bidimensional **A**, tem dimensão **m x n**, ou seja, de **m** linhas e **n** colunas:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

- **OBS:** Um vetor corresponde a uma matriz **m x 1** (no caso de um **vetor coluna**), ou uma matriz **1 x n** (no caso de um **vetor linha**).



# Exemplos de uso de Matriz

- Na resolução de sistemas de equações lineares:

- Dado um sistema linear do tipo:  $\mathbf{A} * \mathbf{X} = \mathbf{B}$ ;

- A solução é obtida resolvendo:  $\mathbf{X} = \mathbf{A}^{-1} * \mathbf{B}$ ;

- Exemplo:

$$\begin{array}{l} 3x + y + 2z = 13 \\ x + y - 8z = -1 \\ -x + 2y + 5z = 13 \end{array} \quad \longrightarrow \quad \begin{array}{ccc} \left[ \begin{array}{ccc} 3 & 1 & 2 \\ 1 & 1 & -8 \\ -1 & 2 & 5 \end{array} \right] & \left[ \begin{array}{c} x \\ y \\ z \end{array} \right] & = \left[ \begin{array}{c} 13 \\ -1 \\ 13 \end{array} \right] \\ \mathbf{A}_{33} & \mathbf{X}_{31} & \mathbf{B}_{31} \end{array}$$

# Exemplos de uso de Matriz

- Na resolução de sistemas de equações lineares:

- Exemplo:

$$\begin{array}{rcl}
 3x + y + 2z & = & 13 \\
 x + y - 8z & = & -1 \\
 -x + 2y + 5z & = & 13
 \end{array}
 \longrightarrow
 \begin{array}{ccc}
 \begin{bmatrix} 3 & 1 & 2 \\ 1 & 1 & -8 \\ -1 & 2 & 5 \end{bmatrix} & \begin{bmatrix} x \\ y \\ z \end{bmatrix} & = \begin{bmatrix} 13 \\ -1 \\ 13 \end{bmatrix} \\
 A_{33} & X_{31} & B_{31}
 \end{array}$$

-->  $A = [[3, 1, 2], [1, 1, -8], [-1, 2, 5]]$

-->  $B = [13, -1, 13]$

-->  $X = A^{-1} * B$  => feito facilmente no Python, ver pacote **numpy**

$X =$  2.  
5.  
1.

Assim, chega-se à solução:

$x = 2, y = 5, z = 1.$

# Exemplos de uso de Matriz

- Na resolução de sistemas de equações lineares:

- Exemplo:

$$\begin{array}{rcl}
 3x + y + 2z = 13 \\
 x + y - 8z = -1 \\
 -x + 2y + 5z = 13
 \end{array}
 \longrightarrow
 \begin{array}{ccc}
 \begin{bmatrix} 3 & 1 & 2 \\ 1 & 1 & -8 \\ -1 & 2 & 5 \end{bmatrix} & \begin{bmatrix} x \\ y \\ z \end{bmatrix} & = \begin{bmatrix} 13 \\ -1 \\ 13 \end{bmatrix} \\
 A_{33} & X_{31} & B_{31}
 \end{array}$$

-->  $A = [[3, 1, 2], [1, 1, -8], [-1, 2, 5]]$

-->  $B = [13, -1, 13]$

→  $X = A^{-1} * B$  => feito facilmente no Python, ver numpy

→  $A\_1 = \text{numpy.linalg.inv}(A)$  # dar import numpy

→  $X = \text{numpy.dot}(A\_1, B)$

$X = 2. \ 5. \ 1.$

Introdução;

**Declaração de matrizes;**

Algumas operações com matrizes;

Algumas funções aplicadas a matrizes;

Exercícios.

# DECLARAÇÃO DE MATRIZES

# Criando Matrizes em lote

- Criando matrizes:

# Criando uma matriz 3x3 de inteiros

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

# Criando uma matriz de 3x3 números reais

```
A = [ [ 1.5, 2.6, 3.7 ], [ 4.1, 5.2, 6.3 ], [ 7.2, 8.4, 9.6 ] ]
```

# Criando uma matriz 3x3 de *strings*

```
A = [ [ 'João', 'Maria', 'José' ], [ 'Arroz', 'Feijão', 'Pão' ], [ 'a', 'b', 'c' ] ]
```

# Matrizes: Operações

Podemos realizar várias operações com matrizes.

Listamos aqui as operações exploradas em nosso curso:

- Acessar e modificar elementos
- Inserir elementos
- Remover elementos
- Descobrir as dimensões (quantidade de linhas e colunas)

# Inicializando por atribuição

$A = \begin{bmatrix} \textcolor{red}{1}, 2, 3 \\ 4, 5, \textcolor{blue}{6} \\ 7, 8, 9 \end{bmatrix}$

$v1 = [1, 2, 3]$

$v2 = [4, 5, 6]$

$v3 = [7, 8, 9]$

$A = [v1, v2, v3]$

$A[0]$  contém  $v1$

$A[0][0]$  contém  $v1[0]$ , o.s.

$A[0][0]$  contém o valor  $\textcolor{green}{1}$

$A[1][2]$  contém o valor  $\textcolor{blue}{6}$

# Matrizes: Operações

Acessar e modificar elementos

# inicializando a matriz A **por atribuição em lote**

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

```
val = A[0][0]           # val = 1
```

```
print(A[1][2])          # 6 será impresso no terminal
```

```
A[2][0] += 5            # A[2][0] = 7 + 5 = 12
```

```
A[0][0] = 0             # A = [ [ 0, 2, 3 ], [ 4, 5, 6 ], [ 12, 8, 9 ] ]
```

```
A[1][0] = 0             # A = [ [ 0, 2, 3 ], [ 0, 5, 6 ], [ 12, 8, 9 ] ]
```

```
A[2][2] = 0             # A = [ [ 0, 2, 3 ], [ 0, 5, 6 ], [ 12, 8, 0 ] ]
```



# Matrizes: Operações

Inserindo uma nova linha

```
A.append( [ 10, 11, 12 ] ) # insere a linha [10, 11, 12] à matriz A
```

```
# A = [ [ 1, 2, 3 ], [ 4, 5, 6 ], [ 7, 8, 9 ], [ 10, 11, 12 ] ]
```

```
# matriz A agora tem dimensão 4 × 3 !
```

Inserindo uma nova coluna

```
A = [ [ 1, 2, 3 ], [ 4, 5, 6 ], [ 7, 8, 9 ] ] # matriz A 3 × 3
```

```
A[0].append(10) # A = [ [ 1, 2, 3, 10 ], [ 4, 5, 6 ], [ 7, 8, 9 ] ]
```

```
A[1].append(11) # A = [ [ 1, 2, 3, 10 ], [ 4, 5, 6, 11 ], [ 7, 8, 9 ] ]
```

```
A[2].append(12) # A = [ [ 1, 2, 3, 10 ], [ 4, 5, 6, 11 ], [ 7, 8, 9, 12 ] ]
```

**Obs.:** Momentaneamente deixa de ser uma matriz

# Matrizes: Operações

Descobrir as dimensões (quantidade de linhas e colunas)

1. `A = [ [ 1, 2, 3 ], [ 4, 5, 6 ] ]`
2. `nLinhas = len(A)`
3. `if nLinhas > 0:`
4.     `nColunas = len(A[0])`
5. `else:`
6.     `nColunas = 0`

Ou simplesmente, sabendo que A tem pelo um elemento `A = [[ ]]`:

`nLinhas = len(A)`                   # retorna o total de linhas

`nColunas = len(A[0])`           # retorna o total de colunas da linha 0

Percorrendo uma matriz posição a posição para ler, modificar, comparar, imprimir etc.

Podemos percorrer uma matriz (tabela), de forma ordenada, por linhas ou por colunas

- Em matrizes, normalmente, é necessidade usar dois laço, um dentro do corpo do outro laço.
- Para percorrer uma matriz temos que percorrer as linhas e para cada linha, percorrer as suas colunas.

	↓ j = 1	↓ j = 2	↓ j = 3	↓ j = 4
→ i = 1	a11	a12	a13	a14
	a21	a22	a23	a24
	a31	a32	a33	a34

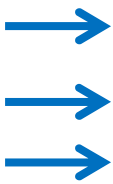
- Em matrizes, normalmente, é necessidade usar dois laço, um dentro do corpo do outro laço.
- Para percorrer uma matriz temos que percorrer as linhas e para cada linha, percorrer as suas colunas.

	↓ j = 1	↓ j = 2	↓ j = 3	↓ j = 4
	a11	a12	a13	a14
→ i = 2	a21	a22	a23	a24
	a31	a32	a33	a34

- Considerando tabelas/matrizes, normalmente, é necessidade usar dois laço, um dentro do corpo do outro.
- Para percorrer uma matriz temos que percorrer as linhas e para cada linha, percorrer as suas colunas.

	↓ j = 1	↓ j = 2	↓ j = 3	↓ j = 4
	a11	a12	a13	a14
	a21	a22	a23	a24
→ i = 3	a31	a32	a33	a34

## Percorrendo uma tabela por linha



a11	a12	a13	a14
a21	a22	a23	a24
a31	a32	a33	a34

```
for lin in range(3):  
    print(f"\nlinha {lin}")
```

Saída:

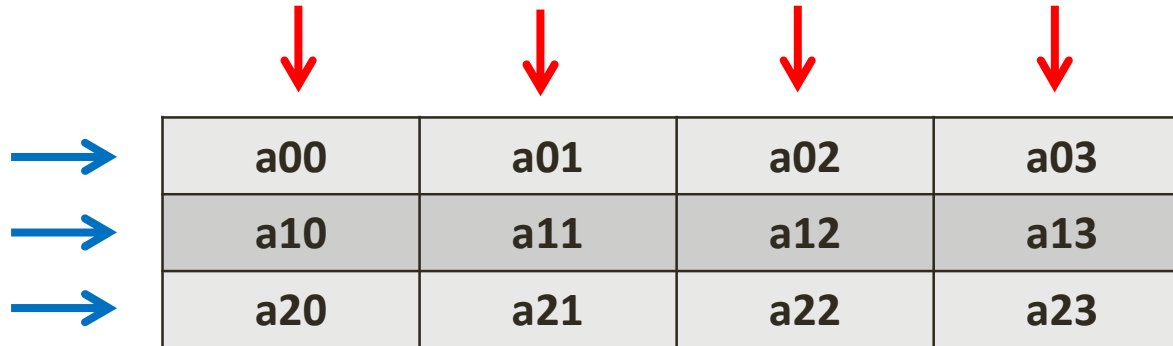
linha 0

linha 1

linha 2

Lembrando sempre que, no Python, as posições dos vetores/matrizes começam na posição 0.

## Percorrendo uma tabela por linha



	a00	a01	a02	a03
	a10	a11	a12	a13
	a20	a21	a22	a23

```
for lin in range(3):
    print(f"\nlinha {lin}")
    for coluna in range(4):
        print(f"col. {coluna}")
    print(f" :: fim da linha {lin}")
```

### Saída:

```
linha 0 : col. 0 col. 1 col. 2 col. 3 :: fim da linha 0
linha 1 : col. 0 col. 1 col. 2 col. 3 :: fim da linha 1
linha 2 : col. 0 col. 1 col. 2 col. 3 :: fim da linha 2
```



Percorrendo a matriz A por linha:

1. laço das linhas # **fixa cada linha**

2. laço das colunas # **percorre todas as colunas da linha fixada**

**A[linha][coluna]**

```
1 print("==== Matrizes ==== ")
2 A = [[1, 2, 3], [4, 6, 8], [9, 7, 5]]
3 print(A)
4
5 for lin in range(3):
6     print(f"\nlinha {lin}: ", end = "")
7     for col in range(3):
8         print(f" A[{lin}, {col}] = {A[lin][col]} || ", end = "")
9
```

Shell ×

```
>>> %Run percorre_tabela.py
```

```
==== Matrizes ====
```

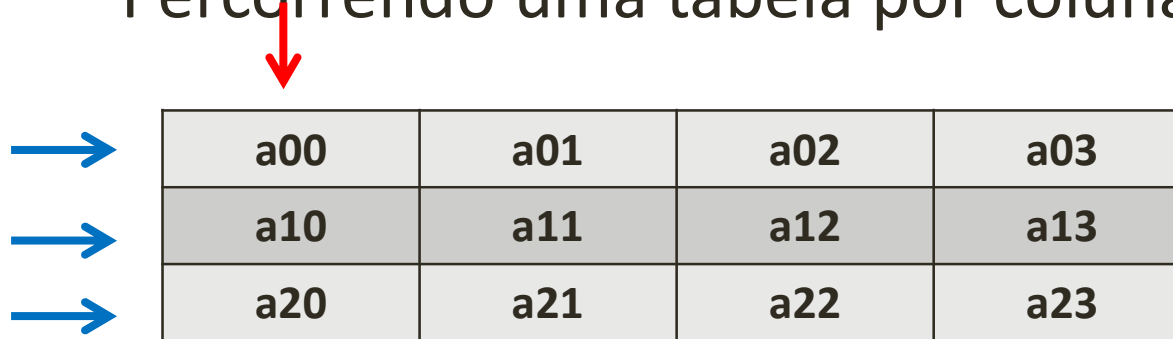
```
[[1, 2, 3], [4, 6, 8], [9, 7, 5]]
```

```
linha 0: A[0, 0] = 1 || A[0, 1] = 2 || A[0, 2] = 3 ||
```

```
linha 1: A[1, 0] = 4 || A[1, 1] = 6 || A[1, 2] = 8 ||
```

```
linha 2: A[2, 0] = 9 || A[2, 1] = 7 || A[2, 2] = 5 ||
```

## Percorrendo uma tabela por coluna



a00	a01	a02	a03
a10	a11	a12	a13
a20	a21	a22	a23

```
for col in range(4):  
    print(f"coluna {col}:")
```

Saída:

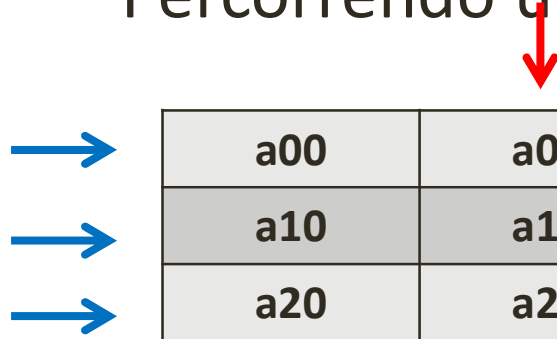
coluna 0:

coluna 1:

coluna 2:

coluna 3:

## Percorrendo uma tabela por coluna



a00	a01	a02	a03
a10	a11	a12	a13
a20	a21	a22	a23

```
for col in range(4):  
    print(f"coluna {col}:")
```

Saída:

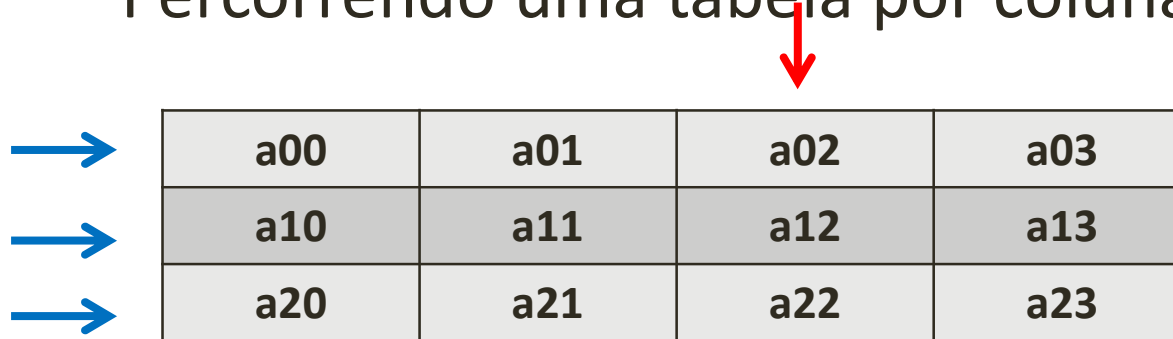
coluna 0:

coluna 1:

coluna 2:

coluna 3:

## Percorrendo uma tabela por coluna



a00	a01	a02	a03
a10	a11	a12	a13
a20	a21	a22	a23

```
for col in range(4):  
    print(f"coluna {col}:")
```

Saída:

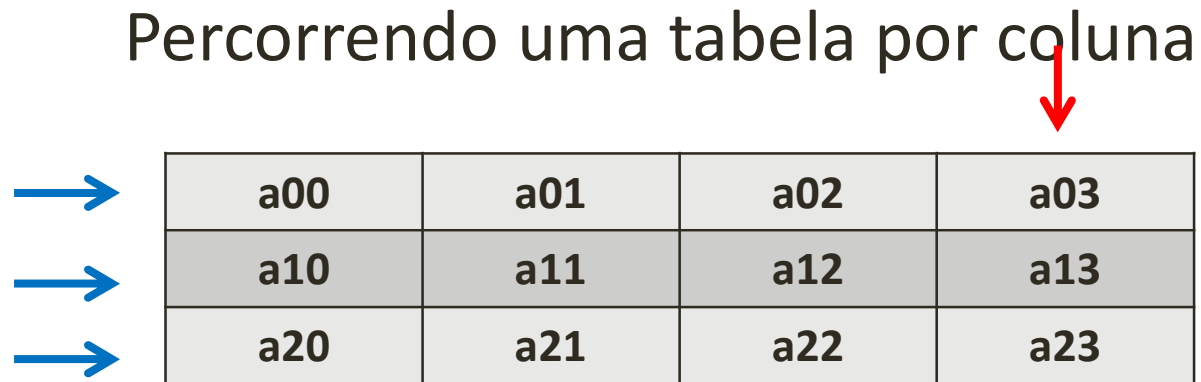
coluna 0:

coluna 1:

coluna 2:

coluna 3:

## Percorrendo uma tabela por coluna



a00	a01	a02	a03
a10	a11	a12	a13
a20	a21	a22	a23

```
for col in range(4):  
    print(f"coluna {col}:")
```

Saída:

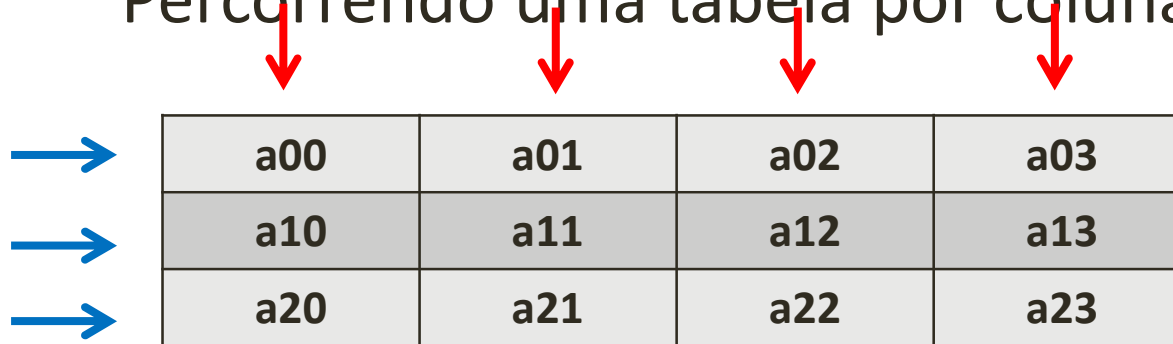
coluna 0:

coluna 1:

coluna 2:

coluna 3:

## Percorrendo uma tabela por coluna



a00	a01	a02	a03
a10	a11	a12	a13
a20	a21	a22	a23

```

for col in range(4):
    print(f"coluna {col}")
    for lin in range(3):
        print(f"linha {lin} ", end = "")
    print(f" :: fim da coluna {col}")
  
```

### Saída:

```

coluna 0: linha 0 linha 1 linha 2
coluna 1: linha 0 linha 1 linha 2
coluna 2: linha 0 linha 1 linha 2
coluna 3: linha 0 linha 1 linha 2
  
```

Percorrendo a matriz A por coluna:

1. laço das colunas # **fixa cada coluna**

2. laço das linhas # **percorre todas as linhas da coluna fixada**

**A[linha][coluna]**

```
1 print("==== Matrices ==== ")
2 A = [[1, 2, 3], [4, 6, 8], [9, 7, 5]]
3 print(A)
4
5 for col in range(3):
6     print(f"\ncoluna {col}: ", end = "")
7     for lin in range(3):
8         print(f" A[{lin}], {col}] = {A[lin][col]} || ", end = "")
9
```

Shell ×

```
>>> %Run percorre_tabela.py
```

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

coluna 0: A[0, 0] = 1 || A[1, 0] = 4 || A[2, 0] = 9 ||
coluna 1: A[0, 1] = 2 || A[1, 1] = 6 || A[2, 1] = 7 ||
coluna 2: A[0, 2] = 3 || A[1, 2] = 8 || A[2, 2] = 5 ||
```

# Matrizes × Vetores

Os vetores têm um único índice, portanto basta **um único laço for** para percorrê-lo.

```
for i in range(n):
    v[i] acessa cada um dos n elementos do vetor v
```

$$V = (v_1, v_2, v_3, \dots, v_n)$$

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

As matrizes têm dois índices, portanto são necessários **dois laços for** para percorrê-las. Percorrendo por linhas:

```
for i in range(tot_linhas):
    for j in range(tot_colunas):
        A[i][j] acessa cada um dos elementos da matriz A
```

Percorrendo A por colunas:

```
for j in range(tot_colunas):
    for i in range(tot_linhas):
        A[i][j] acessa cada um dos elementos da matriz A
```



# Criando um vetor elemento a elemento

Lendo um vetor com cinco valores reais:

```
1 vetor = []
2 for i in range(5):
3     num = float(input(f"v[{i}] = "))
4     vetor.append(num)
5     print(f"Conteúdo do vetor: {vetor}")
6 print("Vetor completo\n==== Fim do programa ====")
7
```

Shell ×

```
>>> %Run lendo_vetor.py
```

```
v[0] = 34
Conteúdo do vetor: [34.0]
v[1] = 56
Conteúdo do vetor: [34.0, 56.0]
v[2] = 32.2
Conteúdo do vetor: [34.0, 56.0, 32.2]
v[3] = 4.5
Conteúdo do vetor: [34.0, 56.0, 32.2, 4.5]
v[4] = 56
Conteúdo do vetor: [34.0, 56.0, 32.2, 4.5, 56.0]
Vetor completo
==== Fim do programa ====
```

# Criando uma Matriz linha a linha

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

Cria/ler uma linha de cada vez em um vetor e depois inclui este vetor na matriz

```
>>> matriz = []
>>> vetor1 = [1, 3, 6]
>>> matriz.append(vetor1)
>>> matriz
[[1, 3, 6]]                                     # matriz 1 x 3
```

# Criando uma Matriz linha a linha

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

Cria/ler uma linha de cada vez em um vetor e depois inclui este vetor na matriz

```
>>> matriz = []
>>> vetor1 = [1, 3, 6]
>>> matriz.append(vetor1)
>>> matriz
[[1, 3, 6]]
>>> len(matriz)           # total de linhas
>>> 1
>>> len(matriz[0])        # total de elementos da linha 0,
>>> 3                     # ou seja, a matriz é 1 x 3
>>> vetor2 = [5, 7, 9]
>>> matriz.append(vetor2)
>>> matriz
[[1, 3, 6], [5, 7, 9]]    # matriz 2 x 3
```

# Criando uma Matriz linha a linha

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

Cria/ler uma linha de cada vez em um vetor e depois inclui este vetor na matriz

```
>>> matriz = []
>>> vetor1 = [1, 3, 6]
>>> matriz.append(vetor1)
>>> matriz
[[1, 3, 6]]                                     # matriz 1 x 3
>>> vetor2 = [5, 7, 9]
>>> matriz.append(vetor2)
>>> matriz
[[1, 3, 6], [5, 7, 9]]                         # matriz 2 x 3
>>> vetor3 = [50, 70, 90]
>>> matriz.append(vetor3)
>>> matriz
[[1, 3, 6], [5, 7, 9], [50, 70, 90]]          # matriz 3 x 3
```

**Problema por usarmos listas para simular Matriz no Python**

# Criando um Matriz elemento a elemento

- Uma matriz é criada incluindo suas linhas, uma a uma na estrutura.
- Portanto, se vamos incluir elemento por elemento, devemos primeiro criar as linhas, armazenadas em um vetores.
- Finalmente, incluiremos este vetor na matriz.
- Este procedimento deve ser repetido tantas vezes tanto for o número de linhas da matriz.

# Criando um Matriz elemento a elemento

```
1 linhas = 3
2 colunas = 2
3 matriz = [ ] # uma lista vazia para representar a matriz
4 for lin in range(linhas):
5     linha = [ ] # uma lista vazia para representar a linha
6     for col in range(colunas):
7         n = float(input(f'M[{lin}][{col}]: '))
8         linha.append(n) # preenchendo valor da coluna
9     print(f"linha {lin}: {linha}")
10    matriz.append(linha) # colocando a linha na matriz
11    print(f"Matriz: {matriz}\n")
12    print("Matriz completa\n==== Fim ==== ")
```

Shell ×

```
M[0][0]: 4.5
M[0][1]: 6.7
linha 0: [4.5, 6.7]
Matriz: [[4.5, 6.7]]

M[1][0]: 9.2
M[1][1]: 3.7
linha 1: [9.2, 3.7]
Matriz: [[4.5, 6.7], [9.2, 3.7]]

M[2][0]: 6.9
M[2][1]: 10
linha 2: [6.9, 10.0]
Matriz: [[4.5, 6.7], [9.2, 3.7], [6.9, 10.0]]

Matriz completa
==== Fim ====
```

# Exercícios

1. Escreva um programa que leia a matriz A usando o comando **eval** (em lote) e posteriormente encontre o maior elemento de cada linha, mostrando o resultado.

$$A = \begin{bmatrix} 1 & 2 & 7 \\ 4 & 6 & -1 \\ -2 & -10 & -8 \end{bmatrix}, \text{ linha 1} \rightarrow \text{maior valor 7, na coluna } \underline{3} \text{ etc.}$$

2. Leia uma matriz A em lote e posteriormente somar os elementos de cada linha da matriz. Informar esta soma

$$\text{Exemplo : } A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 6 & -2 \end{bmatrix}, \text{ então } V = \begin{bmatrix} 3 \\ 7 \\ 4 \end{bmatrix}$$