Introdução ao Processamento de Dados
Turma 3 (2020.1)

Introdução a Matrizes

Gilson. A. O. P. Costa (IME/UERJ)

gilson.costa@ime.uerj.br

Cópia de Vetores

- Para copiar o valor de uma variável de um tipo simples, e.g., *int*, *float*, *string*, basta fazer uma atribuição:

```
>>> a = 10
>>> b = a
>>> print('a = ', a, '; b = ', b)
a = 10 ; b = 10
>>> b = 5
>>> print('a = ', a, '; b = ', b)
a = 10 ; b = 5
```

Cópia de Vetores

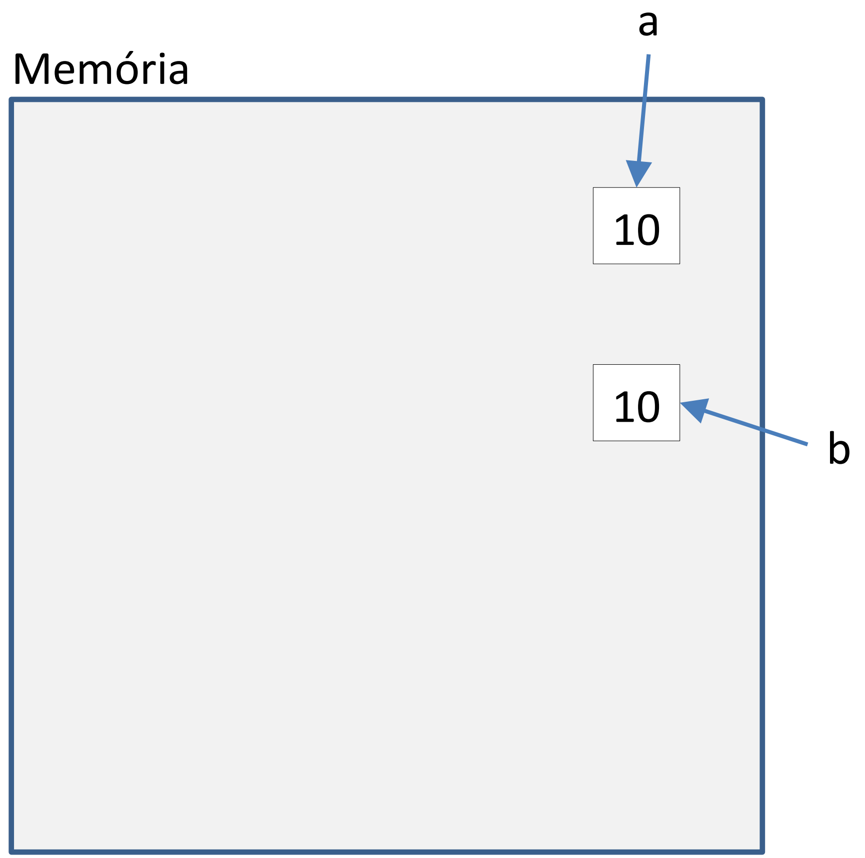
- Para copiar o valor de uma variável de um tipo simples, e.g., *int*, *float*, *string*, basta fazer uma atribuição.
- A cópia de vetores/listas não funciona da mesma maneira (no Python)!

```
>>> vetor_a = [10, 20, 30]
>>> vetor_b = vetor_a
>>> print('vetor_a = ', vetor_a, '; vetor_b = ', vetor_b)
vetor_a = [10, 20, 30] ; vetor_b = [10, 20, 30]
>>> vetor_b[1] = 0
>>> print('vetor_a = ', vetor_a, '; vetor_b = ', vetor_b)
vetor_a = [10, 0, 30] ; vetor_b = [10, 0, 30]
```

Cópia de Vetores

- Na realidade, um atribuição do tipo *vetor_b = vetor_a* vai criar apenas um novo nome (apontador) para o vetor identificado por *vetor_a*.

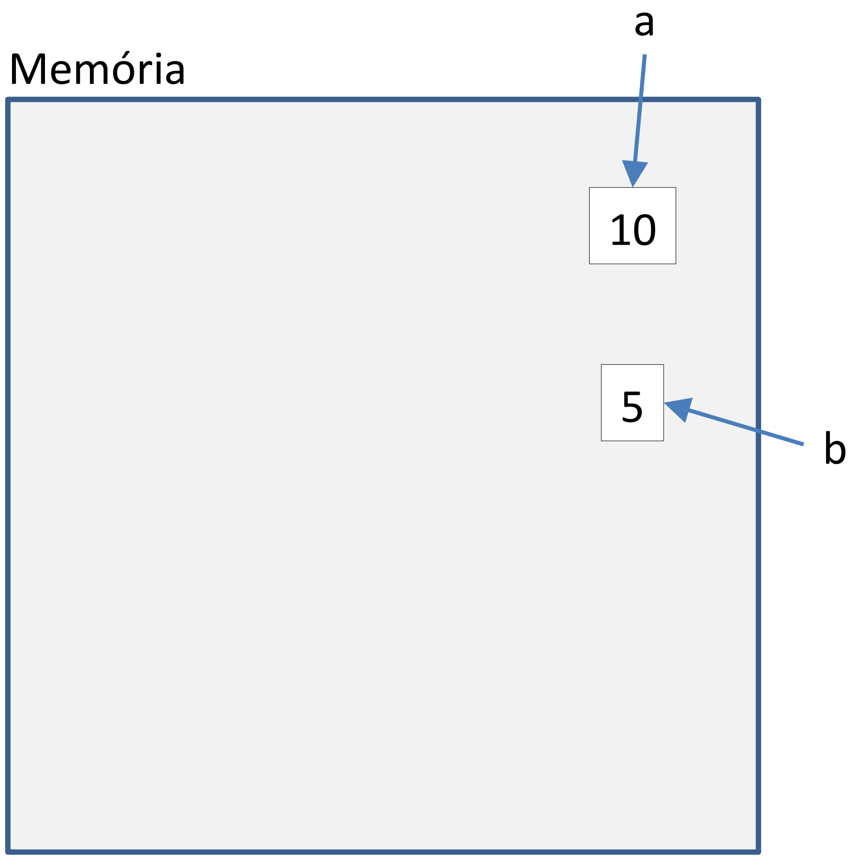
```
>>> a = 10
>>> b = a
>>> b = 5
```



Cópia de Vetores

- Na realidade, um atribuição do tipo *vetor_b = vetor_a* vai criar apenas um novo nome (apontador) para o vetor identificado por *vetor_a*.

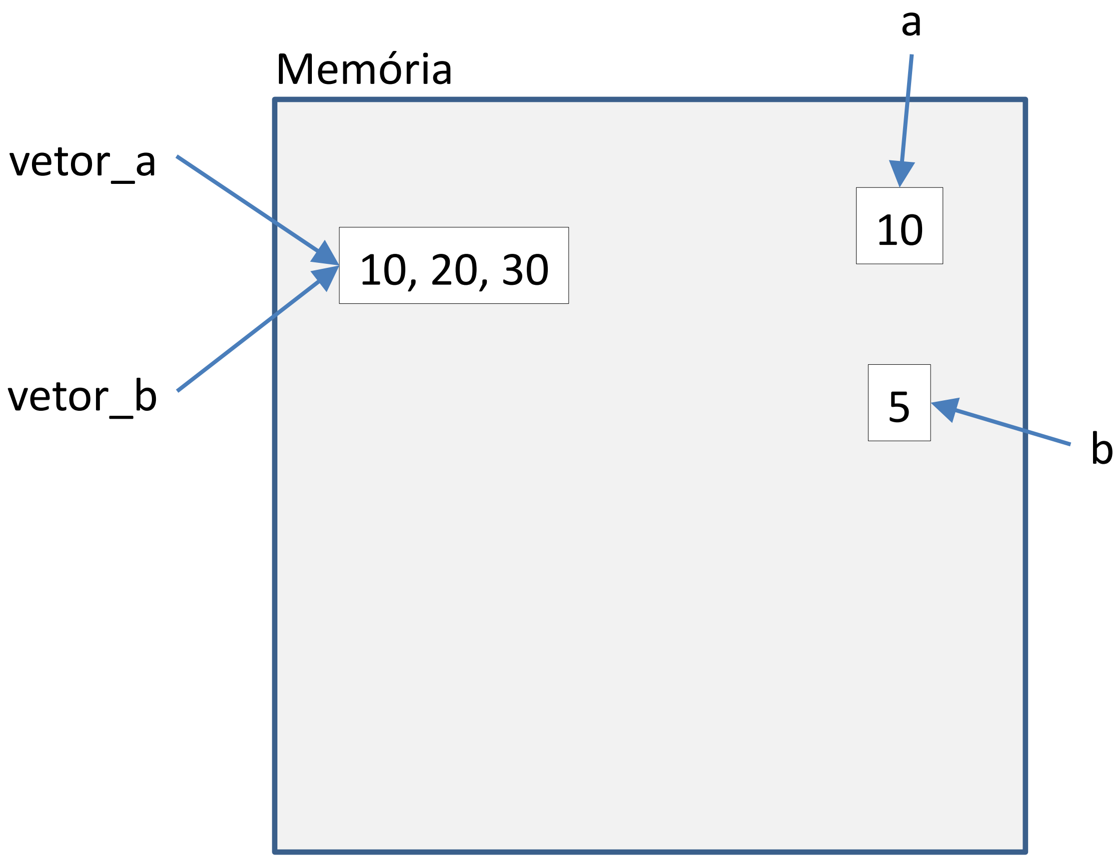
```
>>> a = 10
>>> b = a
>>> b = 5
```



Cópia de Vetores

- Na realidade, um atribuição do tipo *vetor_b = vetor_a* vai criar apenas um novo nome (apontador) para o vetor identificado por *vetor_a*.

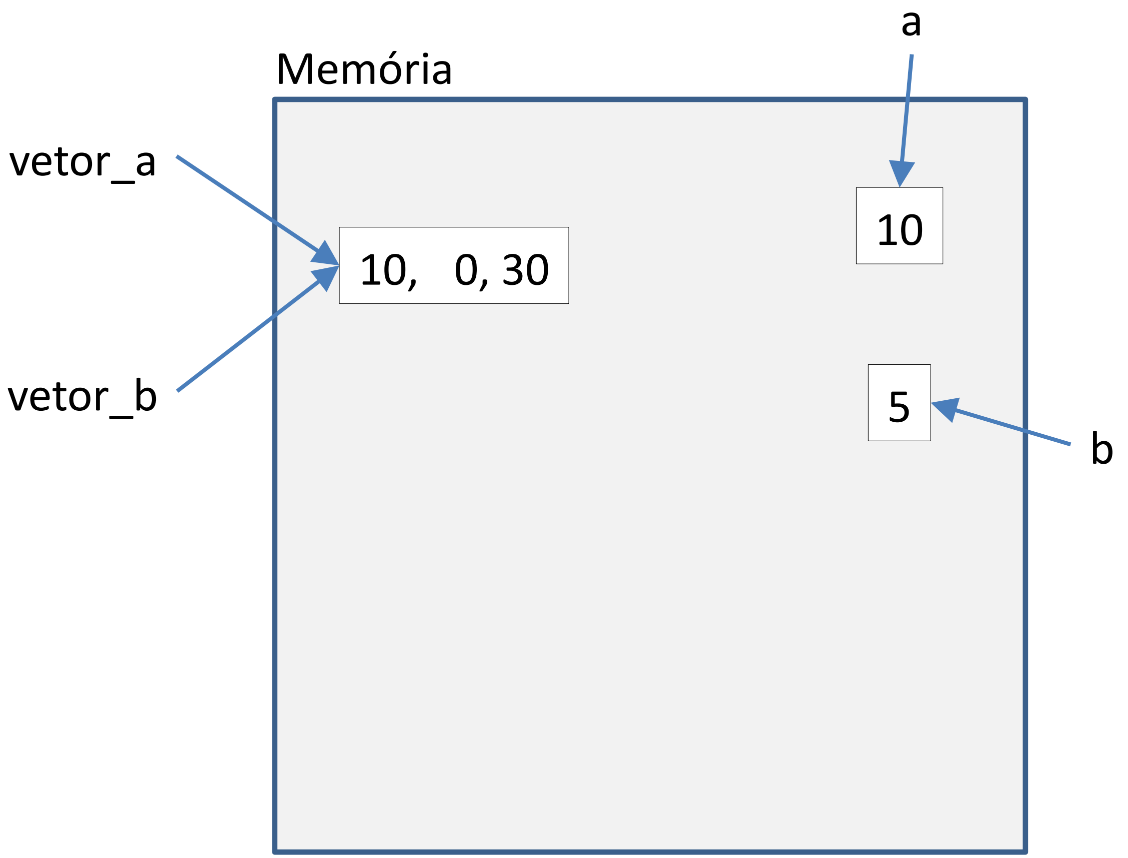
```
>>> vetor_a = [10, 20, 30]
>>> vetor_b = vetor_a
>>> vetor_b[1] = 0
```



Cópia de Vetores

- Na realidade, um atribuição do tipo *vetor_b = vetor_a* vai criar apenas um novo nome (apontador) para o vetor identificado por *vetor_a*.

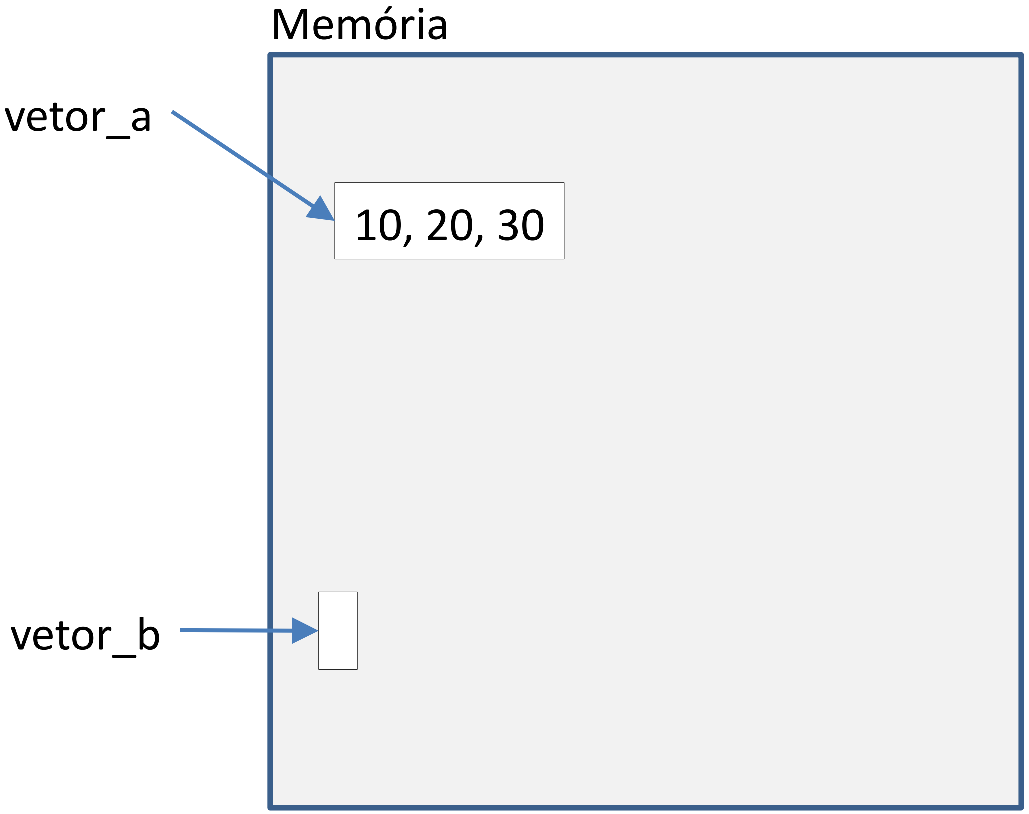
```
>>> vetor_a = [10, 20, 30]
>>> vetor_b = vetor_a
>>> vetor_b[1] = 0
```



Cópia de Vetores

- Para copiar os elementos do vetor/lista para uma nova estrutura de dados na memória, pode-se fazê-lo elemento a elemento.

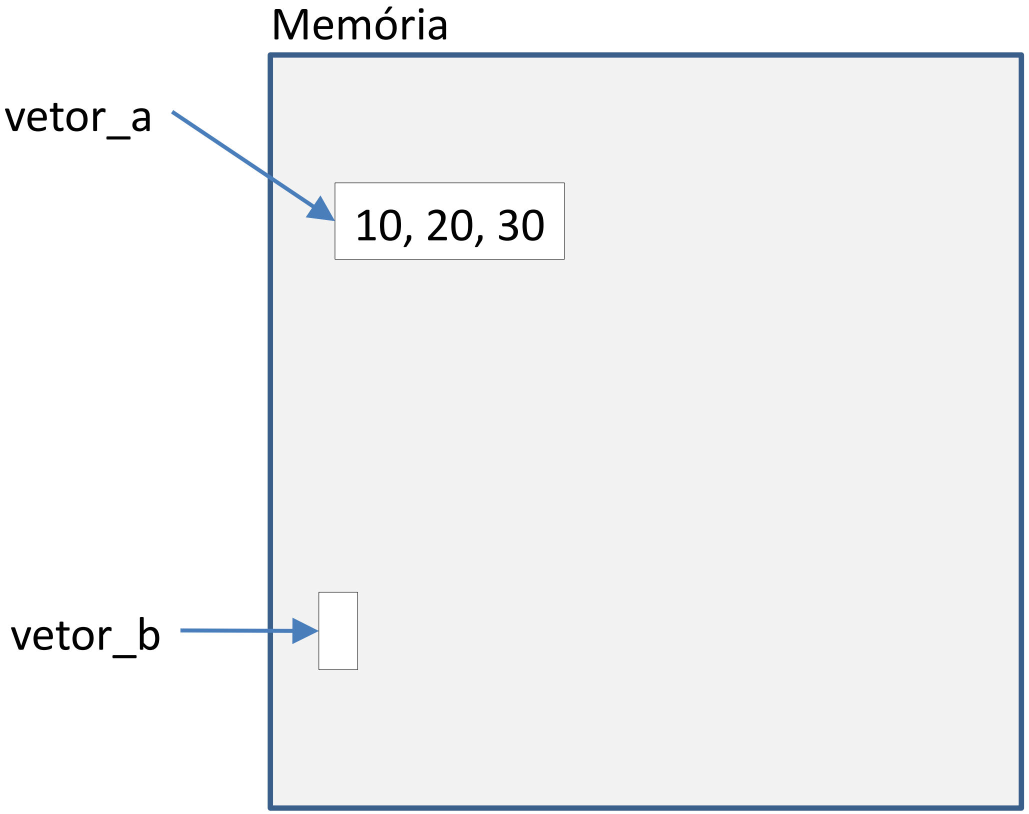
```
vetor_a = [10, 20, 30]
vetor_b = []
```



Cópia de Vetores

- Para copiar os elementos do vetor/lista para uma nova estrutura de dados na memória, pode-se fazê-lo elemento a elemento.

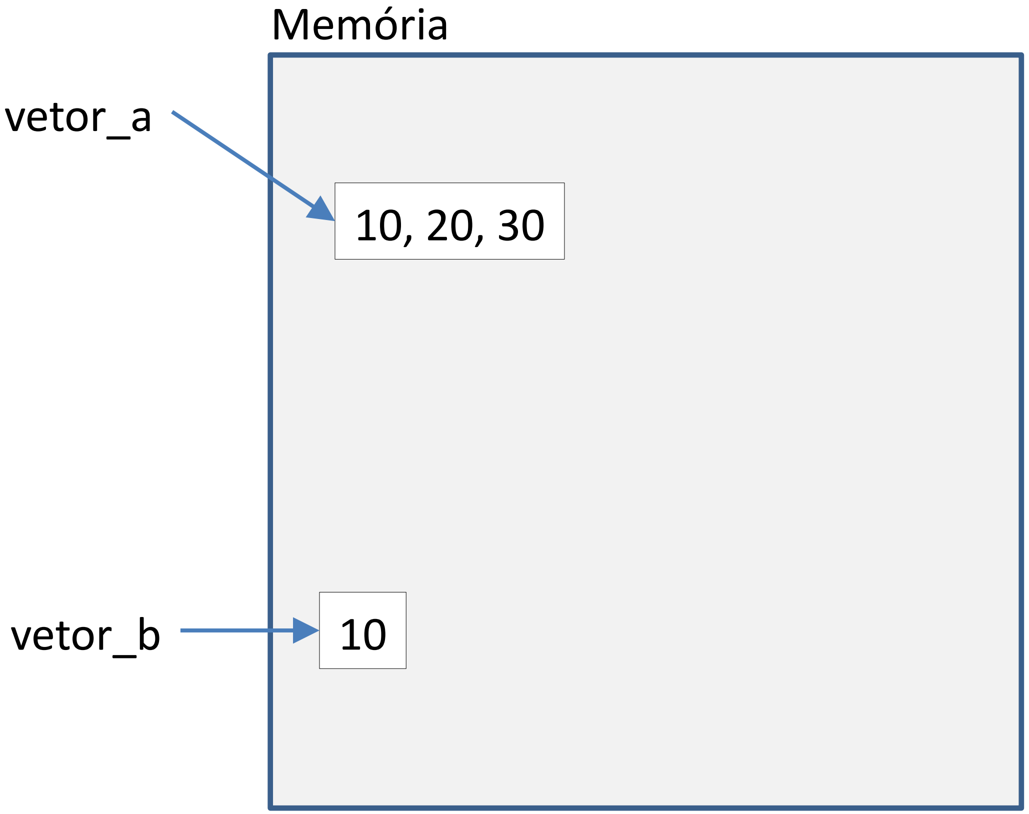
```
vetor_a = [10, 20, 30]
vetor_b = []
for i in range(0, len(vetor_a)):
    vetor_b.append(vetor_a[i])
```



Cópia de Vetores

- Para copiar os elementos do vetor/lista para uma nova estrutura de dados na memória, pode-se fazê-lo elemento a elemento.

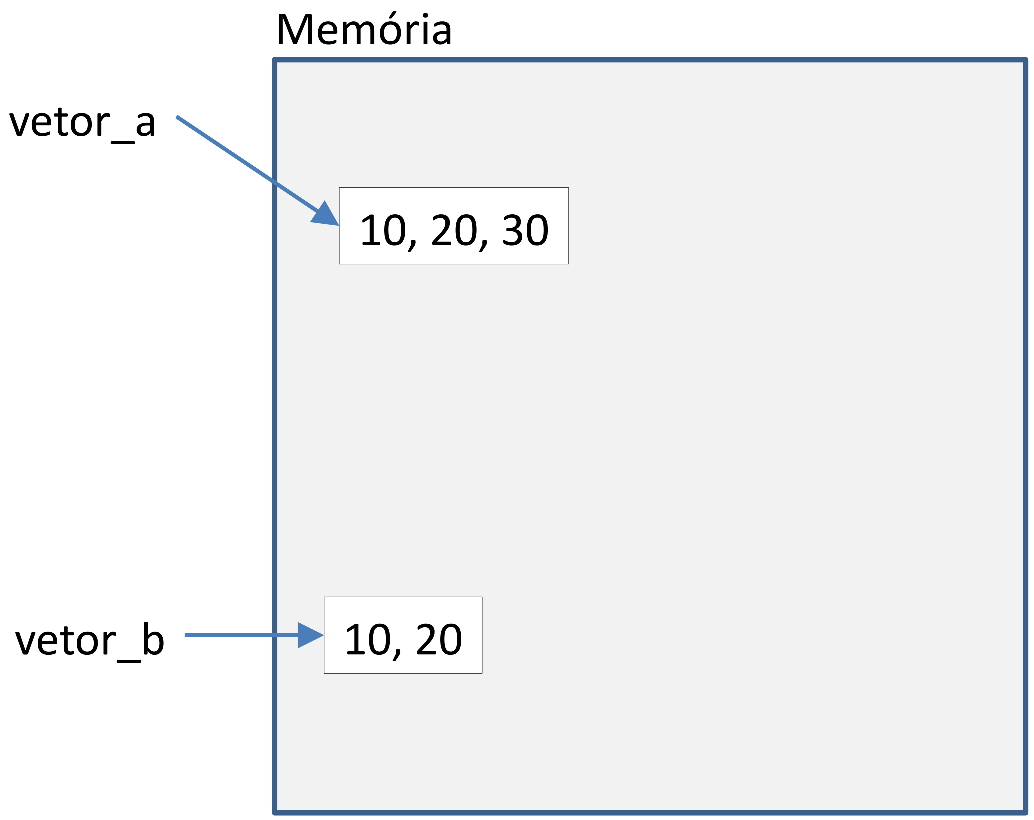
```
vetor_a = [10, 20, 30]
vetor_b = []
for i in range(0, len(vetor_a)):
   vetor_b.append(vetor_a[i])
```



Cópia de Vetores

- Para copiar os elementos do vetor/lista para uma nova estrutura de dados na memória, pode-se fazê-lo elemento a elemento.

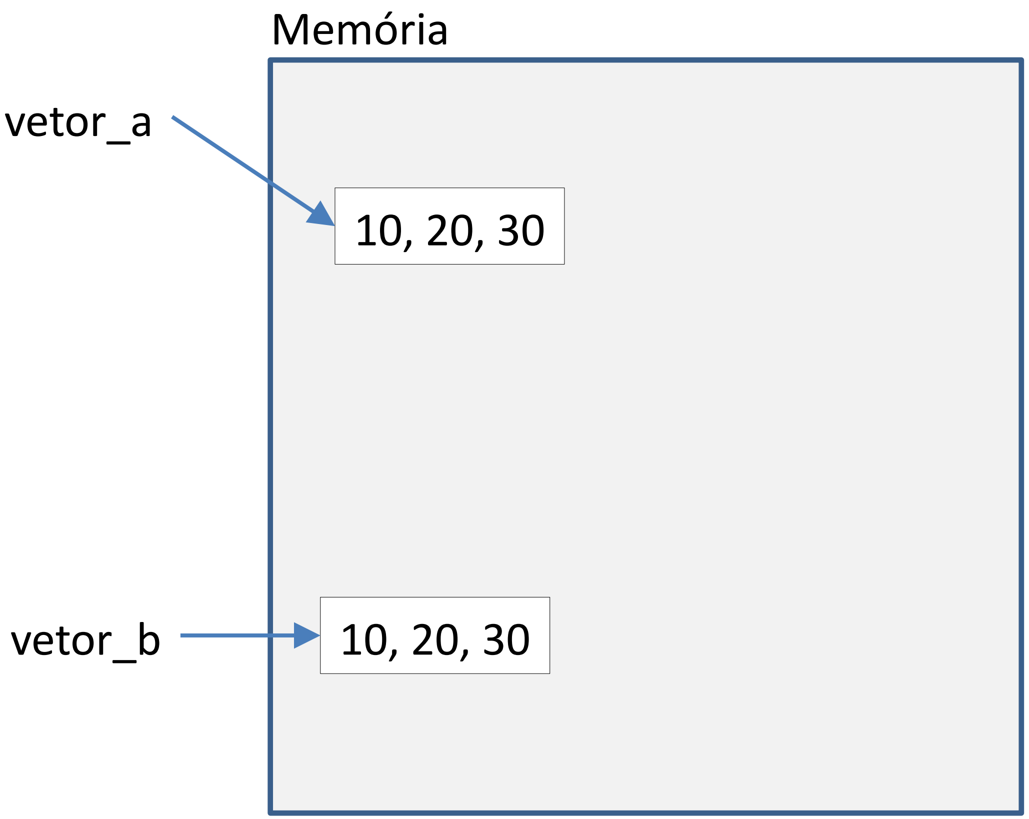
```
vetor_a = [10, 20, 30]
vetor_b = []
for i in range(0, len(vetor_a)):
    vetor_b.append(vetor_a[i])
```



Cópia de Vetores

- Para copiar os elementos do vetor/lista para uma nova estrutura de dados na memória, pode-se fazê-lo elemento a elemento.

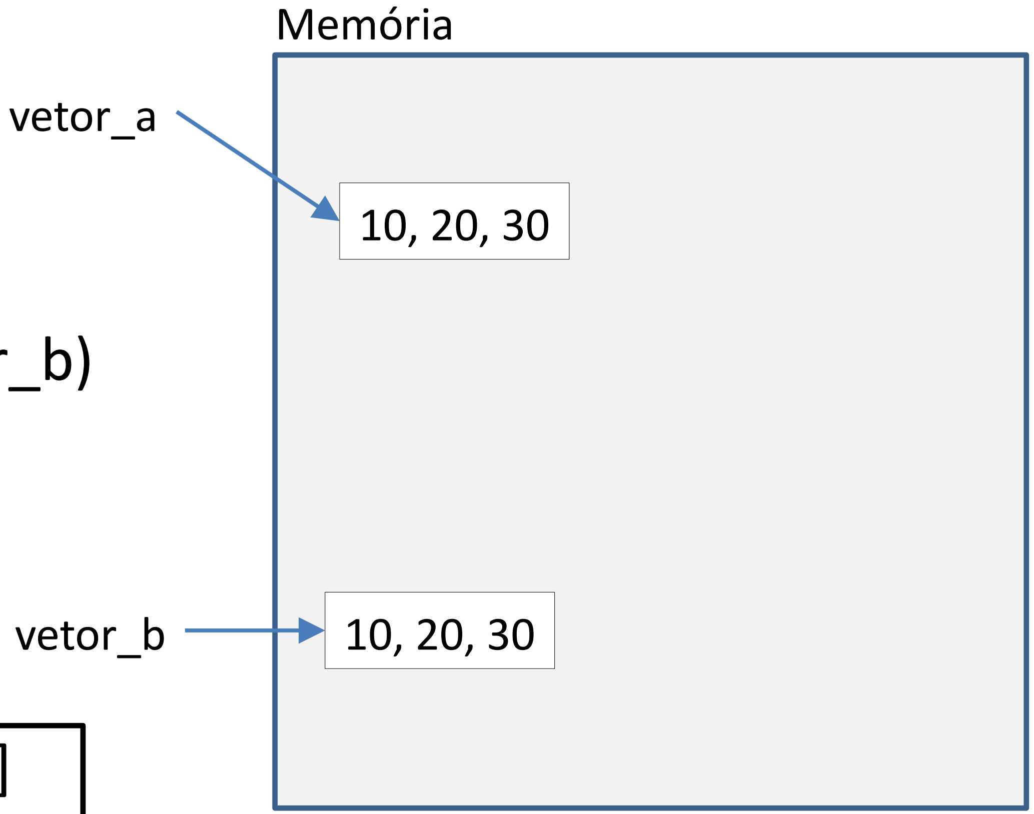
```
vetor_a = [10, 20, 30]
vetor_b = []
for i in range(0, len(vetor_a)):
    vetor_b.append(vetor_a[i])
```



Cópia de Vetores

- Para copiar os elementos do vetor/lista para uma nova estrutura de dados na memória, pode-se fazê-lo elemento a elemento.

```
vetor_a = [10, 20, 30]
vetor_b = []
for i in range(0, len(vetor_a)):
    vetor_b.append(vetor_a[i])
print('vetor_a =', vetor_a, '; vetor_b =', vetor_b)
vetor_b[1] = 0
```

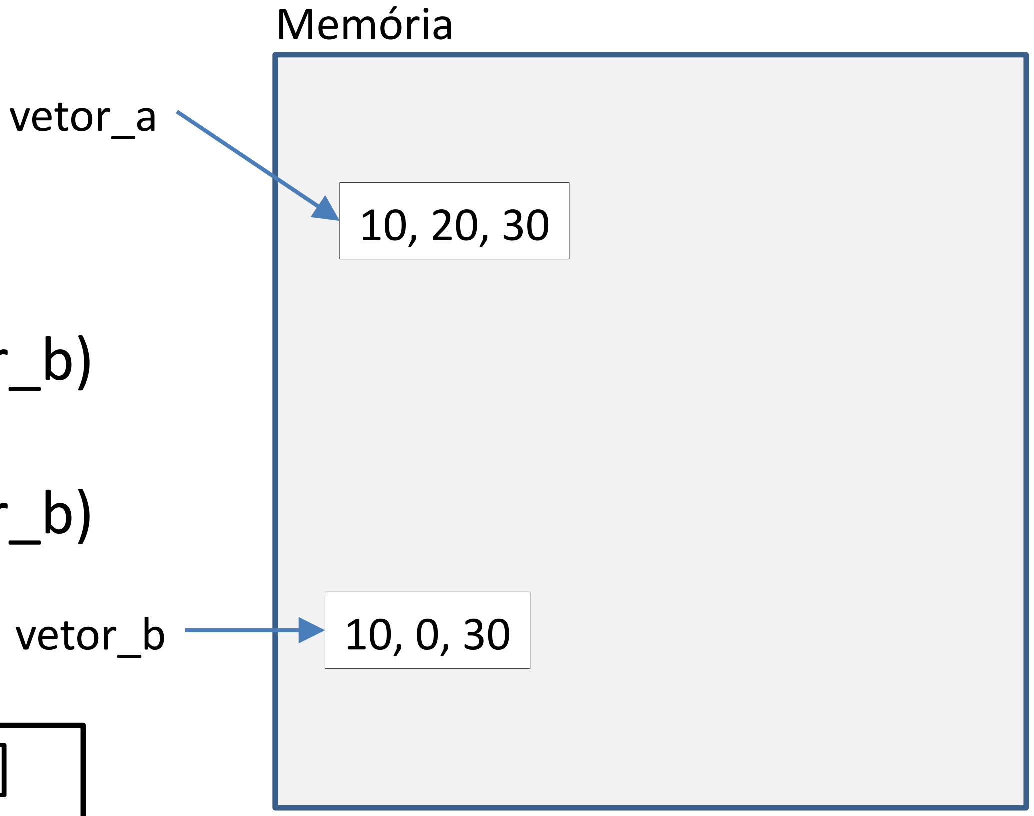



```
vetor_a = [10, 20, 30] ; vetor_b = [10, 20, 30]
```

Cópia de Vetores

- Para copiar os elementos do vetor/lista para uma nova estrutura de dados na memória, pode-se fazê-lo elemento a elemento.

```
vetor_a = [10, 20, 30]
vetor_b = []
for i in range(0, len(vetor_a)):
    vetor_b.append(vetor_a[i])
print('vetor_a =', vetor_a, '; vetor_b =', vetor_b)
vetor_b[1] = 0
print('vetor_a =', vetor_a, '; vetor_b =', vetor_b)
```




```
vetor_a = [10, 20, 30] ; vetor_b = [10, 20, 30]
vetor_a = [10, 20, 30] ; vetor_b = [10, 0, 30]
```

Cópia de Vetores

- Uma outra maneira é usar a função **copy** do módulo **copy** (que faz uma cópia *rasa* – ***shallow copy***).

```
import copy
vetor_a = [10, 20, 30]
vetor_b = copy.copy(vetor_a)
print('vetor_a =', vetor_a, '; vetor_b =', vetor_b)
vetor_a[0] = 0
print('vetor_a =', vetor_a, '; vetor_b =', vetor_b)
```

```
vetor_a = [10, 20, 30] ; vetor_b = [10, 20, 30]
vetor_a = [0, 20, 30] ; vetor_b = [10, 20, 30]
```

Funções úteis para Vetores

- Vimos a função **len(...)**, que retorna a dimensão (número de elementos) de um vetor/lista.
- Há diversas outras funções/operadores interessantes:

  max(vetor): retorna o valor máximo do vetor/lista **vetor**.

  min(vetor): retorna o valor mínimo do vetor/lista **vetor**.

  sum(vetor): retorna a soma dos elementos do vetor/lista **vetor**.

  valor in vetor: retorna **True** se **valor** (variável ou constante) faz parte do vetor/lista **vetor**.

  valor not in vetor: retorna **True** se **valor** (variável ou constante) não faz parte do vetor/lista **vetor**.

Funções úteis para Vetores

- Pode-se replicar os elementos do vetor/lista usando o operador ***** :

```
>>> v = [1, 2, 3]
>>> z = v * 3
>>> print(v)
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

Funções úteis para Vetores

- Pode-se concatenar vetores/listas usando o operador **+** :

```
>>> v = [1, 2, 3]
>>> w = [4, 5, 6]
>>> y = v + w
>>> print(y)
[1, 2, 3, 4, 5, 6]
```

Matrizes

- Algumas vezes necessitamos de **vetores multidimensionais** para resolver um problema.
- Um exemplo comum são as matrizes **bidimensionais**.
- Nesse caso, é necessário um índice para cada dimensão.
- São necessários **dois índices** para acessar um elemento de um matriz.
- Na realidade, em Python matrizes são **vetores de vetores**.

Matrizes

Exemplo:

```
>>> M = [[1, 5, 6, 7], [3, 2, 9, -1], [0, 4, -2, 5], [2, -3, 8, -7]]
>>> print(M[1])
[3, 2, 9, -1]
>>> print(M[1][2])
9
>>> M[0][0] = 0
>>> print(M)
[[0, 5, 6, 7], [3, 2, 9, -1], [0, 4, -2, 5], [2, -3, 8, -7]]
```

M =

1	**5**	**6**	**7**
3	**2**	**9**	**-1**
0	**4**	**-2**	**5**
2	**-3**	**8**	**-7**

Indexação de Matrizes

M[**primeiro_indice**][**segundo_indice**]

- O **primeiro_indice** refere-se às **linhas** da matriz.
- O **segundo_indice** refere-se às **colunas** da matriz.

M =

1	5	6	7
3	2	9	-1
0	4	-2	5
2	-3	8	-7

Tal como para vetores, usar **referências inválidas** vai provocar um erro:

```
M[0][4] = 0 # vai dar erro!
print(M[4][4]) # também vai dar erro!
```

Indexação de Matrizes

- Na realidade, um vetor de vetores (lista de listas) só é considerado um matriz, se os vetores internos tiverem a mesma dimensão.

  NM = [[1, 5, 6, 7], [3, 2, 9], [0, 4, -2, 5]] # não é considerado matriz

  CM = [[1, 5, 6, 7], [3, 2, 9, -1], [0, 4, -2, 5]] # é considerado matriz

- Como saber as dimensões de uma matriz (número de linhas e de colunas)?
- Usando o comando **len(...)**:

```
>>> len(CM)  # número de linhas
3
>>> len(CM[0])  # número de colunas
4
```

Inicializando uma Matriz

- Pode-se inicializar através de uma atribuição simples:

  M = [[1, 5, 6, 7], [3, 2, 9, -1], [0, 4, -2, 5], [2, -3, 8, -7]]

- Mas não se pode inicializar de forma análoga a vetores:

```
>>> M = [[0]*4]*3
>>> print(M)
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
>>> M[0][0] = 1
>>> print(M)
[[1, 0, 0, 0], [1, 0, 0, 0], [1, 0, 0, 0]]
```

- Cada linha representa o mesmo vetor na memória!

Inicializando uma Matriz

- Uma possibilidade é usar a seguinte instrução:

  M = [[0 for i in range(colunas)] for j in range(linhas)]

- Exemplo:

```
>>> colunas = 4
>>> linhas = 3
>>> M = [[0 for i in range(colunas)] for j in range(linhas)]
>>> print(M)
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
```

Inicializando uma Matriz

- Uma possibilidade é usar a seguinte instrução:

  M = [[0 for i in range(colunas)] for j in range(linhas)]

- Exemplo:

```
>>> colunas = 4
>>> linhas = 3
>>> M = [[0 for i in range(colunas)] for j in range(linhas)]
>>> M[0][0] = 1
>>> print(M)
[[1, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
```

Lendo uma Matriz

Exemplo:

```
linhas = int(input('Número de linhas: '))
colunas = int(input('Número de colunas: '))
mat = []
for i in range(0, linhas):
    mat.append([])
    for j in range(0, colunas):
        mat[i].append(int(input('Elemento [{}][{}]: '.format(i, j))))
print(mat)
```

Escrevendo uma Matriz

Escrevendo na forma de matriz:

```
linhas = int(input('Número de linhas: '))
colunas = int(input('Número de colunas: '))
mat = []
for i in range(0, linhas):
    mat.append([])
    for j in range(0, colunas):
        mat[i].append(int(input('Elemento [{}][{}]: '.format(i, j))))
for i in range(len(mat)):
    for j in range(len(mat[i])):
        print('{: ^3}'.format(mat[i][j]), end =" ")
    print()
```

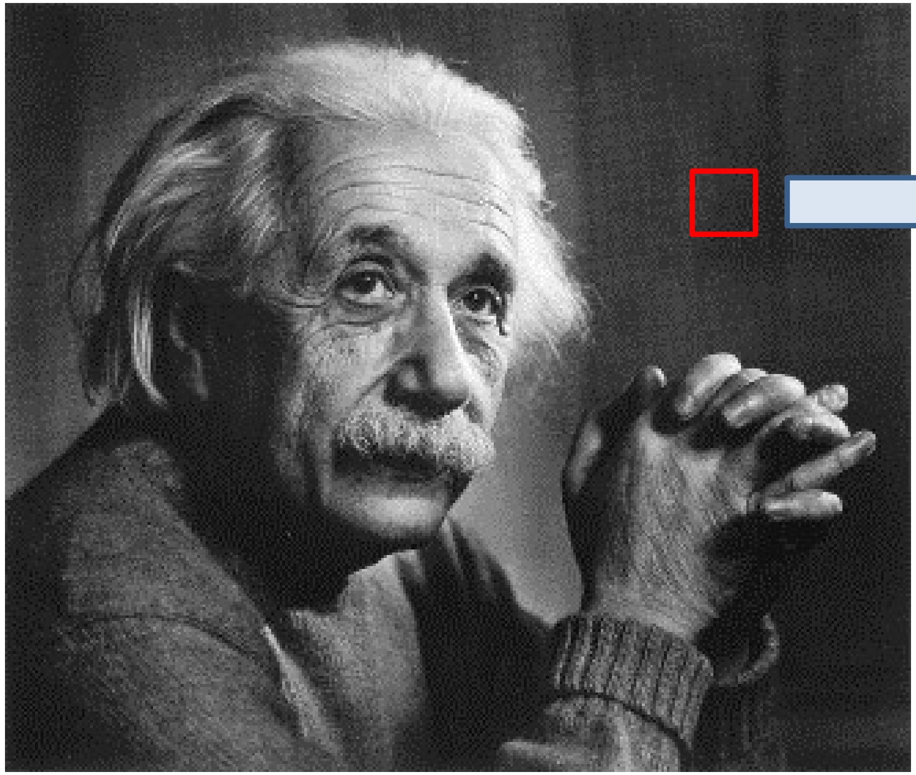
Matriz Transposta

- A transposta de uma matriz é uma matriz que apresenta os mesmos elementos de , só que colocados em uma posição diferente.
- Ela é obtida transportando-se ordenadamente os elementos das linhas de para as colunas da transposta

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \Rightarrow A^t = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

- Observe que as dimensões de A^t são invertidas em relação às dimensões de , e.g., e

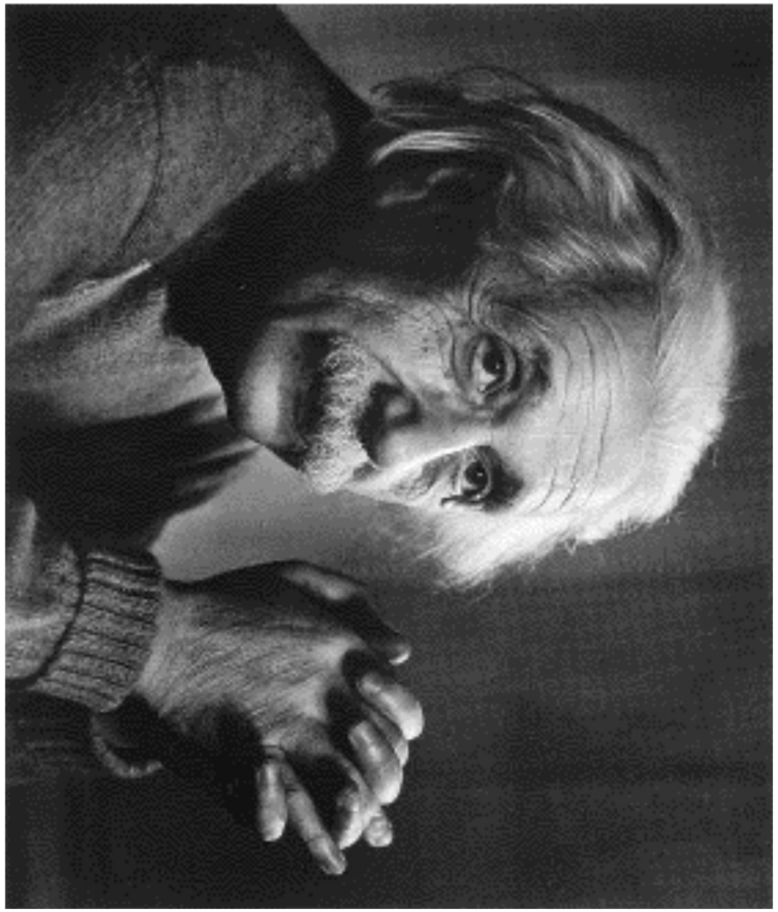
Matriz Transposta



36	36	36	36	36
36	36	45	45	54
36	36	45	54	54
36	45	54	54	63
36	45	54	63	63

$: \quad \subset \mathbb{R}^2$

Matriz Transposta



Matriz transposta

Matriz Transposta

Exercício: leia uma matriz A e calcule a sua transposta.

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \Rightarrow A^t = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

```
# inserir o código para ler uma matriz A
linhas =  len(A)
colunas = len(A[0])
# repare que a ordem está invertida na linha seguinte
At = [[0 for i in range(linhas)] for j in range(colunas)]
for i in range(linhas):
    for j in range(colunas):
        At[j][i]=A[i][j]
print('Matriz A:')
# inserir o código para escrever a matriz A
print('Matriz transposta de A:')
# inserir o código para escrever a matriz At
```

Cópia de Matrizes

- Parecido com cópia de vetores: mesmo problema.

```
>>> mat_a = [[10, 20], [30, 40]]
>>> mat_b = mat_a
>>> print('mat_a = ', mat_a, '; mat_b = ', mat_b)
mat_a = [[10, 20], [30, 40]] ; mat_b = [[10, 20], [30, 40]]
>>> mat_a[1][1] = 44
>>> print('mat_a = ', mat_a, '; mat_b = ', mat_b)
mat_a = [[10, 20], [30, 44]] ; mat_b = [[10, 20], [30, 44]]
```

Cópia de Matrizes

- Pode-se criar uma matriz com as mesmas dimensões da matriz, e depois ir copiando elemento a elemento.
- Uma outra maneira é usar a função **deepcopy** do módulo **copy** (que faz uma cópia *profunda* – ***deep copy***).

```
import copy
mat_a = [[10, 20], [30, 40]]
mat_b = copy.deepcopy(mat_a)
print('mat_a =', mat_a, '; mat_b =', mat_b)
mat_a[1][1] = 44
print('mat_a =', mat_a, '; mat_b =', mat_b)
```

```
mat_a = [[10, 20], [30, 40]] ; mat_b = [[10, 20], [30, 40]]
mat_a = [[10, 20], [30, 44]] ; mat_b = [[10, 20], [30, 40]]
```

Propriedades de uma Matriz

Exercício: escreva um programa que some os elementos da diagonal principal de uma matriz quadrada.

```
# inserir o código para ler uma matriz quadrada A
n =  len(A)
soma = 0
for i in range(n):
   soma += A[i][i]
print('Matriz A:')
# inserir o código para escrever a matriz A
print('Soma da diagonal principal: ', soma)
```

Multiplicação de Matrizes

Multiplicação de matriz por vetor (matriz coluna):

$$=$$

Multiplicação de Matrizes

Multiplicação de matriz por vetor (matriz coluna):

$$\begin{bmatrix} 2 & 3 & -1 \\ 0 & 6 & 2 \\ 7 & 8 & 10 \end{bmatrix}_{3\times 3} \times \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix}_{3\times 1} = \begin{bmatrix} 0 \\ -4 \\ 16 \end{bmatrix}_{3\times 1}$$

Multiplicação de Matrizes

$$\begin{bmatrix} 2 & 3 & -1 \\ 0 & 6 & 2 \\ 7 & 8 & 10 \end{bmatrix}_{3\times 3} \times \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix}_{3\times 1} = \begin{bmatrix} 0 \\ -4 \\ 16 \end{bmatrix}_{3\times 1}$$

Multiplicação de matriz por vetor (matriz coluna):

```
# inserir o código para ler uma matriz A
# inserir o código para ler um vetor B
if len(A[0]) != len(B):
    print('Não é possível multiplicar! Dimensões não casam.')
else:
    C = [[0 for i in range(len(B[0]))] for j in range(len(A))]
    for i in range(len(A)):
        soma = 0
        for j in range(len(B)):
            soma += A[i][j]*B[j][0]
        C[i][0] = soma
# inserir o código para escrever a matriz A
# inserir o código para escrever o vetor B
# inserir o código para escrever o vetor C = B * A
```

Multiplicação de Matrizes

$$\begin{bmatrix} 2 & 3 & -1 \\ 0 & 6 & 2 \\ 7 & 8 & 10 \end{bmatrix}_{3\times 3} \times \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix}_{3\times 1} = \begin{bmatrix} 0 \\ -4 \\ 16 \end{bmatrix}_{3\times 1}$$

Multiplicação de matriz por vetor (matriz coluna):

```
# inserir o código para ler uma matriz A
# inserir o código para ler um vetor B
if len(A[0]) != len(B):
    print('Não é possível multiplicar! Dimensões não casam.')
else:
    C = [[0] for j in range(len(A))]
    for i in range(len(A)):
        soma = 0
        for j in range(len(B)):
            soma += A[i][j]*B[j][0]
        C[i][0] = soma
# inserir o código para escrever a matriz A
# inserir o código para escrever o vetor B
# inserir o código para escrever o vetor C = B * A
```

Multiplicação de Matrizes

Multiplicação de matriz por matriz:

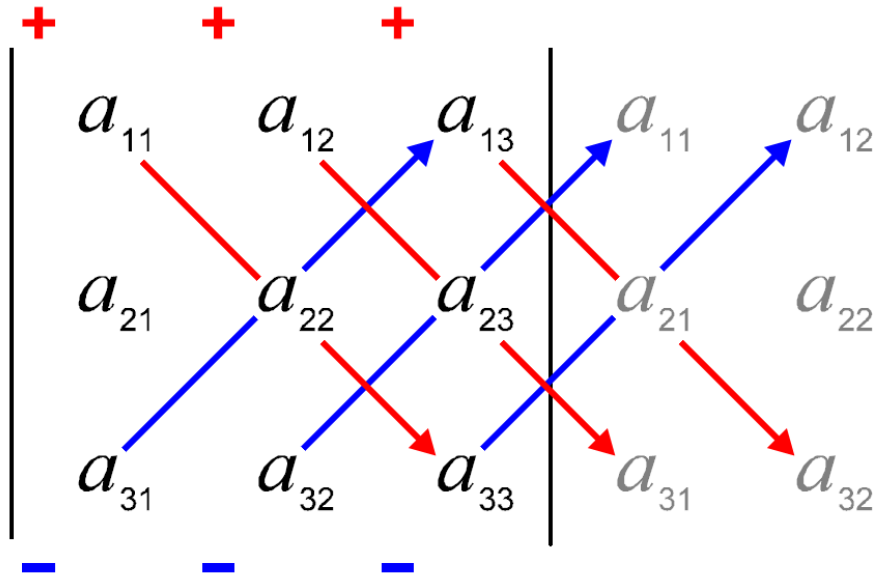
$$\begin{bmatrix} 2 & 3 \\ -1 & 0 \\ 6 & 2 \end{bmatrix}_{3\times 2} \times \begin{bmatrix} 3 & 1 & 2 \\ -1 & 1 & 0 \end{bmatrix}_{2\times 3} = \begin{bmatrix} 3 & 5 & 4 \\ -3 & -1 & -2 \\ 16 & 8 & 12 \end{bmatrix}_{3\times 3}$$

Multiplicação de Matrizes

Exercício: escrever um programa que lê duas matrizes e retorna o produto matricial.

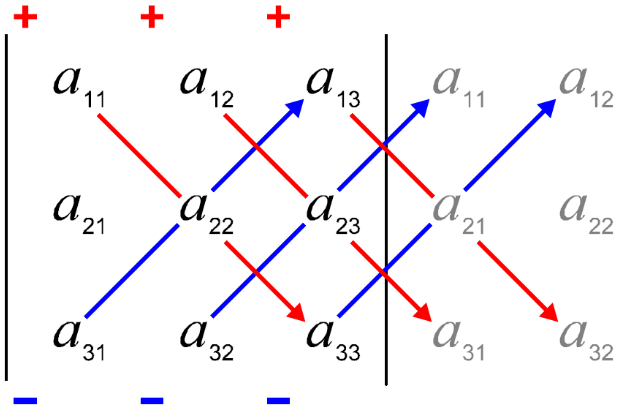
Determinante de uma Matriz

Regra de **Sarrus** (para matrizes 2×2 e 3×3):



Determinante de uma Matriz

- Regra de Sarrus (para matrizes 3×3).
- Uma forma simples de implementar:




```
# inserir o código para ler uma matriz A
DetA = (A[0][0] * A[1][1] * A[2][2])
DetA = DetA + (A[0][1] * A[1][2] * A[2][0])
DetA = DetA + (A[0][2] * A[1][0] * A[2][1])
DetA = DetA - (A[2][0] * A[1][1] * A[0][2])
DetA = DetA - (A[2][1] * A[1][2] * A[0][0])
DetA = DetA - (A[2][2] * A[1][0] * A[0][1])
print(DetA)
```

$$det\left(\begin{bmatrix} 1 & 1 & 0 \\ -3 & -1 & 2 \\ 3 & 2 & 0 \end{bmatrix}\right) = -2$$

- Há formas mais elegantes de calcular o determinante!

Sistemas de Equações Lineares

- Exemplo: sistema com três equações e três incógnitas:

$$
\begin{aligned}
{}_1 &+ {}_1 + {}_1 = {}_1 \\
{}_2 &+ {}_2 + {}_2 = {}_2 \\
{}_3 &+ {}_3 + {}_3 = {}_3
\end{aligned}
$$

- Incógnitas: ,
- Coeficientes: , , , , , , , , , , ,

Sistemas de Equações Lineares

- Exemplo: sistema com três equações e três incógnitas:

$$
\begin{aligned}
{}_1 + {}_1 + {}_1 &= {}_1 \\
{}_2 + {}_2 + {}_2 &= {}_2 \\
{}_3 + {}_3 + {}_3 &= {}_3
\end{aligned}
$$

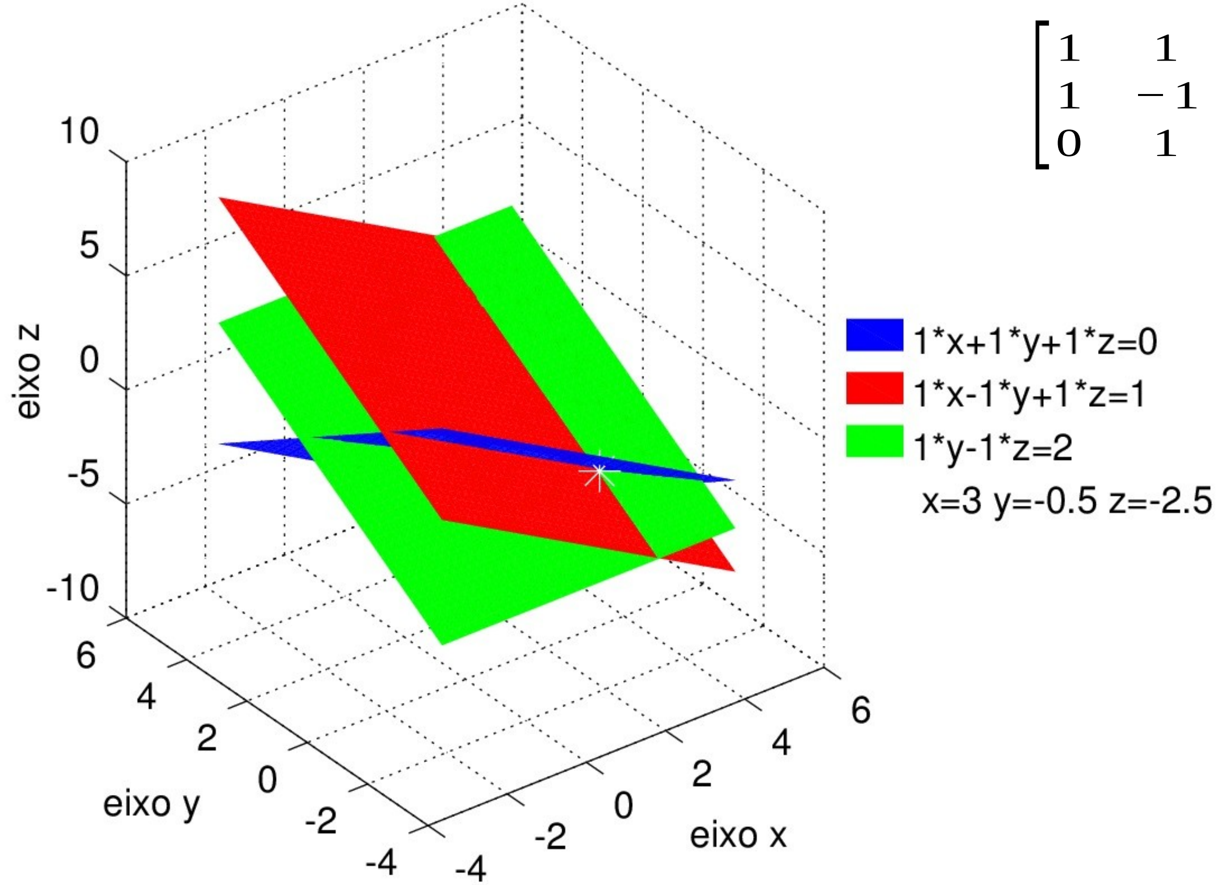
- Podem ser representados de forma vetorial:

$$
\begin{bmatrix} {}_1 & {}_1 & {}_1 \\ {}_2 & {}_2 & {}_2 \\ {}_3 & {}_3 & {}_3 \end{bmatrix} \times \begin{bmatrix} \; \end{bmatrix} = \begin{bmatrix} {}_1 \\ {}_2 \\ {}_3 \end{bmatrix}
$$

Sistemas de Equações Lineares

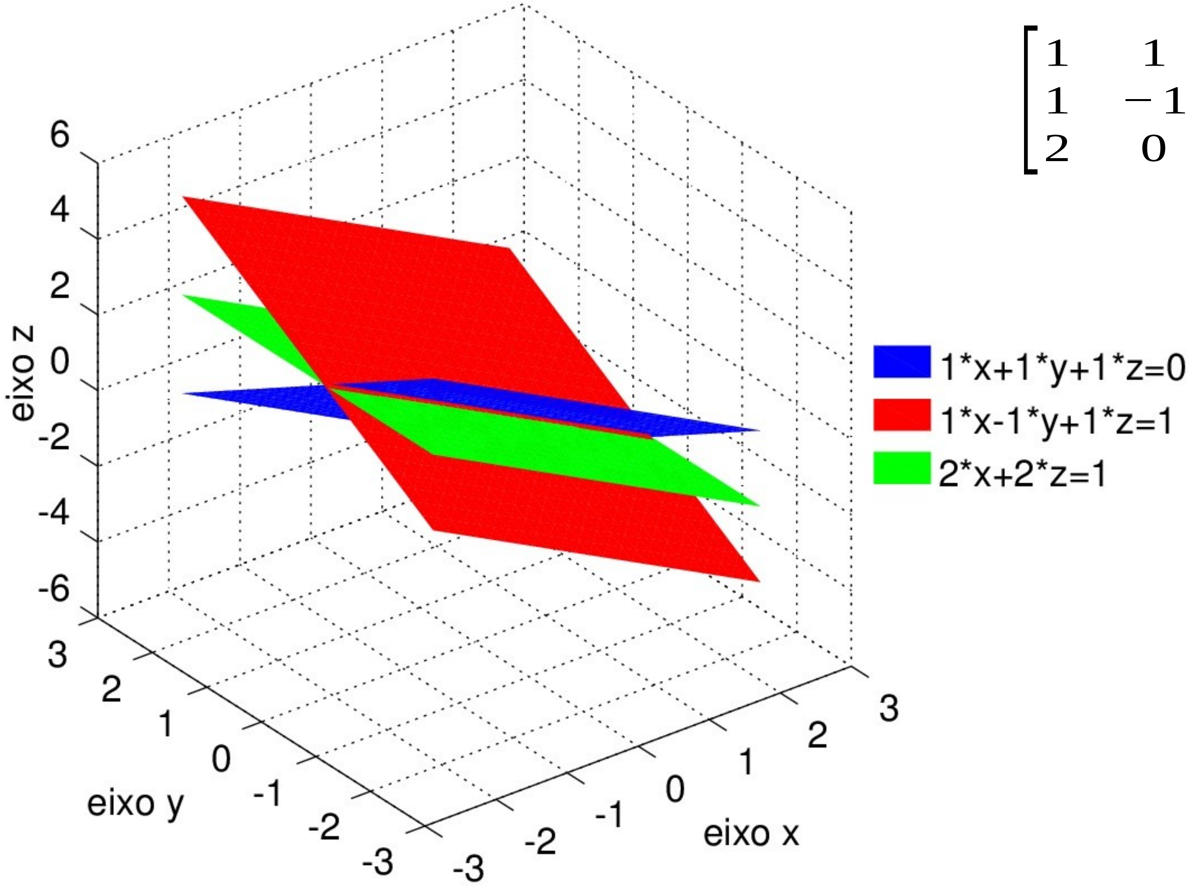
Sistema consistente e determinado: uma única solução.

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 1 \\ 0 & 1 & -1 \end{bmatrix} \times \begin{bmatrix} \\ \\ \\ \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}$$



Sistemas de Equações Lineares

Sistema consistente e indeterminado: infinitas soluções.

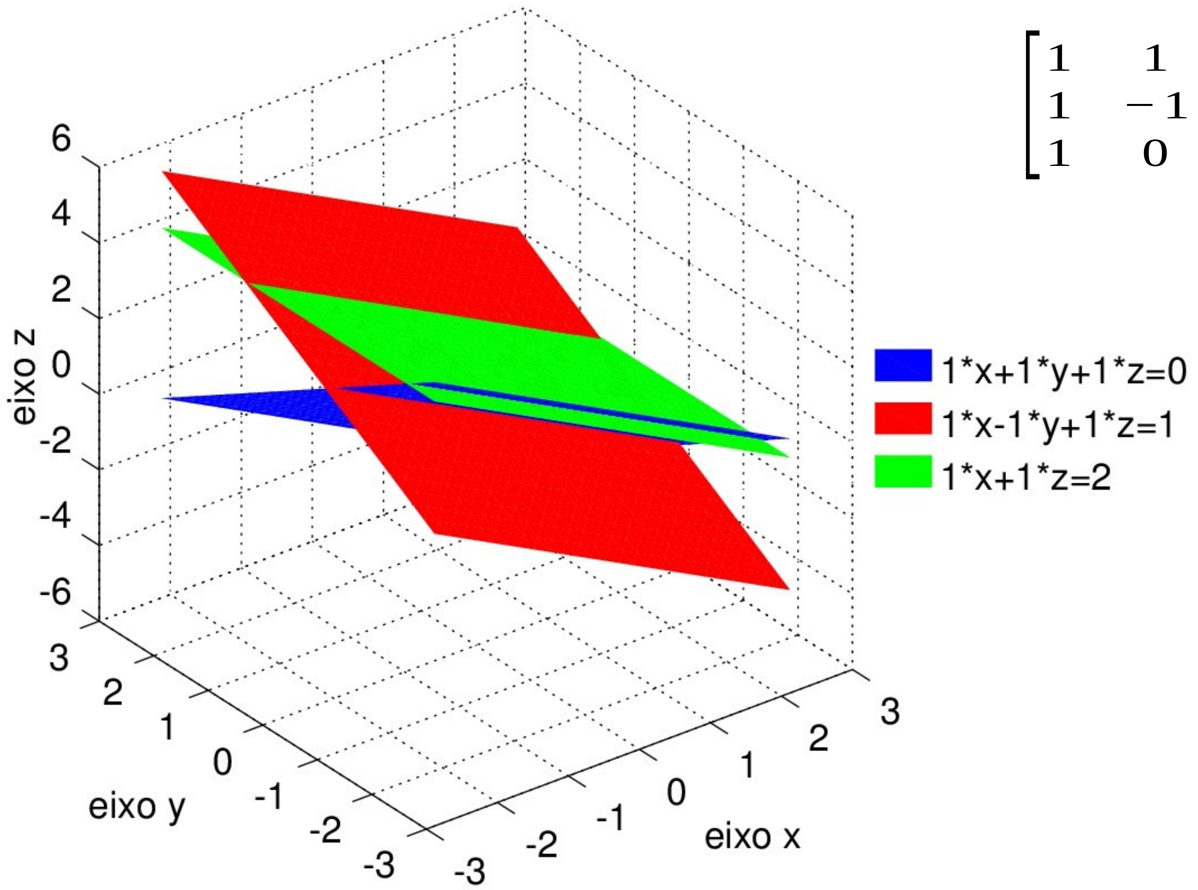


$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 1 \\ 2 & 0 & 2 \end{bmatrix} \times \begin{bmatrix} \\ \\ \\ \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$

Sistemas de Equações Lineares

Sistema inconsistente: sem solução.

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} \\ \\ \\ \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}$$



Sistemas de Equações Lineares

Solução pela regra de Cramer:

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix} \times \begin{bmatrix} \ \\ \ \\ \ \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

$$= \left(\begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix} \right) \qquad = \left(\begin{bmatrix} \mathbf{1} & 1 & 1 \\ \mathbf{2} & 2 & 2 \\ \mathbf{3} & 3 & 3 \end{bmatrix} \right) \qquad = \frac{\quad}{\quad}$$

$$= \left(\begin{bmatrix} 1 & \mathbf{1} & 1 \\ 2 & \mathbf{2} & 2 \\ 3 & \mathbf{3} & 3 \end{bmatrix} \right) \qquad = \frac{\quad}{\quad}$$

$$= \left(\begin{bmatrix} 1 & 1 & \mathbf{1} \\ 2 & 2 & \mathbf{2} \\ 3 & 3 & \mathbf{3} \end{bmatrix} \right) \qquad = \frac{\quad}{\quad}$$

Sistemas de Equações Lineares

Solução pela regra de Cramer.

- Se : sistema consistente e determinado.
- Se e : sistema consistente e indeterminado.
- Se e : sistema inconsistente.

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix} \times \begin{bmatrix} \\ \\ \\ \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \qquad = \frac{}{} \qquad = \frac{}{} \qquad = \frac{}{}$$

Sistemas de Equações Lineares

Exercício: escrever um programa que leia os coeficientes de um sistema de equações lineares e calcule o valor das incógnitas usando a regra de Cramer.

Introdução ao Processamento de Dados
Turma 3 (2020.1)

Introdução a Matrizes

Gilson. A. O. P. Costa (IME/UERJ)

gilson.costa@ime.uerj.br