

Programação em Python

<https://advancedinstitute.ai>



Programação Python

Operações Matriciais com o ***Numpy*** - Continuação

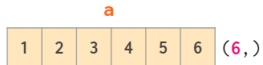
Referências e Fontes das Imagens

- ❑ NumPy Illustrated: The Visual Guide to NumPy
- ❑ Parallel and High Performance Computing (Book)
- ❑ Learning Numpy Array (Book)

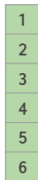
Vetores-coluna vs Vetores-linha

- ☐ No contexto 2D, os vetores de linha e coluna são tratados de forma diferente
- ☐ Normalmente o NumPy armazena matrizes 1D sempre que possível
- ☐ Por padrão, as matrizes 1D são tratadas como vetores de linha em operações 2D *shape* `(n,)` ou `(1,n)`
- ☐ Para construir vetores coluna: `np.reshape(-1,1), a[:, np.newaxis]`

Vetores-coluna vs Vetores-linha



a.reshape(-1,1)

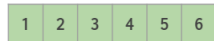


(6,1)

= **a[:, None]**



a.reshape(1,-1)

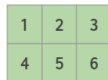


(1,6)

= **a[None, :]**



a.reshape(2,3)



(2,3)

= **a.reshape(2,-1)**

view

Aritmética de Matrizes

- ❑ Operadores *element-wise*: $+$, $-$, $*$, $/$, $//$ e $**$
- ❑ Multiplicação de Matrizes:

The diagram illustrates various NumPy operations using 2x2 matrices. Each matrix is represented as a 2x2 grid of colored boxes with numbers inside. The colors are: orange for the first operand, blue for the second operand, and green for the result.

Element-wise Addition:

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

Element-wise Subtraction:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 2 \\ 3 & 3 \end{bmatrix}$$

Element-wise Multiplication (*):

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} * \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 8 \end{bmatrix}$$

Element-wise Division (/):

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} / \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} = \begin{bmatrix} 0.5 & 2. \\ 3. & 2. \end{bmatrix}$$

Matrix Multiplication (@):

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} @ \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} = \begin{bmatrix} 2 & 4 \\ 6 & 8 \end{bmatrix}$$

Power ():**

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} ** \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 3 & 16 \end{bmatrix}$$

Broadcasting

- NumPy tenta realizar uma operação mesmo que os **operandos não tenham a mesma forma**;
 - Número são *promovidos a vetores*:

<table><tr><td>1</td><td>2</td></tr></table>	1	2	+	<table><tr><td>3</td></tr></table>	3	=	<table><tr><td>4</td><td>5</td></tr></table>	4	5		<table><tr><td>1</td><td>2</td></tr></table>	1	2	*	<table><tr><td>3</td></tr></table>	3	=	<table><tr><td>3</td><td>6</td></tr></table>	3	6		
1	2																					
3																						
4	5																					
1	2																					
3																						
3	6																					
<table><tr><td>1</td><td>2</td></tr></table>	1	2	-	<table><tr><td>3</td></tr></table>	3	=	<table><tr><td>-2</td><td>-1</td></tr></table>	-2	-1		<table><tr><td>1</td><td>2</td></tr></table>	1	2	/	<table><tr><td>3</td></tr></table>	3	=	<table><tr><td>0.33</td><td>0.67</td></tr></table>	0.33	0.67	np.float64	
1	2																					
3																						
-2	-1																					
1	2																					
3																						
0.33	0.67																					
						<table><tr><td>1</td><td>2</td></tr></table>	1	2	//	<table><tr><td>2</td></tr></table>	2	=	<table><tr><td>0</td><td>1</td></tr></table>	0	1	np.int32						
1	2																					
2																						
0	1																					
						<table><tr><td>3</td><td>4</td></tr></table>	3	4	**	<table><tr><td>2</td></tr></table>	2	=	<table><tr><td>9</td><td>16</td></tr></table>	9	16							
3	4																					
2																						
9	16																					

Broadcasting Cont.

□ *Broadcasting* de Matrizes

- NumPy permite operações mistas entre um vetor e uma matriz, e até mesmo entre dois vetores:

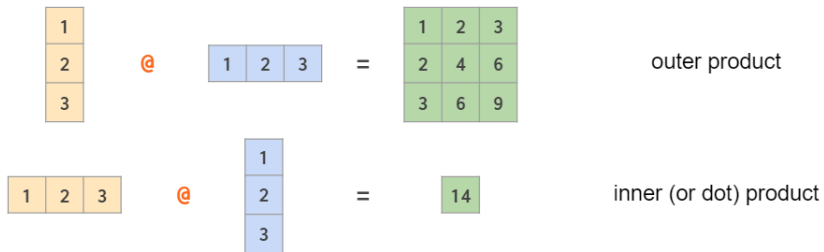
The image illustrates three NumPy broadcasting operations using 3x3 matrices. In each example, the first matrix is a 3x3 orange matrix with values 1, 2, 3 in the first row; 4, 5, 6 in the second row; and 7, 8, 9 in the third row.

Example 1: Normalization
The orange matrix is divided (indicated by a red slash) by a 3x3 blue matrix with the value 9 in the first column and 9 in the other two columns. The result is a 3x3 green matrix with values: first row [.1, .2, .3], second row [.4, .5, .7], and third row [.8, .9, 1.].

Example 2: Multiplying several columns at once
The orange matrix is multiplied (indicated by a red asterisk) by a 3x3 blue matrix with the value -1 in the first column and 0 in the other two columns. The result is a 3x3 green matrix with values: first row [-1, 0, 3], second row [-4, 0, 6], and third row [-7, 0, 9].

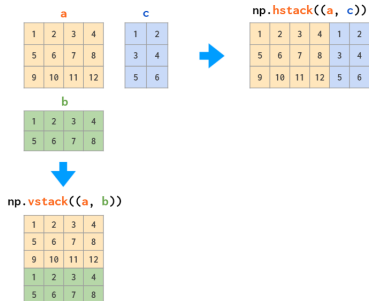
Example 3: Row-wise normalization
The orange matrix is divided (indicated by a red slash) by a 3x3 blue matrix with the value 3 in the first row, 6 in the second row, and 9 in the third row, with 3s in the other two columns. The result is a 3x3 green matrix with values: first row [.3, .7, 1.], second row [.6, .8, 1.], and third row [.8, .9, 1.].

Broadcasting Cont.

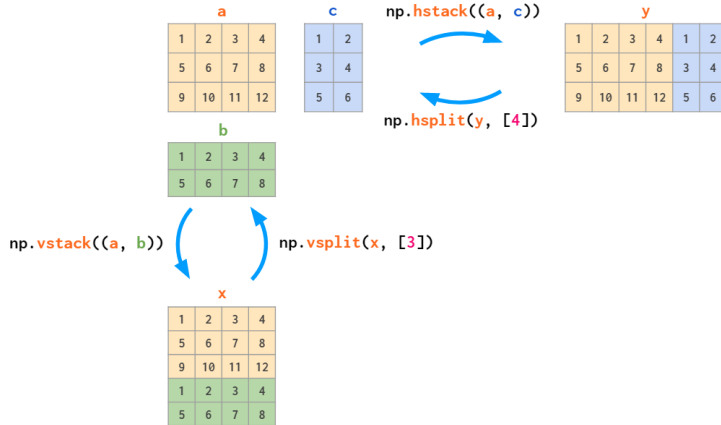


Unificação de Matrizes - `hstack()` e `vstack()`

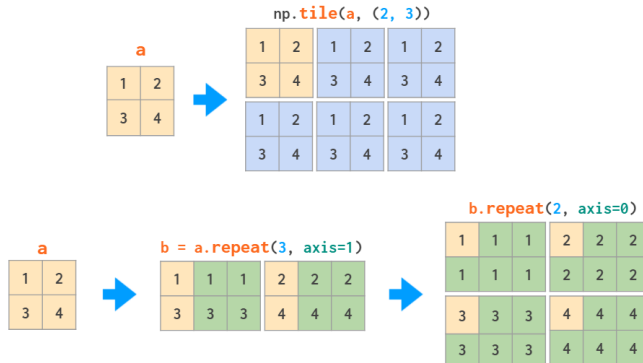
- Unificação na **horizontal** ou na **vertical**
- Cuidado com o shape em vetores coluna



Separação de Matrizes - `hstack()` e `vsplit()`



Replicação de Matrizes - `tile()`



Remoção de linhas e colunas - delete()

a

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

`np.delete(a, [1, 3], axis=1)`

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



1	3	5
6	8	10
11	13	15

`np.delete(a, 1, axis=0)`

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



1	2	3	4	5
11	12	13	14	15

`np.delete(a, np.s_[1:-1], axis=1)`

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



1	5
6	10
11	15

`= np.delete(a, slice(1,-1), axis=1)`

Inserção de linhas e colunas - `insert()`

`np.insert(h, [1, 2], 0, axis=1)`

1	0	3	0	5
6	0	8	0	10
11	0	13	0	15



h

1	3	5
6	8	10
11	13	15

0 1 2

`np.insert(v, 1, 7, axis=0)`

1	2	3	4	5
7	7	7	7	7
11	12	13	14	15



v

1	2	3	4	5
11	12	13	14	15

`np.insert(u, [1], w, axis=1)`

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



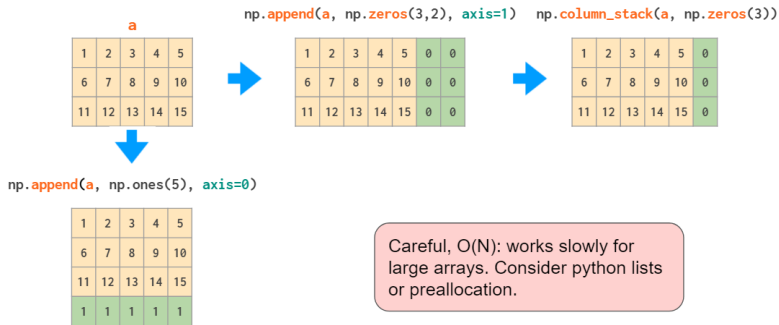
u

1	5
6	10
11	15

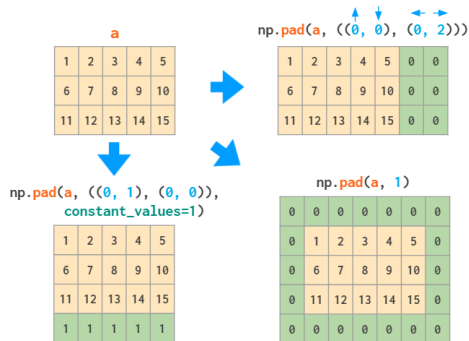
w

2	3	4
7	8	9
12	13	14

Inserção de linhas e colunas - `append()`

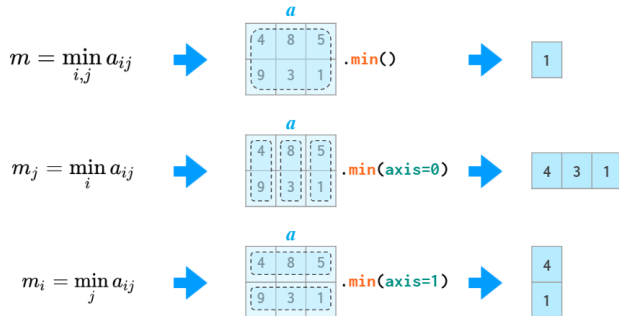


Inserção de linhas e colunas - pad()



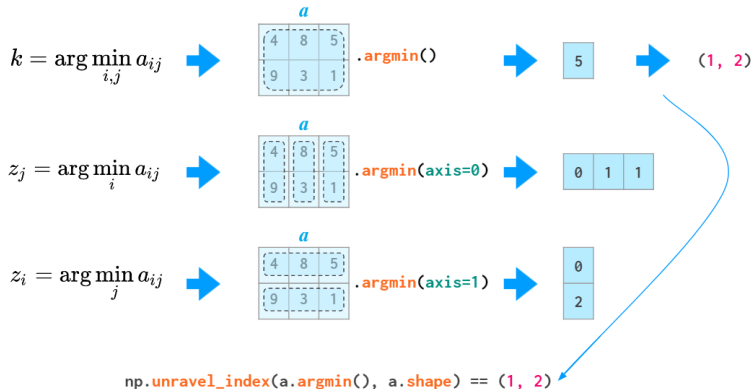
Estatística em Matrizes

- Funções estatísticas básicas: min/max, argmin/argmax, mean/median/percentile, std/var

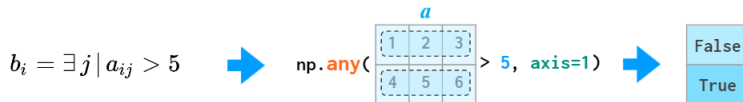
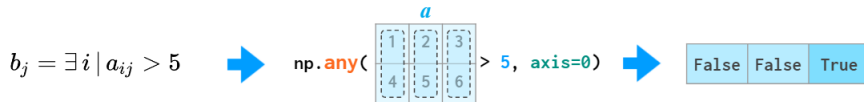
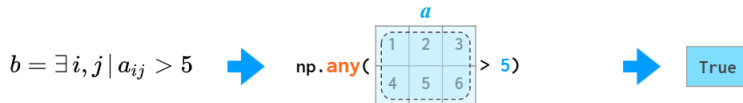


Estatística em Matrizes

□ Retornando as posições:

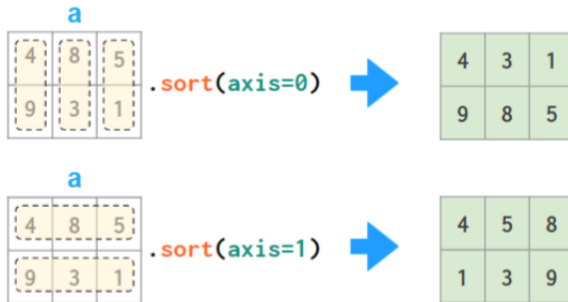


Quantificadores - `all()` e `any()`



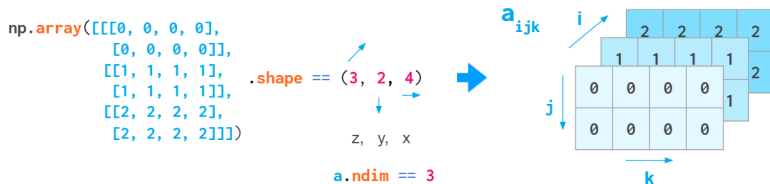
Ordenação de Matrizes

- Utilização do parâmetro `axis` para especificar linha/coluna



Matrizes no mundo 3D

- Quando você cria uma matriz 3D a partir do vetor 1D (usando `reshape()` ou convertendo uma lista Python aninhada, o significado dos índices é **(z, y, x)**)



Indexação em 3D

- Ao trabalhar com imagens RGB, a ordem (y, x, z) é mais utilizada:
 - Duas coordenadas de pixel e a última é a coordenada de cor
 - `a[i,j]` dá uma tupla **(r,g,b)** tupla do pixel **(i,j)**

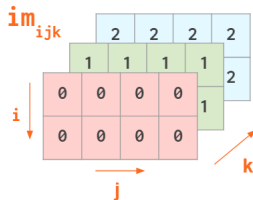
```
np.array([ [[0,1,2], [0,1,2], [0,1,2], [0,1,2]],  
          [[0,1,2], [0,1,2], [0,1,2], [0,1,2]] ]) .shape == (2, 4, 3)
```

↓ →
y, x, z



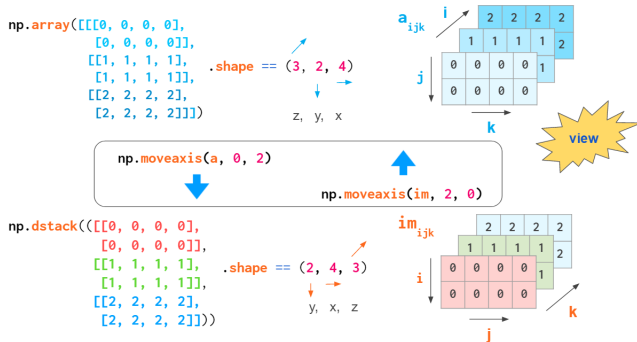
```
np.dstack([ [0, 0, 0, 0],  
            [0, 0, 0, 0],  
            [1, 1, 1, 1],  
            [1, 1, 1, 1],  
            [2, 2, 2, 2],  
            [2, 2, 2, 2] ]) .shape == (2, 4, 3)
```

↓ →
y, x, z



Indexação em 3D - Cont.

- Utilizando a função `moveaxis()`
 - Operação simples (computacionalmente barata) que gera uma visão dos dados



Dúvidas?